ICONOS
Instituto de Investigación en Comunicación y Cultura

METODOLOGÍA DE LA PROGRAMACIÓN A TRAVÉS DE PSEUDOCÓDIGO Y SU RELACIÓN SINTÁCTICA CON EL LENGUAJE DE PROGRAMACIÓN PHP.

TESIS

QUE PARA OBTENER EL TITULO DE

MAESTRÍA EN COMUNICACIÓN CON MEDIOS VIRTUALES

PRESENTA:

ALEJANDRO GONZÁLEZ REYES

ASESOR:

MTRA. ROSELENA VARGAS VELASCO

MÉXICO, CDMX. JULIO 2015

RECONOCIMIENTO DE VALIDEZ OFICIAL DE ESTUDIOS DE LA SECRETARÍA DE EDUCACIÓN PÚBLICA SEGÚN ACUERDO NO. 2005625 CON FECHA 22 DE SEPTIEMBRE DE 2005. Y CLAVE 2012

ÍNDICE

INTRODUCCIÓN **4**

CAPÍTULO 1 ..... **8**

1.1 ALGORITMOS Y PSEUDOCÓDIGO 11
1.1.1 Algoritmos *11*
1.1.1.1 Características, partes y representación de los Algoritmos *12*
1.1.2 Pseudocódigo *21*
1.1.2.1 Pseudo-Intérprete *24*
1.2 ASPECTOS BÁSICOS PRESENTES EN EL PSEUDOCÓDIGO 27
1.2.1 Identificadores *29*
1.2.1.2 Variables *32*
1.2.1.2 Constantes *40*
1.2.2 Entrada y salida de información *44*
1.2.2.1 Entrada *46*
1.2.2.2 Salida *49*
1.2.3 Tipos de datos *55*
1.2.4 Operadores *64*
1.2.4.1 Operadores Aritméticos *64*
1.2.4.2 Operadores de Comparación *69*
1.2.4.3 Operadores Lógicos *74*
1.2.4.4 Prioridad de Operadores *83*
1.2.5 Comentarios *87*
Conclusión *93*

CAPÍTULO 2 ..... **98**

2.1 ASPECTOS DE NIVEL INTERMEDIO PRESENTES EN EL PSEUDOCÓDIGO 102
2.1.1 Estructuras de Control *103*
2.1.1.1 Estructuras de Selección *104*
2.1.1.1.1 Si Entonces *105*
2.1.1.1.2 Si Entonces Sino *109*
2.1.1.1.3 Si Entonces Sino Anidado *114*
2.1.1.1.4 Según (Selección Múltiple) *119*
2.1.1.2 Estructuras de Repetición *131*
2.1.1.2.1 Para *132*
2.1.1.2.2 Mientras *142*
2.1.1.2.3 Repetir Mientras Que *149*
2.1.2 Funciones primitivas del lenguaje (Pre-construidas) *161*
2.1.3 Funciones del programador (Externas) *177*
Conclusión *200*

CAPÍTULO 3 ..... **206**

3.1 ASPECTOS BÁSICOS PRESENTES TANTO EN EL PSEUDOCÓDIGO COMO EN EL LENGUAJE DE PROGRAMACIÓN PHP 209

3.1.1 Identificadores: Variables y Constantes 211
3.1.2 Constantes 212
3.1.3 Entrada de Información 214
3.1.4 Salida de Información 218
3.1.5 Tipos de Datos 221
3.1.6 Operadores Aritméticos 224
3.1.7 Operadores de Comparación 225
3.1.8 Operadores Lógicos 227
3.1.9 Comentarios 228
3.2 ASPECTOS DE NIVEL INTERMEDIO PRESENTES TANTO EN EL PSEUDOCÓDIGO COMO EN EL LENGUAJE DE PROGRAMACIÓN PHP 233
3.2.1 Estructura de Selección Si Entonces 234
3.2.2 Estructura de Selección Si Entonces Sino 236
3.2.3 Estructura de Selección Si Entonces Sino (Anidado) 237
3.2.4 Estructura de Selección Múltiple (Según) 241
3.2.5 Estructura de Repetición Para 243
3.2.6 Estructura de Repetición Mientras 246
3.2.7 Estructura de Repetición Repetir Mientras Que 247
3.2.8 Funciones primitivas del lenguaje (Pre-construidas) 248
3.2.9 Funciones del programador (Externas) 252
3.3 SOLUCIONES DE PROGRAMACIÓN: MIGRAR DEL PSEUDOCÓDIGO AL LENGUAJE DE PROGRAMACIÓN PHP 257
3.3.1 Leer datos proporcionados por el usuario 259
3.3.2 Tomar decisiones 265
3.3.3 Tomar múltiples decisiones 272
3.3.4 Repetir instrucciones 279
3.3.5 Trabajar con funciones pre-construidas 286
3.3.6 Trabajar con funciones personalizadas 294
Conclusión 299
CONCLUSIONES 303

Introducción

Actualmente se vive en una sociedad cada vez más competitiva donde el desarrollo tecnológico juega un papel fundamental en la forma de vida de las personas, lo que genera un impacto transcendental en muchas áreas del conocimiento. En el sector educativo; por ejemplo, las Tecnologías de la Información y la Comunicación (TIC) han demostrado que pueden ser de gran apoyo tanto para profesores, como para estudiantes. Puesto que, al implementar la tecnología en la educación, lo que se pretende es facilitar el trabajo de los docentes a través de la generación y publicación de materiales académico-didácticos que provean a los estudiantes de un mayor número de elementos para enriquecer su proceso de enseñanza aprendizaje.

Bajo este mismo orden de ideas, y en vista de un panorama contemporáneo genérico que evoluciona vertiginosamente por el uso de las computadoras en casi todos los procesos de su desarrollo. Es normal que al profesional del Diseño Digital se le demanden nuevas habilidades especificas en el terreno del desarrollo, ya sea en la industria de los videojuegos, las aplicaciones Web, los sistemas informáticos móviles, entre otros.

Si regresamos la mirada unos diez años atrás, es posible observar ciertos cambios revolucionarios que han sido impulsados por la era digital, y con ellos, la forma de cómo ha evolucionado la manera de comunicarnos y expresarnos en sociedad. *Smartphones*, Computadoras, *Tablets*, *Smart TV*, *Smartwatches*, etc.; son solo algunos ejemplos de esta diversidad tecnológica. Sin embargo, para dominar e interactuar con cada uno de

estos dispositivos, es importante desarrollar aplicaciones ***software*** que nos provean de las interfaces (ya sean gráficas o por línea de comandos) y funciones operativas necesarias para poder manipularlos.

En este sentido, el presente trabajo de investigación tiene por objeto generar un guía de apoyo teórico práctica que dote de conocimientos a los estudiantes del área de Diseño Digital de ICONOS sobre temas relacionados con los fundamentos de la programación básica, todo ello mediante el uso de un Pseudolenguaje asistido por computadora que les permita centrar su atención en los aspectos más importantes de la programación sin tener que preocuparse por atender las problemáticas ocasionadas a consecuencia de la sintaxis estricta que conlleva adoptar un lenguaje de programación formal (como es el caso de PHP) desde los primeros inicios de su formación académica en el sector de la programación.

Cabe señalar que he seleccionado este centro de estudios debido a que en su momento (años 2013 – 2015) formé parte integral de su matrícula estudiantil activa en modalidad Maestría. Asimismo, durante mi estancia identifiqué ciertas problemáticas en el plan de estudios correspondiente a la Licenciatura de Diseño Digital, debido a que está conformado por algunas materias relacionadas con el diseño y desarrollo de sitios Web, mismas que por cuestiones de tiempo y espacio son abordadas por cada uno de los docentes mediante uso de lenguajes de programación formales como es el caso de JavaScript o PHP (con conexión a bases de datos), dejando de lado las bases fundamentales de la algoritmia computacional.

Sumado a lo expuesto, cabe señalar que el abordar de manera somera o nula la Metodología de la programación, entre las que se encuentran:

1. La identificación del problema.
2. Planteamiento de alternativas de solución (análisis).
3. Elección de una alternativa.
4. Desarrollo de una solución (Algoritmo).
5. Codificación de la solución mediante el uso de un Pseudolenguaje.
6. Evaluación (testeo).

Genera en el estudiante un déficit de atención y frustración ocasionados comúnmente por la sobrecarga de "documentación técnica" que conlleva aprender un nuevo lenguaje de programación formal, debido a que no cuenta con las bases fundamentales (lógica de programación) para la solución de problemas mediante el uso de un ordenador. En otras palabras, es importante que el estudiante de Diseño Digital aprenda en primer lugar a identificar y/o analizar correctamente los problemas, y centre su atención en el diseño de alternativas de solución a través de un Pseudolenguaje flexible y "en español", que le permita dejar de lado los formalismos propios de un lenguaje de programación y su relación (de este) con otras tecnologías.

El propósito de esta investigación es proporcionar a los estudiantes del área de Diseño Digital de ICONOS que recién inician su formación en el ámbito de la programación, una serie de problemas representativos, los cuales han sido resueltos algorítmicamente con detalle mediante el uso de PSeInt; una herramienta de software que asiste a los estudiantes en sus primeros pasos en programación. Mediante un simple e intuitivo Pseudolenguaje en español (complementado con un editor de diagramas de flujo), les permite centrar su atención en los conceptos fundamentales de la algoritmia computacional, minimizando las dificultades propias de un lenguaje y proporcionando un entorno de trabajo con numerosas ayudas y recursos didácticos.

Los problemas que se plantean en este trabajo de investigación están enfocados en utilizar los paradigmas clásicos de la programación, de modo que el estudiante se vaya familiarizando paso a paso en la solución de problemas cada vez más complejos. De aquí que el formato de esta investigación dedique su primer capítulo en abordar el paradigma de la programación secuencial, seguido de los paradigmas orientados a la programación estructurada y funcional, así como la redacción de un tercer y último capítulo dedicado a la comparación de algunos elementos gramaticales presentes tanto en el Pseudocódigo como en el Lenguaje de Programación PHP.

En definitiva, el objetivo de este trabajo de investigación no es el de establecer un patrón para la solución de problemas mediante el uso de un ordenador, más bien se trata de proporcionar unas líneas de ayuda generales a los estudiantes del área de Diseño Digital para desarrollar una lógica de programación adecuada que les permita más adelante sumergirse en diferentes escenarios laborales relacionados con temas de desarrollo, entre las que destacan: los programas de escritorio, los sistemas informáticos móviles, las aplicaciones Web, la programación de videojuegos, etc.

En hora buena, espero que la información plasmada en esta investigación sea de gran ayuda para el lector, tanto como lo ha sido para mí el reto y satisfacción de poder escribirla.

Capítulo 1

Fundamentos Gramaticales del Pseudocódigo mediante Programación Secuencial

Cuando un estudiante se enfrenta por primera vez con el reto de aprender a programar, con frecuencia los docentes recurren a la enseñanza de los Algoritmos computacionales elaborados en lápiz y papel para explicar las bases fundamentales de la algoritmia computacional y, posteriormente, eligen un lenguaje de programación como herramienta de trabajo que les permita codificar dichas soluciones para que sea posible ejecutarlas a través de un ordenador.

Sin embargo, es probable que los medios descritos con anterioridad provoquen que el estudiante desvié su atención de los conceptos fundamentales que debe aprender, debido a la cantidad de tiempo que se requiere para diseñar y evaluar un Algoritmo basado en papel, así como el conocer los detalles técnicos que conlleva adoptar un lenguaje de programación desde los primeros inicios de su formación.

Por tal motivo, el presente capítulo tiene por objeto elaborar una guía tutorial que aborde las bases fundamentales de la algoritmia computacional a través de Pseudocódigo empleando la herramienta de *software* PSeInt.

Se trata básicamente de responder a la pregunta de ¿Qué elementos gramaticales del Pseudolenguaje se deben considerar para aprender las

bases fundamentales de la algoritmia computacional?, para lo cual surge el planteamiento de la hipótesis que argumenta que los elementos gramaticales necesarios para aprender las bases fundamentales de la algoritmia computacional son:

a) Identificadores
b) Variables y Constantes
c) Entradas y Salidas de Información
d) Tipos de Datos (Numéricos, Caracteres y Booleanos)
e) Tipos de Operadores (Aritméticos, Comparación y Lógicos)
f) Comentarios

En este sentido es necesario remitirse a los aportes que diferentes autores sostienen acerca de cada uno de estos aspectos, así como los beneficios que conlleva considerarlos durante los primeros inicios de la formación académica de un estudiante en el campo de la programación. Al emplear la herramienta de *software* PSeInt en la mayoría de los temas abordados dentro de este capítulo, se pretende facilitar al estudiante la tarea de escribir algoritmos en este Pseudolenguaje, puesto que presenta un entorno de trabajo con numerosas ayudas que permiten ejecutar, localizar errores y analizar paso a paso la lógica de sus algoritmos; como si se tratase de un lenguaje de programación real.

Al finalizar la lectura de este capítulo, el lector tendrá los conocimientos y habilidades necesarias para resolver algunos problemas mediante el uso de un ordenador. Por ejemplo: una calculadora básica, determinar áreas y perímetros de figuras geométricas clásicas, cálculo de promedios, entre otros.

Es por ello que, para ayudarle a sacar el mayor partido al texto y saber dónde se encuentra en cada momento de las explicaciones proporcionadas a lo largo de este capítulo, la mayoría de los ejemplos se presentan en un formato de captura de pantalla, siendo posible obtener su correspondiente código fuente (archivo PSeInt) a través de la siguiente <URL>.

1.1 Algoritmos y Pseudocódigo

Las computadoras como máquinas creadas por el hombre no son más que herramientas de trabajo que solo pueden realizar las tareas para las que han sido programadas, puesto que, no tienen ninguna inteligencia y por consiguiente son incapaces de hacer algo por sí mismas. En este sentido, la potencialidad de estas herramientas, ésta a la espera de que una persona le saque provecho y, para ello, lo único que se necesita es proporcionarle una serie de instrucciones. En este caso los Algoritmos y el Pseudocódigo.

1.1.1 Algoritmos

Las instrucciones que le son brindadas a una computadora para que las ejecute en beneficio de la solución de un problema, constituyen lo que se conoce como Algoritmo. Según Juan Carlos López García en su obra Algoritmos y Programación (Guía para docentes), establece que:

> En el ámbito de la computación, los Algoritmos son una herramienta que permite describir claramente un conjunto finito de instrucciones, ordenadas secuencialmente y libres de ambigüedad, que debe llevar a cabo un computador para lograr un resultado previsible… (López 21).

Sumado a lo expuesto, Trejos define un Algoritmo como:

> … Un conjunto de pasos secuenciales y ordenados que permiten lograr un objetivo. Que sean pasos secuenciales significa que deben ser ejecutados uno después del otro y que sean pasos

ordenados quiere decir que deben llevar un orden quasi-obligatorio… (Trejos 18).

Bajo este mismo orden de ideas, podemos concluir que para poder implementar la solución de un problema mediante el uso de una computadora es necesario establecer un conjunto de pasos ordenados que permitan alcanzar un resultado; los cuales, dentro del ámbito de la computación, constituyen lo que se conoce como Algoritmo.

1.1.1.1 Características, partes y representación de los Algoritmos

En términos generales, Pinales y Velázquez autores del libro Problemario de Algoritmos resueltos con Diagramas de Flujo y Pseudocódigo, establecen que, un Algoritmo debe cumplir con las siguientes características fundamentales:

1. Preciso. Debe indicar el orden en el cual debe realizarse cada uno de los pasos que conducen a la solución del problema.
2. Definido. Esto implica que el resultado nunca debe cambiar bajo las mismas condiciones del problema, éste siempre debe ser el mismo.
3. Finito. No se debe caer en repeticiones de procesos de manera innecesaria; deberá terminar en algún momento (Pinales y Velázquez 15).

Por ejemplo, para obtener la suma y resta de dos números proporcionados por un usuario. Los pasos requeridos para determinar la solución al problema (Algoritmo), serían como los que se ilustran en la siguiente imagen:

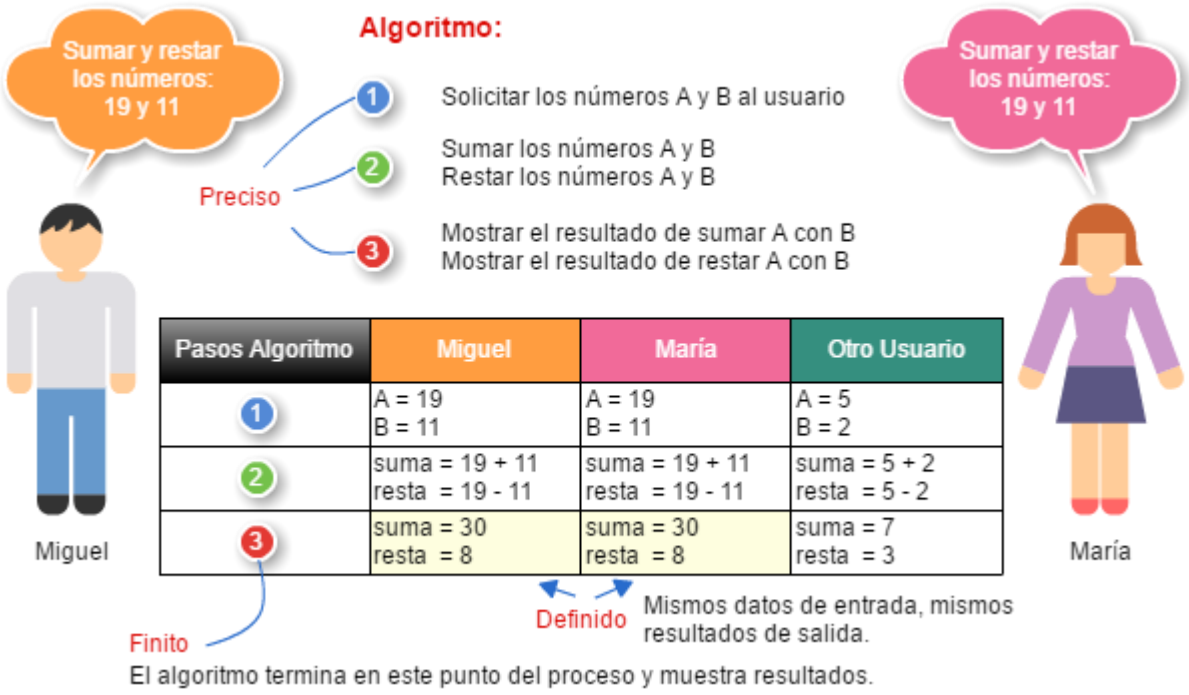


Imagen 1: Características de los Algoritmos - Fuente: Diseñada por el autor.

Si se observa detalladamente la imagen anterior, el Algoritmo describe tres pasos a seguir (Preciso) para encontrar la solución al problema. En primer lugar, se requiere conocer los dos números a sumar y restar, para ello, el paso número uno, declara una instrucción de solicitud de información al usuario. En seguida, el paso dos establece que los números proporcionados por el usuario se deben sumar y restar. Finalmente, en el paso tres se establece que se deben mostrar los resultados calculados, marcando así, el término de las tareas correspondientes a dicho Algoritmo (Finito).

Sumado a lo expuesto, para probar la eficiencia del Algoritmo (véase la tabla ilustrada en la imagen 1), los usuarios Miguel y María ingresan los datos 19 y 11 como respuesta a la solicitud de información indicada en el paso uno. Posteriormente, estos números son sumados y restados con la ayuda de los operadores de adición (+) y sustracción (-) declarados en el

paso dos; y como es de suponerse, en el paso tres, se dan a conocer sus correspondientes resultados “30 y 8”.

Llegado a este punto, se puede establecer que el Algoritmo anterior se encuentra bien definido, puesto que, al existir los mismos datos de entrada (19 y 11), este arroja los mismos resultados de salida (30 y 8). Ahora bien, en un escenario diferente (Otro Usuario), los resultados calculados por el Algoritmo siguen siendo correctos (7 y 3), sin embargo, estos se encuentran relacionados con base en los nuevos datos de entrada (5 y 2).

Bajo este mismo orden de ideas, durante la elaboración de un Algoritmo se deben describir tres partes fundamentales, las cuales son:

- Entrada: Describe los datos que se necesitan para poder comenzar a solucionar el problema.
- Proceso: Describe los cálculos y operaciones necesarias para alcanzar el resultado esperado (solución del problema).
- Salida: Describe la información que se mostrará como respuesta al problema.

Para poder ejemplificar cada uno de estos conceptos, la siguiente imagen expone una problemática sencilla, misma que es solucionada a partir del conocimiento de algunos datos que aparecen inmersos en la descripción del problema:

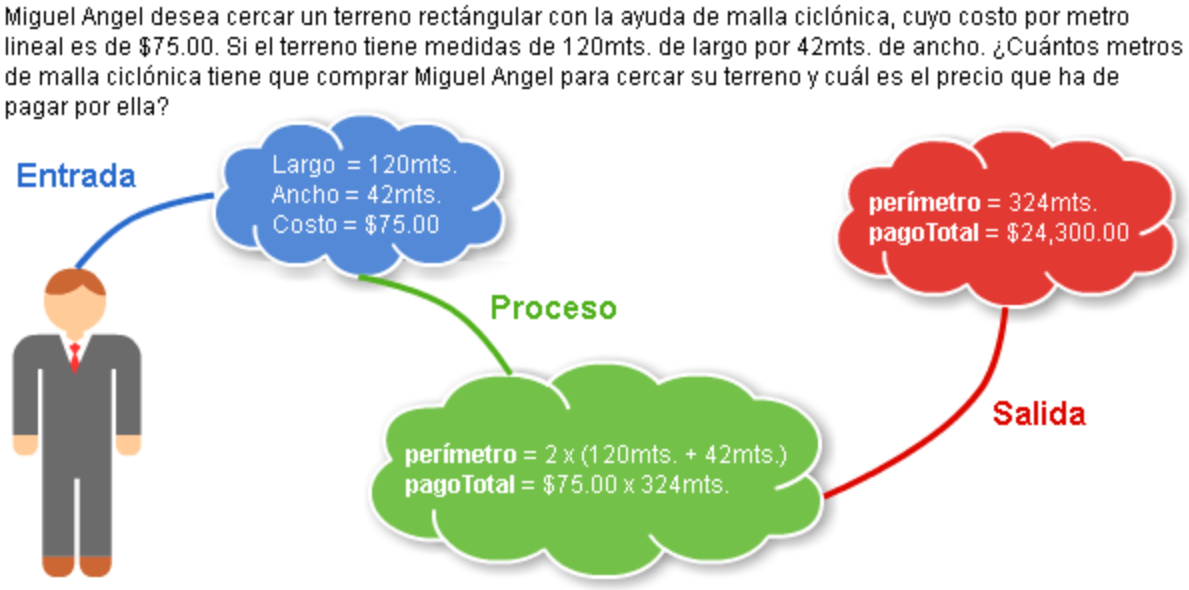


Imagen 2: Partes de un Algoritmo - Fuente: Diseñada por el autor.

Como se puede observar, la información proporcionada a un Algoritmo hace referencia a su entrada. Posteriormente, como parte del proceso, los datos de entrada son manipulados a través de diferentes cálculos matemáticos, los cuales generan uno o más resultados que constituyen su correspondiente salida.

En este sentido, se puede decir que, para poder solucionar un problema mediante el uso de un ordenador, es necesario que durante su análisis se identifiquen correctamente sus datos de entrada; de lo contrario, en la fase de proceso, será imposible determinar su correspondiente solución.

Ahora bien, para representar cada una de las instrucciones que aparecen listadas en un Algoritmo, se suele utilizar diferentes herramientas de diseño, cada una de ellas con sus ventajas y desventajas. Por ejemplo:

- Lenguaje natural: Es sencillo de utilizar, los pasos necesarios para la solución del problema son descritos en un lenguaje convencional, es

decir, en un lenguaje hablado por humanos para poder comunicarse. Sin embargo, debido a su sencillez, es posible que se presenten problemas de ambigüedad (una instrucción puede interpretarse de muchas formas), lo que provoca que sus descripciones se tornen algo difíciles de comprender.

- Diagrama de Flujo: Consiste en la representación gráfica de un Algoritmo, cada paso del proceso es representado por un símbolo diferente (caja) que contiene una breve descripción de la tarea realizada. A su vez, cada uno de estos símbolos se encuentra unido mediante el uso de flechas (denominadas líneas de flujo), mismas que indican la secuencia u orden en que las tareas (acciones) se deben ejecutar. Sin embargo, para la solución de problemas complejos, este tipo de técnica no resulta ser la más favorable, ya que su elaboración suele ser laboriosa.
- Pseudocódigo: Es un lenguaje de especificación (descripción) de Algoritmos, utiliza una notación similar a la de un lenguaje de programación estándar, pero las palabras clave que utiliza están escritas en lenguaje castellano (lenguaje natural estandarizado). Si bien es cierto que su proximidad a un lenguaje de programación puede resultar en un obstáculo al inicio para un estudiante, la idea es que éste pueda ver la solución de un problema desde un punto de vista más cercano a la realidad (como un programa de computadora).

Para ilustrar cada una de estas herramientas (de diseño) durante la fase de elaboración de un Algoritmo, la siguiente imagen comparativa expone los pasos a seguir para determinar el promedio general de dos calificaciones parciales obtenidas por un alumno a lo largo de un semestre:

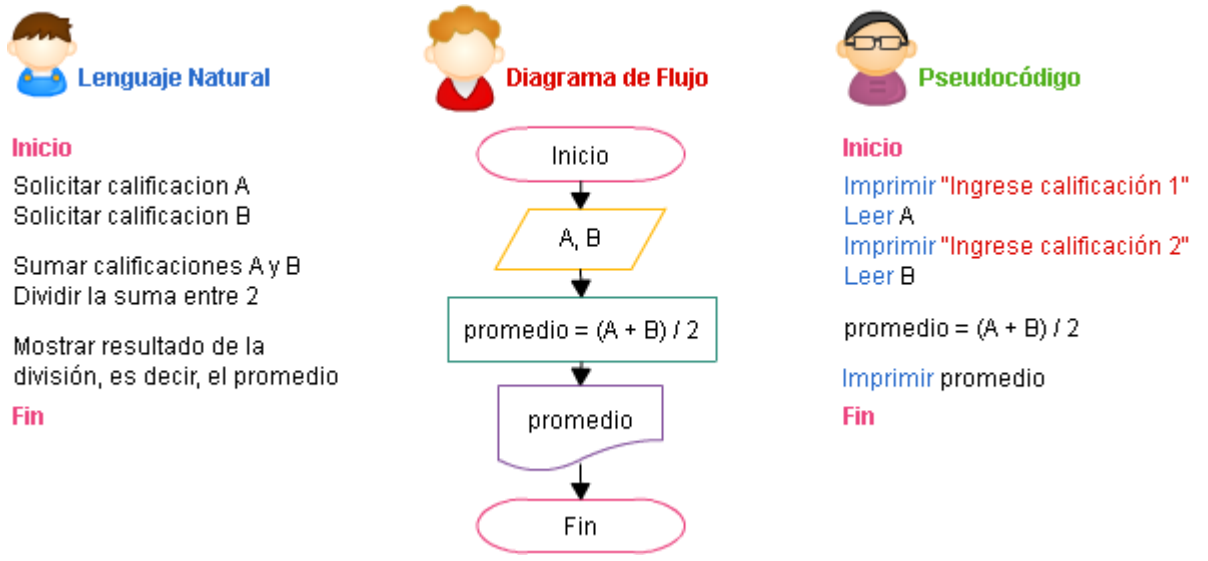


Imagen 3: Herramientas para el diseño de Algoritmos - Fuente: Diseñada por el autor.

Como se puede observar, ambos diseños atienden la misma problemática, pero, desde una perspectiva diferente. El Algoritmo elaborado marca un punto de inicio y un punto final (finito), cada una de sus instrucciones indican lo que se debe hacer y la suma de estas conducen a la solución del problema (preciso). Al ingresar los mismos datos de entrada, por ejemplo, **A = 10** y **B = 9.6**, la fórmula **promedio = (10 + 9.6) / 2**, siempre retorna el mismo resultado de salida "**9.8**", es decir, cada una de las instrucciones declaradas en el Algoritmo se encuentran bien definidas.

Vinculado al concepto, para poder determinar la solución al problema. En un primer momento, el Algoritmo describe los datos de entrada necesarios para calcular el promedio general, es decir, las dos calificaciones parciales obtenidas por el alumno a lo largo del semestre. Luego, como parte del proceso, los datos de entrada son manipulados mediante una fórmula matemática, misma que, al término de las operaciones, retorna un resultado que se corresponde con el promedio de los datos de entrada (es decir, las dos calificaciones parciales). Finalmente, el promedio general es

mostrado al usuario (salida de información), marcando así, el desenlace de las operaciones correspondientes al Algoritmo.

Con base en la situación descrita, se puede comprobar que el Lenguaje natural como herramienta de diseño de Algoritmos es sencillo de utilizar, cualquier persona que pueda establecer comunicación directa con otra, debería estar capacitada para escribir instrucciones a ejecutar por una computadora; sin embargo, tal y como se ha comentado con anterioridad, esta sencillez provoca que la redacción de algunas instrucciones, en su momento se tornen algo complejas de comprender, debido a que por naturaleza el lenguaje natural es intrínsecamente ambiguo. Por ejemplo, en una receta de cocina, es común encontrarse con pasos del tipo "**una pizca de sal**" y "**agregar al gusto**"; al no especificar claramente la cantidad de ingredientes a utilizar, el platillo final (resultado) puede verse afectado, debido a la interpretación que le dé el usuario en su momento.

Ahora bien, con respecto al Diagrama de Flujo en apariencia su elaboración[1] se puede presentar como una tarea sencilla. Puesto que cada instrucción de entrada, proceso y salida aparece descrita dentro de una caja con forma diferente. Sin embargo, a medida que se diseñan soluciones a problemas complejos con la ayuda de este tipo de herramienta, suele convertirse en una actividad laboriosa, debido a que en ocasiones requieren de una cantidad de espacio considerable para su correspondiente ilustración. Además, cuentan con demasiadas ramificaciones al momento de tomar una decisión y ante el error (de una sola instrucción), es posible que se invierta un tiempo considerable en atender su re-estructuración (véase la imagen 4).

[1] Las flechas ayudan a comprender el orden en que se debe ejecutar cada una de las instrucciones para obtener el resultado esperado. Además, para limitar su alcance, se utilizan pequeños indicadores (en forma de ovalo) que marcan el inicio y final de las acciones llevadas a cabo por el Algoritmo.

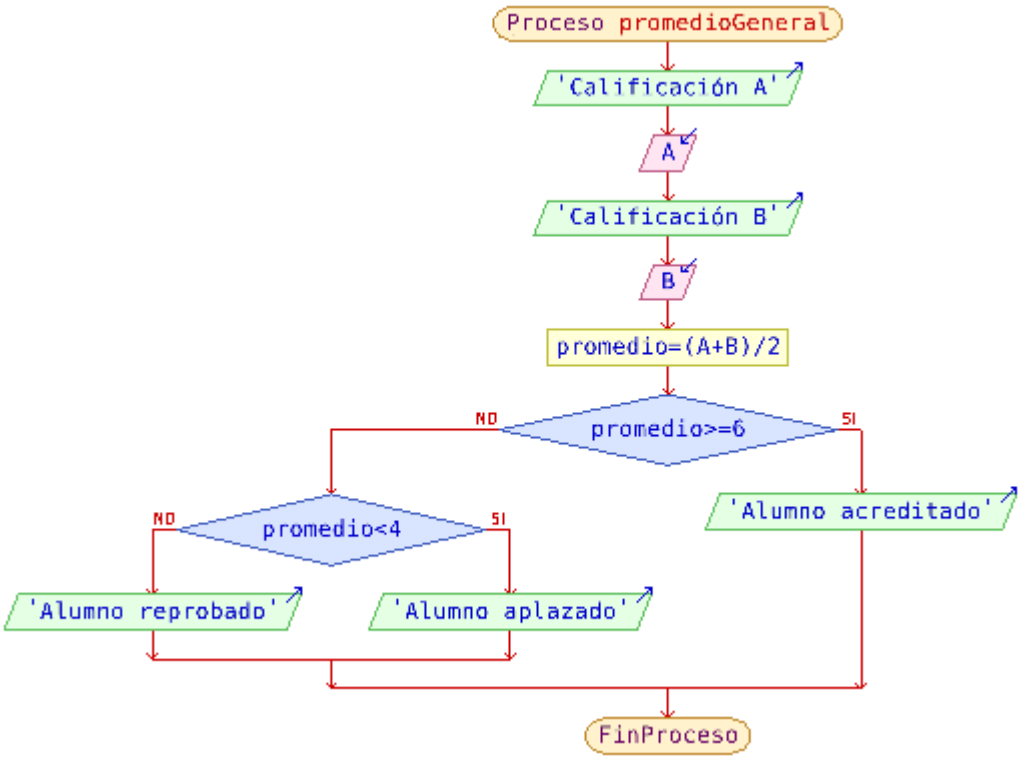


Imagen 4: Diagrama de Flujo, una actividad laboriosa - Fuente: Diseñada por el autor.

Finalmente, al igual que el Lenguaje natural, el Pseudocódigo como herramienta de diseño de Algoritmos utiliza instrucciones escritas en español (salvo las fórmulas matemáticas) que especifican paso a paso las acciones que se deben ejecutar para resolver una problemática en cuestión. Sin embargo, el lenguaje empleado para declarar cada una de estas instrucciones, se encuentra debidamente estandarizado (similar a los lenguajes de programación), es decir, para mostrar información al usuario se hace uso de la palabra reservada "**Imprimir**" (salida), en caso de que el Algoritmo necesite información proveniente desde el exterior se utiliza el término "**Leer**" (entrada) y, para realizar algún tipo de procedimiento

riguroso, se emplean operaciones básicas[2] construidas a partir de diferentes operadores aritméticos.

Llegado a este punto, se puede concluir que durante la elaboración de un Algoritmo este debe cumplir con tres características fundamentales: 1) debe indicar el orden de ejecución de cada paso, 2) los resultados arrojados nunca deben cambiar bajo las mismas condiciones iniciales del problema y, 3) sus acciones deben terminar en algún momento dado. Además, para garantizar que el Algoritmo se conduce en la dirección correcta, este debe describir claramente los datos de entrada que se necesitan para comenzar a abordar el problema, indicar los cálculos o procedimientos necesarios para poder decretar una solución y, describir la información que se mostrará como respuesta (salida). Finalmente, cada una de las instrucciones declaradas en un Algoritmo pueden ser descritas a través de diversas herramientas de diseño, tales como: lenguaje natural, diagramas de flujo y pseudocódigo; ambas permiten ilustrar detalladamente las acciones a seguir, sin embargo, la pertinencia de cada una de ellas depende directamente de la complejidad del problema a resolver.

En este sentido, la herramienta que se utiliza a lo largo de este capítulo para la solución de problemas a través del ordenador, es el Pseudocódigo escrito en español; puesto que se considera importante que el lector comprenda las bases de la programación sin relacionarlas directamente con un lenguaje de programación en particular. Una vez que el lector asimile correctamente estos conceptos y los asocie en la solución de diferentes problemáticas del mundo real, fácilmente podrá migrar los

[2] Por ejemplo: sumas, restas, multiplicaciones, divisiones, potencias, radicales, entre otros.

Pseudocódigos realizados a un lenguaje de programación formal, tal como es el caso de PHP[3], JavaScript[4], Python[5], entre otros.

1.1.2 Pseudocódigo

En el ámbito de la programación, el Pseudocódigo es una de las técnicas más utilizadas para el diseño de Algoritmos. "… Esta herramienta permite pasar casi de manera directa la solución del problema a un lenguaje de programación específico…" (Pinales y Velázquez 17). Puesto que su objetivo principal, consiste en que el programador centre toda su atención en determinar la solución al problema y, luego, tenga prioridad en la sintaxis o reglas que distinguen al lenguaje de programación por utilizar.

Según Joel de la Cruz Villar, "Todo pseudocódigo debe posibilitar la descripción de: Instrucciones de entrada/salida, Instrucciones de proceso, Sentencias de control del flujo de ejecución, [así como] Acciones compuestas [subprocesos]" (De la Cruz 28). Es decir, al ser una herramienta para el diseño de Algoritmos, estos deben contar con los medios necesarios para solicitar entradas de información, incorporar las instrucciones o sentencias que permitan alterar los valores de entrada durante el procesamiento de la solicitud, y poseer los mecanismos necesarios para emitir las salidas de información (resultados).

Bajo este mismo orden de ideas, Juan Carlos Casale en su obra Introducción a la Programación argumenta que:

[3] Es un lenguaje de programación del lado del servidor diseñado originalmente para el desarrollo de sitios Web dinámicos.
[4] Es un lenguaje de programación del lado del cliente utilizado principalmente para mejorar la interfaz de usuario (formularios de registro) correspondiente a un sitio Web.
[5] Es un lenguaje de programación multiplataforma que puede ser utilizado para el desarrollo de diferentes proyectos, tales como: aplicaciones Web, juegos, programas administrativos, hasta motores de búsqueda.

… el pseudocódigo es parecido a un lenguaje de programación en su escritura y, como tal, contiene un determinado **léxico**. Se trata de letras o caracteres que serán válidos para escribir las instrucciones que deseamos transmitir. La **sintaxis** es la especificación de palabras clave en combinación con otras que usaremos para formar las oraciones. Por último, la **semántica** es el significado que les daremos a dichas frases (Casale 71). Por ejemplo:

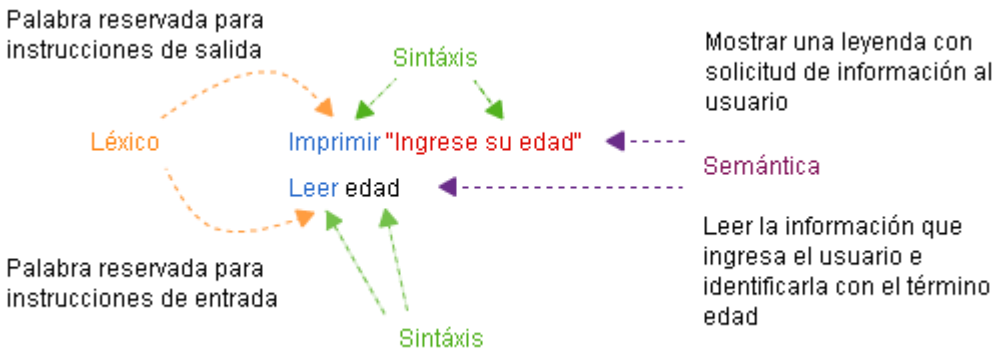


Imagen 5: Léxico, Sintaxis y Semántica - Fuente: Diseñada por el autor.

Como se puede observar, palabras reservadas como **Imprimir** y **Leer**, es parte de la terminología del Pseudocódigo escrito en español que facilita considerablemente el aprendizaje y uso de los fundamentos de la programación, puesto que la mayoría de estas instrucciones son por poco similares a las que utilizan los lenguajes de programación reales (véase la imagen 6) para poder comunicarse con el ordenador.

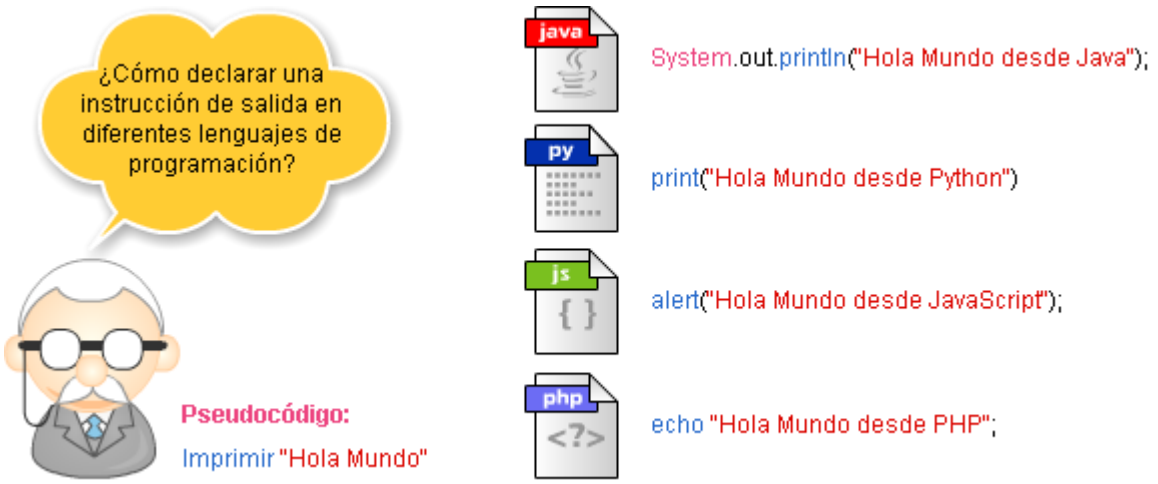


Imagen 6: Instrucciones de salida declaradas en diferentes lenguajes de programación - Fuente: Diseñada por el autor.

Llegado a este punto, se puede deducir que la tarea intelectual, es decir, la que requiere de una capacidad de pensamiento profundo y razonamiento crítico, **es la que se corresponde con la construcción del Pseudocódigo**; puesto que representa la solución detallada del problema. En cuanto a la escritura del programa, es una labor que puede resultar sencilla, siempre y cuando se conozca a detalle las reglas semánticas y sintácticas que constituyen al lenguaje de programación seleccionado.

En el ámbito de la computación, los Algoritmos son más importantes que los lenguajes de programación o las computadoras. Un lenguaje de programación es tan solo un medio para expresar un Algoritmo (el cual puede venir representado en forma de Pseudocódigo o Diagrama de Flujo) y una computadora es solo un procesador para ejecutarlo. Ambos persiguen una finalidad en común: conseguir que el Algoritmo se ejecute y lleve a cabo todas sus tareas correspondientes. En este sentido, Cairó establece que un programa:

> … es un conjunto de instrucciones que sigue la computadora para alcanzar un resultado específico. El programa se escribe en un

> lenguaje de programación a partir de un diagrama de flujo [o pseudocódigo] diseñado con anterioridad (Cairó 31).

Con base en los supuestos anteriores, se puede decir que el programar consiste en establecer una comunicación directa con una computadora para indicarle cuáles serán las instrucciones y el orden en que se deben ejecutar para llevar a cabo una determinada tarea. Los programas se basan en Pseudocódigos o en Diagramas de Flujo, los cuales nos ayudan a solucionar algún tipo de problema. Por tanto, si un programa es la traducción de un Algoritmo, entonces éste también debe heredar sus características básicas, es decir, tiene que ser preciso, estar bien definido y, por consiguiente, ser finito.

Resumiendo lo tratado, podemos concluir que un Pseudocódigo es un lenguaje falso que permite describir de manera informal las instrucciones declaradas en un Algoritmo. La terminología y sentencias de control que utiliza para ilustrar dichas instrucciones permite pasar casi de manera inmediata la solución de un problema a un lenguaje de programación específico. Su pertinencia como herramienta de diseño de Algoritmos radica en que es fácil de entender por un ser humano, es parecido a un lenguaje de programación en su escritura y permite modelar soluciones a problemas con un mayor grado de complejidad.

1.1.2.1 Pseudo-Intérprete

Un Pseudo-Intérprete consiste en un entorno de desarrollo integrado (véase la imagen 7) que proporciona las herramientas necesarias para facilitar al programador la escritura, ejecución y prueba de los Algoritmos

mediante la técnica del pseudocódigo; como si estuvieran codificados en un lenguaje de programación real.

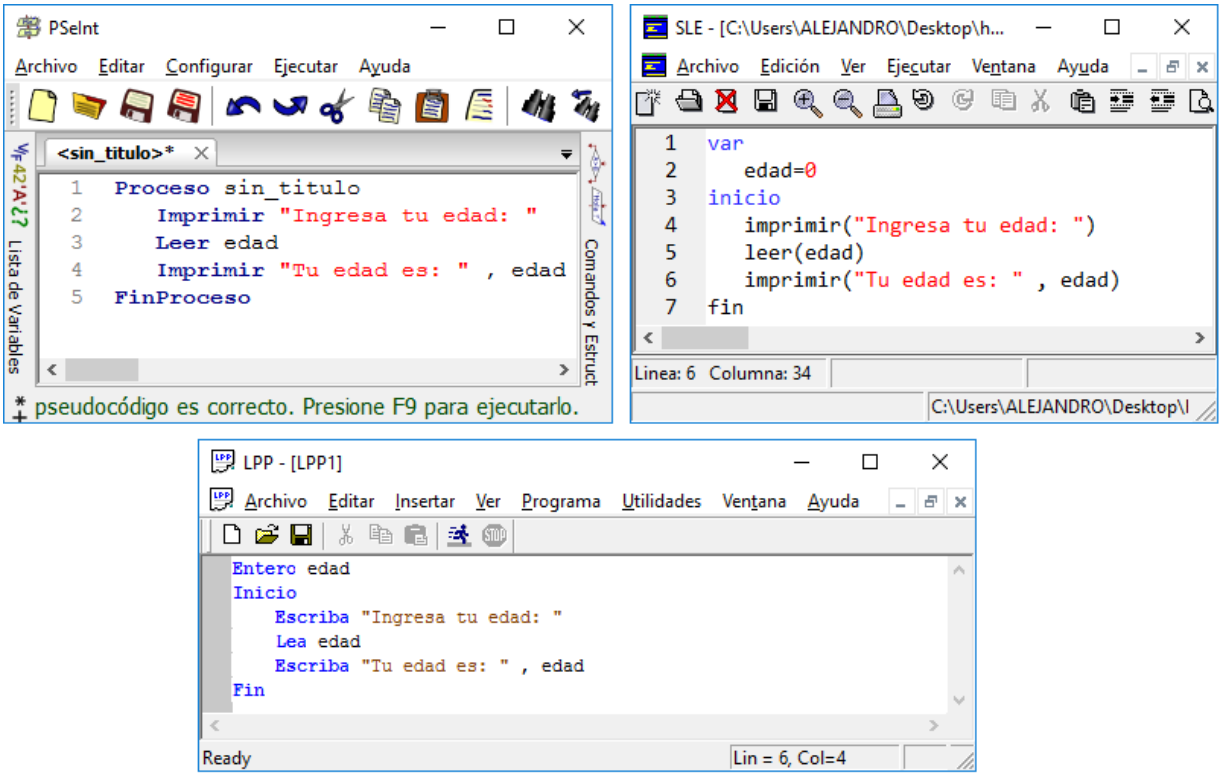


Imagen 7: Ejemplos de *software* del tipo Pseudo-Intérprete - Fuente: Diseñada por el autor.

Como ya se sabe, un Pseudocódigo es un lenguaje falso que permite describir las instrucciones declaradas en un Algoritmo para que puedan ser leídas fácilmente por un ser humano. Sin embargo, éstas no pueden ser interpretadas directamente por un ordenador, dado que no se encuentran codificadas previamente a un lenguaje de programación.

Es en este punto donde entra en juego la importancia de un Pseudo-Intérprete, ya que, gracias a sus herramientas, permite que el ordenador interprete (ejecute) el pseudocódigo capturado y, por consiguiente, sea posible observar su funcionamiento y verificar los resultados generados. Tal y como se muestra en la siguiente imagen:

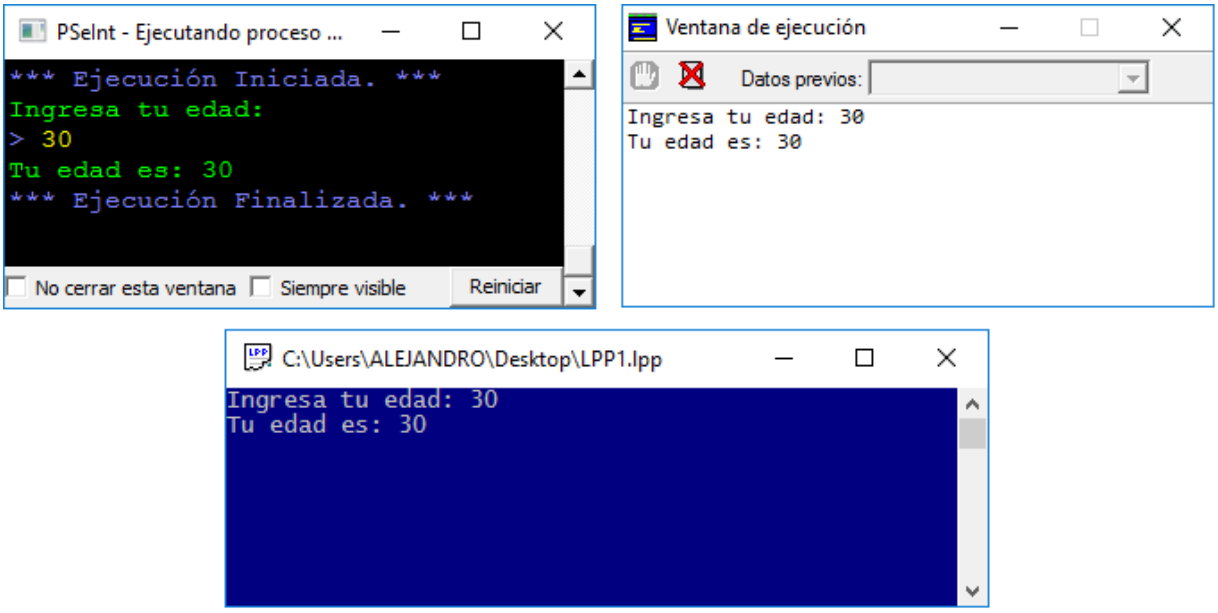


Imagen 8: Ejecución de Pseudocódigo mediante el uso de Pseudo-Interpretes - Fuente: Diseñada por el autor.

PSeInt es una herramienta de *software* libre multiplataforma[6] (Windows, Linux y MacOS) desarrollada especialmente para asistir a un estudiante en sus primeros pasos en programación. Mediante un simple e intuitivo Pseudolenguaje en español, le permite centrar su atención en los conceptos fundamentales de la algoritmia computacional, debido a que minimiza las dificultades propias que conlleva adoptar de un lenguaje de programación y proporciona un entorno de trabajo interactivo con numerosas ayudas y recursos didácticos.

El pseudocódigo se suele utilizar comúnmente como primera técnica de contacto para introducir a los estudiantes en los conceptos básicos de la programación, sin tener que lidiar con las particularidades de sintaxis de un lenguaje real. Identificadores, variables, constantes, expresiones, operaciones aritméticas, operadores de comparación, estructuras de control, entre otros; son solo algunos de los elementos gramaticales que todo estudiante debe afianzar para sumergirse en el mundo de la

[6] Programa o aplicación que puede utilizarse en diversos Sistemas Operativos.

programación. Es por ello que, para el desarrollo de los siguientes apartados correspondientes a este capítulo de la investigación, se trabajará de cerca con el *software* PSeInt, puesto que facilita la tarea de escribir Algoritmos a través de un conjunto de herramientas adicionales que permiten encontrar y atender los errores cometidos durante la fase de su codificación, así como comprender la lógica y comportamiento que guarda cada uno de ellos durante el proceso de ejecución.

Llegado a este punto, podemos concluir que el Pseudocódigo es un lenguaje informal diseñado para la lectura humana y no para ser interpretado directamente por un ordenador. Dada esta condición, los Pseudo-Intérpretes actúan como verdaderos entornos de desarrollo, puesto que ofrecen un conjunto de herramientas que posibilitan al programador capturar, ejecutar y probar sus pseudocódigos; como si estuvieran codificados en un lenguaje de programación real.

Una vez realizada la primera toma de contacto con los Algoritmos y el pseudocódigo como herramienta para su diseño, dedicaremos el siguiente apartado en descubrir cuáles son los elementos básicos presentes en el diseño de pseudocódigos para la solución de problemas mediante el uso del ordenador.

1.2 Aspectos básicos presentes en el Pseudocódigo

En el apartado anterior se dio una introducción a los Algoritmos y Pseudocódigos, a través de cuatro ejemplos se presentó la manera de cómo solucionar un problema mediante la redacción de una serie de pasos ordenados. Sin embargo, como se comprobará dentro de poco, escribir

Pseudocódigos sirve para mucho más que solicitar información al usuario para procesarla mediante operaciones de suma o resta, obtener un promedio, o determinar el perímetro de una figura geométrica.

En este apartado inicialmente se presentan los conceptos básicos del Pseudolenguaje, tales como: los identificadores, las variables y constantes, las entradas y salidas de información, y los tipos de datos. Posteriormente, se hace un recorrido por los operadores aritméticos, los operadores de comparación (relación) y los operadores lógicos. Finalmente, se expone la prioridad entre operadores (orden de prelación) y la forma de incluir comentarios (documentación técnica) en los respectivos pseudocódigos.

Con la finalidad de mejorar la comprensión de los contenidos presentados en este numeral, se llevará a cabo la unificación de los términos Algoritmo y Pseudocódigo bajo la expresión de "Pseudocódigo digital". Es importante recordar que PSeInt es una herramienta de *software* interactiva que permite escribir, ejecutar y probar los pseudocódigos como si se tratase de un lenguaje de programación real. En este sentido, desde la perspectiva del propio investigador, es pertinente que a partir de este momento el lector se familiarice con el término de "Pseudocódigo digital", puesto que cada uno de los ejemplos abordados dentro de este apartado, están codificados mediante el Pseudolenguaje que propone PSeInt; y al momento de su ejecución, estos tienen un comportamiento similar al de un programa normal de computadora. A continuación, la explicación de las bases fundamentales de la algoritmia computacional.

1.2.1 Identificadores

Los identificadores son utilizados por los programadores para señalar por un nombre auto-explicativo aquellos espacios en Memoria RAM[7] dónde se almacena de forma temporal algún tipo de información o dato, los cuales más adelante se deben invocar a través del Pseudocódigo digital para su correspondiente procesamiento y poder emitir así una serie de resultados al usuario final. Según Pinales y Velázquez en su obra Problemario de Algoritmos resueltos con Diagramas de Flujo y Pseudocódigo, argumentan que:

> Los identificadores son los nombres que se les asignan a los objetos, los cuales se pueden considerar como variables o constantes, éstos intervienen en los procesos que se realizan para la solución de un problema... (Pinales y Velázquez 16)

En este sentido, se puede decir que, durante el proceso de desarrollo de un Pseudocódigo digital, es importante que el programador identifique previamente los datos que se necesitan para comenzar a trabajar, y a su vez, proceda a asignarles un nombre auto-explicativo que permita su posterior localización en Memoria RAM con la finalidad de operarlos (tal como: sumarlos, restarlos, multiplicarlos, dividirlos, etc.) durante el proceso de solución del problema. Por ejemplo, para sumar dos números proporcionados por un usuario, es necesario identificar tres espacios en Memoria RAM para el almacenamiento de los datos: dos de ellos para los operandos (números otorgados por el usuario), y el restante para el resultado generado (suma de los dos números); tal como se muestra en la siguiente figura:

[7] Memoria principal de la computadora, donde residen programas y datos, sobre la que se pueden efectuar operaciones de lectura y escritura de forma temporal.

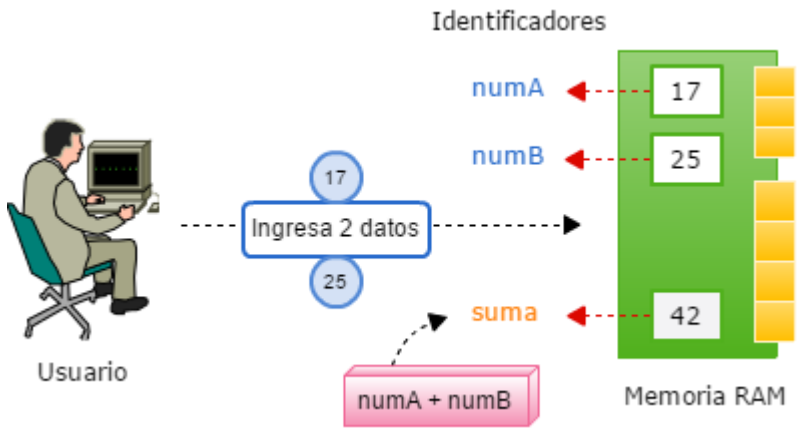


Imagen 9: Declaración de identificadores para la suma de dos números - Fuente: Diseñada por el autor.

Con base en la imagen anterior, es importante destacar que el nombre de un identificador se forma de acuerdo a ciertas reglas, mismas que pueden tener alguna variante de acuerdo al lenguaje de programación utilizado. Por tal motivo, para los efectos prácticos que persigue este apartado de la investigación, se deben considerar los siguientes criterios tras su declaración:

1. Deben comenzar obligatoriamente por una letra (a-z A-Z).
2. El resto de caracteres pueden contener: letras, números (0-9), y el símbolo de guion bajo (_).
3. No pueden contener espacios, comas, puntos, ni operadores (+, -, *, /, ^).
4. No pueden contener acentos, diéresis, ni eñes.
5. No deben coincidir con palabras reservadas del lenguaje para no generar ambigüedad. Algunas palabras clave utilizadas por PSeInt al momento de redactar estas líneas de la investigación son: **Imprimir**, **Leer**, **Si**, **Sino**, **Según**, **Hacer**, **Mientas**, **Repetir**, **Para**, **Y**, **O**, **NO**, entre otras. Para mayor información consulte el Anexo correspondiente a PSeInt en su apartado palabras y funciones reservadas del lenguaje.

6. El nombre asignado a un identificador debe estar relacionado con la información referenciada. Por ejemplo, si un identificador hace referencia a un dato que representa la edad de un usuario, posibles nombres válidos para este identificador serían: **edad**, **edad_usuario**, **edadUsuario**, etc.

Bajo este mismo orden de ideas, la siguiente tabla muestra una serie de ejemplos con la declaración de identificadores válidos y no válidos, mismos que pueden ser utilizados durante el proceso de desarrollo de un Pseudocódigo digital.

Identificadores válidos	Identificadores no válidos	Error generado
precioProducto	Precio del producto	Espacios
descuento	$_descuento	Símbolo de pesos $
areaTriangulo	Area_triángulo	Acento
hipotenusa	c	Falta de contexto
iva	i.v.a	Puntos

Tabla 1: Identificadores válidos y no válidos - Fuente: Elaborada por el autor.



¿Por qué utilizar identificadores?

Los identificadores juegan un papel importante en el desarrollo de los Pseudocódigos digitales, puesto que permiten registrar por un nombre auto-explicativo los datos con los que se trabajará a lo largo de la solución de un problema.

Una vez que se ha abordado el concepto de identificador, su importancia, y los criterios para su declaración. Es momento de conocer los diferentes tipos de identificadores que intervienen durante el proceso de desarrollo de un Pseudocódigo digital, es decir, las variables y las constantes.

1.2.1.2 Variables

Según Juan Carlos Casale en su obra Introducción a la Programación, establece que "… una variable es un espacio en la memoria de la computadora que permite almacenar temporalmente un dato durante la ejecución de un proceso, y cuyo contenido puede cambiar mientras corre un programa" (Casale 75). Como se puede inferir, se trata de un tipo especial de identificador que hace referencia a información que puede variar durante el proceso de solución de un problema. Por ejemplo: determinar el descuento a otorgar a un cliente con base en la compra de ciertos artículos, calcular el sueldo a pagar a un trabajador conforme a su salario y los días trabajados, así como obtener la hipotenusa de un triángulo rectángulo a partir de datos conocidos como son su base y altura.

Como se puede observar en el último ejemplo, tanto la **base** como la **altura** son variables que se proporcionan al Pseudocódigo digital para que éste pueda funcionar adecuadamente, y el valor referenciado en **hipotenusa**, es decir, la solución al problema, depende directamente de los datos que se le proporcionen (base y altura).

A manera de ejemplo ilustrativo, la siguiente imagen representa el proceso para determinar la hipotenusa de un triángulo rectángulo a partir de dos datos conocidos, como son: su "base y altura", mismos que son proporcionados previamente por el usuario final.

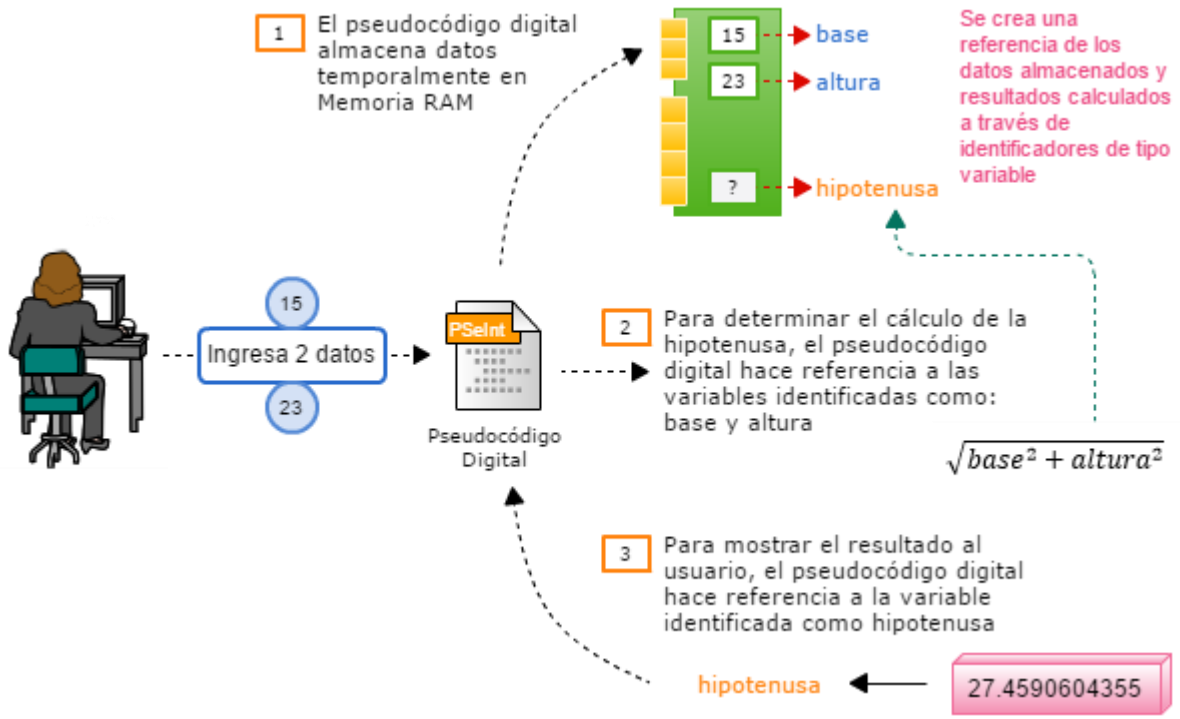


Imagen 10: Declaración de identificadores de tipo variable para determinar la hipotenusa de un triángulo rectángulo - Fuente: Diseñada por el autor.

De la imagen anterior, se puede observar que existen dos formas diferentes para hacer referencia a un valor almacenado en Memoria RAM del ordenador. Por un lado, se tienen los datos de entrada que proporciona el usuario al Pseudocódigo digital (**base** y **altura**), y por el otro, el resultado de un proceso operacional (**hipotenusa**) que se realiza internamente como parte del conjunto de instrucciones que son indicadas por el programador para la solución del problema. Estos aspectos son considerados dentro de la documentación de PSeInt como "**la lectura** y **la asignación**" de valores a una variable.

Bajo este mismo orden de ideas, se puede decir que para almacenar un conjunto de datos de entrada (que son proporcionados por un usuario) en Memoria RAM, previamente se tiene que hacer una lectura de esos datos y, por consiguiente, proceder a asignarles un identificador (de tipo

variable) que permita hacer referencia a cada uno de ellos dentro del Pseudocódigo digital. Por tanto, la siguiente ilustración muestra un ejemplo válido en PSeInt para la lectura de información:

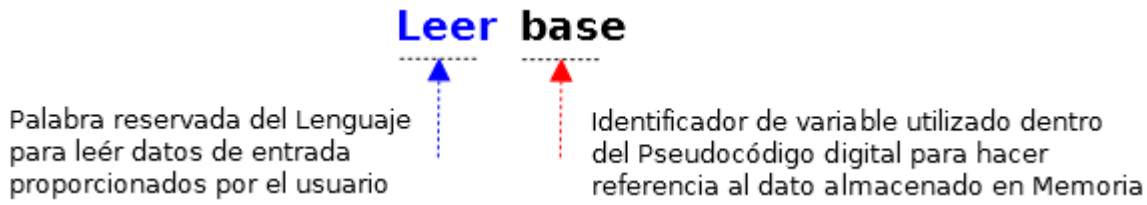


Imagen 11: Proceso de lectura de información en PSeInt - Fuente: Diseñada por el autor.

Del mismo modo, para **conocer** e **identificar** el valor proporcionado como altura (por el usuario), bastaría con declarar una instrucción como la siguiente: **Leer altura**.

Una vez que el Pseudocódigo digital cuenta con todos los datos necesarios provenientes del exterior, es decir, la **base** y la **altura** (para el caso de este ejemplo). Llega el momento de proceder a operar dichos datos mediante el uso de diversas fórmulas matemáticas, las cuales, tras finalizar su operación, retornan una serie de resultados que deben almacenarse temporalmente en Memoria RAM del ordenador para su posterior consulta dentro del Pseudocódigo digital. De este modo, resulta evidente hacer uso de las variables para hacer referencia a los datos retornados por una fórmula matemática. Si bien estos datos no pueden ser tratados a nivel de lectura (puesto que los datos no provienen de un proceso de entrada) como en el caso anterior; el proceso de **asignación** es el camino a seguir para el cumplimiento de este nuevo objetivo en cuestión.

En atención a la problemática expuesta en el párrafo anterior, la siguiente figura ilustra el proceso de asignar un valor retornado por una expresión matemática (fórmula) a una variable:

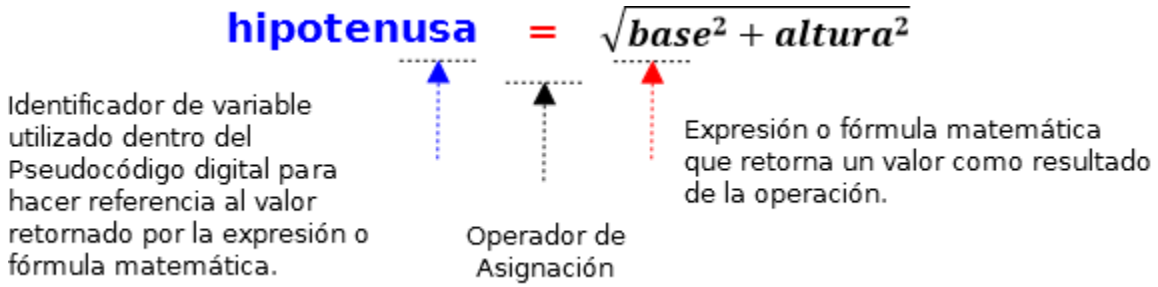


Imagen 12: Proceso de asignación de información a un identificador de tipo variable - Fuente: Diseñada por el autor.

Como se puede observar, el proceso de asignación consta de tres partes importantes: una variable, un operador de asignación y una expresión. Al ejecutarse una asignación, primero se evalúa el contenido de la expresión de la derecha y luego se asigna el resultado a la variable de la izquierda.

Sumado a lo expuesto, es importante mencionar que la asignación de nuevos valores a una misma variable provoca la pérdida total de los datos almacenados con anterioridad en dicho casillero, debido a que el proceso de sobre-escritura que se lleva a cabo internamente en Memoria RAM origina que prevalezcan los datos recientes sobre los anteriores. Tal y como se muestra en la siguiente figura:

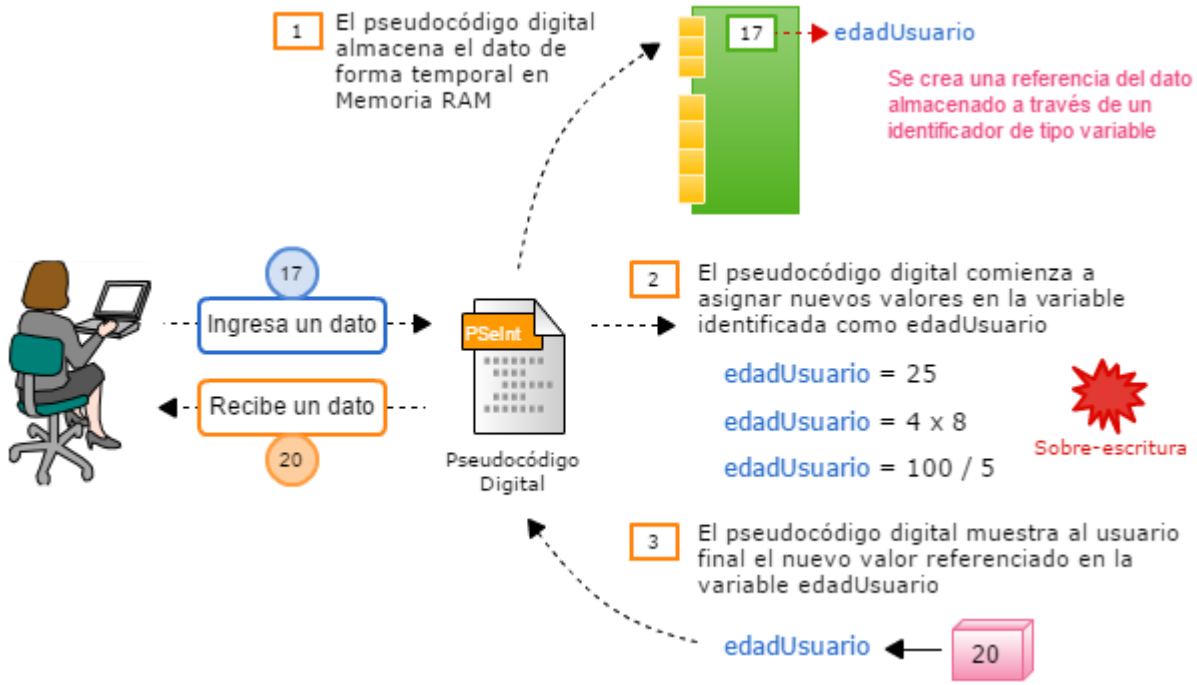


Imagen 13: Sobre-escritura de valores almacenados en una misma variable - Fuente: Diseñada por el autor.

Como se puede observar en la figura anterior, el dato con valor 17 proporcionado originalmente por el usuario es almacenado de forma directa en Memoria RAM del ordenador, e internamente en el Pseudocódigo digital es referenciado a través de un identificador de variable llamado **edadUsuario**. Más adelante como parte del proceso, dicha variable se le asigna un nuevo conjunto de valores y/o resultados, los cuales sobre-escriben de manera continua la información previamente referenciada, es decir; el valor 25 pasa a tomar el lugar que ocupaba originalmente el valor 17, posteriormente el valor 32 (resultado de multiplicar 4 por 8) sobre-escribe al valor 25, finalmente el valor 20 (resultado de dividir 100 entre 5) pasa a ocupar el lugar que tenía en memoria el valor 32. En este sentido, cuando el Pseudocódigo digital hace referencia nuevamente al valor almacenado en el casillero **edadUsuario**, este retorna como resultado un valor almacenado de edad igual a 20, y no el valor original de 17.

