



Autor:
PATRICIO CELI V.

2023

Fundamentos de Programación

Basado en PSEint

Primera Edición

```
ae-file-history
ae-output
ae-rad
ae-ruby-debug
ae-search-in-files
ae-shell
ae-subprocess-inspector
ae-term
test-shutdown-after-startup
lib
tcl
test
README
arcadia.genspec
arcadia.todo
```

ITQ

WWW.ITQ.EDU.EC

INVESTIGACIÓN

ISBN: 978-9942-8921-5-7





FUNDAMENTOS DE PROGRAMACIÓN BASADOS EN PSEINT

AUTOR:

PATRICIO CELI V.

PRIMERA EDICIÓN

2023

TRABAJO EN EDICIÓN:



DIRECCIÓN EDITORIAL: SANTIAGO DEL CASTILLO G.

EDICIÓN EXTERNA: JONATHAN CHALCO S.

Este material está protegido por derechos de autor. Queda estrictamente prohibida la reproducción total o parcial de esta obra en cualquier medio sin la autorización escrita de los autores y el equipo editorial. El incumplimiento de esta prohibición puede conllevar sanciones establecidas en las leyes de Ecuador.

Todos los derechos están reservados.

ISBN: 978-9942-8921-5-7



DEDICATORIA

Dedico esta obra a Dios por ser el mi guía, camino y propósito de vida.

A mis Padres Walter y Patricia, por ser la razón de mi vida, quién con su sonrisa y presencia me motiva día a día a seguir creciendo en todo sentido, personal, profesionalmente y sobre todo espiritualmente.

Alejandra, hermana mi modelo a seguir, mi ejemplo de lucha, quien con su experiencia y don de personalidad me motiva a seguir siendo el primero en lo que me propongo, solo decirte gracias.

Bryan, hermano que con su temple y seriedad ayudan a formar carácter para seguir logrando metas.

A todas aquellas personas que confían en el trabajo realizado.

Patricio Celi V.

AGRADECIMIENTO

Agradezco a Dios y a la vida por permitirme cumplir mis sueños.

Patricio Celi V.

SOBRE EL AUTOR

Profesional en Análisis de Sistemas, estudiante de Ingeniería Informática, enloquecido por el Desarrollo de Software, amante de la Tecnología y soñador innato.

Mis competencias se referencian en perfiles profesionales como docente, consultor, lector, tutor y capacitador especializado en temas referentes al desarrollo de software.

Docente de Educación Superior con la firme convicción de que debemos poner nuestro grano de arena a las personas que comienzan en el mundo del Desarrollo, además enseñar es una forma de reforzar nuestros conocimientos.

Patricio Celi V.

CONTENIDO

DEDICATORIA.....	3
AGRADECIMIENTO.....	4
SOBRE EL AUTOR	5
INTRODUCCIÓN AL CONTENIDO DEL LIBRO	11
1. CAPÍTULO I.....	12
1.1. LA PROGRAMACIÓN	12
1.2. PROGRAMACIÓN ESTRUCTURADA	13
1.3. PROGRAMACIÓN MODULAR	14
1.4. PROGRAMACION ORIENTADA A OBJETOS	14
1.5. LENGUAJE DE PROGRAMACIÓN	15
1.6. TIPOS DE APLICATIVOS	16
1.6.1 APLICACIONES DE CONSOLA	16
1.6.2 APLICACIONES DE ESCRITORIO	17
1.6.3 APLICACIONES WEB	18
1.6.4. APP MOVILES	19
1.7. FEEDBACK	20
1.8. EVALUACIÓN	21
2. CAPÍTULO II.....	22
2.1. INTRODUCCIÓN A LA HERRAMIENTA PSEINT.....	22
2.1.1 CARACTERISTICAS Y FUNCIONALIDADES.....	22
2.1.2 ENTORNO DEL WORKFLOW.....	23
2.2 EDITORES	24
2.2.1 EDITOR DE DIAGRAMA DE FLUJO	24
2.2.2 EDITOR SEUDOCODIGO	25
2.3. EVALUACIÓN	27
3. ALGORITMOS Y PSEUDOCODIGO	28
3.1 QUE SON LOS ALGORITMOS	28
3.2 SEUDOCODIGO	29
3.2.1 ESTRUCTURAS DE SECUENCIACIÓN.....	30
DIAGRAMA DE FLUJO	31
PSEUDOCÓDIGO	31
.....	31
.....	31
.....	31
.....	31

.....	31
.....	31
3.2.1.1 EJEMPLO DE SECUENCIACIÓN MULTIPLICACIÓN	31
3.2.1.2 EJEMPLO DE SECUENCIACIÓN CONVERSIÓN	32
3.2.2 ESTRUCTURAS DE SELECCIÓN.....	32
3.2.2.1 EJEMPLO CON SELECCIÓN SI	33
3.2.2.2 EJEMPLO DE SI - OPERACIONES.....	34
3.2.2.3 EJEMPLO DE SEGÚN – DIA DE LA SEMANA	35
3.2.2.4 EJEMPLO DE SEGÚN Y SI	35
3.2.3 ESTRUCTURAS DE REPETICIÓN	36
3.2.3.1 EJEMPLO DE REPETIR.....	37
3.2.3.2 EJEMPLO DE MIENTRAS.....	38
3.2.3.3 EJEMPLO DE PARA	39
3.2.4 EVALUACIÓN EJERCICIOS PROPUESTOS	39
4. PROGRAMACIÓN MODULAR EN PSEUDOCÓDIGO	41
4.1 QUE SON SUBPROCESOS O FUNCIONES.....	41
4.2 EJEMPLOS DE MODULARIZACIÓN	42
4.2.1 EJEMPLO PERÍMETRO Y ÁREA	42
4.2.2 EJEMPLO IMPUESTO Y CÁLCULO	43
4.2.3 EJEMPLO LISTA NÚMEROS	44
4.2.3 EJEMPLO OPERACIONES MATEMÁTICA	45
4.3 VARIABLES LOCALES, GLOBALES y PARÁMETROS	46
4.3.1 VARIABLES GLOBALES	46
4.3.2 VARIABLES LOCALES	47
4.3.3 PARÁMETROS POR VALOR Y REFERENCIA.....	47
4.3.4 EJEMPLOS DE PASO DE PARÁMETROS	48
4.3.4.1 EJEMPLO DE PASO POR VALOR	48
4.3.4.2 EJEMPLO DE PASO POR REFERENCIA	48
4.4. EJERCICIOS PROPUESTOS DE SUBPROCESOS	49
5. ARREGLOS	50
5.1 ARREGLOS UNIDIMENSIONALES	50
5.1.1 EJERCICIOS CON ARREGLOS UNIDIMENSIONALES	52
5.1.1.1 EJERCICIO RELLENO DE ARREGLO	52
5.2 ARREGLOS MULTIDIMENSIONALES	56
5.2.1 EJERCICIOS CON ARREGLOS BIDIMENSIONALES	58

5.2.1.1 EJERCICIO CARGAR MATRIZ.....	58
5.2.1.2 EJERCICIO SUMA DE MATRICES.....	59
.....	60
5.2.1.3 EJERCICIO ESTUDIANTES.....	62
5.2.1.4 EJERCICIO PRODUCTOS	63
5.3.1 EJERCICIOS PROPUESTOS ARREGLOS Y MATRICES.....	65
6. RECURSIVIDAD	67
6.1 DEFINICIÓN	67
6.2 TIPOS DE RECURSIVIDAD	68
6.3 EJERCICIOS PROPUESTOS DE RECURSIVIDAD	71
7. ALGORITMOS A CÓDIGO FUENTE	72
8. BIBLIOGRAFÍA	76

ÍNDICE DE IMÁGENES

Figura 1: Programación	12
Figura 2: Programación Estructurada.....	13
Figura 4: Programación Orientada a Objetos.....	16
Figura 5: Ejemplo Aplicación de Consola	17
Figura 6: Ejemplo de Aplicación de Escritorio	18
Figura 7: Ejemplo Aplicaciones Web	19
Figura 8: Ejemplo de Aplicaciones Móviles	20
Figura 9: Pseint	22
Figura 10: Entorno de Pseint	23
Figura 11: Ejemplo de Editor de Diagrama de Flujo.....	24
Figura 12: Editor de Pseudocódigo.....	25
Figura 13: Paso a Paso de Aplicación	26
Figura 14: Ejemplo Primer Algoritmo.....	29
Figura 15: Ejemplo Secuenciación	31
Figura 16: Ejemplo de Conversión.....	32
Figura 17: Ejemplo SI	33
Figura 18: Ejemplo de SI - Operaciones.....	34
Figura 19: Ejemplo Segun Día de la Semana	35
Figura 20: Ejemplo Segun - Si	36
Figura 21: Ejemplo Repetir	38
Figura 22: Ejemplo Mientras	38
Figura 23: Ejemplo Para	39
Figura 24: Ejemplo Perímetro - Area.....	42
Figura 25: Ejemplo Impuesto - Cálculo.....	43
Figura 26: Ejemplo Lista Números.....	44
Figura 27: Ejemplo de Operaciones Matemáticas	45
Figura 29: Ejecución Ejercicio Operaciones.....	46
Figura 30: Ejemplo de Referencia por Valor	48

Figura 31: Ejemplo de Paso por Referencia	48
Figura 33: Verifica tus respuestas	49
Figura 34: Sintaxis Arreglo Unidimensional	51
Figura 35: Sintaxis Arreglo Bidimensional	51
Figura 36: Ejercicio Rellenar Arreglo	52
Figura 37: Ejecución Ejemplo Ordenar Lista de Menor a Mayor	52
Figura 38: Ordenar Lista de Menor a mayor	53
Figura 39: Ejemplo Desviación Estándar	54
Figura 40: Ejecución de Desviación Estándar	54
Figura 41: Ejemplo Ordenamiento Burbuja	55
Figura 42: Ejecución Ordenamiento Burbuja	56
Figura 43: Sintaxis de Arreglo Bidimensional	57
Figura 44: Ejemplo de Carga Matriz	58
Figura 45: Ejemplo Suma Matrices.....	60
Figura 46: Ejecución de Suma Matrices	61
Figura 47: Ejemplo Estudiantes	62
Figura 48: Ejecución Ejemplo Estudiantes.....	62
Figura 49: Ejemplo Ventas.....	63
Figura 50: Ejecución Total ventas.....	64
Figura 51: Verifica las resoluciones	66
Figura 52: Ejemplo Recursividad - Factorial	67
Figura 53: Ejemplo de Recursividad - Fibonacci	68
Figura 54: Ejemplo de Recursividad Directa.....	68
Figura 55: Ejemplo Recursividad Indirecta	69
Figura 56: Ejemplo de Recursividad Múltiple.....	70
Figura 57: Ejemplo de Recursividad Lineal.....	70
Figura 58: Verifica las resoluciones.	71
Figura 59: Exportación de Código Fuente	72
Figura 60: Ejemplo Suma de 2 Números	73
Figura 61: Ejemplo Código Html.....	73
Figura 62: Código Fuente JAVA	74
Figura 63: Ejemplo Código Fuente PHP.....	74
Figura 64: Ejemplo Código Fuente Javascript.....	75
Figura 65: Ejemplo Código Fuente Matlab.....	75

ÍNDICE DE TABLAS

Tabla 1: Estructuras de Secuenciación	31
Tabla 2: Estructuras de Selección.....	32
Tabla 3: Estructuras de Repetición.....	36
Tabla 4: Arreglo Unidimensional	50
Tabla 5: Arreglo Relleno	51
Tabla 6: Ejemplo Visual Arreglo Bidimensional.....	57

Tabla 7: Relleno Arreglo Bidimensional.....	58
---	----

INTRODUCCIÓN AL CONTENIDO DEL LIBRO

La programación basada en pseudocódigos es una técnica que se utiliza en la programación para diseñar algoritmos de una manera más sencilla y fácil de entender. PSeInt es un programa que nos permite crear pseudocódigos de una forma visual y fácil de utilizar; es una herramienta muy útil para principiantes en la formación del Desarrollo de Software, ya que les permite diseñar algoritmos de una forma sencilla y sin necesidad de tener conocimientos previos de programación.

Una de las ventajas de utilizar PSeInt es que permite la creación de pseudocódigos de una manera más estructurada, lo que facilita su comprensión y su posterior conversión a un lenguaje de programación. Además, PSeInt cuenta con una gran cantidad de funciones y estructuras de control, lo que permite al programador diseñar algoritmos complejos de una forma más sencilla.

Otra ventaja de utilizar PSeInt es que es un programa de código abierto y completamente gratuito, lo que lo hace accesible para cualquier persona que quiera iniciarse en el mundo de la programación. Además, cuenta con una gran comunidad de usuarios y desarrolladores que lo utilizan y lo mejoran continuamente.

A continuación encontraremos la información suficiente para que el estudiante de desarrollo de software pueda comenzar en el mundo de la programación de una manera fácil y divertida, con un software de muy intuitivo y cómodo de aprender como lo es Pseint, al final de cada capítulo encontraremos ejercicios de refuerzo mismo que el autor ya los ha realizado con anterioridad y se encuentran cargados en un repositorio para su revisión, es importante que el estudiante escanee el código QR de cada capítulo y encontrara la solución de los ejercicios propuestos.

Este es el punto de partida para sumergirse en el mundo de la programación y del desarrollo de software, tienes madera para el reto.

1. CAPÍTULO I

1.1. LA PROGRAMACIÓN

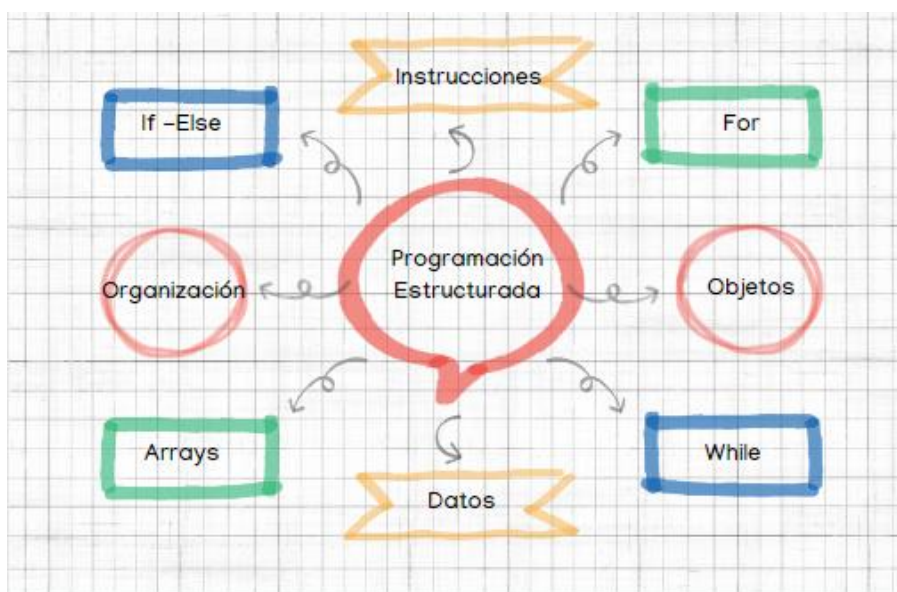
La programación expande la mente, ayuda a pensar en forma sistémica y ordenada, permitiendo que las personas mejoren y automaticen tareas que realizan en sus trabajos de la vida diaria.

La programación es el proceso de creación de programas de computadora. Esta explicación se puede explicar de la siguiente manera. Un programa no es más que decirle a la computadora qué, en qué formato y cómo llegar al usuario. En otras palabras, es un tipo de tecnología para traducir los deseos humanos a lenguaje de máquina, como por ejemplo cálculos de área de una figura geométrica, cálculo del impuesto a la renta de un sistema de facturación o calcular pagos de nómina en una empresa.

Programar, sea cual sea el lenguaje utilizado CSS, lenguaje C, C++, C#, PHP, Java, JavaScript, Python, entre otros, consiste en escribir en un lenguaje que entienda la máquina, es decir, cómo nos comunicamos con un ordenador y los algoritmos, estos van de la mano de la evolución tecnológica en el siglo XX y la creación de computadores de alto alcance en capacidad de almacenamiento, transacción y procesamiento.

Por lo tanto, saber programar es ser capaz de crear aplicaciones web y móviles, páginas web, interfaces-usuario, softwares de programación y pilotar y programar robots informatizados con la finalidad de automatizar procesos y con el paso del tiempo eliminar las tareas manuales dentro de un entorno empresarial.

Figura 1: Programación



Fuente: Autor

1.2. PROGRAMACIÓN ESTRUCTURADA

La programación estructurada tuvo su punto de partida en la década de 1960, donde los lenguajes de programación predominantes eran COBOL, PASCAL, C y FORTRAN 90, entre otros. (López Román, 2011)

Según (Pérez López, n.d.) en su artículo científico Programación Estructurada afirma que la programación estructurada es un paradigma de programación imperativa que se apoya en tres pilares fundamentales:

- **Estructuras básicas:** los programas se escriben usando sólo unos pocos componentes constructivos básicos que se combinan entre sí mediante composición.
- **Recursos abstractos:** los programas se escriben sin tener en cuenta inicialmente el ordenador que lo va a ejecutar ni las instrucciones de las que dispone el lenguaje de programación que se va a utilizar.
- **Diseño descendente por refinamiento sucesivo:** los programas se escriben de arriba abajo a través de una serie de niveles de abstracción de menor a mayor complejidad, pudiéndose verificar la corrección del programa en cada nivel.

La arquitectura de un programa consistía en datos y en un conjunto de módulos jerarquizados, detallado en la siguiente ilustración.

Figura 2: Programación Estructurada



Fuente: Autor

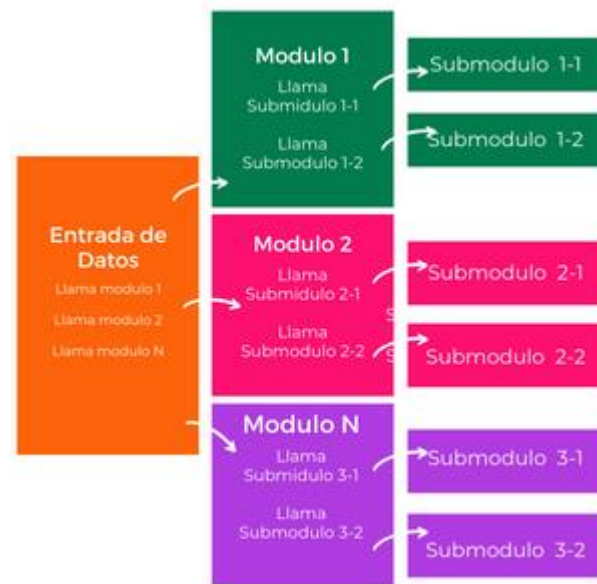
1.3. PROGRAMACIÓN MODULAR

La programación modular o mejor conocida como descendente o top-down, considera la programación como la subdivisión del aplicativo en pequeños modulo más simples, con la premisa de que el mantenimiento sea más fácil y comprensibles.