Por otra parte, Joel De la Cruz Villar mediante su libro Algoritmos y Diagramas de Flujo aplicados en PHP menciona que las variables "se utilizan para contener datos de distintos tipos: números, letras, cadenas de caracteres, valores lógicos, etc. ..." (De la Cruz 22). Visto de esta forma y orientando la atención al ejemplo principal tratado a lo largo de estas líneas (el cálculo de la hipotenusa de un triángulo rectángulo a partir de datos de entrada como son la base y la altura), resulta indiscutible el almacenamiento de datos de tipo numérico (ya sea de tipo entero o con punto decimal) para la solución del problema.

Sin embargo, es importante mencionar que acorde a la problemática a solucionar, puede haber ocasiones donde sea necesario hacer uso de distintos tipos de datos. Por ejemplo, para el desarrollo de un Pseudocódigo digital que determine el pago de sueldos y bonos de productividad a empleados que laboran dentro de una empresa; quizá sea conveniente solicitar como datos de entrada: el nombre del empleado (tipo cadena de caracteres), los días laborados (tipo numérico), el rango que desempeña dentro de la empresa (a través de un menú que permita seleccionar una opción, A, B, C; tipo carácter), así como el número de retardos que sumó a la quincena (tipo numérico). Finalmente, proceder a trabajar con dichos datos mediante una serie de criterios (fórmulas matemáticas) y/o políticas establecidas por la propia empresa, mismas que al concluir su ejecución, sirvan una serie de resultados relacionados con el pago a emitir a cada trabajador.

Con la intensión de ilustrar las ideas expuestas hasta el momento, la siguiente imagen muestra parte del proceso de referenciar diferentes tipos de datos a un conjunto de variables utilizadas dentro de un Pseudocódigo digital:

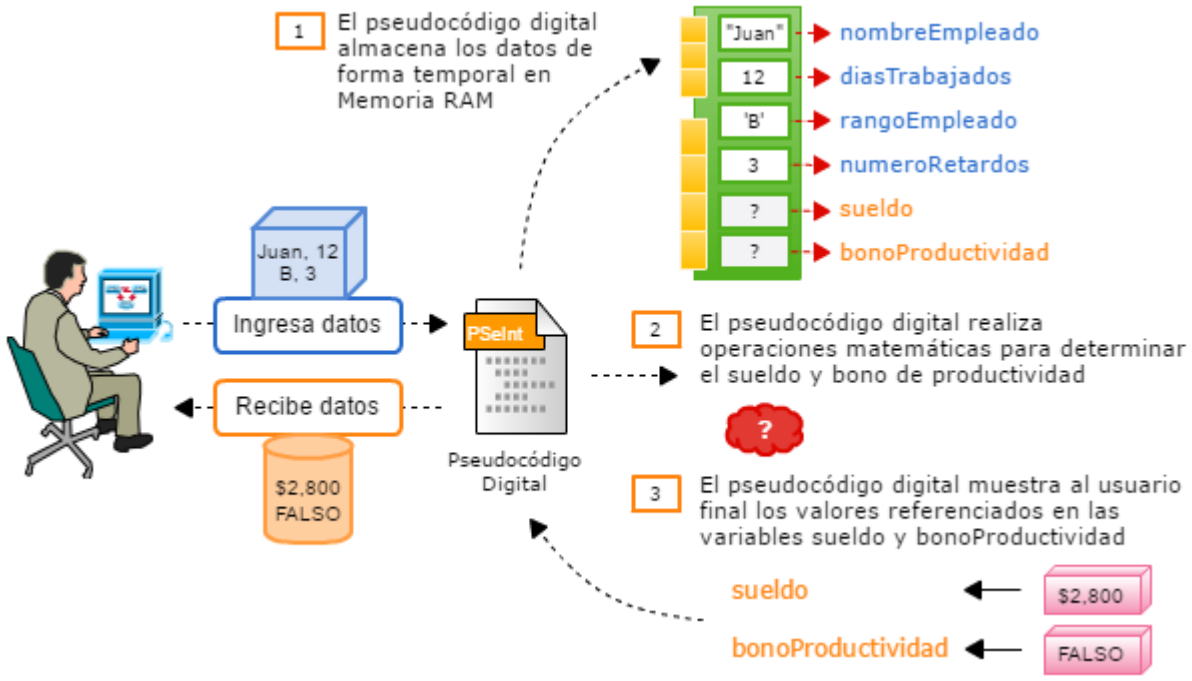


Imagen 14: Las variables pueden contener diversos tipos de datos - Fuente: Diseñada por el autor.

Si bien la imagen anterior no muestra los procedimientos operacionales llevados a cabo para la obtención del sueldo ($ 2,800) y el bono de productividad (FALSO) correspondientes a un trabajador; ésta permite formarse una idea, de cómo estos tipos de datos son almacenados y referenciados internamente en los diferentes casilleros (reservados) de la Memoria RAM del ordenador.

Más adelante dentro de este mismo capítulo, se dedica un apartado especial a los diferentes tipos de datos soportados por PSeInt, su declaración a partir de datos de entrada (proporcionados por el usuario), así como su correspondiente asignación en el proceso de solución del problema.



¿Por qué utilizar variables?

Las variables en los Pseudocódigos digitales siguen una lógica similar a las variables utilizadas en otros ámbitos como las matemáticas, es decir, son elementos que se emplean para almacenar y hacer referencia a otro valor que puede cambiar durante el proceso de solución de un problema. Gracias a las variables es posible crear "Pseudocódigos genéricos" que funcionan siempre igual independientemente de los valores concretos utilizados.

Si no existieran las variables, un Pseudocódigo digital que obtiene el promedio de dos calificaciones parciales podría escribirse como:

```
promedio = (7.5 + 9.2) / 2
```

Sin embargo, la solución anterior es poco eficiente, puesto que sólo sirve para el caso de que ambas calificaciones coincidan con los valores 7.5 y 9.2. En cualquier otro caso, la solución al problema fracasa y, por consiguiente, se obtiene un resultado incorrecto. Ahora bien, si el Pseudocódigo digital se rehace de la siguiente manera:

```
calificacionA = 7.5
calificacionB = 9.2
promedio = (calificacionA + calificacionB) / 2
```

Los elementos calificacionA y calificacionB son **variables** que almacenan los valores que el Pseudocódigo digital utiliza. El promedio se calcula siempre en función del valor almacenado por las variables, por lo que este Pseudocódigo digital funciona correctamente para cualquier par de calificaciones indicadas. Si se modifica el valor de las variables calificacionA y calificacionB, la solución al problema sigue funcionando correctamente.

Una vez comentado los aspectos generales que conlleva trabajar con las variables, es momento de dar cabida a otro tipo de identificador que es utilizado frecuentemente para hacer referencia a datos de naturaleza estática dentro de un Pseudocódigo digital, es decir, las constantes.

1.2.1.2 Constantes

Dentro del campo de la programación, se le considera constante a un tipo de identificador que hace referencia a un valor fijo que no puede cambiar a lo largo de la ejecución de un Pseudocódigo digital. Es decir, a diferencia de los valores referenciados mediante el uso de variables, las constantes no pueden sobre-escribir la referencia hacia sus contenidos, de modo que éstos siempre prevalecen intactos y, por consiguiente, siempre están disponibles a lo largo del proceso de solución del problema. Bajo este mismo orden de ideas, Francisco Javier Pinales y César Eduardo Velázquez ejemplifican que:

> …en problemas donde se utiliza el valor de PI[8], si el lenguaje que se utiliza para codificar el programa y ejecutarlo en la computadora no lo tiene definido, entonces se puede establecer de forma constante estableciendo un identificador llamado PI y asignarle el valor correspondiente de la siguiente manera:
> PI = 3.1416 (Pinales y Velázquez 16).

Como se puede observar, según el problema a solucionar mediante un Pseudocódigo digital, es probable que existan una serie de datos que puedan ser tratados internamente bajo el enfoque de constantes. Por ejemplo: el nombre de la empresa y/o desarrollador de un programa de computadora, el número de versión de una aplicación móvil, el impuesto al valor agregado (16%), el número e[9] (2.71828), etc. Frente a este escenario, cabe resaltar la similitud que existe en la forma de declarar y

[8] Es la relación entre la longitud de una circunferencia y su diámetro. Es una constante en geometría euclidiana.
[9] Es un número irracional importante en el área de las matemáticas. Se suele llamar número de Euler por Leonhard Euler. Su valor es aproximadamente igual a 2.71828 y se relaciona con muchos interesantes resultados, como ser la base de los logaritmos naturales y su aparición en el estudio del interés compuesto (préstamos bancarios).

asignar valores a una constante y a una variable; puesto que ambas hacen uso de un identificador, seguido de un signo de asignación (=), y por consiguiente el valor referenciado. En este sentido, y con la finalidad de hacer frente a dicha problemática, algunos lenguajes de programación proponen hacer uso de identificadores escritos con letras mayúsculas (PI) para la declaración de constantes, y la combinación de letras minúsculas y mayúsculas para la declaración de variables; tal y como se muestra en la siguiente tabla:

Declaración de Constantes	Declaración de Variables
PI = 3.1416	**Leer** edadUsuario
NOMBRE_EMPRESA = "MonoSoft"	centimetros = metros x 100
VERSION = 2.4	dolares = pesosMexicanos / 17.50
IVA = 0.16	situacionAcademica = "Aprobado"
BOLETO_METRO = 5	impuesto = subtotal x **IVA**

Tabla 2: Declaración de Constantes y Variables - Fuente: Elaborada por el autor.

Con base en los supuestos anteriores, se puede llegar a vislumbrar que el tipo de datos que pueden ser referenciados a través de una constante es el mismo que el de una variable. Sin embargo, existen otra serie de diferencias que es importante hacer notar tras el trabajo con estos dos tipos de identificadores:

1. Los datos asignados a una constante son más simples que los de una variable, debido a que carecen de cálculos o procedimientos aritméticos previos como es el caso de una fórmula matemática.
2. Los datos que provienen de un proceso de entrada (dígase por el usuario) no pueden ser almacenados a través de una constante, puesto que este valor no se conoce al inicio de la ejecución del

Pseudocódigo digital, y por tanto su valor cambiaría de acuerdo a las necesidades del usuario.

3. Las variables pueden hacer uso de los valores referenciados por una constante, con la finalidad de operarlos como parte del proceso de solución a un problema (impuesto = subtotal x **IVA**).

Llegado a este punto, es importante hacer énfasis en la diferencia que existe entre un valor constante y un valor referenciado mediante el uso de un identificador de tipo constante. Para ello, sírvase de ejemplo la siguiente imagen:

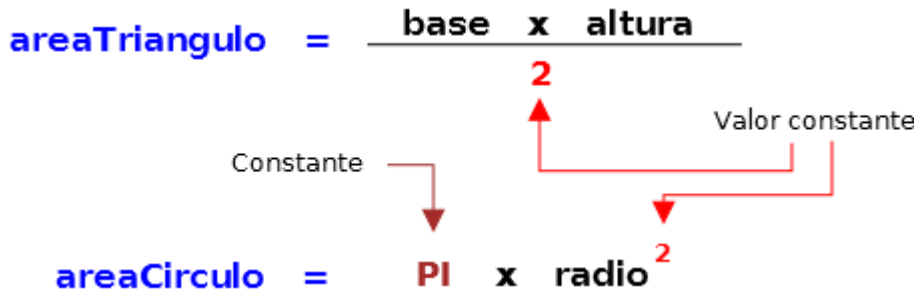


Imagen 15: Diferencia entre valor constante y valor referenciado por un identificador de tipo constante - Fuente: Diseñada por el autor.

Como se puede observar, el proceso de declarar un identificador de tipo constante para la referencia de valores estáticos dentro de un Pseudocódigo digital no es un proceso obligatorio del cual deba preocuparse seriamente un programador, más bien se trata de un estilo personal para la escritura de instrucciones o procesos que se deben llevar a cabo para la solución de un problema. Ahora bien, si durante el proceso de solución se observa la necesidad de requerir frecuentemente un valor (por ejemplo, 3.1416), lo mejor es asignarlo a un identificador de tipo constante cuya signatura (nombre del identificador) permita asociarlo fácilmente con el valor actualmente referenciado.

Bajo este mismo orden de ideas, al momento de redactar estas líneas de la investigación, PSeInt no cuenta con soporte para la declaración de constantes por parte del programador. Sin embargo, es importante mencionar que incorpora una constante pre-construida, la cual lleva por nombre PI y hace referencia al valor aproximado 3.14159265359.



¿Por qué utilizar constantes?

Una constante es un valor que no puede ser alterado o modificado durante la ejecución de un Pseudocódigo digital, únicamente puede ser leído. Por conveniencia, su nombre suele escribirse en mayúsculas en la mayoría de los lenguajes de programación.

Básicamente el uso de las constantes es igual que en las matemáticas. La constante toma un valor determinado y lo mantiene durante todo el problema (por ejemplo, **PI = 3.1416**). El hecho de que sean constantes permite conocer su valor independientemente de la situación.

La ventaja de usar una constante en lugar de colocar el valor directamente en una expresión, es que, si se define una constante y se le asigna un valor, este es automáticamente sustituido por esa constante en todo el Pseudocódigo digital, liberando al programador de tener que recordar en todo momento dicho valor, puesto que, cada vez que se requiera utilizar, con simplemente colocar el identificador de la constante, su valor asignado se recuperaría.

Una vez que el lector cuenta con las bases suficientes para el trabajo relacionado con identificadores, variables y constantes. Toca el turno a las entradas y salidas de información; después de todo, los Pseudocódigos digitales requieren de algún medio de comunicación que les permita interactuar con sus usuarios finales, y poder así, solicitar o entregar una serie de informaciones relacionadas con el proceso de solución del problema para el cual fueron desarrollados.



Resumen: Identificadores, variables y constantes.

En esta sección se abordaron los identificadores, las variables y las constantes. A estas alturas, se debería comprender que:

- Los identificadores permiten registrar por un nombre auto-explicativo los datos con los que se trabajará a lo largo de la solución de un problema.
- El nombre de un identificador debe comenzar obligatoriamente por una letra y el resto de caracteres pueden ser: números. letras y el símbolo de guion bajo. Además, no pueden contener espacios, acentos, puntos, comas, diéresis, eñes, ni operadores (+, -, *, /, ^).
- Las variables son un tipo especial de identificador que se pueden utilizar para almacenar y hacer referencia a valores que pueden cambiar a medida que el Pseudocódigo digital avanza.
- Existen dos formas de crear una variable y/o asignarle un valor: la lectura y la asignación. Si se lee o asigna un valor en una variable que no existe, esta se crea. Si la variable ya existe, esta toma el nuevo valor, provocando la pérdida total de los valores almacenados con anterioridad.
- Los diversos tipos de datos que puede contener una variable son por lo general: números, letras, cadenas de caracteres y valores lógicos.
- Las constantes son un tipo especial de identificador que almacenan y hacen referencia a valores que no pueden ser alterados o modificados durante la ejecución de un Pseudocódigo digital, es decir, sólo pueden ser asignados una vez y leídos muchas veces. Por ejemplo, PI, el número e, el nombre de una empresa, entre otros.
- El nombre utilizado para identificar a una constante, por conveniencia, suele escribirse con letras mayúsculas en la mayoría de los lenguajes de programación.

1.2.2 Entrada y salida de información

Según Joel de la Cruz Villar argumenta que "los dispositivos de entrada / salida permiten que el usuario interactué con el sistema. Por medio de los dispositivos de entrada el usuario ingresa los datos a procesar en el

sistema y los dispositivos de salida muestran el resultado" (De la Cruz 24). Por ejemplo, en un Pseudocódigo digital para la conversión de unidades de medida. En un primer momento se requiere de la solicitud de algún tipo de información al usuario para poder operar. Posteriormente, enterado de la situación, proporcione dicha información al Pseudocódigo digital para su correspondiente procesamiento. Finalmente, como parte del proceso de solución al problema, se espera éste (Pseudocódigo digital) devuelva los resultados encontrados; tal y como se muestra en la siguiente imagen:

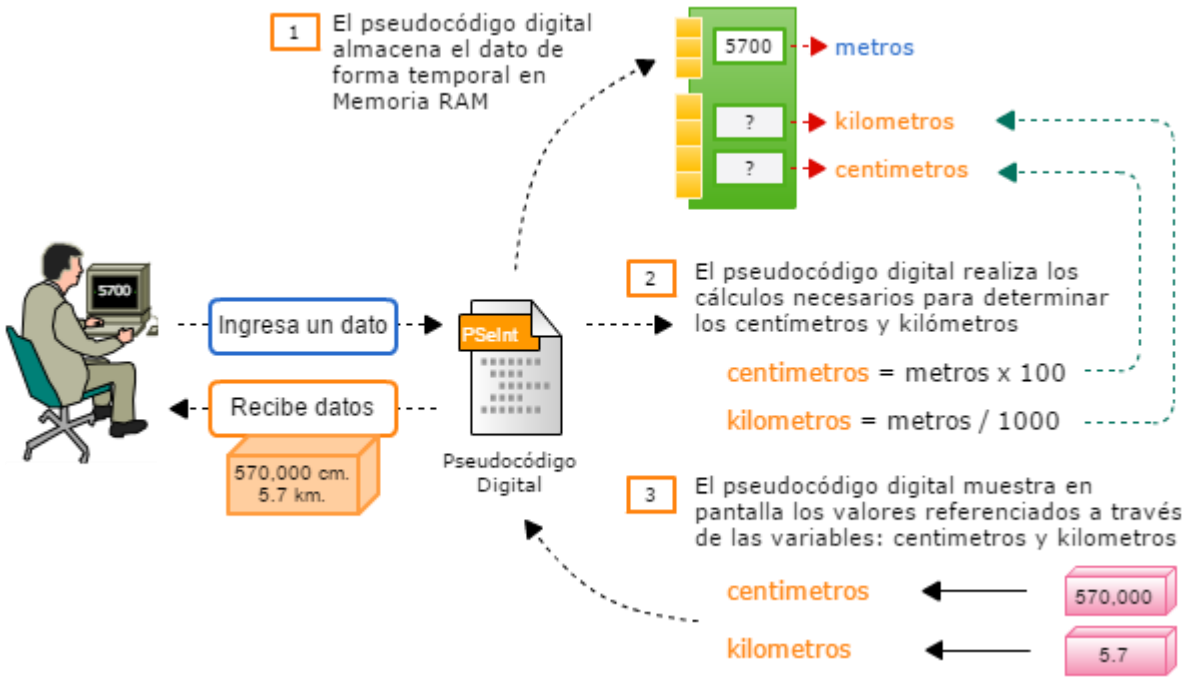


Imagen 16: Entrada, Proceso y Salida - Fuente: Diseñada por el autor.

De la imagen anterior, resulta evidente el uso de un "**teclado**" como dispositivo de entrada (de información) para ingresar la cantidad expresada en metros y, por otra parte, el uso de un "**monitor**" como dispositivo de salida para solicitar o mostrar una serie de informaciones relevantes al usuario, tales como los datos que necesita el Pseudocódigo digital para poder operar (metros), así como los resultados calculados de la conversión (centímetros y kilómetros).

Vinculado al concepto, De la Cruz añade que:

> Las instrucciones de [Entrada / Salida] dan acceso al programador a las funciones básicas de estos dispositivos, permitiéndole capturar datos de los dispositivos de entrada y asignarlos a variables para operar con ellos y mostrar resultados del proceso en los dispositivos de salida (De la Cruz 24).

Con base en la situación descrita, se puede decir que la declaración de instrucciones de "Entrada / Salida" juegan un papel importante para el programador, puesto que gracias a ellas, es posible escribir Pseudocódigos digitales cada vez más complejos que permiten establecer una comunicación directa con los usuarios a través del uso de diferentes dispositivos conectados al ordenador (**teclado** / **monitor**); y por consiguiente, durante el proceso de solución, sea posible ofrecer una serie de resultados "cambiantes" de acuerdo a su solicitud.

1.2.2.1 Entrada



Imagen 17: Teclado PC - Fuente: https://es.wikipedia.org/wiki/Teclado_(informática)

Como se ha mencionado con anterioridad, las instrucciones de entrada se utilizan para capturar los datos que provienen desde el exterior, de modo que sea posible asignarlos en variables que permitan su posterior localización dentro del Pseudocódigo digital.

Bajo este mismo orden de ideas, la documentación oficial de PSeInt establece que "la instrucción **Leer** permite ingresar información desde el ambiente" (PSeInt 1). Es decir, cada vez que el Pseudocódigo digital

necesite de la intervención del usuario para el ingreso de algún tipo de dato o información que se requiera durante el proceso, es necesario hacer uso de la palabra reservada **Leer** para proceder a capturar los datos que provienen desde el teclado del ordenador. Sin embargo, para que estos datos puedan prevalecer y ser de utilidad, se necesita de un identificador de tipo variable que permita identificarlos y referenciarlos dentro del Pseudocódigo digital, tal y como se muestra en la siguiente imagen:

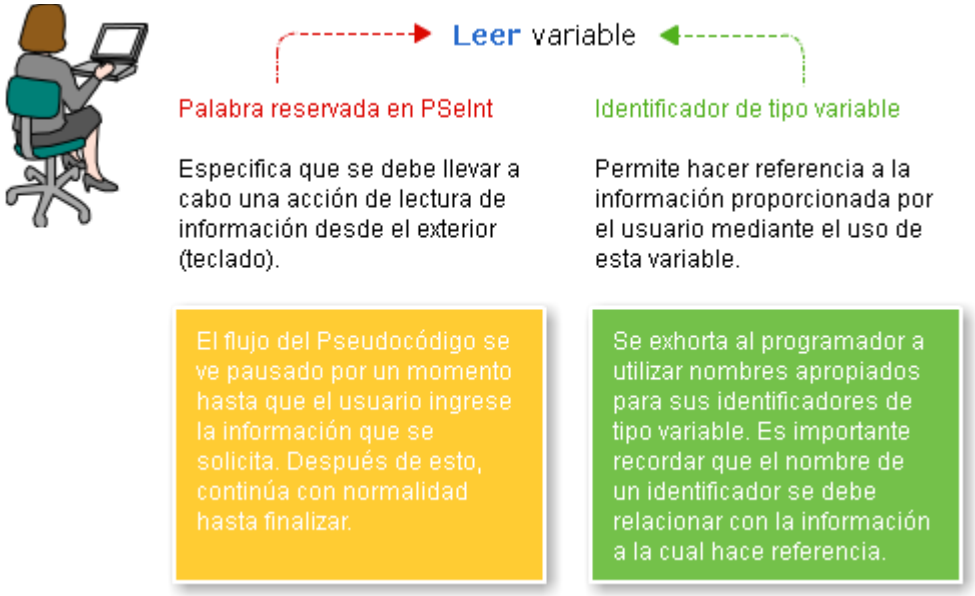


Imagen 18: Estructura instrucción Leer - Fuente: Diseñada por el autor.

En este sentido, con la finalidad de reforzar algunos de los ejemplos abordados hasta el momento, la siguiente tabla muestra el proceso de lectura que se debe llevar a cabo dentro de cada Pseudocódigo digital, para que los datos proporcionados por el usuario sean considerados durante el proceso de solución del problema:

Fases	**Nombre del Pseudocódigo digital**		
	Suma de dos números	Hipotenusa de un triángulo rectángulo	Convertidor de unidades de medida, metros a centímetros y kilómetros

Entrada	**Leer** numA **Leer** numB	**Leer** base **Leer** altura	**Leer** metros

Tabla 3: Ejemplos de lectura de información - Fuente: Elaborada por el autor.

Con base en los supuestos anteriores, es importante mencionar que el proceso de lectura de información es considerado por PSeInt como una "operación destructiva", debido a que la información proporcionada por el usuario es referenciada internamente mediante el uso de variables, las cuales tras el proceso de análisis (lectura), son creadas automáticamente por PSeInt para hacer referencia al contenido almacenado en la Memoria RAM. Sin embargo, si por algún motivo dichas variables ya existen previamente en el Pseudocódigo digital, la referencia hacia sus contenidos es reemplazada por la nueva información (véase la imagen 19), es decir, tiene a lugar un proceso de sobre-escritura de información del cual ya se ha hablado previamente en esta investigación (véase el apartado correspondiente a las variables).

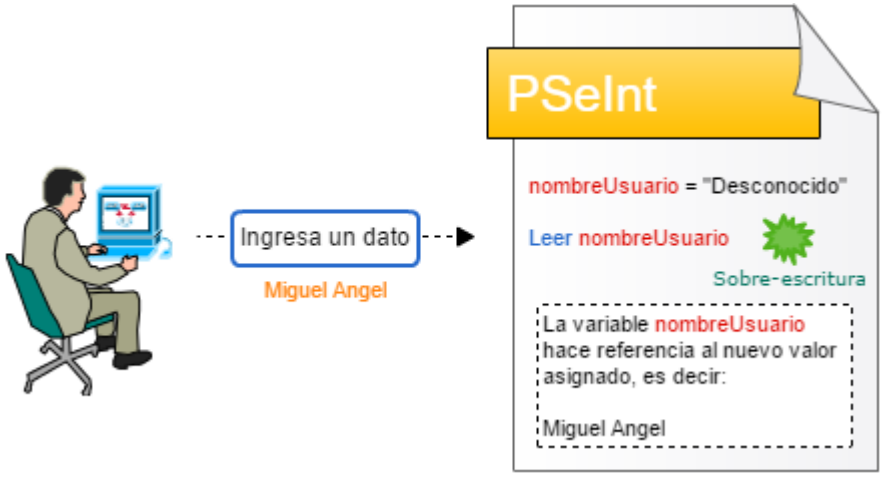


Imagen 19: Operación destructiva en el proceso de lectura de información - Fuente: Diseñada por el autor.



¿Por qué utilizar instrucciones de entrada?

Las instrucciones de entrada se utilizan para capturar los datos que provienen desde el exterior (por ejemplo, un teclado), de modo que sea posible asignarlos en variables que permitan su posterior localización dentro del Pseudocódigo digital.

PSeInt utiliza la palabra reservada **Leer** para permitir el ingreso de información desde el teclado del ordenador.

Una vez que se ha abordado todo lo referente a la lectura de información, es momento de continuar con las instrucciones que permiten al programador establecer comunicación con los dispositivos de salida conectados al ordenador (**monitor**), con la finalidad de brindar información a los usuarios acerca del proceso de solución del problema.

1.2.2.2 Salida



Imagen 20: Monitor - Fuente: https://es.wikipedia.org/wiki/Monitor_de_computadora

Según Miguel Angel Rodríguez Almeida, las instrucciones de salida "sirven para presentar en pantalla […] comentarios, constantes, contenido de las variables y resultado de [algunas] expresiones" (Rodríguez 14). En este sentido, resulta incuestionable utilizar este tipo de instrucciones para informar al usuario acerca de los datos o información que el Pseudocódigo digital necesita para poder operar, mostrar el contenido referenciado por las variables y constantes utilizadas internamente a lo largo de su ejecución, así como exhibir los resultados obtenidos tras el procesamiento de la información (evaluación de las expresiones).

En atención a los supuestos anteriores, la documentación de PSeInt argumenta que "la instrucción **Escribir** permite mostrar valores al ambiente" (PSeInt 1). Dicho de otra forma, la palabra reservada **Escribir** permite reportar al usuario los resultados correspondientes de un proceso, así como él envió de notificaciones donde se requiera de su intervención para la entrada de información. Bajo este mismo orden de ideas, la siguiente imagen muestra la estructura de la instrucción **Escribir** bajo tres escenarios; el primero de ellos muestra un mensaje de bienvenida al usuario, el segundo envía una notificación al usuario para el ingreso de información, y el tercero expone el contenido referenciado por una variable:

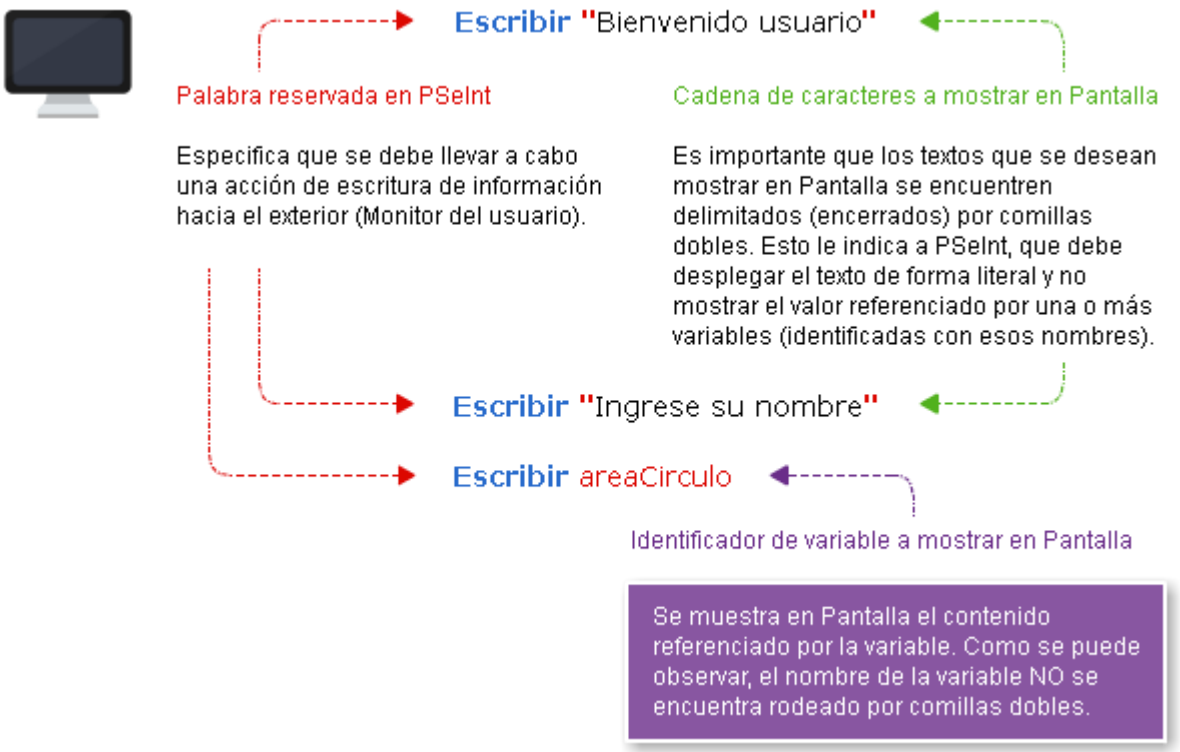


Imagen 21: Estructura instrucción Escribir - Fuente: Diseñada por el autor.

Con base en la situación descrita con anterioridad, es importante mencionar que, al momento de redactar estas líneas de la investigación,

PSeInt ofrece otro conjunto de instrucciones alternativas para presentar información en pantalla: **Imprimir** y **Mostrar**. En este sentido, y con la firme intención de estandarizar las salidas de información correspondientes a cada uno de los ejemplos tratados a lo largo de este capítulo de la investigación, se utilizará la palabra reservada **Imprimir** para tal propósito, debido a que por criterio personal del autor (y algunos lenguajes de programación como PHP, Python, etc.) dicha instrucción tiende a generar un mayor contexto en el programador para tareas relacionadas con la salida de información a través de la pantalla de un ordenador.

Ahora bien, con el fin de puntualizar en cada uno de los aspectos tratados hasta el momento, la siguiente imagen ilustra detalladamente el proceso de desarrollar un Pseudocódigo digital en PSeInt, cuyo objetivo consiste en determinar el área de un triángulo rectángulo a partir de ciertos datos proporcionados por el usuario, como son: su base y altura.

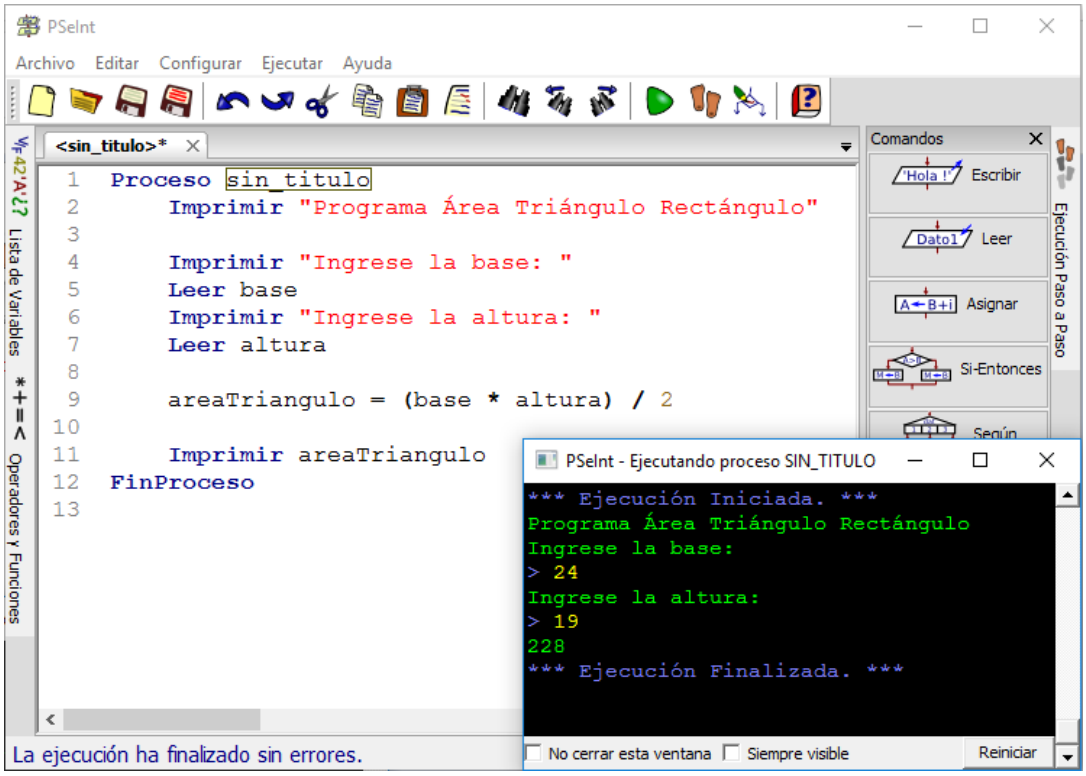


Imagen 22: Pseudocódigo digital para determinar el área de un triángulo rectángulo - Fuente: Diseñada por el autor.

Como se puede observar, las líneas 1 y 12 marcan el inicio y final del Pseudocódigo digital identificado como **sin_titulo** en PSeInt. La línea 2 imprime por pantalla un mensaje informativo al usuario que muestra el nombre del programa y el objetivo que persigue. Las líneas 4 y 6 notifican al usuario acerca de los datos que necesita el Pseudocódigo digital para poder operar, es decir, la base y altura. Las líneas 5 y 7 se encargan de leer la información que proporciona el usuario por teclado para almacenarla en Memoria RAM del ordenador y hacer referencia a la misma dentro del Pseudocódigo digital mediante las variables **base** y **altura**. La línea 9 es la instrucción encargada de resolver la problemática en cuestión, es decir, la fórmula que obtiene el área del triángulo rectángulo a partir de los datos proporcionados previamente por el usuario. En este caso se trata de una asignación que hace referencia a el resultado de operar **(base * altura) / 2**. Finalmente, la línea 11 muestra en pantalla el valor

referenciado por la variable **areaTriangulo**, mismo que representa la solución al problema.

Al ejecutar el ejemplo anterior con la ayuda de la tecla de función **F9**[10], PSeInt abre una nueva ventana (véase la imagen 22) donde se muestra la información de salida declarada en el Pseudocódigo digital (mediante la palabra reservada **Imprimir**), así como las pautas (corchete angular **>**) para el ingreso de información mediante el uso de teclado (instrucción **Leer**), que tras confirmar con la tecla **Enter**, permiten que el flujo de las instrucciones declaradas continúen hasta finalizar su cometido (mostrar el resultado final al usuario).



¿Sabía qué?

En versiones recientes del Pseudo-Intérprete PSeInt es posible utilizar las palabras clave **Algoritmo** y **FinAlgoritmo** como sinónimos alternativos a las instrucciones **Proceso** y **FinProceso.**

Llegado a este punto, es normal que existan dudas con respecto a la línea de código número 9 del ejemplo anterior. Sin embargo, esta situación no debería ser motivo de preocupación para el lector, puesto que, más adelante dentro de este mismo capítulo de la investigación, se aborda todo lo relacionado a los operadores aritméticos y su declaración conjunta para la construcción de fórmulas matemáticas complejas.

[10] Las funciones asignadas a cada tecla especial de su ordenador, depende directamente de la configuración de su teclado y del sistema operativo que actualmente tenga instalado. Para mayor información, sírvase en leer la documentación que acompaña a su equipo, o presione el icono **Ejecutar…** que aparece junto a la barra de herramientas de PSeInt.



¿Por qué utilizar instrucciones de salida?

Las instrucciones de salida se utilizan para presentar en pantalla: solicitudes de información al usuario, mensajes de notificación, el contenido de las variables y constantes, así como el resultado de evaluar ciertas expresiones.

PSeInt utiliza la palabra reservada **Imprimir** para mostrar información a través de la pantalla del ordenador.

Una vez que el lector cuenta con las bases suficientes para trabajar con entradas y salidas de información, toca el turno a los tipos de datos soportados por PSeInt, ya que, en virtud de la problemática a resolver, los Pseudocódigos digitales necesitan una variedad de datos para poder trabajar, los cuales, en la mayoría de las ocasiones vienen representados en formato numérico (entero y/o decimal), carácter, conjunto de caracteres, lógicos, entre otros.



Resumen: Entrada y salida de información.

En esta sección se abordaron las entradas y salidas de información. A estas alturas, se debería comprender que:

- Los dispositivos de entrada y salida permiten que el usuario interactúe con los programas instalados en una computadora, a través de ellos es posible ingresar los datos a procesar en el programa y, por consiguiente, observar los resultados generados.
- Las instrucciones de entrada/salida dan acceso al programador a las funciones básicas de estos dispositivos, permitiéndole capturar los datos que provienen desde los dispositivos de entrada y asignarlos en variables para poder operar con ellos y mostrar los resultados del proceso en los dispositivos de salida.
- PSeInt utiliza la palabra reservada **Leer** para permitir la entrada de información desde el exterior (por ejemplo, un teclado), y el término **Imprimir,** para todo lo concerniente a las salidas de información (por ejemplo, a través de un monitor).

En conclusion, para que un usuario pueda establecer comunicación directa con un Pseudocódigo digital, es necesario que el programador internamente utilice instrucciones de entrada/salida para la solicitud, ingreso y devolución de información. De ser así, el Pseudocódigo digital en cuanto a utilidad se convierte en una herramienta más versátil para el usuario, puesto que atendería a solicitudes provenientes desde el exterior y, por consiguiente, los resultados generados estarían en relación con la satisfacción de dicha solicitud. Sin instrucciones de entrada/salida, es posible que los Pseudocódigos digitales funcionen correctamente, sin embargo, es importante recordar que, durante la elaboración de un Algoritmo, éste debe describir tres partes fundamentales: las entradas, el proceso y las salidas.

1.2.3 Tipos de datos

Los tipos de datos pueden ser considerados como el tipo de información que admite o reconoce un Pseudocódigo digital para su correspondiente procesamiento. Como ya se ha mencionado con anterioridad, los datos con los que trabaja un Pseudocódigo digital son almacenados temporalmente

en Memoria RAM del ordenador, y posteriormente, se hace referencia a cada uno de ellos mediante el uso de una serie de identificadores, los cuales pueden ser de tipo variable o constante.

Vinculado al concepto, la documentación oficial de PSeInt establece que los tipos de datos simples admitidos para el desarrollo de Pseudocódigos digitales pueden ser clasificados en tres grupos básicos:

- **Numérico:** números, tanto enteros como reales. Para separar decimales se utiliza el punto. Ejemplos: 12 23 0 -2.3 3.14
- **Lógico:** solo puede tomar dos valores: **VERDADERO** o **FALSO**.
- **Carácter:** caracteres o cadena de caracteres encerrados entre comillas (pueden ser dobles o simples). Ejemplos ‘hola’ “hola mundo” ‘123’ ‘FALSO’ ‘etc’ (PSeInt 1).

Ahora bien, para almacenar grandes volúmenes de información dentro de un Pseudocódigo digital. PSeInt cuenta con otro tipo de dato conocido como **Arreglo**, mismo que puede ser considerado como una estructura de datos homogénea (todos sus datos son del mismo tipo) que permite almacenar un determinado número de datos bajo un mismo identificador. Es decir, a diferencia de los tipos de datos simples, los Arreglos pueden almacenar más de un valor del mismo tipo. Sin embargo, para acceder a cada uno de sus contenidos, se debe hacer a través de la posición que ocupan dentro del mismo (véase un ejemplo más adelante dentro de este mismo numeral).

Con base en los supuestos anteriores, la siguiente imagen ilustra una serie de ejemplos prácticos que hacen uso de datos de tipo numérico, carácter, y cadena de caracteres. Así como su proceso de asignación y referencia mediante el uso de variables declaradas a través de PSeInt.

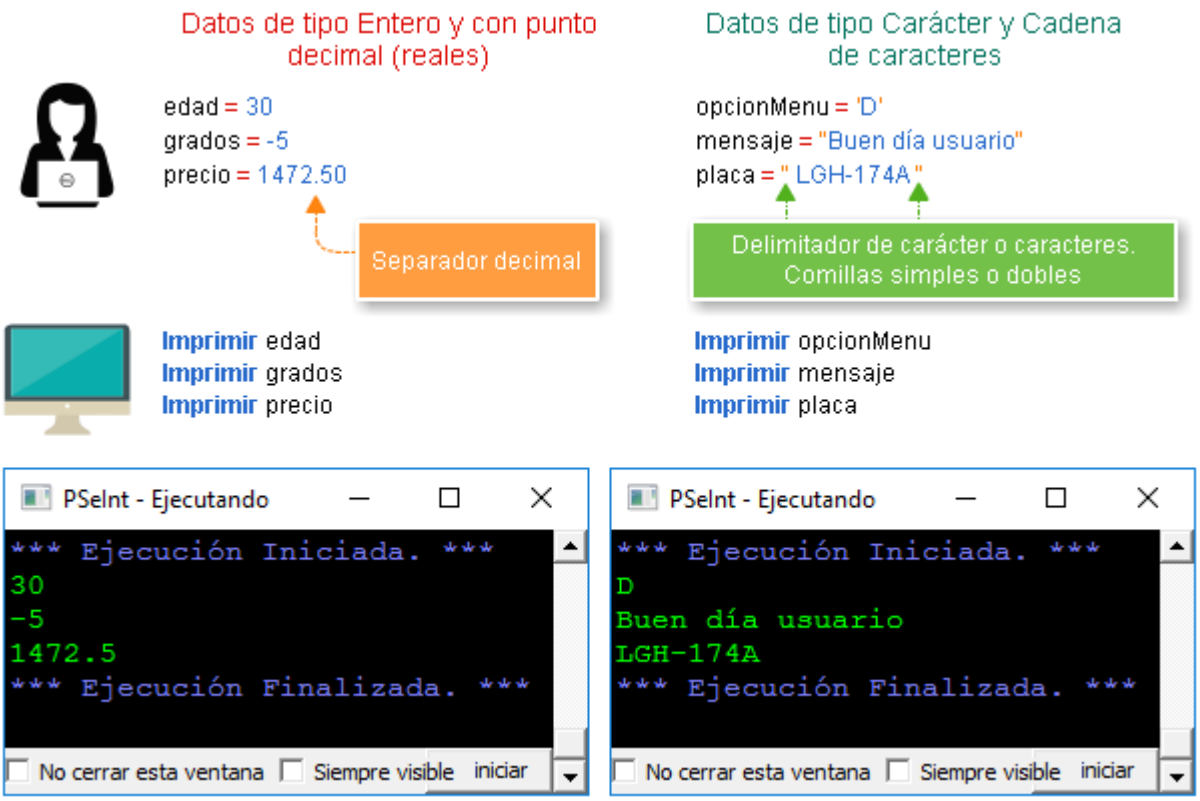


Imagen 23: Tipos de dato Numérico y Carácter - Fuente: Diseñada por el autor.

Como se puede observar en el ejemplo del lado izquierdo, los valores de tipo numérico asignados y referenciados a través de una variable pueden estar expresados en formato entero o decimal, su valor asociado puede estar dentro del rango de los números positivos como negativos, y el separador de cifras decimales (números reales) debe estar expresado mediante el punto decimal. No obstante, es importante mencionar que, al momento de redactar estas líneas, PSeInt no cuenta con un símbolo que cumpla con la función de separador de miles.

Ahora bien, con respecto a los datos de tipo carácter y cadena de caracteres es importante que, durante su asignación a una variable estos se encuentren delimitados mediante el uso de comillas simples o dobles (evítese mezclarlas). De lo contrario, en proporción al número de palabras

que formen la cadena de texto “mal formateada”, PSeInt tratará de llevar a cabo una de las siguientes acciones:

1. Una palabra o carácter: PSeInt considera que se trata de una asignación de variables, es decir, la palabra o carácter es considerado como una variable más en el Pseudocódigo digital, cuyo valor se le asigna a la otra variable.
2. Dos o más palabras: PSeInt considera que se trata de un conjunto de variables que se deben evaluar previamente como parte de una expresión matemática.

Bajo este mismo orden de ideas, la siguiente figura muestra una serie de errores clásicos que comúnmente los programadores principiantes cometen al trabajar con datos de tipo carácter o cadena de caracteres.

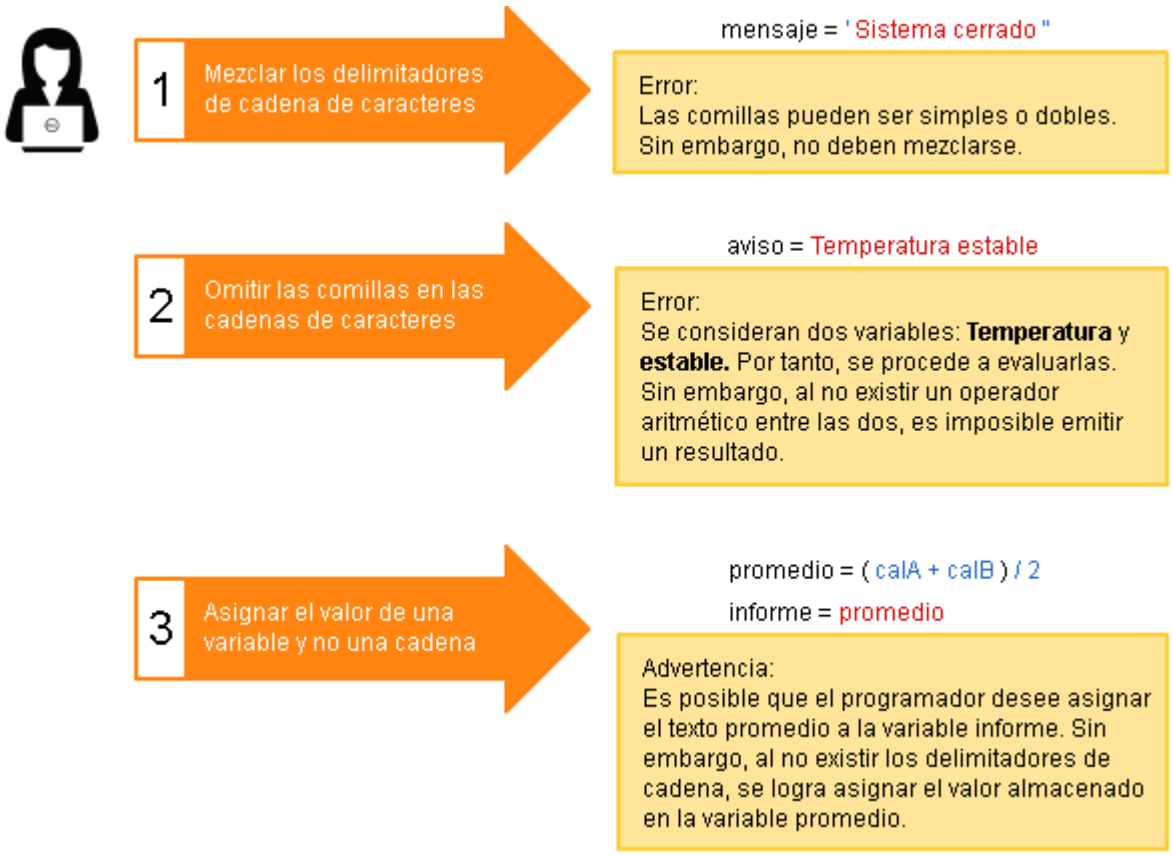


Imagen 24: Errores clásicos al trabajar con datos de tipo carácter y cadena de caracteres - Fuente: Diseñada por el autor.

Durante los primeros pasos como programador, es probable que la mayoría de los Pseudocódigos digitales que se desarrollen sólo hagan uso de datos de tipo numérico, carácter, o cadena de caracteres. Sin embargo, a medida que se avanza en la formación como programador profesional, es común que los Pseudocódigos digitales tengan que hacer algún tipo de comprobación adicional o almacenar grandes volúmenes de información mediante estructuras de datos complejas. En este sentido, se hace imprescindible conocer otro tipo de datos (véase la imagen 25) para satisfacer dichas necesidades.

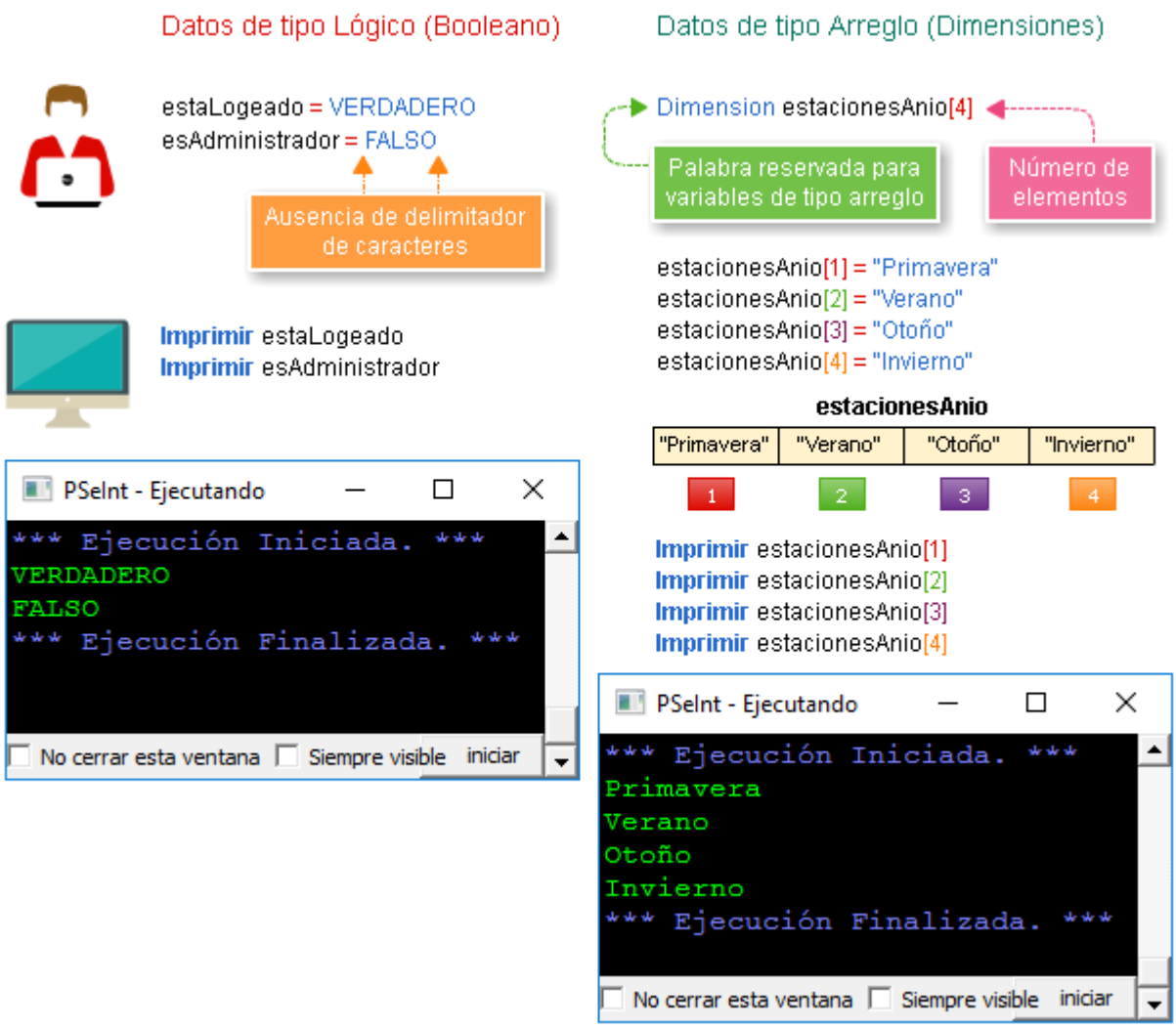


Imagen 25: Tipos de dato Lógico y Arreglo - Fuente: Diseñada por el autor.

No debe existir preocupación por el momento si no queda del todo claro cada uno de los ejemplos citados en la imagen anterior. Por ahora es importante que se asimile que un tipo de dato lógico sólo puede contener uno de dos valores posibles, **FALSO** o **VERDADERO**, y que su aplicación dentro de un Pseudocódigo digital está relacionada con la validación o comprobación de ciertas acciones. Por ejemplo: si el usuario ha iniciado sesión correctamente, si la calificación reportada por un alumno es suficiente para acreditar una materia, si el total de la compra realizada por un cliente conlleva a otorgarle un cierto descuento sobre la misma, etc.

Por otra parte, los arreglos "… se caracterizan por el hecho de que con un nombre (identificador de variable estructurada) se hace referencia a un grupo de casillas en memoria …" (Cairó 8). Es decir, en ocasiones es probable que dentro de un Pseudocódigo digital se tenga la necesidad de trabajar con una gran cantidad de datos que guardan cierta relación entre sí (como los días de la semana, los meses o las estaciones del año). En lugar de utilizar una variable distinta para poder asignarlos y referenciarlos (lo que acarrea una labor tediosa), es mejor agruparlos dentro de un mismo conjunto identificado bajo un nombre común que pueda ser tratado como una sola entidad (variable).

En el ejemplo de la imagen anterior, se tiene la necesidad de almacenar un conjunto de datos relacionados con las estaciones climatológicas del año. Puesto que un arreglo permite agrupar e identificar un conjunto de valores relacionados entre sí, se procede a reservar un espacio en Memoria RAM mediante la declaración de un identificador de variable "**estacionesAnio**" acompañado de un número entre corchetes (**[4]**) que representa los espacios necesarios para la referencia o asignación de información dentro de la estructura (dimensión del arreglo).

Frente a esta situación, para poder asignar y recuperar cada uno de los datos dentro del arreglo, se hace uso del identificador de variable **estacionesAnio** seguido de la posición (**[1]**, **[2]**, **[3]**, **[4]**,…, [N]) que guarda el elemento dentro del arreglo.

Llegados a este punto, es importante mencionar que dada la naturaleza de los datos de tipo arreglo y la posibilidad de utilizarlos para hacer referencia a grandes conjuntos de datos del mismo tipo, éstos no forman parte del objetivo que persigue este trabajo de investigación; ya que, desde la perspectiva del propio autor, su uso se orienta más al manejo y

administración de estructuras de datos complejas, lo que da como resultado el que se le dedique una nueva línea de investigación.

Una vez que se han abordado los diferentes tipos de datos soportados por PSeInt y su declaración a nivel de Pseudocódigo digital, es momento de entrar en detalle con el procesamiento de cada uno de ellos mediante el uso de distintos operadores, tales como: los aritméticos, los de tipo lógico, y los de comparación.



Resumen: Tipos de datos.

En esta sección se abordaron los diferentes tipos de datos que se pueden gestionar durante el proceso de solución de un problema mediante el uso de un ordenador.

Al igual que en la vida real, un Pseudocódigo digital se basa en diferentes tipos de información para conseguir que cada una de sus instrucciones se ejecute adecuadamente. Por ejemplo, para calcular el total de una compra, se necesita conocer previamente el precio y la cantidad de los artículos adquiridos (información numérica). Para personalizar el ticket de dicha compra con el nombre del cliente, se necesita conocer anticipadamente ese nombre (información de tipo carácter). Asimismo, para saber si es necesario aplicar un descuento sobre la compra, es necesario que se verifique previamente si el monto total cumple con una determinada condición (información de tipo lógica).

Los lenguajes de programación reales normalmente clasifican la información en diferentes tipos y tratan a cada uno de ellos de un modo distinto. En PSeInt los cuatro tipos de datos soportados hasta el momento son los **numéricos**, **carácter**, **lógicos** y **arreglos**. Al definir la naturaleza de los datos (PSeInt lo hace de forma automática para los datos de entrada), se favorece a su interpretación y manipulación dentro del Pseudocódigo digital, ya que, de no hacerlo, es posible que se presenten errores y confusiones al momento de su procesamiento. Por ejemplo:

Caso 1	Caso 2	Caso 3
numA = "30" numB = 18	numA = "30" numB = "18"	numA = 30 numB = 18
suma = numA + numB	suma = numA + numB	suma = numA + numB

- En el caso 1 se trata de sumar una cadena de caracteres y un número, el resultado de la operación es un error, puesto que no es posible operar datos de diferente tipo.
- En el caso 2 se trata de sumar dos cadenas de caracteres, el resultado de la operación es una nueva cadena con el valor "3018", es decir, al sumar dos cadenas de caracteres estas se unen.
- En el caso 3 se trata de sumar dos valores de tipo numérico, el resultado de la operación es el valor 48.

1.2.4 Operadores

Los operadores pueden describirse como un conjunto de símbolos que denotan una serie de operaciones que deben realizarse sobre una determinada cantidad de operandos. En el campo de la informática, existen diversos operadores "matemáticos" que pueden utilizarse para el desarrollo de los Pseudocódigos digitales, tales como: los operadores aritméticos, los operadores lógicos y los operadores de comparación o relación.

1.2.4.1 Operadores Aritméticos

Según Joel de la Cruz Villar "los operadores aritméticos permiten realizar cualquier operación aritmética (suma, resta, multiplicación y división) [entre operandos]" (De la Cruz 22). El resultado de una operación aritmética puede ser un valor numérico del tipo entero o decimal, y puede estar dentro del rango de los números positivos o negativos.

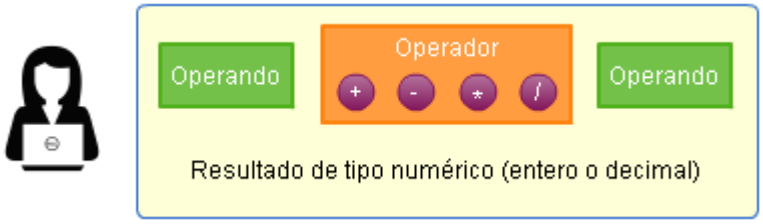


Imagen 26: Esquema que representa una fórmula construida a partir de operadores aritméticos - Fuente: Diseñada por el autor.

En la siguiente tabla se muestran los diferentes operadores aritméticos básicos soportados por PSeInt al momento de redactar estas líneas de la investigación.

Operador	Significado	Ejemplo	Resultado
+ (signo más)	Suma	12.5 + 144.2	156.7
- (signo menos)	Resta	489 - 1253	-764
* (asterisco)	Multiplicación	1200 * 11	13200
/ (barra inclinada)	División	153 / 9	17
^ (acento circunflejo)	Potencia o exponenciación	5 ^ 3	125
% (signo de porcentaje)	Módulo (residuo o sobrante de la división entera)	81 % 5	1

Tabla 4: Ejemplo Operadores Aritméticos - Fuente: Elaborada por el autor.

Sumado a lo expuesto, es importante mencionar que, de acuerdo con la problemática a resolver mediante el desarrollo de un Pseudocódigo digital, es normal que se tenga que recurrir a la combinación de dos o más operadores aritméticos como parte del proceso de solución al problema (una tarea común y frecuente dentro de cualquier lenguaje de programación). Por ejemplo, para obtener el promedio general de dos calificaciones parciales de un alumno (véase la imagen 27), se requiere en primer lugar sumar las dos calificaciones y posteriormente, dividir el resultado entre el número de ellas, es decir, entre 2.

```
(primerParcial + segundoParcial) / 2
```

Imagen 27: Expresión aritmética para obtener el promedio de dos calificaciones parciales - Fuente: Diseñada por el autor.

Como se puede observar, en una expresión matemática (fórmula) pueden intervenir una cantidad diferente de elementos, tales como: números, constantes, variables, símbolos de operación, paréntesis y nombres de funciones especiales (más adelante en este mismo capítulo). Además, la operación conjunta de cada uno de ellos genera como resultado un nuevo valor numérico que puede ser mostrado (en pantalla del usuario) o

asignado a otra variable dentro del Pseudocódigo digital. Tal y como se muestra en la siguiente imagen:

```
promedioGeneral = (primerParcial + segundoParcial) / 2

Imprimir (primerParcial + segundoParcial) / 2
```

Imagen 28: Expresión aritmética asignada a una variable e impresa directamente en pantalla del ordenador - Fuente: Diseñada por el autor.

Ahora bien, con la finalidad de reforzar cada uno de los supuestos anteriores, la siguiente figura ilustra el proceso de desarrollo de un Pseudocódigo digital[11] que determina el resultado de seis operaciones aritméticas (suma, resta, multiplicación, división, exponenciación y módulo) a partir de dos datos proporcionados por el usuario.

[11] La primera línea de código hace referencia al nombre del proceso en turno "**calculadora_basica**". Puesto que el Pseudocódigo digital requiere de dos números proporcionados por el usuario para poder operar, las líneas de código 3 a 6 se encargan del proceso de entrada de información, de este modo, los valores ingresados por teclado quedan referenciados en el código a través de las variables **numA** y **numB**. Las líneas 8 a 13 son las responsables de llevar a cabo el proceso de solución al problema, es decir, contienen las expresiones matemáticas necesarias (suma, resta, multiplicación, división, exponenciación y residuo) para operar los datos proporcionados por el usuario y asignar los resultados en seis nuevas variables para su posterior impresión por pantalla (líneas 15 a 20). Finalmente, la línea de código 22 representa el final del proceso **calculadora_basica**.

```
1  Proceso calculadora_basica
2
3      Imprimir "Ingrese primer número"
4      Leer numA
5      Imprimir "Ingrese segundo número"
6      Leer numB
7
8      suma = numA + numB
9      resta = numA - numB
10     multiplicacion = numA * numB
11     division = numA / numB
12     exponenciacion = numA ^ numB
13     residuo = numA % numB
14
15     Imprimir suma
16     Imprimir resta
17     Imprimir multiplicacion
18     Imprimir division
19     Imprimir exponenciacion
20     Imprimir residuo
21
22 FinProceso
```

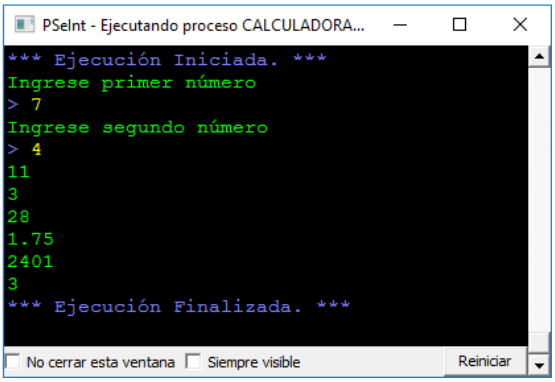


Imagen 29: Pseudocódigo digital que determina seis operaciones aritméticas básicas - Fuente: Diseñada por el autor.

Llegado a este punto de la investigación, seguramente se habrá observado que los Pseudocódigos digitales desarrollados hasta el momento emiten sus resultados sin un mensaje que oriente al usuario acerca de la función que desempeñan dentro del mismo. En este sentido, una posible mejora progresiva, consiste en utilizar el signo coma (**,**) para añadir un mensaje descriptivo a cada resultado generado; tal y como se ilustra en la siguiente captura de pantalla:

```
1  Proceso programa_circulo
2      Imprimir "****************************************"
3      Imprimir "*   Bienvenido al Programa Círculo   *"
4      Imprimir "****************************************"
5      Imprimir ""
6
7      Imprimir "Ingrese radio en centímetros"
8      Leer radio
9
10     areaCirculo = PI * (radio ^ 2)
11
12     Imprimir "El área es: " , areaCirculo , "cm. cuadrados"
13 FinProceso
14
```

```
*** Ejecución Iniciada. ***
****************************************
*   Bienvenido al Programa Círculo   *
****************************************

Ingrese radio en centímetros
> 5
El área es: 78.5398163397cm. cuadrados
*** Ejecución Finalizada. ***
```

Imagen 30: Salida de información debidamente formateada - Fuente: Diseñada por el autor.

Como se puede observar, se trata de un Pseudocódigo digital que determina el área de un círculo a partir de su correspondiente radio, mismo que en la línea de código número 12 imprime (en pantalla del ordenador) una cadena de texto formada a partir de la concatenación (unión) de tres sub-cadenas. La primera de ellas con el contenido **"El área es: "**, seguida del valor referenciado por la variable **areaCirculo**, y finalmente, una tercera cadena cuyo contenido hace referencia al texto **"cm. cuadrados"**.

Sumado a lo expuesto, es probable que algunos lectores con cierta experiencia en programación asocien el comportamiento del signo coma (**,**) de PSeInt como el de un operador de texto (presente en la mayoría de los lenguajes de programación). Sin embargo, es importante enfatizar que su uso dentro de este Pseudo-Intérprete queda limitado a la instrucción **Imprimir** para concatenar diferente tipo de contenido sobre una misma

línea de salida, es decir, no se puede utilizar como parte de una expresión para generar un nuevo resultado.

Ejemplo PSeInt	Resultado
Imprimir "Hola estimado " , nombreUsuario	Correcto
resultado = "Hola estimado " , nombreUsuario	Incorrecto (error)

Tabla 5: Declaración correcta e incorrecta del operador de concatenación en PSeInt - Fuente: Elaborada por el autor.



¿Por qué utilizar operadores aritméticos?

Para ejecutar las operaciones matemáticas como suma, resta, multiplicación, división, exponenciación y módulo, es necesario utilizar los operadores aritméticos.

- Los valores implicados en el cálculo de una operación aritmética pueden venir representados por variables, constantes, llamadas a funciones especiales y números literales.
- El resultado de una operación aritmética puede ser un valor numérico de tipo entero o decimal y puede estar dentro del rango de los números positivos o negativos.

1.2.4.2 Operadores de Comparación

Según Juan Carlos Casale en su obra Introducción a la Programación, argumenta que los operadores de comparación "se utilizan para establecer una relación entre dos valores …" (Casale 86). Es decir, su finalidad es comparar dos operandos y generar como resultado un valor lógico: **VERDADERO** o **FALSO**.

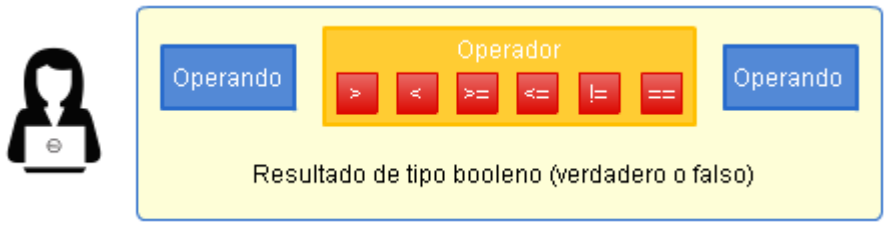


Imagen 31: Esquema que representa la construcción de una expresión a partir de operadores de comparación - Fuente: Diseñada por el autor.

Los operadores de comparación trabajan sobre valores del mismo tipo y su aplicación dentro de un Pseudocódigo digital está directamente relacionada con la toma de decisiones que se debe llevar a cabo durante la fase de proceso.

Bajo este mismo orden de ideas, la siguiente tabla tiene por objetivo ilustrar los diferentes operadores de comparación soportados por PSeInt al momento de redactar estas líneas de la investigación.

Operador	Significado	Ejemplo	Resultado
> (signo mayor que)	Mayor que	15.9 > 11.99	FALSO
< (signo menor que)	Menor que	125 < 1340	VERDADERO
>= (signo mayor o igual que)	Mayor o igual que	19 >= 14	VERDADERO
<= (signo menor o igual que)	Menor o igual que	30 <= 30	VERDADERO
!= (signo distinto de)	Distinto de	100 != 100	FALSO
== (signo idéntico a)	Idéntico a (Igualdad)	1985 == 1985	VERDADERO

Tabla 6: Ejemplo Operadores de Comparación - Fuente: Elaborada por el autor.

Al igual que los operadores aritméticos, los valores que intervienen en una comparación pueden estar representados a través de números, constantes o variables. Sin embargo, también es posible comparar valores alfanuméricos (caracteres), pero, siempre y cuando se tome como base los siguientes criterios:

> Cuando se comparan caracteres alfanuméricos, se hace uno a uno, de izquierda a derecha. Si las variables son de diferente longitud, pero exactamente iguales hasta el último carácter del más corto, entonces se considera que el más corto es el menor. Sólo son iguales dos datos alfanuméricos si son iguales su longitud y sus componentes. […] Las letras minúsculas tienen mayor valor que las mayúsculas. (Rodríguez 4)

Con la finalidad de ejemplificar el aporte que hace Rodríguez sobre la comparación caracteres, la siguiente tabla ilustra el comportamiento que se obtiene tras comparar diferentes cadenas de caracteres mediante la ayuda de los operadores de comparación disponibles en PSeInt.

Comparación	Resultado
"ACEPTAR" == "aceptar"	FALSO
"Cancelar" != "Abortar"	VERDADERO
"alejandro" > "ALEJANDRO"	VERDADERO
"XYZ" < "XYZZZZ"	VERDADERO
"a" > "z"	FALSO

Tabla 7: Ejemplo comparación cadena de caracteres - Fuente: Elaborada por el autor.