Una de las características más importantes de la programación modular es que abarca problemas de mayor complejidad de una forma más desglosada de tal forma que los módulos hagan tareas específicas, incluso que algunos de ellos dependan de otros para funcionar.

Además, se enfoca en el desarrollo de programas bien estructurados, documentados, con mucha facilidad de entender y dar mantenimiento.

Figura 3: Programación Modular



Fuente: Autor

1.4. PROGRAMACION ORIENTADA A OBJETOS

Según (Echéverría et al., 2004), la Programación Orientada a Objetos supone un cambio en la concepción del mundo de desarrollo de software, introduciendo un mayor nivel de abstracción que permite mejorar las características del código final.

Las principales características de este modelo de programación son los siguientes:

- Manejo de Clases y Objetos, esta se centra en el desarrollo de aplicaciones enfocando su trabajo en los seres mas no en las acciones.
- El concepto Clase se define como una entidad que encapsula los datos.
- Conceptos como Composición, Herencia, Polimorfismos
- Mediante conceptos como la composición, herencia y polimorfismo se consigue simplificar el desarrollo de sistemas. La composición y la herencia nos permiten construir clases a partir de otras clases, aumentando en gran medida la reutilización.

Cuando escribe un programa de computación en un lenguaje orientado a objetos está creando en su computadora un modelo de alguna parte del mundo real. Las partes con que se construye el modelo provienen de los objetos que aparecen en el dominio del problema. Estos objetos deben estar representados en el modelo computacional que se está creando.(Barnes & Kölling, n.d.)

Por ejemplo, en el caso que nuestro objetivo sea crear un modelo de sistema que nos permita registrar las calificaciones de los estudiantes de x institución, un tipo de entidad o clase con la que se debe trabajar es ESTUDIANTE, donde nos surgen ciertas incógnitas sobre la definición de la entidad Estudiante.

- ¿Qué nombre tiene el estudiante?
- ¿Qué apellido tiene el estudiante?
- ¿Qué nacionalidad tiene?
- ¿Cuál es su ubicación?

1.5. LENGUAJE DE PROGRAMACIÓN

Un lenguaje de programación es un conjunto de instrucciones y reglas que se utilizan para crear software y aplicaciones informáticas. Estos lenguajes son utilizados por los desarrolladores para comunicarse con las computadoras y realizar otras tareas de forma automática. Cada lenguaje de programación tiene su propia sintaxis y semántica que rigen cómo se escriben e interpretan los comandos.

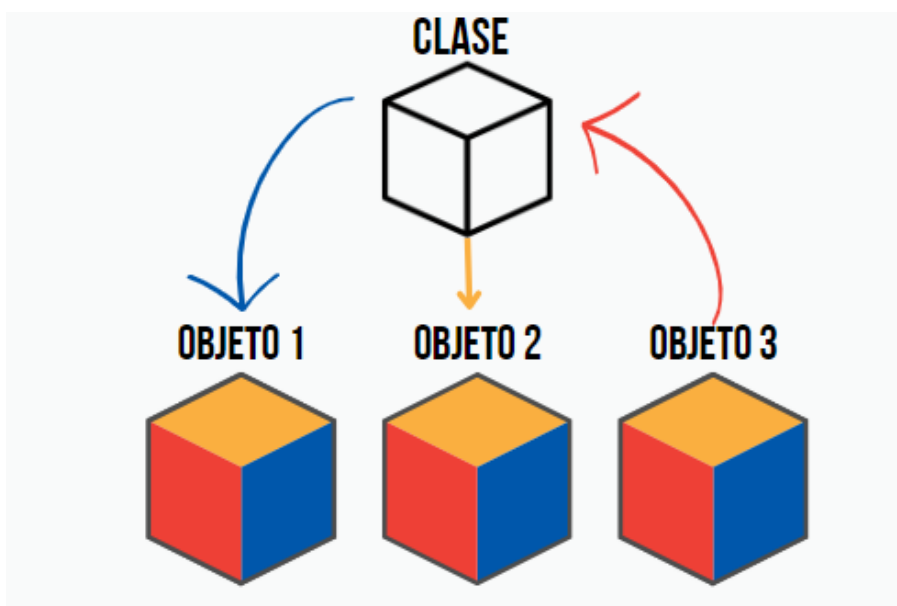
Los lenguajes de programación son herramientas importantes en el desarrollo de software porque permiten a los programadores convertir sus ideas en código de trabajo. Además, te permiten crear programas para diferentes sistemas operativos, dispositivos y plataformas. Existen diferentes tipos de lenguajes de programación, como lenguajes de bajo y alto nivel, que se utilizan para diferentes propósitos y en diferentes contextos.



Hay muchos lenguajes de programación disponibles en la actualidad, cada uno con sus propias características y beneficios. Algunos de los lenguajes de programación más populares son C++, Java, Python, Ruby y JavaScript.

Los desarrolladores deben elegir el lenguaje de programación adecuado para cada proyecto, teniendo en cuenta factores como el propósito del software, la plataforma de destino, la complejidad del proyecto y la experiencia del equipo de desarrollo.

Figura 3: Programación Orientada a Objetos



Fuente: Autor

1.6. TIPOS DE APLICATIVOS

1.6.1 APLICACIONES DE CONSOLA

Una aplicación de consola es un programa informático que se ejecuta en la línea de comandos de un sistema operativo. A diferencia de las aplicaciones con interfaz gráfica, las aplicaciones de consola no tienen una interfaz de usuario visual y se controlan exclusivamente mediante la entrada y salida de texto en la línea de comandos.

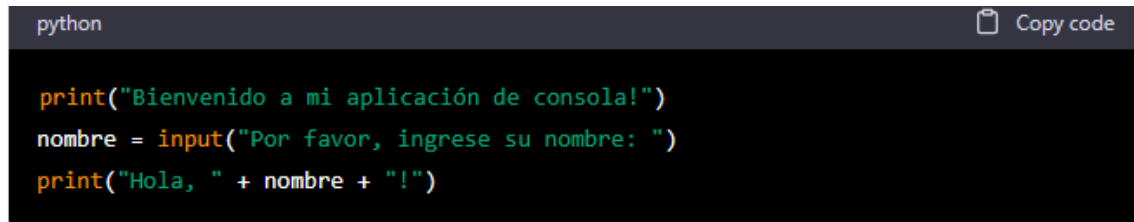
Estas aplicaciones son útiles para realizar tareas específicas de manera automatizada, como la gestión de archivos, la configuración de redes o la ejecución de scripts.

Las aplicaciones de consola se escriben utilizando lenguajes de programación como C++, Java o Python, y suelen ser utilizadas por programadores y administradores de sistemas. Aunque no son tan populares como las aplicaciones con interfaz gráfica, las aplicaciones

de consola son esenciales en muchos contextos, como la automatización de procesos, la gestión de servidores o la programación de tareas programadas.

A continuación, es un ejemplo de una aplicación de consola de Python que muestra un mensaje de bienvenida y solicita al usuario que ingrese su nombre de usuario:

Figura 4: Ejemplo Aplicación de Consola

A screenshot of a code editor window titled 'python'. The code inside is a simple Python script for a console application. It has three lines: a print statement with a welcome message, an input statement that prompts the user for their name, and another print statement that concatenates 'Hola, ' with the user's name and an exclamation mark. A 'Copy code' button is visible in the top right corner of the editor.

```
python

print("Bienvenido a mi aplicación de consola!")
nombre = input("Por favor, ingrese su nombre: ")
print("Hola, " + nombre + "!")
```

Fuente: Autor

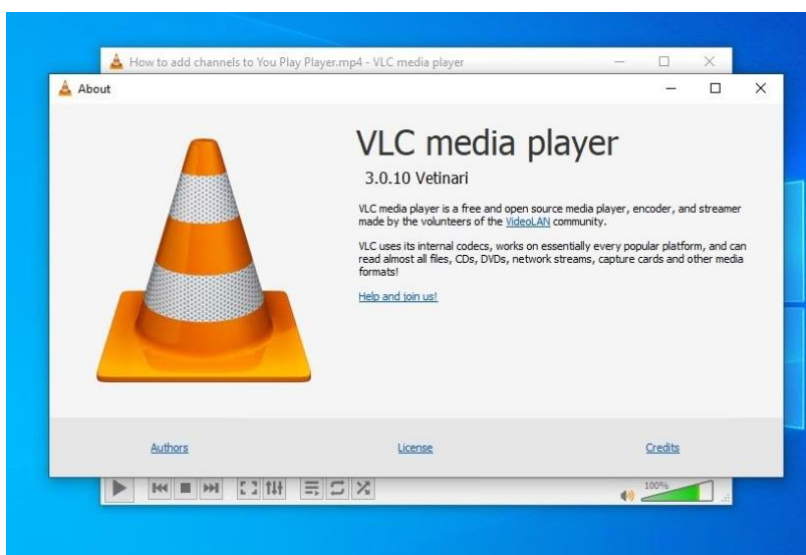
1.6.2 APLICACIONES DE ESCRITORIO

Las aplicaciones de escritorio son programas informáticos que se ejecutan en el equipo local del usuario y se utilizan para realizar tareas específicas. Según Sánchez y Gómez, "las aplicaciones de escritorio se ejecutan en la máquina del usuario, ofrecen un alto rendimiento y no requieren una conexión a Internet para su funcionamiento". (Sánchez & Gómez, 2018)

Estas aplicaciones pueden ser desarrolladas utilizando diferentes lenguajes de programación, como C#, Java o Python, y suelen tener una interfaz gráfica de usuario que permite la interacción con el programa a través de botones, menús y otras opciones visuales.

Un ejemplo de una aplicación de escritorio en C# es el reproductor multimedia VLC. Esta aplicación permite reproducir una amplia variedad de formatos de audio y video en el equipo local del usuario. Además, cuenta con diferentes características y herramientas que facilitan la reproducción y la gestión de archivos multimedia, como la creación de listas de reproducción y la edición de metadatos.

Figura 5: Ejemplo de Aplicación de Escritorio



Fuente: Autor

1.6.3 APLICACIONES WEB

Las aplicaciones web son programas informáticos que se ejecutan en un servidor web y se acceden a través de un navegador web en el equipo del usuario. Según la investigación de Yang et al., "las aplicaciones web son un tipo de aplicación distribuida que permite a los usuarios acceder y manipular información a través de Internet utilizando un navegador web". (Yang et al., 2019) Estas aplicaciones se desarrollan utilizando diferentes tecnologías web como HTML, CSS, JavaScript y lenguajes de programación de servidor como PHP, Java o C#.

Las aplicaciones web ofrecen una serie de ventajas en comparación con las aplicaciones de escritorio. Según el estudio de Reimers y Junglas, "las aplicaciones web permiten un fácil acceso desde cualquier lugar con conexión a Internet y ofrecen la capacidad de actualizar y mantener el software de manera más sencilla para el desarrollador". (Reimers & Junglas, 2016) Además, estas aplicaciones pueden ser accesibles desde cualquier dispositivo con conexión a Internet, lo que las hace especialmente útiles en entornos empresariales y educativos.

Un ejemplo de una aplicación web en C# es ASP.NET, un marco de trabajo de desarrollo web de Microsoft. Esta tecnología permite a los desarrolladores crear aplicaciones web dinámicas utilizando C# y una variedad de otras tecnologías web, como HTML, CSS y JavaScript. Con ASP.NET, los desarrolladores pueden crear aplicaciones web complejas, como sistemas de gestión de contenido, tiendas en línea y sistemas de seguimiento de clientes.

En resumen, las aplicaciones web son programas informáticos que se ejecutan en un servidor web y se acceden a través de un navegador web. Estas aplicaciones se desarrollan utilizando diferentes tecnologías web y ofrecen ventajas como el fácil acceso desde cualquier lugar con conexión a Internet y la capacidad de actualizar y mantener el software de manera más sencilla para el desarrollador.

Figura 6: Ejemplo Aplicaciones Web



Fuente: Autor

1.6.4. APP MOVILES

Las aplicaciones móviles son programas diseñados para ser utilizados en dispositivos móviles, como smartphones y tabletas. Estas aplicaciones se descargan e instalan en el dispositivo del usuario y pueden ofrecer una amplia variedad de funciones, como juegos, redes sociales, aplicaciones de productividad y herramientas de navegación. Según un estudio realizado por la Revista de Investigación de Marketing en el 2019, el uso de aplicaciones móviles ha aumentado significativamente en los últimos años, y se espera que siga creciendo en el futuro.(Escalas & Bettman, 2019)

Un ejemplo de aplicación móvil muy popular es WhatsApp, una aplicación de mensajería instantánea que permite a los usuarios enviar mensajes de texto, fotos, videos y realizar llamadas de voz y video a través de Internet.

Según un artículo publicado en la revista International Journal of Computer Science and Mobile Computing en el 2017, WhatsApp es una de las aplicaciones móviles más utilizadas en todo el mundo, con más de mil millones de descargas en Google Play Store.(Gupta & Jain, 2017) Además, esta aplicación ha sido clave en la transformación de la comunicación interpersonal, y ha permitido a los usuarios mantenerse conectados en todo momento y lugar.

Figura 7: Ejemplo de Aplicaciones Móviles

Fuente: Autor

1.7. FEEDBACK

En cuanto a la programación, es importante tener en cuenta que existen muchos lenguajes y frameworks que se pueden utilizar para desarrollar aplicaciones, y cada uno tiene sus ventajas y desventajas. Por lo tanto, es importante seleccionar el lenguaje y el framework adecuados según las necesidades y requerimientos específicos de cada proyecto.

Por otra parte, en relación con los tipos de aplicaciones, hay muchas categorías diferentes, incluyendo aplicaciones móviles, aplicaciones web, aplicaciones de escritorio, aplicaciones de realidad virtual, entre otras. Cada tipo de aplicación tiene sus propias características y requisitos de programación, por lo que es importante comprender las diferencias entre ellas para poder seleccionar el tipo de aplicación más adecuado para el proyecto en cuestión.

En general, en un recuento en cuanto a la programación y tipos de aplicaciones es importante seleccionar el lenguaje y framework adecuados y tener en cuenta las características y requisitos específicos de cada tipo de aplicación para poder desarrollar aplicaciones de calidad y eficientes.

Además, es importante mantenerse al tanto de las últimas tendencias y desarrollos en diseño y tecnología para brindar soluciones innovadoras y competitivas.

1.8. EVALUACIÓN

- Realice un cuadro comparativo de los lenguajes de programación más utilizados en el Siglo XXI.
- Realice una investigación sobre cuáles son los tipos de aplicaciones y lenguajes de programación más utilizados en las empresas de desarrollo en el Ecuador e indique sus impresiones sobre la misma, indicando el punto en el cual se encuentra nuestro país en cuanto a la innovación de software.

2. CAPÍTULO II

2.1. INTRODUCCIÓN A LA HERRAMIENTA PSEINT

Figura 8: Pseint



Fuente: Autor

PSeInt es una herramienta de programación estructurada que facilita la enseñanza de la lógica de programación a través de ejercicios interactivos. Con ella, los estudiantes pueden aprender conceptos fundamentales como la lógica y la sintaxis de manera visual y efectiva.

La revista electrónica Sistemas & Telemática, PSeInt se ha convertido en una herramienta fundamental para la enseñanza de la programación en universidades e institutos de educación secundaria en Latinoamérica.(Dávila & Amézquita, 2018)

Otro estudio, publicado en el libro "Herramientas informáticas para la enseñanza de la programación", afirma que PSeInt es un software ideal para que los docentes enseñen a sus estudiantes cómo resolver problemas a través de la programación. Además, permite a los estudiantes comprender mejor los conceptos de programación y estructurar sus ideas para crear programas más eficientes.(Torres & Buelvas, 2015)

PSeInt es una herramienta de programación estructurada que se ha convertido en una herramienta fundamental para la enseñanza de la programación en instituciones educativas; gracias a su capacidad para enseñar conceptos complejos de manera visual e interactiva, los estudiantes pueden aprender los fundamentos de la programación de manera efectiva.

2.1.1 CARACTERÍSTICAS Y FUNCIONALIDADES

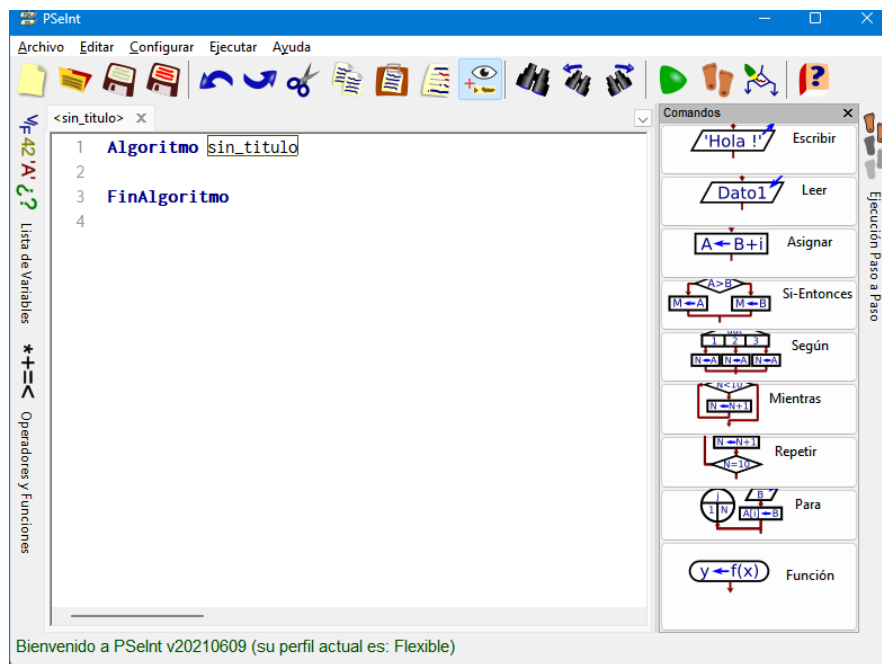
PSeInt es un lenguaje de programación educativo que permite aprender conceptos básicos de programación de manera sencilla y accesible.

A continuación, se detallan las diferentes funcionalidades y características que contiene PSeint para el aprendizaje de fundamentos de programación.

- Una de las funcionalidades de PSeInt es la capacidad de crear diagramas de flujo, que son representaciones visuales de un algoritmo en el que se utilizan símbolos y conectores para mostrar la secuencia de instrucciones.
- **Facilidad de uso:** PSeInt es muy fácil de usar, ya que su interfaz es intuitiva y sus comandos son simples y fáciles de entender.
- **Flexibilidad:** PSeInt permite crear diagramas de flujo de manera flexible, lo que significa que los usuarios pueden personalizar el diseño y los símbolos según sus necesidades.
- **Visualización:** La representación visual de los algoritmos hace que sea más fácil entender la secuencia de instrucciones y visualizar el flujo de datos.
- **Documentación:** PSeInt permite documentar el algoritmo en el mismo diagrama de flujo, lo que hace que sea más fácil para otros usuarios entender el proceso y hacer mejoras.
- **Portabilidad:** PSeInt es un programa multiplataforma, lo que significa que se puede ejecutar en diferentes sistemas operativos.

2.1.2 ENTORNO DEL WORKFLOW

Figura 9: Entorno de Pseint



Fuente: Autor

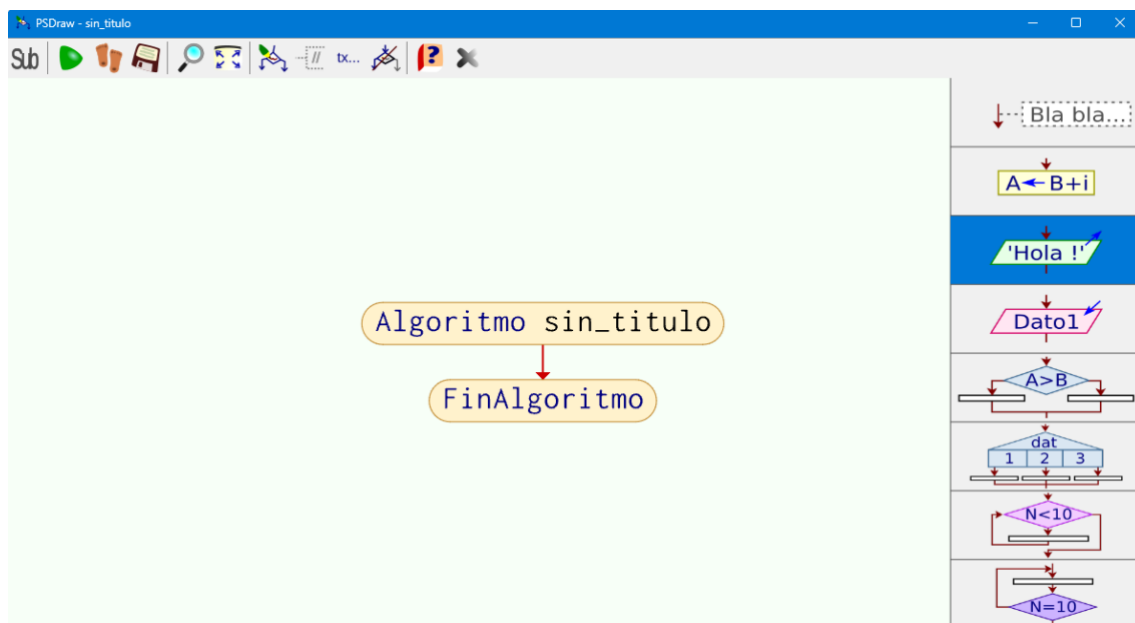
2.2 EDITORES

El ambiente de programación Pseint cuenta con varios editores para realizar diferentes actividades dentro del entorno:

2.2.1 EDITOR DE DIAGRAMA DE FLUJO

Para acceder al editor de Diagramas de Flujo se debe dirigir al Menú 'Archivo' y en la opción 'Editar Diagramas de Flujo' donde se desplegará la siguiente pantalla.

Figura 10: Ejemplo de Editor de Diagrama de Flujo

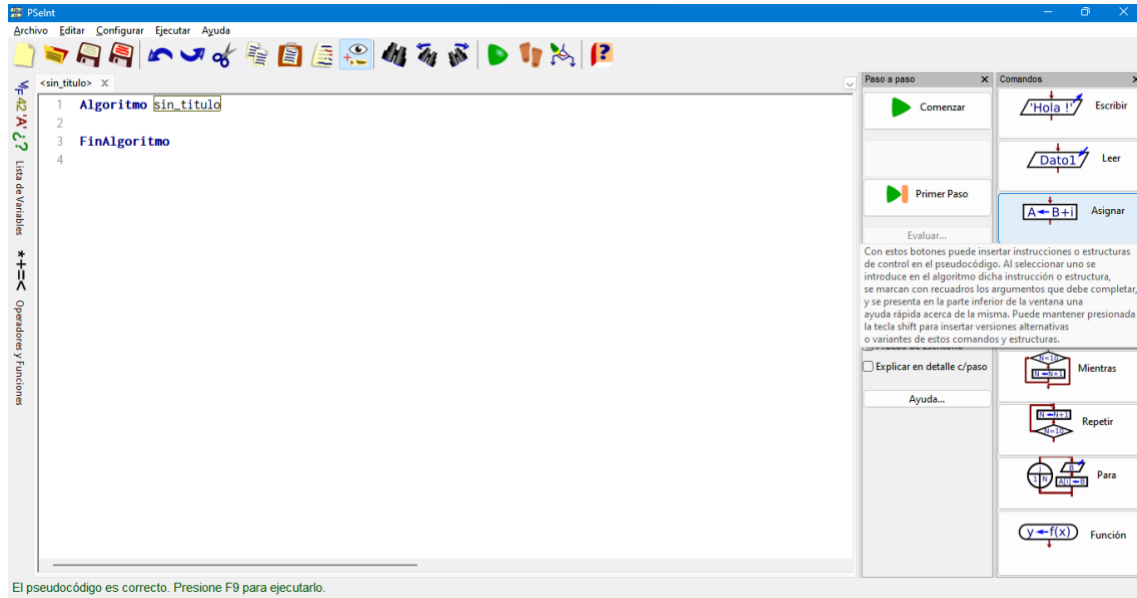


Fuente: Autor

Es importante mencionar que la última versión del aplicativo Pseint las herramientas para la gestión de los diagramas se encuentran ocultas y se muestran al realizar un paseo del puntero del ratón por la parte derecha de la ventana.

2.2.2 EDITOR SEUDOCODIGO

Figura 11: Editor de Pseudocódigo



Fuente: Autor

El editor de pseudocódigo aparece abierto por defecto al iniciar la aplicación, sus controles de paso a paso y comandos importantes se encuentran activos en la parte derecha de la ventana.

La opción paso a paso o prueba de escritorio es primordial para el nuevo desarrollador por que le da la posibilidad de que vaya interactuando línea por línea secuencial del código ingresado, haciendo que sea mucho más comprensible la sintaxis de los comandos, como las variables cambian a medida que avanza la ejecución, y la finalización de resultados al terminar la realización del código ingresado.

Figura 12: Paso a Paso de Aplicación

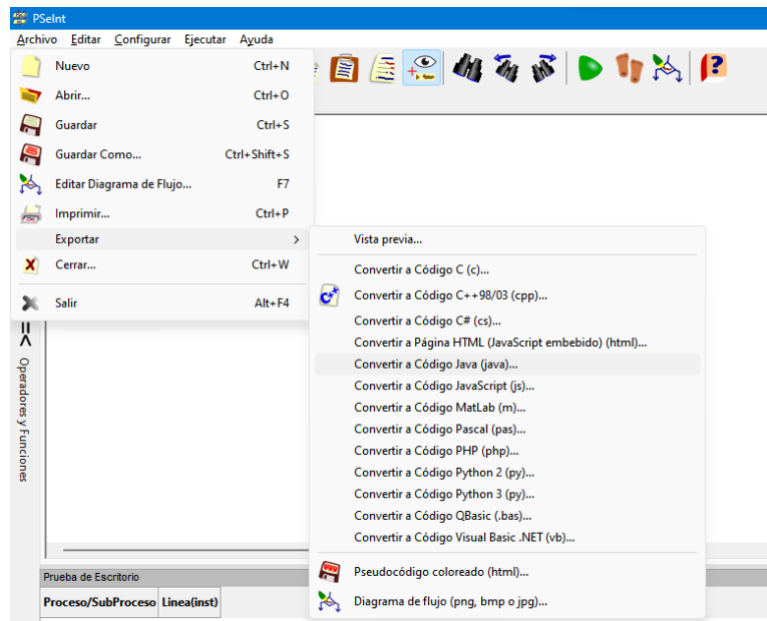


Fuente: Autor

Pseint como herramienta de desarrollo cuenta con un apartado para guardar el pseudocódigo realizado después de haberlo probado correctamente, para ello se debe seleccionar el menú 'Archivo' y la opción 'Guardar Como', escoger la ubicación donde se van a guardar los proyectos, asignarle un nombre representativo al ejercicio y dar clic en el botón 'Guardar', cabe mencionar que estos archivos se guardaran en el ordenador con la extensión '.psc' para reconocerlos fácilmente.

Una funcionalidad extra del aplicativo Pseint es que nos facilita la posibilidad de exportación del pseudocódigo a los más importantes lenguajes de programación que se manejan en la actualidad.

Figura 14: Ejemplo Creación de Proyecto



Fuente: Autor

2.3. EVALUACIÓN

- Explorar el entorno de trabajo de Pseint, realizar capturas de las herramientas que más le parecieron interesantes, guardar un proyecto en el computador, además de exportar el código en cualquiera de los lenguajes propuestos.

3. ALGORITMOS Y PSEUDOCODIGO

3.1 QUE SON LOS ALGORITMOS

Los algoritmos son secuencias de instrucciones lógicas y precisas que se utilizan en la informática para resolver problemas y tomar decisiones. Estas instrucciones se diseñan para procesar datos y realizar operaciones matemáticas o lógicas de manera eficiente. Los algoritmos se utilizan en una gran variedad de aplicaciones, como en el análisis de datos, la inteligencia artificial, la seguridad informática, la robótica, entre otras.

Según el libro "Introducción a la programación con Python" de Ana Isabel Mata Sánchez y Ana Belén del Pino García, los algoritmos son la base de la programación y la informática, y son esenciales para el desarrollo de programas y aplicaciones informáticas. (Mata Sánchez & del Pino García, 2017) En el libro "Introducción a los algoritmos y estructuras de datos" de Luis Joyanes Aguilar, se explica detalladamente la definición de algoritmos y su aplicación en la resolución de problemas en informática, además de presentar diferentes tipos de algoritmos y su eficiencia en términos de tiempo y espacio. (Joyanes Aguilar, 2015)

Los algoritmos tienen características muy marcadas para su desarrollo a continuación las más importantes:

- Simple, claro y preciso.
- Lógico.
- Secuencial y sistemático.
- Tener principio y fin.

Un algoritmo de ejemplo simple sería:

INICIO

Pídale al usuario que ingrese dos números enteros

Suma dos números y almacena el resultado en una variable

Mostrar resultados en pantalla

FIN

Este algoritmo requiere que el usuario ingrese dos números enteros, luego los sume y muestre el resultado en la pantalla. Este es un ejemplo muy básico y simple de un

algoritmo, pero sirve para ilustrar cómo se pueden diseñar secuencias lógicas de instrucciones para resolver un problema particular.

En la práctica, los algoritmos suelen ser mucho más complejos y pueden implicar muchos pasos y decisiones. Por ejemplo, el algoritmo para encontrar el camino más corto entre dos puntos en un mapa podría incluir el cálculo de distancias, rutas alternativas y obstáculos en el camino.

3.2 SEUDOCODIGO

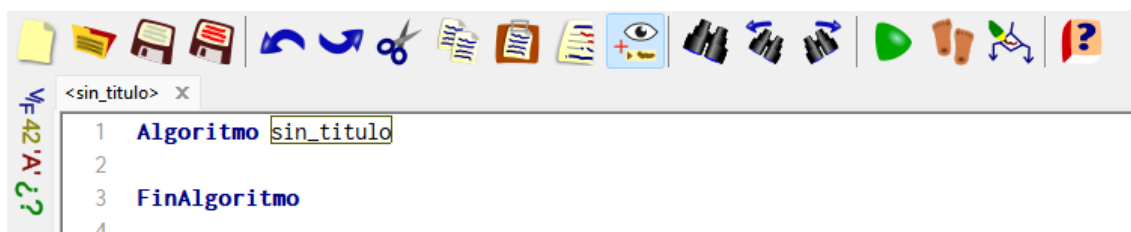
El pseudocódigo es una forma de representar algoritmos de manera informal, utilizando un lenguaje parecido al natural y sin preocuparse por detalles sintácticos específicos. El objetivo del pseudocódigo es permitir que los programadores expresen sus ideas y diseños de algoritmos de manera más clara y comprensible para ellos mismos y para otros programadores.

Según el libro "Fundamentos de programación" de Luis Joyanes Aguilar, el pseudocódigo es un lenguaje intermedio entre el lenguaje natural y el lenguaje de programación, que se utiliza para describir algoritmos en términos más cercanos a la realidad y menos abstractos que el lenguaje de programación propiamente dicho. (Joyanes Aguilar, 2015)

En el libro "Algoritmos y programación en Java" de Carlos Emilio Villalobos Cruz, se explica cómo utilizar el pseudocódigo para diseñar algoritmos paso a paso, y cómo estos algoritmos pueden luego ser traducidos a lenguajes de programación para su implementación en sistemas informáticos. (Villalobos Cruz, 2017)

A continuación, se muestra un ejemplo de algoritmo por defecto realizado por el aplicativo en Pseint.

Figura 13: Ejemplo Primer Algoritmo



Fuente: Autor

Es importante mencionar que las palabras en fuente negrita y de color azul se denominan palabras reservadas es decir sirven para dar algún tipo de procedimiento o acción al

algoritmo, además existen alineados a la izquierda los números de línea en color gris donde los mismos detallan el avance del algoritmo, así como las palabras en color negro y fuente normal que se denominan nombres de variables o proceso, estas si pueden ser modificadas de acuerdo con la preferencia del programador.

Por otra parte, existen un grupo de palabras delimitadas por ‘//’ donde se detallan los comentarios del algoritmo, estos pueden ser ingresados o no dentro del programa, estos ayudan a afianzar el conocimiento del proceso a realizar o dar una idea al momento de exportar la documentación del algoritmo.

3.2.1 ESTRUCTURAS DE SECUENCIACIÓN

La secuenciación en PSeInt se basa en el orden en que se escriben las instrucciones en el programa y en cómo se estructura el programa.

Por ejemplo, si se tiene un programa que solicita al usuario ingresar un número y luego muestra el doble de ese número, la secuencia de instrucciones sería la siguiente:

1. Mostrar mensaje "Ingrese un número:"
2. Leer número ingresado por el usuario y guardarlo en una variable
3. Calcular el doble del número ingresado
4. Mostrar el resultado del cálculo anterior

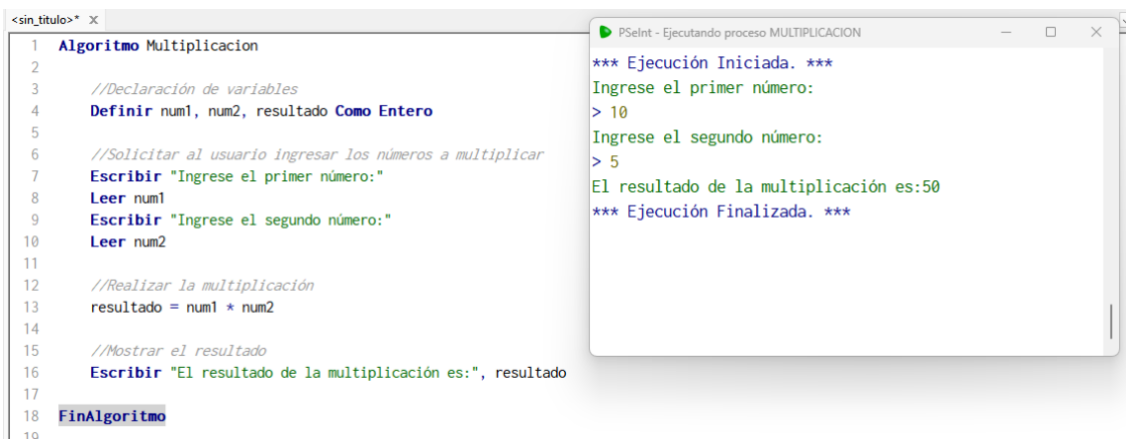
Es importante que las instrucciones se escriban en el orden correcto para que el programa funcione correctamente. Si se altera la secuencia de instrucciones, el programa puede producir resultados inesperados o errores.

Dentro de las palabras reservadas que indican secuenciación se encuentran las siguientes:

3.2.1.1 EJEMPLO DE SECUENCIACIÓN MULTIPLICACIÓN

Realizar un pseudocódigo que realice la multiplicación de dos números ingresados por teclado en Pseint.

Figura 14: Ejemplo Secuenciación



Fuente: Autor

En este ejemplo, las variables num1, num2 y resultado se declaran números.

Luego se le pide al usuario que ingrese dos números para ser multiplicados por la función Leer y almacenados en las variables apropiadas.

Luego se realiza la multiplicación usando el operador "*" y el resultado se almacena en la variable "resultado". Finalmente, el resultado se muestra en la pantalla mediante la función "Escribir".

Tabla 1: Estructuras de Secuenciación

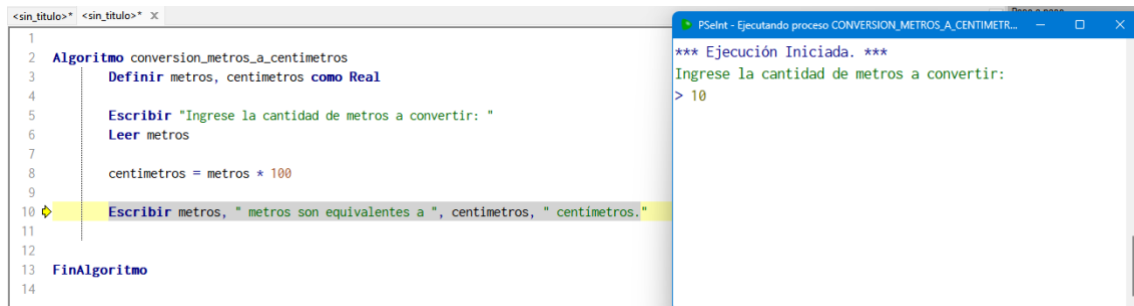
DIAGRAMA DE FLUJO	PSEUDOCÓDIGO
 Escribir	Escribir lista_de_expresiones
 Leer	Leer lista_de_variables
 Asignar	variable ← expresión

Fuente: Autor

3.2.1.2 EJEMPLO DE SECUENCIACIÓN CONVERSIÓN

Realizar un pseudocódigo que realice la conversión de metros a centímetros.