Como se puede observar, en la primera comparación se trata de verificar si las cadenas (ACEPTAR con mayúscula y aceptar con minúsculas) son idénticas, sin embargo, debido a que el operador de identidad exige que los operandos sean exactamente iguales (y en el caso de caracteres, escritos con el mismo molde de letra) se origina un resultado **FALSO**. En la segunda comparación, al tratarse de caracteres cuyo contenido es diferente uno del otro, se genera un resultado **VERDADERO** al utilizar el operador diferente de. Por consiguiente, en la tercera comparación se evidencia claramente que las letras minúsculas tienen mayor peso que las

mayúsculas, al tratar de comparar caracteres con el mismo mensaje, pero escritos con diferente molde de letra. Ahora bien, en la comparación que ocupa el lugar número cuatro, se observa que los primeros tres caracteres de ambas cadenas son idénticos, sin embargo, la cadena ubicada al lado derecho del operador de comparación (menor que) es más extensa que la otra, y genera un resultado **VERDADERO**. Finalmente, en la comparación número cinco se demuestra que el orden de comparación que se sigue para los caracteres (letras con el mismo molde de letra) está determinado por su ubicación en el esquema del abecedario, para el caso del ejemplo citado, resulta **FALSO** que el carácter "a" sea mayor que el carácter "z", debido a que, por orden, "z" es mayor que sus antecesores "a, b, c…".

Si bien hasta el momento puede parecer engorroso cada uno de los criterios y procedimientos que llevan a cabo los lenguajes de programación para la comparación de caracteres, lo cierto es que, para la mayoría de los desarrollos, sólo se utilizan los operadores de comparación idéntico a (==) y diferente de (!=). Por ejemplo, en un programa de carácter administrativo es normal que los usuarios se autentifiquen mediante una contraseña de apertura. En este sentido, resulta evidente hacer una comparación del tipo "idéntico a" para verificar que los datos de acceso coinciden con los datos previamente registrados. Ahora bien, si por algún motivo se presenta un error de conexión, es decir, que el estado del programa fuese "diferente de" conectado. Lo pertinente sería informar al usuario de lo sucedido, ya sea mediante un mensaje de notificación, o a través de una ventana emergente con el formulario de acceso.



¿Sabía qué?

El uso más común para los operadores de comparación es en las estructuras de selección (más adelante en esta misma investigación), dónde se quiere saber si una proposición es verdadera o falsa.

Llegado a este punto y con la finalidad de ejemplificar la mayoría de los aspectos tratados hasta el momento, en la siguiente imagen se muestra el desarrollo de un Pseudocódigo digital que lleva a cabo la comparación de dos valores proporcionados por el usuario, al finalizar su ejecución, exhibe por pantalla el resultado lógico de cada comparación.

```
1  Proceso comparacion
2      Imprimir "-- Operadores de Comparación --"
3
4      Imprimir "Ingrese un primer valor"
5      Leer valorA
6      Imprimir "Ingrese un segundo valor"
7      Leer valorB
8
9      Imprimir valorA > valorB
10     Imprimir valorA < valorB
11     Imprimir valorA >= valorB
12     Imprimir valorA <= valorB
13     Imprimir valorA != valorB
14     Imprimir valorA == valorB
15 FinProceso
16
```

```
PSeInt - Ejecutando proceso ...
*** Ejecución Iniciada. ***
-- Operadores de Comparación --
Ingrese un primer valor
> 25
Ingrese un segundo valor
> 24.9
VERDADERO
FALSO
VERDADERO
FALSO
VERDADERO
FALSO
*** Ejecución Finalizada. ***
No cerrar esta ventana  Siempre visible  Reiniciar
```

Imagen 32: Pseudocódigo digital Operadores de Comparación - Fuente: Diseñada por el autor.

Como se puede apreciar, se trata de un Pseudocódigo digital el cual se le ha identificado con el nombre de "**comparacion**". Todo comienza con la línea de código número 2 que tiene como propósito mostrar un mensaje en pantalla acerca del objetivo que se persigue con dicho desarrollo. Las líneas 4 a 7 forman parte del proceso de entrada de información y crean

una referencia de los valores proporcionados por el usuario a través de las variables identificadas como **valorA** y **valorB**. Las líneas de código 9 a 14 llevan a cabo dos actividades simultáneas dentro del Pseudocódigo digital, por un lado, comparan los valores mediante el uso de operadores de comparación, y por el otro, exponen los resultados en pantalla del ordenador. Si bien hubiese sido posible utilizar algunas variables para asignar los resultados de cada una de las comparaciones y posteriormente mostrarlas mediante la instrucción **Imprimir**, este ejemplo demuestra que es posible visualizar directamente los resultados de una expresión sin tener que recurrir a la declaración de variables adicionales. Finalmente, la línea de código número 15 indica a PSeInt el final de los procedimientos llevados a cabo por el Pseudocódigo digital.



¿Por qué utilizar operadores de comparación?

Los operadores de comparación son utilizados para comparar dos valores. Su uso más común es en las estructuras de selección, donde es posible tomar algún tipo de decisión en función del resultado de la comparación.

- Los valores implicados en una comparación pueden ser del tipo numérico, carácter, cadena de caracteres y lógicos.
- El resultado que devuelve una operación de comparación es un valor de tipo lógico, es decir, falso o verdadero.

1.2.4.3 Operadores Lógicos

Según Osvaldo Cairó Battistutti "los operadores lógicos son operadores que permiten formular condiciones complejas a partir de condiciones simples …" (Cairó 18). Es decir, actúan como operadores de unión al momento de comparar dos o más condiciones simples. Por tanto, el

resultado que producen es un valor de tipo lógico o booleano (**FALSO** o **VERDADERO**).

Generalmente, se les utiliza en combinación con las estructuras de control (más adelante en esta misma investigación) para modificar el flujo de ejecución de los Pseudocódigos digitales, de modo que su uso está directamente relacionado con la formulación de condiciones compuestas, mismas que al evaluarse como verdaderas, permiten que se ejecute una serie de instrucciones exclusivas definidas previamente por el programador.

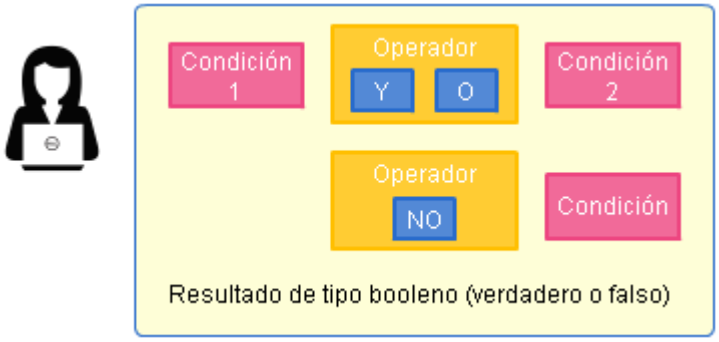


Imagen 33: Esquema que representa la construcción de expresiones a partir de operadores lógicos - Fuente: Diseñada por el autor.

Con base en los supuestos anteriores, la siguiente tabla tiene por objeto exhibir los diferentes operadores lógicos soportados por PSeInt al momento de redactar estas líneas de la investigación.

Operador	Significado	Ejemplo	Resultado
Y	Conjunción	(12 > 7) **Y** (“aceptar” == “aceptar”)	VERDADERO
O	Disyunción	(9 >= 17) **O** FALSO	FALSO
NO	Negación	**NO** (1985 == 1985)	FALSO

Tabla 8: Ejemplo Operadores Lógicos - Fuente: Elaborada por el autor.

Operador Y (Conjunción): Evalúa dos o más condiciones, emite un resultado **VERDADERO** siempre y cuando todas las condiciones involucradas en el proceso sean verdaderas (es decir, se cumplan), de lo contrario el resultado que emite será **FALSO**. Por ejemplo:

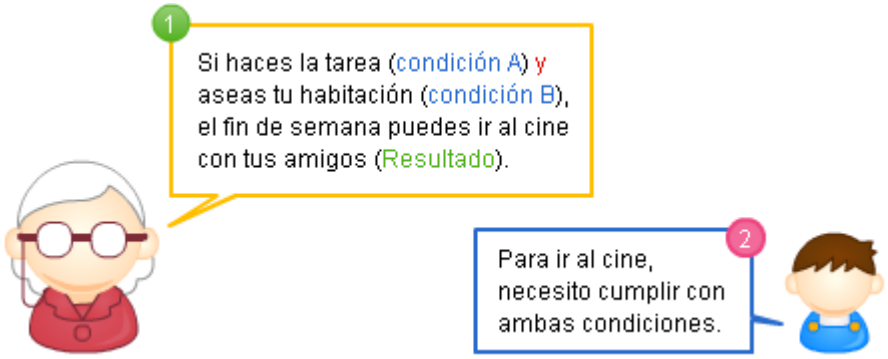


Imagen 34: Ejemplo de condiciones compuestas mediante el operador de conjunción - Fuente: Diseñada por el autor.

Al evaluar el enunciado anterior mediante el operador de conjunción (**Y**) se obtiene los siguientes resultados:

Condición A	Condición B	Proceso de Evaluación	Resultado Y (Conjunción)
VERDADERO	VERDADERO	VERDADERO **Y** VERDADERO	VERDADERO
VERDADERO	FALSO	VERDADERO **Y** FALSO	FALSO
FALSO	VERDADERO	FALSO **Y** VERDADERO	FALSO
FALSO	FALSO	FALSO **Y** FALSO	FALSO

Tabla 9: Tabla de Verdad Operador de Conjunción - Fuente: Elaborada por el autor.

Operador O (Disyunción): Evalúa dos o más condiciones, emite un resultado **FALSO** siempre y cuando todas las condiciones involucradas en el proceso sean falsas (es decir, no se cumplan), de lo contrario el resultado que emite será **VERDADERO**. Por ejemplo:

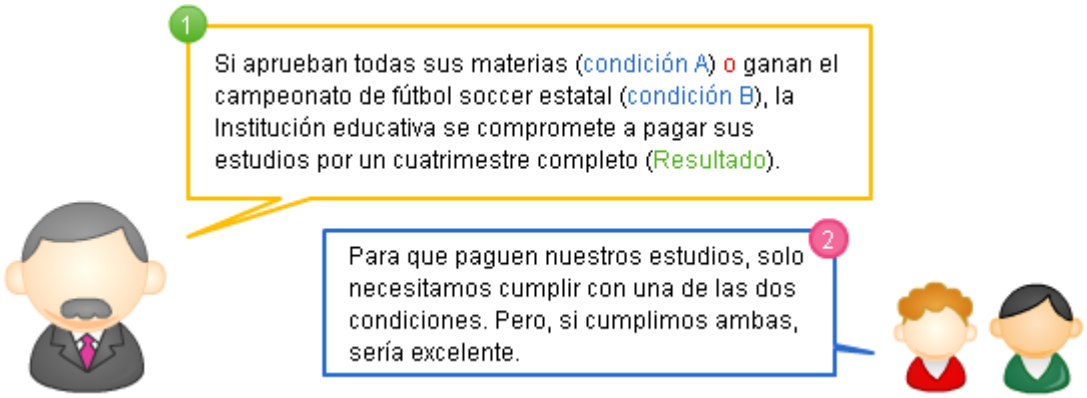


Imagen 35: Ejemplo de condiciones compuestas mediante el operador de disyunción - Fuente: Diseñada por el autor.

Al evaluar dicho enunciado mediante el operador de disyunción (**O**) se obtiene que:

Condición A	Condición B	Proceso de Evaluación	Resultado O (Disyunción)
VERDADERO	VERDADERO	VERDADERO **O** VERDADERO	VERDADERO
VERDADERO	FALSO	VERDADERO **O** FALSO	VERDADERO
FALSO	VERDADERO	FALSO **O** VERDADERO	VERDADERO
FALSO	FALSO	FALSO **O** FALSO	FALSO

Tabla 10: Tabla de Verdad Operador de Disyunción - Fuente: Elaborada por el autor.

Operador NO (Negación): Evalúa una condición, emite un resultado **VERDADERO** si la condición involucrada en el proceso es evaluada como falsa y, por el contrario, si la condición es evaluada como falsa el operador de negación emite un resultado **VERDADERO**. Por ejemplo:

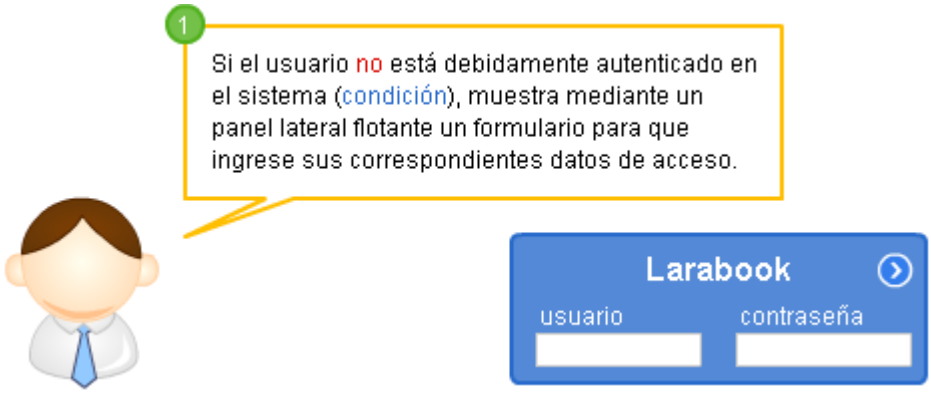


Imagen 36: Ejemplo de condición simple construida mediante el operador de negación - Fuente: Diseñada por el autor.

Al evaluar el enunciado anterior mediante el operador de negación (NO), se obtiene que:

Condición	Proceso de Evaluación	Resultado NO (Negación)
VERDADERO	**NO** VERDADERO	FALSO
FALSO	**NO** FALSO	VERDADERO

Tabla 11: Tabla de Verdad Operador de Negación - Fuente: Elaborada por el autor.

De la tabla anterior, se puede entender que si el usuario se encuentra debidamente autenticado en el sistema (**VERDADERO**), el panel lateral flotante con el formulario de acceso no se muestra en pantalla (resultado **FALSO**). Sin embargo, si el usuario no se encuentra *logeado* actualmente en el sistema (**FALSO**), el panel lateral flotante sí que se muestra en pantalla del ordenador (resultado **VERDADERO**) y exhorta al usuario a que se autentifique nuevamente con la ayuda de los controles disponibles en el formulario de acceso.



¿Sabía qué?

Dependiendo de la problemática a resolver, es posible utilizar más de un operador lógico para formular proposiciones más complejas (véase el segundo ejercicio correspondiente al apartado: funciones primitivas del lenguaje).

Con la finalidad de reforzar los conocimientos aprendidos hasta el momento correspondientes al trabajo con operadores lógicos, el siguiente Pseudocódigo digital realiza una serie de cuestionamientos al usuario con la finalidad de evaluarlos y emitir un resultado lógico en el monitor:

```
1   Proceso operadores_logicos
2       Imprimir "---------- OPERADORES LÓGICOS ------------"
3       Imprimir ""
4       Imprimir "ENUNCIADO:"
5       Imprimir "-----------------------------------------"
6       Imprimir "Si haces la tarea y/o aseas tu habitación,"
7       imprimir "el fin de semana puedes ir al cine con tus"
8       Imprimir "amigos."
9       Imprimir "-----------------------------------------"
10      Imprimir ""
11
12      Imprimir "Realizó la tarea: (verdadero/falso)"
13      Leer condicionA
14      Imprimir "Aseó su cuarto:   (verdadero/falso)"
15      Leer condicionB
16
17      Imprimir ""
18      Imprimir "¿Puede ir al cine el fin de semana?"
19      Imprimir "-----------------------------------------"
20      Imprimir "Operador Y (Conjunción): " , (condicionA Y condicionB)
21      Imprimir "Operador O (Disyunción): " , (condicionA O condicionB)
22      Imprimir "--------- NEGACIÓN ------------------------"
23      Imprimir "¿Realizó la tarea?: " , (NO condicionA)
24      Imprimir "¿Aseó su cuarto?:   " , (NO condicionB)
25  FinProceso
```

Imagen 37: Pseudocódigo digital Operadores Lógicos - Fuente: Diseñada por el autor.

El ejemplo anterior es simple, las líneas 2 a 10 se encargan de mostrar información referente al Pseudocódigo digital, así como la formulación de un enunciado el cuál más adelante deberá atender el usuario mediante una serie de respuestas del tipo **FALSO** o **VERDADERO**. Las líneas de código 12 a 15 están comisionadas para formular los cuestionamientos necesarios al usuario con base al enunciado anterior y recaban cada una de las respuestas proporcionadas en variables independientes identificadas como **condicionA** y **condicionB**. Las líneas 17 a 24 se encargan de procesar los datos y emiten las salidas de información correspondientes al usuario, siendo de especial interés (para el caso que ocupa este ejemplo) las líneas de código 20-24; donde se observa claramente el uso de operadores lógicos (de conjunción, disyunción y negación) para evaluar las respectivas expresiones lógicas.

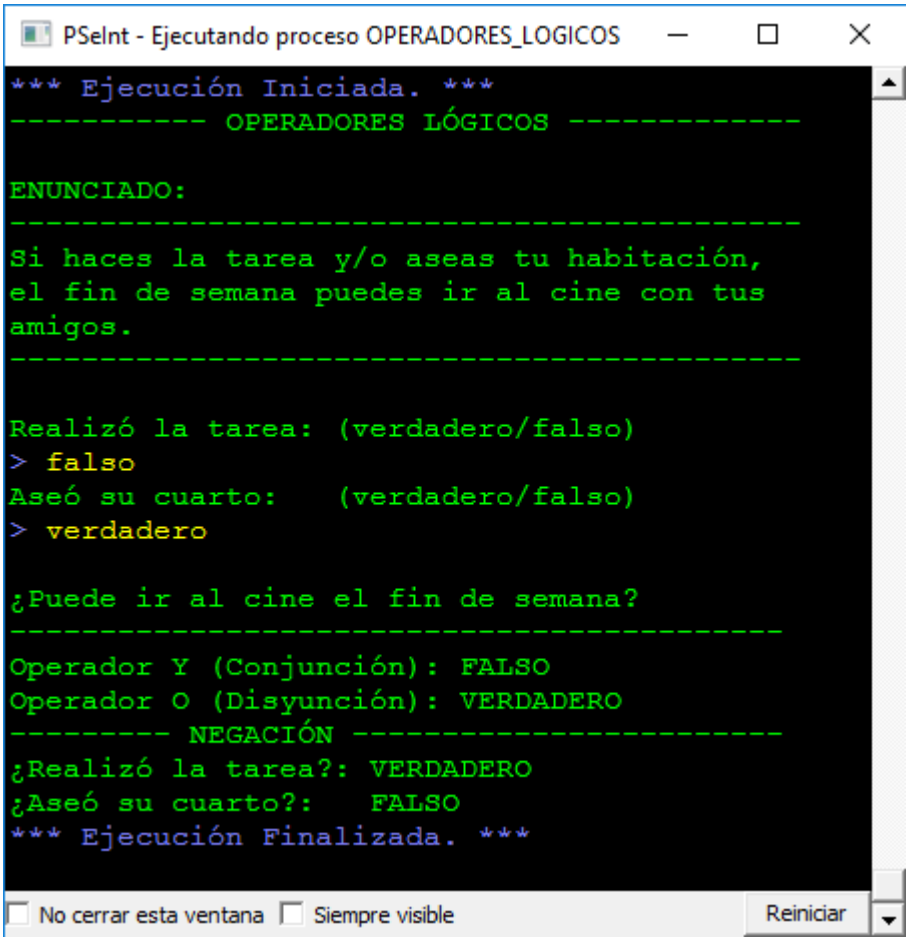


Imagen 38: Ejecución Pseudocódigo digital Operadores Lógicos - Fuente: Diseñada por el autor.

Al ejecutar el Pseudocódigo digital, PSeInt despliega una serie de informaciones (título y enunciado) en pantalla seguida de dos cuestionamientos clave, los cuales deben ser atendidos por el usuario mediante respuestas de tipo booleano, es decir, **FALSO** o **VERDADERO** (escritas en mayúsculas o minúsculas). Una vez ingresadas las respuestas, PSeInt comienza a evaluarlas mediante el uso de expresiones lógicas, y al final, emite los resultados pertinentes a cada operación.



¿Por qué utilizar operadores lógicos?

Los operadores lógicos se utilizan para agrupar expresiones lógicas (comparaciones y valores de tipo lógico) y poder así comparar proposiciones más complejas.

- El resultado que producen los operadores lógicos es un valor booleano, es decir, falso o verdadero.

El modo en que actúan los operadores lógicos se resume en las llamadas tablas de verdad. Por ejemplo:

a	b	a Y b	a O b
VERDADERO	VERDADERO	VERDADERO	VERDADERO
VERDADERO	FALSO	FALSO	VERDADERO
FALSO	VERDADERO	FALSO	VERDADERO
FALSO	FALSO	FALSO	FALSO

Operador O (Disyunción): De la tabla de verdad anterior, se deduce que, si al menos una de las dos expresiones es verdadera, el resultado será verdadero.

Operador Y (Conjunción): De la tabla de verdad anterior, se deduce que, si al menos una de las dos expresiones es falsa, el resultado será falso.

a	NO a
VERDADERO	FALSO
FALSO	VERDADERO

Operador NO (Negación): De la tabla de verdad anterior, se deduce que, al utilizar el operador de negación, los valores se intercambian.

1.2.4.4 Prioridad de Operadores

La prioridad de operadores desde la perspectiva de Gil Rubio "… describe la secuencia en que se van a evaluar las operaciones solicitadas en una instrucción …" (Gil 60). Es decir, en una expresión que contiene varios operadores matemáticos, éstos deben ser ejecutados con base en una jerarquía predeterminada que permita resolver dicha operación (véase la imagen 39). Si una fórmula contiene operadores con la misma prioridad, estos se evalúan de izquierda a derecha. Sin embargo, para cambiar el orden de evaluación, es importante que se delimite entre paréntesis la parte de la fórmula que se requiera calcular primero.

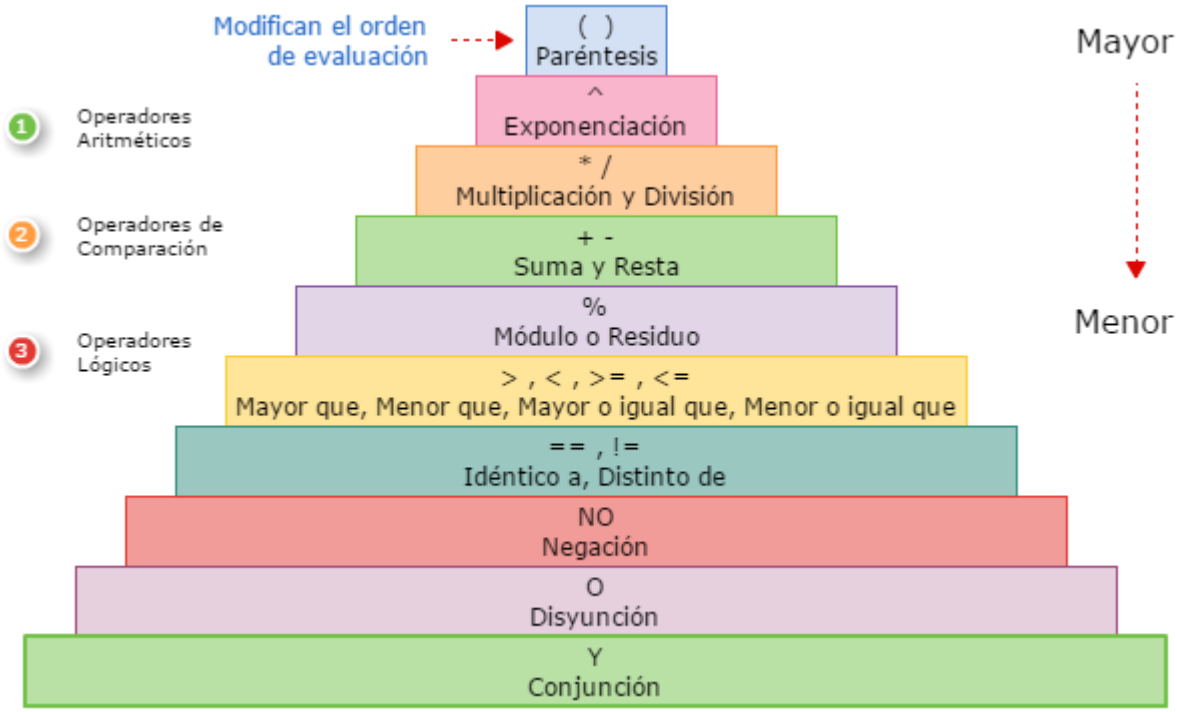


Imagen 39: Pirámide que representa la prioridad entre operadores - Fuente: Diseñada por el autor.

El motivo de orillar a los ordenadores a seguir esta jerarquía de operadores para realizar sus correspondientes operaciones, se debe principalmente a que es la única manera de garantizar que una

determinada expresión se evalué y arrojé el mismo resultado en cualquier ordenador. Dicho en otras palabras, la prioridad de los operadores permite a un ordenador evaluar de una y solo una forma una determinada expresión matemática para que no exista espacio u oportunidad para ambigüedades.

Sumado a lo expuesto, es importante enfatizar que cuando se utilizan paréntesis, el ordenador detecta en primer lugar los que aparecen a nivel más interno y dentro de ellos aplica la pirámide de jerarquía de operadores. Luego, evalúa las operaciones que aparecen delimitadas por paréntesis externos, hasta descartar la presencia de estos. Finalmente, realiza una evaluación de cada uno de estos resultados calculados y emite un resultado general. Por ejemplo:

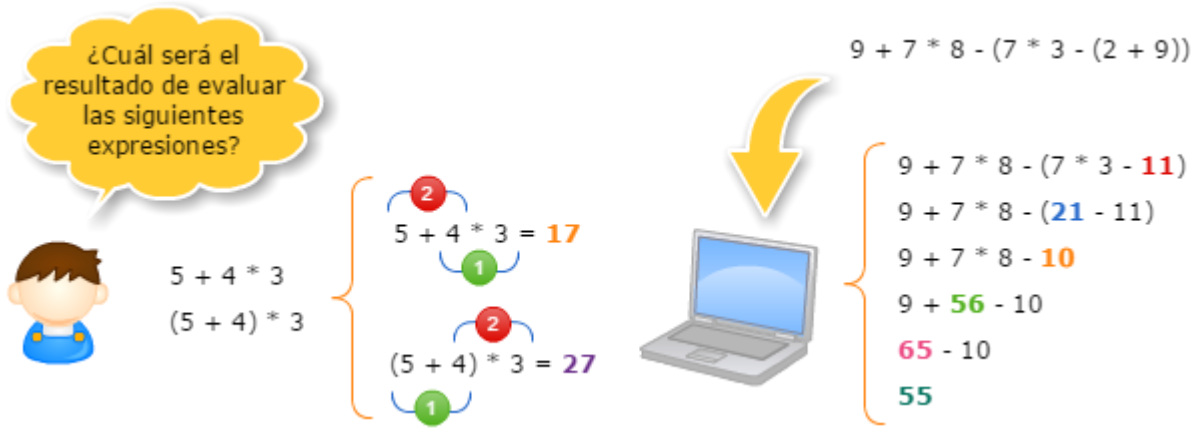


Imagen 40: Los paréntesis permiten alterar la prioridad entre operadores - Fuente: Diseñada por el autor.

Como se puede apreciar, la prioridad de los operadores permite aclarar de forma inequívoca el orden que se debe seguir durante el cálculo de una determinada expresión matemática. Además, para cambiar el orden predeterminado de las operaciones, se debe emplear el uso de paréntesis, pero, solo en aquellas zonas de la expresión matemática donde se requiera forzar la prioridad de su procesamiento.

Frente a este escenario, es probable que en ciertas ocasiones la solución propuesta (fórmula) a un determinado problema pudiese parecer correcta. Sin embargo, es importante tener en cuenta que ésta siempre estará determinada por el orden en el que aparecen sus correspondientes operadores.



Expresión	Procedimiento	Resultado	Satisface la necesidad
8.2 + 10.0 / 2	8.2 + 5.0	**13.2**	⛔
(8.2 + 10.0) / 2	18.2 / 2	**9.1**	✅

Imagen 41: Cuando se formula una expresión matemática, es importante tener presente la prioridad de los operadores - Fuente: Diseñada por el autor.



¿Por qué es importante conocer la prioridad de los operadores?

La **prioridad de los operadores** permite aclarar de forma inequívoca el orden que se debe seguir durante el cálculo de una determinada expresión matemática, de tal manera que no exista espacio u oportunidad para que se generen ambigüedades mientras dura el proceso. No obstante, si es necesario alterar ese orden, se puede recurrir a escribir las distintas partes de la expresión matemática entre paréntesis, lo que provoca que se calculen en primer lugar. Sin embargo, las operaciones escritas dentro de los paréntesis continuarán ejecutándose en el orden de prioridad de los distintos operadores.

Llegado hasta este punto, seguramente el lector se habrá percatado que conforme se abordan nuevos ejemplos, la mayoría de estos tienden a crecer en cuanto a tamaño (mayor número de líneas de código), lo que provoca que su lectura y comprensión se torne cada vez más compleja, puesto que no existen (a nivel de código) apartados con documentación

que orienten al programador a razonar cada una de las funciones que desempeñan las líneas de código presentes en el Pseudocódigo digital.

En este sentido, el siguiente apartado aborda todo lo referente a los comentarios, un mecanismo presente en la mayoría de los lenguajes de programación que permite a los programadores documentar aquellas líneas de código sobresalientes de sus programas, con la finalidad de que más adelante (si así lo desean), sean ellos u otros colegas los que realicen los mantenimientos o actualizaciones necesarias al programa (código) sin poner en riesgo su funcionamiento básico (puesto que al existir documentación previa, es más sencillo comprender las tareas que desempeña cada línea de código, y por consiguiente, sea más sencillo mejorarlas o añadir nuevas funcionalidades).



Resumen: Operadores.

En esta sección se abordaron los diferentes tipos de operadores involucrados en la solución de problemas mediante el uso de un ordenador.

Las variables y constantes por si solas son de poca utilidad. Para hacer Pseudocódigos digitales realmente útiles son necesarias otro tipo de herramientas. En términos generales, los **operadores** permiten manipular el valor de las variables, realizar operaciones matemáticas con sus valores y comparar su contenido. De esta forma, los operadores permiten a los Pseudocódigos digitales realizar cálculos complejos y tomar decisiones lógicas en función de comparaciones y otros tipos de condiciones.

PSeInt soporta tres tipos de operadores para el procesamiento de información: **operadores aritméticos**, **operadores de comparación** y **operadores lógicos**. Por si esto fuera poco, también tiene la posibilidad de simular el comportamiento de un tipo de operador especial (presente en la mayoría de los lenguajes de programación) para la concatenación de cadenas de caracteres, llamado operador de texto; sin embargo, su funcionamiento en PSeInt queda limitado en combinación con la instrucción **Imprimir**.

Por lo tanto, al momento de procesar información es importante tener presente que:

- Para ejecutar las operaciones matemáticas de suma, resta, multiplicación, división, exponenciación y módulo, es necesario utilizar los operadores aritméticos.
- Los operadores de comparación son utilizados para comparar dos valores. Su uso más común es en las estructuras de selección, donde es posible tomar algún tipo de decisión en función del resultado de la comparación (falso o verdadero).
- Los operadores lógicos se utilizan para agrupar expresiones lógicas y poder así comparar proposiciones más complejas.

1.2.5 Comentarios

Los comentarios son considerados como una capacidad presente en la mayoría de los lenguajes de programación que permite a los

programadores describir el funcionamiento interno de cada uno de sus programas. Usualmente son añadidos con el propósito de hacer el código fuente más fácil de entender con vistas a su mantenimiento o reutilización, puesto que, con el paso del tiempo, es fácil olvidar como funciona internamente cada instrucción declarada en el programa.

Bajo este mismo orden de ideas, Cairó añade que "es conveniente cuando realizamos una tarea compleja poner comentarios que expresen o ayuden a entender lo que hicimos" (Cairó 7). En este sentido, resulta evidente comentar sólo aquellas líneas de código que por su función sean **importantes para el programa**; de modo que permita conocer a futuro, los procedimientos que se han llevado a cabo para determinar la solución del problema.

Con base en lo expuesto, PSeInt permite introducir comentarios a los Pseudocódigos digitales mediante el uso de la doble barra **//**. Es decir, todo lo que precede a **//**, hasta el fin de línea, es considerado como parte del comentario. De esta forma, al momento de la ejecución, PSeInt ignora todo el contenido precedido por la doble barra, y centra su atención en el procesamiento de las instrucciones restantes.



¿Sabía qué?

Al agregar una barra adicional a un comentario, es decir, **///**. PSeInt resalta con un color diferente la información declarada en dicho comentario.

Con la finalidad de ejemplificar cada una de las ideas expuestas hasta el momento, la siguiente imagen ilustra el diseño de un Pseudocódigo digital

cuyas instrucciones se encuentran debidamente documentadas mediante el uso de comentarios:

```
1   Proceso comentarios
2       ///Programa Rectángulo: Obtiene área y perímetro
3       ///Desarrollador: Alejandro González
4       ///Versión: 1.0
5       ///Fecha: 13 Mayo 2016
6
7       //Información de Cabecera
8       Imprimir "---------------------------------------"
9       Imprimir "PROGRAMA RECTÁNGULO - ÁREA Y PERÍMETRO"
10      Imprimir "---------------------------------------"
11      Imprimir ""
12      //Datos de entrada
13      Imprimir "Ingrese base (expresada en metros)"
14      Leer medidaBase
15      Imprimir "Ingrese altura (expresada en mts.)"
16      Leer medidaAltura
17      //Proceso:
18      //Obtener perímetro rectángulo
19      perimetro = (medidaBase * 2) + (medidaAltura * 2)
20      //Obtener área rectángulo
21      area = medidaBase * medidaAltura
22      //Salida de información
23      Imprimir "Perímetro: " , perimetro , " mts."
24      Imprimir "Área:      " , area , " mts. cuadrados"
25  FinProceso
```

Imagen 42: Los comentarios permiten documentar instrucciones relevantes en los Pseudocódigos - Fuente: Diseñada por el autor.

Como se puede observar, las líneas de código 2 a 5 sostienen una serie de comentarios (resaltados) que indican al programador que se trata de un Pseudocódigo digital que obtiene el área y perímetro de una figura rectangular. A su vez, muestran el nombre de su desarrollador principal, el número de versión del producto (solución) y la fecha de su lanzamiento.

Sumado a lo expuesto, el comentario declarado en la línea número 7 informa que las siguientes cuatro instrucciones son utilizadas dentro del

Pseudocódigo digital para mostrar información de cabecera. En seguida, el comentario de la línea 12 indica al programador que las instrucciones 13-16 permiten la entrada de información por parte del usuario. Más adelante, la línea de código número 17 establece que a partir de ese momento se comienza con el proceso de solución al problema, para lo cual, las líneas 19 y 21 son las encargadas de procesar los datos proporcionados por el usuario a través de una serie de expresiones matemáticas que permiten obtener tanto el perímetro como el área de la figura geométrica en cuestión (rectángulo), y al mismo tiempo, asignan los resultados de dicho proceso en las variables identificadas como **perimetro** y **area** respectivamente.

Finalmente, el comentario declarado en la línea de código número 22, establece que las siguientes dos instrucciones son las encargadas de emitir los resultados de la operación (perímetro y área del rectángulo) en la pantalla del ordenador.

Bajo este mismo orden de ideas, al ejecutar paso a paso el ejemplo anterior (**Tecla F5**[12]), se puede observar cómo el contenido declarado después de la doble barra es ignorado por el intérprete de PSeInt. Esto es así, debido a que los comentarios consisten en información significativa para el programador, mas no para el Pseudocódigo digital (véase la imagen 43).

[12] Recuerde que las funciones asignadas a cada tecla especial de su ordenador, depende directamente de la configuración de su teclado y del sistema operativo que actualmente tenga instalado. Para mayor información, sírvase en leer la documentación que acompaña a su equipo, o presione el icono **Ejecutar paso a paso** que aparece junto a la barra de herramientas de PSeInt.

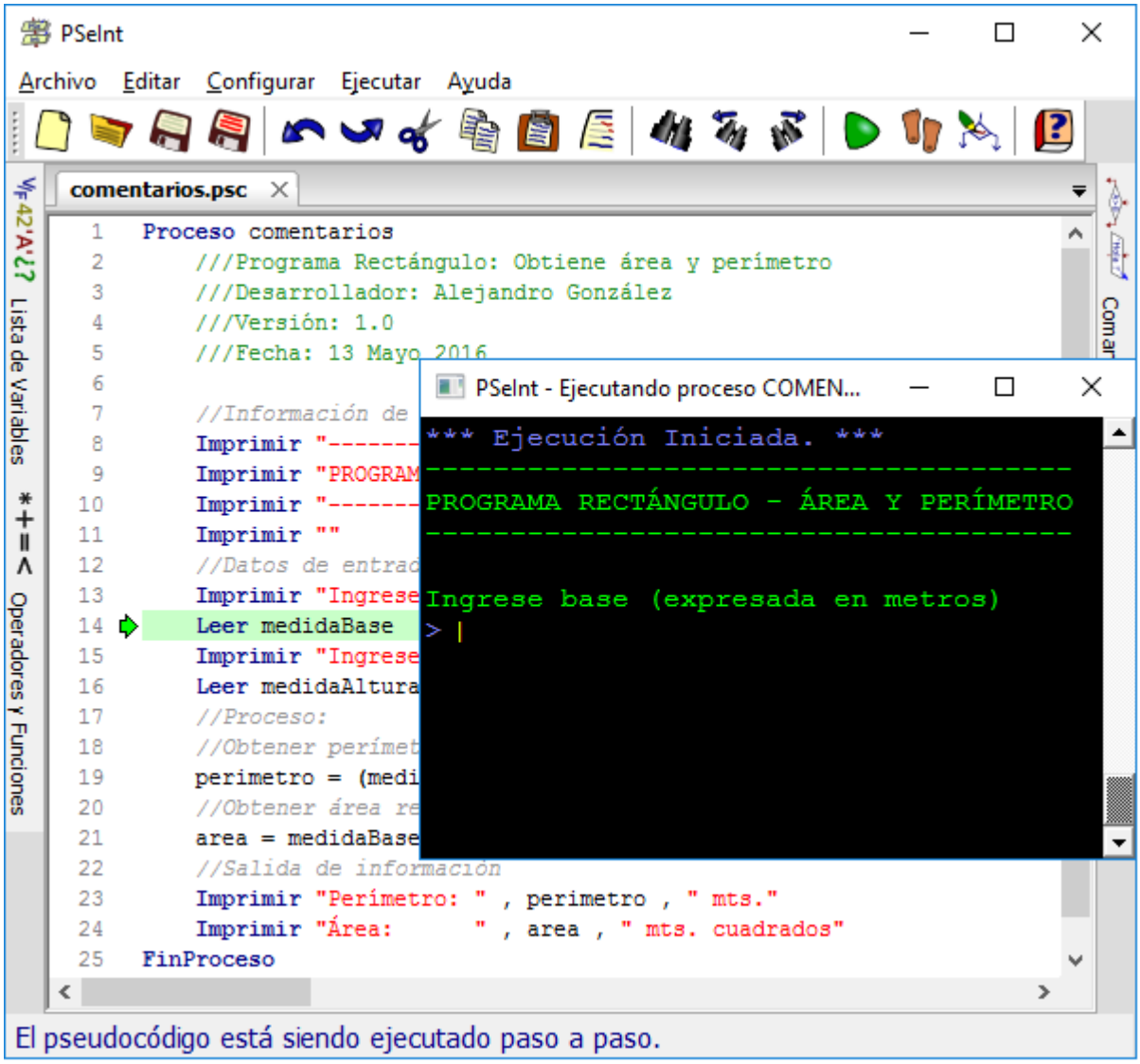


Imagen 43: Ejecución paso a paso en PSeInt - Fuente: Diseñada por el autor.

La ejecución paso a paso (**Tecla F5**) disponible en PSeInt, permite realizar un seguimiento más detallado sobre de la ejecución y comportamiento de un Pseudocódigo digital. Es decir, permite observar en tiempo real qué instrucciones y en qué orden se ejecutan, así como también observar el contenido de las variables o expresiones durante el proceso.



Resumen: Comentarios.

En esta sección se abordaron las distintas formas de agregar comentarios a los Pseudocódigos digitales desarrollados mediante PSeInt.

Los comentarios son una herramienta de gran valor para un Pseudocódigo digital moderadamente extenso o complejo que se quiere mantener en uso (quizá con alguna modificación) por una larga temporada de tiempo.

Es fácil olvidar como funciona internamente un Pseudocódigo digital, pero, sobre todo el por qué uno escribió el código en el modo en que lo hizo. En este sentido, es una buena idea añadir comentarios para ayudarnos a sobre-llevar la lógica del procedimiento declarado y a explicar cualquier instrucción relativamente difícil o compleja que forme parte del mismo.

En PSeInt es posible añadir comentarios de una sola línea haciendo uso de la doble barra **//**. Sin embargo, al añadir una tercera barra diagonal **///**, la presentación de los comentarios mejora (cambia de color), posibilitando al programador a utilizarlos al inicio de sus Pseudocódigos digitales para explicar de forma generalizada el objetivo que persiguen, indicar la fecha de su desarrollo, incluir el número de versión y proporcionar información acerca de los derechos de autor.

Por último, se recomienda añadir comentarios en todos los lugares donde se crea que serán de utilidad más tarde. Si una línea de código es particularmente obvia, probablemente no se necesita un comentario. Por ejemplo, no hay razón para añadir un comentario en una instrucción simple como **Imprimir "Hola usuario"**, porque es obvio lo que hace en el Pseudocódigo digital (muestra en pantalla la leyenda **Hola usuario**).

Los Pseudocódigos digitales vistos hasta el momento han consistido en simples secuencias de instrucciones; sin embargo, en la vida real existen problemáticas más complejas que no pueden ser resultas a través de un esquema sencillo. Por ejemplo, repetir una misma acción un número determinado de veces, evaluar una expresión y realizar acciones diferentes con base al resultado generado, reutilizar un bloque de instrucciones en diferentes partes del proceso, etc. Es por ello, que dedicaremos el siguiente capítulo en descubrir cuáles son los elementos avanzados

presentes en el diseño de pseudocódigos para la solución de problemas "complejos" mediante el uso de un ordenador.

Conclusión

Después de haber analizado los fundamentos gramaticales del Pseudocódigo mediante programación secuencial, merece la pena detenerse un momento en la implementación de éstos para la solución de problemas a través del uso de un ordenador, dado que permiten al estudiante centrar su atención en el diseño de diferentes alternativas, sin perder tiempo en los detalles técnicos que conlleva adoptar un lenguaje de programación desde los primeros inicios de su formación como programador.

De no ser así, es probable que el estudiante termine por aprender a desarrollar programas de computadora con la ayuda de un lenguaje de programación en particular. Sin embargo, la mayoría del tiempo invertido durante su formación, realmente lo habrá malgastado en solucionar problemas de sintaxis (por ejemplo: la falta de puntos y comas al finalizar cada instrucción, alguna comilla doble sin cerrar, asignar valores a una variable que no corresponden con su tipo de dato definido, etc.), lo que resulta en la pérdida de oportunidades para encarar retos cada vez mayores que le ayuden a mejorar su habilidad de razonamiento (lógico-matemático).

Con base en los ejemplos analizados y las definiciones que hacen los distintos autores seleccionados para este capítulo de la investigación, se verifica parcialmente la hipótesis que argumenta: los elementos

gramaticales necesarios para aprender las bases fundamentales de la algoritmia computacional son:

a) Identificadores
b) Variables y Constantes
c) Entradas y Salidas de Información
d) Tipos de Datos (Numéricos, Caracteres y Booleanos)
e) Tipos de Operadores (Aritméticos, Comparación y Lógicos)
f) Comentarios

Puesto que, como se pudo observar a lo largo de la redacción de este capítulo, la mayoría de los ejemplos abordados omiten el uso del elemento gramatical de tipo constante para la solución problemas. Aun cuando su presencia fuese de carácter indiscutible, por ejemplo: para determinar el área de un círculo ($PI * radio^2$), el valor de la constante PI puede ser sustituido de forma directa por el valor numérico 3.14159265359, de tal manera, que su uso dentro de este escenario, queda limitado a un estilo de programación personal y no como un elemento sumamente importante que se debe dominar para aprender las bases fundamentales de la algoritmia computacional.

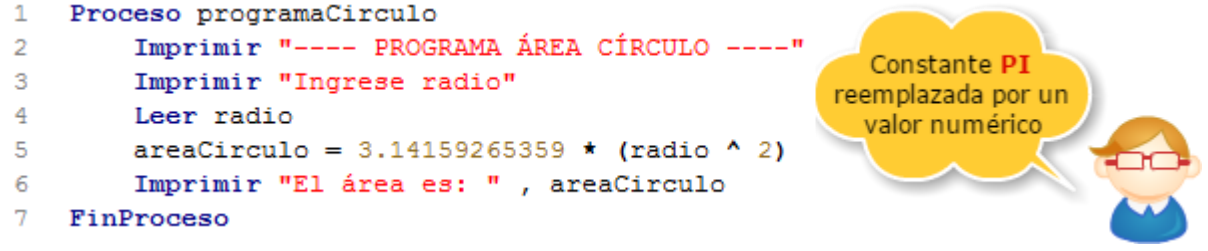


Imagen 44: Los identificadores de tipo constante pueden ser sustituidos de forma directa por un valor constante - Fuente: Diseñada por el autor.

Por otra parte, para mejorar la presentación de los resultados generados por los Pseudocódigos, es importante que éstos se encuentren

debidamente acompañados de una serie de informaciones descriptivas que brinden contexto suficiente al usuario, acerca de su presencia en alguna parte especifica del proceso. En este sentido, al retornar la mirada hacia los ejemplos expuestos dentro de este mismo capítulo, se puede percibir que aquellos que utilizan el operador de texto (signo coma en PSeInt) como recurso estético para concatenar sus salidas, fomentan considerablemente la comprensión de los resultados exhibidos al usuario, de tal manera, que su cometido dentro de este apartado de la investigación, se torna indispensable para aprender las bases fundamentales de la algoritmia computacional, puesto que un Pseudocódigo que entrega información puntual y detallada como respuesta, mejora la fidelidad, credibilidad y entendimiento de sus usuarios.

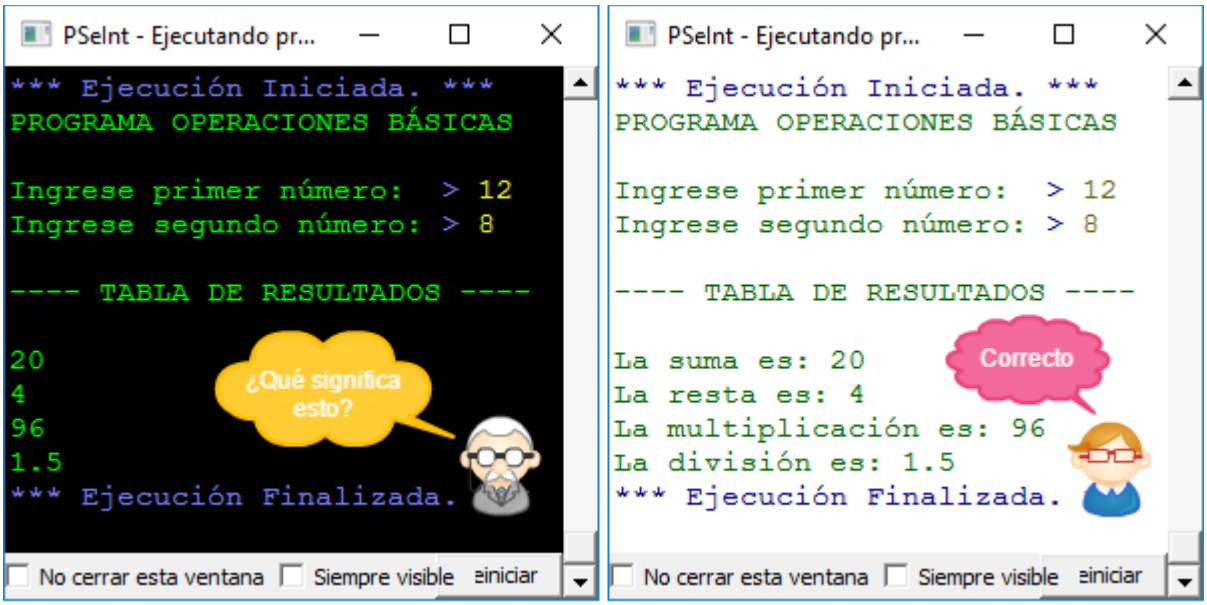


Imagen 45: La concatenación en las salidas de información, fomenta considerablemente la comprensión de los resultados exhibidos al usuario - Fuente: Diseñada por el autor.

De esta manera, la tesis establece que: los elementos gramaticales necesarios para aprender las bases fundamentales de la algoritmia computacional son:

a) Identificadores
b) Variables
c) Entradas y Salidas de Información
d) Tipos de Datos (Numéricos, Caracteres y Booleanos)
e) Tipos de Operadores (Aritméticos, Comparación, Lógicos y de Texto)
f) Comentarios

Sumado a estos aspectos, también es importante considerar el uso de una buena herramienta digital (Pseudo-Intérprete) que asista a el estudiante en sus primeros pasos como programador, de modo que le proporcione un entorno de trabajo con numerosas ayudas, asistencias y recursos didácticos, que le permitan ejecutar, probar, localizar errores y comprender la lógica de los algoritmos desarrollados.

Finalmente, con la intensión de resumir y relacionar cada uno de los factores expuestos dentro de este capítulo, se propone el uso del siguiente esquema como material de apoyo didáctico para la solución de problemas mediante el uso de un ordenador.

CAPÍTULO 1

Fundamentos Gramaticales del Pseudocódigo mediante Programación Secuencial.

Para solucionar problemas mediante el uso de una computadora, se requiere la construcción de:

Algoritmos

1

Características:

Precisos

Finitos

Definidos

Partes que deben describir:

Entrada → Proceso → Salida

Herramientas para representar el diseño de un Algoritmo

2

- Lenguaje Natural
- Diagramas de Flujo
- Pseudocódigos

Herramientas para escribir, ejecutar y probar Algoritmos por medio de un ordenador

3

Pseudo-Intérpretes

- PSeInt
- SLE
- LPP

4 Aspectos básicos presentes en el Pseudocódigo

4.1 Solicitar y mostrar información al exterior

- Instrucciones de Entrada
- Instrucciones de Salida

4.2 La naturaleza de la información puede ser de tipo:

- Numérica
- Lógica
- Carácter(es)
- Arreglos

4.3 Para almacenar información dentro de un Pseudocódigo se utilizan:

- Variables
- Constantes

4.4 Para hacer refencia a información que maneja un Pseudocódigo se hace uso de:

- Identificadores

4.5 El proceso de información que lleva a cabo internamente un Pseudocódigo es posible gracias al uso de:

- Operadores Aritméticos
- Operadores de Comparación
- Operadores Lógicos

4.6 En la medida que un Pseudocódigo crece es importante hacer uso de:

- Comentarios



PROGRAMACIÓN SECUENCIAL

Capítulo 2

Fundamentos Gramaticales del Pseudocódigo mediante Programación Estructurada y Funcional

Hasta el momento se han abordado conceptos y técnicas esenciales presentes en el desarrollo de Pseudocódigos digitales para la solución de problemas mediante el uso de un ordenador. Sin embargo, en la vida real existen tareas más complejas que no pueden ser resultas empleando un esquema tan sencillo, es decir, en ocasiones existe la necesidad de repetir una misma acción un número determinado de veces o evaluar una expresión y realizar acciones diferentes con base al resultado generado.

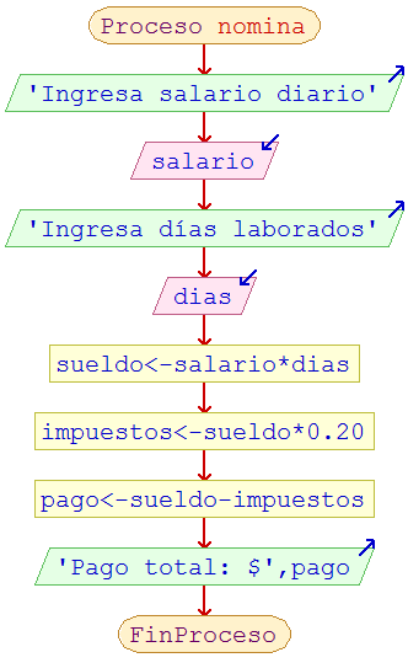


Imagen 46: Diagrama de Flujo a partir de un programa secuencial - Fuente: Diseñada por el autor.

Cuando se habla de control de flujo, se hace referencia al control que tienen los programadores sobre el orden de ejecución de las diferentes instrucciones declaradas en un programa de computadora. En el capítulo anterior, se han desarrollado Pseudocódigos digitales cuyo flujo se dice que es secuencial, es decir, cada una de sus instrucciones se ejecutan en secuencia, una después de otra, en el mismo orden en que fueron declaradas dentro del código. Tal y como se ilustra en la imagen 46.

Sin embargo, este tipo de ejecución (programación secuencial) tiene grandes

limitantes. Por ejemplo, imagínese que los accionistas de un centro educativo solicitan el desarrollo de una pequeña aplicación para determinar el importe a cobrar por concepto de colegiatura a cada uno de sus estudiantes. Ahora bien, por motivos de promoción el centro educativo cuenta con un programa de recompensas en el cual aplica un determinado descuento sólo en aquellos estudiantes cuyo promedio supere ciertos límites: si el promedio es mayor a 8.0 puntos, el descuento será del 5% y si supera los 9.5 puntos el descuento es del 40%.

En un caso descrito como el anterior, el flujo secuencial no será de mucha utilidad, puesto que surgen ciertas incógnitas durante el proceso imposibles de resolver, por ejemplo: ¿cómo define el Pseudocódigo digital el descuento a aplicar?, ¿será el 5% o el 40%? Según la descripción del problema, el descuento depende del promedio obtenido por cada estudiante, pero ¿Cómo decide eso el Pseudocódigo digital?

Para resolver este tipo de situaciones existe la denominada programación estructurada, que posee características que permiten desarrollar de forma flexible todo tipo de Pseudocódigos digitales a partir de tres tipos fundamentales de estructuras de control: Secuencial, Alternativa y Repetitiva. Por otra parte, al implementar el paradigma de la programación funcional, se simplifica la escritura y lectura de las instrucciones declaradas en el código, puesto que las funciones permiten encapsular tareas específicas, mismas que pueden ser reutilizadas en cualquier parte del proceso para la solución del problema.

Por tal motivo, el presente capítulo tiene por objeto ser un complemento "de nivel intermedio" a la guía tutorial que se ha desarrollado al inicio de esta investigación, mismo que aborde las bases fundamentales de la

algoritmia computacional, pero bajo un enfoque orientado a la programación estructurada y funcional.

Se trata básicamente de responder a la pregunta de ¿Qué elementos gramaticales del Pseudolenguaje se deben considerar para aprender las bases fundamentales de la algoritmia computacional mediante programación estructurada y funcional?, para lo cual surge el planteamiento de la hipótesis que argumenta que los elementos gramaticales necesarios para aprender las bases fundamentales de la algoritmia computacional mediante programación estructurada y funcional son:

a) Estructuras de Control (Selección y Repetición)
b) Funciones Primitivas del Lenguaje
c) Funciones del Programador

En este sentido, para abordar de manera puntual el objetivo que se persigue dentro de dicho capítulo, es necesario remitirse nuevamente a los aportes que diferentes autores sostienen acerca de cada uno de estos aspectos, así como los beneficios que conlleva considerarlos durante la formación académica de un estudiante en el campo de la programación.

Al finalizar la lectura de este capítulo, el lector contará con los conocimientos y habilidades necesarias para resolver problemas de mediana complejidad a través del uso de un ordenador. Por ejemplo: determinar si un usuario tiene la mayoría de edad, ejecutar un determinado proceso (subprograma) con base en la opción seleccionada por un usuario, realizar diferentes conversiones de unidades de medida, generar la tabla de multiplicar de un número determinado, entre otros.

Es por ello que, para ayudarle a sacar el mayor partido al texto y saber dónde se encuentra en cada momento de las explicaciones proporcionadas a lo largo de este capítulo, la mayoría de los ejemplos citados se presentan nuevamente en un formato de captura de pantalla, donde es posible obtener su correspondiente código fuente (archivo PSeInt) a través de la siguiente <URL>.

2.1 Aspectos de nivel intermedio presentes en el Pseudocódigo

Hasta el momento se ha hecho un largo recorrido por los elementos básicos presentes en el Pseudocódigo para la solución de problemas mediante el uso de un ordenador. Sin embargo, el crear una variable y asignar un valor en ella para que más adelante sea manipulado como parte del proceso, con frecuencia no será de mucha ayuda. Es por ello, que en el presente apartado se realiza una exposición puntual acerca de las estructuras de control que permiten elaborar Pseudocódigos digitales complejos, hasta llegar a la definición de funciones para organizar y reutilizar instrucciones que se encargan de resolver una tarea en específico.

En un primer momento se presentan las estructuras de control que nos permiten crear condiciones y realizar iteraciones. Posteriormente, se hace un recorrido por uno de los principales conceptos de la programación procedural: las funciones. Revisaremos el repertorio de funciones predefinidas en PSeInt, aprenderemos a definir nuestras propias funciones, cómo pasar parámetros, cómo retornar un resultado y cómo reutilizarlas en nuestro código. Finalmente, con el objeto de reforzar cada uno de los conceptos abordados a lo largo de este apartado, el anexo B presenta una serie de casos prácticos (por temática) mismos que son solucionados y explicados paso a paso, a través del diseño de distintos Pseudocódigos. A continuación, la segunda parte de las bases fundamentales de la algoritmia computacional.

2.1.1 Estructuras de Control

Los ejemplos que se han trabajado a lo largo del primer capítulo de esta investigación son muy habituales: aceptan uno o más valores de entrada, realizan una serie de cálculos con esos datos y retornan uno o más resultados. Sin embargo, en la realidad los Pseudocódigos digitales no son tan sencillos; muchos de ellos necesitaran tomar decisiones complejas y ejecutar diferentes operaciones durante el proceso, o ejecutar en repetidas ocasiones un determinado número de instrucciones. Todo ello de acuerdo a una serie de condiciones específicas marcadas por el programador.

En este sentido, Javier Eguíluz a través de su obra Introducción a JavaScript menciona que:

> Para realizar este tipo de programas son necesarias las **estructuras de control de flujo**, que son instrucciones del tipo *"si se cumple esta condición, hazlo; si no se cumple, haz esto otro"*. También existen instrucciones del tipo *"repite esto mientras se cumpla esta condición"* (Eguíluz 27).

Es decir, al implementar las estructuras de control en Pseudocódigos digitales, estos dejan de ser una sucesión lineal de instrucciones, puesto que a través de ellas es posible modificar el flujo de su ejecución (esto es, el orden en que se ejecuta el código) y presentarlos ante los usuarios con un comportamiento más o menos inteligente, debido a que pueden realizar diferentes acciones de acuerdo a los resultados obtenidos tras una serie de comparaciones lógicas. A continuación, los diferentes tipos de estructuras de control.

2.1.1.1 Estructuras de Selección

En determinados momentos, los Pseudocódigos digitales "… requieren ser selectivos en lo que respecta a las acciones que deben seguir, basándose en una respuesta de un determinado cuestionamiento que se formuló para la solución del problema planteado" (Pinales y Velázquez 49). Por tal motivo, PSeInt dispone de un conjunto de estructuras de selección que pone a disposición del programador para modificar el comportamiento de los Pseudocódigos digitales, mismas que tras analizar el resultado devuelto por una condición, permiten que se ejecute un conjunto de acciones alternativas como respuesta al problema.

Bajo este mismo orden de ideas, las estructuras de selección se pueden clasificar en:

- Si Entonces
- Si Entonces Sino
- Si Entonces Sino (Anidado)
- Según (Selección Múltiple)

Ahora bien, la presencia y disponibilidad de cada una de estas clasificaciones en los diferentes lenguajes de programación reales frecuentemente queda sujeta a una serie de decisiones que pueden venir directamente de parte de su creador, empresa, o grupo de colaboradores. Sin embargo, en la mayoría de las ocasiones los lenguajes de programación toman como base la clasificación anterior para adaptarla a su sintaxis correspondiente, y en algunos casos, añaden ciertas mejoras que les permiten destacar ante los demás (lenguajes de programación), lo

que genera un impacto favorable en su comunidad de desarrolladores, puesto que brindan una mayor flexibilidad a la hora de programar.

2.1.1.1.1 Si Entonces



Imagen 47: Diagrama de Flujo Estructura de Selección "Si Entonces" - Fuente: Diseñada por el autor.

Desde la perspectiva de Javier Eguíluz "la estructura más utilizada en [PSeInt] y en la mayoría de lenguajes de programación es la estructura [Si-Entonces]. Se emplea para tomar decisiones en función de una condición …" (Eguíluz 27). Es decir, al implementar este tipo de estructura selectiva como parte de la solución a un problema, se evalúa una determinada condición y, en caso de ser verdadera, se ejecuta un conjunto de instrucciones (operaciones) alternativas escritas en el Pseudocódigo digital. Si dicha condición no se cumple (es decir, el resultado de la evaluación es falso), ninguna de las instrucciones alternativas es ejecutada. Finalmente, la secuencia de instrucciones continúa con normalidad.

De la imagen anterior se tiene que:

- **Condición:** Expresa la condición o conjunto de condiciones que se deben evaluar. Estas expresiones deben retornar como respuesta un resultado lógico, es por ello que frecuentemente se les ve acompañadas de operadores de comparación y operadores lógicos.
- **Operación:** Representa la operación o conjunto de operaciones que se deben realizar (ejecutar) si la condición resulta ser verdadera.

A nivel de Pseudocódigo digital, la estructura de selección "**Si Entonces**" queda representada por las siguientes líneas de código:

```
Si condición Entonces
    instrucciones
FinSi
```

Imagen 48: Estructura de Selección "Si Entonces" - Fuente: Diseñada por el autor.

La estructura anterior comienza por la palabra reservada **Si**, seguida de la **condición** (o conjunto de condiciones) que se debe(n) comprobar para la ejecución del bloque de instrucciones, y el bloque propiamente dicho (**instrucciones**) escrito entre las palabras reservadas **Entonces** y **FinSi,** siendo esta última, la que marca el final a dicha estructura de selección.



¿Todo parece confuso?

Otra forma de comprender cómo funciona internamente la estructura de selección **Si Entonces**, es a través del siguiente enunciado:

"**Si** la **condición** es verdadera, **entonces** ejecuta las instrucciones declaradas en este bloque"

Con la finalidad de reforzar los conceptos abordados hasta el momento, el siguiente Pseudocódigo digital pide al usuario el ingreso de dos calificaciones parciales correspondientes a un alumno, posteriormente obtiene el promedio general y comprueba si su valor es superior (mayor que) a 9.5. En caso de cumplirse esta condición, el Pseudocódigo digital imprime alternativamente la cadena de caracteres "Informe: Beca del

50%". Finalmente, muestra por pantalla el promedio general de dicho alumno.

```
1   Proceso condicionalSiEntonces
2       //Datos de Entrada
3       Imprimir "Ingrese calificación 1° Parcial"
4       Leer parcialA
5       Imprimir "Ingrese calificación 2° Parcial"
6       Leer parcialB
7       //Obteniendo el promedio general
8       promedio = (parcialA + parcialB) / 2
9       //Comprueba si el alumno tiene derecho a beca,
10      //en este caso, el promedio debe ser mayor a 9.5
11      Si promedio > 9.5 Entonces
12          Imprimir "Informe: Beca del 50%"
13      FinSi
14      //Salida de Información
15      Imprimir "Promedio General: " , promedio
16  FinProceso
```

Imagen 49: Estructura de selección "Si Entonces" implementada en un Pseudocódigo digital escrito en PSeInt - Fuente: Diseñada por el autor.

En el ejemplo anterior, el bloque de instrucciones que se debe ejecutar si la condición se cumple (línea de código 11) está formado por una sola línea de código (línea 12). Sin embargo, en algunas ocasiones el número de líneas pertenecientes a este bloque "**Si Entonces**" puede crecer considerablemente (según la problemática a resolver), por lo que la sangría[13] declarada dentro de dicho bloque, juega un papel fundamental para la lectura y comprensión del código correspondiente.

Ahora bien, una vez que se ejecuta dicho Pseudocódigo digital (**Tecla F9**) y se comienza a ingresar las calificaciones parciales correspondientes, el comportamiento de sus salidas de información cambia con base al promedio general calculado. Tal y como se ilustra en la siguiente imagen:

[13] En informática, este tipo de técnica o buena práctica en el código de programación es conocida como identación. Sin embargo, al ser un anglicismo de la palabra inglesa *identation*, esta investigación emplea el término "sangría" para referirse a los mismos efectos de uso.

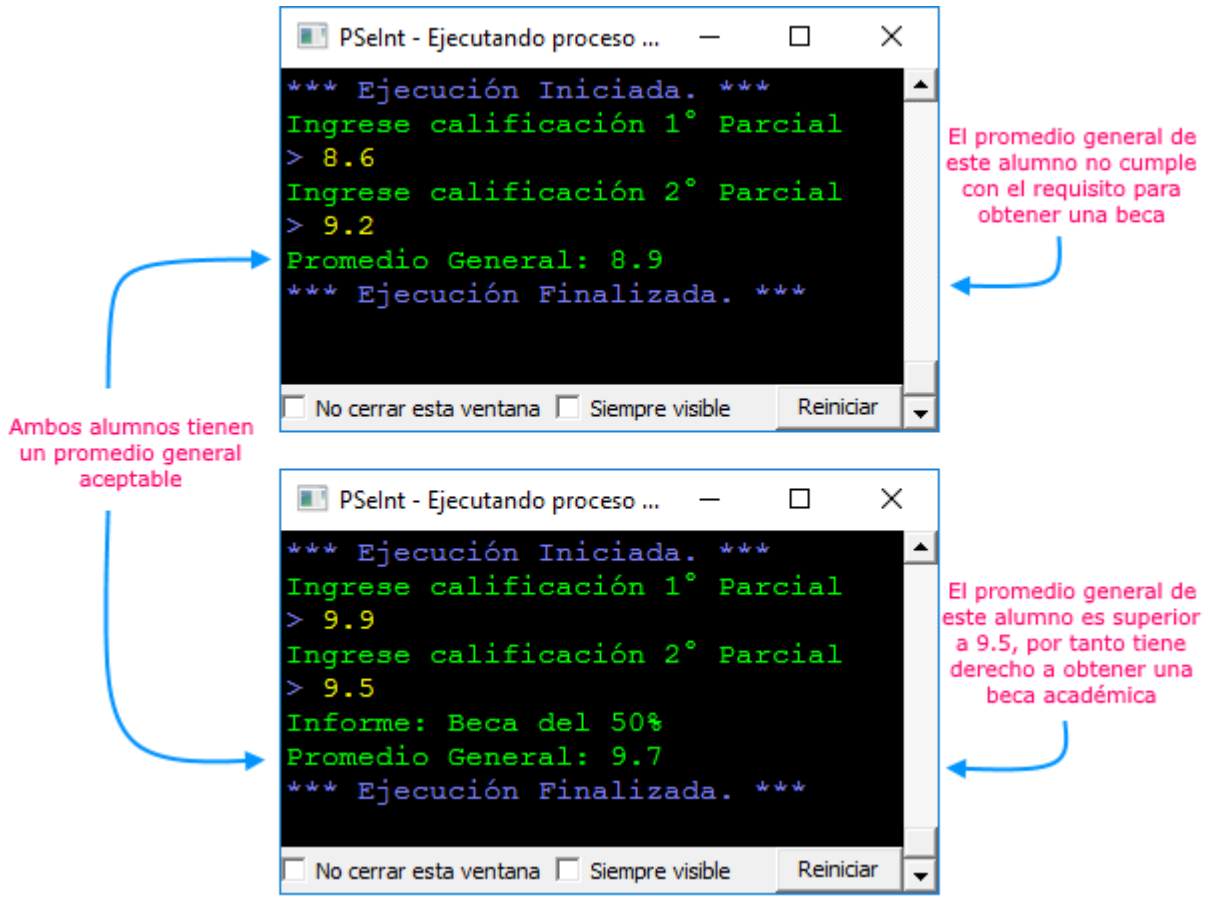


Imagen 50: Ejecución Pseudocódigo digital estructura de selección "Si Entonces" - Fuente: Diseñada por el autor.

Llegado a este punto, es importante mencionar que la condición que se declara como parte de una estructura de selección "**Si Entonces**" puede estar formulada a partir de una o más condiciones, lo que provoca que su construcción se torne cada vez más compleja debido a la intervención de diferentes operadores lógicos y de comparación. Sin embargo, ante esta situación siempre se debe tener en cuenta que el resultado que retorna cada una de estas condiciones viene representado por un valor de tipo lógico, es decir, **FALSO** o **VERDADERO**.

Ver ejercicio – Anexo C1



¿Por qué utilizar la estructura de selección Si Entonces?

La estructura de selección **Si Entonces** permite desviar el flujo de ejecución de un Pseudocódigo digital hacia una instrucción o conjunto de instrucciones dependiendo del cumplimiento de alguna condición.

2.1.1.1.2 Si Entonces Sino

Según Miguel Angel Rodríguez Almeida en su obra Metodología de la Programación a través de Pseudocódigo, "a menudo necesitamos realizar dos procesos completamente distintos, dependiendo de si cumple o no la/s condición/es de entrada [de la estructura Si Entonces]" (Rodríguez 31). En este sentido, la estructura de selección "**Si Entonces Sino**" evalúa una determinada condición (simple o compleja) y, en caso de ser verdadera, ejecuta un conjunto de acciones identificadas como A. En caso contrario, es decir, cuando la condición se evalúa como falsa, ejecuta otro conjunto de acciones alternativas identificadas como B. Finalmente, el flujo del Pseudocódigo digital continúa de manera secuencial hasta terminar.

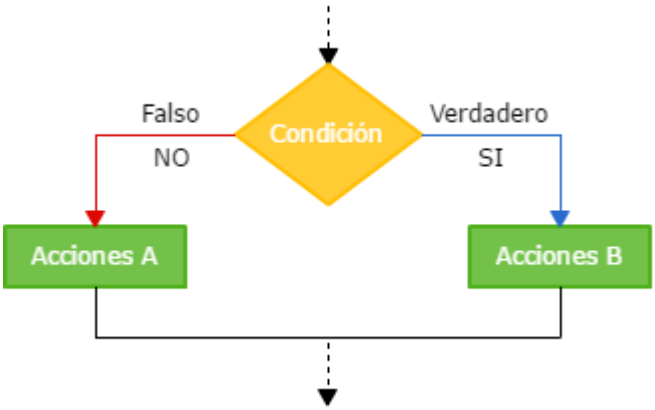


Imagen 51: Diagrama de Flujo Estructura de Selección "Si Entonces Sino" - Fuente: Diseñada por el autor.