Figura 15: Ejemplo de Conversión



Fuente: Autor

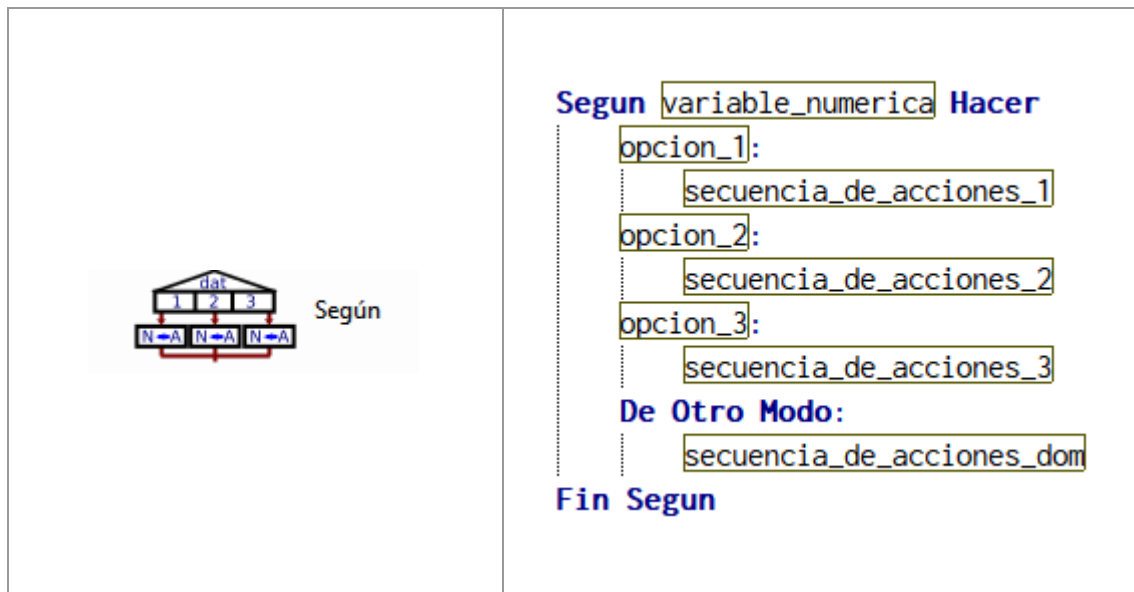
Este programa le solicita al usuario ingresar la cantidad de metros a convertir. Luego, utiliza la fórmula de conversión de metros a centímetros, que es multiplicar la cantidad de metros por 100. Finalmente, muestra el resultado de la conversión en centímetros al usuario.

3.2.2 ESTRUCTURAS DE SELECCIÓN

La selección es una estructura de control que permite ejecutar un conjunto de instrucciones solo si se cumple una condición determinada. En PSeInt, la selección se implementa mediante las estructuras "Si" y "Segun".

Tabla 2: Estructuras de Selección

DIAGRAMA DE FLUJO	PSEUDOCÓDIGO
<p>Si-Entonces</p>	<pre> Si expresion_logica Entonces acciones_por_verdadero SiNo acciones_por_falso Fin Si </pre>

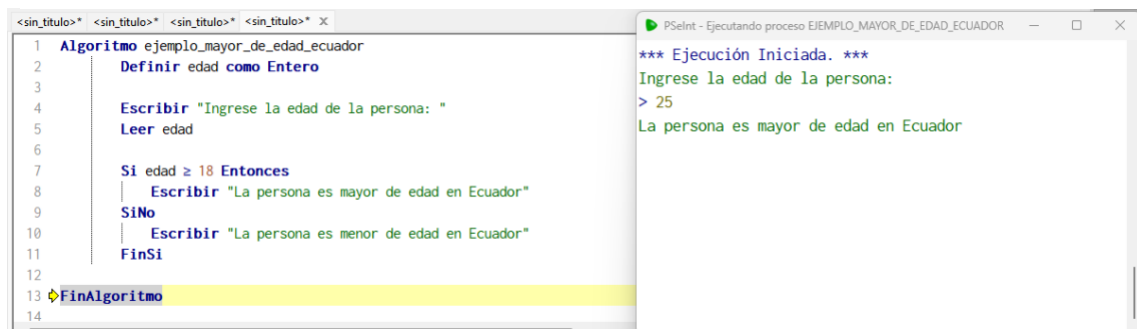


Fuente: Autor

3.2.2.1 EJEMPLO CON SELECCIÓN SI

Realizar un pseudocódigo en el cual se indique si una persona es mayor de edad en Ecuador, ingresando su edad y utilizando comandos de selección.

Figura 16: Ejemplo SI



Fuente: Autor

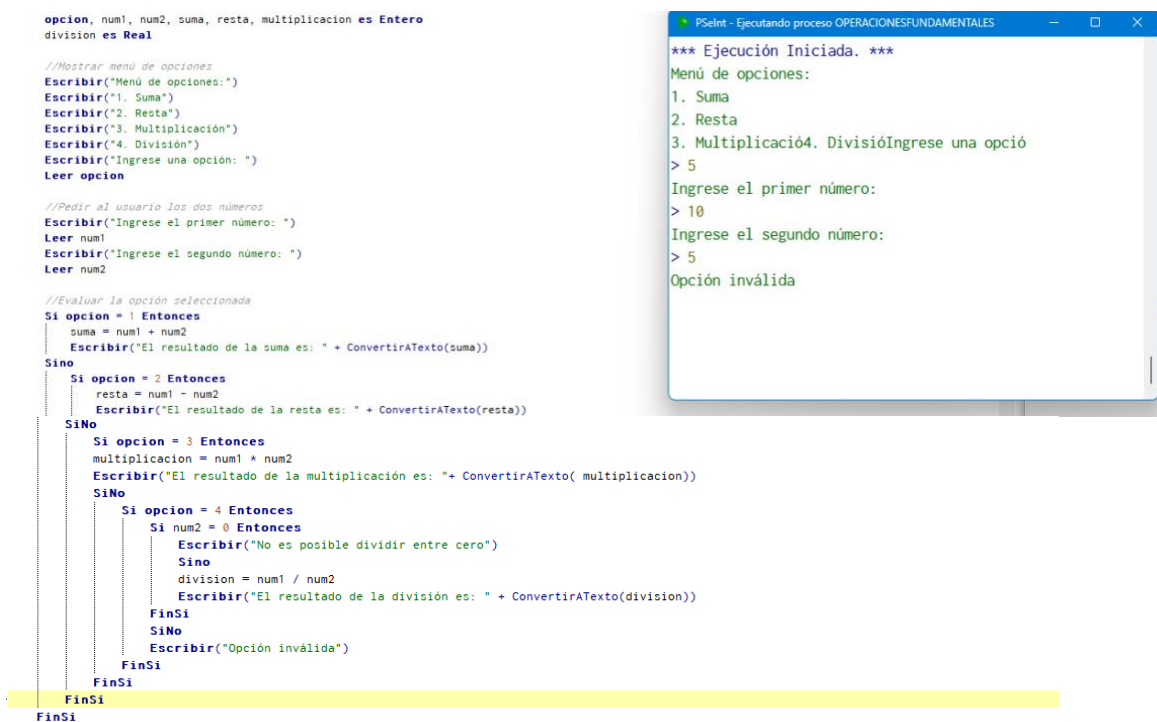
Este programa le solicita al usuario que ingrese la edad de una persona. Luego, utiliza una estructura "Si" para evaluar si la edad es mayor o igual a 18 años, que es la edad mínima para ser considerado mayor de edad en Ecuador. Si la edad es mayor o igual a 18 años, el programa muestra un mensaje indicando que la persona es mayor de edad en

Ecuador. Si la edad es menor a 18 años, el programa muestra un mensaje indicando que la persona es menor de edad en Ecuador.

3.2.2.2 EJEMPLO DE SI - OPERACIONES

Generar un menú donde tengamos las 4 operaciones fundamentales entre dos números, indicar que se debe realizar todo con si en PSeint.

Figura 17: Ejemplo de SI - Operaciones



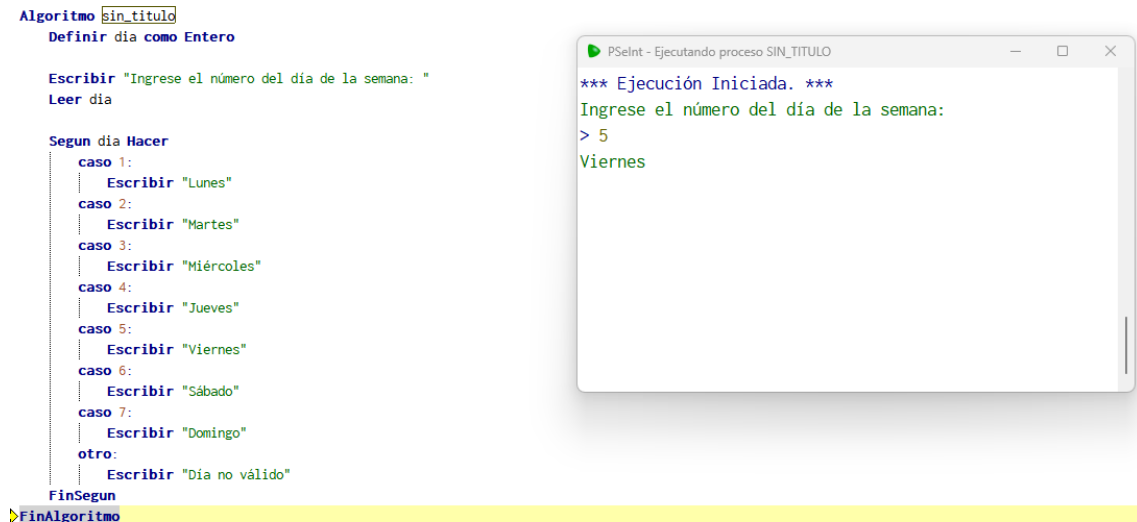
Fuente: Autor

Este programa muestra un menú de opciones para el usuario, donde puede seleccionar una de las cuatro operaciones fundamentales para realizar con dos números ingresados previamente. El programa utiliza la estructura condicional "Si...Entonces...Sino" para evaluar la opción seleccionada por el usuario y realizar la operación correspondiente. En el caso de la división, se incluye una validación para evitar la división entre cero.

3.2.2.3 EJEMPLO DE SEGÚN – DÍA DE LA SEMANA

Realizar un pseudocódigo en el cual se ingrese el número del día de la semana y devuelva el día en palabras si el valor no está dentro del rango indique día no valido, utilizar comandos de selección.

Figura 18: Ejemplo Segun Día de la Semana



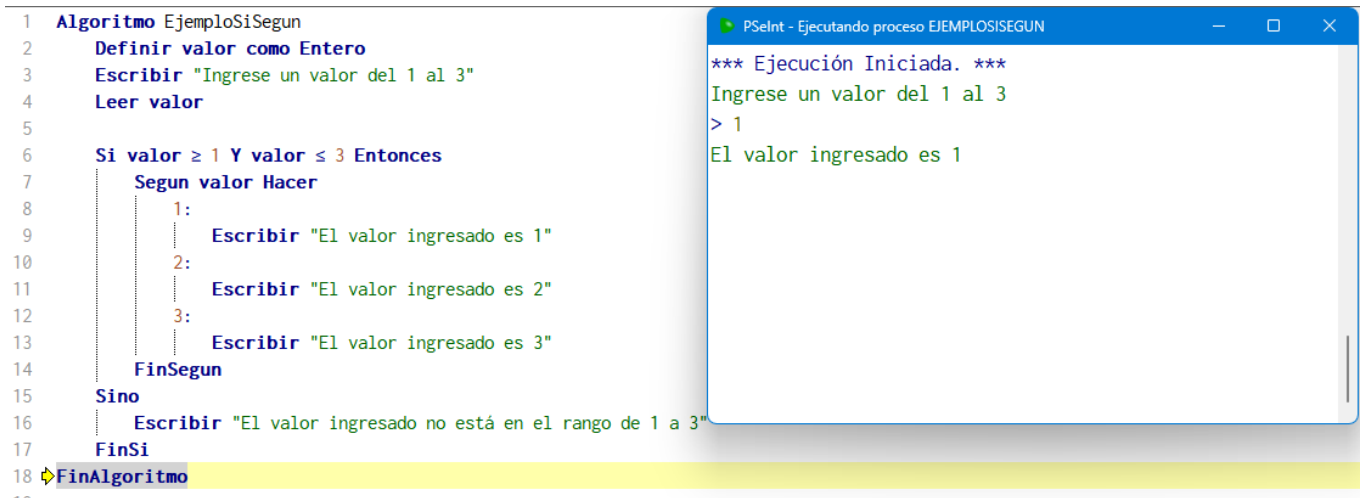
Fuente: Autor

Este programa le solicita al usuario que ingrese el número del día de la semana. Luego, utilice la estructura "Segun" para evaluar el valor ingresado y mostrar el nombre del día correspondiente. Si el valor ingresado no corresponde a un día válido (es decir, está fuera del rango de 1 a 7), el programa muestra un mensaje que indica que el día no es válido.

3.2.2.4 EJEMPLO DE SEGÚN Y SI

Realizar un pseudocódigo en el cual se ingrese por teclado un número del 1 al 3, y regresar el valor ingresado, controlando que el número no sea mayor a 3 ni menor a 0, utilizar comandos de selección.

Figura 19: Ejemplo Segun - Si



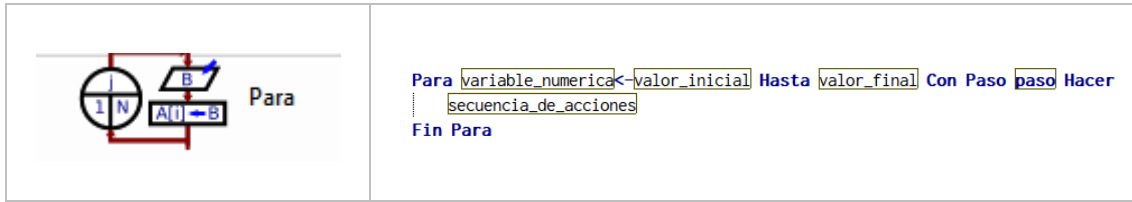
Fuente: Autor

En este ejemplo, primero le pedimos al usuario que ingrese un valor del 1 al 3. Luego, utilizamos el comando "si" para verificar si el valor ingresado está dentro del rango permitido. Si es así, utilizamos el comando "según" para mostrar un mensaje diferente dependiendo del valor ingresado. Si el valor no está dentro del rango permitido, utilizamos el comando "sino" para mostrar un mensaje indicando que el valor no es válido.

3.2.3 ESTRUCTURAS DE REPETICIÓN

Tabla 3: Estructuras de Repetición

DIAGRAMA DE FLUJO	PSEUDOCÓDIGO
<p>Mientras</p>	<pre> Mientras expresion_logica Hacer secuencia_de_acciones Fin Mientras </pre>
<p>Repetir</p>	<pre> Repetir secuencia_de_acciones Hasta Que expresion_logica </pre>



Fuente: Autor

Los comandos de repetición en PSeInt son herramientas que permiten automatizar la ejecución de una serie de instrucciones varias veces. Estos comandos son muy útiles en programación, ya que permiten simplificar el código y evitar la repetición de código innecesario.

En PSeInt, existen tres comandos de repetición principales: el comando "mientras", el comando "Para" y el comando "repetir".

El comando "mientras" permite repetir una serie de instrucciones mientras una determinada condición sea verdadera.

El comando "Para" es utilizado para crear una estructura de repetición con un contador que va incrementando de forma automática en cada iteración del ciclo. Este comando es especialmente útil cuando se necesita repetir una serie de instrucciones un número determinado de veces.

Por otro lado, el comando "repetir" permite repetir una serie de instrucciones al menos una vez, y seguir repitiéndolas mientras la condición sea verdadera.

Para utilizar los comandos de repetición en PSeInt, es importante conocer la sintaxis y las estructuras de control de flujo de este lenguaje de programación. Es importante tener en cuenta que un uso incorrecto de los comandos de repetición puede resultar en bucles infinitos o errores en la ejecución del programa. Por lo tanto, es necesario planificar cuidadosamente el uso de los comandos de repetición en cada programa y asegurarse de que se estén utilizando correctamente.

3.2.3.1 EJEMPLO DE REPETIR

Realizar un pseudocódigo en el cual genere un número aleatorio de 4 cifras valide si es mayor a 1000 sino seguir imprimiendo hasta encontrar uno menor a 1000.

Figura 20: Ejemplo Repetir

```

1 Algoritmo numero_aleatorio
2   Definir num Como Entero
3   Repetir
4     num ← Aleatorio(1000, 9999)
5     Escribir "Número aleatorio: ", num
6   Hasta Que num ≥ 1000
7   Escribir "Fin del programa"
8 FinAlgoritmo

```

```

*** Ejecución Iniciada. ***
Número aleatorio: 7109
Fin del programa
*** Ejecución Finalizada. ***

```

Fuente: Autor

Este programa genera un número aleatorio de cuatro cifras utilizando la función "Aleatorio" de PSeInt, y lo guarda en la variable "num". Luego, entra en un ciclo "repetir-hasta que", donde se imprime el número aleatorio y se genera uno nuevo mientras el número sea menor que 1000. Una vez que se genera un número mayor o igual a 1000, el programa termina y se imprime "Fin del programa".

3.2.3.2 EJEMPLO DE MIENTRAS

Realizar un pseudocódigo donde se solicite Algoritmo que suma n números positivos y cuenta los negativos, en caso de querer terminar el bucle ingresar 0.

Figura 21: Ejemplo Mientras

```

1 Algoritmo Algoritmodetarea
2   n=1
3   Mientras n ≠ 0 hacer
4     Escribir "Dame un numero"
5     leer n
6     si n < 0 entonces
7       negativo = negativo + 1
8     SiNo
9       positivo = positivo + n
10    FinSi
11  FinMientras
12  Escribir "El total de negativos es: ",negativo
13  Escribir "La suma de positivos es: ",positivo
14 FinAlgoritmo

```

```

*** Ejecución Iniciada. ***
Dame un numero
> -10
Dame un numero
> -3
Dame un numero
> 5
Dame un numero
> 19
Dame un numero
> 0
El total de negativos es: 2
La suma de positivos es: 24

```

Fuente: Autor

En este ejemplo, el programa utiliza la estructura "Repetir" para solicitar al usuario que ingrese números y realizar la suma de los números positivos y el conteo de los números negativos. El bucle se termina cuando se ingresa un 0.

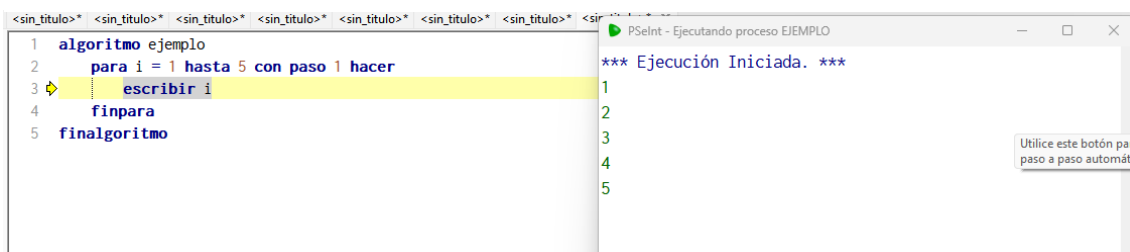
Cada número ingresado se evalúa mediante la estructura "si" para determinar si es positivo, negativo o cero. Si es positivo, se suma al acumulador "positivos=". Si es negativo, se incrementa el contador "negativo". Finalmente, se imprimen los resultados.

Este es un ejemplo de cómo utilizar la estructura "repetir" en PSeInt para realizar una tarea que solicita al usuario que ingrese números y realiza operaciones sobre ellos.

3.2.3.3 EJEMPLO DE PARA

Realizar un pseudocódigo donde se utiliza el comando "PARA" para imprimir los números del 1 al 5:

Figura 22: Ejemplo Para



Fuente: Autor

En este ejemplo, la variable "i" se inicializa con el valor 1, se incrementa en 1 en cada iteración y se detiene cuando alcanza el valor 5. En cada iteración, se escribe en la pantalla el valor de "i", que irá desde 1 hasta 5.

3.2.4 EVALUACIÓN EJERCICIOS PROPUESTOS

- Calcular la potencia dados dos números ingresados por teclado, base y exponente, realizar el cálculo con multiplicaciones sucesivas.
- Leer por teclado los 3 vértices de un triángulo y determinar si es un triángulo escaleno isósceles, escaleno o equilátero, además desplegar su área.
- Realizar un algoritmo el cual nos permita convertir un numero en hexadecimal.
- Contar el número de veces que una palabra aparece dentro de un párrafo ingresado por teclado, controlar que el párrafo tenga más de 100 palabras.

- Convertir un numero binario a decimal



4. PROGRAMACIÓN MODULAR EN PSEUDOCÓDIGO

4.1 QUE SON SUBPROCESOS O FUNCIONES

Un proceso es un conjunto de acciones o instrucciones que se ejecutan de forma secuencial para llevar a cabo una tarea específica. Estos procesos se pueden componer de subprocesos, que son bloques de código independientes que realizan tareas específicas y que se pueden llamar desde diferentes partes del programa. Los subprocesos permiten modularizar el código y hacerlo más fácil de entender y mantener.

Los subprocesos son una técnica de programación que permite dividir un programa en partes más pequeñas y manejables. Cada subproceso es un bloque de código que se puede llamar desde cualquier parte del programa para realizar una tarea específica. Los subprocesos son muy útiles para modularizar el código y mejorar su legibilidad, mantenimiento y reutilización. Además, permiten descomponer algoritmos complejos en tareas más pequeñas y manejables.

En el libro "Programación en PSeInt: Algoritmos y estructuras de datos" de R. Domínguez y L. Méndez se profundiza en el uso de subprocesos y se destaca su importancia en la programación estructurada y modular. (Domínguez & Méndez, 2015)

Además, se menciona que los subprocesos mejoran la reutilización del código, ya que se pueden llamar múltiples veces desde diferentes partes del programa. En otro libro llamado "Programación estructurada en Pascal y PSeInt" de J. L. Sierra y J. A. Recio, se describe cómo los subprocesos son una técnica importante para modularizar el código y reducir su complejidad. (Sierra & Recio, n.d.)

Sin embargo, es importante mencionar que se debe invocar o llamar a los subprocesos desde un proceso principal o un subproceso antecesor, dentro de las características más importantes los subprocesos se centran en tareas específicas y repetitivas, o cuando las tareas son muy extensas y se requiere realizar una resolución por partes.

La modularización o trabajo de desarrollo con procesos y subprocesos presenta varias ventajas. Algunas de las ventajas de la modularización en PSeInt son:

Reutilización de código: La modularización permite dividir el código en módulos o subrutinas que pueden ser utilizados en diferentes partes del programa. De esta manera, se puede evitar la repetición de código y ahorrar tiempo en la creación de nuevos programas.

Facilidad de mantenimiento: Al dividir el código en módulos o subrutinas, es más fácil de mantener y solucionar problemas. Si se encuentra un error en una parte del código, sólo se tiene que modificar esa parte y no todo el programa.



Claridad en el código: Al dividir el código en módulos o subrutinas, se puede lograr una mayor claridad y organización en el código, lo que facilita su comprensión y lectura.

Mejora en la eficiencia del programa: Si se divide el código en módulos o subrutinas, el programa puede ser más eficiente, ya que se pueden optimizar cada uno de ellos y reducir el tiempo de ejecución del programa.

Mejora en la calidad del programa: La modularización permite realizar pruebas y depuración de forma más sencilla, lo que mejora la calidad del programa en términos de fiabilidad y robustez.

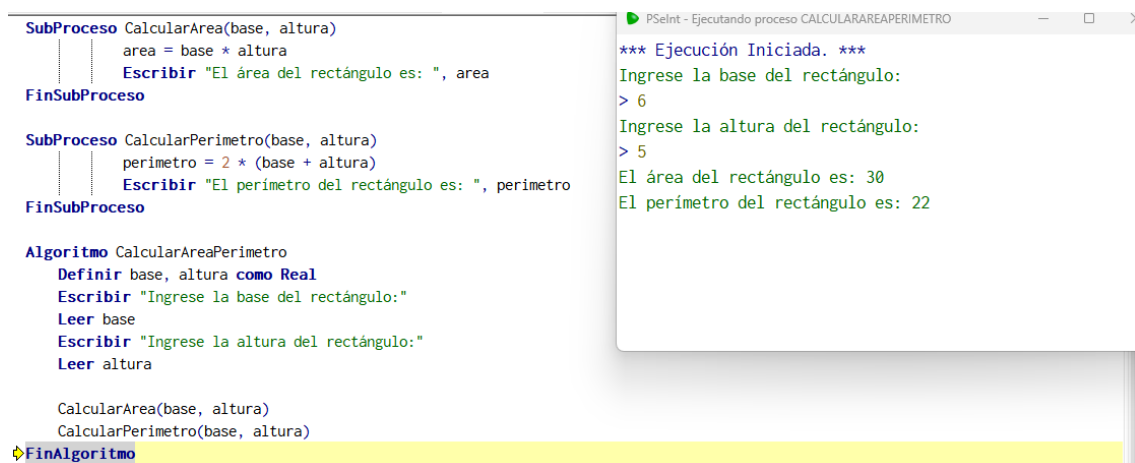
En conclusión, los subprocesos en PSeInt son una técnica fundamental de programación que permite modularizar y simplificar el código. Están ampliamente documentados en la literatura especializada, y se consideran una buena práctica de programación.

4.2 EJEMPLOS DE MODULARIZACIÓN

4.2.1 EJEMPLO PERÍMETRO Y ÁREA

Calcular el área y el perímetro de un rectángulo; en lugar de escribir todo el código en una sola sección, podemos dividirlo en dos módulos: uno para calcular el área y otro para calcular el perímetro.

Figura 23: Ejemplo Perímetro - Area



Fuente: Autor

En este ejemplo, la función **CalcularArea** y **CalcularPerimetro** son los módulos que calculan el área y el perímetro, respectivamente. En el algoritmo principal, simplemente leemos los valores de la base y la altura del rectángulo, y luego llamamos a los dos

módulos para que realicen los cálculos necesarios. Este enfoque hace que el código sea más modular, fácil de leer y mantener en caso de que necesitemos hacer cambios en el futuro.

4.2.2 EJEMPLO IMPUESTO Y CÁLCULO

Calcular el precio final de un producto incluyendo el impuesto y un descuento opcional. Podemos dividir el código en tres módulos: uno para calcular el impuesto, otro para calcular el descuento y otro para calcular el precio final.

Figura 24: Ejemplo Impuesto - Cálculo

```

SubProceso res ← CalcularImpuesto(precio, tasa)
    impuesto = precio * tasa
    res ← impuesto
FinSubProceso

SubProceso res ← CalcularDescuento(precio, porcentaje)
    descuento = precio * porcentaje / 100
    res ← descuento
FinSubProceso

SubProceso CalcularPrecioFinal(precio, tasa, porcentaje)
    impuesto = CalcularImpuesto(precio, tasa)
    descuento = CalcularDescuento(precio, porcentaje)
    precio_final = precio + impuesto - descuento
    Escribir "El precio final es: ", precio_final
FinSubProceso

Algoritmo CalcularPrecioFinal
    Definir precio, tasa, porcentaje como Real
    tasa = 0.12
    Escribir "Ingrese el precio del producto:"
    Leer precio

    Escribir "Ingrese el porcentaje de descuento (deje en 0 si no hay descu
    Leer porcentaje

    CalcularPrecioFinal(precio, tasa, porcentaje)
FinAlgoritmo
  
```

*** Ejecución Iniciada. ***

Ingrese el precio del producto:

> 20

Ingrese el porcentaje de descuento (deje en 0 si no hay d

escuento):

> 0

El precio final es: 22.4

Fuente: Autor

En este ejemplo, la función **CalcularImpuesto** y **CalcularDescuento** son los módulos que calculan el impuesto y el descuento, respectivamente.

La función **CalcularPrecioFinal** es el módulo que llama a los otros dos módulos para realizar los cálculos necesarios y luego calcular el precio final.

En el algoritmo principal, simplemente leemos el precio del producto, la tasa de impuesto y el porcentaje de descuento (si lo hay), y luego llamamos a la función **CalcularPrecioFinal** para obtener el precio final del producto.

4.2.3 EJEMPLO LISTA NÚMEROS

Crear un programa que le permita al usuario ingresar una lista de números y luego encontrar el número mayor en esa lista. Podemos dividir el código en dos módulos:

- **Primero:** para leer la lista de números y
- **Segundo:** Encontrar el número mayor en esa lista.

Figura 25: Ejemplo Lista Números

```

SubProceso res ← LeerListaNumeros(cantidad, listaNumeros)

    Para i = 1 Hasta cantidad Con Paso 1 Hacer
        Escribir "Ingrese el número ", i, ":"
        Leer listaNumeros[i]
    FinPara
    res ← listaNumeros
FinSubProceso

SubProceso EncontrarMayor(listaNumeros, cantidad)
    mayor = listaNumeros[1]
    Para i = 2 Hasta cantidad Con Paso 1 Hacer
        Si listaNumeros[i] > mayor Entonces
            mayor = listaNumeros[i]
        FinSi
    FinPara
    Escribir "El número mayor es: ", mayor
FinSubProceso

Algoritmo EncontrarMayorNumero
    Definir cantidad como Entero
    Escribir "Ingrese la cantidad de números en la lista:"
    Leer cantidad
    Definir listaNumeros como entero;
    Dimension listaNumeros[cantidad];
    listaNumeros = LeerListaNumeros(cantidad, listaNumeros[cantidad])
    EncontrarMayor(listaNumeros, cantidad)
FinAlgoritmo
  
```

Fuente: Autor

En este ejemplo, la función **LeerListaNumeros** es el módulo que lee la lista de números ingresados por el usuario y los almacena en una lista, mientras que la función **EncontrarMayor** es el módulo que encuentra el número mayor en esa lista.

En el algoritmo principal, simplemente leemos la cantidad de números en la lista, luego llamamos a la función **LeerListaNumeros** para leer la lista de números y almacenarla en una lista, y finalmente llamamos a la función **EncontrarMayor** para encontrar el número mayor en esa lista.

4.2.3 EJEMPLO OPERACIONES MATEMÁTICA

Crear un programa que le permita al usuario ingresar dos números y luego realizar varias operaciones matemáticas con ellos, como sumarlos, restarlos, multiplicarlos y dividirlos. Podemos dividir el código en cuatro módulos: uno para leer los números, otro para sumarlos, otro para restarlos, otro para multiplicarlos y otro para dividirlos.

Luego, podemos llamar a cada uno de estos módulos en secuencia para realizar las operaciones matemáticas deseadas.

Figura 26: Ejemplo de Operaciones Matemáticas

```

3  SubProceso res←Sumar(num1, num2)
4      resultado = num1 + num2
5      res← resultado
6  FinSubProceso
7
8  SubProceso res←Restar(num1, num2)
9      resultado = num1 - num2
10     res←resultado
11 FinSubProceso
12
13 SubProceso res←Multiplicar(num1, num2)
14     resultado = num1 * num2
15     res←resultado
16 FinSubProceso
17
18 SubProceso res←Dividir(num1, num2)
19     resultado = num1 / num2
20     res←resultado
21 FinSubProceso
22
23 Algoritmo
24     num1,
25     Escri
26     Leer num1
27     Escribir "Ingrese el segundo número:"
28     Leer num2
29     resultadoSuma = Sumar(num1, num2)
30     resultadoResta = Restar(num1, num2)
31     resultadoMultiplicacion = Multiplicar(num1, num2)
32     resultadoDivision = Dividir(num1, num2)
33     Escribir "El resultado de la suma es:", resultadoSuma
34     Escribir "El resultado de la resta es:", resultadoResta
35     Escribir "El resultado de la multiplicación es:", resultadoMultiplicacion
36     Escribir "El resultado de la división es:", resultadoDivision
37 FinAlgoritmo

```

Fuente: Autor

Figura 27: Ejecución Ejercicio Operaciones

```
*** Ejecución Iniciada. ***  
Ingrese el primer número:  
> 5  
Ingrese el segundo número:  
> 6  
El resultado de la suma es:11  
El resultado de la resta es:-1  
El resultado de la multiplicación es:30  
El resultado de la división es:0.8333333333
```

Fuente: Autor

En este ejemplo, las funciones Sumar, Restar, Multiplicar y Dividir son módulos que realizan cada una de las operaciones matemáticas en función de los dos números que se les pasan como argumentos y devuelven el resultado correspondiente.

En el algoritmo principal, llamamos a cada uno de estos módulos en secuencia para realizar las operaciones matemáticas deseadas y luego imprimimos los resultados correspondientes.

4.3 VARIABLES LOCALES, GLOBALES y PARÁMETROS

4.3.1 VARIABLES GLOBALES

Las variables globales son aquellas que se declara fuera de cualquier procedimiento o función y puede ser accedida desde cualquier parte del programa. Esto significa que su alcance es global y no está limitado por el contexto de un procedimiento o función específica.

Las variables globales son útiles cuando se requiere que varias partes del programa accedan y modifiquen la misma información, ya que permiten que dicha información se comparta fácilmente entre diferentes partes del código.

Es importante mencionar que el uso excesivo de variables globales puede hacer que el programa sea difícil de entender, mantener y depurar, la accesibilidad a estas variables exhorta que puedan ser modificadas desde cualquier parte del código o ser modificadas por diferentes subprocesos al mismo tiempo esto provoca comportamientos inesperados en el flujo del programa. (Joyanes Aguilar, 2015)

4.3.2 VARIABLES LOCALES

Una variable local es aquella que se declara dentro de un procedimiento o función y solo puede ser accedida dentro de su contexto. Esto significa que su alcance está limitado a la parte del programa donde se declaró y no puede ser accedida desde fuera del procedimiento o función en el que se definió.

Las variables locales son útiles porque permiten que los datos se usen temporalmente en un procedimiento o función sin afectar otras partes del programa. Además, al estar limitadas a un contexto específico, pueden ser reutilizadas en diferentes partes del programa sin interferir con otras variables con el mismo nombre.

Cabe mencionar que, al ser variables locales, dentro de su subproceso o proceso el valor asignado mientras terminar la ejecución es su tiempo de validez, en contexto se pierde el valor cuando termina el proceso a la que pertenecen.

4.3.3 PARÁMETROS POR VALOR Y REFERENCIA

Las variables son locales para cada método donde se definen o declaran, es decir, solo se pueden usar en este método. Si intenta utilizar esta variable en otro proceso/subproceso para el que no está declarado, el aplicativo generará un mensaje de error.

Las variables globales se definen fuera del cuerpo de un método y puede ser visto y manipulado por todos los métodos del algoritmo.

En la definición de los parámetros de un proceso o subproceso se puede agregar las palabras claves “Por Valor” o “Por Referencia” para indicar como se pasa cada parámetro. Si no se define por defecto el paso será Por valor a excepción de los arreglos

El paso por referencia implica que, si en el subproceso se modifica el contenido del argumento, este modificará la variable que se utilizó en la invocación o llamada, es decir, se opera sobre la variable original (trabajan en el mismo espacio de memoria).

El paso por Valor si en el subproceso se modifica el contenido del argumento, este no se verá reflejado en la variable que se utilizó en la invocación, es decir se opera sobre una copia de a variable (diferente espacio de memoria).

4.3.4 EJEMPLOS DE PASO DE PARÁMETROS

4.3.4.1 EJEMPLO DE PASO POR VALOR

Incrementar valor a una variable en un subproceso donde se pare un parámetro por valor.

Este programa define un subproceso llamado aumentar, que toma un parámetro m por valor y lo incrementa en uno. Luego, el programa principal define una variable a y le asigna el valor de 5. A continuación, llama al subproceso aumentar y le pasa la variable a como argumento. Finalmente, imprime el valor de a.