Bajo este mismo orden de ideas, al interpretar el diagrama de flujo anterior se tiene que:

- **Condición:** Expresa la condición o conjunto de condiciones a evaluar para decidir qué grupo de acciones se deben ejecutar.
- **Acciones A:** Representa las acciones que se deben ejecutar si la condición resulta ser verdadera.
- **Acciones B:** Indica el conjunto de acciones que se deben ejecutar si la condición resulta ser falsa.

Con base en los supuestos anteriores, la sintaxis que se utiliza para representar la estructura de selección "**Si Entonces Sino**" a nivel de Pseudológico digital queda determinada por la sentencia:

```
Si condición Entonces
    instrucciones caso verdadero
Sino
    instrucciones caso falso
FinSi
```

Imagen 52: Estructura de Selección "Si Entonces Sino" - Fuente: Diseñada por el autor.

Como se puede observar, gran parte de la estructura anterior es idéntica en cuanto a declaración y funcionamiento a la estructura de selección "**Si Entonces**", solo que, en este caso particular, se agrega un bloque adicional **Sino** con instrucciones que deben ejecutarse en caso de que el resultado de la condición sea evaluado como **FALSO**. En este sentido, se tiene que el flujo de un Pseudocódigo digital sólo puede tomar "uno y sólo uno" de dos caminos disponibles. Además, con cada nueva ejecución, este retorna información distinta al usuario (un resultado catalogado como A u

otro resultado catalogado como B), a consecuencia de las comprobaciones internamente realizadas.



¿Todo parece confuso?

Otra forma de razonar el comportamiento interno de la estructura de selección **Si Entonces Sino**, es a través del siguiente enunciado:

"**Si** la **condición** es verdadera, **entonces** ejecuta éste bloque de instrucciones; **de lo contrario**, ejecuta las siguientes instrucciones alternativas"

Con la finalidad de ejemplificar lo expuesto hasta el momento, el siguiente Pseudocódigo digital tiene como finalidad determinar si un usuario "tiene o no" la mayoría de edad a partir de dicho dato proporcionado:

```
1  Proceso condicionalSiEntoncesSino
2      //Datos de entrada:
3      //Solicitar la edad del usuario.
4      Imprimir "Ingrese su edad"
5      Leer edadUsuario
6      //Verificar si el usuario es menor o
7      //mayor de edad.
8      Si edadUsuario < 18 Entonces
9          //Si la edad del usuario es menor
10         //a 18 años entonces es menor de
11         //edad.
12         Imprimir "Informe: Usted es menor de edad"
13     Sino
14         //En caso contrario, el usuario
15         //tiene la mayoría de edad
16         Imprimir "Informe: Usted es mayor de edad"
17     Fin Si
18 FinProceso
```

Imagen 53: Estructura de selección "Si Entonces Sino" implementada en un Pseudocódigo digital escrito en PSeInt - Fuente: Diseñada por el autor.

Una vez que se ha analizado detalladamente el ejemplo anterior, se tiene que las líneas 4-5 son las encargadas de solicitar los datos de entrada al usuario, es decir, su edad. Posteriormente, las líneas de código 8-17 representan la estructura de selección "**Si Entonces Sino**", misma que está compuesta por una condición simple dónde se compara que el valor referenciado por la variable **edadUsuario** sea menor a 18. En caso de cumplirse dicha condición (caso verdadero), se ejecuta la instrucción declarada en la línea 12; de lo contario (**Sino**), es la línea de código número 16 la que termina por ejecutarse (caso falso).

Una vez que se ha terminado de evaluar la condición y, por consiguiente, se ha ejecutado una de las dos acciones posibles declaradas en la estructura de selección, el flujo del Pseudocódigo digital continúa con normalidad hasta finalizar los procedimientos correspondientes.

Llegado a este punto de la explicación, una vez que se ejecuta dicho Pseudocódigo digital (**Tecla F9**) y se ingresa la edad correspondiente a un usuario; según el valor de ésta, internamente se genera uno de dos posibles resultados para ser mostrados en pantalla. Tal y como se ilustra en la siguiente imagen:

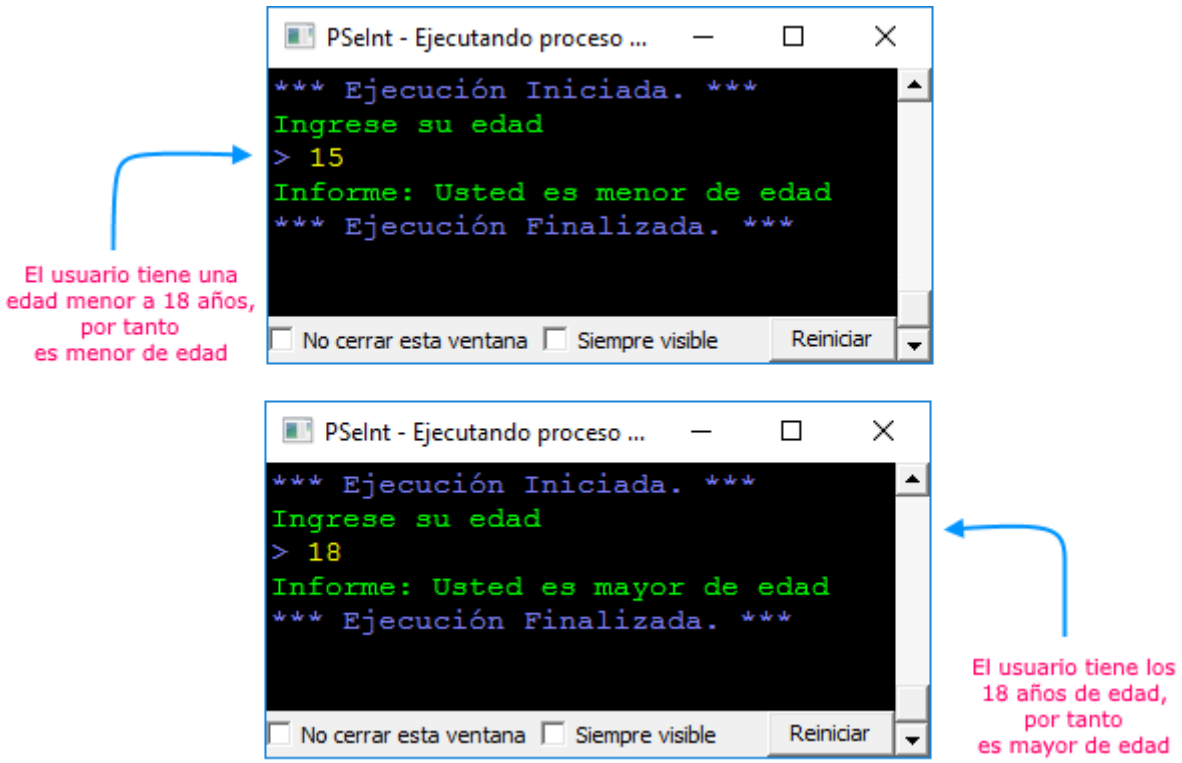


Imagen 54: Ejecución Pseudocódigo digital estructura de selección "Si Entonces Sino" - Fuente: Diseñada por el autor.

Finalmente, es importante señalar que, si bien en la temática del ejemplo anterior no aparecen instrucciones declaradas inmediatamente después de la estructura de selección "**Si Entonces Sino**", es posible que en algunas ocasiones exista la necesidad de agregar líneas de código adicionales con la finalidad de encontrar una solución factible al problema.

Ver ejercicio – Anexo C2



¿Por qué utilizar la estructura de selección Si Entonces Sino?

La estructura de selección **Si Entonces Sino**, como su nombre bien lo indica, permite dividir el flujo de ejecución un Pseudocódigo digital en dos caminos distintos. El camino que se tome dependerá de una condición o conjunto de condiciones, donde es posible declarar (internamente) una o más instrucciones.

2.1.1.1.3 Si Entonces Sino Anidado

Con base en los diferentes aportes que hace Miguel Angel Rodríguez Almeida acerca de la toma de decisiones durante la fase de desarrollo de un Pseudocódigo digital, el autor establece que:

> En el formato general para la sentencia [Si Entonces Sino], las instrucciones [del caso verdadero y caso falso] no están limitadas a ser instrucciones imperativas; pueden ser expresiones condicionales y surge la posibilidad de usar instrucciones [Si Entonces Sino anidadas] (Rodríguez 32).

En este sentido, la estructura de selección "**Si Entonces Sino Anidado**" permite resolver cualquier situación en la que se necesite evaluar más de una condición lógica y por consiguiente ejecutar una de varias acciones disponibles.

Para explicar el comportamiento que tiene a lugar este tipo de estructura selectiva dentro de un Pseudocódigo digital, la siguiente figura ilustra su correspondiente diagrama de flujo. Obsérvese con detenimiento que en lugar de ejecutar dos acciones diferentes para cada uno de los casos "**VERDADERO** / **FALSO**", es posible incluir una segunda estructura de

selección "**Si Entonces Sino**", de manera que sea posible ejecutar un máximo de tres acciones posibles.

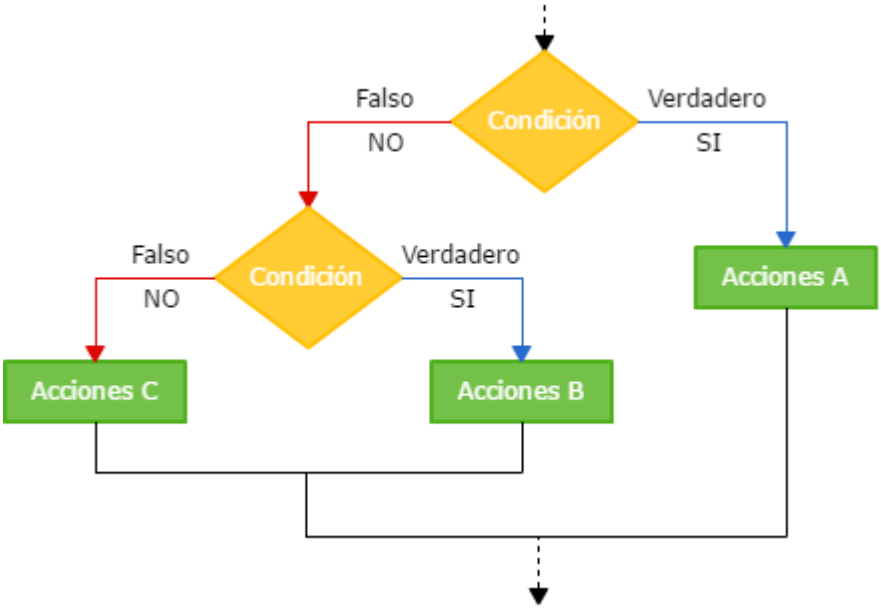


Imagen 55: Diagrama de Flujo Estructura de Selección "Si Entonces Sino Anidado" - Fuente: Diseñada por el autor.

Como se puede observar en el diagrama anterior, este tipo de estructura selectiva aumenta la flexibilidad y función de la estructura de selección "**Si Entonces Sino**", ya que permite aumentar el número de posibles condiciones a evaluar, así como el número de acciones que se pueden ejecutar.



¿Todo parece confuso?

Otra forma de analizar el comportamiento interno de la estructura de selección **Si Entonces Sino Anidado**, es a través del siguiente enunciado:

"**Si** la **condición** es verdadera, **entonces** ejecuta éste bloque de instrucciones; **de lo contrario**, evalúa la siguiente condición..."

A manera de ejemplo ilustrativo, el siguiente Pseudocódigo digital realiza una comprobación a partir de la edad proporcionada por un usuario. Finalmente, muestra por pantalla una de las siguientes leyendas acompañadas de su correspondiente nombre:

- "**Menor de edad**", si la edad proporcionada es menor a 18 años.
- "**Mayor de edad**", si la edad se encuentra entre los 18 y 69 años.
- "**Tercera edad**", si la edad es mayor o igual a los 70 años.

```
1  Proceso estructuraSiEntoncesAnidado
2      //------ Datos de Entrada --------
3      Imprimir "Ingrese su nombre"
4      Leer nombreUsuario
5      Imprimir "Ingrese su edad"
6      Leer edad
7      //-------- Proceso de Información ----------
8      Si edad < 18 Entonces
9          //si el usuario tiene menos de 18 años
10         //entonces es...
11         informe = "Menor de edad"
12     Sino
13         Si edad >= 18 Y edad < 70 Entonces
14             //si la edad del usuario esta entre
15             //18 y los 69 años, entonces es...
16             informe = "Mayor de edad"
17         Sino
18             //si la edad del usuario es de 70 años
19             //en adelante, entonces ya es parte de la...
20             informe = "Tercera edad"
21         FinSi
22     FinSi
23     //---------- Salida de Información ----------
24     Imprimir "NOMBRE DEL USUARIO: " , nombreUsuario
25     Imprimir "INFORME: " , informe
26 FinProceso
```

Imagen 56: Estructura de selección "Si Entonces Sino Anidado" implementada en un Pseudocódigo digital escrito en PSeInt - Fuente: Diseñada por el autor.

En el ejemplo anterior, la primera estructura de selección "**Si Entonces Sino**" (líneas 8-22) verifica que la edad proporcionada por el usuario sea

menor a 18 años, de ser así, vincula la cadena de texto "Menor de edad" hacia la variable **informe**. En caso contrario, se ejecuta una segunda estructura de selección "**Si Entonces Sino**" (líneas 13-21) misma que se encarga de comprobar si la edad del usuario oscila entre los 18 y 69 años, de cumplirse esta condición, se genera una cadena de texto con la leyenda "Mayor de edad" y se hace referencia a ella mediante el uso de la variable **informe**; en caso contrario, es decir, si ninguna de las dos condiciones anteriores se cumple, se termina por asignar el valor "Tercera edad" en la variable antes mencionada.

Bajo este mismo orden de ideas, una vez que se ejecuta dicho Pseudocódigo digital (**Tecla F9**) y se procede a ingresar el nombre del usuario y su correspondiente edad; según el valor de ésta última, internamente se genera uno de tres posibles resultados para ser mostrados en pantalla. Tal y como se ilustra en la siguiente imagen:

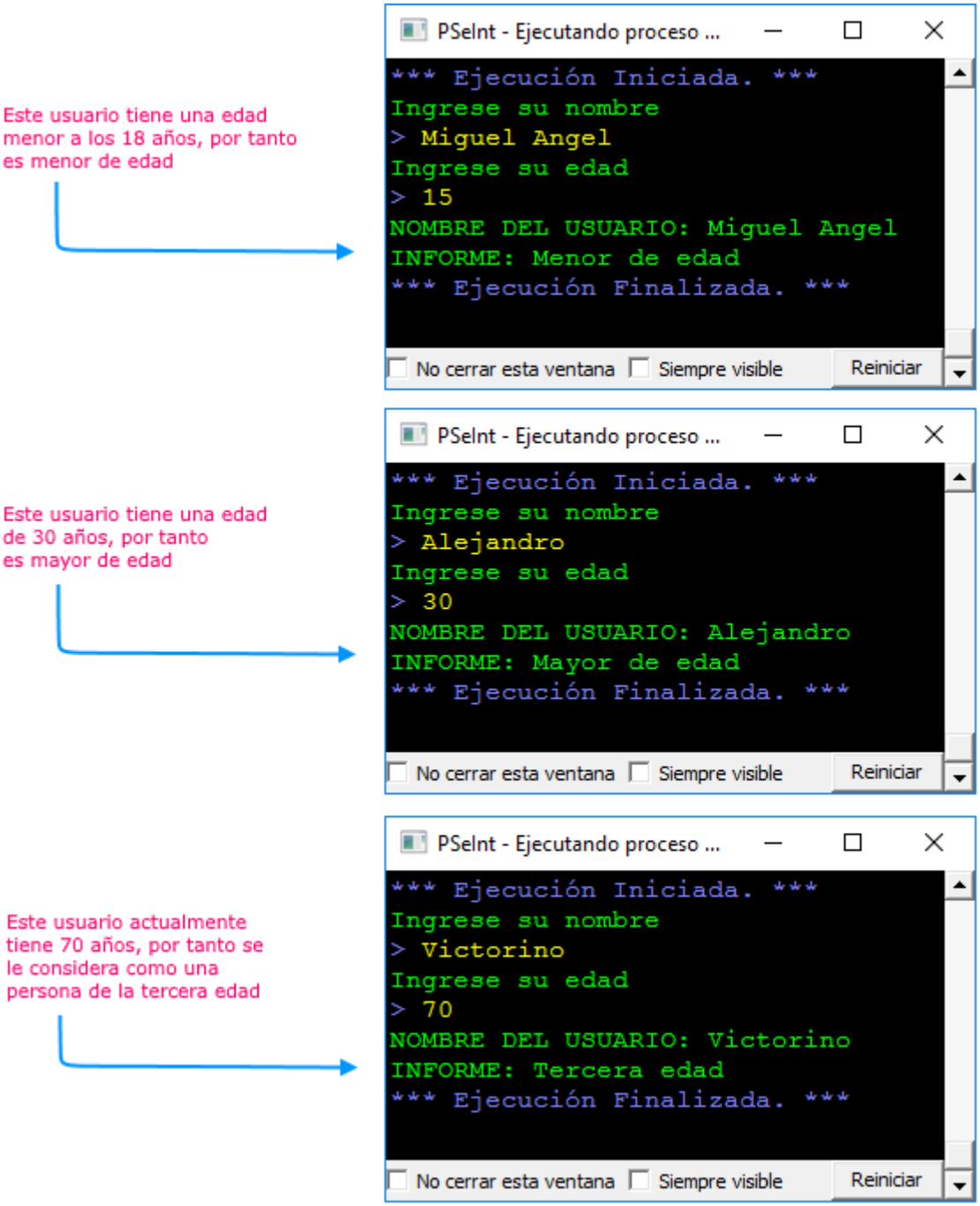


Imagen 57: Ejecución Pseudocódigo digital estructura de selección "Si Entonces Sino Anidado" - Fuente: Diseñada por el autor.

Con base en los supuestos anteriores, se ha podido comprobar que la estructura de selección "**Si Entonces Sino Anidado**" ofrece una mayor flexibilidad al momento de evaluar más de una condición lógica durante el

proceso de solución de un problema. Sin embargo, es importante señalar que, el abusar de este tipo de estructuras puede conducir a encadenamientos (anidaciones) difíciles de mantener en el código (véase el ejercicio del Anexo C3), por lo que tiene sentido utilizar otro tipo de estructura selectiva que permita llevar a cabo este tipo de actividades de una manera más prolija.

Ver ejercicio – Anexo C3



¿Por qué utilizar la estructura de selección Si Entonces Sino Anidado?

La estructura de selección **Si Entonces Sino Anidado**, como su nombre bien lo indica, aumenta la flexibilidad y función de la estructura de selección "Si Entonces Sino", puesto que permite incrementar el número de posibles condiciones a evaluar, así como la cantidad de acciones diferentes que se pueden ejecutar.

2.1.1.1.4 Según (Selección Múltiple)

Según Juan Carlos Casale en su obra Introducción a la Programación, establece que "otra de las estructuras de comparación múltiple [disponibles, consiste en] una decisión especializada que nos [permite] evaluar una variable con distintos resultados posibles, ejecutando para cada caso una serie de instrucciones específicas …" (Casale 101). En este sentido, la estructura de selección "**Segun**" permite la ejecución de un bloque de instrucciones en función del valor que tome una determinada expresión. Tal y como se ilustra en la siguiente imagen:

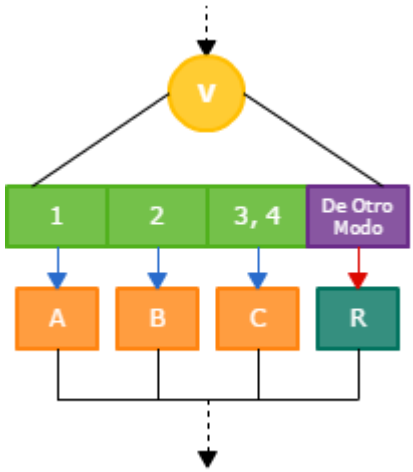


Imagen 58: Diagrama de Flujo Estructura de Selección Múltiple "Segun" - Fuente: Diseñada por el autor.

Para el caso de la imagen anterior, la expresión viene determinada por la variable "**V**", si el valor que contiene (resultado) coincide con alguna de las opciones declaradas "**1**, **2**, **3** o **4**", comienza la ejecución del bloque correspondiente "**A**, **B** o **C**". En caso de que el valor no coincida con alguna de las opciones disponibles, se ejecuta el bloque de instrucciones "**R**" especificado en la opción por defecto (**de otro modo**).

La expresión que se evalúa al inicio de la declaración de la estructura de selección "**Segun**" puede ser cualquiera que dé como resultado un valor de tipo: numérico, carácter o cadena de caracteres. En este sentido, se puede decir que este tipo de estructura selectiva no se encuentra limitada sólo a una de dos opciones disponibles (**FALSO** / **VERDADERO**), sino que, por el contrario, su trabajo radica en verificar el valor que retorna una determinada expresión y lo coteja a partir de una serie de opciones disponibles (de ahí que se le conozca como una estructura de selección múltiple).

Bajo este mismo orden de ideas, la sintaxis que guarda la estructura de selección "**Segun**" a nivel de Pseudocódigo digital, queda representada por:

```
Segun expresión Hacer
    opción1:
        instrucciones_opción1
    opción2:
        instrucciones_opción2
    opciónN:
        instrucciones_opciónN
    De Otro Modo:
        instrucciones_por_defecto
FinSegun
```

Imagen 59: Estructura de Selección Múltiple "Segun" - Fuente: Diseñada por el autor.

Como se puede observar, dicha sintaxis no difiere en mucho del diagrama de flujo presentado con anterioridad. Todo comienza con la declaración de la palabra reservada **Segun** seguida de una **expresión** a evaluar (en este caso puede ser una operación matemática, o un valor de tipo numérico o carácter referenciado por una variable). Posteriormente, se coteja el valor retornado por la expresión con cada una de las opciones disponibles (**opción1**, **opción2, opciónN**) en la estructura. En caso de que el valor coincida con alguna de estas opciones, se procede a ejecutar el bloque de instrucciones declaradas en dicha opción; **de otro modo**, se termina por ejecutar las instrucciones declaradas en el bloque por defecto.

Sumado a lo expuesto, es importante enfatizar que la declaración de este último bloque (**de otro modo**) es opcional y no puede ser considerado

como un error de programación si este se encuentra ausente, sin embargo, se recomienda su declaración, puesto que en la mayoría de las ocasiones es importante contar con un plan de acción alternativo (sobre todo en programación) que informe al usuario acerca de lo sucedido durante la fase de procesamiento.

Con la finalidad de ejemplificar las ideas expuestas hasta el momento, el siguiente Pseudocódigo digital muestra al usuario un menú de selección para que elija ejecutar uno de tres procesos disponibles: un convertidor de medidas, una calculadora básica, y un programa de geometría para el cálculo de área y perímetro de una figura cuadrática.

```
1   Proceso programaTodoEnUno
2       Imprimir "PROGRAMA TODO EN UNO - AGSISTEMAS"
3       Imprimir "A) Convertidor de medidas"
4       Imprimir "B) Calculadora básica"
5       Imprimir "C) Programa geometría (cuadrado)"
6       Leer opcionUsuario
7       //Verificando que tipo de proceso se desea ejecutar
8       Segun opcionUsuario Hacer
9           "A":
10              Imprimir "Ingrese cantidad en metros"
11              Leer metros
12              Imprimir "Centímetros: " , (metros * 100)
13              Imprimir "Kilómetros: " , (metros / 1000)
14              Imprimir "Pulgadas: ", (metros * 100) / 2.54
15          "B":
16              Imprimir "Ingrese primer operando"
17              Leer numA
18              Imprimir "Ingrese segundo operando"
19              Leer numB
20              Imprimir "Suma: " , (numA + numB)
21              Imprimir "Resta: " , (numA - numB)
22              Imprimir "Multiplicación: " , (numA * numB)
23              Imprimir "División: " , (numA / numB)
24          "C":
25              Imprimir "Ingrese distancia lado A"
26              Leer distanciaA
27              Imprimir "Perímetro: " , (distanciaA * 4)
28              Imprimir "Área: " , (distanciaA ^ 2) //lado x lado
29          De Otro Modo:
30              Imprimir "Opción incorrecta..."
31      FinSegun
32  FinProceso
```

Imagen 60: Estructura de selección "Segun" implementada en un Pseudocódigo digital escrito en PSeInt - Fuente: Diseñada por el autor.

En el ejemplo anterior, las líneas de código 2-5 muestran por pantalla un menú de navegación al usuario, el cual consiste de tres opciones disponibles mismas que pueden ser activadas mediante el ingreso de los caracteres **A**, **B** o **C** del teclado (línea 6).

Ahora bien, para verificar que tipo de proceso se debe ejecutar (tres opciones disponibles) se hace uso de la estructura de selección múltiple "**Segun**", dónde la línea de código número 8 verifica el valor referenciado por la variable **opcionUsuario** y comienza a cotejarlo con los casos

declarados en la estructura, es decir, comprueba si el valor coincide con las opciones **A**, **B** o **C** respectivamente.

Llegado a este punto, es importante observar con detenimiento las líneas de código 9, 15 y 24. Al tratarse de una estructura de selección múltiple que incorpora opciones de tipo carácter, estos deben delimitarse mediante el uso de comillas simples o dobles. Además, la declaración del bloque de instrucciones correspondientes a una opción puede ser tan extensa como se desee, no existen límites, lo importante es encontrar una solución factible al problema.

Finalmente, si la opción seleccionada por el usuario no coincide con alguno de los casos disponibles (**A**, **B** o **C**) en dicha estructura selectiva. Se procede a ejecutar el bloque de instrucciones declaradas en la opción por defecto (si es que está disponible) y, por consiguiente, el flujo del Pseudocódigo digital continúa con normalidad hasta finalizar los procedimientos correspondientes.

Bajo este mismo orden de ideas, una vez que se ejecuta dicho Pseudocódigo digital (**Tecla F9**) y se procede a seleccionar una opción con base en su menú; según el valor seleccionado por el usuario, internamente se ejecuta uno de cuatro casos disponibles. Tal y como se ilustra en la siguiente imagen:

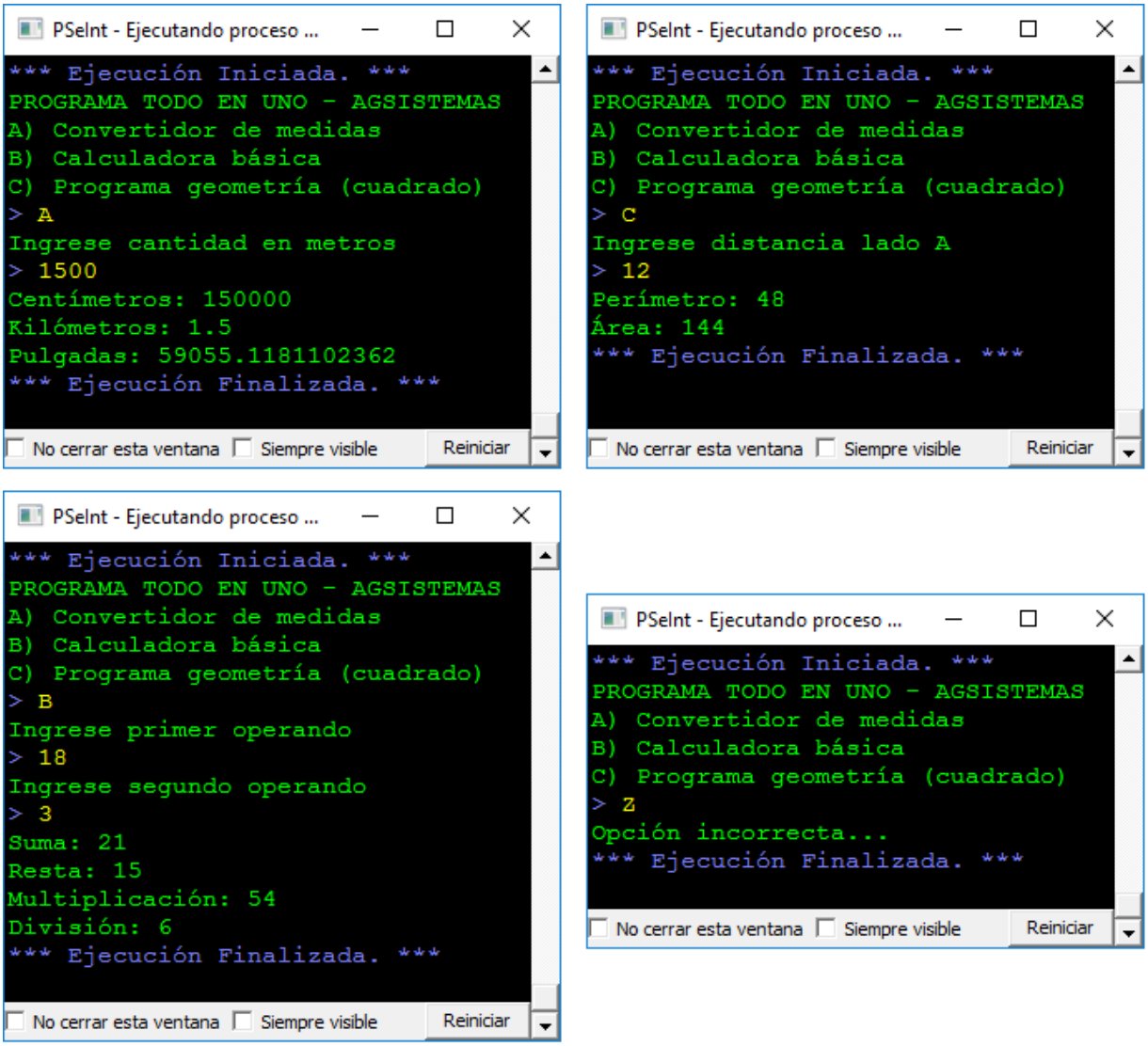


Imagen 61: Ejecución Pseudocódigo digital estructura de selección "Segun" - Fuente: Diseñada por el autor.

Ahora bien, ¿Qué pasa si como respuesta al menú de opciones se ingresa un carácter expresado en letra minúscula?, ¿acaso el Pseudocódigo digital funcionará correctamente?, véase este comportamiento con la ayuda de la siguiente imagen:

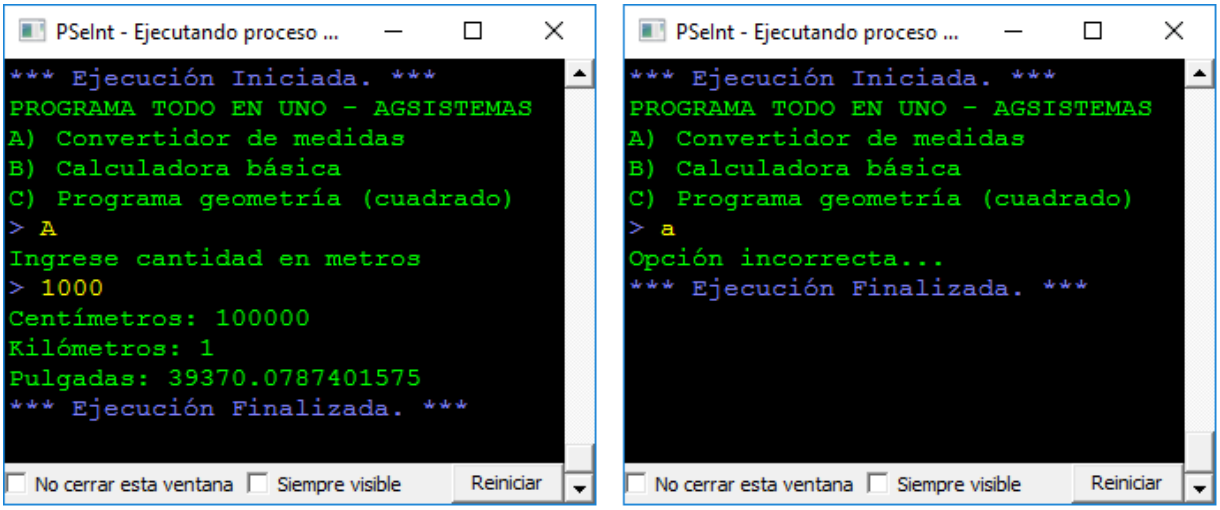


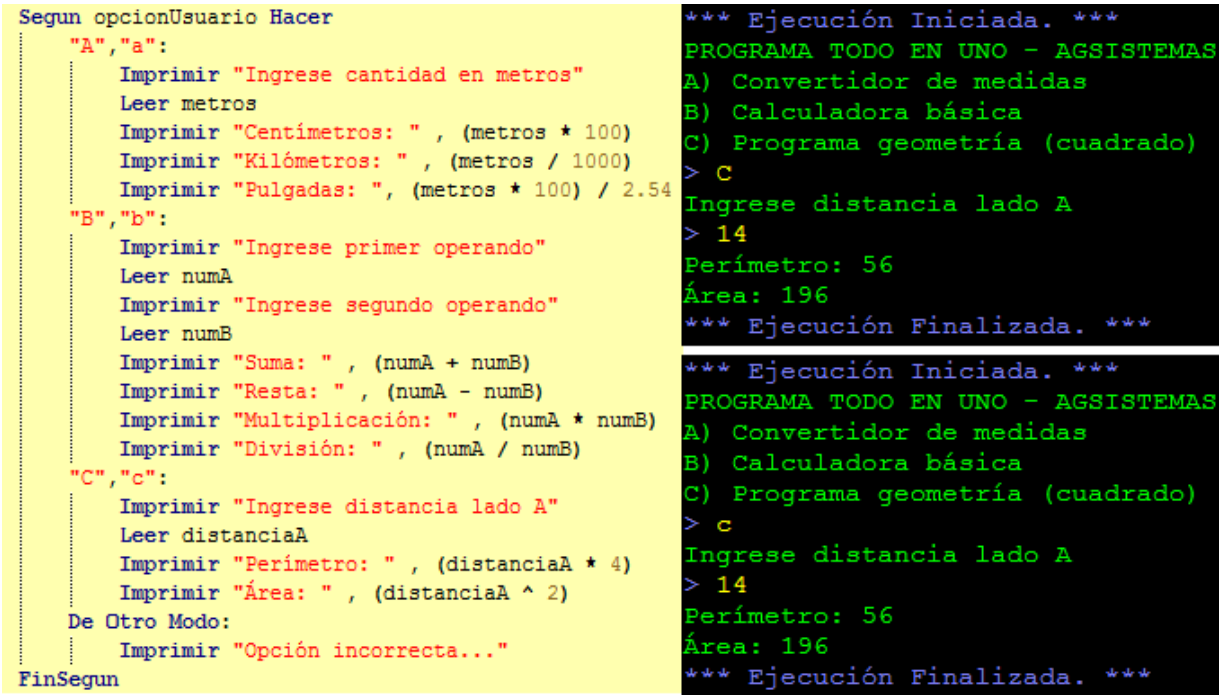
Imagen 62: La comparación que realiza una estructura de selección "Segun", es sensible a mayúsculas y minúsculas - Fuente: Diseñada por el autor.

Al parecer el Pseudocódigo digital declarado con anterioridad se comporta exigente con el usuario y le orilla a ingresar los datos sólo con letra mayúscula para las opciones del menú. Para solucionar este tipo de inconvenientes, la estructura de selección múltiple permite anidar dos o más opciones para la ejecución de una misma tarea, es decir, para el caso que ocupa este ejemplo, se podría indicar que la opción marcada con el valor "**A**" o la opción marcada con el valor "**a**", ambas permitieran que se ejecute la tarea del convertidor de medidas, tal y como se muestra en la siguiente figura:

```
8     Segun opcionUsuario Hacer
9         "A","a":
10            Imprimir "Ingrese cantidad en metros"
11            Leer metros
12            Imprimir "Centímetros: " , (metros * 100)
13            Imprimir "Kilómetros: " , (metros / 1000)
14            Imprimir "Pulgadas: ", (metros * 100) / 2.54
```

Imagen 63: Una opción declarada en una estructura de selección "Según", puede comparar más de un valor - Fuente: Diseñada por el autor.

Como se puede observar en la imagen anterior (línea 9), el signo coma (**,**) es el responsable de que dos o más opciones se aniden ("**A**", "**a**") y, por consiguiente, ejecuten el mismo bloque de instrucciones declarado para dicha selección (convertidor de medidas). En este sentido, tras realizar las actualizaciones correspondientes en el código anterior, la siguiente figura ilustra el nuevo comportamiento del Pseudocódigo digital (versión 2.0):



```
Segun opcionUsuario Hacer
    "A","a":
        Imprimir "Ingrese cantidad en metros"
        Leer metros
        Imprimir "Centímetros: " , (metros * 100)
        Imprimir "Kilómetros: " , (metros / 1000)
        Imprimir "Pulgadas: ", (metros * 100) / 2.54
    "B","b":
        Imprimir "Ingrese primer operando"
        Leer numA
        Imprimir "Ingrese segundo operando"
        Leer numB
        Imprimir "Suma: " , (numA + numB)
        Imprimir "Resta: " , (numA - numB)
        Imprimir "Multiplicación: " , (numA * numB)
        Imprimir "División: " , (numA / numB)
    "C","c":
        Imprimir "Ingrese distancia lado A"
        Leer distanciaA
        Imprimir "Perímetro: " , (distanciaA * 4)
        Imprimir "Área: " , (distanciaA ^ 2)
    De Otro Modo:
        Imprimir "Opción incorrecta..."
FinSegun
```

```
*** Ejecución Iniciada. ***
PROGRAMA TODO EN UNO - AGSISTEMAS
A) Convertidor de medidas
B) Calculadora básica
C) Programa geometría (cuadrado)
> C
Ingrese distancia lado A
> 14
Perímetro: 56
Área: 196
*** Ejecución Finalizada. ***
```

```
*** Ejecución Iniciada. ***
PROGRAMA TODO EN UNO - AGSISTEMAS
A) Convertidor de medidas
B) Calculadora básica
C) Programa geometría (cuadrado)
> c
Ingrese distancia lado A
> 14
Perímetro: 56
Área: 196
*** Ejecución Finalizada. ***
```

Imagen 64: Estructura de selección "Segun" con opciones que comparan más de un valor - Fuente: Diseñada por el autor.

Una vez llegado a este punto, se debería contar con las herramientas suficientes para comenzar a desarrollar Pseudocódigos digitales cada vez más complejos que requieran tomar algún tipo de decisión durante el proceso. Como se ha podido observar, existen diferentes estructuras de selección que pueden utilizarse para la toma de decisiones. La pertinencia de cada una de ellas, es decir, su uso, depende en gran medida de la habilidad que se tenga como programador para comprender el problema,

pero, sobre todo el que se pueda asociarlas e implementarlas durante el proceso de solución de dicho problema.

Sumado a lo expuesto, es importante recordar que no existe un límite para la escritura de código dentro de estas estructuras selectivas, lo primordial es estar convencido de que se ha encontrado una solución factible al problema. Si por algún motivo se presenta la necesidad de anidar o fusionar dos o más estructuras; lo pertinente es proceder a hacerlo y no limitarse. Tal y como se ilustra en el ejercicio publicado en el Anexo C4, algunos problemas requieren de un análisis más detallado, y de acuerdo a su complejidad, es posible que no exista mejor alternativa que el terminar por fusionar diferentes estructuras selectivas.

Ver ejercicio – Anexo C4



¿Por qué utilizar la estructura de selección Segun?

La estructura de selección **Segun** (también conocida como estructura de selección múltiple) permite evaluar una expresión que puede tomar n cantidad de valores distintos. Según sea el valor que tome la expresión, el flujo de ejecución del Pseudocódigo digital seguirá un camino determinado entre los n posibles especificados.

Los Pseudocódigos digitales que se han desarrollado a lo largo de este apartado incluyen conceptos de programación esenciales y avanzados tales como entradas, salidas, asignaciones, expresiones, operaciones, instrucciones secuenciales y estructuras de selección. Sin embargo, muchos de los problemas con los que se topa un programador en la vida real, requieren que se repita una misma instrucción (con diferentes datos) un determinado número de veces. Por ejemplo, generar la tabla de

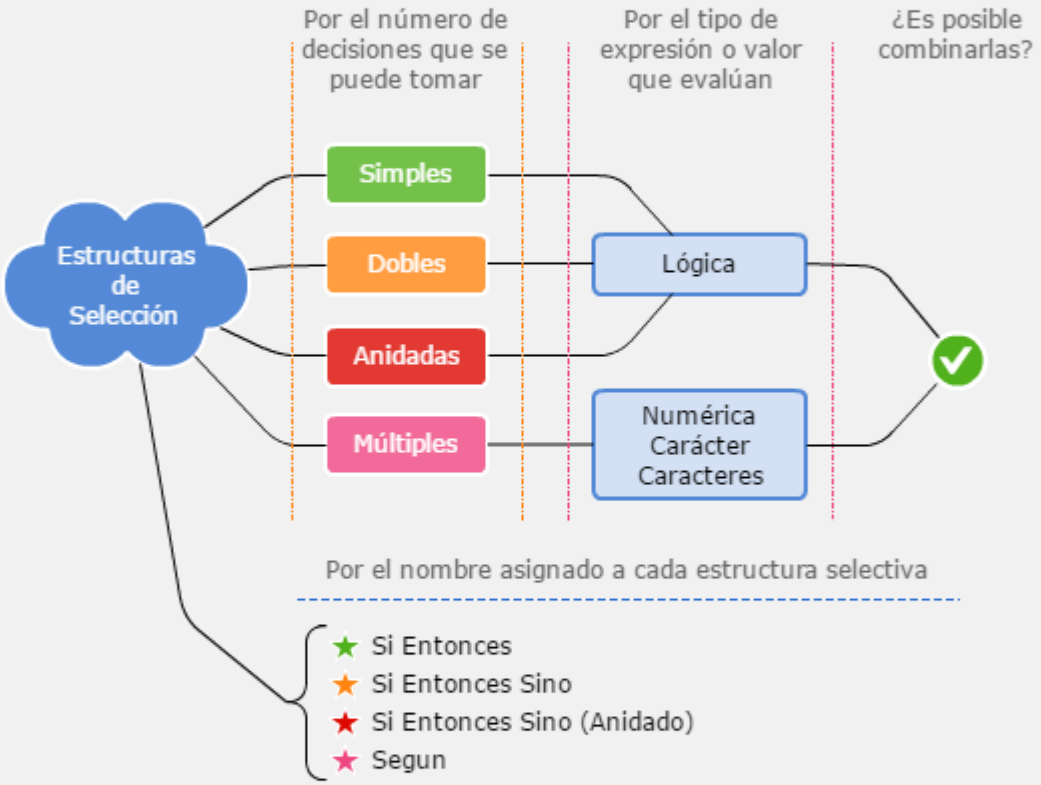
multiplicar de un número, recorrer un conjunto de registros, acumular un conjunto de valores de entrada, etc. En este sentido, al igual que los lenguajes de programación reales, los Pseudo-Intérpretes incluyen estructuras de control especiales que permiten repetir uno o varios procesos "n cantidad de veces", como si se tratasen de ciclos que parten desde un punto de inicio y llegan hasta un punto final. Dichas estructuras reciben el nombre de estructuras de repetición, y serán el foco de atención del siguiente numeral.



Resumen: Estructuras de Selección.

En esta sección se abordaron las diferentes Estructuras de Selección que permiten a un Pseudocódigo digital tomar decisiones y ejecutar un determinado conjunto de instrucciones de acuerdo al cumplimiento de una o varias condiciones.

Los elementos más importantes que se han comentado hasta el momento se resumen a través del siguiente esquema:



Como se puede observar, la diferencia central que existe entre cada estructura selectiva radica en el número de decisiones que se pueden tomar durante el proceso de solución de un problema. A nivel condicional, la diferencia entre ellas se enfatiza por el tipo de valor o expresión que deben evaluar para llevar a cabo la ejecución de un determinado bloque de instrucciones (rutas alternativas). Además, si por algún motivo las características que ofrecen cada una de estas estructuras es insuficiente para determinar la solución de un problema; es posible combinar su funcionamiento, así como utilizar cada uno de sus resultados generados para producir una solución final.

2.1.1.2 Estructuras de Repetición

En algunas ocasiones es posible que los Pseudocódigos digitales necesiten repetir una serie de instrucciones de manera consecutiva. Si bien estas instrucciones son las mismas en cada paso, "… lo único que varía son los valores de los datos con los que [se] está operando" (Rodríguez 17). Por ejemplo, para mostrar en pantalla los números del 1 al 10, la instrucción que se debe repetir constantemente es "**Imprimir**", sin embargo, el valor numérico mostrado deberá incrementar en cada paso de la repetición. En este sentido, dado que los ordenadores son buenos para hacer trabajos repetitivos, tiene sentido que la mayoría de los lenguajes de programación (y algunos Pseudo-Intérpretes) incluyan estructuras de control para llevar a cabo la ejecución de tareas repetitivas, mismas que pueden clasificarse en:

- **Estructuras de repetición bien definidas:** Permiten establecer el número de veces que debe ejecutarse una determinada tarea.
- **Estructuras de repetición condicionales:** Permiten que se ejecute repetidamente una tarea, siempre y cuando se cumpla una determinada condición lógica.

Bajo este mismo orden de ideas, uno de los grandes inconvenientes al momento de trabajar con estructuras de repetición es que, al ser declaradas de manera incorrecta, estas tienden a romper una de las tres características que deben cumplir los Algoritmos: la de ser finitos (deben finalizar en algún momento). Ante esta situación, Cairó argumenta que:

> "Todo ciclo debe terminar de ejecutarse luego de un número finito de veces, por lo que es necesario en cada iteración del

> mismo, evaluar las condiciones necesarias para decidir si se debe seguir ejecutando o si debe detenerse. En todo ciclo, siempre debe existir una condición de parada o fin de ciclo" (Cairó 107).

Es por ello que antes de realizar cualquier tipo de tarea repetitiva como parte del proceso de desarrollo de un Pseudocódigo digital, manualmente el programador debe especificar cuantas veces se tiene que ejecutar dicha tarea (al precisar un punto de inicio y un punto de parada); o en su defecto, establecer los criterios condicionales (expresiones lógicas) que se deben evaluar para que esta se repita. Ya que, de lo contrario, se corre el riesgo de que el Pseudocódigo digital entre en un ciclo (o bucle) infinito, es decir, ejecute una o varias instrucciones de forma indefinida, puesto que, su condición de parada jamás se cumpliría.

Una vez que se han descrito los pormenores de las estructuras de repetición, es momento de abordar cada una de ellas. Se cita en un primer momento su diagrama de flujo, seguido de su interpretación, declaración de sintaxis y un caso práctico.

2.1.1.2.1 Para

Según los diferentes aportes que hace Joel de la Cruz Villar acerca de las estructuras de repetición en su libro Algoritmos y Diagramas de Flujo aplicados a PHP, establece que:

> Cuando se sabe el número de interacciones que tendrán que ejecutarse [como parte de un proceso], es recomendable usar la estructura [Para], esta estructura no solo ejecuta un número [determinado de] acciones, además cuenta internamente el

> número de interacciones, esto elimina la necesidad de un contador, lo que no sucede con [otro tipo de estructuras de repetición] (De la Cruz 144).

En este sentido, se puede decir que la estructura de repetición "**Para**" es utilizada en aquellos Pseudocódigos digitales en los que se conoce previamente el número de veces que se debe repetir un determinado bloque de instrucciones (véase el diagrama de flujo ilustrado en la imagen 65). A su vez, utiliza un contador que se declara e inicializa como parte de la definición de dicha estructura, mismo que incrementa o decrementa su valor en cada iteración.

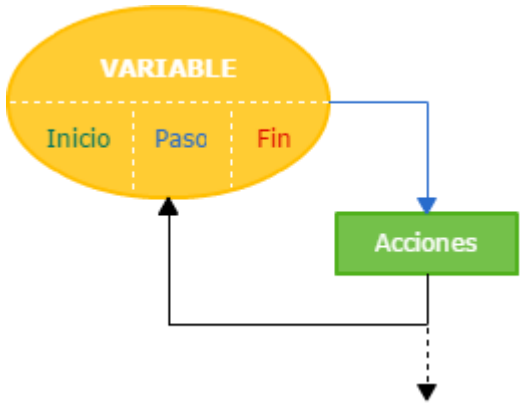


Imagen 65: Diagrama de Flujo Estructura de Repetición "Para" - Fuente: Diseñada por el autor.

De la figura anterior se observa que, al ingresar al bloque de repetición, la **VARIABLE** se le asigna un valor de **Inicio** y comienza a ejecutar las **Acciones** declaradas en el cuerpo del ciclo. Luego incrementa o reduce el valor de la variable en el número de unidades establecidas para cada **Paso**, y antes de ejecutar la siguiente repetición, comprueba si el valor actual de la **VARIABLE** es menor o igual al valor **Final** establecido. En caso de ser **VERDADERO**, se ejecutan nuevamente las **Acciones** declaradas hasta que el valor de la **VARIABLE** exceda al valor **Final**.

Ahora bien, en caso de omitir el valor de **Paso**, el incremento de la **VARIABLE** al término de cada iteración (repetición) por defecto será en **1** unidades.

Bajo este mismo orden de ideas, las siguientes líneas de código ilustran la estructura básica (sintaxis) de un ciclo "**Para**" representado a nivel de Pseudocódigo digital:

```
Para variable = valInicio Hasta valFinal Con Paso valIncDec Hacer
        instrucciones
FinPara
```

Imagen 66: Estructura de Repetición "Para" - Fuente: Diseñada por el autor.

Y en caso de omitir el valor de paso o incremento personalizado:

```
Para variable = valInicio Hasta valFinal Hacer
        instrucciones
FinPara
```

Imagen 67: Estructura de Repetición "Para" (sin valor de paso) - Fuente: Diseñada por el autor.

Con la finalidad de ejemplificar las ideas expuestas hasta el momento, el siguiente Pseudocódigo digital implementa la estructura de repetición "**Para**", con la finalidad de generar la tabla de multiplicar de un número proporcionado por el usuario, hasta llegar a 10.

```
1   Proceso tablaMultiplicar
2       //---- Datos de Entrada
3       Imprimir "Programa Tablas de Multiplicar"
4       Imprimir "-----------------------------"
5       Imprimir "Ingrese un número"
6       Leer num
7       //Indicamos la tabla de multiplicar generada
8       Imprimir ""
9       Imprimir "Generando Tabla del " , num
10      Imprimir ""
11      //Se requiere multiplicar el número proporcionado
12      //por el usuario, desde 1 hasta 10
13      Para contador = 1 Hasta 10 Hacer
14          //Generamos una salida por cada multiplicación en turno
15          //8 x 1 = 8
16          //8 x 2 = 16, etc.
17          Imprimir num , " x " , contador , " = " , (num * contador)
18      FinPara
19  FinProceso
```

Imagen 68: Estructura de repetición "Para" implementada en un Pseudocódigo digital escrito en PSeInt - Fuente: Diseñada por el autor.

Como se puede observar, en el ejemplo anterior las líneas 5-6 se encargan de solicitar al usuario el número a partir del cual se debe proceder a generar su correspondiente tabla de multiplicar. Así mismo, la línea de código 9 reafirma esta situación al mostrar en pantalla un encabezado descriptivo de las acciones a generar. Más adelante en las líneas de código 13-18 se declara la construcción de un bloque de repetición del tipo "**Para**", mismo que especifica que cada una de sus instrucciones internas (línea 17) se deben ejecutar al menos 10 veces. Esto es así porque la variable de control utilizada dentro de esta misma construcción, "**contador**", se inicializa en **1** (punto de inicio) y su valor debe incrementar en la unidad (uno) al término de cada repetición (puesto que no se declaró de forma explícita el valor de paso). De manera que la estructura de repetición debe comprobar al término de cada ciclo (vuelta) el valor actual de "**contador**" para verificar que este no exceda el máximo permitido (es decir 10) y, poder así, repetir las instrucciones declaradas dentro de este bloque. En caso contrario, sale del ciclo (lo termina) y

continúa la ejecución de las instrucciones que aparecen inmediatamente después a este.

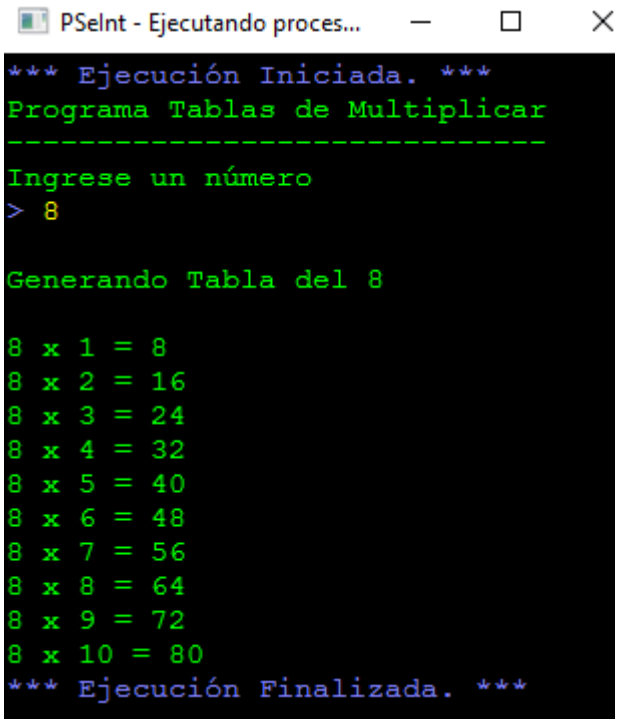


Imagen 69: Ejecución Pseudocódigo digital estructura de repetición "Para" - Fuente: Diseñada por el autor.

Ahora bien, resulta importante destacar la serie de concatenaciones (uniones) que tienen a lugar en la línea de código número 17. En este punto, lo que se busca es generar una salida personalizada por cada resultado procesado, por ejemplo: si el usuario ingresa el número 8, las salidas de información hacen referencia a este número, seguido del carácter "**x**", el valor actual de la variable de control "**contador**", el carácter "**=**", y el resultado correspondiente a dicha operación (véase la imagen 69).



¿Sabía qué?

La estructura de repetición **Para** frecuentemente es utilizada al lado de las variables de tipo arreglo para recorrer cada uno de sus elementos.

Por otra parte, cuando se trabaja con estructuras de repetición, los programadores frecuentemente tienen la necesidad de almacenar los resultados generados al término de un ciclo y reutilizarlos en el siguiente, como si se tratara de un **acumulado** o **acarreo** de resultados.

Bajo esto mismo orden, Rodríguez establece que un acumulador:

> Es una variable que nos va a permitir guardar un valor que se incrementa o decrementa de forma no constante durante el proceso. En un instante determinado tendrá un valor y al siguiente tendrá otro valor igual o distinto (Rodríguez 19).

Por ejemplo, durante el desarrollo de un Pseudocódigo digital que obtiene el subtotal correspondiente a una venta, resulta evidente acumular el precio de cada artículo (adquirido por el cliente) para sumarlo con el precio del siguiente, y así sucesivamente. Tal y como se ilustra en la siguiente imagen:

```
1  Proceso acarreosPuntoVenta
2      Imprimir "--------------------------------------"
3      Imprimir "     TIENDAS GARRIS S.A. DE C.V.      "
4      Imprimir "--------------------------------------"
5      Imprimir ""
6      //--- Datos de Entrada
7      Imprimir "Número de artículos para vender"
8      Leer cantidadArticulos
9      //--- Proceso
10     subtotal = 0    //Al inicio de la venta el subtotoal es de $0.00
11     Para contador = 1 Hasta cantidadArticulos Hacer
12         //Solicitar el precio de cada artículo
13         Imprimir "Ingrese Precio Artículo #" , contador
14         Leer precio
15         //Sumar el precio del artículo actual al subtotal, y el
16         //resultado lo acumulamos (acarreamos) para la siguiente
17         //iteración
18         subtotal = subtotal + precio
19     FinPara
20     //Determinar impuestos y totales
21     iva = subtotal * 0.16
22     total = subtotal + iva
23     //--- Salida de Información
24     Imprimir ""
25     Imprimir "-------- COMPROBANTE DE PAGO ---------"
26     Imprimir ""
27     Imprimir "Número de Artículos: " , cantidadArticulos
28     Imprimir "Subtotal: $ " , subtotal
29     Imprimir "IVA:      $ " , iva
30     Imprimir "Total:    $ " , total
31 FinProceso
```

Imagen 70: Declaración de un acumulador de valores dentro de una estructura de repetición "Para" - Fuente: Diseñada por el autor.

Las líneas de código 11 a 19 declaran la construcción de una estructura "**Para**", misma que repite un conjunto de acciones (líneas 12-18) según la cantidad de artículos a vender por el usuario (véase la imagen 71). Luego, las líneas 13-14 se encargan de solicitar el precio del artículo en cuestión y crean una referencia hacia el mismo mediante el uso de la variable **precio**. Posteriormente, la línea de código número 18 se encarga de sumar el precio del artículo actual con el valor referenciado en la variable **subtotal**, misma que en la línea 10 es inicializada en cero, debido a que en ese momento se desconoce el precio de cada artículo.

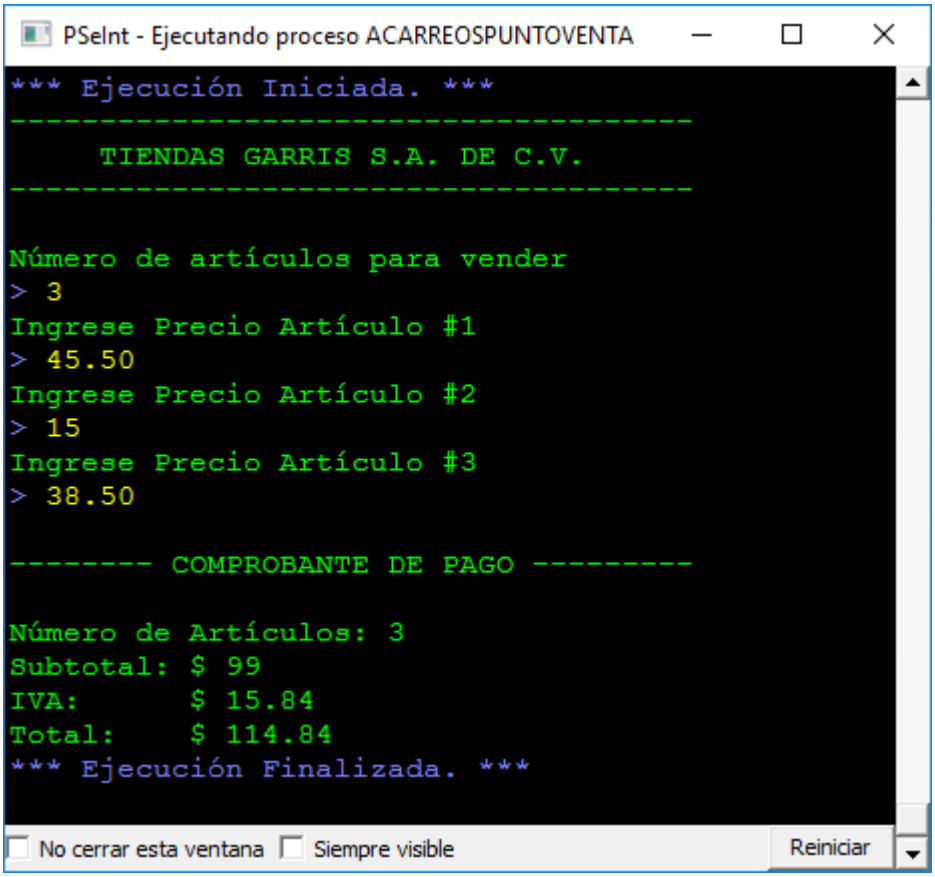


Imagen 71: Ejecución Pseudocódigo digital que implementa la declaración de un acumulador de valores - Fuente: Diseñada por el autor.

Sin embargo, el resultado de dicha expresión (**subtotal + precio**) es asignado nuevamente a la variable **subtotal**, lo que origina que su valor referenciado se sobre-escriba con base en el nuevo resultado generado y, por consiguiente, se le considere en las próximas iteraciones correspondientes a dicho proceso. Tal y como se ilustra en la siguiente tabla:

Ciclo	Valor de las variable durante cada ciclo		
	precio	subtotal + precio	subtotal
Antes del ciclo	-	-	0
1	45.50	0 + 45.50	45.50
2	15	45.50 + 15	60.50

3	38.50	60.50 + 38.50	**99.00**

Tabla 12: Esquema de trabajo interno, acumulador de valores - Fuente: Elaborada por el autor.

Como se puede observar, antes de que se ejecute la estructura de repetición el valor asignado a la variable **subtotal** es de **0** pesos (línea de código10). Posteriormente, al ejecutar la primera iteración dicho valor es sobre-escrito por **45.50** (resultado de sumar el **precio** del artículo actual al valor anterior de la variable **subtotal**, es decir 0 pesos). En el segundo ciclo, nuevamente se llevan a cabo las mismas operaciones, pero en esta ocasión, intervienen los valores **45.50** y **15**, dónde la suma de estos (**60.50** pesos) otra vez es almacenada en la variable **subtotal**, es decir, se acumula su valor para ser reutilizado en la siguiente iteración. Finalmente, en la tercera repetición se suma el **precio** del último artículo (**38.50** pesos) al subtotal anterior (**60.50** pesos) y genera como resultado los $ **99.00** pesos, mismos que aparecen al cierre de las operaciones correspondientes a la venta anterior.

Sumado a lo expuesto, es importante mencionar que la combinación de estructuras de repetición y selección (véase el ejercicio del Anexo C5), permite que se desarrolle una lógica de programación más compleja, misma que puede ser implementada en cada uno de los desarrollos (Pseudocódigos digitales) con la finalidad de lograr resultados cada vez más interesantes e innovadores.

Ver ejercicio – Anexo C5

Ahora bien, con base en la problemática a resolver por cada Pseudocódigo digital, frecuentemente los programadores se topan con la necesidad de realizar algún tipo de incremento personalizado durante la repetición de un

conjunto de instrucciones, o mejor aún, recorrerlas en orden inverso (véase el ejercicio del Anexo C6). En este sentido, es importante considerar la palabra reservada **Con Paso** disponible en cada estructura de repetición **Para**. Al implementar dicho término con un valor diferente a la unidad, se recorren las instrucciones declaradas dentro del ciclo de forma personalizada, es decir, de manera no contigua (de dos en dos, de tres en tres, etc.) y, con un valor negativo es posible transitar por su contenido en sucesión inversa.

Ver ejercicio – Anexo C6

Llegado a este punto, el lector debería contar con el contexto suficiente para poder implementar este tipo de estructura (ciclo **Para**) en cada uno de sus desarrollos. Sin embargo, tal y como se argumentó al inicio de éste apartado, existen otro tipo de estructuras repetitivas cuya labor se orienta más al cumplimiento de una determinada condición lógica, es decir, el programador no establece el número de veces que debe ejecutarse una determinada tarea, sino que ésta se repite n cantidad de veces siempre y cuando se cumpla una o más condiciones lógicas.



¿Por qué utilizar la estructura de repetición Para?

La estructura de repetición **Para** es utilizada en aquellos casos donde se conoce previamente el número de veces que se debe repetir la ejecución de un bloque de instrucciones. Esta estructura corresponde a las denominadas estructuras bien definidas, por lo que, internamente cuenta con un mecanismo de incremento o decremento automático (personalizado por el programador), que elimina la necesidad de efectuar operaciones adicionales para prevenir la presencia de ciclos infinitos.

2.1.1.2.2 Mientras

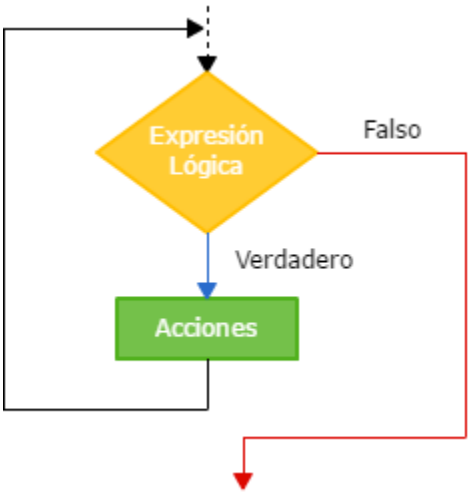


Imagen 72: Diagrama de Flujo Estructura de Repetición "Mientras" - Fuente: Diseñada por el autor.

La estructura de repetición "**Mientras**" permite ejecutar una secuencia de instrucciones mientras una condición sea evaluada como verdadera. Ante esta situación, Cairó añade que este tipo de estructura es la "… adecuada para utilizar en un ciclo cuando no sabemos el número de veces que éste se ha de repetir …" (Cairó 115). Por ejemplo, la venta de un conjunto de artículos sin conocer previamente el número de ellos, el total de ingresos percibidos (minuto a minuto) por concepto de donaciones referentes a una cruzada de salud social, etc.

Bajo estas circunstancias, el número de repeticiones resulta ser incierto (indeterminado) y en ocasiones es posible que las acciones declaradas dentro de la estructura no se lleguen a ejecutar nunca (debido a la condición inicial). Sin embargo, para evitar un comportamiento inesperado durante el proceso, es decir, un ciclo infinito. Deben existir dos cuestionamientos (lectura) o en su defecto dos asignaciones (sobre-escritura) de información que afecten la condición, una antes de entrar en el ciclo y otra antes de finalizar el mismo.

Ahora bien, con base en el diagrama de flujo anterior, se observa que al ingresar al bloque de repetición "**Mientras**", una determinada condición o conjunto de condiciones deben ser evaluadas. Si la(s) condición(es) resulta(n) ser verdadera(s), se ejecuta una vez el conjunto de acciones

declaradas en el cuerpo de dicho ciclo. Al finalizar se vuelve(n) a evaluar la(s) condición(es), y si resulta(n) nuevamente ser verdadera(s), la ejecución de las acciones se repite; en caso contrario, se sale del ciclo y, por consiguiente, se ejecutan las instrucciones declaradas fuera del mismo.

En este sentido, la sintaxis que representa la estructura de repetición "**Mientras**" a nivel de Pseudocódigo digital queda determinada por:

```
Mientras condición Hacer
    acciones
FinMientras
```

Imagen 73: Estructura de Repetición "Mientras" - Fuente: Diseñada por el autor.

Con la finalidad de ejemplificar las ideas expuestas hasta el momento, el siguiente Pseudocódigo digital se encarga de realizar la venta de un conjunto de artículos, mismos que en un inicio se desconoce su cantidad exacta. Al finalizar, emite un informe detallado por pantalla, dónde se indica el número de artículos vendidos, el subtotal, los impuestos a retener, así como el total a pagar por el cliente.

```
1   Proceso estructuraMientras
2       Imprimir "--- CENTRO COMERCIAL DOÑA LUCHA ---"
3       Imprimir ""
4       ///--- Datos de Entrada
5       //Primer cuestionamiento para entrar o no en la estructura
6       //de repetición Mientras
7       Imprimir "¿Desea realizar una venta? (si/no)"
8       Leer realizarVenta
9       Imprimir ""
10      //Declaración de acumuladores
11      subtotal = 0
12      numeroProductos = 0
13      ///--- Proceso de Información
14      //Repetir mientras el usuario desee realizar una venta
15      Mientras realizarVenta == "si" O realizarVenta == "SI" Hacer
16          Imprimir "Ingrese precio del producto"
17          Leer precioProducto
18          subtotal = subtotal + precioProducto
19          //Segundo cuestionamiento: si el usuario desea agregar
20          //otro producto a la venta actual, el ciclo se repite; en caso
21          //contrario, sale del ciclo.
22          Imprimir "¿Desea agregar otro producto? (si/no)"
23          Leer realizarVenta
24          //Acumulador para contabilizar el número de productos vendidos
25          numeroProductos = numeroProductos + 1
26      FinMientras
27      //Determinando impuestos y total a pagar por el cliente
28      iva = subtotal * 0.16
29      total = subtotal + iva
30      ///--- Salida de Información
31      Imprimir ""
32      Imprimir "--- COMPROBANTE DE PAGO ---"
33      Imprimir ""
34      Imprimir "Núm. Productos: " , numeroProductos
35      Imprimir "Subtotal: $ " , subtotal
36      Imprimir "IVA:      $ " , iva
37      Imprimir "Total:    $ " , total
38  FinProceso
```

Imagen 74: Estructura de repetición "Mientras" implementada en un Pseudocódigo digital escrito en PSeInt - Fuente: Diseñada por el autor.

Como se puede observar, las líneas 2-3 muestran en pantalla una cabecera poco atractiva donde se menciona el nombre del centro comercial que utiliza dicha solución (Pseudocódigo digital). Seguido de las líneas 7-8 que preguntan al usuario si desea o no realizar una venta. Posteriormente, el bloque de repetición "**Mientras**" declarado en las líneas 15-26 se

encarga de verificar la respuesta proporcionada y en caso de que el usuario acepte el realizar una venta (es decir, si su respuesta a la pregunta anterior contiene las cadenas de caracteres: "si" o "SI"), las acciones declaradas dentro del bloque (líneas 16-25) son ejecutadas una vez y al finalizar, se vuelve al inicio de la estructura para volver a comprobar las condiciones; en caso contrario, jamás se entra en el ciclo y el Pseudocódigo digital continua con la ejecución de las instrucciones declaradas fuera de dicha estructura (líneas 28-37).

Ahora bien, con la finalidad de evitar un comportamiento inesperado durante la ejecución del Pseudocódigo digital, es decir, un ciclo infinito. Las líneas de código 22-23 declaradas en el cuerpo de la estructura repetitiva se encargan de cuestionar nuevamente al usuario si desea o no agregar otro producto a la venta actual. En caso de ser afirmativo, las acciones se repiten nuevamente; en caso contrario (es decir, una respuesta diferente a las cadenas de caracteres "**si**" o "**SI**"), las condiciones declaradas al inicio de la estructura ya no se cumplen y por consiguiente el ciclo finaliza.