Cuando se llama al subproceso aumentar con a como argumento, se crea una copia del valor de a y se le asigna al parámetro m dentro del subproceso. Luego, se incrementa la copia de m en uno, pero esto no afecta el valor original de a. Por lo tanto, cuando se imprime el valor de a después de llamar al subproceso, todavía es 5.

Figura 28: Ejemplo de Referencia por Valor

```

1 SubProceso aumentar(m Por Valor)
2   m←m+1
3 FinSubProceso
4
5 Proceso parametros
6   Definir a Como Entero
7   a←5
8
9   aumentar(a)
10  Escribir a
11 FinProceso
12
*** Ejecución Iniciada. ***
5

```

Fuente: Autor

4.3.4.2 EJEMPLO DE PASO POR REFERENCIA

Realiza el mismo ejemplo incrementar valor a una variable en un subproceso donde se pare un parámetro por referencia y analicemos la diferencia.

Figura 29: Ejemplo de Paso por Referencia

```

SubProceso aumentar(m Por Referencia)
  m←m+1
FinSubProceso

Proceso parametros
  Definir a Como Entero
  a←5

  aumentar(a)
  Escribir a
FinProceso

```

*** Ejecución Iniciada. ***
6

Fuente: Autor

Este programa define un subproceso llamado aumentar, que toma un parámetro m por referencia y lo incrementa en uno. Luego, el programa principal define una variable a y le asigna el valor de 5. A continuación, llama al subproceso aumentar y le pasa la variable a como argumento. Finalmente, imprime el valor de a.

Cuando se llama al subproceso aumentar con a como argumento y m como parámetro por referencia, se pasa una referencia a la variable a en lugar de una copia de su valor. Dentro del subproceso, la referencia se utiliza para acceder y modificar el valor original de a. Por lo tanto, cuando se incrementa m dentro del subproceso, también se incrementa el valor de a. Por lo tanto, cuando se imprime el valor de a después de llamar al subproceso, ahora es 6.

En resumen, la diferencia clave entre este código y el código anterior es la forma en que se pasan los parámetros. En este caso, se utiliza el paso por referencia en lugar del paso por valor.

4.4. EJERCICIOS PROPUESTOS DE SUBPROCESOS

- Descomponer un número entero ingresado en sus números primos en PSeint usando subprocesos.
- Calcular la edad a partir de una fecha ingresada, usando sus subprocesos.
- Eliminar los espacios de una frase ingresada por teclado con subprocesos.

Figura 30: Verifica tus respuestas



Fuente: Autor

5. ARREGLOS

5.1 ARREGLOS UNIDIMENSIONALES

Un arreglo unidimensional es una estructura de datos que permite almacenar y acceder a una colección de elementos de un mismo tipo en una sola dimensión. En PSeInt, los arreglos unidimensionales se declaran indicando el nombre del arreglo y el número máximo de elementos que puede contener. Cada elemento del arreglo se identifica por un índice que indica su posición en la secuencia.

Los arreglos unidimensionales se utilizan comúnmente para almacenar y manipular grandes conjuntos de datos, como listas de números, caracteres o cadenas de texto.

Con PSeInt, es posible realizar operaciones comunes en arreglos unidimensionales, como agregar o eliminar elementos, buscar elementos específicos, ordenar y filtrar elementos, y realizar cálculos matemáticos.

La comprensión de los arreglos unidimensionales es fundamental en la programación, ya que se utilizan en numerosos lenguajes de programación, así como en aplicaciones de base de datos y análisis de datos.

Existen diversas operaciones que se pueden realizar con los arreglos unidimensionales en PSeInt, como la asignación de valores, la lectura de valores, la búsqueda de elementos, el ordenamiento y la impresión de los elementos del arreglo. Estas operaciones se realizan mediante ciclos y estructuras de control de flujo. (Joyanes Aguilar, 2015)

Gráficamente los arreglos unidimensionales se ven de la siguiente manera:

Tabla 4: Arreglo Unidimensional

1	2	3	4	5

Fuente: Autor

Para declarar un arreglo unidimensional en PSeInt, se utiliza la siguiente sintaxis:

Figura 31: Sintaxis Arreglo Unidimensional

```

Algoritmo Array
  Definir arreglo como Entero
  Dimension arreglo[5]

FinAlgoritmo

```

Fuente: Autor

Aunque para llenar de datos el arreglo añadimos los siguientes comandos:

Figura 32: Sintaxis Arreglo Bidimensional

```

Algoritmo Array
  Definir arreglo como Entero
  Dimension arreglo[5]

  arreglo(1) = 5
  arreglo(2) = 8
  arreglo(3) = 9
  arreglo(4) = 11
  arreglo(5) = 24

FinAlgoritmo

```

Fuente: Autor

Tabla 5: Arreglo Relleno

1	2	3	4	5
5	8	9	11	24

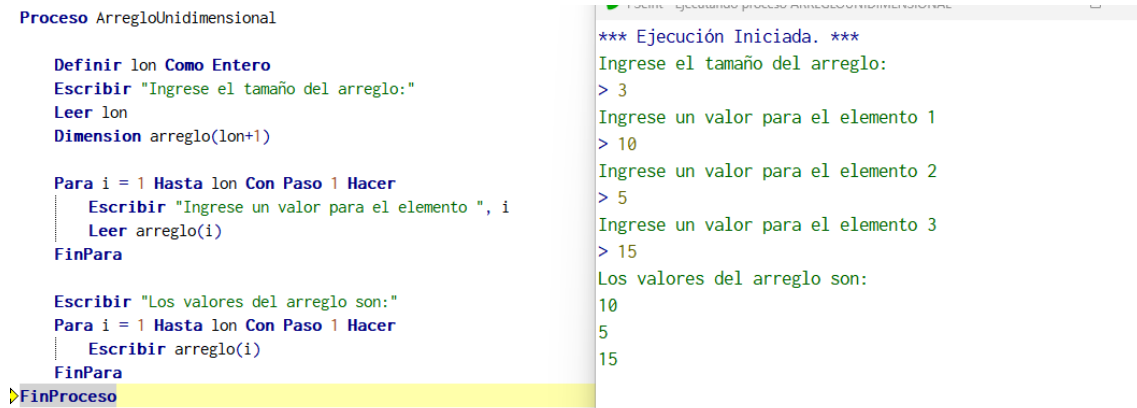
Fuente: Autor

5.1.1 EJERCICIOS CON ARREGLOS UNIDIMENSIONALES

5.1.1.1 EJERCICIO RELLENO DE ARREGLO

Crear un arreglo unidimensional donde el usuario ingrese la longitud y rellenarlo de datos numéricos.

Figura 33: Ejercicio Rellenar Arreglo



```

Proceso ArregloUnidimensional
    Definir lon Como Entero
    Escribir "Ingrese el tamaño del arreglo:"
    Leer lon
    Dimension arreglo(lon+1)

    Para i = 1 Hasta lon Con Paso 1 Hacer
        Escribir "Ingrese un valor para el elemento ", i
        Leer arreglo(i)
    FinPara

    Escribir "Los valores del arreglo son:"
    Para i = 1 Hasta lon Con Paso 1 Hacer
        Escribir arreglo(i)
    FinPara
FinProceso
  
```

```

*** Ejecución Iniciada. ***
Ingrese el tamaño del arreglo:
> 3
Ingrese un valor para el elemento 1
> 10
Ingrese un valor para el elemento 2
> 5
Ingrese un valor para el elemento 3
> 15
Los valores del arreglo son:
10
5
15
  
```

Fuente: Autor

Este código le pedirá al usuario que ingrese el tamaño del arreglo, lo creará con esa longitud, pedirá al usuario que ingrese un valor para cada elemento del arreglo y luego mostrará los valores ingresados en pantalla.

5.1.1.2 EJERCICIO ORDENAR UNA LISTA DE NÚMEROS

Crear un arreglo unidimensional en PSeInt para ordenar una lista de números de menor a mayor.

Figura 34: Ejecución Ejemplo Ordenar Lista de Menor a Mayor

```

*** Ejecución Iniciada. ***
Ingrese la cantidad de números que desea ordenar (máximo
10):
> 4
Ingrese el número 1:
> 5
Ingrese el número 2:
> 4
Ingrese el número 3:
> 15
Ingrese el número 4:
> 9
Los números ordenados son:
4
5
9
15
  
```

Fuente: Autor

Figura 35: Ordenar Lista de Menor a mayor

```

Algoritmo OrdenarNumeros
  Definir cantidad, i, j, temp como Entero
  Definir numeros como Entero
  Dimension numeros[10]

  Escribir "Ingrese la cantidad de números que desea ordenar (máximo 10): "
  Leer cantidad

  Para i ← 0 Hasta cantidad-1 Con Paso 1 Hacer
    Escribir "Ingrese el número ", i+1, ": "
    Leer numeros[i]
  FinPara

  Para i ← 0 Hasta cantidad-2 Con Paso 1 Hacer
    Para j ← i+1 Hasta cantidad-1 Con Paso 1 Hacer
      Si numeros[i] > numeros[j] Entonces
        temp ← numeros[i]
        numeros[i] ← numeros[j]
        numeros[j] ← temp
      FinSi
    FinPara
  FinPara

  Escribir "Los números ordenados son:"
  Para i ← 0 Hasta cantidad-1 Con Paso 1 Hacer
    Escribir numeros[i]
  FinPara
FinAlgoritmo

```

Fuente: Autor

En este ejemplo, se declara un arreglo unidimensional llamado números con capacidad para almacenar 10 valores enteros. Luego, se solicita al usuario que ingrese la cantidad de números que desea ordenar y se utiliza un bucle “Para” para solicitar al usuario que ingrese cada número.

Después, se utiliza otro par de bucles Para para ordenar los números ingresados de menor a mayor utilizando el método de ordenamiento por selección.

Finalmente, se muestra la lista de números ordenados al usuario.

5.1.1.3 EJERCICIO CALCULAR DESVIACIÓN ESTANDAR

Calcular la desviación estándar de un conjunto de datos ingresados por el usuario.

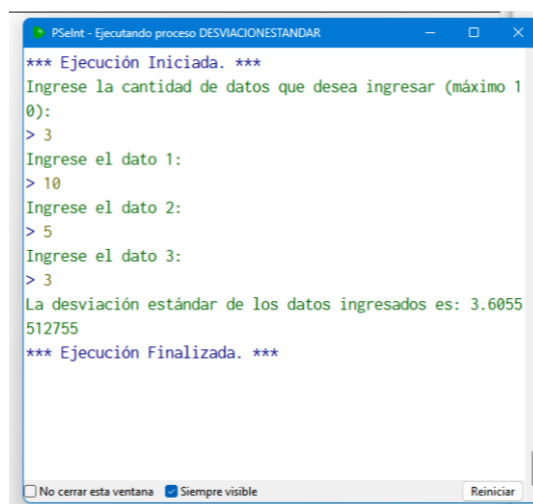
Figura 36: Ejemplo Desviación Estándar

```

1  Algoritmo DesviacionEstandar
2      Definir cantidad, i como Entero
3      Definir datos como Real
4      Dimension datos[10]
5      Definir promedio, suma, desviacion como Real
6
7      Escribir "Ingrese la cantidad de datos que desea ingresar (máximo 10): "
8      Leer cantidad
9
10     Para i  $\leftarrow$  0 Hasta cantidad-1 Con Paso 1 Hacer
11         Escribir "Ingrese el dato ", i+1, ": "
12         Leer datos[i]
13     FinPara
14
15     suma  $\leftarrow$  0
16     Para i  $\leftarrow$  0 Hasta cantidad-1 Con Paso 1 Hacer
17         suma  $\leftarrow$  suma + datos[i]
18     FinPara
19
20     promedio  $\leftarrow$  suma / cantidad
21
22     suma  $\leftarrow$  0
23     Para i  $\leftarrow$  0 Hasta cantidad-1 Con Paso 1 Hacer
24         suma  $\leftarrow$  suma + (datos[i] - promedio)  $\uparrow$  2
25     FinPara
26
27     desviacion  $\leftarrow$  raiz(suma / (cantidad - 1))
28
29     Escribir "La desviación estándar de los datos ingresados es: ", desviacion
30 FinAlgoritmo

```

Figura 37: Ejecución de Desviación Estándar



Fuente: Autor

En este ejemplo, se utiliza un arreglo unidimensional llamado datos para almacenar los datos ingresados por el usuario.

Luego, se utiliza un bucle Para para solicitar al usuario que ingrese cada dato y se calcula el promedio de los datos ingresados.

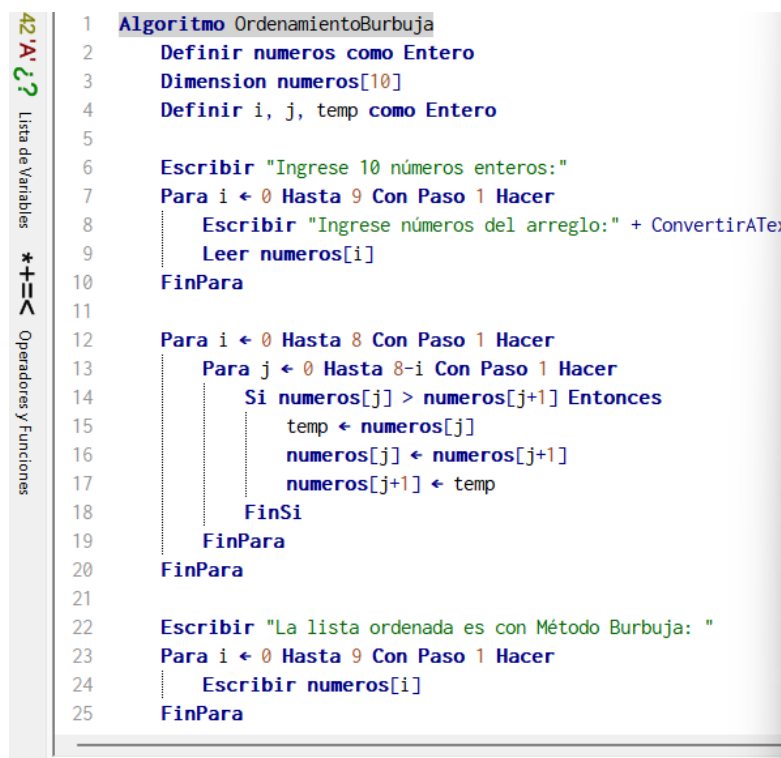
Después, se utiliza otro par de bucles Para para calcular la suma de los cuadrados de las desviaciones de cada dato respecto al promedio, y finalmente se calcula la desviación estándar utilizando la fórmula correspondiente.

Finalmente, se muestra el resultado de la desviación estándar al usuario.

5.1.1.4 EJERCICIO METODO DE ORDENAMIENTO BURBUJA

Ordenar una lista de números de a mayor utilizando el algoritmo de burbuja.

Figura 38: Ejemplo Ordenamiento Burbuja



```

1  Algoritmo OrdenamientoBurbuja
2  Definir numeros como Entero
3  Dimension numeros[10]
4  Definir i, j, temp como Entero
5
6  Escribir "Ingrese 10 números enteros:"
7  Para i ← 0 Hasta 9 Con Paso 1 Hacer
8      Escribir "Ingrese números del arreglo:" + ConvertirATexto(i)
9      Leer numeros[i]
10  FinPara
11
12  Para i ← 0 Hasta 8 Con Paso 1 Hacer
13      Para j ← 0 Hasta 8-i Con Paso 1 Hacer
14          Si numeros[j] > numeros[j+1] Entonces
15              temp ← numeros[j]
16              numeros[j] ← numeros[j+1]
17              numeros[j+1] ← temp
18          FinSi
19      FinPara
20  FinPara
21
22  Escribir "La lista ordenada es con Método Burbuja: "
23  Para i ← 0 Hasta 9 Con Paso 1 Hacer
24      Escribir numeros[i]
25  FinPara
  
```

Fuente: Autor

Figura 39: Ejecución Ordenamiento Burbuja

```

Ingrese números del arreglo:0
> 10
Ingrese números del arreglo:1
> 8
Ingrese números del arreglo:2
> 5
Ingrese números del arreglo:3
> 6
Ingrese números del arreglo:4
> 3
Ingrese números del arreglo:5
> 6
Ingrese números del arreglo:6
> 4
Ingrese números del arreglo:7
> 2
Ingrese números del arreglo:8
> 3
Ingrese números del arreglo:9
> 3
La lista ordenada es con Método Burbuja:
2
3
3
3
4
5
5
5
8
10
*** Ejecución Finalizada. ***

```

Fuente: Autor

En este ejemplo, se utiliza un arreglo unidimensional llamado 'numeros' para almacenar una lista de 10 números enteros ingresados por el usuario.

Luego, se utiliza un algoritmo de ordenamiento llamado "ordenamiento de burbuja" para ordenar los números de menor a mayor. El algoritmo funciona comparando cada par de elementos adyacentes e intercambiándolos si están en el orden incorrecto. Este proceso se repite varias veces hasta que la lista esté completamente ordenada.

Finalmente, se muestra la lista ordenada utilizando un bucle Para.

Este ejemplo es un poco más complejo que los anteriores, pero es un buen ejemplo de cómo utilizar un arreglo unidimensional para ordenar datos en un programa.

5.2 ARREGLOS MULTIDIMENSIONALES

Los arreglos bidimensionales son una estructura de datos útil que permite almacenar información organizada en dos dimensiones. (Luis et al., n.d.)

Una matriz bidimensional se declara con una sintaxis similar a la de los arreglos unidimensionales(Herrera et al., n.d.) , pero especificando las dimensiones de la matriz. Algunos de los usos más comunes para los arreglos bidimensionales son la representación de tableros de juegos(Luis et al., n.d.) , la creación de tablas de datos y horarios.

En el capítulo 8 del libro "PSeInt: Algoritmos y Estructuras de Datos" se profundiza en el manejo de los arreglos, incluyendo los arreglos unidimensionales y bidimensionales. (Herrera et al., n.d.)

En conclusión, los arreglos bidimensionales son una herramienta poderosa que permite almacenar información organizada en dos dimensiones(Luis et al., n.d.) misma que puede ser eliminada, modificada y actualizada por el usuario dentro de la ejecución de código en cualquier momento.