Bajo este mismo orden de ideas, una vez que se ejecuta dicho Pseudocódigo digital (**Tecla F9**) y se procede a realizar algunas ventas; el comportamiento que presenta es parecido al que se ilustra a través de las siguientes imágenes comparativas:

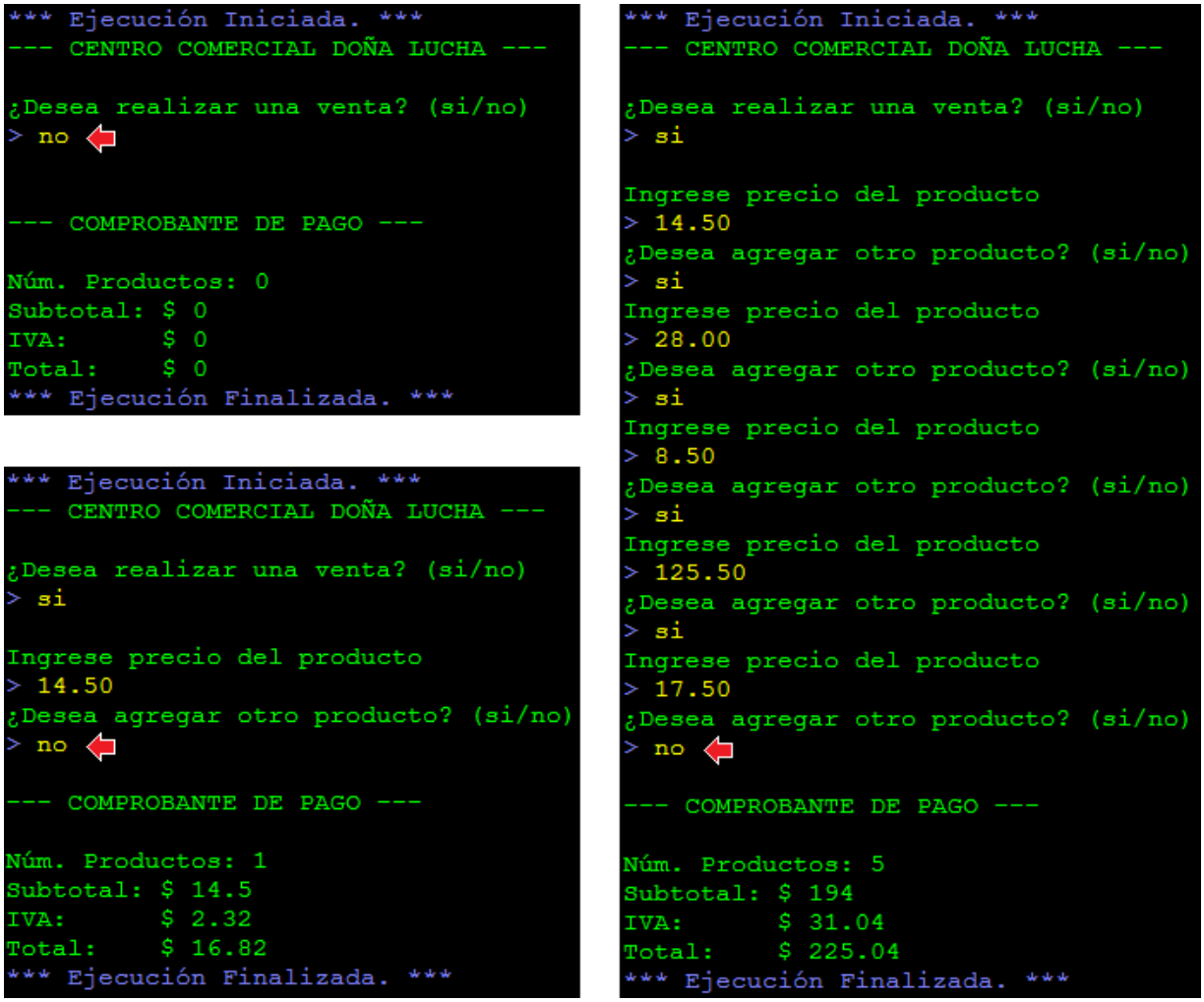


Imagen 75: Ejecución Pseudocódigo digital estructura de repetición "Mientras" - Fuente: Diseñada por el autor.

En la primera captura de pantalla, el usuario responde de manera negativa al primer cuestionamiento (¿Desea realizar una venta?) que hace el Pseudocódigo digital, por tanto, las instrucciones declaradas dentro de la estructura de repetición "**Mientras**" jamás se ejecutan y, por consiguiente, se genera un comprobante de pago de $ 0.00 pesos, es decir, las instrucciones declaradas inmediatamente después del ciclo repetitivo si son ejecutadas, sin embargo, al no existir productos para vender, estas arrojan como resultado unas cantidades nulas.

En la segunda ilustración (lado inferior izquierdo), el usuario responde de manera afirmativa al primer cuestionamiento, esto hace que la condición declarada en la parte superior de la estructura de repetición se cumpla y se ejecute una vez el bloque de instrucciones declaradas en su cuerpo. Una vez que el usuario ingresa el precio del producto, el Pseudocódigo digital hace un nuevo cuestionamiento para agregar otro producto diferente (obsérvese con detenimiento el nombre de la variable donde se asigna dicha respuesta, ésta es la misma que se utiliza para asignar el valor del primer cuestionamiento. En este sentido, su contenido se sobre-escribe y esto ocasiona que el ciclo termine en algún momento), en esta ocasión la respuesta del usuario es negativa y por tanto la segunda iteración no se lleva a cabo, sin embargo, las instrucciones que aparecen declaradas después del bloque de repetición si son ejecutadas, de tal manera, que es posible observar el detalle general de la venta.

Finalmente, en la tercera captura de pantalla, el usuario responde de manera afirmativa por cinco veces consecutivas, lo que origina que internamente se lleve a cabo la venta de cinco artículos distintos. Después de ello, en el sexto cuestionamiento el usuario responde de forma negativa, de tal forma que el ciclo repetitivo finaliza y, por consiguiente, el Pseudocódigo digital culmina de forma satisfactoria.



¿Sabía qué?

La estructura de repetición **Mientras** con frecuencia es utilizada por los programadores para recorrer los registros que retorna una consulta hecha a una base de datos.

Llegado a este punto, es importante mencionar que si bien es posible combinar diferentes estructuras de control para satisfacer la solución a

tareas de programación complejas. En ocasiones, resulta interesante trabajar con ellas de forma separada y reutilizar cada uno de sus resultados hasta producir una solución general. Tal y como se ilustra en el ejercicio del Anexo C7 correspondiente a este trabajo de investigación.

Ver ejercicio – Anexo C7

Hasta el momento, cada uno de los ejemplos abordados dentro de este apartado han solicitado información a los usuarios antes de iniciar la ejecución de las acciones declaradas dentro de la estructura "**Mientras**" y, por consiguiente, antes de finalizar la ejecución de las mismas (es decir, antes de finalizar un ciclo completo). Sin embargo, no siempre tiene porque ser así. También es posible realizar ciclos con la estructura repetitiva "**Mientras**" donde previamente se conozca el número de vueltas a realizar (véase el ejercicio del Anexo C8). Para ello, es importante declarar una variable que cumpla con la función de un contador de vueltas, e inicializar su contenido antes de ingresar al ciclo. Después, al término de cada iteración, proceder a modificar su valor con base en la lógica de solución del problema; de esta forma, se previene la presencia de comportamientos inesperados durante la ejecución del Pseudocódigo digital, y se logra que en algún momento éste finalice de forma satisfactoria.

Ver ejercicio – Anexo C8

Como se puede observar, a diferencia de la estructura "**Para**", la estructura de repetición "**Mientras**" se convierte en una herramienta interesante para el programador en aquellas situaciones donde se

desconoce la cantidad de ciclos (iteraciones) que se deben ejecutar para producir un resultado. Sin embargo, es importante recordar que dentro del bloque de instrucciones correspondientes a dicha estructura repetitiva debe existir un punto de inflexión que cambie el valor de la condición; de lo contrario, ésta siempre se evalúa como verdadera, lo que genera la presencia de un ciclo infinito que da cabida a un error grave de programación, es decir, la ejecución del Pseudocódigo digital nunca finaliza.



¿Por qué utilizar la estructura de repetición Mientras?

La estructura de repetición **Mientras** es utilizada en aquellos casos cuando se desconoce el número de veces que se debe repetir la ejecución de un bloque de instrucciones. Esta estructura corresponde a las denominadas estructuras condicionales, es decir, sus instrucciones se repiten mientras se cumpla una determinada condición. En este sentido, es importante que, al finalizar la ejecución de cada ciclo, exista la declaración de una expresión que modifique el valor de la condición, de lo contrario, ésta siempre se evalúa como verdadera, lo que genera la presencia de un ciclo infinito, es decir, que la ejecución del bloque de instrucciones declarado en el cuerpo de la estructura repetitiva nunca finalice.

2.1.1.2.3 Repetir Mientras Que

Desde la perspectiva de Gil Rubio, la estructura **Repetir Mientras Que** "… es similar al bucle generado con la instrucción [Mientras], con la diferencia de que la condición de ejecución se comprueba tras la ejecución del bloque de sentencias …" (Gil 74). En este sentido, dado que la condición se evalúa al final, las acciones declaradas dentro del cuerpo del ciclo se ejecutan por lo menos una vez. Sin embargo, para volver a repetirse, se debe proceder a evaluar la condición, y si esta resulta ser verdadera, las acciones declaradas dentro del cuerpo se ejecutan nuevamente. Pero, en caso de

que la comprobación sea falsa, el ciclo finaliza y el flujo del Pseudocódigo digital continúa con la ejecución de toda instrucción declarada fuera de dicha estructura repetitiva (véase la imagen 76).



Imagen 76: Diagrama de Flujo Estructura de Repetición "Repetir Mientras Que" - Fuente: Diseñada por el autor.

Sumado a lo expuesto, a fin de evitar ciclos infinitos durante la ejecución de un Pseudocódigo digital que implemente este tipo de estructura. El cuerpo del ciclo debe contener la declaración de una instrucción que modifique la variable de control del mismo, de modo que en algún momento ésta pueda ser evaluada como falsa y por consiguiente finalice la ejecución del ciclo.

Ahora bien, con la finalidad de implementar este tipo de estructura en cada uno de los desarrollos, las siguientes líneas representan la estructura básica (sintaxis) de un ciclo "**Repetir Mientras Que**" a nivel de Pseudocódigo digital.

```
Repetir
    acciones
Mientras Que condición
```

Imagen 77: Estructura de Repetición "Repetir Mientras Que" - Fuente: Diseñada por el autor.

Dónde:

- **acciones**: Representan la lista de instrucciones que se desea repetir hasta que la condición sea falsa.
- **condición**: Representa la expresión lógica (simple o compuesta) que se debe evaluar para realizar la siguiente iteración.

Para ejemplificar cada una de las ideas expuestas hasta el momento, el siguiente Pseudocódigo digital muestra un menú de opciones para la venta de boletos de viaje terrestre hacia diferentes destinos. Una vez que el usuario ha terminado de realizar una venta, se vuelve a mostrar el menú de opciones para realizar una venta nueva, es decir, las instrucciones declaradas en dicho Pseudocódigo digital se repiten. Sin embargo, si el usuario selecciona la opción salir (publicada también en el menú), se procede a mostrar una leyenda de despedida y, por consiguiente, el Pseudocódigo digital finaliza como de costumbre.

```
Proceso autoTransportes
    //Repetir las siguientes acciones hasta que el usuario
    //decida salir del sistema
    Repetir
        ///Entrada de Datos
        Imprimir "AUTO-TRANSPORTES AGUILA DEL CENTRO"
        Imprimir "-----------------------------------"
        Imprimir "1) TOLUCA                  $ 50.00"
        Imprimir "2) LERMA                   $ 40.00"
        Imprimir "3) LA MARQUESA             $ 30.00"
        Imprimir "4) CUAJIMALPA              $ 20.00"
        Imprimir "0) SALIR DEL SISTEMA"
        Imprimir ""
        Imprimir "SELECCIONE DESTINO: "
        Leer destino
        ///Proceso de Información
        //Si el destino es diferente de cero, entonces
        //iniciamos el proceso de venta de boletos de viaje
        Si destino != 0 Entonces
            Imprimir "NUMERO DE ASIENTOS: "
            Leer numAsientos
            subtotal = 0
            mensaje = "Operación exitosa."
            //Verificando el destino seleccionado
            Segun destino Hacer
                1:
                    subtotal = numAsientos * 50
                2:
                    subtotal = numAsientos * 40
                3:
                    subtotal = numAsientos * 30
                4:
                    subtotal = numAsientos * 20
                De Otro Modo:
                    mensaje = "Opción incorrecta"
            FinSegun
            ///Salida de Información
            //Si el destino coincide con alguno de los publicados en el menú,
            //se inicia la generación del ticket de pago (imprimir boleto)
            Si mensaje != "Opción incorrecta" Entonces
                iva = subtotal * 0.16
                total = subtotal + iva
                Imprimir ""
                Imprimir "SUBTOTAL: $ ", subtotal
                Imprimir "IVA:      $ ", iva
                Imprimir "TOTAL:    $ ", total
            FinSi
            //Al final de la venta, siempre se emite un mensaje de notificación
            Imprimir "MENSAJE:  " , mensaje
            //Se detiene el flujo del programa hasta que el usuario
            //presione una tecla para continuar.
            Imprimir "Presione una tecla para continuar"
            Esperar Tecla
            //Se limpia pantalla para mostrar el menú nuevamente, de
            //lo contrario este aparece al final de la venta anterior
            Limpiar Pantalla
        Sino
            //En caso de que el usuario decida salir del sistema...
            Imprimir "Sistema cerrado satisfactoriamente"
        FinSi
        //Comprobando que el bloque de instrucciones anteriores se repita
        //mientras el destino seleccionado sea diferente de cero
    Mientras Que destino != 0
FinProceso
```

Imagen 78: Estructura de repetición "Repetir Mientras Que" implementada en un Pseudocódigo digital escrito en PSeInt - Fuente: Diseñada por el autor.

Como se puede observar, las líneas 4 a 63 delimitan el cuerpo de la estructura "**Repetir Mientras Que**", donde sus acciones declaradas de forma interna se ejecutan siempre la primera vez que se inicia el Pseudocódigo digital. Sin embargo, antes de finalizar dicha estructura repetitiva se verifica que el valor asignado en la variable de control "**destino**" sea diferente de cero para volver a repetir nuevamente las acciones declaradas dentro del ciclo; de lo contrario, éste finaliza correctamente y, por consiguiente, se continúa con la ejecución de las instrucciones que aparecen en seguida del mismo.

Bajo este mismo orden de ideas, las líneas de código 6 a 12 muestran un menú de opciones en pantalla con los destinos disponibles para poder viajar. Luego, las instrucciones declaradas en las líneas 14 a 15 se encargan de solicitar al usuario la opción de destino seleccionada y en seguida se crea una referencia a dicho valor a través de la variable "**destino**", misma que es evaluada más adelante por la estructura de control "**Repetir Mientras Que**" (línea 63), a fin de supervisar el número de iteraciones por realizar y evitar así la presencia de ciclos infinitos durante su ejecución.

Sumado a lo expuesto, para verificar si es factible continuar o no con el proceso de venta de boletos de viaje terrestre, las líneas de código 19 a 60 declaran una estructura de selección "**Si Entonces Sino**" que se encarga de comprobar el destino seleccionado por el usuario. En caso de que este sea diferente de cero, el Pseudocódigo digital continúa con la operación de venta; en caso contrario (es decir, que la opción seleccionada sea cero), se imprime un mensaje de despedida (línea 59) y éste finaliza de forma satisfactoria (debido a que la condición del bloque de repetición se evalúa como **FALSO**).

Ahora bien, en el caso de que un usuario seleccione un destino diferente de cero, el Pseudocódigo digital continúa con la ejecución las instrucciones declaradas en las líneas 20 a 56 y solicita en primer lugar el número de asientos o boletos a vender (líneas 20-21) e inicializa algunos valores por defecto en las variables identificadas como "**subtotal**" y "**mensaje**" (líneas 22-23). Esto es así, debido a que, por una parte, el Pseudocódigo digital desconoce hasta este punto del proceso, el precio del boleto de viaje correspondiente al destino seleccionado (por tanto, no sabe cuánto cobrar), y por la otra, como la operación de venta fluye con normalidad, se da por hecho de que se trata de un procedimiento exitoso.

Una vez que el Pseudocódigo digital cuenta con toda la información necesaria para llevar a cabo la venta de boletos, las líneas 25 a 36 se encargan de verificar el destino seleccionado por el usuario, e internamente comienzan a realizar operaciones matemáticas con base en el precio establecido. Si la operación de venta finaliza de forma exitosa, se imprime por pantalla los datos generales de dicha venta (líneas 41-46); en caso contrario, sólo se muestra un mensaje con el error producido (opción de destino incorrecta).

Llegado a este punto de la explicación, el Pseudocódigo digital funciona correctamente. Sin embargo, es importante recordar que una vez terminado el proceso de una venta, éste se vuelve a ejecutar desde el principio (líneas 5-62) y, en consecuencia, provoca que los detalles de la venta anterior sean imperceptibles por el usuario, puesto que su contenido se desplaza de forma inmediata de la pantalla. En este sentido, para prevenir dicho comportamiento inesperado, la instrucción "**Esperar Tecla**" declarada en línea 53 detiene el flujo de ejecución, es decir, pausa por un momento el Pseudocódigo digital en ese punto del proceso y espera a que

el usuario presione una tecla (cualquiera) para poder reanudar su operación (véase la imagen 79).

```
*** Ejecución Iniciada. ***
AUTO-TRANSPORTES AGUILA DEL CENTRO
----------------------------------
1) TOLUCA                  $ 50.00
2) LERMA                   $ 40.00
3) LA MARQUESA             $ 30.00
4) CUAJIMALPA              $ 20.00
0) SALIR DEL SISTEMA

SELECCIONE DESTINO:
> 2
NUMERO DE ASIENTOS:
> 6

SUBTOTAL: $ 240
IVA:      $ 38.4
TOTAL:    $ 278.4
MENSAJE:  Operación exitosa.
Presione una tecla para continuar
```

Imagen 79: Ejecución Pseudocódigo digital estructura de repetición "Repetir Mientras Que" - Fuente: Diseñada por el autor.

Ahora bien, una vez que se ha corregido el problema de visibilidad correspondiente al detalle de la venta, tiene a lugar un nuevo inconveniente. Al repetir nuevamente las acciones declaradas en la estructura repetitiva, estas aparecen en pantalla inmediatamente después a las anteriores, es decir, se adjuntan. Ante esta situación, la instrucción "**Limpiar Pantalla**" que aparece en la línea de código 56, previene este comportamiento "normal" en el Pseudocódigo digital y lleva a cabo una limpieza general de los contenidos actualmente mostrados en pantalla, de esta manera, se prepara un nuevo espacio disponible, el cual es utilizado por la siguiente iteración para mostrar sus resultados correspondientes (detalles de la nueva venta).



¿Sabía qué?

Otro caso de uso típico de la estructura **Repetir Mientras Que** es cuando se requiere validar la información de entrada que proporciona un usuario. Si la información contiene algún error, entonces se debe solicitar nuevamente la entrada de dicha información y se procede a validar otra vez, de tal manera, que esto se repite hasta que la entrada sea correcta.

Como se ha podido observar, a lo largo de este ejemplo (venta de boletos de viaje terrestre), la estructura "**Repetir Mientras Que**" se le ha utilizado de forma general para volver a ejecutar todas las instrucciones declaradas en el programa. Sin embargo, al igual que las otras estructuras de su tipo abordadas con anterioridad, es posible utilizarla para iterar sobre un conjunto de acciones específicas. Tal y como se muestra en el ejemplo del Anexo C9 correspondiente a este mismo trabajo de investigación.

Ver ejercicio – Anexo C9



¿Por qué utilizar la estructura Repetir Mientras Que?

La estructura **Repetir Mientras Que** es similar a la estructura de repetición "Mientras", pero con la diferencia crucial de que la condición se evalúa siempre al final de la declaración del bloque de instrucciones. Esta particularidad permite que el cuerpo del ciclo se repita al menos una vez, mientras que la segunda repetición dependerá del resultado de la evaluación de la condición. En este sentido, la diferencia central que existe entre estas estructuras radica en el número mínimo de veces que se ejecuta su correspondiente bloque de instrucciones. En el caso de la estructura "Mientras", el bloque de instrucciones se ejecuta un mínimo de cero veces, sin embargo, en la estructura "Repetir Mientras Que", el mínimo es una vez.

Llegado a este punto, se da por finalizado el trabajo referente a las estructuras repetitivas y, por consiguiente, el apartado correspondiente a las estructuras de control. Ahora, se debería tener el conocimiento suficiente para modificar el flujo de ejecución de un Pseudocódigo digital; ya sea que éste necesite tomar una decisión o repetir un conjunto de instrucciones durante el proceso, es importante tener presente que en la mayoría de las ocasiones es posible asociar o combinar este tipo de

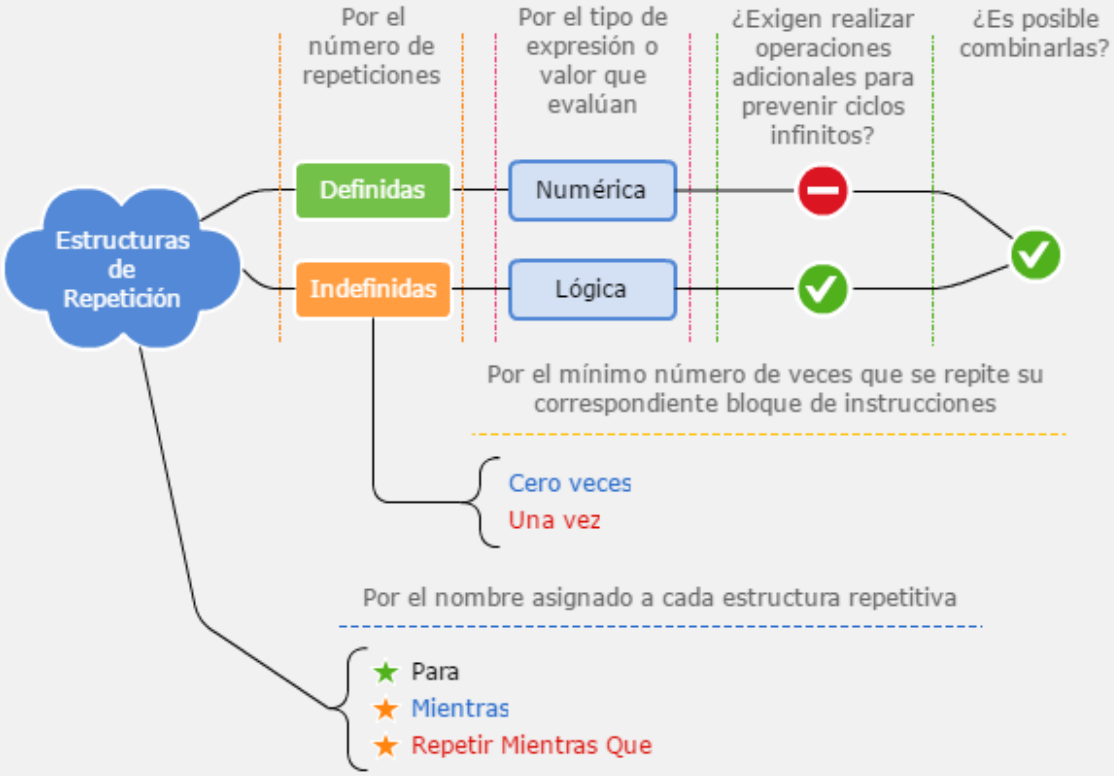
estructuras (selectivas/repetitivas) para atender algún tipo de necesidad en especial.



Resumen: Estructuras de Repetición.

En esta sección se abordaron las diferentes Estructuras de Repetición que permiten a un Pseudocódigo digital ejecutar un conjunto de instrucciones un número determinado de veces (de acuerdo al cumplimiento de una o varias condiciones).

Los elementos más importantes que se han comentado hasta el momento se resumen a través del siguiente esquema:



Como se puede observar, la diferencia central que existe entre cada estructura repetitiva radica en el conocimiento que se tenga acerca del número de veces que se debe ejecutar un determinado bloque de instrucciones durante el proceso de solución de un problema. A nivel condicional, la diferencia entre ellas se enfatiza por el tipo de valor o expresión que deben evaluar para llevar a cabo la ejecución continua de un conjunto de instrucciones. Además, para prevenir la presencia de ciclos infinitos durante su actuación, algunas estructuras obligan al programador a declarar un cierto tipo de expresión que modifique la o las variables involucradas en la condición, de modo que ésta sea falsificada en algún momento y así finalice su ejecución. Finalmente, del mismo modo que en las estructuras de repetición, es posible combinar su funcionamiento y utilizar cada uno de sus resultados generados para producir una solución final.

En el siguiente numeral se aborda todo lo referente al trabajo con las Funciones pre-construidas del Pseudolenguaje, las cuales permiten al desarrollador resolver tareas de programación cada vez más complejas, sin que exista la necesidad de escribir toda la lógica requerida para determinar su solución.



Resumen: Estructuras de Control.

Las Estructuras de Control juegan un papel fundamental en el desarrollo de los Pseudocódigos digitales, puesto que permiten modificar el flujo de ejecución de cada una de sus instrucciones (con base en condiciones), lo que genera que su comportamiento se perciba de una forma más o menos inteligente, debido a que éstos durante el proceso pueden tomar una serie de decisiones, así como repetir continuamente un conjunto de instrucciones.

Los elementos más importantes que se han comentado dentro de este apartado son los siguientes:

- Una estructura de control tiene un único punto de entrada y un único punto de salida (véase los diagramas de flujo de cada estructura).
- Existen dos tipos fundamentales de estructuras de control: Selección y Repetición.
- Las estructuras de selección se presentan en cuatro formas diferentes: Simples, Dobles, Anidadas y Múltiples.
- Para todas las estructuras de selección excepto la de tipo Múltiple, la decisión que toman se basa en una expresión que genera un resultado de tipo lógico.
- Las estructuras de repetición se presentan en dos formas distintas: Definidas e Indefinidas.
- Las estructuras de repetición definidas permiten establecer el número de veces que debe ejecutarse una determinada tarea.
- Las estructuras de repetición indefinidas permiten ejecutar continuamente una tarea, siempre y cuando se cumpla una determinada condición lógica.
- Uno de los grandes inconvenientes al momento de trabajar con las estructuras de repetición (por lo general las de tipo indefinido) son los ciclos infinitos. Para evitarlos, es importante que el programador declare una expresión que modifique la o las variables involucradas en la condición, de modo que ésta sea falsificada en algún momento del proceso y así finalice su ejecución.

2.1.2 Funciones primitivas del lenguaje (Pre-construidas)

Desde el punto de vista matemático, una función es una expresión que toma uno o más valores de entrada llamados argumentos y produce un resultado único de salida (véase el ejemplo ilustrado en la imagen 80), es decir, existe una relación entre un conjunto dado y otro generado, de forma que a cada elemento de entrada "x" le corresponde un único elemento de salida f(x). Algunos ejemplos de funciones matemáticas son: los logaritmos y las funciones trigonométricas (seno, coseno, tangente, etc.).

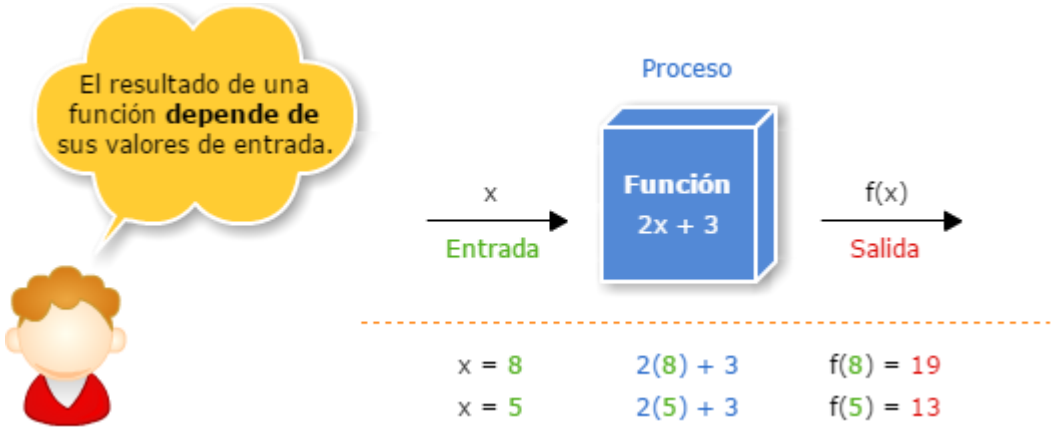


Imagen 80: Representación gráfica de una función - Fuente: Diseñada por el autor.

En el ámbito de la programación, las funciones tienen exactamente el mismo significado. Internamente "… se realizan ciertos cálculos con una o más variables de entrada, y se produce un único resultado, que podrá ser un valor numérico, alfanumérico o lógico …" (Casale 121). Dicho de otra forma, las funciones agrupan un conjunto de instrucciones o declaraciones que permiten realizan una tarea concreta. A su vez, pueden estar presentes de forma primitiva en un lenguaje de programación (también en algunos Pseudo-Intérpretes), o en su defecto pueden ser creadas por el propio programador (funciones externas).

Al momento de redactar estas líneas de la investigación, PSeInt incorpora 21 funciones pre-construidas, mismas que son clasificadas en: Funciones matemáticas y Funciones de cadena de caracteres, tal y como se muestra en la siguiente tabla:

Biblioteca de Funciones Predefinidas en PSeInt			
Función	**Significado**	**Ejemplo e Implementación**	
RAIZ(x)	Raíz cuadrada de x	RAIZ(81)	
ABS(x)	Valor absoluto de x	ABS(-544)	
TRUNC(x)	Parte entera de x	TRUNC(2.9785)	
REDON(x)	Entero más cercano a x	REDON(6.5874)	
SEN(x)	Seno de x	SEN(2.4)	Los valores que reciben estas funciones trigonométricas deben estar expresados en radianes.
COS(x)	Coseno de x	COS(1)	
TAN(x)	Tangente de x	TAN(2.7)	
ASEN(x)	Arco seno de x	ASEN(0.8)	
ACOS(x)	Arco coseno de x	ACOS(0.2)	
ATAN(x)	Arco tangente de x	ATAN(-0.4)	
AZAR(x)	Entero aleatorio en el rango [0 ; x-1]	AZAR(1523)	
ALEATORIO(a,b)	Entero aleatorio en el rango [a ; b]	ALEATORIO(2,9)	
EXP(x)	Función Exponencial de x	EXP(4)	
LN(x)	Logaritmo Natural de x	LN(7)	
MAYUSCULAS(s)	Retorna una copia de la cadena "s" con todos sus caracteres en mayúsculas	MAYUSCULAS("alex")	
MINUSCULAS(s)	Retorna un copia de la cadena "s" con todos sus caracteres en minúsculas	MINUSCULAS("MIGUEL")	
LONGITUD(s)	Cantidad de caracteres de la cadena "s"	LONGITUD("maría")	
CONCATENAR(s1,s2)	Retorna una nueva cadena resultado de unir las cadenas "s1" y "s2"	CONCATENAR("maría","reyes")	
SUBCADENA(s,x,y)	Retorna una nueva cadena que consiste en la parte de la cadena "s" que va desde la posición indicada en "x" hasta la	SUBCADENA("Alejandro",5,7)	

	posición "y" (incluye ambos extremos)	
CONVERTIRANUMERO(x)	Recibe una cadena de caracteres que contiene un número y lo devuelve en una variable numérica.	CONVERTIRANUMERO("5.4")
CONVERTIRATEXTO(s)	Recibe un número y lo devuelve en una variable de tipo cadena	CONVERTIRATEXTO(5.4)

Tabla 13: Ejemplo Biblioteca de Funciones Predefinidas en PSeInt - Fuente: Elaborada por el autor.

Como se puede llegar a observar, cada una de estas funciones se encarga de resolver una problemática en específico. En este sentido, durante el proceso de desarrollo de un Pseudocódigo digital éstas resultan ser de ayuda invaluable para el programador, ya que le evitan tener que escribir toda la lógica necesaria para resolver una determinada operación.



¿Sabía qué?

Si los valores dados a una función **"en tiempo de ejecución"** no son los que se esperaban, como pasar una cadena de caracteres en lugar de un valor numérico, PSeInt emite un error e interrumpe la operación. En este caso el error se debe a que no existe una coincidencia entre los tipos de datos gestionados.

Bajo este mismo orden de ideas, la problemática del triángulo rectángulo abordada con anterioridad, sólo ha sido posible determinar el valor de su correspondiente área a través del uso de datos conocidos como son su base y altura. Sin embargo, para conocer su perímetro, es necesario determinar previamente el valor de su hipotenusa. A este respecto, resulta evidente el uso de la función pre-construida **RAIZ** para obtener la medida de dicho segmento (hipotenusa) y, por consiguiente, sumarlo a la base y altura para determinar su respectivo perímetro. Tal y como se ilustra en el siguiente ejemplo:

```
1   Proceso programaTrianguloRectangulo
2       Imprimir "--- PROGRAMA TRIÁNGULO RECTÁNGULO ---"
3       Imprimir ""
4       ///--- Entrada de Datos
5       Imprimir "Ingrese la base: "
6       Leer base
7       Imprimir "Ingrese la altura: "
8       Leer altura
9       ///--- Proceso
10      //Determinando hipotenusa, perímetro y área
11      hipotenusa = RAIZ((base ^ 2) + (altura ^2))
12      perimetro = base + altura + hipotenusa
13      area = (base * altura) / 2
14      ///--- Salida de Información
15      Imprimir ""
16      Imprimir "Hipotenusa: " , hipotenusa
17      Imprimir "Área:       " , area
18      Imprimir "Perímetro:  " , perimetro
19  FinProceso
```

Imagen 81: Función pre-construida RAIZ implementada en un Pseudocódigo digital escrito en PSeInt - Fuente: Diseñada por el autor.

De el ejemplo anterior, es importante mirar con atención la expresión declarada en la línea número 11, según lo comentado hasta el momento (véase la tabla 13), la función **RAIZ(x)** obtiene la raíz cuadrada del valor "**x**" pasado como parámetro. En este caso, "**x**" se encuentra representada por la operación **(base ^ 2) + (altura ^ 2)**, misma que tras ser evaluada, devuelve un resultado en formato numérico, el cual se convierte en el valor de entrada para la función **RAIZ**.

Sumado a lo expuesto, una vez que se ejecuta dicho Pseudocódigo digital (**Tecla F9**) y se procede a ingresar algunos valores correspondientes (base y altura); su comportamiento es el que se ilustra a través de la siguiente captura de pantalla:

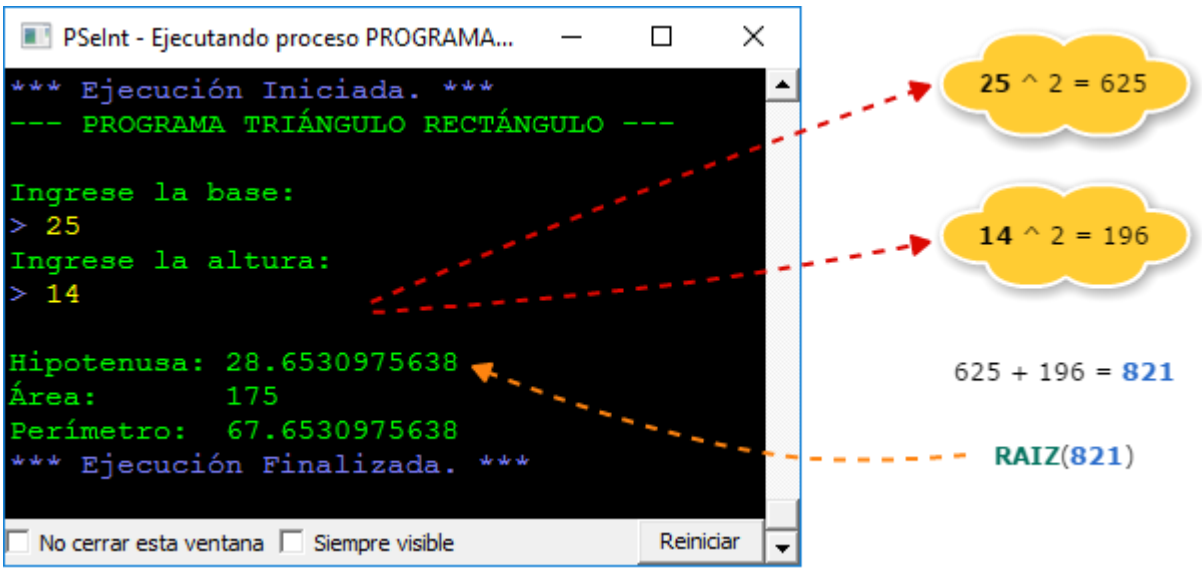


Imagen 82: Ejecución Pseudocódigo digital función pre-construida RAIZ - Fuente: Diseñada por el autor.

Como se puede observar, las funciones pre-construidas permiten al programador abstraerse por completo de la lógica y los detalles necesarios para determinar un cierto resultado. En este sentido, solo tiene que preocuparse por invocarlas en aquellas partes (en el código) donde las necesite, y por consiguiente utilizar los resultados generados para satisfacer las necesidades del problema principal.

Ahora bien, una vez que se conoce la importancia de las funciones pre-construidas y la forma de invocarlas desde el código principal. Es momento de abordar otro ejemplo práctico, pero esta vez con funciones que permitan trabajar exclusivamente con cadena de caracteres.



¿Sabía qué?

La mayoría de los lenguajes de programación implementan un extenso número de funciones que brindan el soporte suficiente para tener acceso a distintos tipos de servicios. Por ejemplo: conexión y manejo de bases de datos, trabajo con archivos, administración de directorios, envío de correo electrónico, etc.

En los últimos años, dado el crecimiento poblacional y el auge de las tecnologías de la información y de la comunicación. La mayoría de las Instituciones educativas se han visto en la necesidad de tener un mejor control sobre los procesos relacionados con su matrícula estudiantil. En este sentido, es común que un estudiante tenga asignado un número de credencial que le permita tener acceso a diferentes tipos de servicios, ya sean académicos, administrativos, de recreación o simplemente como un medio de identificación. La mayoría de estos códigos son generados a partir de un conjunto de reglas (también conocido como patrón) establecidas internamente por cada Institución, donde la mayoría de su información, se obtiene a partir de los datos personales de cada alumno. Por ejemplo: su nombre, apellidos, fecha de nacimiento, sexo, tipo de sangre, etc.

Con el objeto de ejemplificar las ideas expuestas hasta el momento, el siguiente listado expone una serie de reglas para generar códigos de credencial correspondientes a estudiantes inscritos en el Instituto de Investigación en Comunicación y Cultura.

1. El código de credencial debe estar formado por 23 caracteres separados en tres grupos por el símbolo menos, es decir: GRUPO1-GRUPO2-GRUPO3
2. El primer grupo de elementos debe contener la palabra ICONOS, por ejemplo: **ICONOS**-GRUPO2-GRUPO3
3. El segundo grupo está integrado a partir de la unión de los siguientes elementos correspondientes a cada estudiante:
    a. Primeras dos letras del apellido paterno (Mayúsculas)
    b. Primera letra del apellido materno (Mayúsculas)
    c. Primera letra del nombre (Mayúsculas)
    d. Dos últimos dígitos del año de nacimiento

e. Dos dígitos del mes de nacimiento
f. Dos dígitos del día de nacimiento
g. Sexo: H si es hombre, M si es mujer

4. El tercer grupo (Homoclave) consiste en un número entero aleatorio generado en el rango: [añoNacimiento ; añoRegistroCredencial]

Con base en los supuestos anteriores, para un estudiante de nombre Miguel Angel González Reyes nacido el 26 de marzo de 2001, y registrado en el año 2016. Su código de credencial quedaría formado por los caracteres: **ICONOS-GORM010326H-2014**

Ahora bien, para una estudiante de nombre Pamela Caballero Torres nacida el 9 de Julio de 1984, y cuyo año de registro data desde el 2011. Un posible código de credencial sería: **ICONOS-CATP870709M-1997**

Como se puede apreciar, en ambos casos la Homoclave (tercer grupo) generada es diferente, puesto que este dato es aleatorio y debe estar dentro del rango definido por [añoNacimiento ; añoRegistroCredencial], es decir, para el caso del estudiante Miguel Angel González Reyes, una Homoclave válida podría ser: **2001**, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, **2016**.



Imagen 83: Prototipo credencial estudiantes ICONOS - Fuente: Diseñada por el autor.

Llegado a este punto, una posible solución (escrita en Pseudocódigo digital) que atienda a la problemática antes expuesta mediante el uso de un ordenador, es como la que se ilustra a continuación:

```
Proceso funcionesCadenaTexto
    Imprimir "-----------------------------------------------------"
    Imprimir " PROGRAMA DE CREDENCIALIZACIÓN ESTUDIANTIL - ICONOS "
    Imprimir "-----------------------------------------------------"
    Imprimir ""
    ///--- ENTRADA DE DATOS
    Imprimir "INGRESE NOMBRE DEL ALUMNO: "
    Leer nombre
    Imprimir "INGRESE PRIMER APELLIDO: "
    Leer primerApellido
    Imprimir "INGRESE SEGUNDO APELLIDO: "
    Leer segundoApellido
    Imprimir "INGRESE FECHA DE NACIMIENTO: (dd-mm-yyyy)"
    Leer fechaNto
    Imprimir "INGRESE SEXO: [H] Hombre - [M] Mujer"
    Leer sexo
    ///--- PROCESO DE INFORMACIÓN
    separador1 = SUBCADENA(fechaNto,3,3)
    separador2 = SUBCADENA(fechaNto,6,6)
    //Verificando si existen los separadores de fecha en la posición correcta
    //además si la longitud de la fecha es idéntica a los 10 caracteres
    Si separador1 == "-" Y separador2 == "-" Y LONGITUD(fechaNto) == 10 Entonces
        //Obteniendo partes separadas correspondientes a la fecha de nacimiento
        dia = SUBCADENA(fechaNto,1,2)
        mes = SUBCADENA(fechaNto,4,5)
        anio = SUBCADENA(fechaNto,7,10)
        //Verificando si el sexo registrado contiene los caracteres H ó M
        Si MAYUSCULAS(sexo) == "H" O MAYUSCULAS(sexo) == "M" Entonces
            Imprimir "INGRESE FECHA ACTUAL DE REGISTRO: (yyyy)"
            Leer fechaActual
            //Generando homoclave
            homoclave = ALEATORIO(CONVERTIRANUMERO(anio),fechaActual)
            //Obteniendo partes separadas del nombre completo del alumno:
            //Luego se convierten las partes obtenidas en Mayúsculas
            iniNom = MAYUSCULAS(SUBCADENA(nombre,1,1))
            iniAPP = MAYUSCULAS(SUBCADENA(primerApellido,1,2))
            iniAPM = MAYUSCULAS(SUBCADENA(segundoApellido,1,1))
            //Obteniendo últimos dos digitos del año de nacimiento
            anioYY = SUBCADENA(fechaNto,9,10)
            //Convirtiendo a mayúsculas el sexo ingresado
            sexo = MAYUSCULAS(sexo)
            Imprimir ""
            Imprimir "Espere un momento, generando credencialización..."
            Esperar 2 Segundos
            Imprimir ""
            Imprimir "ALUMNO(A):  ",nombre," ",primerApellido," ",segundoApellido
            ///--- SALIDA DE INFORMACIÓN:
            //Mostrando credencial generada en formato: ICONOS-XXXXYYMMDDS-####
            Imprimir "CREDENCIAL: " Sin Saltar
            Imprimir "ICONOS-",iniAPP,iniAPM,iniNom,anioYY,mes,dia,sexo,"-",homoclave
        Sino
            Imprimir "Formato incorrecto: Sexo"
        FinSi
    Sino
        Imprimir "Formato incorrecto: Fecha de Nacimiento"
    FinSi
FinProceso
```

Imagen 84: Pseudocódigo digital que implementa diferentes funciones pre-construidas disponibles en PSeInt - Fuente: Diseñada por el autor.

Las líneas 2-5 muestran una cabecera minimalista donde se describe el nombre del programa (escrito en Pseudocódigo digital) y las siglas de la Institución educativa. Posteriormente, para proceder a generar el código de credencial, es necesario contar con ciertos datos personales del estudiante, tales como: su nombre completo, fecha de nacimiento y sexo. Para ello, las líneas 7-16 se encargan de solicitar dichos datos al usuario, y para el caso de datos rigurosos, se ofrece información adicional que describe puntualmente su formato de entrada (por ejemplo, en la fecha de nacimiento deben aparecer: dos dígitos para el día, dos dígitos para el mes y cuatro dígitos para el año, separados por un signo menos. A su vez, para el sexo debe aparecer: una letra **H** para el sexo masculino y una letra **M** para el sexo femenino).

Sumado a lo expuesto, como parte del proceso de generación del código de credencial, las líneas 18-19 se encargan de extraer los separadores (signos menos) de la fecha de nacimiento para que más adelante se utilicen para comprobar su existencia en el lugar correcto y pueda considerarse como valido el formato de fecha ingresado. Es en este punto donde se hace necesario utilizar la función pre-construida **SUBCADENA**, misma que obtiene un extracto o parte del texto pasado como argumento.

Bajo este mismo orden de ideas, si uno observa detalladamente el formato de entrada correspondiente a la fecha de nacimiento "**dd-mm-yyyy**" y a su vez se procede a identificar la posición numérica que ocupan sus separadores (signos menos), es posible llegar a la conclusión de que estos se encuentran en las posiciones 3 y 6 respectivamente. Además, si a este análisis se le suma la tarea que desempeña la función **SUBCADENA**, se tiene que las instrucciones declaradas en las líneas 18-19 son las correctas para obtener los separadores de fecha, tal y como se ilustra en la siguiente imagen:

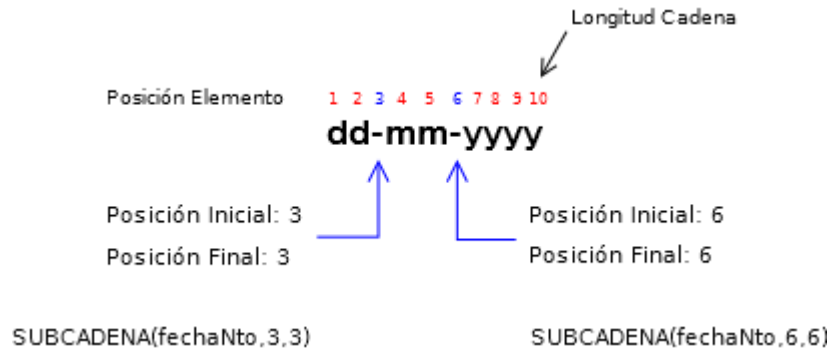


Imagen 85: Esquema de trabajo interno, función pre-construida SUBCADENA - Fuente: Diseñada por el autor.

Una vez que se tiene los extractos de texto almacenados (signos menos), la línea de código 22 se encarga de validar el formato de la fecha de nacimiento proporcionada. En este sentido, si el primer y segundo extracto de texto almacenado coinciden con un símbolo menos (-) y la longitud de la fecha (número de caracteres) es idéntico a 10, se considera como un formato de fecha válido. En caso contrario, existe un formato incorrecto y, por consiguiente, es inútil seguir con el proceso. Si bien esta validación no merece un reconocimiento significativo por su aportación a la seguridad cibernética (puesto que no se han validado los valores numéricos que intervienen en la fecha), sí que permite generar el contexto suficiente en el lector para que en la medida de lo posible valide toda la información que proviene desde del exterior, es decir, los datos que proporciona el usuario en el Pseudocódigo digital.

Ahora bien, en el caso de que el formato de fecha sea correcto, las líneas 24-26 extraen y almacenan por separado los dígitos correspondientes al día, mes y año de la fecha de nacimiento. Posteriormente, en la línea 28 se comprueba que el dato proporcionado como sexo coincida con el formato especificado, es decir, **H** o **M**. Pero, ¿qué pasa si estos datos son ingresados en letra minúscula?, bueno, con la finalidad de no hacer más extensa la condición, se hace uso de la función **MAYUSCULAS**, cuya labor

es convertir el dato pasado como argumento a su equivalente en mayúsculas. En este sentido, si el usuario ingresa un carácter "**h**" como valor para el dato sexo, durante la comprobación este se convierte en "**H**", de tal manera, que la validación es aprobada sin problema alguno.

Otra forma de llevar a cabo esta comprobación, es mediante uso de una regla condicional compuesta por cuatro casos posibles, es decir:

```
Si sexo == "H" O sexo == "M" O sexo == "h" O sexo == "m" Entonces
```

Imagen 86: "Regla Condicional Compuesta" para determinar el sexo de un usuario - Fuente: Diseñada por el autor.

Ambos casos son equivalentes y, por consiguiente, funcionan de la misma manera. Sin embargo, uno hace uso de la función pre-construida **MAYUSCULAS** para simplificar la regla condicional, y el otro opera de una forma más clásica.

Una vez que el dato correspondiente al sexo del estudiante es comprobado y este resulta ser válido, se procede a solicitar como último dato de entrada el año actual en el que se lleva a cabo el registro (líneas 29-30). Con todos estos datos referenciados a través de distintas variables en el Pseudocódigo digital, en la línea 32 se genera la Homoclave para dicho estudiante. Para ello, se hace uso de la función **ALEATORIO** la cual genera un número entero al azar dentro del rango indicado como argumento, y la función **CONVERTIRANUMERO** para transformar la subcadena que contiene los cuatro dígitos del año de nacimiento a un valor numérico. En este sentido, si un usuario registra un estudiante con fecha de nacimiento **09-07-1985** el 14 de mayo de **2016**, la línea de código 32 lleva a cabo las acciones que se muestran a continuación:



Imagen 87: Esquema de trabajo interno, funciones pre-construidas CONVERTIRANUMERO y ALEATORIO - Fuente: Diseñada por el autor.

Como se puede observar, la tarea para generar una Homoclave es un proceso sencillo de dos pasos, en primer lugar, se convierte a un valor numérico los cuatro dígitos correspondientes al año de nacimiento (puesto que la función **SUBCADENA** devuelve como resultado un extracto de información en formato de cadena de caracteres). Posteriormente, se genera un número entero aleatorio dentro del rango **[añoNacimiento ; añoRegistro]**, para lo cual, es importante contar con el año de nacimiento (en formato numérico) y con el año de registro. Ahora bien, con la finalidad de simplificar este proceso (de dos líneas en una sola), es posible pasar una función como argumento de otra función, es decir, combinarlas. En este sentido, su interpretación y forma de trabajar es de adentro hacia afuera. Por ejemplo, en la ilustración anterior primero se convierte a número el valor referenciado por la variable "**anio**" (función identificada de color azul) y, en seguida, se genera un número aleatorio dentro del rango "valor devuelto por la función **CONVERTIRANUMERO**, hasta el valor referenciado por la variable **fechaActual**" (función identificada de color verde).

Bajo este mismo orden de ideas (combinación de funciones pre-construidas), las líneas 35-37 extraen los caracteres necesarios tanto del nombre, apellido paterno y apellido materno para formar el código de credencial, y al mismo tiempo (en la misma línea de código), son convertidos a mayúsculas para ser asignados a través de nuevas variables (**iniNom**, **iniAPP**, **iniAPM**) dentro del Pseudocódigo digital (véase la imagen 88).

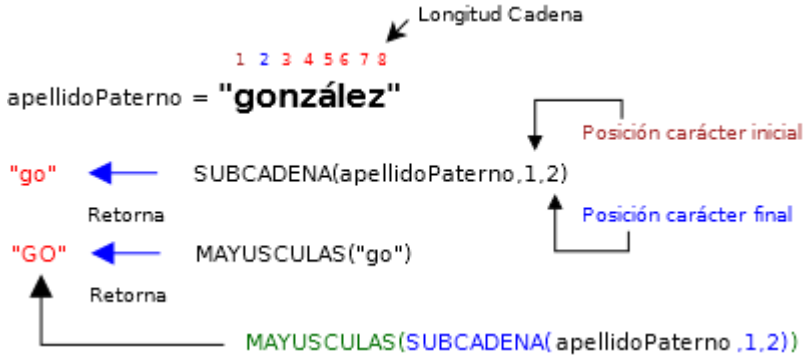


Imagen 88: Esquema de trabajo interno, funciones pre-construidas SUBCADENA Y MAYUSCULAS - Fuente: Diseñada por el autor.

Llegado a este punto del proceso, solo hace falta obtener los dos últimos dígitos del año de nacimiento (línea 39) y verificar que realmente el carácter utilizado para el sexo del estudiante se encuentre debidamente escrito con letra mayúscula (línea 41). Una vez resuelto estos inconvenientes, se puede decir que el proceso ha terminado y solo queda mostrar las salidas correspondientes al usuario. Para ello, el Pseudocódigo digital hace una pausa de dos segundos (línea 44), y una vez transcurrido este tiempo, emite información en pantalla; de tal manera, que es la línea de código número 50 la que se encarga de mostrar el código de credencial generado (combina cada parte con base en las reglas establecidas por el Instituto).

Una vez explicado a detalle las líneas de código más sobresalientes en el Pseudocódigo digital, la siguiente captura de pantalla muestra el resultado de su ejecución:

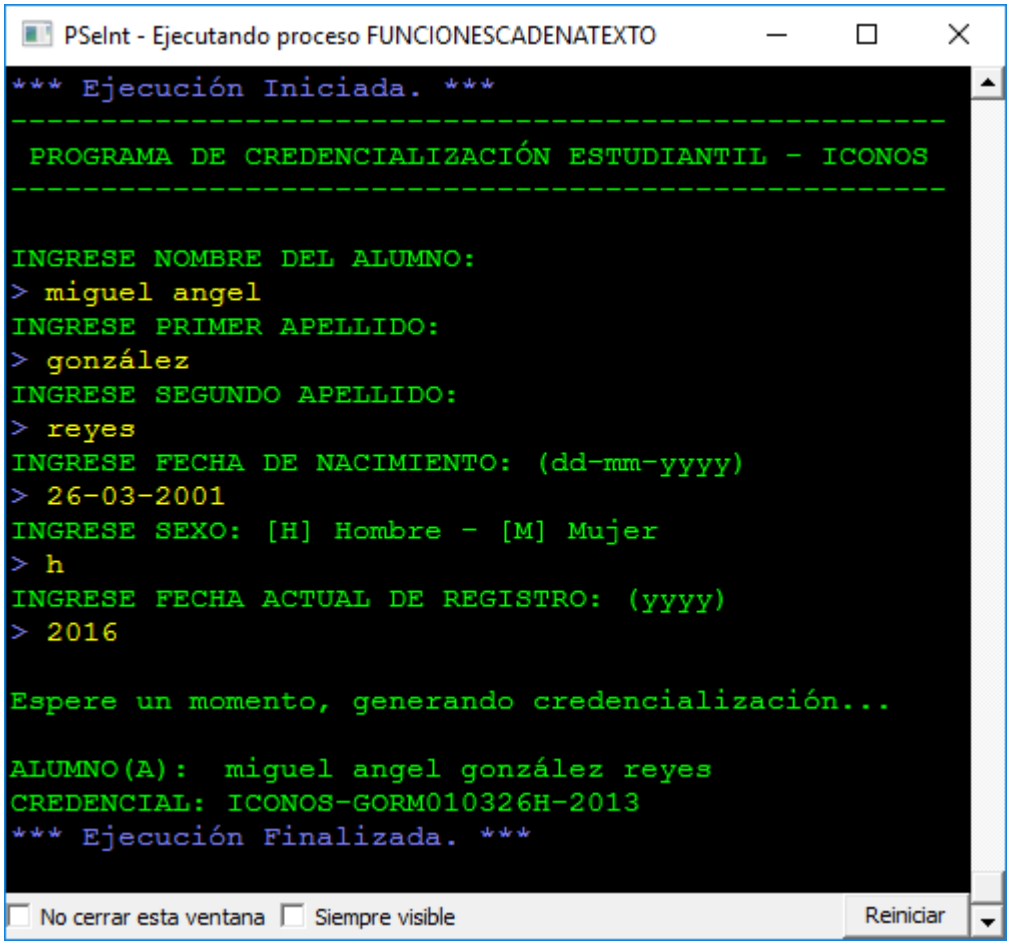


Imagen 89: Ejecución Pseudocódigo digital funciones pre-construidas disponibles en PSeInt - Fuente: Diseñada por el autor.

Como se puede apreciar, algunos datos en el Pseudocódigo digital son ingresados con caracteres escritos en minúsculas. Dado que la fecha de nacimiento y el sexo son correctos, es decir, satisfacen la validación. Al final, el código de credencial es generado con base en los lineamientos establecidos por la propia Institución (27 caracteres, tres grupos separados con el símbolo menos, letras mayúsculas, etc.).

Llegado a este punto, los ejemplos abordados dentro de este apartado deberían proporcionar el contexto suficiente para implementar cada una de estas funciones pre-construidas durante el análisis y diseño de los Pseudocódigos digitales. Sin embargo, es importante mencionar que su disponibilidad y presencia depende en gran medida del lenguaje de programación seleccionado. De modo que, si en algún momento se requiere utilizar alguna de estas funciones y desafortunadamente no aparece como parte primitiva del lenguaje. Es probable que su implementación si se encuentre definida, pero bajo el uso de otro nombre; por ejemplo: un sinónimo, otro idioma, abreviaturas, etc.

Vinculado al concepto, merece la pena mencionar que todo lenguaje de programación tiene su documentación en línea y allí siempre se puede encontrar las mejores soluciones y respuestas a los problemas que se presentan durante el proceso de desarrollo de un proyecto.

No obstante, si a pesar de los esfuerzos mencionados se llega a la conclusión de que la función por utilizar no se encuentra disponible de forma primitiva en el lenguaje; se tienen dos caminos. El primero hasta cierto punto es el más sencillo y consiste en cambiar de lenguaje de programación, de tal manera que se verifique previamente si las funciones por utilizar se encuentran definidas de forma primitiva. El segundo requiere de una curva de aprendizaje relativamente alta, pero al final brinda la posibilidad de crear las funciones que uno necesita. De allí pues, que sea el foco de atención del siguiente numeral.



Resumen: Funciones primitivas del lenguaje.

En esta sección se ha hecho un recorrido por la biblioteca de funciones predefinidas en PSeInt, a través de dos ejemplos prácticos, se explicó la forma de invocar cada una de estas funciones durante la solución de diferentes tareas de programación. Desde un sencillo cálculo para obtener la raíz cuadrada de un valor numérico, hasta procedimientos más complejos que involucran la generación de números enteros aleatorios, así como la comparación, extracción y conversión de cadenas de caracteres.

Como se ha podido observar, las **funciones primitivas del lenguaje** resultan ser de mucha utilidad para el programador, puesto que le evitan tener que escribir toda la lógica necesaria para determinar la solución a un problema en específico, dicho de otra forma, las funciones primitivas se comportan como auténticas cajas negras, donde el programador desconoce por completo su implementación (interna) y solo se limita a utilizarlas (invocarlas) para satisfacer una necesidad en común.



En la mayoría de las ocasiones, los lenguajes ya implementan una solución (en formato de función) para los problemas clásicos de programación, sólo hace falta indagar un momento por toda su documentación. En caso de no tener éxito con la localización de una función, es importante recordar que con frecuencia los lenguajes de programación llevan a cabo su registro mediante el uso de un nombre diferente de identificador. Por ejemplo: un sinónimo, abreviaturas, siglas, otro idioma, etc.

2.1.3 Funciones del programador (Externas)

En el apartado anterior se han desarrollado Pseudocódigos digitales cada vez más complejos que hacen uso de numerosas funciones que ya vienen integradas de forma primitiva en el Pseudolenguaje de PSeInt; algunos

ejemplos de ellas son: **RAIZ**, **ALEATORIO**, **SEN**, **COS**, **TAN**, **LONGITUD**, **TRUNC**, **SUBCADENA**, etc. Como se ha podido observar, la ejecución de cada una de estas funciones consiste en un proceso sencillo, el cual puede ser resumido a través de los siguientes pasos.:

1. Invocar la función mediante su nombre de identificador
2. Indicar cuales son los valores (también llamados argumentos) con los que va a trabajar la función.
3. Asignar, operar, comparar o imprimir el resultado devuelto por la función.

Bajo este mismo orden e ideas, PSeInt no restringe su uso solo a las funciones pre-construidas, también ofrece la posibilidad de diseñar funciones personalizadas que implementen toda la lógica necesaria para realizar alguna tarea en específico, misma que hasta el momento no ha sido cubierta de forma primitiva por el Pseudo-Intérprete PSeInt.

En este sentido, Gil Rubio mediante su obra Creación de Sitios Web con PHP5, argumenta que:

> "… Una función es un trozo de código que permite desarrollar una tarea concreta y bien definida, que se encuentra separado del resto de instrucciones del programa y al que se le ha dado un nombre para que, posteriormente, pueda ser referenciado …" (Gill 89).

A este respecto, se puede decir que una función agrupa un conjunto de instrucciones "de uso común" en un solo componente reciclable, mismo puede ser invocado en diferentes partes del Pseudocódigo digital a medida que se necesite. Por ejemplo, volviendo la mirada al generador de códigos

de credencial desarrollado en el apartado anterior, se puede llegar a observar que en repetidas ocasiones la función **SUBCADENA** es invocada para obtener los trozos de texto necesarios a partir del nombre, apellidos y fecha de nacimiento del estudiante. Por su parte, la función **MAYUSCULAS** se solicita de manera frecuente para garantizar que los datos de la credencial cumplan con los requisitos establecidos, es decir, que los caracteres del segundo grupo de elementos (iniciales del nombre, apellidos y sexo) aparezcan escritos en letras mayúsculas.

Generalmente cuando uno decide diseñar una función externa, se hace con diferentes propósitos en mente, por ejemplo:

- **Reducir la cantidad de código duplicado:** Las funciones solo se diseñan una vez, pero pueden invocarse muchas veces a lo largo de la ejecución del Pseudocódigo digital.
- **Dividir el problema en tareas más simples:** Con base en el refrán divide y vencerás, las funciones permiten dividir el Pseudocódigo digital en pequeños módulos que atienden una tarea específica y bien definida, con el fin de hacerlo más legible (fácil de leer) y manejable (código ordenado).

Sumado a lo expuesto, Vikram Vaswani menciona que "… esta práctica no sólo reduce la cantidad de código duplicado que [se debe] escribir, también hace que [los códigos de los programas] sean más limpios, eficientes y fáciles de mantener" (Vaswani 122). Es decir, cuando se agrupa cierta cantidad de código mediante una función, sus correspondientes tareas de mantenimiento o actualización se simplifican, puesto que son llevadas a cabo dentro de un mismo lugar (en la definición de la función), lo que origina que los cambios realizados sean visibles en

cada una de las partes donde la función es ejecutada, sin poner en riesgo la integridad del código principal.

Antes de comenzar a abordar la sintaxis necesaria para definir una función en PSeInt, es necesario atender los diferentes componentes que pueden estar presentes o ausentes en toda función:

- **Parámetros:** representan los datos de entrada con los que va a trabajar la función.
- **Valor de retorno:** representa la salida o resultado generado por la función, con el fin de que sea utilizado como parte de otro proceso.

Llegado a este punto, el esquema básico para definir una función externa en PSeInt queda representado por:

```
Funcion _NOMBRE( )
    acciones a ejecutar
FinFuncion
```

Imagen 90: Estructura básica de una "Función Externa" - Fuente: Diseñada por el autor.

Donde:

- **Función** (o **Subproceso**, son equivalentes)**:** Indica el inicio de la definición de una función.
- **_NOMBRE:** indica el nombre del identificador utilizado para hacer referencia a la función dentro del Pseudocódigo digital. Las reglas para los nombres de las funciones son las mismas que para cualquier identificador de variable. Sin embargo, para los efectos prácticos de esta investigación, se propone que inicien con el símbolo de guion

bajo seguido de un conjunto de caracteres (nombre de la función) escritos con letras mayúsculas.

- **( ):** Los paréntesis agrupan la lista de parámetros con los cuales va a trabajar internamente la función. Si la función no recibe ningún valor, estos deben declararse vacíos.
- **acciones a ejecutar:** contiene toda la lógica de programación necesaria para resolver una tarea específica.

Con la finalidad de ejemplificar las ideas expuestas hasta el momento, el siguiente Pseudocódigo digital expone la definición de una función externa que se encarga de mostrar un mensaje de bienvenida durante 3 segundos, la cual es invocada al inicio del código principal como un simple pretexto de cortesía para el usuario final.

```
1   ///Función del Programador
2   //Muestra un mensaje de bienvenida al usuario
3   Funcion _MENSAJEBIENVENIDA()
4       Imprimir "-------------------------------------"
5       Imprimir "    Bienvenido estimado usuario.     "
6       Imprimir "-------------------------------------"
7       Imprimir ""
8       Imprimir "Espere un momento por favor..."
9       Esperar 3 Segundos
10      Limpiar Pantalla
11  FinFuncion
12
13  Proceso programaPrincipal
14      //Invocando la función externa
15      _MENSAJEBIENVENIDA()
16      Imprimir "Ingrese un número"
17      Leer numA
18      Imprimir "Ingrese otro número"
19      Leer numB
20      Imprimir "La suma es: " , (numA + numB)
21  FinProceso
```

Imagen 91: Pseudocódigo digital que implementa la definición de una función externa - Fuente: Diseñada por el autor.

Como se puede apreciar, las instrucciones correspondientes a la función **_MENSAJEBIENVENIDA** (líneas 3-11) se encuentran debidamente separadas del proceso principal (líneas 13-21). Internamente, la función agrupa siete instrucciones, de las cuales, cinco de ellas se encargan de mostrar información por pantalla (líneas 4-8) referente al mensaje de bienvenida. Posteriormente, la instrucción número seis (línea de código 9) se encarga de pausar el flujo de ejecución durante 3 segundos. Finalmente, la orden número siete (línea 10) lleva a cabo una limpieza de información en pantalla y prepara el espacio necesario para mostrar otro tipo de informaciones.

Sumado a lo expuesto, con la finalidad de ejecutar las acciones declaradas en la definición de la función externa **_MENSAJEBIENVENIDA**. La línea de código número 15 perteneciente al proceso principal, invoca a dicha función externa del mismo modo que una función primitiva del lenguaje, es decir, establece el nombre del identificador de función seguido de sus correspondientes paréntesis.

Ahora bien, en el momento que se ejecuta dicho Pseudocódigo digital (proceso principal), éste inicia operaciones a partir de la línea de código número 13. Posteriormente, cuando toca el turno a la instrucción declarada en la línea 15 (función externa), el flujo de ejecución se traslada hasta la línea 3 y comienza a procesar cada una de las instrucciones declaradas internamente en la función **_MENSAJEBIENVENIDA** (líneas 4-10). Al finalizar su ejecución, el flujo del Pseudocódigo digital regresa nuevamente al bloque principal y procesa la instrucción que sigue a la declaración de dicha función externa, es decir, la línea de código 16 (donde se quedó). En caso de que exista más de una función externa declarada en el cuerpo del proceso principal, el comportamiento descrito con anterioridad se repite una y otra vez hasta finalizar.

La siguiente captura de pantalla ilustra el comportamiento del Pseudocódigo digital anterior tras su correspondiente ejecución:

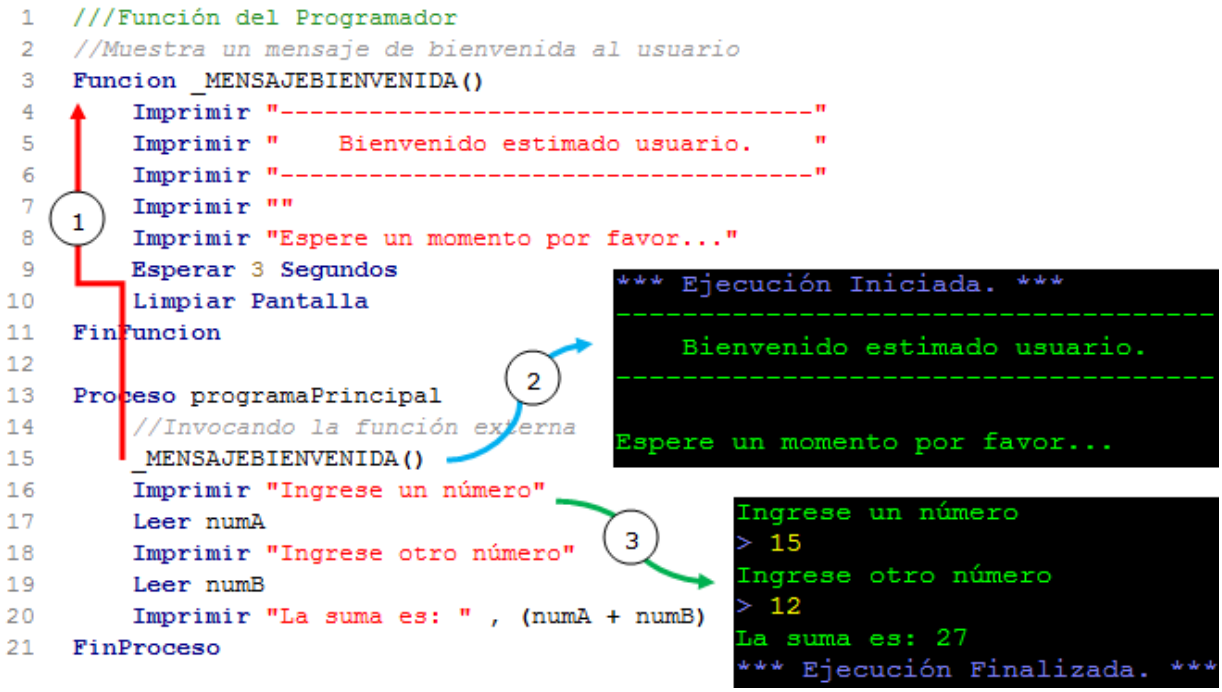


Imagen 92: Ejecución Pseudocódigo digital función externa - Fuente: Diseñada por el autor.

Como se puede observar, las instrucciones declaradas en la función anterior son poco eficaces, ya que no hace falta realizar un procedimiento adicional como este para mostrar un simple mensaje de bienvenida al usuario. Sin embargo, sí que permite formarse una idea de cómo hacer los Pseudocódigos digitales más eficientes, puesto que, al encapsular el código dentro de una función, éste se puede reutilizar las veces que sea necesario en el proceso principal, sin que exista la necesidad de tener declarado código duplicado que solo viene a complicar a futuro su legibilidad, manejabilidad, mantenimiento o actualización.

Por otra parte, para que una función externa genere un resultado diferente en cada ejecución, es importante que ésta cuente con un listado de

parámetros con los cuales pueda trabajar, e internamente modifique cada uno de estos datos, de modo que le sea posible producir un resultado final. Dicho de otra forma, como los datos de entrada destinados a la función cambian en cada llamada, la salida que genera siempre es diferente (las salidas están en relación con las entradas).

En este sentido, la sintaxis que guarda la definición de una función que acepta valores como parámetros de entrada, queda determinada por:

```
Funcion _NOMBRE(parametroA, parametroB, parametroN )
    acciones a ejecutar
FinFuncion
```

Imagen 93: Estructura básica de una "Función Externa con Parámetros" - Fuente: Diseñada por el autor.

Donde:

- **parametroA**, **parametroB**, **parametroN:** representan cada uno de los valores con los que puede trabajar internamente la función. En caso de que la función trabaje con más de un valor, estos deben ser separados por comas.