Gráficamente los arreglos unidimensionales se ven de la siguiente manera:

Tabla 6: Ejemplo Visual Arreglo Bidimensional

	0	2	3
1			
2			

Fuente: Autor

Aunque para llenar de datos de la matriz añadimos los siguientes comandos:

Figura 40: Sintaxis de Arreglo Bidimensional

```

Algoritmo arrayBimensional
  definir num como Entero
  Dimension num[2,3]

  num(0,0)←12
  num(0,1)←10
  num(0,2)←1

  num(1,0)←6
  num(1,1)←1
  num(1,2)←0

FinAlgoritmo
  
```

Fuente: Autor

Donde gráficamente queda de la siguiente manera:

Tabla 7: Relleno Arreglo Bidimensional

	0	2	3
0	12	10	1
1	6	1	0

Fuente: Autor

5.2.1 EJERCICIOS CON ARREGLOS BIDIMENSIONALES

5.2.1.1 EJERCICIO CARGAR MATRIZ

Construir un algoritmo que recorra una matriz de 3 x 3 debo llenar esta matriz antes, imprima todos los números que estén bajo un valor ingresado por el usuario.

Figura 41: Ejemplo de Carga Matriz

```

1  Proceso Arreglo
2      Dimension A[3,3];
3      Definir A, i, j Como Entero;
4
5      Para i<=0 Hasta 2 Con Paso 1 Hacer
6          Para j<=0 Hasta 2 Con Paso 1 Hacer
7              Escribir "De un numero";
8              Leer A[i,j];
9          FinPara
10     FinPara
11
12     Escribir "La matriz ingresada es ";
13     Para i<=0 Hasta 2 Con Paso 1 Hacer
14         Escribir " "; //Esto es solo para dar un poco de formato
15         Para j<=0 Hasta 2 Con Paso 1 Hacer
16             Escribir Sin Saltar A[i,j], " ";
17         FinPara
18     FinPara
19     Escribir " ";
20 FinProceso

```

De un numero
> 1
De un numero
> 2
De un numero
> 3
De un numero
> 4
De un numero
> 5
De un numero
> 6
De un numero
> 7
De un numero
> 8
De un numero
> 9
La matriz ingresada es
1 2 3
4 5 6
7 8 9
*** Ejecución Finalizada. ***

Fuente: Autor

El código muestra un ejemplo básico de cómo declarar, llenar y mostrar una matriz de 3x3 en PSeInt, un programa para la enseñanza de algoritmos y programación.

Primero, se define la matriz A de dimensiones 3x3. Luego, mediante dos ciclos anidados, se solicita al usuario que ingrese un número en cada celda de la matriz, utilizando la instrucción "Leer" para asignar el valor ingresado en cada posición i,j de la matriz.

Se muestra la matriz ingresada al usuario mediante otro conjunto de ciclos anidados. Para imprimir cada elemento de la matriz, se utiliza la instrucción "Escribir Sin Saltar", que permite mostrar varios elementos en una sola línea separados por un espacio.

En resumen, el código demuestra cómo se puede crear y trabajar con matrices en PSeInt utilizando ciclos anidados y las instrucciones "Leer" y "Escribir". Este ejemplo es bastante simple, pero podría ser la base para la creación de programas más complejos que involucren matrices de diferentes tamaños y formas.

5.2.1.2 EJERCICIO SUMA DE MATRICES

Crear un algoritmo que ingrese los datos de dos matrices de 3x3 y remita la información de cada matriz y la suma de estas en una nueva matriz.

Figura 42: Ejemplo Suma Matrices

```

Proceso SumaMatrices
    Dimension A[3,3], B[3,3], C[3,3]; // Definimos las matrices A, B y C, todas de tamaño 3x3
    Definir i, j Como Entero;

    // Pedimos al usuario que ingrese los elementos de la matriz A
    Escribir "Ingrese los elementos de la matriz A:";
    Para i<=0 Hasta 2 Con Paso 1 Hacer
        Para j<=0 Hasta 2 Con Paso 1 Hacer
            Escribir "A[" , i, "]" , j, "]: ";
            Leer A[i,j];
        FinPara
    FinPara

    // Pedimos al usuario que ingrese los elementos de la matriz B
    Escribir "Ingrese los elementos de la matriz B:";
    Para i<=0 Hasta 2 Con Paso 1 Hacer
        Para j<=0 Hasta 2 Con Paso 1 Hacer
            Escribir "B[" , i, "]" , j, "]: ";
            Leer B[i,j];
        FinPara
    FinPara

    // Realizamos la suma de matrices A + B y almacenamos el resultado en C
    Para i<=0 Hasta 2 Con Paso 1 Hacer
        Para j<=0 Hasta 2 Con Paso 1 Hacer
            C[i,j] ← A[i,j] + B[i,j];
        FinPara
    FinPara

    // Mostramos las tres matrices al usuario
    Escribir "La matriz A es:";
    Para i<=0 Hasta 2 Con Paso 1 Hacer
        Escribir " "; // Esto es solo para dar un poco de formato
        Para j<=0 Hasta 2 Con Paso 1 Hacer
            Escribir Sin Saltar A[i,j], " ";
        FinPara
    FinPara
    Escribir " "; // Imprimimos una línea en blanco para dar formato

    Escribir "La matriz B es:";
    Para i<=0 Hasta 2 Con Paso 1 Hacer
        Escribir " "; // Esto es solo para dar un poco de formato
        Para j<=0 Hasta 2 Con Paso 1 Hacer
            Escribir Sin Saltar B[i,j], " ";
        FinPara
    FinPara
    Escribir " "; // Imprimimos una línea en blanco para dar formato

    Escribir "La matriz resultante C es:";
    Para i<=0 Hasta 2 Con Paso 1 Hacer
        Escribir " "; // Esto es solo para dar un poco de formato
        Para j<=0 Hasta 2 Con Paso 1 Hacer
            Escribir Sin Saltar C[i,j], " ";
        FinPara
    FinPara
    Escribir " "; // Imprimimos una línea en blanco para dar formato

FinProceso

```

Fuente: Autor

Figura 43: Ejecución de Suma Matrices

```
> 4
B[1][1]:
> 5
B[1][2]:
> 6
B[2][0]:
> 7
B[2][1]:
> 8
B[2][2]:
> 9
La matriz A es:

1 2 3
4 5 6
7 8 9
La matriz B es:

1 2 3
4 5 6
7 8 9
La matriz resultante C es:

2 4 6
8 10 12
14 16 18
*** Ejecución Finalizada. ***
```

Fuente: Autor

Este ejemplo solicita al usuario que ingrese los elementos de dos matrices, A y B, y luego realiza la suma de matrices $A + B$, almacenando el resultado en la matriz C. Finalmente, se muestran las tres matrices al usuario utilizando ciclos anidados.

5.2.1.3 EJERCICIO ESTUDIANTES

Crear una matriz para almacenar datos de estudiantes y luego se calcula el promedio de cada uno de ellos.

Figura 44: Ejemplo Estudiantes

```

Proceso PromedioEstudiantes
    // Definimos la matriz calificaciones, con 5 filas (para 5 estudiantes) y 3 columnas (para 3 calificaciones cada uno)
    Definir calificaciones Como Real
    Dimension calificaciones[5,3] ;
    Definir i, j Como Entero;

    Para i<0 Hasta 4 Con Paso 1 Hacer
        Para j<0 Hasta 2 Con Paso 1 Hacer
            Escribir "Ingrese la calificación ", j+1, " del estudiante ", i+1;
            // Leemos la calificación correspondiente y la almacenamos en la matriz calificaciones
            Leer calificaciones[i,j];
        FinPara
    FinPara
    Definir suma Como Real
    Definir promedio Como Real
    Para i<0 Hasta 4 Con Paso 1 Hacer
        suma ← 0;
        Para j<0 Hasta 2 Con Paso 1 Hacer
            // Calculamos la suma de las calificaciones del estudiante i
            suma ← suma + calificaciones[i,j];
        FinPara
        // Calculamos el promedio dividiendo la suma entre el número de calificaciones (en este caso, 3)
        promedio ← suma / 3;
        Escribir "El promedio del estudiante ", i+1, " es ", promedio;
    FinPara
FinProceso
  
```

Fuente: Autor

Figura 45: Ejecución Ejemplo Estudiantes

```

> 5
Ingrese la calificación 3 del estudiante 3
> 4
Ingrese la calificación 1 del estudiante 4
> 3
Ingrese la calificación 2 del estudiante 4
> 2
Ingrese la calificación 3 del estudiante 4
> 1
Ingrese la calificación 1 del estudiante 5
> 6
Ingrese la calificación 2 del estudiante 5
> 8
Ingrese la calificación 3 del estudiante 5
> 6
El promedio del estudiante 1 es 10
El promedio del estudiante 2 es 8
El promedio del estudiante 3 es 5
El promedio del estudiante 4 es 2
El promedio del estudiante 5 es 6.666666667
*** Ejecución Finalizada. ***
  
```

☒ No cerrar esta ventana ☒ Siempre visible

Reiniciar

Fuente: Autor

En este ejemplo, primero definimos la matriz calificaciones para almacenar las calificaciones de los estudiantes. Luego, mediante dos bucles Para, leemos las calificaciones de cada estudiante y las almacenamos en la matriz.

A continuación, usamos otro bucle Para para calcular el promedio de cada estudiante, sumando las tres calificaciones y dividiendo el resultado entre 3. Finalmente, imprimimos el promedio de cada estudiante en la pantalla.

5.2.1.4 EJERCICIO PRODUCTOS

Realizar una matriz para almacenar datos de ventas de productos y luego se calcula el total de ventas de cada producto.

Figura 46: Ejemplo Ventas

Proceso TotalVentas

definir ventas **Como Entero**

// Definimos la matriz ventas, con 4 filas (para 4 productos) y 3 columnas (para 3 meses)

Dimension ventas[4,3] ;

Definir i, j **Como Entero**;

Para i←0 **Hasta** 3 **Con Paso** 1 **Hacer**

Para j←0 **Hasta** 2 **Con Paso** 1 **Hacer**

// Leemos las ventas correspondientes y las almacenamos en la matriz ventas

Escribir "Ingrese las ventas del producto ", i+1, " en el mes ", j+1;

Leer ventas[i,j];

FinPara

FinPara

Definir total **Como Entero**;

Para i←0 **Hasta** 3 **Con Paso** 1 **Hacer**

// Calculamos el total de ventas del producto i sumando las ventas de cada mes

total ← 0;

Para j←0 **Hasta** 2 **Con Paso** 1 **Hacer**

total ← total + ventas[i,j];

FinPara

Escribir "El total de ventas del producto ", i+1, " es ", total;

FinPara

FinProceso

Fuente: Autor

Figura 47: Ejecución Total ventas

```

Ingrese las ventas del producto 2 en el mes 3
> 36
Ingrese las ventas del producto 3 en el mes 1
> 12
Ingrese las ventas del producto 3 en el mes 2
> 15
Ingrese las ventas del producto 3 en el mes 3
> 18
Ingrese las ventas del producto 4 en el mes 1
> 16
Ingrese las ventas del producto 4 en el mes 2
> 12
Ingrese las ventas del producto 4 en el mes 3
> 10
El total de ventas del producto 1 es 71
El total de ventas del producto 2 es 61
El total de ventas del producto 3 es 45
El total de ventas del producto 4 es 38
*** Ejecución Finalizada. ***

```

Fuente: Autor

En este ejemplo, primero definimos la matriz ventas para almacenar los datos de ventas de los productos. Luego, mediante dos bucles Para, leemos las ventas de cada producto en cada mes y las almacenamos en la matriz.

A continuación, usamos otro bucle Para para calcular el total de ventas de cada producto, sumando las ventas de cada mes. Finalmente, imprimimos el total de ventas de cada producto en la pantalla.

5.3.1 EJERCICIOS PROPUESTOS ARREGLOS Y MATRICES

- Ingresar una matriz cuadrada por teclado y definir si la misma cumple con el principio de simetría.

- El usuario las dimensiones de dos matrices (n y m), y luego pide que se ingresen los valores de cada matriz. Luego, realiza las siguientes operaciones:

Calcula la suma de las matrices A y B, y almacena el resultado en la matriz C.

Calcula la traspuesta de la matriz C, y almacena el resultado en la matriz D.

Calcula el producto de la matriz A y la traspuesta de la matriz C, y almacena el resultado en la matriz E.

Cada operación se imprime en pantalla para que el usuario pueda ver el resultado.

- Crear dos matrices A y B de dimensiones 3×3 y una matriz C de dimensiones 2×2 . El objetivo es realizar las siguientes operaciones:

Llenar las matrices A y B con valores enteros aleatorios entre 1 y 10.

Calcular la suma de las matrices A y B y guardar el resultado en la matriz C.

Mostrar en pantalla las matrices A, B y C.

- Tienes una matriz A de dimensiones 4×4 . El objetivo es realizar las siguientes operaciones:

Llenar la matriz A con valores enteros aleatorios entre 1 y 100.

Encontrar el valor máximo y el valor mínimo de la matriz A y mostrarlos en pantalla.

Calcular la suma de los elementos de la diagonal principal de la matriz A y mostrar el resultado en pantalla.

Calcular la suma de los elementos que están por encima de la diagonal secundaria de la matriz A y mostrar el resultado en pantalla.

- Verificar si un numero existe dentro de una matriz si encuentra el numero devolver su posición caso contrario devolver 0.

El programa de ingresar la matriz por teclado.



El programa debe permitir al usuario ingresar el tamaño de la matriz.

Figura 48: Verifica las resoluciones



Fuente: Autor

6. RECURSIVIDAD

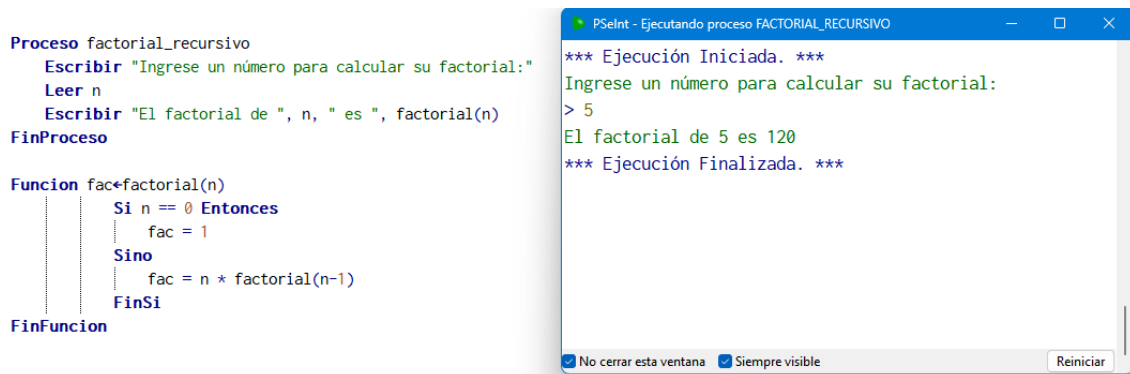
6.1 DEFINICIÓN

La recursividad es un concepto importante en la programación y se refiere a la capacidad de una función para llamarse a sí misma. En otras palabras, una función recursiva es aquella que se llama a sí misma para resolver un problema.

La recursividad se puede utilizar en situaciones en las que el problema se puede descomponer en subproblemas más pequeños, cada uno de los cuales se resuelve utilizando la misma función. Esto puede simplificar la solución del problema y hacer que el código sea más fácil de entender y mantener.

En el libro "Estructuras de datos en Java" de Luis Joyanes Aguilar, se puede encontrar información sobre la recursividad.(Joyanes Aguilar, 2015) En particular, se describen varios ejemplos de funciones recursivas, como la función factorial.

Figura 49: Ejemplo Recursividad - Factorial



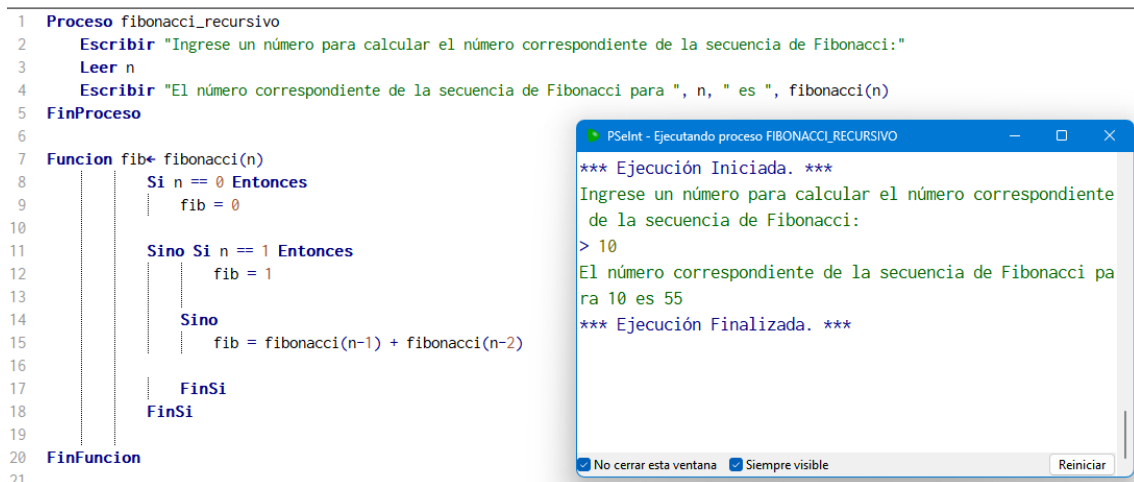
Fuente: Autor

La función factorial es recursiva, ya que se llama a sí misma para resolver el subproblema de calcular la factorial de $n-1$. En el caso base, cuando n es cero, la función devuelve 1. De lo contrario, la función multiplica n por el resultado de llamar a la función factorial con $n-1$ como argumento.

El proceso **factorial_recursivo** solicita al usuario un número y luego llama a la función factorial para calcular su factorial utilizando recursividad. El resultado se muestra en pantalla.

Otro ejemplo de función recursiva que se puede encontrar en el libro es la función de Fibonacci.