Sobre la base de lo explicado, el siguiente Pseudocódigo digital define una función externa que acepta dos parámetros para llevar a cabo una conversión de medidas:

```
1  ///Función del Programador
2  //Convierte una cantidad expresada en milímetros, centímetros o
3  //kilómetros a su equivalente en metros.
4  Funcion _AMETROS(cantidad, unidadMedida)
5      Segun unidadMedida Hacer
6          "mm","MM","Mm":
7              //milímetros a metros.
8              resultado = cantidad / 1000
9              Imprimir cantidad,"mm. es equivalente a: ",resultado,"mts."
10         "cm","CM","Cm":
11             //centímetros a metros.
12             resultado = cantidad / 100
13             Imprimir cantidad,"cm. es equivalente a: ",resultado,"mts."
14         "km","KM","Km":
15             //kilómetros a metros.
16             resultado = cantidad * 1000
17             Imprimir cantidad,"km. es equivalente a: ",resultado,"mts."
18         De Otro Modo:
19             //Mensaje por defecto en caso de que la unidad de medida
20             //proporcionada no sea compatible con la función.
21             Imprimir "Unidad de medida ",unidadMedida,". no soportada"
22     FinSegun
23 FinFuncion
24
25 Proceso programaPrincipal
26     Imprimir "----- Convertidor de medidas con Funciones -----"
27     Imprimir ""
28     _AMETROS(150,"cm")
29     _AMETROS(740,"mm")
30     _AMETROS(2.5,"km")
31     _AMETROS(800,"yd")
32 FinProceso
```

Imagen 94: Pseudocódigo digital que implementa la definición de una función externa con parámetros - Fuente: Diseñada por el autor.

La función anterior **_AMETROS** queda delimitada por las líneas 4-23, acepta dos parámetros "**cantidad** y **unidadMedida**" (línea 4) mismos que representan la distancia y la unidad de medida que se debe convertir a su equivalente en metros planos. El bloque declarado en las líneas 5-22 (estructura de selección) se encarga de verificar el tipo de unidad de medida ingresada a la función y compara su valor con uno de tres casos posibles: milímetros (línea 6), centímetros (línea 10) o kilómetros (línea 14). En caso de que la unidad de medida coincida con alguno de ellos, se

procede a realizar las acciones declaradas para dicho caso, es decir, convertir la distancia ingresada a su equivalente en metros planos y mostrar el resultado en pantalla. En caso contrario, se muestra un mensaje al usuario que indica que la unidad de medida ingresada no es compatible con la función (líneas 18-21).

Para ejecutar la función **_AMETROS** desde el entorno del proceso principal (líneas 25-32), es necesario escribir su nombre seguido de paréntesis, e internamente pasar la lista de valores con los cuales trabajará la función (véase la imagen 95).

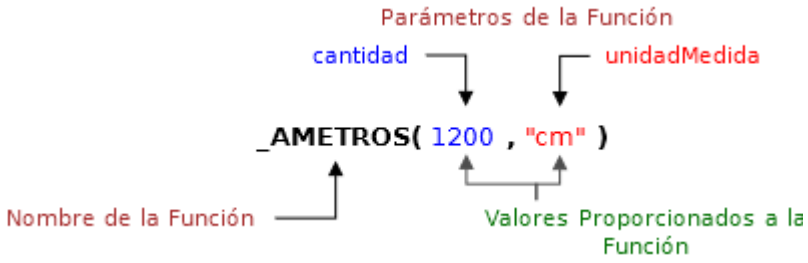


Imagen 95: Invocar una función externa con parámetros como parte del proceso principal de un Pseudocódigo digital - Fuente: Diseñada por el autor.

Técnicamente, el proceso de enviar información a una función se le conoce como "**pasar un argumento**". En el Pseudocódigo digital anterior, la función **_AMETROS** es invocada en cuatro ocasiones (líneas 28-31). Luego, en cada llamada recibe información diferente como argumentos. Como los valores enviados a la función cambian cada vez que ésta es invocada, permite que su comportamiento sea diferente en cada ejecución y, por consiguiente, que sus resultados generados sean versátiles. Tal y como se muestra en la siguiente imagen:

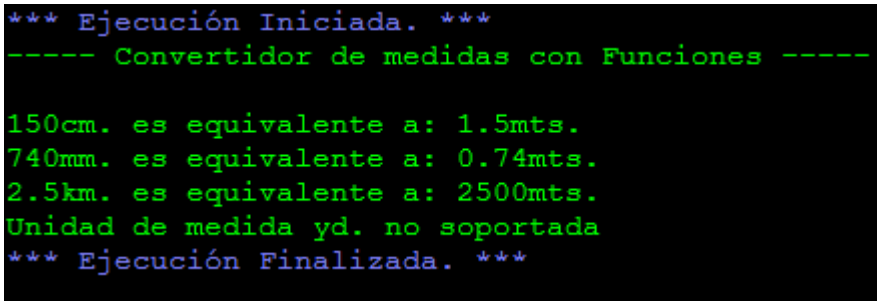
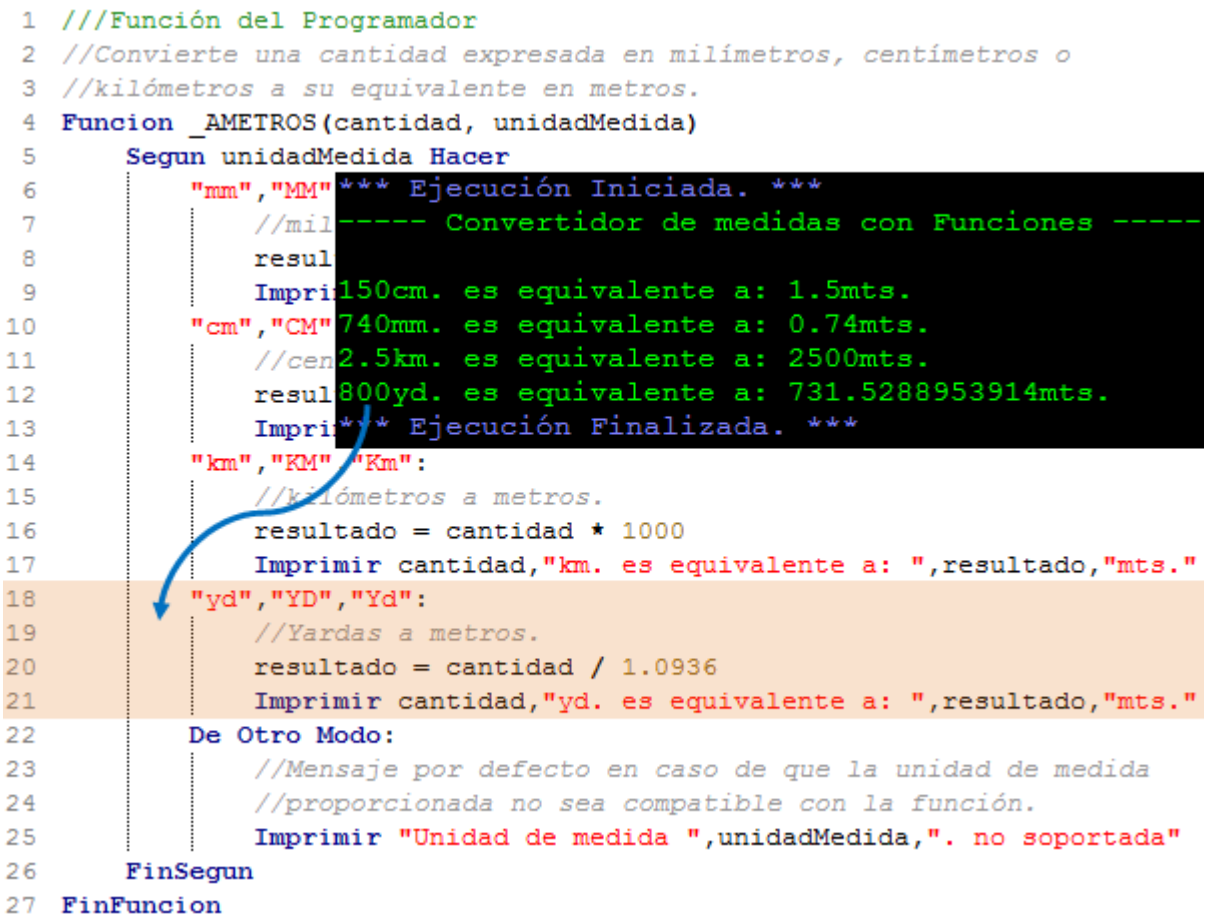


Imagen 96: Ejecución Pseudocódigo digital función externa con parámetros - Fuente: Diseñada por el autor.

De la imagen anterior, se puede observar que, en los tres primeros casos, la función **_AMETROS** imprime un resultado numérico diferente. Esto es así, debido a que los argumentos pasados a la función son distintos para cada ejecución. Sin embargo, en su última llamada ésta imprime un mensaje de error, ya que la unidad de medida pasada como valor (yardas) aun no es soportada por la función.

En este sentido, si más adelante se decide realizar una actualización a dicha función (véase el ejemplo ilustrado en la imagen 97), los cambios se llevarían a cabo en un solo lugar, es decir, dentro de la definición de la función. Esto permite que las tareas de mantenimiento se faciliten y, por consiguiente, se garantice la integridad del proceso principal.




```
///Función del Programador
//Convierte una cantidad expresada en milímetros, centímetros o
//kilómetros a su equivalente en metros.
Funcion _AMETROS(cantidad, unidadMedida)
    Segun unidadMedida Hacer
        "mm","MM":
            //mil
            resul
            Impri
        "cm","CM":
            //cen
            resul
            Impri
        "km","KM","Km":
            //kilómetros a metros.
            resultado = cantidad * 1000
            Imprimir cantidad,"km. es equivalente a: ",resultado,"mts."
        "yd","YD","Yd":
            //Yardas a metros.
            resultado = cantidad / 1.0936
            Imprimir cantidad,"yd. es equivalente a: ",resultado,"mts."
        De Otro Modo:
            //Mensaje por defecto en caso de que la unidad de medida
            //proporcionada no sea compatible con la función.
            Imprimir "Unidad de medida ",unidadMedida,". no soportada"
    FinSegun
FinFuncion
```

Imagen 97: Proceso de actualización de una función externa - Fuente: Diseñada por el autor.

Hasta el momento, cada una de las funciones externas que se han diseñado dentro de este apartado realizan un cálculo, y luego, ellas mismas se encargan de presentar el resultado directamente en pantalla. Sin embargo, en la mayoría de las ocasiones las funciones son diseñadas con el propósito de generar un resultado y devolverlo de manera explícita al proceso principal para que se encargue de utilizarlo como parte de un nuevo procesamiento.

Bajo este mismo orden de ideas, la siguiente declaración muestra la sintaxis necesaria para retornar un valor desde la definición de una función:

```
Funcion variableDeRetorno = _NOMBRE(parametros)
    acciones a ejecutar
FinFuncion
```

Imagen 98: Estructura básica de una "Función Externa con Valor de Retorno" - Fuente: Diseñada por el autor.

Donde:

- **variableDeRetorno:** indica la variable (definida internamente) que se debe retornar al momento de invocar la función.
- **=:** indica que la función devuelve un valor, y éste se encuentra almacenado en la variable identificada como **variableDeRetorno**.

En atención a los supuestos anteriores, el siguiente Pseudocódigo digital muestra la definición de una función externa que retorna el resultado de convertir una cantidad expresada en grados Celsius a grados Fahrenheit:

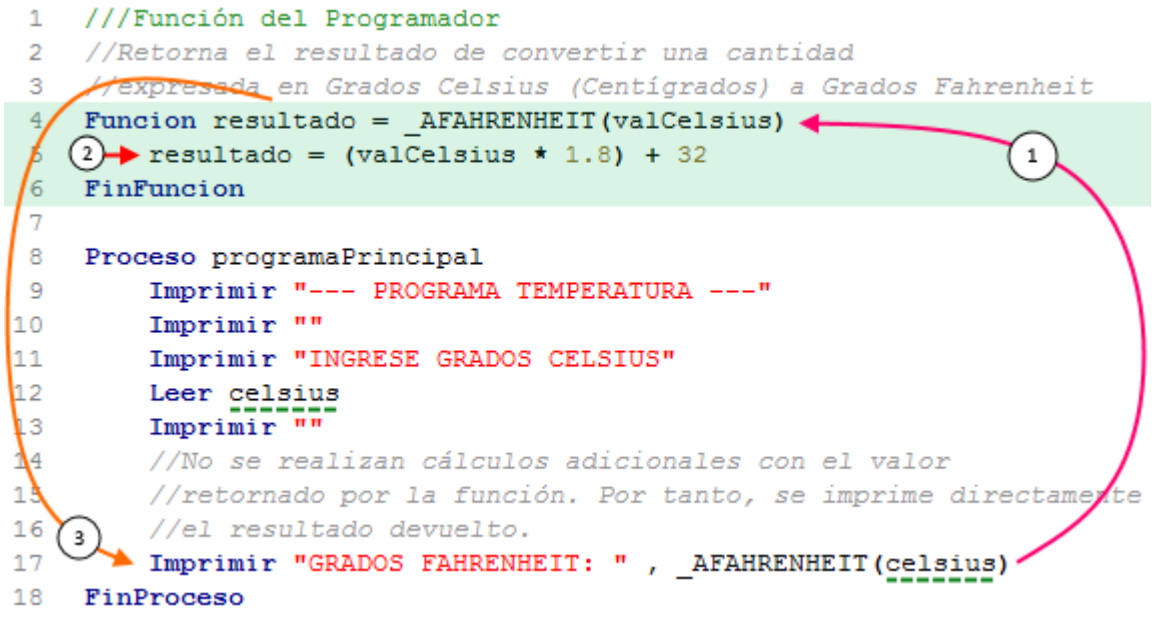


Imagen 99: Pseudocódigo digital que implementa la definición de una función externa con retorno de valor - Fuente: Diseñada por el autor.

Tal como se observa en el ejemplo anterior, la definición de la función **_AFAHRENHEIT** declarada en las líneas 4-6, recibe un solo argumento (**valCelsius**), mismo que es operado internamente (línea 5) para obtener su equivalente a grados Fahrenheit. Al finalizar, el resultado de la conversión es retornado directamente por la función (véase la imagen 100) y comisiona al proceso principal el compromiso de su captura y procesamiento.

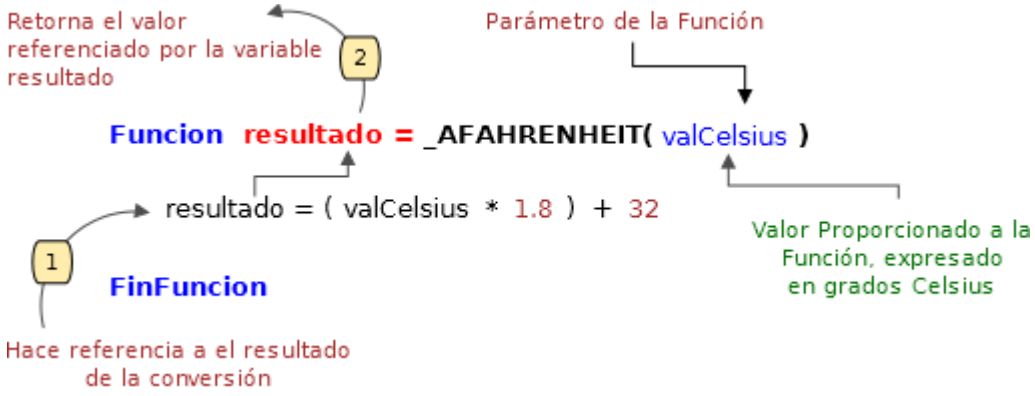


Imagen 100: Esquema de trabajo interno, función externa con retorno de valor - Fuente: Diseñada por el autor.

Aunado a la situación, en las líneas 11-12 correspondientes al proceso principal, se solicita al usuario el ingreso de una cantidad numérica expresada en grados Celsius. Como el objetivo del Pseudocódigo digital consiste en mostrar sólo el resultado de la conversión, la línea de código 17 imprime directamente en pantalla el resultado retornado por la función, sin que exista la necesidad de atrapar previamente su valor mediante el uso de una asignación, o se pase como argumento hacia otra función para un procesamiento adicional.

De este modo, una vez que se ejecuta el Pseudocódigo digital (**Tecla F9**), el comportamiento que se obtiene es parecido al que se muestra en la siguiente imagen:

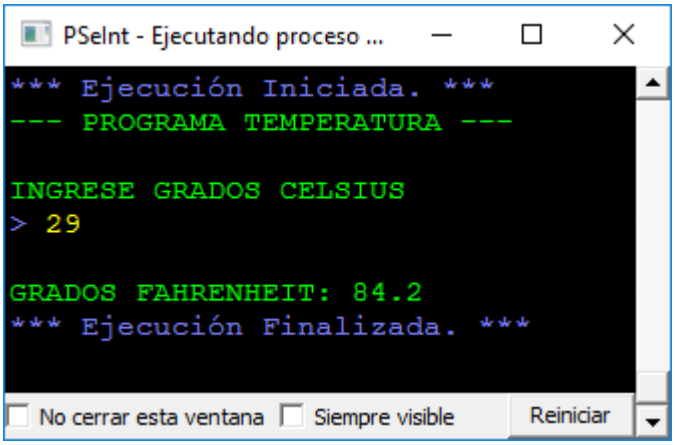


Imagen 101: Ejecución Pseudocódigo digital función externa con retorno de valor - Fuente: Diseñada por el autor.

Al llegar a este punto de la investigación, posiblemente surja la pregunta: ¿Qué pasa si una función hace uso de variables con el mismo nombre de identificador que en el proceso principal? Según lo explicado en un apartado anterior correspondiente a este mismo trabajo de investigación (véase el tema de las variables), la asignación de nuevos valores a una misma variable ocasiona que estos prevalezcan en lugar de los anteriores, es decir, se sobre-escriban. Sin embargo, este tipo de comportamiento sólo es válido si la asignación frecuente de valores (a una misma variable) se lleva a cabo dentro de un mismo ámbito. Dicho de otra forma, el valor asignado a una variable sólo se puede sobre-escribir si ésta se encuentra declarada en el mismo **Proceso** o **SubProceso** (Función externa).

Con la finalidad de ejemplificar las ideas expuestas hasta el momento, el siguiente Pseudocódigo digital define dos funciones externas para determinar si un cliente es acreedor o no a un premio. Se trata de una solución por computadora que genera de forma aleatoria un código participante (entre 1 y 2000) y lo asigna directamente a un cliente para que éste participe en la rifa de varios automóviles último modelo. Si el código participante es un número par (primera función) y el año de nacimiento del cliente coincide con año bisiesto (segunda función), el

cliente es ganador absoluto del premio (un automóvil último modelo); de lo contrario, se emite una leyenda que le exhorta a seguir participando.

```
1   ///Función del Programador
2   //Retorna VERDADERO si el número pasado como argumento es par
3   Funcion resultado = _ESPAR(valNumero)
4       Si valNumero % 2 == 0 Entonces
5           resultado = VERDADERO
6       Sino
7           resultado = FALSO
8       FinSi
9   FinFuncion
10
11  ///Función del Programador
12  //Retorna VERDADERO si el año pasado como argumento es bisiesto
13  Funcion resultado = _ESBISIESTO(valAnio)
14      Si valAnio % 4 == 0 Y (valAnio % 100 != 0 O valAnio % 400 == 0) Entonces
15          resultado = VERDADERO
16      Sino
17          resultado = FALSO
18      FinSi
19  FinFuncion
20
21  Proceso programaPrincipal
22      codigoGenerado = ALEATORIO(1,2000)
23      resultado = "Permiso SEGOB 20160709-145"
24      Imprimir "PROGRAMA CONCURSOS - PARTICIPANTE No. " , codigoGenerado
25      Imprimir ""
26      Imprimir "INGRESE NOMBRE DEL CLIENTE"
27      Leer nombre
28      Imprimir "INGRESE AÑO DE NACIMIENTO"
29      Leer anioNacimiento
30      Imprimir ""
31      Si _ESPAR(codigoGenerado) Y _ESBISIESTO(anioNacimiento) Entonces
32          Imprimir "** FELICIDADES: HA GANADO UN AUTOMÓVIL ***"
33      Sino
34          Imprimir "******* SIGA PARTICIPANDO *******"
35      FinSi
36      Imprimir ""
37      Imprimir resultado
38  FinProceso
```

Imagen 102: Declaración de una misma variable en diferentes ámbitos del Pseudocódigo digital - Fuente: Diseñada por el autor.

Como se ha podido observar, la línea 22 correspondiente al proceso principal genera de forma automática el código participante para dicho cliente (un número aleatorio dentro del rango 1 a 2000). Luego, en la línea

23 se crea una asignación a propósito para demostrar que el ámbito donde se encuentra declarada la variable "**resultado**", en este caso el **Proceso programaPrincipal**, es independiente al de las funciones **_ESPAR** y **_ESBISIESTO**. Posteriormente, en las líneas 25-28 se solicita al usuario el ingreso de los datos personales del cliente participante, es decir, su nombre completo y año de nacimiento. Más adelante, en la línea de código 30 se comprueba que el código participante sea de naturaleza par (mediante la función **_ESPAR**), y a su vez, que el año de nacimiento del cliente coincida con año bisiesto (función **_ESBISIESTO**). Si ambas condiciones resultan ser verdaderas, el Pseudocódigo digital emite el mensaje: "**FELICIDADES: USTED HA GANADO UN AUTOMÓVIL**" (línea 31); de lo contrario, muestra una leyenda que exhorta al cliente a seguir participando (línea 33).

Sumado a lo expuesto, se debe observar con detenimiento como ambas funciones externas "**_ESPAR** y **_ESBISIESTO**" son definidas para devolver un valor de tipo booleano tras su ejecución. Dicho valor se encuentra referenciado (en ambos casos) mediante el identificador de variable "**resultado**" que, al estar definido dentro del ámbito de una función diferente, se consigue que su valor asignado no entre en conflicto con el de los demás (por ejemplo, el **Proceso programaPrincipal**) y, por tanto, no se sobre-escriba.

Ahora bien, para determinar si un número es de naturaleza par. La función **_ESPAR** (líneas 3-9), internamente verifica que el valor pasado como argumento sea divisible entre dos (línea 4). En caso de que el resultado de la división sea un número entero (es decir, si el residuo o sobrante es idéntico a cero), se trata de un número par; en caso contrario, el valor pasado como argumento es impar.

Sin embargo, para decidir si un año es bisiesto o no, las condiciones se tornan un poco más complicadas. Para ello, es importante mencionar que un año bisiesto es aquel que tiene 366 días en lugar de 365 días normales, de modo que el 29 de febrero es el día agregado a dicho año. Esto sucede para corregir el desfase real de la duración de un año completo, que es de 365 días y 6 horas (un cuarto de día adicional). Tal y como se ilustra en la siguiente imagen:

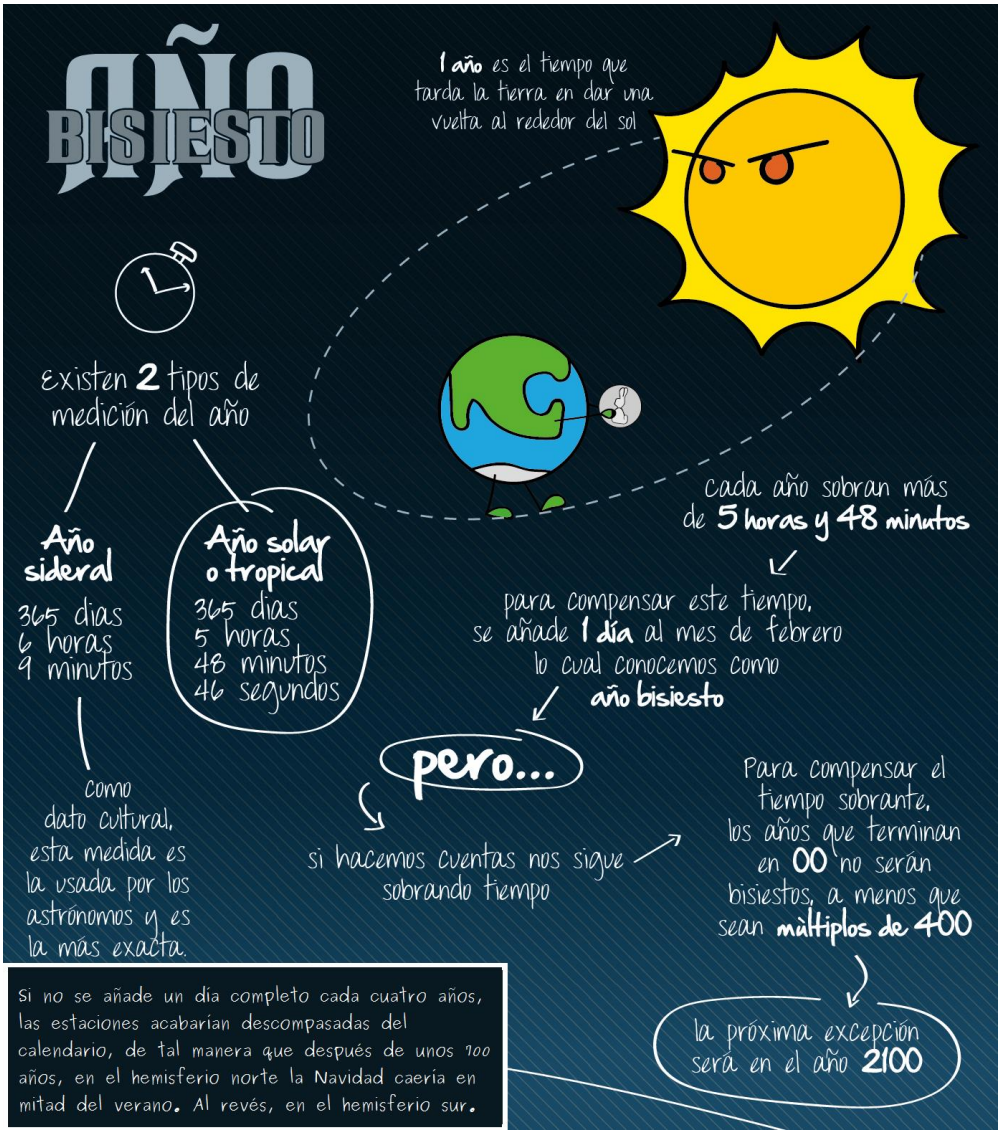


Imagen 103: Año bisiesto – Fuente: Adaptada por el autor - http://sipse.com/infografias/el-2016-es-anio-bisiesto-198.html

Bajo este mismo orden de ideas, la función **_ESBISIESTO** (líneas 13-19) internamente verifica que el valor proporcionado como argumento cumpla con cada una de estas características (línea 14), es decir, que el valor sea divisible entre 4 (año bisiesto cada cuatro años) y que, a su vez, sea diferente de un año secular (inicio de siglo, terminación 00), o que sea divisible entre 400 (de los cuatro años seculares, solo uno es bisiesto). En caso de que las condiciones resulten ser verdaderas, el valor proporcionado corresponde a un año bisiesto; de lo contrario, se trata de un año normal.

Con la finalidad de validar las ideas expuestas, la siguiente captura de pantalla ilustra el comportamiento del Pseudocódigo digital tras su ejecución:

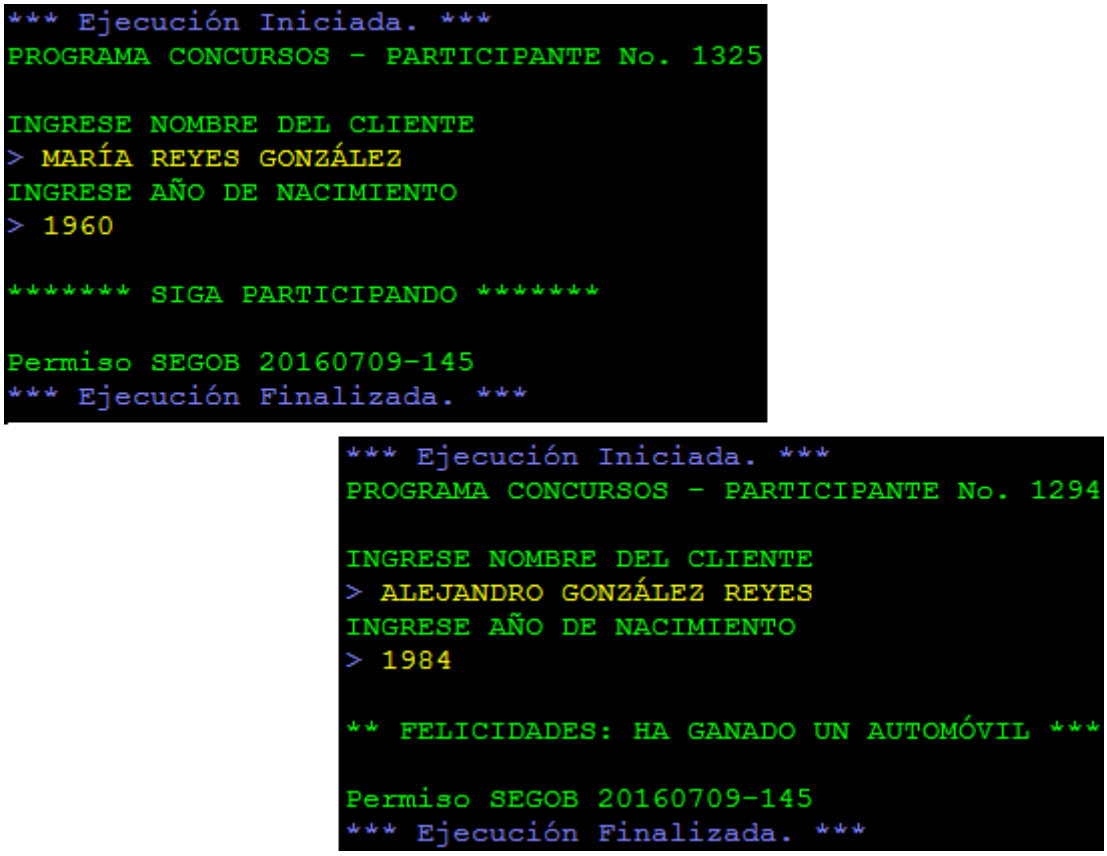


Imagen 104: La sobre-escritura de información sólo es posible si las variables con el mismo nombre se encuentran declaradas dentro de un mismo ámbito - Fuente: Diseñada por el autor.

Como resultado de la ejecución, es posible apreciar que en ambos casos se genera un código de cliente participante diferente (un número par y otro impar), además el año de nacimiento reportado por cada cliente coincide con año bisiesto. Sin embargo, sólo uno de ellos puede ser acreedor del premio, ya que, según las reglas establecidas para el concurso, el código participante debe ser de naturaleza par y el año de nacimiento debe coincidir con año bisiesto.

Dado lo anterior, también es posible verificar como el mensaje original asignado a la variable "**resultado**" (Permiso SEGOB 20160709-145) se muestra sin cambio alguno al finalizar la ejecución del Pseudocódigo digital. Esto es así, debido a que su declaración se encuentra en un ámbito

diferente (proceso principal) al de las dos funciones externas y, por tanto, no es posible sobre-escribir su valor.

Llegado a este punto, los ejemplos que se han abordado dentro de este apartado deberían proporcionar el contexto suficiente para reducir el código duplicado escrito en cada uno de los Pseudocódigos digitales, de tal forma que la solución propuesta al problema, resulte ser legible, eficiente y fácil de mantener.

Tal como se ha comentado en la sección anterior, en la mayoría de las ocasiones los lenguajes de programación incorporan funciones predefinidas que permiten atender dificultades clásicas durante la fase de desarrollo de un Pseudocódigo digital. Por ejemplo: una raíz cuadrada, una función trigonométrica, etc. Sin embargo, como cada problemática es diferente, es imposible que un lenguaje de programación cubra todas las necesidades de manera predeterminada. En este sentido, las funciones externas permiten al programador agrupar aquellas instrucciones de uso exclusivo bajo un contenedor, que más adelante puede ser invocado en distintas partes del proceso de solución a medida que se necesite.

Bajo este mismo orden de ideas, es importante enfatizar que los programadores profesionales con frecuencia acostumbran a coleccionar sus funciones externas con la firme intención de reutilizarlas o adaptarlas en diferentes proyectos de desarrollo, de esta manera su biblioteca de funciones personalizadas crece y, contiene la base de conocimientos y experiencias que se necesitan para afrontar nuevos proyectos de desarrollo sin que exista la necesidad de volver a reinventar "parte de la solución general" al problema.

A manera de título ilustrativo, para desarrollar un Pseudocódigo digital que determine la suma de los números pares ingresados por un usuario, sería un grave error volver a escribir toda la lógica de programación necesaria para determinar si un número es de naturaleza par o impar. En su lugar, una decisión acertada sería reutilizar la función **_ESPAR** diseñada con anterioridad para implementarla como solución parcial al problema general (véase la imagen 105), es decir, utilizar el valor devuelto por la función y compararlo como parte de una expresión condicional para saber si el valor ingresado por el usuario se debe sumar o no.

```
11  Proceso nuevoProgramaPrincipal
12      Imprimir "Ingrese número de elementos: "
13      Leer elementos
14      sumaPares = 0
15      Para contador = 1 Hasta elementos Hacer
16          Imprimir "Ingrese valor numérico: "
17          Leer num
18          Si _ESPAR(num) Entonces
19              sumaPares = sumaPares + num
20          FinSi
21      FinPara
22      Imprimir "La suma de los números pares es: " , sumaPares
23  FinProceso
```



Imagen 105: Las funciones externas pueden reutilizarse o adaptarse en futuros proyectos - Fuente: Diseñada por el autor.



Resumen: Funciones del programador.

En esta sección se ha hecho un largo recorrido por uno de los conceptos más importantes de la programación procedural: **las funciones**.

Una función es un conjunto de instrucciones que se agrupan para realizar una tarea concreta y se pueden reutilizar fácilmente en diferentes partes del Pseudocódigo digital a medida que se necesite. Al definir una función no solo se reduce la cantidad de código duplicado que se debe escribir, también hace que el código sea más limpio, eficiente y fácil de mantener. En este sentido, al declarar parte de la lógica de un Pseudocódigo digital mediante el uso de una función, las tareas de mantenimiento o actualización se simplifican, puesto que son llevadas a cabo dentro de un solo lugar, lo que origina que los cambios realizados se reflejen en cada una de las partes donde la función es invocada, sin que exista la necesidad de poner en riesgo la integridad del proceso principal.

Como se ha podido observar, las **funciones del programador** pueden ser diseñadas de tres formas diferentes:

- **Sin parámetros:** Generalmente se utilizan para mostrar información constante (nombre del producto, empresa fabricante, número de versión, etc.) en diferentes partes del Pseudocódigo digital.
- **Con paso de parámetros:** Especialmente útiles cuando se desea pasar valores (provenientes desde el exterior) a una función para que genere un resultado diferente en cada ejecución.
- **Con valor de retorno:** Su utilidad radica en generar un resultado y devolverlo de manera explícita al proceso principal (desde donde fue invocada la función), para que se encargue de utilizarlo como parte de otros procedimientos.

Con esta lección se da por terminado el apartado correspondiente a los Aspectos de nivel intermedio presentes en el Pseudocódigo. En el siguiente capítulo se hace una aproximación directa al lenguaje de programación PHP utilizado frecuentemente por los programadores para el desarrollo de aplicaciones Web. La intensión de dicho capítulo consiste en demostrar al lector que los conocimientos adquiridos hasta el momento son de vital

importancia para aprender a modelar soluciones (Pseudocódigos digitales) que más adelante puedan ser migradas a un lenguaje de programación para que sean interpretadas directamente por un ordenador.

Conclusión

Después de haber analizado detalladamente los fundamentos gramaticales del Pseudocódigo mediante programación estructurada y funcional, merece la pena destacar que las implementaciones de estos dos tipos de paradigmas durante la fase de desarrollo, permiten que se modele soluciones a problemas complejos mediante el uso de una computadora. Puesto que, al dividir y encapsular parte de los procesos en estructuras de control y funciones, se pueden llegar a construir sistemas robustos de fácil entendimiento, pero, sobre todo sencillos de modificar.

Si bien es cierto que el desarrollo de Pseudocódigos digitales bajo un paradigma secuencial permite atender diferentes problemáticas del mundo real. Muchos de los problemas con los que se topa un programador en la práctica requieren de características de repetición de instrucciones con diferentes conjuntos de valores, tomar decisiones con base al cumplimiento de una o varias condiciones o, dividir la lógica del problema principal en subprocesos más sencillos.

En este sentido, a partir de los ejemplos analizados y las definiciones que hacen los distintos autores seleccionados para este capítulo de la investigación, se verifica la hipótesis que argumenta: los elementos gramaticales necesarios para aprender las bases fundamentales de la algoritmia computacional mediante programación estructurada y funcional son:

a) **Estructuras de Control (Selección y Repetición)**
b) **Funciones Primitivas del Lenguaje**
c) **Funciones del Programador**

Puesto que, como se pudo observar a lo largo de las líneas del presente capítulo, los conceptos de programación esenciales tales como identificadores, variables, entradas, salidas, tipos de datos, operadores, expresiones y comentarios; permiten que el usuario interactué con los Pseudocódigos digitales y, por consiguiente, estos procedan a realizar los cálculos necesarios para determinar una solución factible al problema. Sin embargo, con la inclusión de las diferentes estructuras de control en el código, es posible modificar el flujo de su ejecución y presentarlos ante los usuarios con un comportamiento más o menos inteligente y dinámico, debido a que pueden realizar diferentes acciones durante el proceso (por ejemplo: aplicar un descuento en la compra de ciertos artículos o repetir un determinado conjunto de instrucciones) a partir de la evaluación de una o varias condiciones lógicas.



Imagen 106: Comparativo Programación Secuencial y Programación Estructurada - Fuente: Diseñada por el autor.

Sumado a lo expuesto, tal y como se ha visto a lo largo de estos dos capítulos de la investigación, un Pseudocódigo digital puede estar compuesto de muchas instrucciones que trabajan en su conjunto para atender una determinada problemática o complementar una tarea. Mientras más complejo sea el problema, más extenso será el código que se tenga que escribir para solucionarlo. Esto puede ocasionar que los Pseudocódigos digitales se vuelvan difíciles de leer, comprender y mantener.

Para evitar esta situación, es impórtate que el problema principal se divida en subproblemas más sencillos, y a continuación dividir esos subproblemas

en otros más simples, hasta que los problemas más pequeños sean fáciles de resolver. De esta manera, cuando se tienen pequeños bloques de código agrupados por funcionalidad, es posible combinar (sumar) cada uno de sus resultados generados y proceder a determinar una solución factible que brinde una respuesta lógica al problema principal.

Bajo este mismo orden de ideas, es importante recordar que las funciones solo se diseñan una vez, pero pueden invocarse muchas veces a lo largo de la ejecución de un Pseudocódigo digital. En este sentido, al implementar este tipo de paradigma de programación en cada uno de los proyectos, ayuda a reducir la cantidad de código duplicado declarado en el cuerpo de los programas, y evita el tener que volver a escribir parte de una solución previamente generada (reutilice el código y no reinvente la rueda, simplemente mejórela).



Imagen 107: Esquema de una aplicación construida a partir de diferentes módulos agrupados en funciones - Fuente: Diseñada por el autor.

Llegado a este punto, con la firme intención de resumir y relacionar cada uno de los factores expuestos dentro de este capítulo, se propone el uso

del siguiente esquema como material de apoyo didáctico para la solución de problemas complejos mediante el uso de un ordenador.

CAPÍTULO 2

Fundamentos Gramaticales del Pseudocódigo mediante Programación Estructurada y Funcional.

1 Aspectos de nivel intermedio presentes en el Pseudocódigo

1.1 Para modificar el flujo de ejecución de las instrucciones escritas en un Pseudocódigo, se requiere utilizar:

PROGRAMACIÓN ESTRUCTURADA

- Estructuras de Control
  - Tipos
    - Selección
    - Repetición
  - Por el número de decisiones que pueden tomar:
    - Simples
    - Dobles
    - Anidadas
    - Múltiples
  - Por el número de repeticiones que pueden ejecutar:
    - Definidas
    - Indefinidas
  - Su combinación permite atender problemas complejos
  - Por el tipo de valor o expresión que evalúan para poder ejecutar sus instrucciones internas:
    - Lógica
    - Numérica
    - Carácter(es) / Numérica

1.2 Para agrupar un conjunto de instrucciones que permitan realizar una tarea concreta, se requiere definir:

- Funciones
  - Definidas por defecto
    - Primitivas
      - Clasificación PSeInt
        - Matemáticas
        - Caracteres
      - Ejemplos

RAIZ	raíz cuadrada	SUBCADENA	
SEN	seno	CONCATENAR	
COS	coseno	LONGITUD	
TAN	tangente	MAYUSCULAS	
REDON	redondear	MINUSCULAS	

  - Difinidas por el programador
    - Externas
      - Sin parámetros — Generan el mismo resultado en cada ejecución y se encargan de mostrarlo en pantalla.
      - Con parámetros — Generan un resultado diferente en cada ejecución y se encargan de presentarlo en pantalla.
      - Con valor de retorno — Generan un resultado (diferente en caso de recibir parámetros) y lo retornan de manera explícita al proceso principal.

PROGRAMACIÓN FUNCIONAL

Es posible combinar ambos paradigmas de programación para desarrollar Pseudocódigos digitales robustos fáciles de comprender y modificar.

Capítulo 3

Del Pseudocódigo al Lenguaje de Programación PHP

El presente capítulo tiene por objetivo generar una guía comparativa que exponga las diferencias y similitudes que existen en la sintaxis utilizada por el Pseudolenguaje PSeInt y el lenguaje de programación PHP.

Al ser PSeInt una herramienta pensada para asistir a los estudiantes en sus primeros pasos en programación, esta debe implementar un Pseudolenguaje configurable que pueda ser utilizado como primera toma de contacto para introducir conceptos básicos presentes en la mayoría de los Lenguajes de Programación reales; siendo para PHP aspectos comunes:

a) Identificadores
b) Variables y Constantes
c) Entradas y Salidas de Información
d) Tipos de Datos (Numéricos, Caracteres y Booleanos)
e) Tipos de Operadores (Aritméticos, Comparación y Lógicos)
f) Comentarios
g) Estructuras de Control (Selección y Repetición)
h) Funciones Primitivas del Lenguaje
i) Funciones del Programador

En este sentido, para abordar de manera puntual el objetivo que se persigue en dicho capítulo, en un primer momento se elabora una tabla comparativa que parte de los fundamentos gramaticales del Pseudolenguaje utilizado por PSeInt y su equivalente al lenguaje de programación PHP. La intensión, es proporcionar información puntual al lector de que programar no significa aprender a escribir programas mediante el uso de un lenguaje de programación como tal, más bien, consiste en saber escribir algoritmos a través de diferentes herramientas universales (diagramas de flujo o pseudocódigo), que más adelante le permiten migrar sus programas, tareas o procedimientos a cualquier lenguaje de programación formal.

La manera más sencilla para familiarizarse con un lenguaje de programación y aprender a estructurar cada una de sus instrucciones, es mediante la exposición de diferentes casos prácticos. Es por ello, que antes de finalizar con la redacción de este capítulo, se presenta al lector un apartado llamado "Soluciones de programación" el cual parte de un conjunto de problemáticas debidamente expuestas, mismas que son atendidas con el desarrollo de diversos algoritmos (con la ayuda de PSeInt), los cuales más adelante son codificados (migrados) a lenguaje de programación PHP.

Como lenguaje de programación, PHP es robusto y permite el desarrollo de aplicaciones Web a gran escala, su facilidad de uso contribuye a que sea posible combinarlo con otras tecnologías (tales como: HTML, CSS, JavaScript, MySQL, entre otras) para el desarrollo de diferentes proyectos. En este sentido, es importante aclarar que el presente capítulo no pretende ser una guía de apoyo para el desarrollo de sitios Web dinámicos, los ejemplos aquí tratados siguen la misma filosofía que los abordados en un capítulo anterior, es decir, solicitan algún tipo de información al

usuario, procesan dicha información y emiten una serie de resultados. Si bien es cierto que en un escenario de la vida real lo factible sería almacenar de forma persistente (en una base de datos) parte de la información que proporciona un usuario al programa (para consultarla más adelante); en este capítulo, se deja de lado este tipo de actividades, puesto que, al igual que las funciones predefinidas de PSeInt (véase el capítulo anterior), la presencia de cada una de estas características (conexión a bases de datos) depende del lenguaje de programación seleccionado (PSeInt no tiene soporte para conexión a bases de datos).

Para ayudarle a sacar el mayor partido al texto y saber dónde se encuentra en cada momento de las explicaciones, la mayoría de los ejemplos tratados en este capítulo, se presentan en un formato de captura de pantalla, siendo posible obtener su correspondiente código fuente (archivos PSeInt, HTML y PHP) a través de la siguiente <URL>.

3.1 Aspectos básicos presentes tanto en el Pseudocódigo como en el Lenguaje de Programación PHP

En los capítulos anteriores de esta investigación se abordó a detalle todo lo referente a los fundamentos gramaticales del Pseudocódigo mediante diferentes paradigmas de la programación, entre los que se encuentran: el modelo secuencial, el modelo estructurado y el modelo funcional.

Como es de su conocimiento, un Pseudocódigo es un lenguaje falso que permite describir las instrucciones declaradas en un Algoritmo para que puedan ser leídas fácilmente por un ser humano. Sin embargo, éstas no pueden ser interpretadas directamente por un ordenador, dado que no se encuentran codificadas previamente a un lenguaje de programación.

Si bien es cierto que los Pseudo-Intérpretes (como PSeInt) posibilitan a los programadores a capturar, ejecutar y probar sus Pseudocódigos digitales mediante el uso de un ordenador; su función principal radica en la simulación de instrucciones, es decir, toda instrucción declarada en el cuerpo de un proceso o subproceso (véase la imagen 108), es ejecutada e interpretada directamente por el Pseudo-Intérprete (dígase nuevamente PSeInt), mas no por el ordenador.

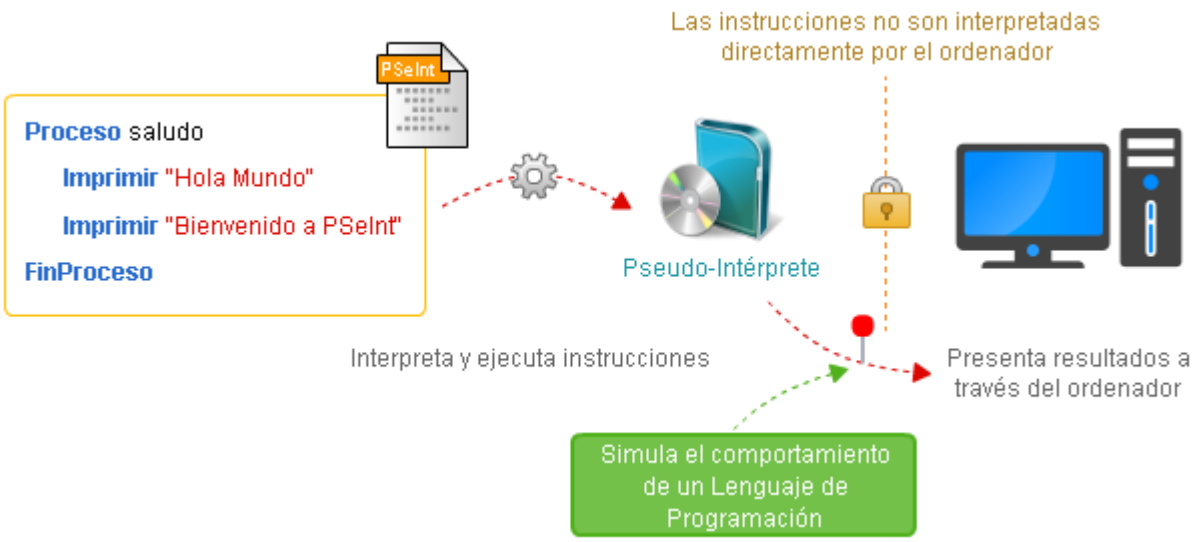


Imagen 108: Pseudo-Intérpretes como simuladores de instrucciones - Fuente: Diseñada por el autor.

En este sentido, el presente apartado expone a manera de tabla comparativa algunos de los elementos gramaticales presentes tanto en el Pseudocódigo como en el Lenguaje de Programación PHP, todo ello, a partir del paradigma de la programación secuencial. El objetivo es evidenciar al lector que los conocimientos adquiridos hasta el momento, permiten abordar de manera casi inmediata "cualquier" lenguaje de programación para la codificación de Algoritmos, puesto que como ya se ha mencionado al inicio de esta investigación (Capítulo 1), los Algoritmos son más importantes que los Lenguajes de Programación o las computadoras, debido a que un Lenguaje de Programación (por ejemplo PHP) es solo un medio para expresar un Algoritmo y una computadora es solo un procesador para ejecutarlo.

A continuación, la explicación de los aspectos básicos presentes tanto en el Pseudocódigo como en el Lenguaje de Programación PHP.

3.1.1 Identificadores: Variables y Constantes

Según lo comentado en el primer capítulo correspondiente a este trabajo de investigación, los identificadores consisten en los nombres que se les asignan aquellos espacios en Memoria RAM (variables, constantes) donde se almacena de forma temporal información útil para los programas.

En PHP los criterios que se consideran para declarar el nombre de un identificador son por poco parecidos a los estudiados para trabajar con el Pseudolenguaje de PSeInt. Sólo que, en PHP los identificadores utilizados para nombrar a las variables deben comenzar con el símbolo de dólar $, y los identificadores destinados para las constantes deben estar declarados en mayúsculas. Tal y como se muestra en la siguiente tabla:

	Pseudo-Intérprete	Lenguaje de Programación
		
Identificador para variables	precioProducto	$precioProducto
	descuento	$descuento
Identificador para constantes	NOMBRE_EMPRESA	NOMBRE_EMPRESA
	VERSION	VERSION

Tabla 14: Comparativo declaración de Identificadores - PHP vs PSeInt - Fuente: Elaborada por el autor.

Por otra parte, al igual que PSeInt, PHP considera un conjunto de palabras reservadas que no se pueden utilizar para la declaración de identificadores. Por ejemplo: ***print***, ***echo***, ***if***, ***else***, ***switch***, ***case***, ***do***,

***while*, *for*, *and*, *or*, *not*,** entre otras[14]. Para mayor información consulte el Anexo correspondiente a PHP en su apartado palabras y funciones reservadas del lenguaje.

3.1.2 Constantes

Como es de nuestro conocimiento, las constantes son un tipo de identificador especial que permiten almacenar un valor que no puede cambiar a lo largo de la ejecución de un programa. Es decir, a diferencia de las variables, sus valores permanecen fijos y por consiguiente no pueden sobre-escribirse.

Al igual que el Pseudolenguaje PSeInt, PHP cuenta de forma predeterminada con la definición de la constante PI como parte primitiva del lenguaje. Su declaración y diferencias son las que se ilustran a continuación mediante el uso de la siguiente tabla:

	Pseudo-Intérprete	Lenguaje de Programación
		
Constantes Predefinidas	PI	M_PI

Tabla 15: Comparativo Constantes Predefinidas - PHP vs PSeInt - Fuente: Elaborada por el autor.

[14] Por tratarse de palabras reservadas definidas en el lenguaje de programación PHP, es importante señalar que ésta investigación no sigue la regla de palabras extranjeras para cada uno de estos términos (estilo itálicas). En su lugar, se utiliza un estilo de fuente diferente compuesto por: negritas-itálicas.

Sin embargo, al tratarse de un lenguaje de programación real. PHP incorpora otro tipo de constantes matemáticas que pueden ser de gran ayuda para el desarrollo de diferentes proyectos. Si bien estas constantes no forman parte del objetivo principal que persigue este trabajo de investigación, usted puede aprender más sobre ellas al visitar la siguiente dirección electrónica:

- http://php.net/manual/es/math.constants.php

Por otra parte, es importante enfatizar que con respecto al almacenamiento de valores mediante el uso de constantes. PHP si cuenta con el soporte necesario para llevar a cabo este tipo de trabajos. Para ello, solo hace falta invocar a la función ***define( )*** en lugar del operador de asignación. Tal y como se muestra en la siguiente tabla:

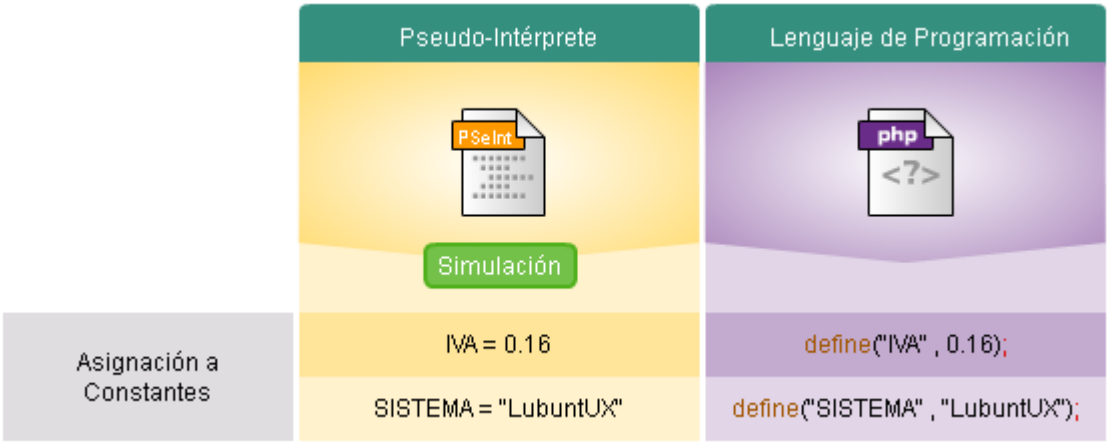


Tabla 16: Comparativo Asignación a Constantes - PHP vs PSeInt - Fuente: Elaborada por el autor.

En este sentido, PHP asigna a cada identificador un valor estático dentro del programa y previene que su contenido sea nuevamente definido (sobre-escrito) accidentalmente por el programador. Por ejemplo, si más adelante se modifica nuevamente el valor almacenado en la constante IVA

por ***define("IVA", 0.13)***, PHP hace caso omiso de los cambios, pero notifica al programador de lo sucedido a través de un mensaje apropiado por pantalla: *Notice: Constant IVA already defined* (Aviso: la constante IVA ya ha sido definida).

Finalmente, para invocar el contenido almacenado en las constantes, el procedimiento es el mismo que el utilizado en PSeInt para invocar el valor de la constante predefinida **PI**, es decir, se escribe el nombre del identificador en aquellos lugares donde sea requerido:

	Pseudo-Intérprete	Lenguaje de Programación
Invocar Constantes	impuestos = subtotal * IVA	$impuestos = $subtotal * IVA;

Tabla 17: Comparativo Invocar Constantes - PHP vs PSeInt - Fuente: Elaborada por el autor.

Si desea obtener más información acerca de la definición de constantes en PHP, sírvase en visitar el siguiente enlace:

- http://php.net/manual/es/language.constants.php

3.1.3 Entrada de Información

Como es de su conocimiento, las instrucciones de entrada se utilizan para capturar los datos que provienen del exterior, mismos que son asignados

en variables que permiten su posterior localización dentro del programa o aplicación.

PHP es un lenguaje de programación originalmente diseñado para el desarrollo Web. En este sentido, es común que la mayoría de los usuarios introduzcan valores en el programa a través de diversos controles de formulario definidos en un archivo HTML. Si bien el tema de HTML no forma parte del objetivo principal de esta investigación, a continuación, se ilustra el proceso que se debe llevar a cabo para recuperar la información enviada por dos campos de formulario HTML a un archivo PHP.

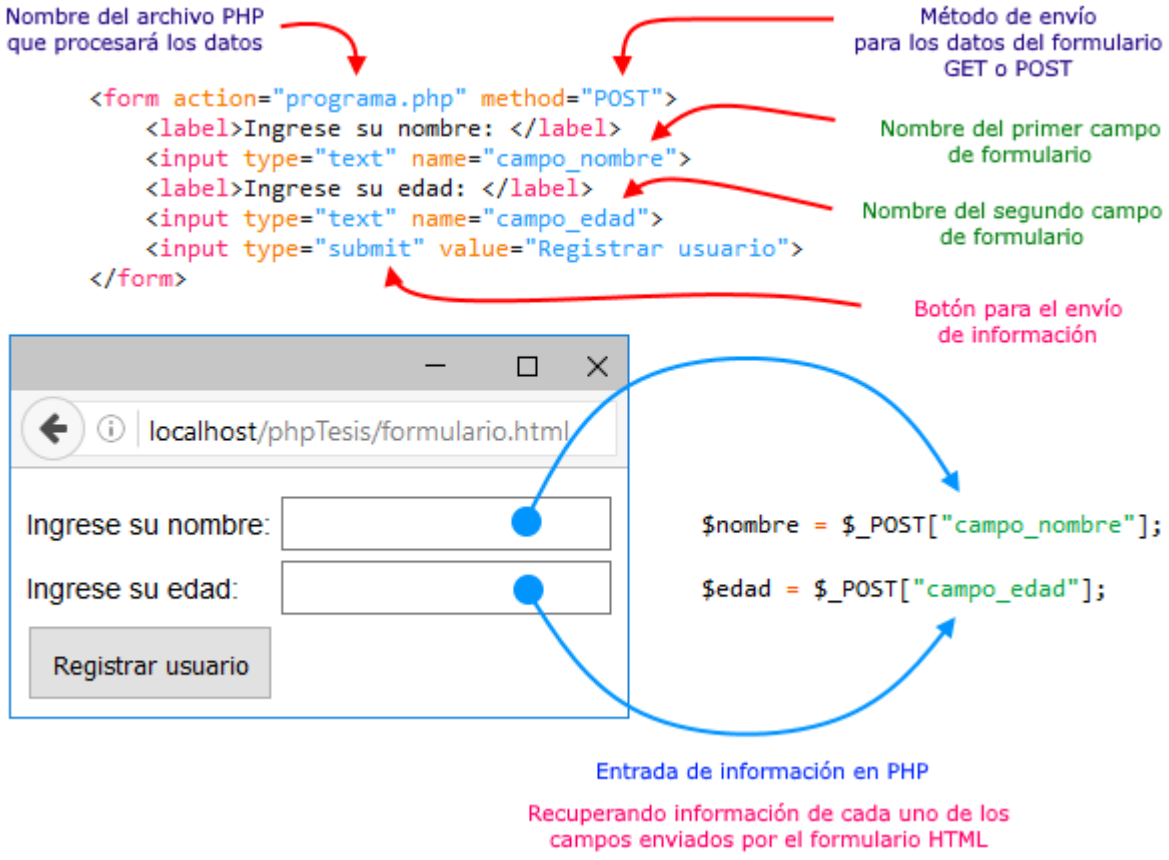


Imagen 109: Esquema para recuperar información enviada desde un formulario HTML a PHP - Fuente: Diseñada por el autor.

La definición de un formulario HTML se encuentra delimitado por la etiqueta de apertura ***<form>*** y la etiqueta de cierre ***</form>***. Para el ingreso de información, internamente puede estar construido a partir de uno o más controles, tales como: campos de texto, cajas de selección (menús), botones de envío, casillas de verificación, botones de radio, así como etiquetas para describir el propósito que cumple cada uno de estos controles dentro de dicho formulario.

Ahora bien, para configurar su comportamiento es necesario especificar algunos atributos opcionales, siendo los más indispensables el atributo ***action*** y el atributo ***method*****,** mismos que se corresponden con el nombre del archivo PHP que se encarga de procesar la información, y el método HTTP que se utiliza para él envió de dicha información (***GET*** o ***POST***)[15].

Por otra parte, para enviar correctamente la información hacia el archivo de programa PHP, es necesario que se asigne un nombre a cada control de formulario encargado de recolectar una parte específica de esa información (mediante el atributo ***name***); ya que, de no hacerlo, se corre el riesgo de que un control de formulario "sin nombre" no envíe sus correspondientes datos y, por consiguiente, estos no se encuentren disponibles al momento de su procesamiento.

Finalmente, para recuperar cada una de las partes de información enviadas por los controles de formulario HTML. En PHP es necesario declarar una variable y asignarle uno de los dos arreglos asociativos disponibles: ***$_GET*** o ***$_POST***; e indicar internamente el nombre del

[15] Por tratarse de palabras comúnmente empleadas en el lenguaje informático HTML para designar el nombre de etiquetas, atributos o valores. Es importante recordar nuevamente al lector, que ésta investigación no sigue la regla de palabras extranjeras para cada uno de estos términos (estilo itálicas). En su lugar, se utiliza un estilo de fuente diferente compuesto por: negritas-itálicas.

control de formulario que contiene parte de la información requerida. Por ejemplo:

- ***$_POST[*"*edadUsuario*"*]***: Permite obtener la información capturada en el control de formulario "**edadUsuario**" enviada por el método ***POST***.
- ***$_GET[*"*campo_autor*"*]***: Permite obtener la información ingresada en el control de formulario "**campo_autor**" enviada por el método ***GET***.

Llegados a este punto, como se puede observar el proceso de entrada de información en PHP difiere por completo con respecto a PSeInt. Sin embargo, es importante puntualizar que en ambos casos es necesario utilizar una variable para el almacenamiento de dicha información. Tal y como se muestra en la siguiente tabla comparativa:

	Pseudo-Intérprete	Lenguaje de Programación
		
Formulario enviado por método POST	Leer base	$base = $_POST["campo_base"];
	Leer altura	$altura = $_POST["campo_altura"];
Formulario enviado por método GET	Leer numA	$numA = $_GET["campo_numA"];
	Leer numB	$numB = $_GET["campo_numB"];

Tabla 18: Comparativo Entrada de Información - PHP vs PSeInt - Fuente: Elaborada por el autor.

A diferencia de PSeInt, el lenguaje de programación PHP no cuenta con la palabra reservada **Leer** para realizar un análisis previo de los datos ingresados por el usuario. En su lugar éste hace uso de los arreglos asociativos ***$_GET*** o ***$_POST*** para acceder a los datos capturados en cada campo de formulario (véase la imagen 110), y al final, lo asigna a una variable para su futuro procesamiento.

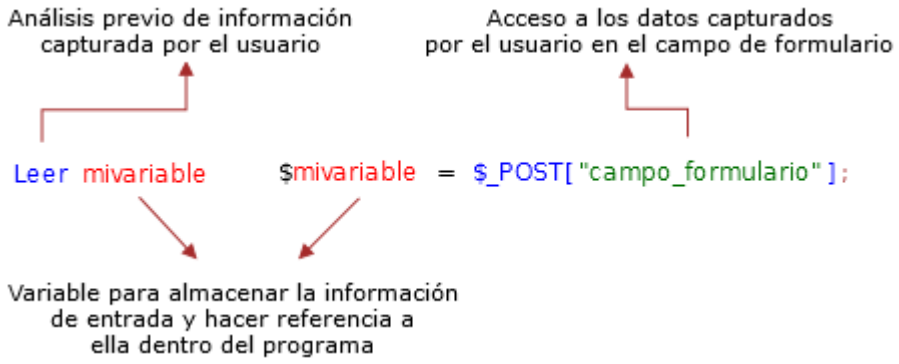


Imagen 110: Comparativo Lectura de Información - PHP vs PSeInt - Fuente: Diseñada por el autor.

Para obtener más información acerca del trabajo con entradas de información desde fuentes externas (formularios HTML) en PHP. Sírvase en visitar el siguiente enlace:

- http://php.net/manual/es/language.variables.external.php

3.1.4 Salida de Información

Con base en lo comentado en el primer capítulo de esta investigación, las salidas de información permiten presentar en pantalla el contenido almacenado en una variable o constante, así como comentarios, y los valores devueltos tras el procesamiento de información mediante el uso de una función o expresión.

A diferencia de PSeInt, el lenguaje de programación PHP emplea la palabra reservada ***echo*** para mostrar el contenido de los objetos (variables, constantes, funciones, expresiones, leyendas o comentarios). Además, cada instrucción de salida debe finalizar con un "punto y coma", tal y como se muestra en la siguiente tabla comparativa:

	Pseudo-Intérprete	Lenguaje de Programación
		
Salidas de Información	Imprimir precioProducto	echo $precioProducto;
	Imprimir "Hola desde PSeInt"	echo "Hola desde PHP";
	Imprimir PI	echo M_PI;

Tabla 19: Comparativo Salida de Información - PHP vs PSeInt - Fuente: Elaborada por el autor.

Por otra parte, al igual que PSeInt, PHP cuenta con otro tipo de instrucciones alternativas para mostrar información por pantalla, siendo ***print***, una de las instrucciones también comúnmente utilizadas.

```
print "Resultados del procesamiento de información";
print $areaTriangulo;
print MI_CONSTANTE;
```

Imagen 111: Instrucción "print" definida en PHP - Fuente: Diseñada por el autor.

Sin embargo, con la finalidad de estandarizar las salidas de información correspondientes a cada uno de los ejemplos tratados a lo largo de este

capítulo, se empleará la palabra reservada ***echo*** para tal propósito. Ya que permite concatenar diferente contenido de salida mediante el uso del signo coma, tal y como se ilustra en la siguiente tabla:

	Pseudo-Intérprete	Lenguaje de Programación
		
Concatenar información	**Imprimir** "Precio: " , precioProducto	**echo** "Precio: " , $precioProducto;

Tabla 20: Comparativo Concatenar Salidas de Información - PHP vs PSeInt - Fuente: Elaborada por el autor.

Como se puede observar, la instrucción ***echo*** permite hacer uso del signo coma para unir diferente tipo de contenido como si se tratase de una sola entidad de texto de salida. Algo que con la instrucción ***print***, sería prácticamente imposible de hacer.

Si desea obtener más información referente al trabajo con las instrucciones ***echo*** y ***print*** de PHP, sírvase en visitar los siguientes enlaces:

- http://php.net/manual/es/function.echo.php
- http://php.net/manual/es/function.print.php

3.1.5 Tipos de Datos

Como es de nuestro conocimiento, los tipos de datos consisten en el tipo de información que admite un lenguaje de programación para su correspondiente procesamiento.

En PHP se consideran los mismos tipos de datos que los utilizados para trabajar en PSeInt, es decir, se tiene soporte para números enteros, números reales, carácter, cadena de caracteres, valores lógicos y arreglos. Sin embargo, al tratarse de un lenguaje de programación diseñado especialmente para la Web, PHP agrega otro tipo de datos especiales (los cuales no forman parte del objetivo de esta investigación) que se consideran indispensables para el desarrollo de aplicaciones avanzadas. Por ejemplo:

- **Recursos:** Es un tipo de dato especial que hace referencia a un recurso externo, tales como: una conexión a una base de datos, un archivo de texto para tareas de lectura y escritura, una conexión mediante protocolo FTP, una conexión a un servidor de correo electrónico, entre otras.
- ***Null*:** Es un tipo de dato especial que indica ausencia de valor en una variable.
- **Objetos:** Asociados directamente con el paradigma de programación orientado a objetos. Los objetos consisten en estructuras de datos complejos que se caracterizan por tener una serie de propiedades y métodos propios que les conceden un comportamiento específico.

Con base en los supuestos anteriores, para asignar cada uno de estos tipos de datos a una variable desde la perspectiva del lenguaje de

programación PHP, la siguiente tabla comparativa ilustra una serie de ejemplos prácticos debidamente clasificados:

	Pseudo-Intérprete	Lenguaje de Programación
Datos de tipo Numérico	edadUsuario = 31	$edadUsuario = 31;
	estaturaUsuario = 1.75	$estaturaUsuario = 1.75;
	temperatura = -2	$temperatura = -2;
Datos de tipo Carácter y Cadena de Caracteres	nombreUsuario = "Alejandro"	$nombreUsuario = "Alejandro";
	inicialUsuario = "A"	$inicialUsuario = "A";
	placaVehiculo = "LJR-245B"	$placaVehiculo = "LJR-245B";
Datos de tipo Lógico	estaLogeado = VERDADERO	$estaLogeado = True;
	esAdministrador = FALSO	$esAdministrador = False;
Datos de tipo Arreglo	Dimension menu(4) menu[1] = "Bienvenidos" menu[2] = "Productos" menu[3] = "Servicios" menu[4] = "Acerca de"	$menu[0] = "Bienvenidos"; $menu[1] = "Productos"; $menu[2] = "Servicios"; $menu[3] = "Acerca de"; $menu[4] = 1985; $menu[5] = 1.72;

Tabla 21: Comparativo Tipos de Datos - PHP vs PSeInt - Fuente: Elaborada por el autor.

De la tabla anterior, se puede llegar a vislumbrar que tanto PSeInt como PHP coinciden en el tratamiento que se les da a los valores de tipo numérico, carácter y cadena de caracteres. Sin embargo, para los valores de tipo lógico y arreglo, surgen las siguientes diferencias:

1. PHP considera los valores lógicos **FALSO** y **VERDADERO** como ***True*** y ***False*** respectivamente.
2. En PHP no existe la necesidad de declarar previamente la dimensión de un arreglo.
3. La posición inicial de un arreglo en PHP empieza con el índice **0** y no en **1** como es el caso de PSeInt.
4. La dimensión de un arreglo en PHP crece a medida que se le necesite.
5. Los valores almacenados dentro de un arreglo en PHP pueden ser de cualquier tipo.

Por otra parte, al igual que las instrucciones de salida, PHP considera el final de una asignación mediante el uso de un punto y coma. Considere esto como una regla primordial al momento de trabajar con PHP, ya que, de no hacerlo, usted invertirá la mayoría de su tiempo en resolver problemas de sintaxis.

Para obtener más información referente al trabajo con los datos de tipo **arreglo**, **recurso**, ***null*** y **objeto**. Sírvase en visitar los siguientes enlaces correspondientes a la documentación de PHP:

- http://php.net/manual/es/language.types.array.php
- http://php.net/manual/es/language.types.resource.php
- http://php.net/manual/es/language.types.null.php
- http://php.net/manual/es/language.types.object.php

3.1.6 Operadores Aritméticos

Según lo comentado con anterioridad, los operadores aritméticos dentro de un lenguaje de programación permiten al desarrollador realizar cualquier tipo de operación (matemática básica) entre dos operandos.

Al igual que PSeInt, PHP considera los mismos tipos de operadores para el trabajo con cálculos matemáticos sencillos. Sin embargo, para el caso de tareas correspondientes a la exponenciación, en lugar de utilizar el acento circunflejo (**^**), PHP hace uso de la función predefinida ***pow( )***. Tal y como se ilustra en la siguiente tabla comparativa:

	Pseudo-Intérprete	Lenguaje de Programación
Suma	numA + numB	$numA + $numB
Resta	2016 - 30	2016 - 30
Multiplicación	subtotal * 0.16	$subtotal * 0.16
División	sumaCalificaciones / 2	$sumaCalificaciones / 2
Potencia	base ^ 4	pow($base , 4)
Módulo	numA % numB	$numA % $numB

Tabla 22: Comparativo Operadores Aritméticos - PHP vs PSeInt - Fuente: Elaborada por el autor.

Observe nuevamente la invocación de la función ***pow( )*** en PHP, si bien esta recibe dos parámetros (**base** y **exponente**) para poder trabajar,

curiosamente su sintaxis de llamada es la misma que se utiliza dentro del ambiente del Pseudolenguaje PSeInt.

Si desea obtener más información referente al trabajo con operadores aritméticos en PHP, sírvase en visitar el siguiente enlace:

- http://php.net/manual/es/language.operators.arithmetic.php

3.1.7 Operadores de Comparación

Con base en lo comentado en el capítulo primero, los operadores de comparación permiten al usuario cotejar dos operandos y generar como resultado un valor de tipo lógico.

Al igual que el Pseudolenguaje PSeInt, PHP brinda soporte completo al programador para el trabajo con operadores de comparación. Sin embargo, la sintaxis utilizada para definir una comparación de tipo **distinto de** o **idéntico a**, varía con respecto a la suministrada por PSeInt. Tal y como se muestra en la siguiente tabla:

	Pseudo-Intérprete	Lenguaje de Programación
		
Mayor que	numA > numB	$numA > $numB
Menor que	1985 < 27	1985 < 27
Mayor o igual que	ventas >= meta	$ventas >= $meta
Menor o igual que	promedio <= 6	$promedio <= 6
Distinto de	nombreUsuario != "José"	$nombreUsuario !== "José"
Idéntico a	opcionUsuario == "Imprimir"	$opcionUsuario === "Imprimir"

Tabla 23: Comparativo Operadores de Comparación - PHP vs PSeInt - Fuente: Elaborada por el autor.

El signo de asignación (=) adicional agregado a la regla comparativa **distinto de** o **idéntico a**, permite a PHP verificar que los valores comparados sean del mismo tipo, es decir, si por algún motivo se llegara a comparar los valores **"17"** y **17**. PHP determinaría que se trata de valores idénticos en cuanto a valor, pero diferentes en cuanto a tipo o naturaleza de información, puesto que **"17"** (encerrado entre comillas) se trata de un valor de tipo cadena, y el valor **17** (sin comillas) se trata de un valor de tipo numérico.

Ahora bien, si lo que le interesa es comparar el contenido y dejar de lado el tipo de información, usted puede utilizar el mismo tipo de operadores comparativos que conoce hasta el momento.

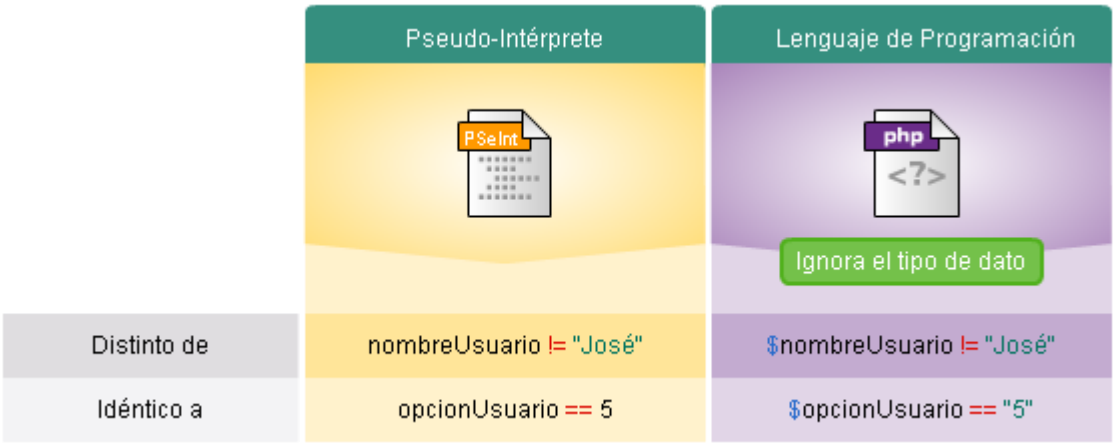


Tabla 24: Operadores de Comparación - Ignorar el Tipo de Dato y Centrarse en el Valor - Fuente: Elaborada por el autor.

Sin embargo, recuerde nuevamente que, al hacer uso de este tipo de comparación, PHP centra su atención en el valor, mas no en su tipo.

Para obtener mayor información acerca del trabajo correspondiente a los operadores de comparación en PHP, sírvase en visitar su documentación a través de la siguiente dirección:

- http://php.net/manual/es/language.operators.comparison.php

3.1.8 Operadores Lógicos

Según lo comentado en el primer capítulo de esta misma investigación, los operadores lógicos permiten generar un valor booleano (**FALSO** o **VERDADERO**) como resultado de comparar dos o más condiciones.

Al igual que PSeInt, PHP ofrece al programador un conjunto de operadores lógicos listos para implementar en una condición. Sin embargo, la sintaxis utilizada difiere por completo con respecto a la de PSeInt. Tal y como se muestra en la siguiente tabla:

	Pseudo-Intérprete	Lenguaje de Programación
		
Y (Conjunción)	ventas > 1200 Y dias >= 13	$ventas > 1200 && $dias >= 13
O (Disyunción)	edad > 18 O estado == "Casado"	$edad > 18 || $estado == "Casado"
NO (Negación)	NO estaLogeado	!$estaLogeado

Tabla 25: Comparativo Operadores Lógicos - PHP vs PSeInt - Fuente: Elaborada por el autor.

Los significados y resultados generados son los mismos, solo que en PHP el **operador de conjunción** se declara mediante dos signos *ampersand* (**&&**), el **operador de disyunción** es representado por dos barras verticales (**||**), y el **operador de negación** se simboliza mediante el signo de exclamación de cierre (**!**).

Si desea profundizar más acerca del trabajo referente a los operadores lógicos en PHP, sírvase en visitar el siguiente enlace:

- http://php.net/manual/es/language.operators.logical.php

3.1.9 Comentarios

Como es de su conocimiento, los comentarios consisten en un mecanismo presente en la mayoría de los lenguajes de programación que permite a los programadores documentar aquellas líneas de código sobresalientes de sus programas, con la finalidad de que sirvan como referencia para otros

programadores participes del proyecto, o como notas aclaratorias para futuras actualizaciones.

Al igual que PSeInt, en PHP se consideran los mismos criterios para la declaración de comentarios de una sola línea, es decir, las dos barras diagonales. Sin embargo, al tratarse de un lenguaje de programación real diseñado para el desarrollo de aplicaciones Web avanzadas, PHP ofrece al programador una alternativa adicional para la declaración de comentarios multi-línea, es decir, para comentar grandes párrafos de texto. Tal y como se muestra en la siguiente tabla comparativa:

	Pseudo-Intérprete	Lenguaje de Programación
		
Comentario Simple	// Comentario de una sola línea	// Comentario de una sola línea
Comentario Resaltado	/// Comentario con otro color	
Comentario Multi-línea		/* Comentario que abarca varias líneas de código, especialmente útil si los comentarios abarcan grandes extensiones de texto */

Tabla 26: Comparativo Declaración de Comentarios - PHP vs PSeInt - Fuente: Elaborada por el autor.

Observe que para declarar un comentario multi-línea en PHP, este debe comenzar con los signos **/*** y finalizar con ***/**. La presencia de este tipo de comentarios en PHP va más allá de un simple gusto o por estética, puesto que con frecuencia (en el desarrollo profesional) se utilizan para

prevenir la ejecución de ciertos bloques de código que aún se encuentran en fases de mantenimiento o desarrollo.

Si desea obtener más información acerca del trabajo con comentarios en PHP, sírvase en visitar el siguiente enlace:

- http://php.net/manual/es/language.basic-syntax.comments.php

Como se puede observar, en términos generales la sintaxis de PHP es muy parecida a la que utiliza el Pseudolenguaje PSeInt para la solución de problemas mediante el uso de un ordenador. Por tanto, todos aquellos lectores que estén familiarizados con el diseño y desarrollo de Algoritmos mediante la técnica de Pseudocódigo, encontrarán una gran facilidad a la hora de seguir la estructura sintáctica de las instrucciones y sentencias que presenta este lenguaje de programación.

Existen muchos factores que intervienen a la hora de desarrollar una aplicación exitosa, y uno de ellos es saber con precisión cuál es la mejor opción disponible para llevar a cabo una determinada tarea. Es por ello que antes de comenzar a programar en cualquier lenguaje de programación (dígase PHP), es necesario conocer los detalles básicos de su sintaxis. Por ejemplo: la declaración de variables, constantes, los diferentes tipos de datos soportados, las entradas y salidas de información, el tipo y uso de los operadores, así como la redacción de comentarios para la documentación (a nivel de código) de los programas.

Sin duda los temas tratados en este numeral son de propósito general, lo que significa que pueden ser aplicados en distintas clases de proyectos. Tal vez sea necesaria una mínima experiencia o conocimiento del lenguaje informático HTML, pero no es algo determinante para los objetivos que

persigue esta investigación; dado que su intervención, es sólo para fines de recolección de información a través de los distintos controles de formulario presentes en la mayoría de los sitios Web.

Una vez comentado y comparado cada uno de los elementos gramaticales presentes tanto el Pseudocódigo como en el Lenguaje de Programación PHP, el siguiente numeral tiene como propósito dar cabida a las diferentes estructuras de control y funciones presentes en este lenguaje para el desarrollo de programas de computadora "relativamente complejos" bajo el paradigma de la programación estructurada y funcional. No obstante, con la finalidad de resumir cada uno de los factores expuestos hasta el momento (dentro de este mismo apartado), se propone el uso del siguiente esquema (*cheatsheet*) como material de referencia correspondiente a la sintaxis de algunos elementos gramaticales PHP existentes.

Programación Web



Hoja de Referencia Rápida - Programación Secuencial con PHP

Variables y Constantes

`$var = valor;`	Variable
`define(cons,valor);`	Constante

Constantes Predefinidas

`M_PI`	Valor de PI

Entrada de Información

`$_POST["campo_f"]`	Método POST
`$_GET["campo_f"]`	Método GET

Salida de Información

`echo "Hola Mundo";`	Imprimir texto
`echo $miVar;`	Imprimir variable
`echo M_PI;`	Imprimir constante

Concatenar Salidas de Info.

`echo "IVA ",$var;`	Unir texto y variable
`echo $v1,$v2,$v3;`	Unir variables

Comentarios

`// descuento`	Línea simple
`/* facturas */`	Múltiples líneas

Tipos de Datos

`$var = 9.5;`	Numéricos
`$var = '@';`	Carácter
`$var = "PHP 7";`	Cadena de Caracteres
`$var = True|False;`	Lógicos
`$ar[0] = "Dania";` `$ar[1] = "María";`	Arreglo

Operadores Aritméticos

`$var1 + $var2`	Suma
`$var1 - $var2`	Resta
`$var1 * $var2`	Multiplicación
`$var1 / $var2`	División
`pow($var,4)`	Potencia
`$var1 % $var2`	Módulo

Operadores de Comparación

`$var1 > $var2`	Mayor que
`$var1 < $var2`	Menor que
`$var1 >= $var2`	Mayor o igual que
`$var1 <= $var2`	Menor o igual que
`$var1 != $var2`	Distinto de
`$var1 == $var2`	Idéntico a

Operadores Lógicos

`$var1 && $var2`	Y (conjunción)
`$var1 || $var2`	O (disyunción)
`!$var`	NO (negación)

3.2 Aspectos de nivel intermedio presentes tanto en el Pseudocódigo como en el Lenguaje de Programación PHP

Según lo comentado en el capítulo anterior de esta misma investigación, no todos los problemas que se nos plantean tienen una solución basada en la ejecución secuencial de instrucciones, por lo que es necesario que los Lenguajes de Programación incorporen diferentes herramientas que les permitan adaptarse a las diferentes situaciones o condiciones que se puedan presentar a la hora de intentar resolver un problema.

Bajo este mismo orden de ideas, las estructuras de control nos permiten modificar el flujo de ejecución de los programas, de modo que estos puedan ser presentados ante los usuarios con un comportamiento más o menos inteligente, debido a que pueden realizar diferentes acciones con base a uno o varios resultados generados a través de una serie de comparaciones lógicas.

En este sentido, PHP al ser un Lenguaje de Programación formal destinado para el desarrollo de aplicaciones Web, cuenta con diferentes estructuras de control similares en cuanto concepto a las que presentan la mayoría de sus competidores (Python, Ruby, Visual Basic .NET), y casi idénticas a las que vienen integradas por defecto en lenguajes como JavaScript, C++, Java y el **Pseudolenguaje PSeInt**.

Sumado a lo expuesto, PHP también cuenta con un amplio conjunto de funciones pre-construidas listas para ser implementadas en la mayoría de los proyectos de desarrollo (cálculos matemáticos, trabajo con cadenas de

caracteres, conexión a bases de datos, etc.). Sin embargo, para casos específicos; al igual que PSeInt, PHP ofrece la posibilidad a los programadores de definir sus propias funciones, de modo que puedan dividir el código en bloques más pequeños agrupados por funcionalidad, los cuales a futuro sean más sencillos, legibles y fáciles de mantener.

De acuerdo con esto, el presente apartado expone de manera puntal un comparativo de los elementos gramaticales presentes tanto en el Pseudocódigo como en el Lenguaje de Programación PHP desde la perspectiva de la programación estructurada y funcional. El objetivo es demostrar al lector que un conocimiento sólido sobre el funcionamiento de cada uno de estos elementos (independientemente del lenguaje) permite que los desarrolladores escriban programas de computadora cada vez más complejos, cuyos requerimientos estén relacionados directamente con la toma de decisiones, la ejecución continua de instrucciones, y/o la simplificación de código mediante la invocación o construcción de diferentes bloques de instrucciones agrupados por funcionalidad.

A continuación, la segunda parte de los elementos gramaticales presentes tanto en el Pseudocódigo como en el Lenguaje de Programación PHP.

3.2.1 Estructura de Selección Si Entonces

Como se comentó en un capítulo anterior, la estructura de selección "**Si Entonces**" evalúa una determinada condición y en caso de ser verdadera, se ejecuta un conjunto de instrucciones. Si dicha condición no se cumple, ninguna de las instrucciones es ejecutada.

En PHP se dispone de una estructura similar para llevar a cabo este tipo de tareas. Sin embargo, la sintaxis que se emplea para su declaración difiere completamente de la utilizada por PSeInt (véase la tabla 27), mas no es así su funcionamiento.

	Pseudo-Intérprete	Lenguaje de Programación
		

Selección Simple	Si condición Entonces
	instrucciones
	FinSi
		if (condición) {
		instrucciones
		}

Tabla 27: Comparativo Estructura de Selección Simple - PHP vs PSeInt - Fuente: Elaborada por el autor.

Como se puede observar, la estructura de selección "**Si Entonces**" es identificada por PHP mediante la palabra reservada ***if***. La condición o condiciones que se evalúan deben ir delimitadas entre paréntesis. La llave de apertura "**{**" sustituye al **Entonces** de PSeInt, y la llave de cierre "**}**" finaliza la declaración de la estructura ***if*** (reemplaza al **FinSi** de PSeInt).

Si desea obtener más información referente al trabajo con la estructura de selección ***if*** de PHP, sírvase en visitar el siguiente enlace:

- http://php.net/manual/es/control-structures.if.php

3.2.2 Estructura de Selección Si Entonces Sino

Según lo comentado en un capítulo anterior, la estructura de selección "**Si Entonces Sino**" consiste en una variante de la estructura "**Si Entonces**", que presenta una opción adicional para indicar un bloque de instrucciones que se deben ejecutar en caso de no cumplirse la condición especificada.

El lenguaje de programación PHP dispone de una estructura similar para llevar a cabo este tipo de tareas. Sin embargo, la sintaxis que se utiliza para su declaración difiere de la que nos provee el Pseudolenguaje PSeInt, mas no es así su funcionamiento.

	Pseudo-Intérprete	Lenguaje de Programación
		
Selección Doble	**Si** condición **Entonces** instrucciones caso verdadero **Sino** instrucciones caso falso **FinSi**	**if** (condición) **{** instrucciones caso verdadero **} else {** instrucciones caso falso **}**

Tabla 28: Comparativo Estructura de Selección Doble - PHP vs PSeInt - Fuente: Elaborada por el autor.

Observé detalladamente la palabra reservada ***else*** declarada en el cuerpo de instrucciones correspondientes a la estructura ***if*** de PHP, al igual que la palabra reservada "**Sino**" de PSeInt, ***else*** extiende la estructura de selección para ejecutar un conjunto de instrucciones en caso de que la condición se evalué como falsa.

Por otra parte, es importante destacar que las llaves que rodean a la instrucción ***else***, permiten distinguir el inicio y final de cada bloque de instrucciones declaradas en la estructura selectiva. En caso de omitirlos, PHP notificará de lo sucedido y enviará un mensaje de error inesperado por pantalla (*Parse error: syntax error, unexpected 'else'*).

Si desea obtener mayor información acerca del trabajo referente a la instrucción ***else*** de PHP, sírvase en visitar su documentación a través de la siguiente dirección:

- http://php.net/manual/es/control-structures.else.php

3.2.3 Estructura de Selección Si Entonces Sino (Anidado)

Según lo visto en el capítulo anterior, la estructura de selección "**Si Entonces Sino Anidado**" permite resolver cualquier situación en la que se necesite evaluar más de una condición lógica y por consiguiente ejecutar una de varias acciones disponibles.

Al igual que en el Pseudolenguaje PSeInt, en PHP se pueden anidar diferentes estructuras de selección ***if-else*** para aumentar el número de opciones disponibles a evaluar, y por consiguiente tener un número más amplio de acciones posibles que se pueden ejecutar.

	Pseudo-Intérprete	Lenguaje de Programación
		
Selección Anidada	**Si** condición1 **Entonces** instrucciones condición 1 **Sino** **Si** condición2 **Entonces** instrucciones condición 2 **Sino** instrucciones caso por defecto **FinSi** **FinSi**	**if** (condición1) **{** instrucciones condición 1 **} else {** **if** (condición2) **{** instrucciones condición 2 **} else {** instrucciones caso por defecto **}** **}**

Tabla 29: Comparativo Estructura de Selección Anidada - PHP vs PSeInt - Fuente: Elaborada por el autor.

Observe detalladamente como la estructura ***if-else*** de PHP declarada dentro del bloque de instrucciones ***else*** (correspondiente a la primera estructura de selección) añade una segunda condición para evaluar en caso de que la primera se valore como falsa. Si por alguna situación la segunda condición llega a fallar, es decir, se evalúa falsa, la primera estructura de selección ***if-else*** se extiende y se ejecuta el tercer bloque de instrucciones declarado como caso por defecto.

Ahora bien, tal y como se comentó en un capítulo anterior. El uso excesivo de este tipo de anidamientos puede conducir a que se generen estructuras de selección cada vez más complejas, y originar que nuestro código (a futuro) sea más complicado de mantener, pero sobre todo de leer.

En este sentido, a diferencia de PSeInt, el lenguaje de programación PHP ofrece al programador una variante de este tipo de estructura de selección anidada para llevar a cabo trabajos con decisiones encadenadas. Tal y como se muestra en la siguiente tabla comparativa:

	Pseudo-Intérprete	Lenguaje de Programación
		 Variante Selección Anidada
Selección Anidada	**Si** condición1 **Entonces** instrucciones condición 1 **Sino** **Si** condición2 **Entonces** instrucciones condición 2 **Sino** instrucciones caso por defecto **FinSi** **FinSi**	**if** (condición1) **{** instrucciones condición 1 **} elseif** (condición2) **{** instrucciones condición 2 **} else {** instrucciones caso por defecto **}**

Tabla 30: Estructura de Selección Anidada elseif disponible en PHP - Fuente: Elaborada por el autor.

Observe detenidamente como en ambas estructuras de selección se define la declaración de dos reglas condicionales. En caso de que la primera falle, PSeInt anida una nueva estructura "**Si Entonces Sino**" en el bloque de instrucciones del caso falso para continuar con la evaluación de la segunda condición. Sin embargo, el lenguaje de programación PHP lleva a cabo este tipo de anidamiento a través de la palabra reservada ***elseif***, el cual permite que el segundo bloque de instrucciones sea completamente independiente del primero, lo que mejora progresivamente la legibilidad de nuestro código.

Bajo este mismo orden de ideas, si más adelante se requiere actualizar dicho código con la inclusión de dos opciones adicionales (véase la tabla 31). La legibilidad del código PHP es por mucho más sencilla que la de PSeInt.

	Pseudo-Intérprete	Lenguaje de Programación
		 Variante Selección Anidada
Selección Anidada	**Si** condición1 **Entonces** instrucciones condición 1 **Sino** **Si** condición2 **Entonces** instrucciones condición 2 **Sino** **Si** condición3 **Entonces** instrucciones condición 3 **Sino** **Si** condición4 **Entonces** instrucciones condición 4 **Sino** instrucciones por defecto **FinSi** **FinSi** **FinSi** **FinSi**	**if** (condición1) **{** instrucciones condición 1 **} elseif** (condición2) **{** instrucciones condición 2 **} elseif** (condición3) **{** instrucciones condición 3 **} elseif** (condición4) **{** instrucciones condición 4 **} else {** instrucciones por defecto **}**

Tabla 31: Ventajas de la Estructura de Selección Anidada elseif disponible en PHP - Fuente: Elaborada por el autor.

Para obtener más información acerca del trabajo con la estructura de selección ***elseif*** de PHP, sírvase en visitar el siguiente enlace:

- http://php.net/manual/es/control-structures.elseif.php

3.2.4 Estructura de Selección Múltiple (Según)

Como es de su conocimiento, la estructura de selección múltiple permite la ejecución de un bloque de instrucciones en función del valor que tome una determinada expresión.

PHP considera una estructura de selección similar para llevar a cabo este tipo de trabajos. Solo que su sintaxis difiere de una u otra forma con respecto a la que nos provee el Pseudolenguaje PSeInt, más no es así su funcionamiento.

	Pseudo-Intérprete	Lenguaje de Programación
		
Selección Múltiple	**Segun** expresión **Hacer** **opción1:** instrucciones opción 1 **opción2, opción3:** instrucciones opción 2 y 3 **opción4:** instrucciones opción 4 **opciónN:** instrucciones opción N **De Otro Modo:** instrucciones por defecto **FinSegun**	**switch** (expresión) **{** **case opción1:** instrucciones opción 1 **break;** **case opción2:** **case opción3:** instrucciones opción 2 y 3 **break;** **case opción4:** instrucciones opción 4 **break;** **case opciónN:** instrucciones opción N **break;** **default:** instrucciones por defecto **}**

Tabla 32: Comparativo Estructura de Selección Múltiple - PHP vs PSeInt - Fuente: Elaborada por el autor.

Como se puede observar, la estructura ***switch*** en PHP es un equivalente de la estructura **Según** en PSeInt. Ambas comprueban el valor que toma una determinada expresión (en PHP ésta se encierra entre paréntesis), y comienzan a compararlo con diferentes casos u opciones disponibles. En este sentido, la palabra **Entonces** es sustituida por una llave de apertura "**{**", y las opciones disponibles comienzan por la palabra reservada ***case***. En caso de que dos o más opciones ejecuten el mismo conjunto de

instrucciones, en lugar de separarlas con el signo coma (como lo haría en PSeInt), estas deben declararse una debajo de la otra.

Sumado a lo expuesto, para indicar el final de un conjunto de instrucciones correspondientes a una opción o caso en particular. A diferencia de PSeInt, PHP hace uso de la palabra reservada ***break***. En caso de omitirse, PHP continuará con la ejecución de las instrucciones establecidas en los casos siguientes, sin que exista la necesidad de comparar el valor inicial (condición).

Finalmente, para la declaración de un conjunto de instrucciones por defecto, PHP hace uso de la palabra reservada ***default*** en lugar **De Otro Modo**; y para indicar el cierre de la estructura selectiva se sustituye la palabra **FinSegun** por una llave de cierre "**}**".

Si desea aprender más acerca de la estructura ***switch*** de PHP, sírvase en visitar su documentación a través del siguiente enlace:

- http://php.net/manual/es/control-structures.switch.php

3.2.5 Estructura de Repetición Para

De acuerdo con lo abordado en un capítulo anterior de esta misma investigación, la estructura de repetición del tipo "**Para**" permite ejecutar una secuencia de instrucciones un número determinado de veces.

En este sentido, PHP ofrece una estructura similar para llevar a cabo este tipo de tareas. Sin embargo, su declaración difiere con respecto a la

sintaxis utilizada por el Pseudolenguaje PSeInt. Tal y como se ilustra en la siguiente tabla comparativa:

	Repetición Definida
Pseudo-Intérprete	**Para** varControl = val_Inicio **Hasta** val_Final **Con Paso** IncDec **Hacer** instrucciones **FinPara**

Programación	for (varControl = val_Inicio ; condición_val_Final ; IncDec) {
	instrucciones
	}

Tabla 33: Comparativo Estructura de Repetición Para - PHP vs PSeInt - Fuente: Elaborada por el autor.

El comportamiento de ambas estructuras repetitivas en la práctica es el mismo, solo que en PHP la palabra reservada "**Para**" es sustituida por su equivalente **for**. Luego, tanto una como otra hacen uso de una variable de control misma que es inicializada y evaluada al comienzo de la primera iteración. Posteriormente, las dos estructuras indican el valor final que debe tomar la variable de control para dejar de repetir el conjunto de instrucciones declaradas dentro del bucle. Sin embargo, a diferencia de PSeInt, PHP lo hace a través de una expresión condicional (donde se involucra el valor final que debe tomar la variable de control). Después de esto, ambas estructuras indican la actualización (valor de paso, ya sea incremento o decremento) que debe recibir la variable de control tras finalizar cada iteración.

Llegado a este punto, para actualizar el valor de paso de la variable de control declarada dentro de la estructura de repetición **for**, PHP requiere

de un conjunto de operadores aritméticos especiales llamados Post-incremento y Post-decremento (véase el ejemplo ilustrado en la tabla 34), mismos que le ayudan a sumar o restar en una unidad el valor actual que debe tomar dicha variable de control tras su paso por cada iteración (actualiza de manera automática su contenido, siempre al final de cada iteración).

SIGNIFICADO	EJEMPLO	EQUIVALENCIA
Post-Incremento	$varControl++	$varControl = $varControl + 1
Post-Decremento	$varControl--	$varControl = $varControl - 1

Tabla 34: Post-Incremento y Post-Decremento en PHP - Fuente: Elaborada por el autor.

Finalmente, PHP sustituye la palabra reservada "**Hacer**" del ciclo "**Para**" por una llave de apertura "**{**", y finaliza el ciclo repetitivo con una llave de cierre "**}**" en lugar del término "**FinPara**".

Por otra parte, es importante enfatizar que PHP hace uso de puntos y comas para separar las expresiones declaradas en la definición del ciclo. Por tanto, nunca olvide su declaración, ya que, de lo contrario, PHP le informará de lo sucedido y enviará un mensaje de error inesperado por pantalla (*Parse error: syntax error, unexpected ['$nombreVariableControl'] (T_VARIABLE), expecting ';'*)

Si desea obtener más información acerca del trabajo referente a la estructura de repetición **for** en PHP, sírvase en visitar el siguiente enlace correspondiente a su documentación:

- http://php.net/manual/es/control-structures.for.php

3.2.6 Estructura de Repetición Mientras

Según lo comentado en un capítulo anterior de esta misma investigación, la estructura de repetición del tipo **Mientras**, permite ejecutar una secuencia de instrucciones siempre y cuando una condición sea evaluada como verdadera.

El lenguaje de programación PHP dispone de una estructura similar para llevar a cabo este tipo tareas (véase la tabla 35). Sin embargo, la sintaxis que se emplea para su declaración difiere más o menos de la utilizada por el Pseudolenguaje PSeInt, mas no es así su funcionamiento.

	Pseudo-Intérprete	Lenguaje de Programación
		

Repetición Indefinida	**Mientras** condición **Hacer**
	instrucciones
	FinMientras
		while (condición) {
		instrucciones
		}

Tabla 35: Comparativo Estructura de Repetición Mientras - PHP vs PSeInt - Fuente: Elaborada por el autor.

Como se puede llegar a vislumbrar, la estructura de repetición del tipo "**Mientras**" es identificada por PHP mediante la palabra reservada ***while***. La condición o condiciones que se deben evaluar están debidamente encerradas con el uso de paréntesis. La llave de apertura "**{**" sustituye al

Hacer de PSeInt y, por consiguiente, la llave de cierre "**}**" finaliza la declaración de la estructura ***while*,** supliendo al **FinMientras** de PSeInt.

Si desea aprender más acerca del trabajo con la estructura de repetición ***while*** de PHP, sírvase en visitar el siguiente enlace:

- http://php.net/manual/es/control-structures.while.php

3.2.7 Estructura de Repetición Repetir Mientras Que

Como es de nuestro conocimiento, la estructura de repetición del tipo "**Repetir Mientras Que**" consiste en una variante de la estructura "**Mientras**", con la diferencia de que la condición de ejecución se comprueba siempre al final del bloque de instrucciones, lo que provoca que las instrucciones declaradas dentro del cuerpo del ciclo se ejecuten por lo menos la primera vez.

Al tratarse de un lenguaje de programación formal, PHP dispone de una estructura de repetición similar para llevar a cabo este tipo de trabajos. Sin embargo, al igual que en sus otras estructuras, esta difiere nuevamente en sintaxis con respecto a PSeInt, mas no es así funcionamiento. Tal y como se muestra en la siguiente tabla comparativa:

	Pseudo-Intérprete	Lenguaje de Programación
		
Repetición Indefinida	**Repetir** instrucciones **Mientras Que** condición	**do {** instrucciones **} while** (condición) ;

Tabla 36: Comparativo Estructura Repetir Mientras Que - PHP vs PSeInt - Fuente: Elaborada por el autor.

Como se puede observar, la estructura ***do-while*** de PHP garantiza el mismo comportamiento que brinda la estructura "**Repetir Mientras Que**" de PSeInt. Los cambios con respecto a su sintaxis son mínimos. La palabra reservada "**Repetir**" se sustituye por su equivalente ***do*** seguido de una llave de apertura "**{**". Las palabras "**Mientras Que**" son reemplazadas por una llave de cierre "**}**" seguido del término ***while***. Finalmente, la condición es encerrada entre paréntesis y concluye con un punto y coma (**;**).

Si desea conocer más a fondo acerca del trabajo con respecto a la estructura de repetición ***do-while*** de PHP, sírvase en visitar el siguiente enlace:

- http://php.net/manual/es/control-structures.do.while.php

3.2.8 Funciones primitivas del lenguaje (Pre-construidas)

Tal y como se comentó en un capítulo anterior, las funciones pre-construidas consisten en un conjunto de instrucciones predefinidas por el

lenguaje de programación para realizar una determinada tarea. Su objetivo principal consiste en abstraen al desarrollador del esfuerzo que representa implementar su lógica de programación, y le orilla a pensar solamente en los resultados obtenidos tras su ejecución.

A diferencia de PSeInt, el lenguaje de programación PHP cuenta con un amplio conjunto de funciones predefinidas para el trabajo con cálculos matemáticos y operaciones con cadenas de texto. Sin embargo, el nombre utilizado como identificador para cada una de ellas, cambia por completo con respecto al manejado por PSeInt. Tal y como se muestra en la siguiente tabla comparativa:

	Funciones Matemáticas		Funciones Caracteres		
	PSeInt	php	PSeInt	php	
Raíz cuadrada	RAIZ(x)	sqrt(x)	MAYUSCULAS(s)	strtoupper(s)	A mayúsculas
Valor absoluto	ABS(x)	abs(x)	MINUSCULAS(s)	strtolower(s)	A minúsculas
Parte entera	TRUNC(x)	floor(x)	LONGITUD(s)	strlen(s)	Cantidad caracteres
Entero más cercano	REDON(x)	round(x)	CONCATENAR(s1,s2)		Unir cadenas
Seno	SEN(x)	sin(x)	SUBCADENA(s,x,y)	substr(s,x,y)	Extracto de un texto
Coseno	COS(x)	cos(x)	CONVERTIRANUMERO(x)	intval(s)	Convierte a número
Tangente	TAN(x)	tan(x)	CONVERTIRATEXTO(s)	strval(s)	Convierte a texto
Arco seno	ASEN(x)	asin(x)			
Arco coseno	ACOS(x)	acos(x)			
Arco tangente	ATAN(x)	atan(x)			
Entero aleatorio [0;x-1]	AZAR(x)	rand(0,x-1)			
Entero aleatorio [a;b]	ALEATORIO(a,b)	rand(a,b)			
Función exponencial	EXP(x)	exp(x)			
Logarítmo natural	LN(x)	log(x)			

Tabla 37: Comparativo Funciones Predefinidas - PHP vs PSeInt - Fuente: Elaborada por el autor.

De la tabla anterior es importante mencionar que:

- Las funciones trigonométricas en PHP comparten las mismas características que las mencionadas para PSeInt, es decir, tanto el valor del parámetro que reciben como el valor retornado por cada una de ellas, se encuentra expresado en radianes.
- La función **AZAR( )** no tiene un equivalente directo en PHP. Sin embargo, su comportamiento es sencillo de simular haciendo uso de la función ***rand( )***, e indicando como parámetros un punto de inicio con valor de cero, y un punto final con valor de x-1, es decir, un valor menor (de una unidad) al límite establecido.
- Si el valor pasado como parámetro a la función ***intval( )*** de PHP es una cadena de texto, se procede a revisar en primer lugar el contenido de su extremo izquierdo; si este coincide con un número, el resultado de la conversión será dicho número, de lo contario, el resultado devuelto será cero.
- La posición inicial de los elementos de tipo cadena en PHP comienzan en 0 y no en 1 como en PSeInt. Es decir, si desea obtener los primeros tres caracteres que aparecen en la cadena "**Ale**jandro", su declaración en PHP sería la siguiente:

```
substr("Alejandro", 0, 3)
```

Imagen 112: Función "substr( )" definida en PHP - Fuente: Diseñada por el autor.

- La función **CONCATENAR( )** de PSeInt no tiene un equivalente directo en PHP. Si bien su comportamiento no se puede simular con la ayuda de otra función, PHP incorpora un operador de texto para

llevar a cabo este tipo de trabajos. Tal y como se ilustra en el siguiente ejemplo:

	Pseudo-Intérprete	Lenguaje de Programación
		
Función CONCATENAR vs Operador de Texto	texto = CONCATENAR("Hola " ,"Alex")	$texto = "Hola " . "Alex" ; $completo = $nombre . $ap1 . $ap2; $leyenda = "Bienvenido: " . $claveID;

Tabla 38: Comparativo Función CONCATENAR vs Operador de Texto - Fuente: Elaborada por el autor.

El operador de texto en PHP (signo de punto), permite concatenar diferente tipo de contenido sobre una misma cadena de texto, retornando dicho resultado para que sea asignado mediante el uso de una variable, y posteriormente pueda ser mostrado por pantalla.

Finalmente, para invocar las acciones declaradas en una función predefinida por PHP, se procede de la misma forma que en PSeInt, es decir, se escribe el nombre de la función seguido de paréntesis (con sus correspondientes argumentos, si es que los hubiese), y para el caso de PHP, un punto y coma adicional para indicar el final de la instrucción. Por ejemplo:

```
echo "La Raíz cuadrada es:" , sqrt($valor);

$nombre = strtotupper($nombreUsuario);

$numCaracteres = strlen($contraseniaUsuario);
```

Imagen 113: Invocar Funciones Predefinidas en PHP - Fuente: Diseñada por el autor.

Si desea obtener mayor información acerca de otras funciones predefinidas por PHP para el trabajo con cálculos matemáticos y operaciones con cadenas de texto, sírvase en visitar los siguientes enlaces:

- http://php.net/manual/es/ref.math.php
- http://php.net/manual/es/ref.strings.php
- http://php.net/manual/es/indexes.functions.php

3.2.9 Funciones del programador (Externas)

Como es de su conocimiento, las funciones definidas por el programador consisten en un trozo de código agrupado en un solo componente reciclable y separado del resto de instrucciones del programa principal, el cual debe estar debidamente identificado para que más adelante pueda ser invocado en diferentes partes del programa para ejecutar una tarea determinada en medida que se le necesite.

Al igual que el Pseudolenguaje PSeInt, PHP ofrece al programador diversas formas para definir una función. Además, su comportamiento y sintaxis es por poco parecida al trabajado con anterioridad. Tal y como se ilustra en la siguiente tabla comparativa:

	Pseudo-Intérprete	Lenguaje de Programación
		

Sin Parámetros	Funcion _NOMBRE( )
	instrucciones a ejecutar
	FinFuncion
		function _NOMBRE( ) {
		instrucciones a ejecutar
		}

Con Parámetros	Funcion _NOMBRE(pmA , pmN)
	instrucciones a ejecutar
	FinFuncion
		function _NOMBRE($pmA , $pmN) {
		instrucciones a ejecutar
		}

Valor de Retorno	Funcion valorR = _NOMBRE(pm)
	instrucciones a ejecutar
	FinFuncion
		function _NOMBRE($pm) {
		instrucciones a ejecutar
		return $valorR ;
		}

Tabla 39: Comparativo Declaración de Funciones Externas - PHP vs PSeInt - Fuente: Elaborada por el autor.

Como se puede vislumbrar, la palabra clave "**Funcion**" de PSeInt es sustituida en PHP por su equivalente ***function***. Luego, los criterios utilizados para nombrar a una función son los mismos que se consideran en PSeInt para identificar a las variables. A su vez, en ambos casos se respeta el uso de paréntesis, pero en PHP se añade al final de ellos una llave de apertura "**{**" para indicar el inicio del cuerpo de la función. Posteriormente, se declara el conjunto de instrucciones; y al final se intercambia el uso de la palabra reservada "**FinFuncion**" por una llave de cierre "**}**" la cual indica el final de la definición de la función.

Por otra parte, si la función por definir en PHP se espera que reciba valores de entrada (es decir, parámetros). Al igual que PSeInt, estos deben ser

declarados entre los paréntesis de la función como una lista de elementos debidamente separados por comas. Sin embargo, tenga muy en cuenta que en PHP (y también en PSeInt) los parámetros de la función actúan como variables locales, por tanto, estos deben iniciar con el símbolo de dólar.

Finalmente, si lo que desea es que una función retorne un valor como resultado de su ejecución. A diferencia de PSeInt, en PHP se debe indicar el valor a devolver mediante la palabra reservada ***return***, siempre al final de las instrucciones declaradas en el cuerpo de la función.

Ahora bien, una vez definido el conjunto de acciones a desempeñar por una función. Su invocación desde PHP es el mismo que el utilizado por PSeInt, es decir, se escribe el nombre del identificador de función seguido de sus correspondientes paréntesis, y para el caso de PHP, un punto y coma adicional para indicar el final de la instrucción. Por ejemplo:

```
_MENSAJEBIENVENIDA( );
$resultado = _CONVERTIRAMETROS($valor, "km");
echo "El resultado de la función es:" , _ARADIANES(180);
echo "Resultado:" , _ACENTIMETROS($valor) * M_PI;
```

Imagen 114: Invocar Funciones Externas en PHP - Fuente: Diseñada por el autor.

Si desea aprender más acerca de la definición de funciones en PHP, sírvase en visitar la documentación publicada a través del siguiente enlace:

- http://php.net/manual/es/functions.user-defined.php

Llegado a este punto, se espera que el lector comprenda por que en los primeros capítulos de esta investigación se abordó todo lo referente al diseño de los Algoritmos mediante la técnica del Pseudocódigo con la ayuda de un *software* como es el caso de PSeInt y no directamente a través de un Lenguaje de Programación formal y estricto como es el caso de PHP.

Tal y como se ha comentado con anterioridad, las estructuras de control son construcciones prefabricadas de cada lenguaje que permiten a los programadores resolver problemáticas complejas que involucren la toma de decisiones o la repetición continua de instrucciones. En este sentido, la falta de comprensión y dominio sobre estos tópicos, "como en todo ámbito laboral", genera una barrera que permite diferenciar con facilidad las competencias y habilidades que domina cada profesional de la programación. Es por ello que, con la redacción de este numeral, nuevamente se pone en evidencia la importancia de los conocimientos adquiridos hasta el momento (capítulo 2 de esta misma investigación), puesto que, como se pudo observar, en términos generales la sintaxis que emplea PHP para la construcción de funciones y estructuras de control es parecida a la que utiliza el Pseudolenguaje PSeInt para el diseño y desarrollo de Algoritmos.

Ahora que se conoce a detalle cada uno de los aspectos de nivel intermedio presentes tanto en el Pseudocódigo como en el Lenguaje de Programación PHP, es momento de ceder el paso a la práctica e implementar cada una de estas características en proyectos que solucionen problemáticas de la vida real; este es el objetivo del siguiente y último numeral de esta investigación. Sin embargo, antes de culminar el presente apartado, se expone el siguiente material de referencia (imagen), con la finalidad de resumir los factores abordados hasta el momento.

Programación Web



Hoja de Referencia Rápida - Programación Estructurada y Funcional con PHP

Selección Simple

```
if (condición) {
   instrucciones
}
```

Selección Doble

```
if (condición) {
   instrucciones caso verdadero
} else {
   instrucciones caso falso
}
```

Selección Múltiple

```
switch (expresión) {
   case opción1:
     instrucciones opción 1
     break;
   case opción2:
   case opción3:
     instrucciones opción 2 y 3
     break;
   case opciónN:
     instrucciones opción N
     break;
   default:
     instrucciones por defecto
}
```

Selección Anidada

```
if (condición_1) {
   instrucciones condición 1
} elseif (condición_2) {
   instrucciones condición 2
} elseif (condición_3) {
   instrucciones condición 3
} elseif (condición_4) {
   instrucciones condición 4
} else {
   instrucciones caso por defecto
}
```

Repetición Definida

```
for (var = valor ; condición ; inc_dec) {
   instrucción A
   instrucción N
}
```

Repetición Indefinida

```
While (condición) {
   instrucción A
   instrucción N
}
```

```
do {
   instrucción A
   instrucción N
} while (condición);
```

Definición de Funciones

Sin Parámetros	`function _NOMBRE( ) { instrucciones a ejecutar }`
Con Parámetros	`function _NOMBRE($pmA, $pmB) { instrucciones a ejecutar }`
Valor de Retorno	`function _NOMBRE($param) { instrucciones a ejecutar return $valorR; }`

3.3 Soluciones de programación: Migrar del Pseudocódigo al Lenguaje de Programación PHP

El presente apartado de la investigación conforma un recorrido por las principales características del Lenguaje de programación PHP bajo un enfoque práctico. Si bien es imposible abordar todas las posibilidades que ofrece dicho lenguaje en unas cuentas páginas, se han recolectado las más interesantes y utilizadas (desde la perspectiva del autor). En este sentido, al finalizar la lectura de este apartado, el lector será capaz de manejar con soltura dicho lenguaje de programación y podrá llevar a cabo el diseño y desarrollo de diferentes proyectos, tanto en el ámbito profesional como en el particular.

Al tratarse de una guía básica para futuros desarrolladores PHP, ésta cuenta con dos objetivos principales: 1) entregar soluciones a problemas comunes y 2) educar a los desarrolladores acerca de la amplia serie de funciones integradas, así como los elementos y estructuras de PHP prefabricadas que pueden ser implementadas en cada uno de sus proyectos. Es por ello que, para la elaboración de dicho material, se presenta la solución a seis ejercicios prácticos ilustrados y explicados paso a paso, cuyo análisis y respuesta se presenta en primer lugar a nivel de Pseudocódigo (PSeInt) y posteriormente se migra a código nativo de PHP; esto con el fin de no dejar ninguna duda durante su proceso de codificación y ejecución.

Gracias a este sistema, se pretende garantizar que una vez realizados, pero, sobre todo, analizados los seis ejercicios que componen esta guía

tutorial, el usuario sea capaz de desenvolverse cómodamente con las herramientas básicas de PHP y sacar el máximo partido de sus múltiples prestaciones.

Es importante recordar al lector que el presente apartado de la investigación está orientado principalmente a desarrolladores novatos e intermedios que trabajan actualmente con algún Pseudolenguaje y desean sumergirse en el campo del desarrollo Web con la ayuda de un Lenguaje de Programación formal como es el caso de PHP. Para este fin, los contenidos citados en este apartado se encuentran estructurados de tal manera que empiezan por resolver problemas fáciles y luego avanzan a otros difíciles o complejos. Esto se hace deliberadamente para dar a los futuros desarrolladores PHP el conocimiento necesario para comprender los listados de códigos que se encontrarán más adelante dentro de esta misma sección.

Finalmente, para trabajar correctamente con los ejemplos que se proponen en este numeral, es necesario contar con una instalación funcional de PHP 5.x, un servidor de aplicaciones Web Apache 2.x, un editor de textos[16] y un navegador Web (no se requiere conexión a Internet). Para ello, puede remitirse al anexo A2, donde encontrará más información al respecto.

Sin más preámbulos al respecto. A continuación, Soluciones de Programación: Migrar del Pseudocódigo al Lenguaje de Programación PHP.

[16] Por lo general, cada Sistema Operativo incluye de forma gratuita un editor de textos como parte de sus accesorios de trabajo (Bloc de Notas, TextEdit, Gedit). Sin embargo, para el desarrollo de esta guía tutorial se propone el uso de Sublime Text, un sofisticado editor de textos con resaltado de código multiplataforma de distribución gratuita y comercial, mismo que puede ser descargado a través de la siguiente dirección electrónica: https://www.sublimetext.com/

3.3.1 Leer datos proporcionados por el usuario

Problema: ¿Desea desarrollar un programa de computadora para determinar el área, perímetro y diámetro de un círculo?

Elementos gramaticales que intervienen en la solución del problema: Identificadores, variables, constantes, entradas y salidas de información, operadores aritméticos, expresiones, asignaciones, comentarios y operador de texto.

Descripción detallada del problema: Emplee sus conocimientos de matemáticas básicas adquiridos durante su estancia en la educación primaria y desarrolle un programa de computadora mismo que, al ingresar un valor numérico que represente el radio de un círculo, determine e imprima su correspondiente diámetro, área, y perímetro.




```
1   Proceso programaCirculo
2       //Entrada de Información
3       Imprimir "Programa Trigonometría - Círculo"
4       Imprimir ""
5       Imprimir "Ingrese radio"
6       Leer radio
7       //Proceso de Información
8       diametro = radio * 2
9       area = PI * (radio ^ 2)
10      perimetro = PI * diametro
11      //Salida de Información
12      Imprimir ""
13      Imprimir "------ Resultados ------"
14      Imprimir "Diámetro: " , diametro
15      Imprimir "Área: " , area
16      Imprimir "Perímetro: " , perimetro
17  FinProceso
```

Imagen 115: Pseudocódigo PSeInt - Fuente: Diseñada por el autor.

Extracto

```
7   <body>
8       <h1>Programa Trigonometría - Círculo</h1>
9       <form action="GetPost.php" method="POST">
10          <label>Ingrese radio</label>
11          <input type="text" name="campo_radio">
12          <input type="submit" value="Procesar">
13      </form>
14  </body>
```

Imagen 116: Formulario HTML - Fuente: Diseñada por el autor.

Explicación Extracto de Formulario HTML:

Todos los formularios HTML comienzan con la etiqueta ***<form>*** y terminan con ***</form>*** (líneas 9-13), para configurar su comportamiento es necesario especificar algunos atributos opcionales, siendo los más indispensables el atributo ***action*** y ***method***.

- El atributo ***action*** define la localización del archivo (en este caso el archivo PHP) que se encargará de recolectar y procesar la información enviada por el formulario.
- El atributo ***method*** especifica el método HTTP que se utilizará para enviar la información (puede ser ***GET*** o ***POST***).

Internamente un formulario se construye a partir de uno o más controles. Estos controles pueden ser campos de texto (de una sola línea o multi-línea), cajas de selección, botones, casillas de verificación (*checkboxes*), o botones de radio. Por lo que la mayoría de las veces, estos controles estarán junto a una etiqueta ***<label>*** que describe su propósito.

El formulario correspondiente al programa de trigonometría es muy simple y contiene tres controles: un ***label*** (línea 10), un **campo de texto** (línea 11), y un ***botón*** para permitir al usuario enviar la información que llenó en el formulario (línea 12).

Antes de enviar la información al archivo especificado por el atributo ***action*** (programa PHP), es necesario dar un nombre único a cada control de formulario que va a recolectar una parte específica de esa información. Para ello, se utiliza el atributo ***name*** (línea 11).

Con base en lo expuesto, podemos concluir que, en el ejemplo anterior, el formulario HTML envía sólo una parte de información llamada "**campo_radio**" a través del método ***POST***. Misma que será procesada por las instrucciones declaradas en el archivo **GetPost.php**.



¿Sabía qué?

HTML significa Lenguaje de Marcado para Hipertextos y su función dentro del desarrollo Web consiste en proveer la estructura semántica para situar los contenidos que serán mostrados dentro de una página Web. HTML hace uso de etiquetas delimitadas entre corchetes angulares "<tag>" para informar al navegador Web como debe interpretar y desplegar la información al usuario. Sin embargo, para que todo esto sea posible, es importante que la extensión de estos archivos termine con **.html**



```php
<?php
    /*** Nombre del archivo: GetPost.php ***/

    //Entrada de Información
    $radio = $_POST["campo_radio"];

    //Proceso de Información
    $diametro = $radio * 2;
    $area = M_PI * pow($radio, 2);
    $perimetro = M_PI * $diametro;

    //Salida de Información
    echo "------ Resultados ------";
    echo "<br>";
    echo "Diámetro: " , $diametro;
    echo "<br>";
    echo "Área: " , $area;
    echo "<br>";
    echo "Perímetro: " , $perimetro;
?>
```

Imagen 117: Programa PHP - Fuente: Diseñada por el autor.

Explicación PHP:

Los programas desarrollados en PHP inician con la etiqueta **<?php** y finalizan con **?>** (líneas 1-20). La mayoría de las veces, usted trabajará con valores enviados desde un formulario HTML, los cuales pueden ser transmitidos en modo ***GET*** o en modo ***POST***. Para recuperar cada uno de estos datos desde PHP, es necesario declarar una variable y asignarle uno de los dos arreglos asociativos disponibles (según el método de envió ***$_GET[ ]*** o ***$_POST[ ]***) e indicar internamente el nombre del control de formulario que contiene parte de la información requerida (línea 5).

Por otra parte, cuando el código del programa se hace extenso. Es una buena idea hacer uso de comentarios que permitan documentar aquellas

partes del programa que resulten importantes durante el proceso (líneas 4, 7 y 12). Recuerde que PHP admite dos tipos de comentarios: los de una sola línea "**//**", y los multi-línea "**/* */**".

Cuando se hace uso de constantes predefinidas por el lenguaje de programación PHP (líneas 9-10). Al igual que PSeInt, éstas se escriben con su nombre en mayúsculas (tal y como fueron definidas), y el valor devuelto por la constante puede ser mostrado directamente en pantalla u operado libremente junto a otro contenido.

Finalmente, para mejorar la presentación de los resultados en pantalla. La instrucción ***echo*** concatena algunos mensajes pertinentes junto a los valores almacenados en las variables **$diametro**, **$area** y **$perimetro** (líneas 15, 17 y 19). Además, para evitar que se produzca una sola línea de texto con los resultados generados, la instrucción ***echo*** imprime la etiqueta HTML ***
*** (líneas 14, 16 y 18) para producir un salto de línea significante en el texto, y mostrarlo como es debido a través de la ventana del navegador Web.



¿Sabía qué?

PHP significa Pre Procesador de Hipertexto y es un lenguaje de programación de código abierto diseñado especialmente para el desarrollo de aplicaciones Web. Su principal característica es que puede ser embebido fácilmente dentro de los documentos HTML o invocar parte de sus etiquetas para formatear las salidas de información. Sin embargo, para ser ejecutado es necesario el uso de un servidor Web, que se encargue de procesarlo (mediante la invocación del interprete PHP) y servirlo a los diferentes clientes que lo soliciten. Para ello, es importante que la extensión de los archivos termine con **.php**

Resultados PSeInt y PHP

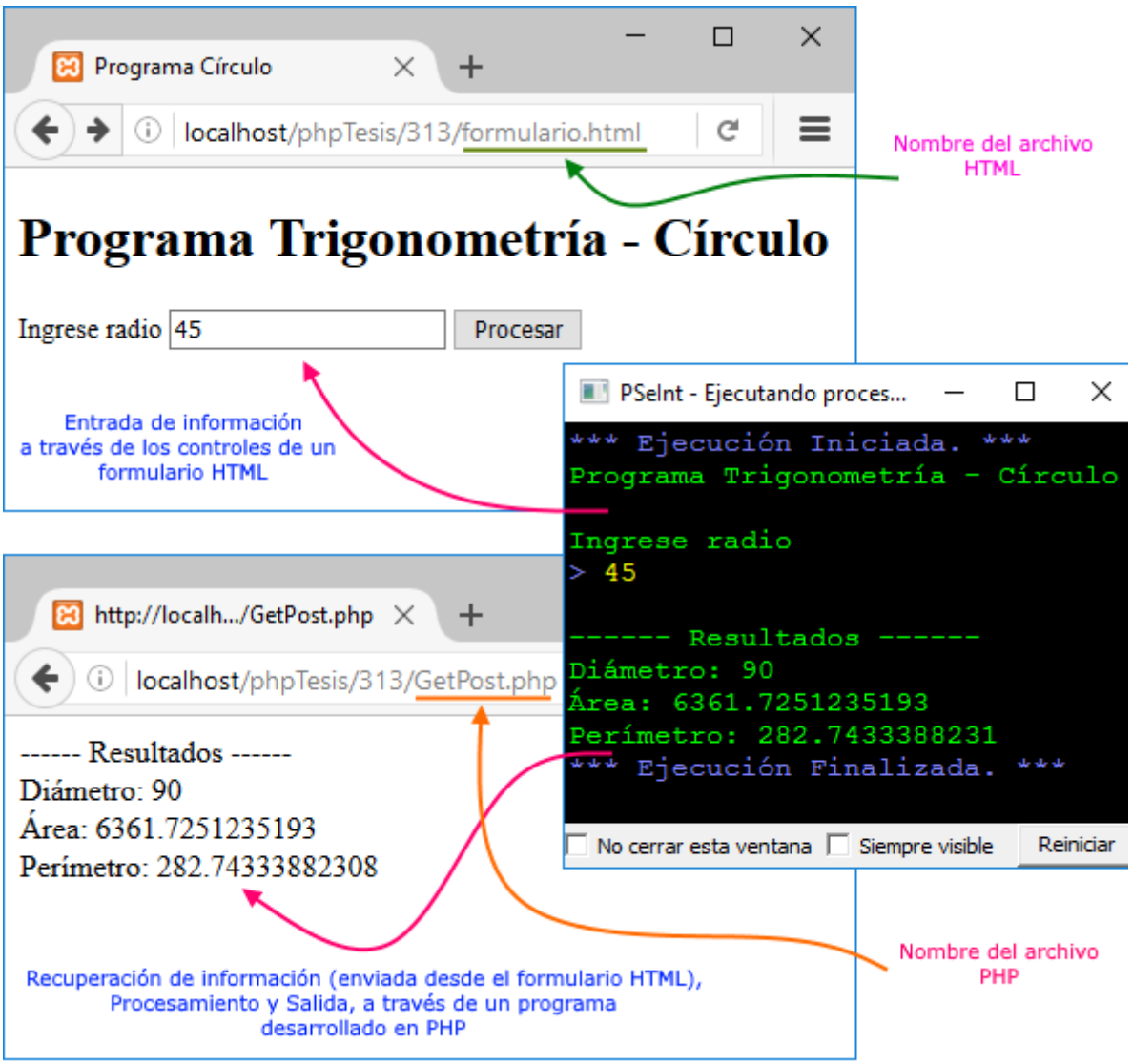


Imagen 118: Comparativo Ejecución PSeInt vs PHP - Fuente: Diseñada por el autor.

3.3.2 Tomar decisiones

Problema: ¿Desea desarrollar un programa de computadora para determinar el monto total a cobrar por concepto de tenencia o refrendo vehicular?

Elementos gramaticales que intervienen en la solución del problema: Identificadores, variables, entradas y salidas de información, operadores aritméticos, operadores de comparación, estructuras de selección, expresiones, asignaciones, comentarios y operador de texto.

Descripción detallada del problema: El Departamento de Tenencia Vehicular, le pide a usted desarrollar un programa mismo que, ingresado el nombre del propietario, la marca del vehículo, el modelo, y el precio de factura. Determine el total a cobrar por concepto de impuesto vehicular. Tome en cuenta que: si el modelo del vehículo es inferior a 2013, éste solo paga un impuesto de $350.00 más IVA por concepto de refrendo. Pero, si el modelo es 2013 o superior, se debe revisar su precio según factura para determinar el monto a pagar.

Sumado a lo expuesto, en caso de que el precio según factura sea inferior a $350,000.00 se debe cobrar un total de $600.00 más IVA por concepto de tenencia tipo VE (Tenencia para vehículos de clase económica); de lo contrario, se debe cobrar un impuesto de $1,500.00 más IVA por concepto de tenencia tipo VL (Tenencia para vehículos de lujo).

Finalmente, el programa debe mostrar un informe detallado a manera de comprobante de pago, imprimiendo datos indispensables tales como: el

nombre del propietario, la marca del vehículo, el modelo, el monto a pagar, y el tipo de tenencia aplicada.

```
Proceso programaTenencia
    ///Entrada de Información
    Imprimir "SISTEMA DE TENENCIA VEHICULAR"
    Imprimir ""
    Imprimir "Nombre del Propietario: " Sin Saltar
    Leer nombre
    Imprimir "Marca del Vehículo: " Sin Saltar
    Leer marca
    Imprimir "Modelo del Vehículo: " Sin Saltar
    Leer modelo
    Imprimir "Precio de Factura $: " Sin Saltar
    Leer factura
    ///Proceso de Información
    Si modelo >= 2013 Entonces
        Si factura < 350000 Entonces
            //Vehículos nuevos y seminuevos económicos $600 + IVA
            total = 600 + (600 * 0.16)
            concepto = "Tenencia VE"
        Sino
            //Vehículos nuevos y seminuevos de lujo $1500 + IVA
            total = 1500 + (1500 * 0.16)
            concepto = "Tenencia VL"
        FinSi
    Sino
        //Vehículos modelo 2012 y anteriores $350 + IVA
        total = 350 + (350 * 0.16)
        concepto = "Refrendo"
    FinSi
    ///Salida de Información
    Imprimir ""
    Imprimir "------ COMPROBANTE DE PAGO ------"
    Imprimir "Nombre del Propietario: " , nombre
    Imprimir "Marca del Vehículo: " , marca
    Imprimir "Modelo: " , modelo
    Imprimir "Monto a pagar: $ " , total
    Imprimir "Pago por concepto de: " , concepto
FinProceso
```

Imagen 119: Pseudocódigo PSeInt - Fuente: Diseñada por el autor.

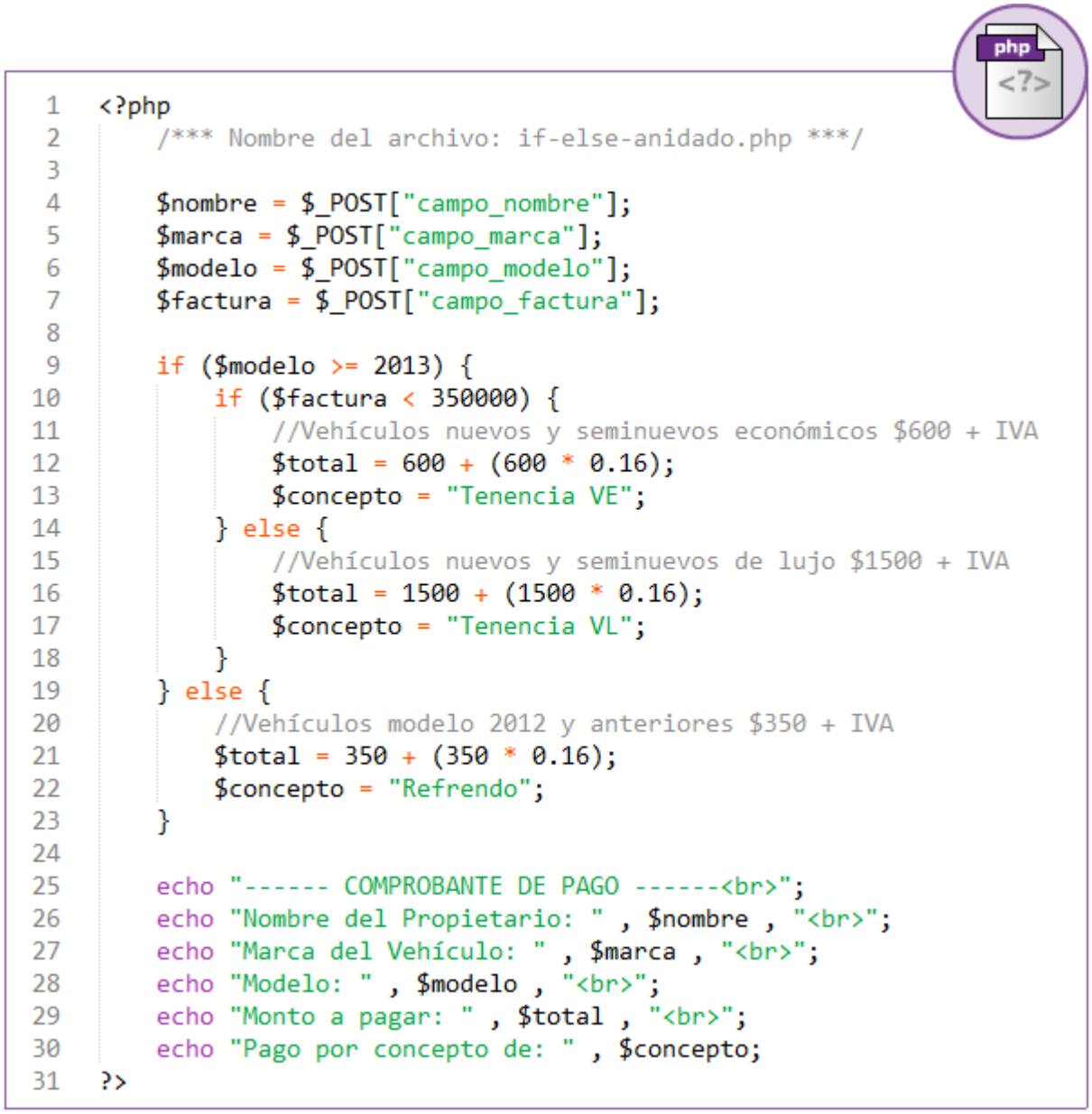
Extracto

```
13  <body>
14      <h1>Sistema de Tenencia Vehicular</h1>
15      <form action="if-else-anidado.php" method="POST">
16          <label>Nombre del Propietario: </label>
17          <input type="text" name="campo_nombre">
18          <br>
19          <label>Marca del Vehículo: </label>
20          <input type="text" name="campo_marca">
21          <br>
22          <label>Modelo del Vehículo: </label>
23          <input type="text" name="campo_modelo">
24          <br>
25          <label>Precio de Factura: $</label>
26          <input type="text" name="campo_factura">
27          <br>
28          <input type="submit" value="Procesar Información">
29      </form>
30  </body>
```

Imagen 120: Formulario HTML - Fuente: Diseñada por el autor.

Explicación Extracto de Formulario HTML:

Nuestro formulario HTML (líneas 15-29) internamente se construye a partir de nueve controles: cuatro etiquetas ***<label>*** (líneas 16, 19, 22 y 25), cuatro campos de texto (líneas 17, 20, 23 y 26), y un botón (línea 28) para el envío de la información. A su vez, el formulario envía por el método ***POST*** de HTTP cuatro partes de información al archivo de programa **if-else-aniado.php** para su correspondiente procesamiento, mismas que se encuentran debidamente identificadas por los nombres "**campo_nombre**", "**campo_marca**", "**campo_modelo**", y "**campo_factura**".



```php
<?php
    /*** Nombre del archivo: if-else-anidado.php ***/

    $nombre = $_POST["campo_nombre"];
    $marca = $_POST["campo_marca"];
    $modelo = $_POST["campo_modelo"];
    $factura = $_POST["campo_factura"];

    if ($modelo >= 2013) {
        if ($factura < 350000) {
            //Vehículos nuevos y seminuevos económicos $600 + IVA
            $total = 600 + (600 * 0.16);
            $concepto = "Tenencia VE";
        } else {
            //Vehículos nuevos y seminuevos de lujo $1500 + IVA
            $total = 1500 + (1500 * 0.16);
            $concepto = "Tenencia VL";
        }
    } else {
        //Vehículos modelo 2012 y anteriores $350 + IVA
        $total = 350 + (350 * 0.16);
        $concepto = "Refrendo";
    }

    echo "------ COMPROBANTE DE PAGO ------<br>";
    echo "Nombre del Propietario: " , $nombre , "<br>";
    echo "Marca del Vehículo: " , $marca , "<br>";
    echo "Modelo: " , $modelo , "<br>";
    echo "Monto a pagar: " , $total , "<br>";
    echo "Pago por concepto de: " , $concepto;
?>
```

Imagen 121: Programa PHP - Fuente: Diseñada por el autor.

Explicación PHP:

El programa PHP encargado de procesar la información lleva por nombre **if-else-anidado.php**, y comienza con la etiqueta de apertura **<?php** y finaliza con **?>** (línea 1-31). En un primer momento se procede a recuperar las partes de información enviada por el formulario HTML

mediante el arreglo asociativo ***$_POST***, y se almacena en variables independientes para su correspondiente procesamiento dentro del programa (líneas 4-7).

Más tarde en la línea de código número 9 se procede a evaluar mediante una estructura de selección ***if-else*** el modelo (año de fabricación) del vehículo registrado. En caso de ser un modelo 2013 o superior, se debe cobrar una cierta cantidad monetaria por concepto de tenencia vehicular (líneas 10-18); de lo contrario, es decir, si el modelo es inferior a 2013, el propietario solo debe girar un pago de $350.00 más IVA por concepto de refrendo (líneas 20-22).

Bajo este mismo orden de ideas, si el vehículo registrado acredita para un pago por concepto de tenencia. Ahora se debe remitir la atención a su precio según factura. Si el precio es inferior a los $350,000.00 se trata de un vehículo de clase media (líneas 11-13) y por tanto el propietario solo debe pagar una cantidad de $600.00 más IVA por concepto de tenencia tipo VE (vehículo serie económica); de lo contario, se trata de un vehículo de clase alta (líneas 15-17) y su costo por concepto de tenencia es de $1,500.00 más IVA (vehículo serie de lujo).

Nuevamente observe la anidación de cada una de estas estructuras de selección y valore la importancia de comentar, pero, sobre todo, de aplicar sangría a cada bloque de instrucciones. En programas cuyo contenido es extenso, estas prácticas son las más recomendables, puesto que permiten presentar el código de manera clara en cuanto a su estructura, y ofrecen al programador una serie de informaciones útiles sobre la funcionalidad de cada bloque de instrucciones, o una instrucción en particular.

Finalmente, para mejorar la presentación y legibilidad de los resultados emitidos en la ventana del navegador, la instrucción ***echo*** lleva a cabo una serie de concatenaciones a partir de tres contenidos diferentes sobre una misma línea de salida. Es decir, en primer lugar, se indica un mensaje descriptivo a manera de concepto, posteriormente se coloca un resultado almacenado mediante una variable, y luego se adjunta una etiqueta HTML ***
*** para producir un salto de línea (líneas 26-30).



¿Sabía qué?

La mayoría de los errores que se comenten durante la codificación de un programa mediante el Lenguaje de Programación PHP, se debe a la falta de un punto y coma ";" al término de cada instrucción u omitir la declaración de una llave de cierre "}" al finalizar un bloque de instrucciones (por ejemplo, estructura de control o la definición de una función).

Resultados PSeInt y PHP

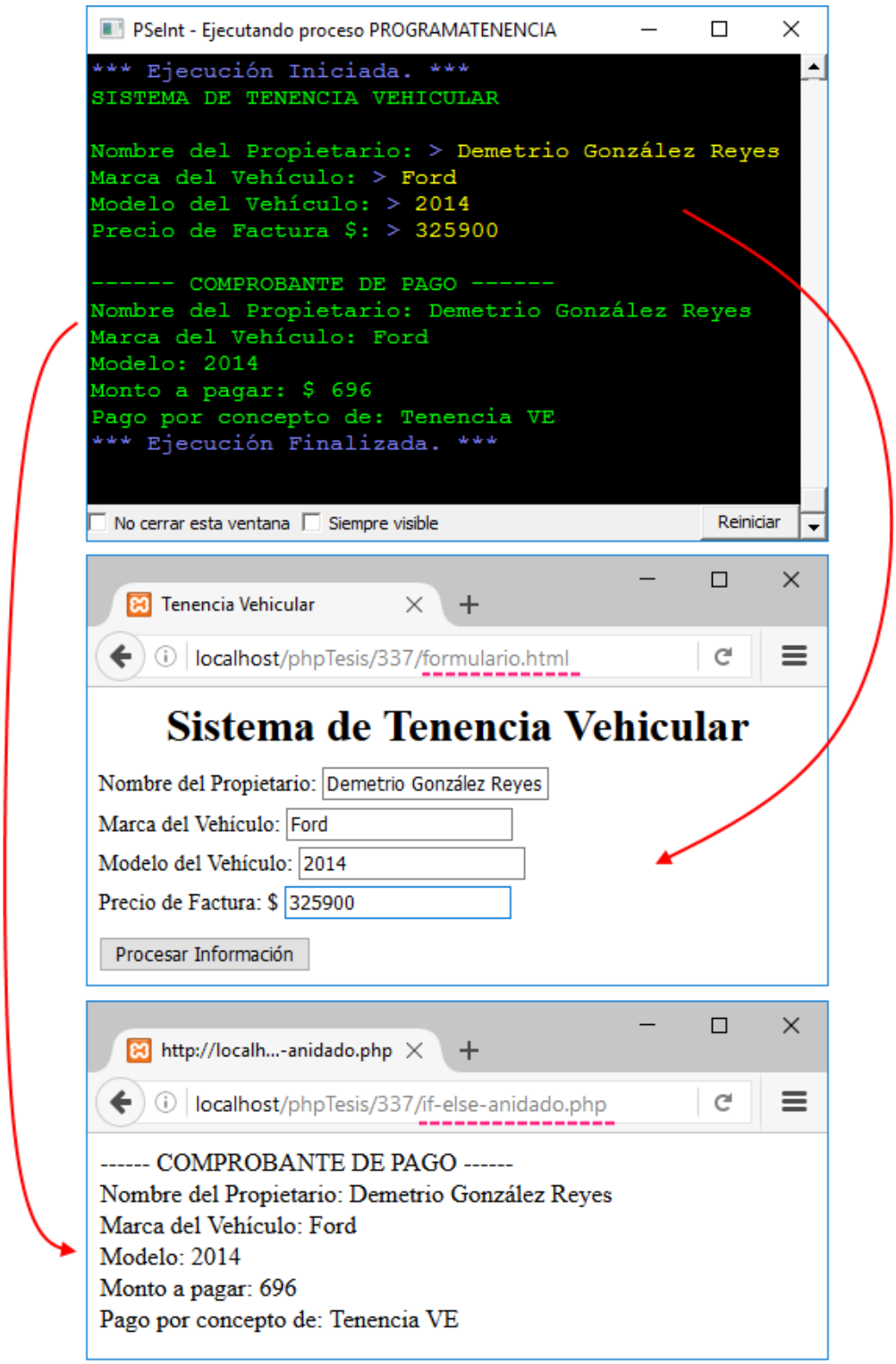


Imagen 122: Comparativo Ejecución PSeInt vs PHP - Fuente: Diseñada por el autor.