Figura 50: Ejemplo de Recursividad - Fibonacci



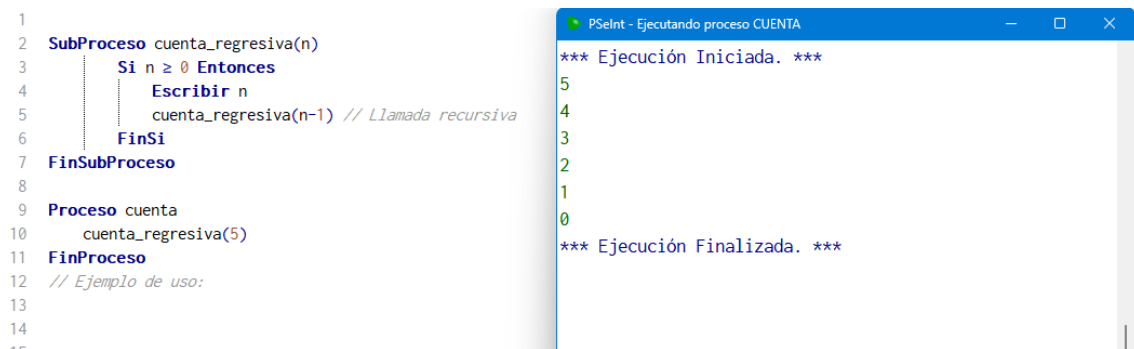
Fuente: Autor

6.2 TIPOS DE RECURSIVIDAD

Existen varios tipos de recursividad, estos están delimitados por la forma en que se invocan por tal entre ellos tenemos los siguientes:

Recursividad directa: es cuando una función se llama a sí misma de manera directa para resolver un subproblema.

Figura 51: Ejemplo de Recursividad Directa



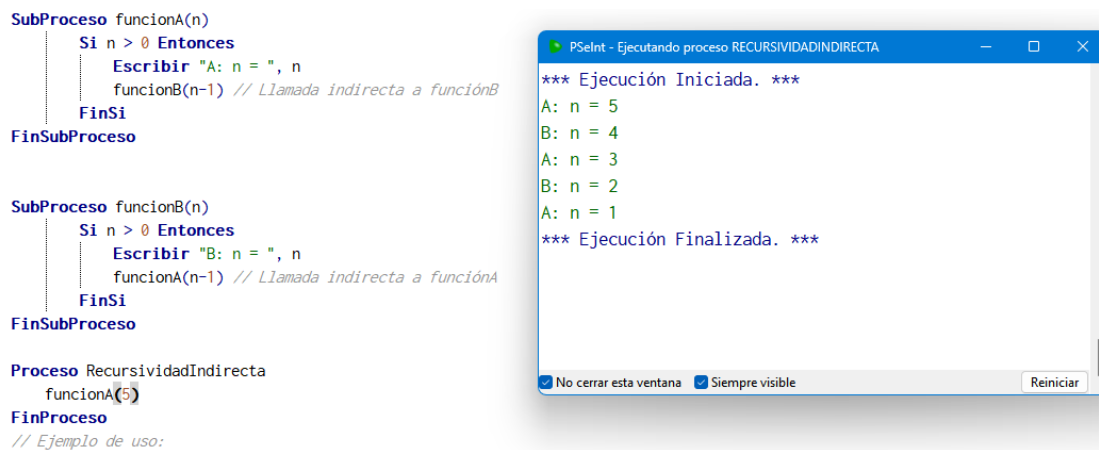
Fuente: Autor

En este ejemplo, la función **cuenta_regresiva** utiliza recursividad directa para imprimir una cuenta regresiva desde el número n hasta cero. La función se llama a sí misma de manera directa para imprimir el número n y luego volver a llamar a la función con $n-1$ como argumento, hasta que n sea menor que cero.

Al ejecutar el proceso con **cuenta_regresiva(5)**, se imprimirán los números 5, 4, 3, 2, 1 y 0 en la consola. Cada llamada recursiva reduce el valor de n en una unidad hasta que se alcanza el caso base ($n < 0$) y se detiene la recursividad.

Recursividad indirecta: es cuando una función se llama a otra función, y esta última función a su vez se llama a la primera, creando así una cadena de llamadas recursivas.

Figura 52: Ejemplo Recursividad Indirecta



Fuente: Autor

En este ejemplo, la función `funcionA` llama indirectamente a la función `funcionB`, y a su vez la función `funcionB` llama indirectamente a `funcionA`. De esta manera se crea un bucle infinito de llamadas recursivas entre ambas funciones.

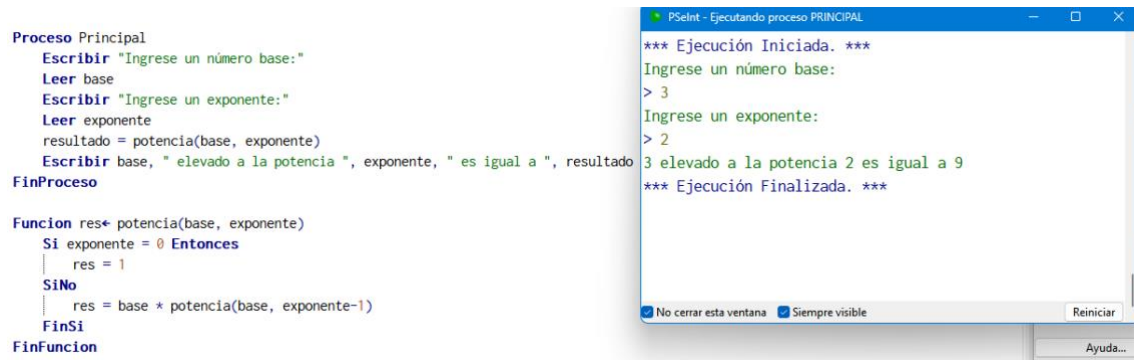
Cuando se ejecuta `funcionA(5)`, se imprimirá en la consola una serie de mensajes alternando entre "A: $n = x$ " y "B: $n = y$ ", donde x e y son valores que van disminuyendo hasta llegar a cero. Esto sucede porque cada función llama a la otra con $n-1$ como argumento, y nunca se llega al caso base para detener la recursividad.

Este ejemplo de recursividad indirecta es útil para comprender cómo una función puede llamar indirectamente a otra función, y cómo puede crearse un bucle infinito de llamadas recursivas que pueden causar un desbordamiento de pila y/o una pérdida de rendimiento.

Recursividad múltiple: es cuando una función se llama a sí misma varias veces para resolver un subproblema. Por ejemplo, en el algoritmo de QuickSort, la función de

partición se llama a sí misma dos veces para ordenar recursivamente los subarrays de elementos menores y mayores al pivote.

Figura 53: Ejemplo de Recursividad Múltiple

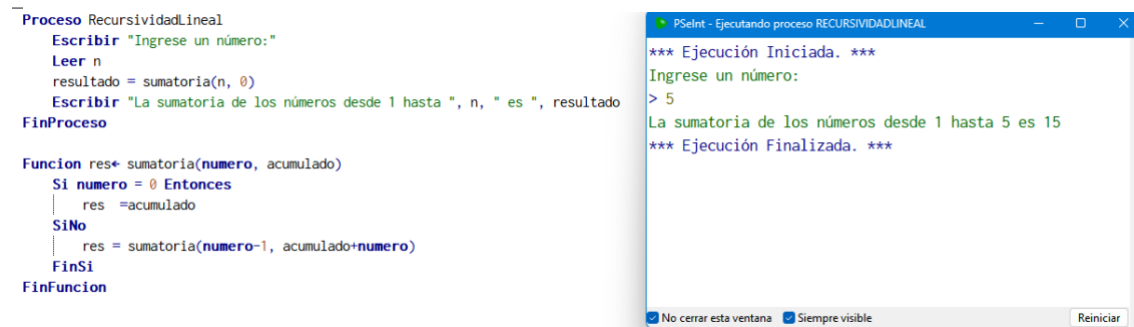


Fuente: Autor

En este ejemplo, la función potencia se llama a sí misma varias veces para calcular la potencia de un número dado. Si el exponente es cero, la función devuelve 1. Si no, se llama a sí misma con el mismo número base y un exponente más pequeño (es decir, exponente-1), lo que a su vez llama a la función potencia con un exponente aún más pequeño, y así sucesivamente hasta que se llega a un exponente de cero. Luego, los resultados se multiplican para obtener la potencia del número original.

Recursividad lineal o de cola: es cuando la llamada recursiva es la última operación realizada por la función, lo que significa que no hay cálculos pendientes cuando se llama a la función recursiva. En este caso, el compilador puede optimizar la llamada recursiva para que no utilice más memoria de la necesaria, ya que no hay variables locales que deban conservarse.

Figura 54: Ejemplo de Recursividad Lineal



Fuente: Autor

En este ejemplo, la función sumatoria se llama a sí misma como última acción en su cuerpo para calcular la sumatoria de los números desde 1 hasta un número dado. Si el número ingresado es cero, la función devuelve el valor acumulado. Si no, se llama a sí misma con el número $\text{numero}-1$ y el valor acumulado $\text{acumulado}+\text{numero}$, y se devuelve el resultado de la llamada recursiva. En cada llamada recursiva, el valor acumulado se va actualizando con la suma del número actual y los números anteriores, hasta que se llega al número 1. En este punto, la llamada recursiva final devuelve el resultado de la sumatoria completa.

6.3 EJERCICIOS PROPUESTOS DE RECURSIVIDAD

- Realizar un programa con recursividad para ordenar un arreglo.
- Realizar un programa que indique si la una palabra ingresada por teclado es palíndroma, usar recursividad.
- Realizar un programa con recursividad para verificar cual es la longitud de una cadena ingresada por teclado.
- Realizar un programa con recursividad que convierta un numero en binario.

Figura 55: Verifica las resoluciones.

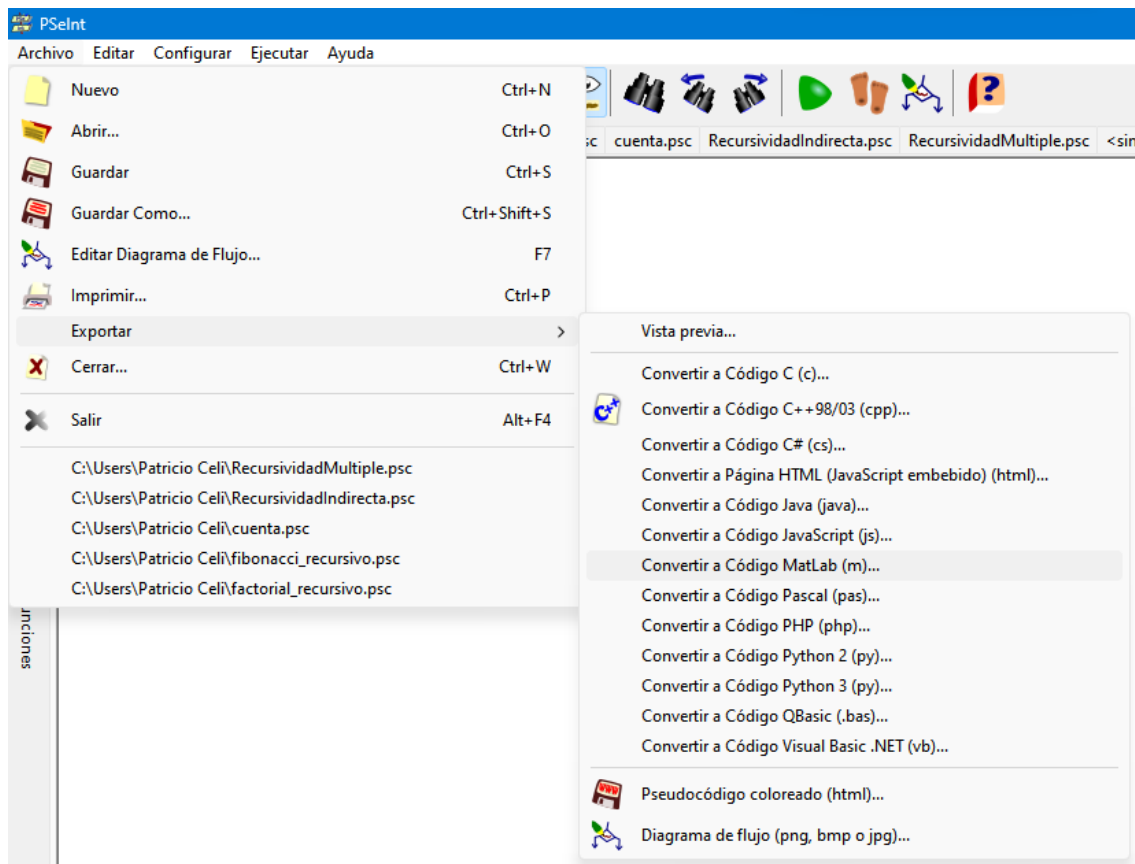


Fuente: Autor

7. ALGORITMOS A CÓDIGO FUENTE

Para convertir el algoritmo realizado en el workflow en código fuente debemos acceder a la pestaña Archivo y seleccionar la opción Exportar, donde encontraremos el IDE de desarrollo por el cual vamos a generar:

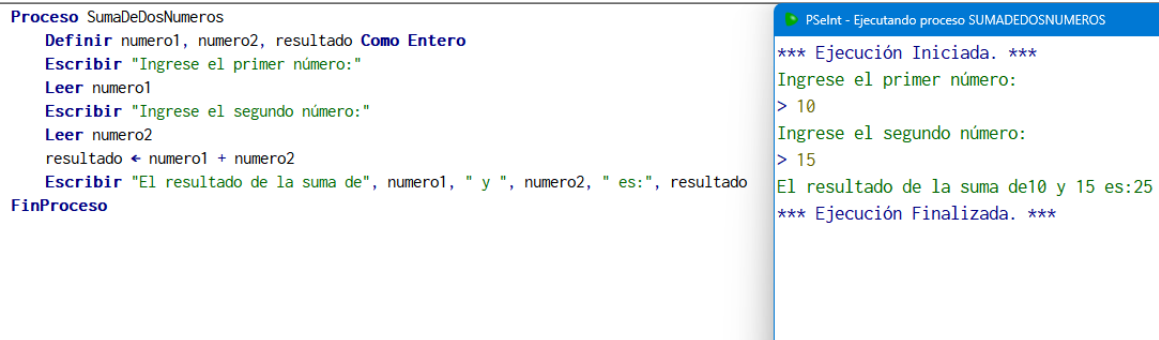
Figura 56: Exportación de Código Fuente



Fuente: Autor

A continuación, usaremos un ejemplo de la suma de dos números y vamos a realizar la exportación de código fuente en diferentes lenguajes.

Figura 57: Ejemplo Suma de 2 Números



Fuente: Autor

Código Fuente en HTML

Figura 58: Ejemplo Código Html

```

<!DOCTYPE html>
<HTML>
  <HEAD>
    <TITLE>SUMADEDOSNUMEROS</TITLE>
  </HEAD>
  <BODY onload="sumadedosnumeros();">
    <SCRIPT type="text/javascript">
      //Este código ha sido generado por el módulo psexport 20180802-w32 de PSeInt.
      //Es posible que el código generado no sea completamente correcto. Si encuentra
      //errores por favor reportelos en el foro (http://pseint.sourceforge.net).

      function sumadedosnumeros() {
        var numero1 = new Number();
        var numero2 = new Number();
        var resultado = new Number();
        document.write("Ingrese el primer número:", '<BR/>');
        numero1 = Number(prompt());
        document.write("Ingrese el segundo número:", '<BR/>');
        numero2 = Number(prompt());
        resultado = numero1+numero2;
        document.write("El resultado de la suma de", numero1, " y ", numero2, " es:", resultado, '<BR/>');
      }
    </SCRIPT>
  </BODY>
</HTML>
  
```

Fuente: Autor

Código Fuente en JAVA

Figura 59: Código Fuente JAVA

```
// En java, el nombre de un archivo fuente debe coincidir con el
// nombre de la clase que contiene,
// por lo que este archivo debería llamarse
"SUMADEDOSNUMEROS.java."

import java.io.*;

public class sumadedosnumeros {

    public static void main(String args[]) throws IOException
    {
        BufferedReader bufEntrada = new BufferedReader(new
InputStreamReader(System.in));
        int numero1;
        int numero2;
        int resultado;
        System.out.println("Ingrese el primer número:");
        numero1 = Integer.parseInt(bufEntrada.readLine());
        System.out.println("Ingrese el segundo número:");
        numero2 = Integer.parseInt(bufEntrada.readLine());
        resultado = numero1+numero2;
        System.out.println("El resultado de la suma
de "+numero1+" y "+numero2+" es: "+resultado);
    }

}
```

Fuente: Autor

Código Fuente en PHP

Figura 60: Ejemplo Código Fuente PHP

```
<?php
// Este código ha sido generado por el módulo psexport 20180802-w32 de PSeInt.
// Es posible que el código generado no sea completamente correcto. Si encuentra
// errores por favor reportelos en el foro (http://pseint.sourceforge.net).

$stdin = fopen('php://stdin','r');
settype($numero1,'integer');
settype($numero2,'integer');
settype($resultado,'integer');
echo 'Ingrese el primer número:',PHP_EOL;
fscanf($stdin,"%d",$numero1);
echo 'Ingrese el segundo número:',PHP_EOL;
fscanf($stdin,"%d",$numero2);
$resultado = $numero1+$numero2;
echo 'El resultado de la suma de',$numero1,' y ', $numero2,' es:',$resultado,PHP_EOL;
?>
```

Fuente: Autor

Código Fuente en Javascript

Figura 61: Ejemplo Código Fuente Javascript

```
w32 de PSeInt.
// Es posible que el codigo generado no sea completamente
correcto. Si encuentra
// errores por favor reportelos en el foro
(http://pseint.sourceforge.net).

function sumadedosnumeros() {
    var numero1 = new Number();
    var numero2 = new Number();
    var resultado = new Number();
    document.write("Ingrese el primer número:", '<BR/>');
    numero1 = Number(prompt());
    document.write("Ingrese el segundo número:", '<BR/>');
    numero2 = Number(prompt());
    resultado = numero1+numero2;
    document.write("El resultado de la suma de", numero1, " y
", numero2, " es:", resultado, '<BR/>');
}
```

Fuente: Autor

Código Fuente en Matlab

Figura 62: Ejemplo Código Fuente Matlab

```
C:\Users\Patricio Celi\OneDrive\Desktop\CodigoFuente> sin_titulo.m
1 % Este codigo ha sido generado por el modulo psexport 20180802-w32 de PSeInt.
2 % Es posible que el codigo generado no sea completamente correcto. Si encuentra
3 % errores por favor reportelos en el foro (http://pseint.sourceforge.net).
4
5
6 function sumadedosnumeros()
7     numero1=0;
8     numero2=0;
9     resultado=0;
10    disp('Ingrese el primer número:');
11    numero1=input('');
12    disp('Ingrese el segundo número:');
13    numero2=input('');
14    resultado=numero1+numero2;
15    disp(['El resultado de la suma de', num2str(numero1), ' y ', num2str(numero2), ' es:', num2str(resultado)]);
16 end
17
18
```

Fuente: Autor