3.3.3 Tomar múltiples decisiones

Problema: ¿Desea desarrollar un programa de computadora para determinar el monto total a cobrar por concepto de colegiatura escolar?

Elementos gramaticales que intervienen en la solución del problema: Identificadores, variables, entradas y salidas de información, operadores aritméticos, estructura de selección múltiple, expresiones, asignaciones, comentarios, acarreo de información y operador de texto.

Descripción detallada del problema: El Centro Universitario de Tenango del Valle AC., le pide a usted desarrollar un programa de computadora mismo que, al ingresar el nombre de un alumno, el número de colegiaturas pendientes, y el día de pago. Determine el total a cobrar por concepto de colegiatura(s) para dicho alumno. Según los estatutos establecidos por la Institución, la colegiatura tiene un costo congelado de $1,400.00. Pero, si ésta se paga en los primeros cuatro días de cada mes, por promoción, es posible obtener un descuento que va desde los $50.00 hasta los $400.00. Por ejemplo:

Descuentos en colegiaturas				
Día 1	Día 2	Día 3	Día 4	Resto del mes
$400.00	$300.00	$200.00	$50.00	$0.00

Tabla 40: Relación de descuentos por pronto pago en colegiaturas - Fuente: Elaborada por el autor.

Sin embargo, este tipo de promociones solo son válidas para colegiaturas del mes, es decir, en el caso de que el alumno tenga un adeudo por concepto de colegiaturas pendientes, éstas no entran en promoción y por tanto se deberá cubrir su costo real de $1,400.00.

Finalmente, el programa debe desplegar un informe detallado con los datos del alumno, el número de colegiaturas pendientes (cuyo monto ya ha sido contabilizado en este recibo), el total a pagar y una leyenda que exhorte al alumno a pagar sus colegiaturas antes del día cinco de cada mes para ser acreedor a un descuento.

```
Proceso sin_titulo
    ///Entrada de Información
    Imprimir "Centro Universitario de Tenango del Valle"
    Imprimir "     Sistema de Pago de Colegiaturas     "
    Imprimir ""
    Imprimir "Nombre del Alumno: " Sin Saltar
    Leer alumno
    Imprimir "Colegiaturas Pendientes: " Sin Saltar
    Leer colegiaturasPendientes
    Imprimir "Día Fecha de Pago: " Sin Saltar
    Leer diaPago
    ///Proceso de Información
    //Colegiatura congelada de $1,400
    //Los primeros 4 días del mes hay descuentos en el pago
    subtotal = 1400 * colegiaturasPendientes
    Segun diaPago Hacer
        1:
            subtotal = subtotal + 1000
        2:
            subtotal = subtotal + 1100
        3:
            subtotal = subtotal + 1300
        4:
            subtotal = subtotal + 1350
        De Otro Modo:
            subtotal = subtotal + 1400
    FinSegun
    ///Salida de Información
    Imprimir ""
    Imprimir "--- COMPROBANTE DE PAGO DE COLEGITURA ---"
    Imprimir ""
    Imprimir "Nombre del Alumno: " , alumno
    Imprimir "Colegiaturas Pendientes: " , colegiaturasPendientes
    Imprimir "Total a Pagar: $" , subtotal
    Imprimir "Pague antes del día 5 para obtener descuentos"
FinProceso
```

Imagen 123: Pseudocódigo PSeInt - Fuente: Diseñada por el autor.

Extracto

```
13  <body>
14      <h1>Centro Universitario de Tenango del Valle</h1>
15      <h2>Sistema de Pago de Colegiaturas</h2>
16      <form action="switch-case.php" method="POST">
17          <label>Nombre del Alumno: </label>
18          <input type="text" name="campo_alumno">
19          <br>
20          <label>Colegiaturas Pendientes: </label>
21          <input type="text" name="campo_colegiaturasPendientes">
22          <br>
23          <label>Día Fecha de Pago: </label>
24          <input type="text" name="campo_diaPago">
25          <br>
26          <input type="submit" value="Procesar Información">
27      </form>
28  </body>
```

Imagen 124: Formulario HTML - Fuente: Diseñada por el autor.

Explicación Extracto de Formulario HTML:

El formulario HTML (líneas 16-27) internamente se construye a partir de siete controles: tres etiquetas ***<label>*** (líneas 17, 20 y 23), tres campos de texto (líneas 18, 21 y 24), y un botón (línea 26) para el envío de la información. Además, el formulario envía tres partes de información al archivo de programa **switch-case.php** para su correspondiente procesamiento, las cuales son enviadas mediante el método ***POST*** y están identificadas bajo los nombres de "**campo_alumno**", "**campo_diaPago**", y "**campo_colegiaturasPendientes**".

```php
<?php
    /*** Nombre del archivo: switch-case.php ***/

    $alumno = $_POST["campo_alumno"];
    $colegiaturasPendientes = $_POST["campo_colegiaturasPendientes"];
    $diaPago = $_POST["campo_diaPago"];

    //Colegiatura congelada de $1,400
    //Los primeros 4 días del mes hay descuentos en el pago
    $subtotal = 1400 * $colegiaturasPendientes;
    switch ($diaPago) {
        case 1:
            $subtotal = $subtotal + 1000;
            break;
        case 2:
            $subtotal = $subtotal + 1100;
            break;
        case 3:
            $subtotal = $subtotal + 1300;
            break;
        case 4:
            $subtotal = $subtotal + 1350;
            break;
        default:
            $subtotal = $subtotal + 1400;
    }

    echo "--- COMPROBANTE DE PAGO DE COLEGIATURA ---<br>";
    echo "<br>";
    echo "Nombre del Alumno: " , $alumno , "<br>";
    echo "Colegiaturas Pendientes: " , $colegiaturasPendientes , "<br>";
    echo "Total a Pagar: $" , $subtotal , "<br>";
    echo "Pague antes del día 5 para obtener descuentos";
?>
```

Imagen 125: Programa PHP - Fuente: Diseñada por el autor.

Explicación PHP:

El programa PHP encargado de procesar la información enviada por el formulario HTML es **switch-case.php**, y comienza con la etiqueta de apertura **<?php** y finaliza con **?>** (líneas 1-34). En un primer momento se

procede a recuperar las partes de información enviadas desde el formulario haciendo uso del arreglo asociativo ***$_POST***, y se almacenan en variables independientes para su correspondiente procesamiento dentro del programa (líneas 4-6). Luego, en la línea de código número 10, se procede a calcular el monto por concepto de colegiaturas pendientes, mismo que se debe considerar (acumular) en el pago de la colegiatura actual que lleva a cabo el alumno.

Bajo este mismo orden de ideas, en la línea 11 se analiza el valor del día de pago mediante una estructura de selección múltiple ***switch-case***. En caso de que coincida con el día 1, el pago de colegiatura actual es de $1,000.00. En caso de ser día 2, la colegiatura es de $1,100.00. Si el día de pago es 3, la colegiatura tiene un costo de $1,300.00. En caso de coincidir con el día 4, la colegiatura es de $1,350.00. En caso contrario, es decir, del día 5 en adelante, la colegiatura carece de descuento y su costo debe ser de $1,400.00.

Observe nuevamente la estructura de selección múltiple ***switch-case*** de PHP y analice las diferencias con respecto al Pseudocódigo de PSeInt. La expresión a comprobar debe ir encerrada entre paréntesis. El cuerpo o contenido (de dicha estructura) va delimitado entre llaves de apertura y cierre. Los posibles casos (opciones) son identificados previamente con la palabra reservada ***case***. Al finalizar las instrucciones correspondientes a cada caso, se debe escribir la palabra reservada ***break*** para impedir que se sigan ejecutando las instrucciones declaradas en los siguientes casos. Además, si existe la necesidad de ejecutar un conjunto de instrucciones por defecto (en caso de que el valor de la expresión no coincida con alguno de los casos especificados), estas se deben declarar como un caso especial haciendo uso de la palabra reservada ***default***.

Finalmente, para mejorar la presentación y legibilidad de los resultados emitidos en la ventana del navegador, la instrucción ***echo*** lleva a cabo una serie de concatenaciones a partir de tres contenidos diferentes sobre una misma línea de salida. Es decir, en primer lugar, se indica un mensaje descriptivo a manera de concepto, posteriormente se coloca un resultado devuelto por una variable, y luego se adjunta una etiqueta HTML ***
*** para producir un salto de línea (líneas 30-33).



¿Sabía qué?

Otra forma de recuperar el contenido enviado a través de los campos de un formulario HTML es mediante el Array asociativo **$_REQUEST[ ]** de PHP. La principal ventaja de este método consiste en que el programador no tiene por qué preocuparse si los datos han sido enviados por GET o POST en el formulario, puesto que su acceso es indiferente en el código.

Resultados PSeInt y PHP

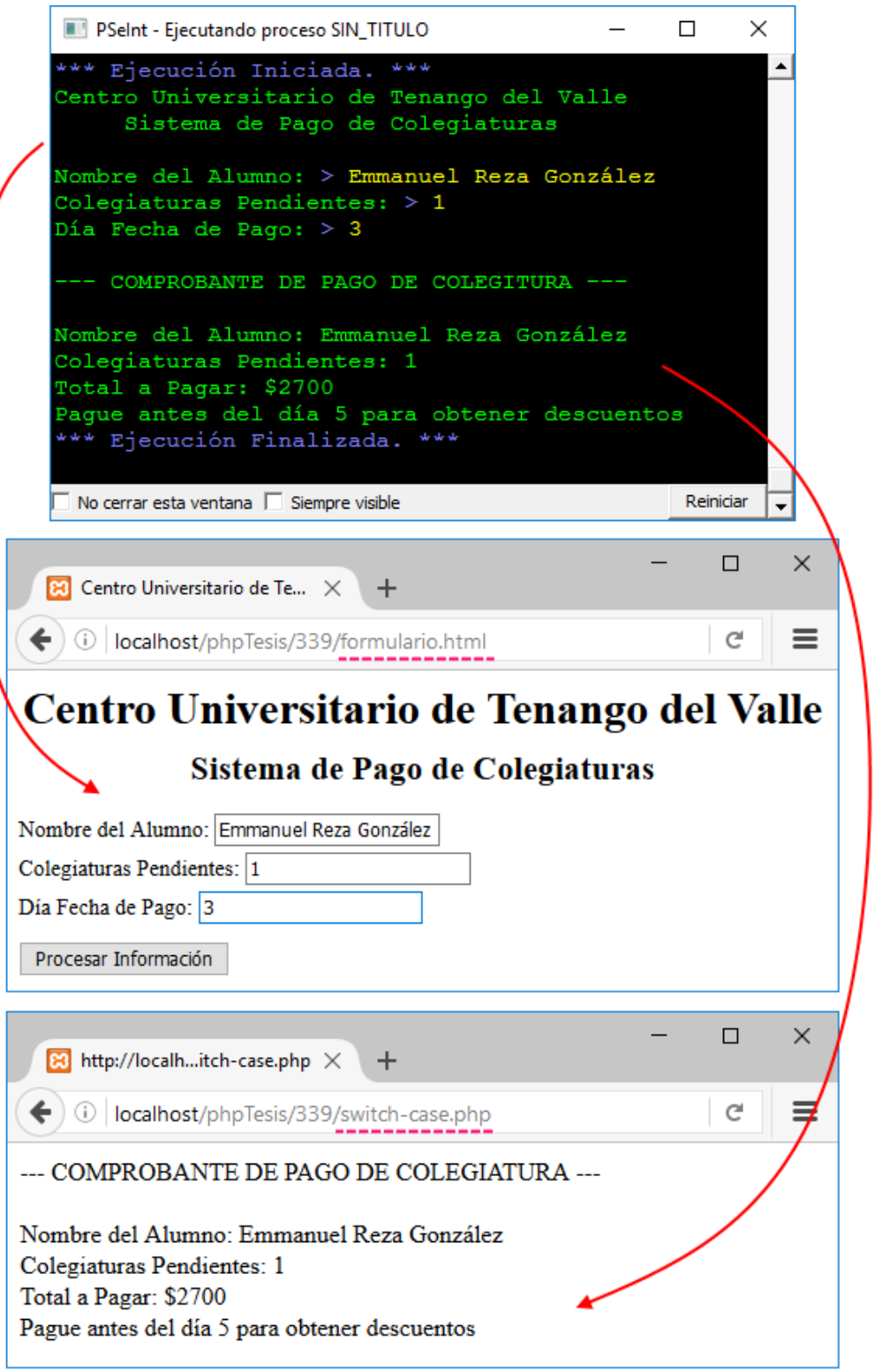


Imagen 126: Comparativo Ejecución PSeInt vs PHP - Fuente: Diseñada por el autor.

3.3.4 Repetir instrucciones

Problema: ¿Desea desarrollar un programa de computadora para determinar los nuevos sueldos que percibirán sus empleados al inicio de cada año?

Elementos gramaticales que intervienen en la solución del problema: Identificadores, variables, entradas y salidas de información, operadores aritméticos, operadores de comparación, estructura de repetición, expresiones, asignaciones, comentarios, acarreo de información y operador de texto.

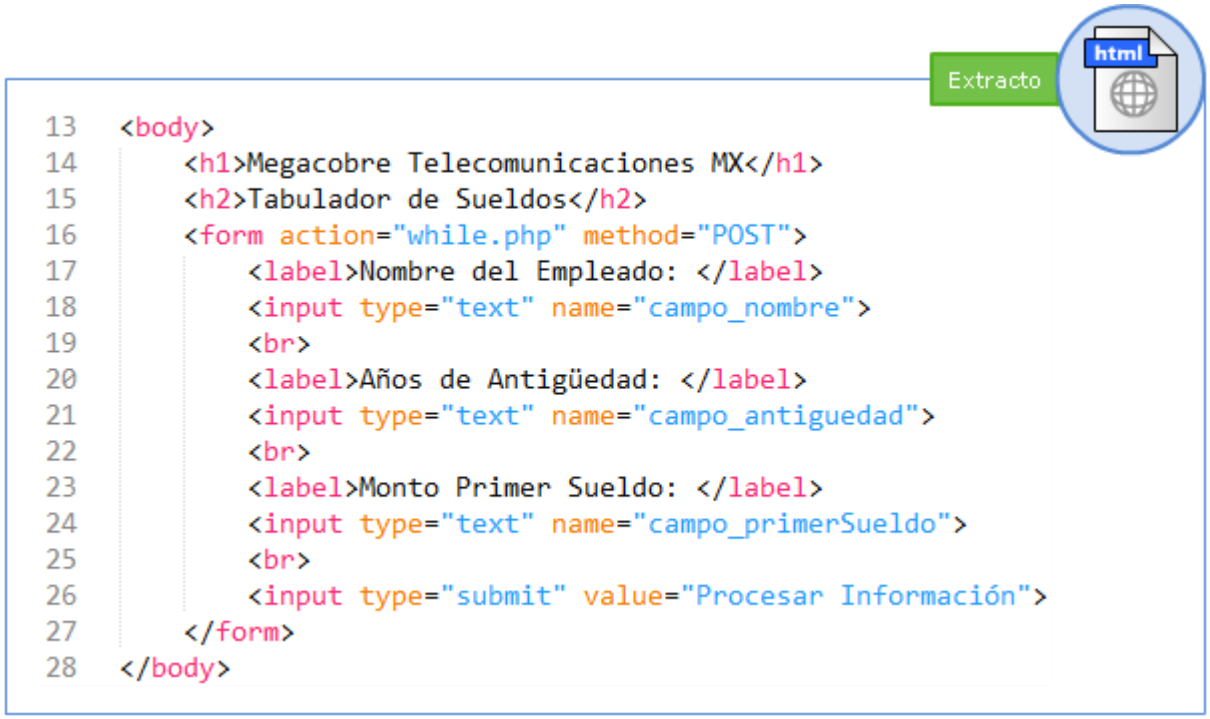
Descripción detallada del problema: La empresa Megacobre Telecomunicaciones MX, le pide a usted desarrollar un programa de computadora mismo que, al ingresar el nombre del empleado, los años de antigüedad, y el primer sueldo otorgado desde su contratación. Determine el sueldo que percibirá dicho empleado para este nuevo año laboral. Considere que, por estatutos establecidos por la propia empresa, cada inicio de año, todo empleado (sin excepción alguna) recibe un aumento de sueldo del 2%.

Finalmente, como parte del proceso principal. El programa debe mostrar un informe detallado (adjunto al resultado) acerca del historial de pagos que ha percibido dicho empleado desde el primer año de su contratación hasta el día de hoy.




```
Proceso sin_titulo
   ///Entrada de Información
   Imprimir "Megacobre Telecomunicaciones MX"
   Imprimir "      Tabulador de Sueldos     "
   Imprimir ""
   Imprimir "Nombre del Empleado: " Sin Saltar
   Leer nombre
   Imprimir "Años de Antigüedad:  " Sin Saltar
   Leer antiguedad
   Imprimir "Monto Primer Sueldo: " Sin Saltar
   Leer primerSueldo
   Imprimir ""
   Imprimir "-- Historial pagos: " , nombre , " --"
   Imprimir ""
   ///Proceso y Salida de Información
   Imprimir "Sueldo percibido año 1: $" , primerSueldo
   contador = 2
   //Generando historial de pago de sueldo por año laborado
   Mientras contador <= antiguedad Hacer
      //Aumento de sueldo en 2% por cada año laborado
      primerSueldo = primerSueldo + (primerSueldo * 0.02)
      Imprimir "Sueldo percibido año " , contador ,": $" , primerSueldo
      contador = contador + 1
   FinMientras
   Imprimir ""
   sueldoActual = primerSueldo + (primerSueldo * 0.02)
   Imprimir "Sueldo para este año: $" , sueldoActual
FinProceso
```

Imagen 127: Pseudocódigo PSeInt - Fuente: Diseñada por el autor.




```
13  <body>
14      <h1>Megacobre Telecomunicaciones MX</h1>
15      <h2>Tabulador de Sueldos</h2>
16      <form action="while.php" method="POST">
17          <label>Nombre del Empleado: </label>
18          <input type="text" name="campo_nombre">
19          <br>
20          <label>Años de Antigüedad: </label>
21          <input type="text" name="campo_antiguedad">
22          <br>
23          <label>Monto Primer Sueldo: </label>
24          <input type="text" name="campo_primerSueldo">
25          <br>
26          <input type="submit" value="Procesar Información">
27      </form>
28  </body>
```

Imagen 128: Formulario HTML - Fuente: Diseñada por el autor.

Explicación Extracto de Formulario HTML:

El formulario HTML (líneas 16-27) internamente se compone a partir de siete controles: tres etiquetas ***<label>*** (líneas 17, 20 y 23), tres campos de texto (líneas 18, 21 y 24), y un botón (línea 26) para el envío de la información. Asimismo, el formulario envía tres partes de información al archivo de programa **while.php** para su correspondiente procesamiento, las cuales son enviadas mediante el método ***POST*** y están debidamente identificadas bajo los nombres: "**campo_nombre**", "**campo_antiguedad**", y "**campo_primerSueldo**".



¿Sabía qué?

Existen otros elementos HTML que son comúnmente utilizados para el desarrollo de formularios Web, por ejemplo: **<textarea>**, **<select>**, **<option>**, **<button>** **<optgroup>**, **<fieldset>**, **<legend>**, **<datalist>**, **<keygen>**, **<output>**, **<progress>** y **<meter>**.



```php
<?php
   /*** Nombre del archivo: while.php ***/

   $nombre = $_POST["campo_nombre"];
   $antiguedad = $_POST["campo_antiguedad"];
   $primerSueldo = $_POST["campo_primerSueldo"];

   echo "-- Historial pagos: " , $nombre , " --<br>";
   echo "<br>";
   echo "Sueldo percibido año 1: $" , $primerSueldo , "<br>";
   $contador = 2;
   //Generando historial de pago de sueldo por año laborado
   while ($contador <= $antiguedad) {
      //Aumento de sueldo en 2% por cada año laborado
      $primerSueldo = $primerSueldo + ($primerSueldo * 0.02);
      echo "Sueldo percibido año ",$contador,": $",$primerSueldo,"<br>";
      $contador = $contador + 1;
   }

   echo "<br>";
   $sueldoActual = $primerSueldo + ($primerSueldo * 0.02);
   echo "Sueldo para este año: $" , $sueldoActual;
?>
```

Imagen 129: Programa PHP - Fuente: Diseñada por el autor.

Explicación PHP:

El programa PHP encargado de procesar la información lleva por nombre **while.php**, y comienza con la etiqueta de apertura **<?php** y finaliza con **?>** (líneas 1-23). En un primer momento se procede a recuperar las tres

partes de información enviadas por el formulario haciendo uso del arreglo asociativo ***$_POST***, y se almacenan en las variables **$nombre**, **$antiguedad**, y **$primerSueldo** para su correspondiente procesamiento (líneas 4-6). Posteriormente, como parte del proceso para generar el historial de pagos, la línea de código número 10 muestra el sueldo que percibió el empleado durante su primer año de contratación.

Bajo este mismo orden de ideas, si el empleado ha laborado en la empresa por más de un año, es necesario conocer el incremento que se le ha aplicado a su sueldo a lo largo de este tiempo. Para ello, la estructura de repetición ***while*** declarada en las líneas 13-18, permite recorrer cada uno de los años laborados (a partir del segundo), ejecutando en cada iteración un conjunto de instrucciones que se corresponden con el incremento del sueldo en un 2% para dicho año (línea 15), su impresión en pantalla para que figure en el historial de pagos (línea 16), y la actualización de la variable de control que impide la presencia de un ciclo infinito durante el proceso de ejecución (línea 17).

Sumado a lo expuesto, para mejorar la presentación y legibilidad de la información presentada en pantalla. La línea de código 20, lleva a cabo un salto de línea para separar el contenido referente al historial de pagos del nuevo sueldo que percibirá el empleado para el reciente año laboral (determinado en la línea 21).

Llegado a este punto, es importante mencionar que el código de programa anterior confía ciegamente en la información proporcionada por el usuario final, es decir, no se lleva a cabo algún tipo de validación (sobre la información de antigüedad) antes de comenzar a operar. Si bien este tipo de rutinas son indispensables por motivos de seguridad e integridad, su omisión en el ejemplo, garantiza su simplicidad y entendimiento.

Finalmente, vale la pena recordar que el comportamiento y configuración de la estructura repetitiva ***while*** de PHP son idénticos a los que proporciona PSeInt mediante la estructura repetitiva **Mientras**. En ambos casos es necesario especificar una condición, un bloque de instrucciones, y una variable de control para prevenir un comportamiento inesperado durante su ejecución (ciclo infinito).



¿Sabía qué?

Algunos sitios y proyectos Web exitosos que usan PHP para ofrecer sus servicios son: **WordPress**, **Facebook**, **Wikipedia**, **Drupal**, **Flickr**, **Moodle**, **Joomla**, **PrestaShop**, **Magento**, entre otros.

Resultados PSeInt y PHP

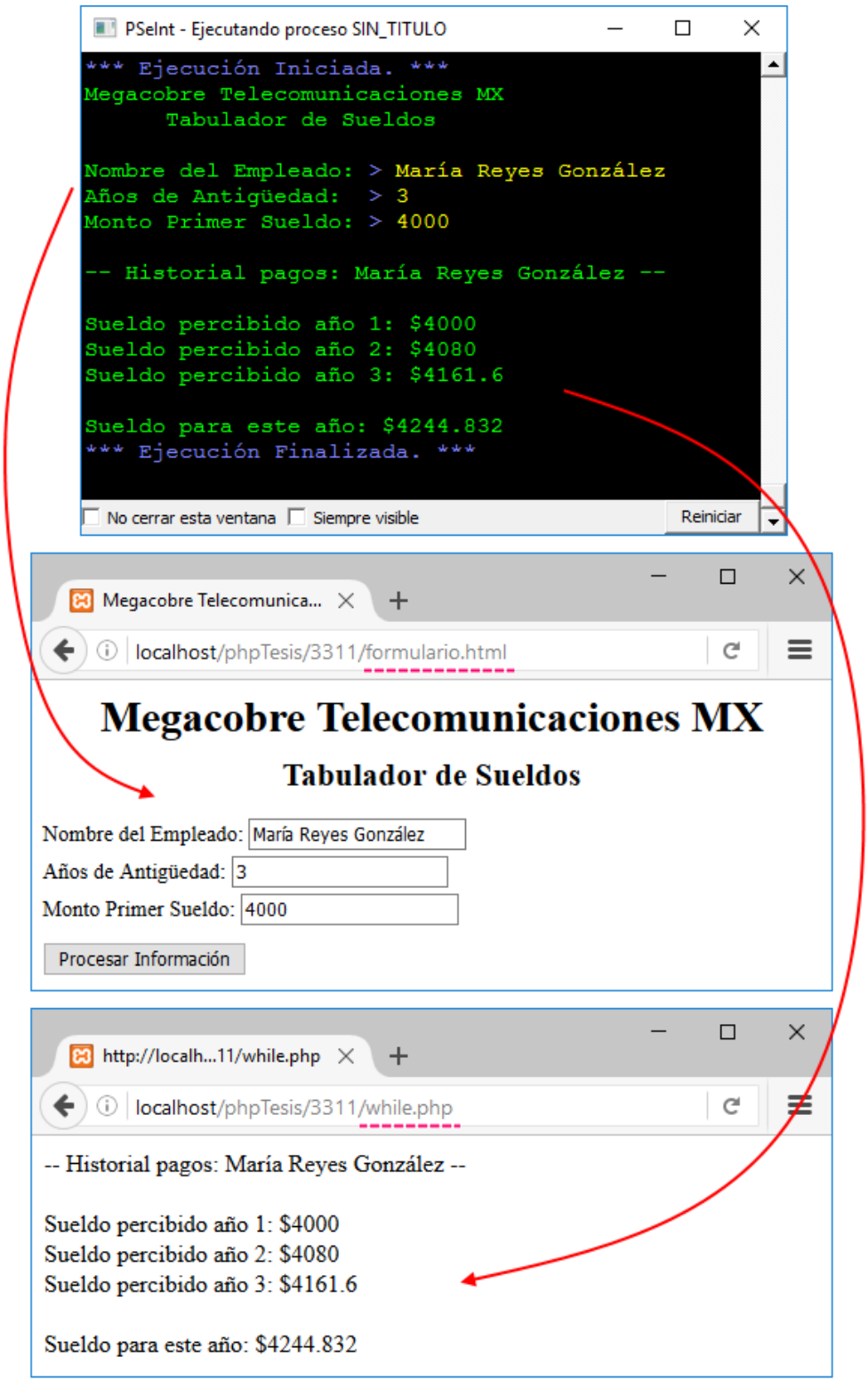


Imagen 130: Comparativo Ejecución PSeInt vs PHP - Fuente: Diseñada por el autor.

3.3.5 Trabajar con funciones pre-construidas

Problema: ¿Desea desarrollar un programa de computadora para el registro y generación de matrículas (placas) vehiculares?

Elementos gramaticales que intervienen en la solución del problema: Identificadores, variables, entradas y salidas de información, operadores de comparación, operadores lógicos, estructuras de selección, expresiones, asignaciones, comentarios, funciones pre-construidas (primitivas) y operador de texto.

Descripción detallada del problema: Por motivos de la nueva ley de tránsito perteneciente a la comunidad de San Miguel Almaya, el Departamento de Registro de Matricula Vehicular le pide a usted desarrollar un programa de computadora para la expedición de matrículas pertenecientes a vehículos de propulsión humana (bicicletas), mismas que deben estar conformadas a partir de un conjunto de reglas (patrón) establecidas por el propio Departamento:

1. La matrícula generada debe estar contener nueve caracteres separados en tres grupos por el símbolo menos, es decir:
   GRUPO1-GRUPO2-GRUPO3
2. El primer grupo de elementos debe estar formado por las iniciales del nombre, apellido paterno, y apellido materno del propietario del vehículo (en letras mayúsculas). Por ejemplo, para una persona de nombre Alejandro González Reyes, el primer grupo de elementos quedaría conformado por las iniciales: **AGR.** Sin embargo, si el nombre o los apellidos inician con las letras **O, I y Q.** Se debe reemplazar dichas iniciales por una letra **X**, es decir, para una

persona de nombre Daniel Osorno Villanueva, las iniciales que deben figurar en la matrícula de su vehículo serian: **DXV**

3. El segundo grupo de elementos está conformado por un número aleatorio que va en el rango de 1 hasta 999. Sin embargo, el formato de dicho grupo debe contener tres caracteres, es decir, si el número generado es 7, entonces debe figurar como 007. Si el número es 45, entonces debe aparecer como 045. Y si el número es 125, entonces en este caso se queda igual (como 125).
4. Finalmente, el tercer grupo de elementos se corresponde con uno de los siguientes identificadores que permiten conocer el uso que se da al vehículo registrado.
    - Identificador tipo **A**: Para vehículos de uso personal.
    - Identificador tipo **B**: Para vehículos de uso comercial.
    - Identificador tipo **C**: Para vehículos de uso profesional.

Con base en los supuestos anteriores, para el registro de una bicicleta de uso personal a nombre de Emmanuel Torres Quintana. Una posible matricula generada quedaría formada por los caracteres: **ETX-007-A**. Así mismo, para un conductor de nombre Miguel Angel González Reyes, cuyo registro es una bicicleta de uso profesional. Su matrícula generada podría contener los caracteres: **MGR-865-C**




```
Proceso programaMatricula
    ///Entrada de Información
    Imprimir "Departamento de Registro de Matrícula Vehicular"
    Imprimir "        Vehículos de Propulsión Humana        "
    Imprimir ""
    Imprimir "Ingrese nombre(s):        " Sin Saltar
    Leer nombre
    Imprimir "Ingrese primer apellido:  " Sin Saltar
    Leer primerApellido
    Imprimir "Ingrese segundo apellido: " Sin Saltar
    Leer segundoApellido
    Imprimir "A) Uso personal"
    Imprimir "B) Uso comercial"
    Imprimir "C) Uso profesional"
    Imprimir "Ingrese clase de uso [A,B,C]: " Sin Saltar
    Leer claseUso
    ///Proceso de Información
    //Obteniendo primer letra del nombre y apellidos en mayúsculas
    letra1 = MAYUSCULAS(SUBCADENA(nombre, 1, 1))
    letra2 = MAYUSCULAS(SUBCADENA(primerApellido, 1, 1))
    letra3 = MAYUSCULAS(SUBCADENA(segundoApellido, 1, 1))
    //Si el nombre o apellidos comienzan con letras O, I, Q
    //sustituir por una letra X
    Si letra1 == "O" O letra1 == "I" O letra1 == "Q" Entonces
        letra1 = "X"
    FinSi
    Si letra2 == "O" O letra2 == "I" O letra2 == "Q" Entonces
        letra2 = "X"
    FinSi
    Si letra3 == "O" O letra3 == "I" O letra3 == "Q" Entonces
        letra3 = "X"
    FinSi
    //Generando un número aletorio en el rango 1 a 999
    numeroGenerado = ALEATORIO(1,999)
    //El número de la matrícula debe ser en formato 00#, 0##, ###
    Si numeroGenerado < 10 Entonces
        numClave = CONCATENAR("00",CONVERTIRATEXTO(numeroGenerado))
    Sino
        Si numeroGenerado < 100 Entonces
            numClave = CONCATENAR("0",CONVERTIRATEXTO(numeroGenerado))
        Sino
            numClave = CONVERTIRATEXTO(numeroGenerado)
        FinSi
    FinSi
    ///Salida de Información
    //La matrícula vehicular debe ser en formato XXX-###-X
    Imprimir ""
    Imprimir "Matricula Vehicular: " Sin Saltar
    Imprimir letra1,letra2,letra3,"-",numClave,"-",MAYUSCULAS(claseUso)
FinProceso
```

Imagen 131: Pseudocódigo PSeInt - Fuente: Diseñada por el autor.

Extracto

```
13  <body>
14      <h1>Departamento de Registro de Matricula Vehicular</h1>
15      <h2>Vehículos de Propulsión Humana</h2>
16      <form action="funcionesPredefinidasPHP.php" method="POST">
17          <label>Ingrese nombre(s): </label>
18          <input type="text" name="campo_nombre">
19          <br>
20          <label>Ingrese primer apellido: </label>
21          <input type="text" name="campo_primerApellido">
22          <br>
23          <label>Ingrese segundo apellido: </label>
24          <input type="text" name="campo_segundoApellido">
25          <br>
26          <label>Seleccione clase de uso: </label>
27          <select name="seleccion_claseUso">
28              <option value="A">Uso personal</option>
29              <option value="B">Uso comercial</option>
30              <option value="C">Uso profesional</option>
31          </select>
32          <br>
33          <input type="submit" value="Procesar Información">
34      </form>
35  </body>
```

Imagen 132: Formulario HTML - Fuente: Diseñada por el autor.

Explicación Extracto de Formulario HTML:

Nuestro formulario HTML (líneas 16-34) internamente se construye a partir de nueve controles: cuatro etiquetas ***<label>*** (líneas 17, 20, 23 y 26), tres campos de texto (líneas 18, 21 y 24), un menú con tres opciones disponibles (líneas 27-31), y un botón (línea 33) para el envío de la información. Además, dicho formulario envía cuatro partes de información al archivo de programa **funcionesPredefinidasPHP.php** para su correspondiente procesamiento, las cuales son enviadas mediante el método ***POST*** y están identificadas bajo los nombres "**campo_nombre**",

"**campo_primerApellido**", "**campo_segundoApellido**" y "**seleccion_claseUso**".

```php
<?php
    /*** Nombre del archivo: funcionesPredefinidasPHP.php ***/

    $nombre = $_POST["campo_nombre"];
    $primerApellido = $_POST["campo_primerApellido"];
    $segundoApellido = $_POST["campo_segundoApellido"];
    $claseUso = $_POST["seleccion_claseUso"];

    //Obteniendo primer letra del nombre y apellidos en mayúsculas
    $letra1 = strtoupper(substr($nombre, 0, 1));
    $letra2 = strtoupper(substr($primerApellido, 0, 1));
    $letra3 = strtoupper(substr($segundoApellido, 0, 1));
    /*Si el nombre o apellidos comienzan con letras O, I, Q
    sustituir por una letra X*/
    if ($letra1 === "O" || $letra1 === "I" || $letra1 === "Q") {
        $letra1 = "X";
    }
    if ($letra2 === "O" || $letra2 === "I" || $letra2 === "Q") {
        $letra2 = "X";
    }
    if ($letra3 === "O" || $letra3 === "I" || $letra3 === "Q") {
        $letra3 = "X";
    }
    //Generando un número aletorio en el rango 1 a 999
    $numeroGenerado = rand(1, 999);
    //El número de la matrícula debe ser en formato 00#, 0##, ###
    if ($numeroGenerado < 10) {
        $numClave = "00" . strval($numeroGenerado);
    } elseif ($numeroGenerado < 100) {
        $numClave = "0" . strval($numeroGenerado);
    } else {
        $numClave = strval($numeroGenerado);
    }

    //La matrícula vehicular debe ser en formato XXX-###-X
    echo "Matricula Vehicular: ";
    echo $letra1,$letra2,$letra3,"-",$numClave,"-",strtoupper($claseUso);
?>
```

Imagen 133: Programa PHP - Fuente: Diseñada por el autor.

Explicación PHP:

El programa PHP encargado de procesar la información lleva por nombre **funcionesPredefinidasPHP.php**, y comienza con la etiqueta de apertura **<?php** y finaliza con **?>** (líneas 1-38). En un primer momento se procede a recuperar las partes de información enviadas desde el formulario HTML haciendo uso del arreglo asociativo ***$_POST***, y se almacenan en variables independientes para su correspondiente procesamiento dentro del programa (líneas 4-7). Posteriormente, con la ayuda de las funciones ***substr( )*** y ***strtoupper( )*** de PHP (**SUBCADENA** y **MAYUSCULAS** en PSeInt), en las líneas de código 10-12 se procede a obtener las letras iniciales correspondientes al nombre, apellido paterno, y apellido materno del propietario del vehículo (primer grupo de elementos), mismas que son almacenadas en las variables **$letra1**, **$letra2**, y **$letra3**. Pero, si alguna de estas coincide con los caracteres O, I o Q. Las instrucciones declaradas en las líneas 16, 19 y 22 se encargan de sobre-escribir su contenido por una letra X.

Bajo este mismo orden de ideas, para determinar el segundo grupo de elementos correspondientes a la matricula vehicular, la función ***rand( )*** de PHP (**ALEATORIO** en PSeInt) declarada en la línea 25, genera un número aleatorio desde 1 hasta 999. Mismo que es analizado mediante una estructura de selección ***elseif*** para garantizar que su formato consista de tres dígitos, es decir, si el número generado es menor a 10, este se convierte a un texto con la función ***strval( )*** de PHP y se le concatenan dos ceros en formato de cadena (mediante el operador de texto de PHP). Pero, en caso de que el número generado sea menor a 100, entonces se vuelve a convertir a texto con la función ***strval( )*** y solo se le concatena un cero. En caso contrario, el número ya contiene los tres dígitos requeridos y solo hace falta convertirlo a texto.

Finalmente, para presentar la matricula generada en la ventana del navegador, la instrucción ***echo*** lleva a cabo una serie de concatenaciones a partir de los contenidos almacenados en las variables **$letra1**, **$letra2**, **$letra3**, **$numClave** (número aleatorio generado en formato de tres dígitos), y **$claseUso**. Separándolos como es debido (tres grupos) con la ayuda de un signo menos (línea 37).

Como habrá podido observar, la invocación y comportamiento de las funciones pre-construidas en PHP son idénticos a los que provee el Pseudolenguaje PSeInt. Sin embargo, recuerde que, en la mayoría de las ocasiones, su disponibilidad y los nombres que distinguen a cada una de estas funciones, varían dependiendo del lenguaje de programación utilizado.



¿Sabía qué?

En el desarrollo Web profesional es importante validar y escapar toda información que proveen los usuarios a través de los controles de un formulario. PHP cuenta con algunas funciones integradas que permiten llevar a cabo este tipo de tareas, por ejemplo: **trim( )**, **isset( )**, **empty( )**, **is_bool( )**, **is_string( )**, **is_int( )**, **is_float( )**, **is_numeric( )**, **is_null( )**, **stripslashes( )**, **htmlentities( )**, **htmlspecialchars( )**, entre otras.

Resultados PSeInt y PHP

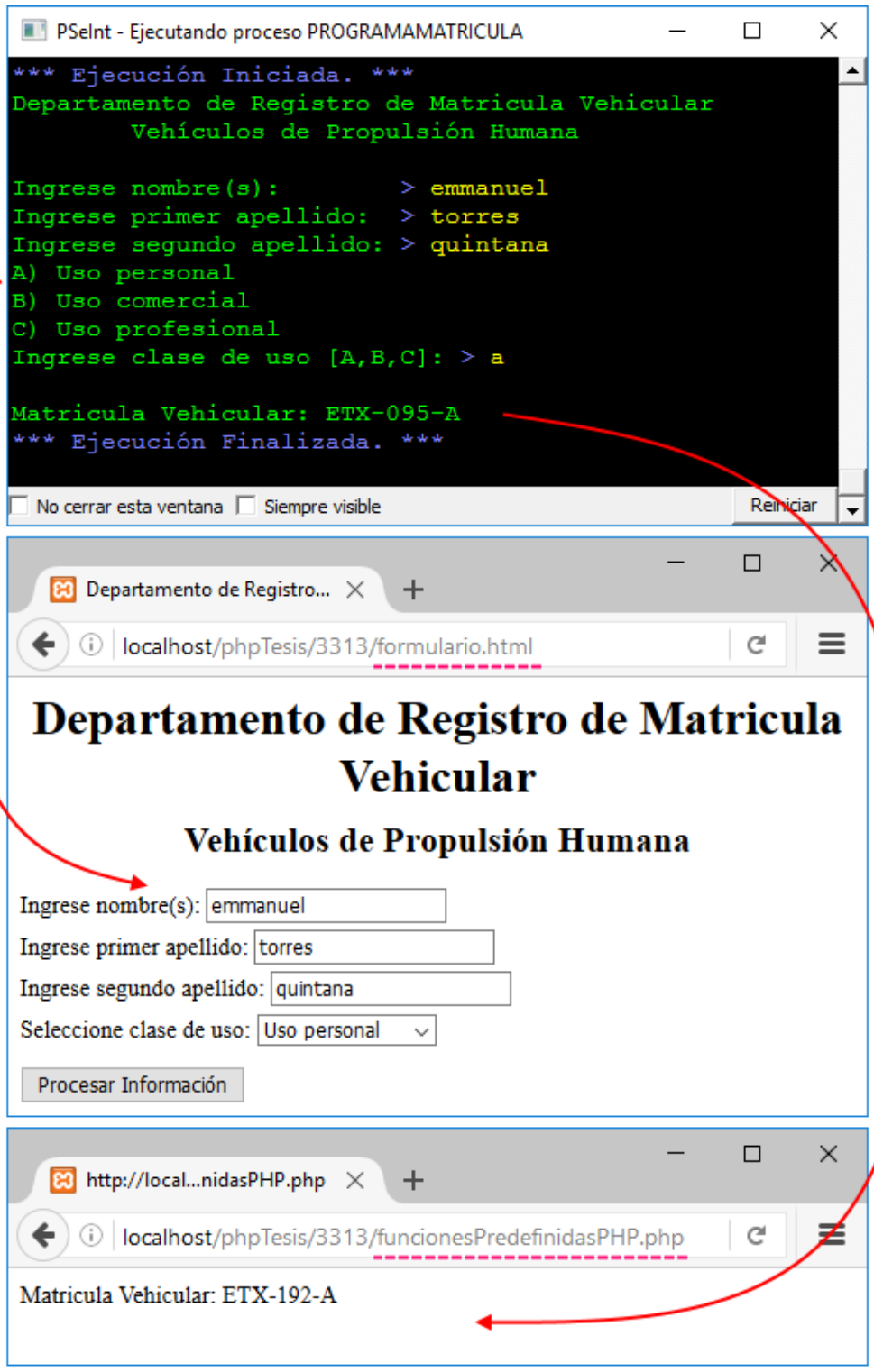


Imagen 134: Comparativo Ejecución PSeInt vs PHP - Fuente: Diseñada por el autor.

3.3.6 Trabajar con funciones personalizadas

Problema: ¿Desea desarrollar un programa de computadora para determinar el perímetro de un triángulo rectángulo a partir de su base y altura?

Elementos gramaticales que intervienen en la solución del problema: Identificadores, variables, entradas y salidas de información, operadores aritméticos, expresiones, asignaciones, comentarios, funciones pre-construidas y personalizadas, así como el operador de texto.

Descripción detallada del problema: Desarrolle una función que permita calcular el valor del perímetro correspondiente a un triángulo rectángulo a partir de datos conocidos como son su base y altura.




```
1  ///Función del programador
2  //Retorna el perímetro de un triángulo rectángulo a partir
3  //de datos conocidos como son su base y altura
4  Funcion perimetro = _PERIMETROTRIANGULO(base, altura)
5      hipotenusa = RAIZ((base ^ 2) + (altura ^ 2))
6      perimetro = base + altura + hipotenusa
7  FinFuncion
8
9  Proceso programaPrincipal
10     Imprimir "Funciones con valor de retorno"
11     Imprimir "  Trabajando con Perímetros   "
12     Imprimir ""
13     Imprimir "Base del Triángulo: " Sin Saltar
14     Leer baseTR
15     Imprimir "Ingrese altura: " Sin Saltar
16     Leer alturaTR
17     Imprimir ""
18     //Invocando la función desde el programa principal
19     Imprimir "El perímetro es: " , _PERIMETROTRIANGULO(baseTR, alturaTR)
20 FinProceso
```

Imagen 135: Pseudocódigo PSeInt - Fuente: Diseñada por el autor.

Extracto

```
13  <body>
14      <h1>Funciones con valor de retorno</h1>
15      <h2>Trabajando con Perímetros</h2>
16      <form action="funcion-valorRetorno.php" method="POST">
17          <label>Base del Triángulo: </label>
18          <input type="text" name="campo_baseTR">
19          <br>
20          <label>Altura del Triángulo: </label>
21          <input type="text" name="campo_alturaTR">
22          <br>
23          <input type="submit" value="Procesar Información">
24      </form>
25  </body>
```

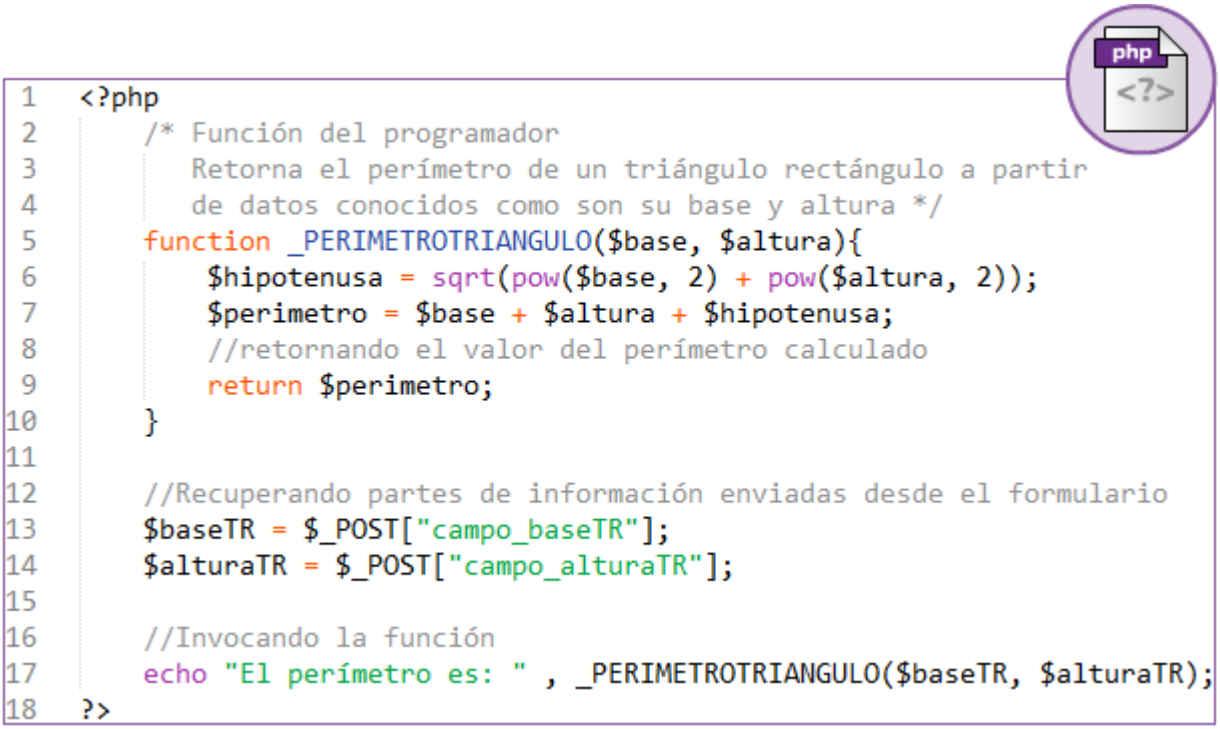
Imagen 136: Formulario HTML - Fuente: Diseñada por el autor.

Explicación Extracto de Formulario HTML:

El formulario (líneas 16-24) internamente se compone de cinco controles: dos etiquetas ***<label>*** (líneas 17 y 20), dos campos de texto (líneas 18 y 21), y un botón (línea 23) para el envío de la información. Además, este formulario envía dos partes de información al archivo de programa **funcion-valorRetorno.php** para su correspondiente procesamiento, mismas que son enviadas a través del método ***POST*** y están debidamente identificadas bajo los nombres "**campo_baseTR**" y "**campo_alturaTR**".

¿Sabía qué?

Los controles de tipo **<input>** también permiten a los usuarios elegir archivos desde su sistema, de modo que sus contenidos puedan ser enviados con un formulario. Sin embargo, para que esto pueda ser posible, es necesario que su atributo **type** contenga el valor establecido a **"file"**. Asimismo, se agregue un nuevo atributo en la etiqueta de formulario **<form>** llamado **enctype**, cuyo valor sea **"multipart/form-data"**.



```php
<?php
    /* Función del programador
       Retorna el perímetro de un triángulo rectángulo a partir
       de datos conocidos como son su base y altura */
    function _PERIMETROTRIANGULO($base, $altura){
        $hipotenusa = sqrt(pow($base, 2) + pow($altura, 2));
        $perimetro = $base + $altura + $hipotenusa;
        //retornando el valor del perímetro calculado
        return $perimetro;
    }

    //Recuperando partes de información enviadas desde el formulario
    $baseTR = $_POST["campo_baseTR"];
    $alturaTR = $_POST["campo_alturaTR"];

    //Invocando la función
    echo "El perímetro es: " , _PERIMETROTRIANGULO($baseTR, $alturaTR);
?>
```

Imagen 137: Programa PHP - Fuente: Diseñada por el autor.

Explicación PHP:

El programa PHP encargado de procesar la información recibe por nombre **funcion-valorRetorno.php** e inicia con la etiqueta de apertura **<?php** y finaliza con **?>** (línea 1-18). Internamente, en las líneas 5-10 se define la función personalizada **_PERIMETROTRIANGULO** que acepta dos parámetros de entrada "**$base** y **$altura**", mismos que representan las medidas de la base y altura correspondientes al triángulo rectángulo del que se desea obtener su perímetro.

Bajo este mismo orden de ideas, en la línea 6 se procede a determinar la hipotenusa de dicho triangulo rectángulo (mediante el teorema de Pitágoras), misma que hace uso de los valores de entrada (**$base** y **$altura**) proporcionados a la función. Luego, en la línea de código número 7 se determina su perímetro (sumando las distancias de sus tres lados) y

en la línea 9, se retorna este resultado (perímetro) como valor devuelto por la función tras su ejecución.

Más tarde, en las líneas 13-14 se recuperan las dos partes de información enviadas por el formulario HTML haciendo uso del arreglo asociativo ***$_POST***, y se almacenan en las variables **$baseTR** y **$alturaTR** para su correspondiente procesamiento dentro del programa.

Sumado a lo expuesto, en la línea de código 17 se invoca a la función **_PERIMETROTRIANGULO** definida con anterioridad, indicando entre sus paréntesis el valor que guardan cada uno de sus argumentos, mismos que se corresponden con la información enviada a través del formulario HTML, y que internamente se le ha recuperado y almacenado en el programa PHP mediante las variables **$baseTR** y **$alturaTR**.

Finalmente, como la función **_PERIMETROTRIANGULO** retorna un valor al momento de su invocación, es importante que éste se le atrape a través de una variable, se utilice directamente en una expresión, o en su defecto, se le imprima directamente en pantalla (línea 17). De lo contrario, se corre el riesgo de que el lenguaje de programación origine un error, debido a que no sabe qué hacer con dicho valor.



¿Sabía qué?

Es posible escribir toda la lógica de las funciones PHP en un solo archivo y posteriormente invocarlo con la ayuda de la función **require_once( )**. Esto permite simplificar aún más el código principal de los programas y fomenta a reutilizar parte de estas soluciones (funciones) en el desarrollo de nuevos proyectos.

Resultados PSeInt y PHP

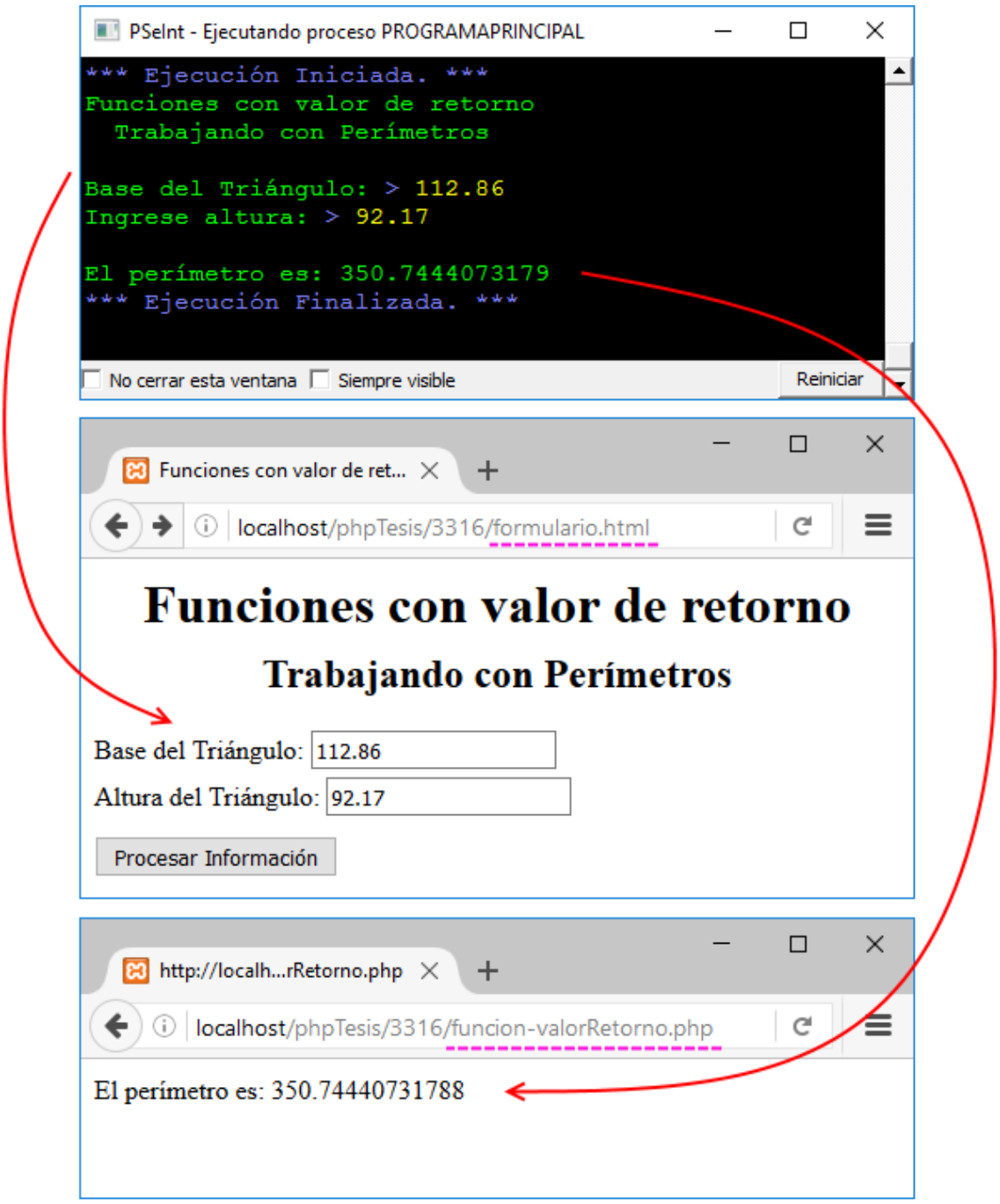


Imagen 138: Comparativo Ejecución PSeInt vs PHP - Fuente: Diseñada por el autor.

Con esta lección se da por terminado el apartado correspondiente a Soluciones de programación: Migrar del Pseudocódigo al Lenguaje de

Programación PHP. No obstante, como se ha podido observar a lo largo de estos seis ejercicios, en términos generales, el Lenguaje de Programación PHP mantiene una estrecha relación sintáctica con el Pseudolenguaje que propone PSeInt para el diseño y desarrollo de Algoritmos. Ahora que se tienen los fundamentos sobre el alcance y las funciones que desempeñan cada uno de estos elementos gramaticales durante la fase de codificación, es momento de dar el siguiente paso en su formación académica como programador profesional. Para ello, se sugiere regresar nuevamente a PSeInt e investigar todo lo referente a las estructuras de datos mediante la declaración de arreglos (Dimensiones en la terminología de PSeInt), puesto que una comprensión sólida sobre cada uno de estos tópicos, le permitirá continuar sin problemas en el trabajo con datos que provienen desde fuentes externas, tales como: una base de datos, un archivo de texto, una API[17] de terceros, entre otros.

Conclusión

Una vez analizado detalladamente los aspectos básicos e intermedios presentes tanto en el Pseudocódigo como en el Lenguaje de Programación PHP y la puesta en marcha de cada uno de ellos durante la solución de seis casos prácticos, merece la pena detenerse un momento y resaltar nuevamente la importancia de los **Pseudocódigos** para describir de manera informal las instrucciones declaradas en un Algoritmo. Puesto que, como se pudo observar a lo largo de la redacción del presente capítulo, la mayoría de los elementos gramaticales, estructuras y funciones presentes a nivel de Pseudocódigo, son por poco similares a las que se emplean

[17] La Interfaz de Programación de Aplicaciones (API) es un conjunto de funciones que ofrecen ciertas bibliotecas para ser utilizadas por otro *software* como una capa de abstracción. Ejemplos clásicos de estas API se pueden encontrar en sitios Web o aplicaciones móviles que consumen datos desde Google Maps, autenticación de usuarios mediante alguna red social como Facebook o Twitter, etc.

durante el desarrollo de proyectos Web profesionales; lo que genera, en que sea considerada como una **técnica de carácter indiscutible** que se debe abordar durante los primeros inicios de la formación de un futuro programador.

Llegado a este punto de la investigación, se espera que el lector por fin comprenda el significado real de la programación y, por consiguiente, dedique una cantidad mayor de tiempo en el análisis de los problemas. Puesto que, como se ha comentado con anterioridad, los Algoritmos son más importantes que los Lenguajes de Programación; dado que estos últimos solo son un medio para expresarlos.

Si bien es cierto que cada Lenguaje de Programación se distingue de los demás por incorporar un conjunto extenso de herramientas y funciones que facilitan el trabajo de un programador durante la fase de desarrollo de determinados proyectos. Es importante recordar al lector que la mayoría de estos recursos solo cubren algunas necesidades básicas de uso común; por ejemplo: la conexión a una base de datos, el acceso a archivos para su correspondiente lectura y escritura, la encriptación y cifrado de datos, etc. Sin embargo, el uso que se le dé a los datos generados a partir de esos recursos, dependerá de las necesidades de cada proyecto. Por tanto, la función de un programador dentro de este escenario, consistirá en manipular y gestionar cada uno de esos datos (a través de instrucciones simples, estructuras de control, funciones predefinidas, etc.) para convertirlos en información útil que genere valor agregado a cada uno de sus clientes, misma que facilite los accesos para una buena toma de decisiones a nivel organizacional, empresarial o personal.

Finalmente, con la intensión de resumir y relacionar cada uno de los factores expuestos dentro de este capítulo, se propone el uso del siguiente

decálogo como material de apoyo para la migración de soluciones escritas en Pseudocódigo a Lenguaje de Programación PHP.

Decálogo migrar
del Pseudocódigo al Lenguaje de Programación PHP



01 Los programas PHP inician y terminan con etiquetas **<?php** y **?>**

Reemplazaré las palabras reservadas **Proceso** y **FinProceso**

02 Las variables en PHP comienzan con el símbolo de dólar **$**

03 Las instrucciones de entrada, salida y asignación terminan con un punto y coma en PHP

04 La mayoría de las palabras reservadas en PSeInt tienen su equivalente a PHP

print, if, else, while, do, for, switch, default, function.

05 La mayoría de las entradas de información en PHP provienen de controles de formulario HTML

Omitiré la palabra reservada **Leer**

06 Las potencias en PHP se llevan a cabo mediante la declaración de la función **pow(x,n)**

Reemplazaré el acento circunflejo **^**

07 Los operadores lógicos en PHP se declaran mediante símbolos **||** (pleca), **&&** (ampersand) y **!** (exclamación)

Reemplazaré las palabras reservadas **Y, O, NO.**

08 La mayoría de las funciones predefinidas en PSeInt tienen su equivalente directo en PHP

sqrt( ), abs( ), sin( ), cos( ), tan( ), log( ), strtoupper( ).

09 La definición de estructuras y funciones en PHP delimitan sus instrucciones entre llaves **{ }**

Reemplazaré las palabras reservadas **Entonces, Hacer, FinSi, FinSegun, FinPara, FinFuncion...**

10 Las funciones con valor de retorno en PHP deben declarar la palabra reservada **return** siempre al final del cuerpo de la función

Conclusiones

Con base en el objetivo principal que persigue este trabajo de investigación: Desarrollo de una guía teórico práctica que dote de conocimientos a los estudiantes del área de Diseño Digital de ICONOS sobre temas relacionados con los fundamentos de programación básica mediante el uso de un Pseudolenguaje que les permita centrarse en los aspectos más importantes de la programación; sin tener que preocuparse por la sintaxis estricta de un lenguaje de programación formal como es el caso de PHP, se tiene que, tras finalizar su redacción; 18 de cada 20 aspectos considerados a lo largo de los primeros dos capítulos, satisfacen las expectativas inicialmente planteadas. Puesto que, un usuario que aborde los fundamentos de la programación a través de un Pseudolenguaje flexible, intuitivo y en español, le permite mejorar progresivamente su habilidad de razonamiento lógico-matemático para la solución de problemas mediante el uso de un ordenador.

En este sentido, los conceptos y ejemplos analizados en el Capítulo 1 relacionados con los fundamentos Gramaticales del Pseudocódigo mediante Programación Secuencial, permitieron que se verificara parcialmente la hipótesis que argumenta que los elementos gramaticales necesarios para aprender las bases fundamentales de la algoritmia computacional son:

a) Identificadores
b) Variables y Constantes
c) Entradas y Salidas de Información

d) Tipos de Datos (Numéricos, Caracteres y Booleanos)
e) Tipos de Operadores (Aritméticos, Comparación y Lógicos)
f) Comentarios

Puesto que, la mayoría de los ejemplos abordados dentro de este numeral omitieron la definición y uso del elemento gramatical de tipo **Constante** para la solución problemas. Además, se pudo observar cierta mejora progresiva en aquellos ejemplos que utilizaron el **Operador de texto** como recurso estético para concatenar sus correspondientes salidas de información, dado que este mecanismo, permitió fomentar considerablemente la comprensión de los resultados exhibidos al usuario.

De esta manera, la tesis establece que: los elementos gramaticales necesarios para aprender las bases fundamentales de la algoritmia computacional son:

a) Identificadores
b) Variables
c) Entradas y Salidas de Información
d) Tipos de Datos (Numéricos, Caracteres y Booleanos)
e) Tipos de Operadores (Aritméticos, Comparación, Lógicos y de Texto)
f) Comentarios

Por otra parte, tal y como se comentó en el segundo Capítulo de esta misma investigación. En la mayoría de las ocasiones, los problemas resueltos a través del uso de un ordenador requieren de un análisis más detallado que una simple sucesión de instrucciones declaradas de forma secuencial. En este sentido, las **Estructuras de Control** permiten al desarrollador elaborar soluciones flexibles de programación capaces de tomar decisiones durante la fase de ejecución, así como repetir un

determinado conjunto de instrucciones. Todo ello, con base en la toma de una o varias decisiones.

Sumado a lo expuesto, en la medida que crece la complejidad y extensión de los programas, se torna cada vez más necesario reutilizar parte de su proceso de solución para simplificar su correspondiente lectura, escritura y comprensión. Por tal motivo, la declaración de **Funciones** permite al programador encapsular tareas específicas (bien definidas), de modo que sea posible invocarlas en diferentes partes del proceso y reutilizar sus resultados generados como parte de la solución general del problema.

En este sentido, a partir de los múltiples ejemplos y definiciones tratadas a lo largo de la redacción de dicho Capítulo, se tiene que, la hipótesis que argumenta que los elementos gramaticales necesarios para aprender las bases fundamentales de la algoritmia computacional mediante programación estructurada y funcional son:

a) Estructuras de Control (Selección y Repetición)
b) Funciones Primitivas del Lenguaje
c) Funciones del Programador

Queda completamente verificada. Puesto que, la inclusión de diferentes estructuras de control en el código de un programa, permite que se modifique el flujo de su ejecución y pueda ser presentado ante los usuarios con un comportamiento más o menos inteligente y dinámico, debido a que puede realizar diferentes acciones (con base a una o varias condiciones) durante el proceso de su ejecución. Por otra parte, al intentar dividir un problema en pequeños subproblemas cuyas respuestas se encuentran agrupadas por funcionalidad, es posible atender la solución general del mismo desde un contexto colaborativo y mejor organizado,

debido a que la suma de sus partes, orientan al programador a reutilizar parte de la solución generada y, evitar así, el tener que volver a escribir instrucciones definidas con anterioridad como parte del mismo proceso.

Llegado a este punto, se puede decir que los objetivos particulares inicialmente planteados para este proyecto de investigación se cumplieron, puesto que la Metodología utilizada para aprender las bases fundamentales de la Programación a través del Pseudocódigo, permitió que se vislumbrara en el Capítulo tercero, **una relación casi directa con el Lenguaje de Programación PHP desde la perspectiva de su sintaxis**.

De modo que, los esquemas y ejemplos tratados a lo largo de dicho apartado, dejan de manifiesto que los **Pseudocódigos** como herramienta para el diseño de Algoritmos, deben ser considerados como una **técnica de carácter indiscutible** que se debe abordar desde los primeros inicios de la formación de un futuro programador, dado que le permiten centrar su atención en el análisis y desarrollo de diferentes alternativas de solución, sin tener que preocuparse por la sintaxis estricta que conlleva adoptar un lenguaje de programación, así como la carga de conceptos técnicos clave relacionados con su implementación.



Imagen 139: Comparativo tecnologías involucradas en el Desarrollo Web *vs* PSeInt – Fuente: Adaptada por el autor - http://codigotech.com/cuanto-gana-un-programador-en-argentina/2016/04/12/

Dentro de este mismo orden de ideas, se sugiere que, para futuros proyectos de investigación relacionados con los tópicos generales tratados en este documento, se aborde todo lo concerniente al trabajo con **Estructuras de Datos** a través de la declaración de **arreglos**. Si bien, hoy en día las Bases de Datos relacionales y documentales cubren dichas necesidades al implementar un lenguaje de consulta para la recuperación de información; los Lenguajes de Programación sólo pueden acceder a estos recursos (Bases de Datos) a través controladores de conexión, los cuales una vez intercomunicados con las fuentes de información (generalmente externa), recuperan el conjunto de datos solicitado por el usuario y proceden a almacenarlos en variables del tipo arreglo (estructuras de datos) para futuras tareas de procesamiento dentro del programa o aplicación; entre las más comunes se encuentran, el recorrido de información, ordenamiento, búsqueda, filtros, paginación, etc.

Otro posible tema de investigación que puede surgir a partir de la elaboración de este proyecto es **el desarrollo de un *software* similar a PSeInt, pero enfocado a la Web**. Como es de nuestro conocimiento, el Pseudolenguaje empleado por PSeInt utiliza diferentes palabras reservadas para la declaración de instrucciones; sin embargo, la ejecución de los Pseudocódigos resultantes se lleva a cabo a través de una consola o terminal similar a la que emplean la mayoría de los Sistema Operativos, lo que limita a futuros programadores interesados en el desarrollo Web a valerse de un solo mecanismo de entrada para la captura de información. En este sentido, gracias a la implementación de JavaScript en el lado del servidor (NodeJS), resulta cada vez más prometedor llevar a cabo una bifurcación del proyecto PSeInt, pero orientado al entrenamiento del desarrollo Web, puesto que en términos generales; JavaScript permite:

- ***Crear servidores Web.***
- ***Ejecutar un conjunto de instrucciones cuando se desencadena un cierto tipo de eventos, por ejemplo: al realizar un cambio en el contenido de un archivo.***
- Comunicarse de forma directa con el Modelo de Objetos de un Documento HTML.
- ***Realizar operaciones de lectura y escritura en archivos.***
- ***Comunicarse con servidores de Bases de Datos.***

Bajo este mismo orden de ideas, parte de las características mencionadas con anterioridad son implementadas con frecuencia en proyectos de uso masivo para el desarrollo Web profesional, entre los que destacan: los Preprocesadores de hojas de estilo CSS, los Motores de plantillas HTML, los Gestores de tareas como *Grunt* o *Gulp*, entre otros.

Finalmente, y no por ello menos importante. Otro tema a indagar a partir de la generación de este proyecto de investigación, consiste en **el desarrollo de videojuegos o animaciones mediante un Lenguaje de Programación Visual como es el caso de Scratch**. Puesto que, en la actualidad, la mayoría de los centros educativos buscan desarrollar habilidades mentales en sus estudiantes mediante el aprendizaje de la programación. De modo que los conocimientos adquiridos dentro del aula, ahora pasan a formar parte integral de proyectos relacionados con las ciencias, matemáticas, simulación o visualización de experimentos, historias relatadas de forma animada, arte interactivo, música, etc.

Fuentes de consulta

Cairó, Osvaldo. *Metodología de la Programación: Algoritmos, Diagramas de Flujo y Programas*. México: Alfaomega, 2005. Impreso.

Casale, Juan Carlos. *Introducción a La Programación*. Buenos Aires: Users, 2012. Impreso.

De la Cruz, Joel. *Algoritmos y Diagramas de Flujo aplicados a PHP*. Lima: Megabyte, 2006. Impreso.

Eguíluz, Javier. *Introducción a JavaScript*. 2009. *Librosweb*. Web. 17-08-2016. <URL>.

Gil, Francisco Javier. *Creación de sitios Web con PHP5*. Madrid: McGraw-Hill, Interamericana De España, 2006. Impreso.

López, Juan Carlos. *Algoritmos y Programación (Guía para Docentes)*. 2009. *Eduteka*. Web. 06-04-16. <URL>.

Novara, Pablo. *PSeInt. Documentación de Programa*. PSeInt. Versión. 20160219. Web. 07-05-16.

Pinales, Francisco Javier y César Eduardo Velázquez. *Problemario de Algoritmos resueltos con Diagramas de Flujo y Pseudocódigo*. México: Universidad Autónoma De Aguascalientes, 2014. Impreso.

Rodríguez, Miguel Angel. *Metodología de la Programación a través de Pseudocódigo*. Madrid: McGraw-Hill; Interamericana de España, 1991. Impreso.

Trejos, Omar Iván. *La Esencia de la Lógica de Programación*. Pereira: Papiro, 1999. Impreso.

Vaswani, Vikram. *Fundamentos De PHP*. México, D.F.: McGraw-Hill, Interamericana Editores S.A. de C.V., 2010. Impreso.