FUNDAMENTOS DE PROGRAMACIÓN
CON DFD-PSEINT-PYTHON

VANESSA LORENA VALVERDE GONZÁLEZ
JULIO EDUARDO CAJAMARCA VILLA
GABRIEL VINICIO MOREANO SÁNCHEZ

CIDE EDITORIAL

FUNDAMENTOS DE PROGRAMACIÓN

CON DFD-PSEINT-PYTHON

FUNDAMENTOS DE PROGRAMACIÓN

CON DFD-PSEINT-PYTHON

Autores

VANESSA LORENA VALVERDE GONZÁLEZ
JULIO EDUARDO CAJAMARCA VILLA
GABRIEL VINICIO MOREANO SÁNCHEZ

Dirección de Publicaciones

ISBN: 978-9942-636-32-4

https://doi.org/10.33996/cide.ecuador.PS2636324

Filiación:
Vanessa Lorena Valverde González
Julio Eduardo Cajamarca Villa
Gabriel Vinicio Moreano Sánchez
Escuela Superior Politécnica de Chimborazo

Dirección editorial: Lic. Pedro Misacc Naranjo, Msc.
Coordinación técnica: Lic. María J. Delgado
Diseño gráfico: Lic. Danissa Colmenares
Diagramación: Lic. Alba Gil
Fecha de publicación: octubre, 2023

La presente obra fue evaluada por pares académicos experimentados en el área.

Catalogación en la Fuente

Fundamentos de Programación con DFD-PSeInt-Python / Vanessa Lorena Valverde González, Julio Eduardo Cajamarca Villa y Gabriel Vinicio Moreano Sánchez - Ecuador: Editorial CIDE, 2023.

262 p.: incluye tablas, figuras; 17,6 x 25 cm.

ISBN: 978-9942-636-32-4

1. Programación-DFD-PSeInt-Python 2. Programación

Índice

Capítulo 3
Generando Código

Capítulo 4
Utilizando archivos en Python

Prólogo

El mundo de la tecnología sigue en su camino evolutivo sin detenerse, fomentando grandes avances en todos los campos de la ciencia, medicina, administración de negocios, economía, mecánica, entre otros. En gran medida estos avances se deben a la evolución y desarrollo que ha tenido la programación de software y sistemas informáticos.

Es natural que todo aprendizaje empiece desde el inicio para de esta manera fomentar el crecimiento natural del conocimiento; en el área del desarrollo de software pasa lo mismo, y este libro no es la excepción.

Al leer el libro “Fundamentos de programación con DFD-PSeInt-Python” me llamó la atención la forma tan detallada de expresar los temas que son tratados por los autores; cada concepto tiene un ejemplo que invita a tener una idea clara de su funcionamiento y aplicación. La manera como se explica el uso de las variables, ciclos, estructura de datos, algoritmos y archivos, llevan al lector en un camino secuencialmente preparado para aquellos que empiezan en el aprendizaje del desarrollo de software.

Desde el inicio, los autores invitan al lector a conocer aspectos generales de la computación, identificando sus dispositivos, sus funciones y características; luego de forma muy didáctica enseñan a usar y manejar variables, constantes, ciclos de repetición, expresiones matemáticas, operadores lógicos, tipo de datos, vectores y matrices.

Estos conceptos son muy bien compaginados con los algoritmos y diagramas de flujos materializándolos de manera muy pedagógica en ejercicios resueltos que ayudan a entender la lógica de programación como elemento fundamental para el desarrollo de programas en cualquier lenguaje de desarrollo de software. La utilización de herramientas como: DFD y PSeInt, generan una motivación adicional al comprobar el funcionamiento de los algoritmos. Todo esto se complementa cuando llega el momento de aplicar lo aprendido en un lenguaje de programación tan eficiente, robusto y multipropósito como “Python”.

Es conocido que Python es un lenguaje de programación de alto nivel y multipropósito, es uno de los lenguajes de programación más utilizados a nivel mundial y fácil de utilizar; la forma como Vanessa Valverde y Julio Cajamarca abordan cada tema y los implementan en este lenguaje de programación, invita a querer conocer más y seguir escribiendo código. La forma en que los actores implementan los ejercicios luego de cada

concepto nos permite evidenciar su experiencia y habilidad de los más de 10 años de docencia en la rama de la ciencia y tecnología, es admirable la manera de integrar la teoría con la práctica, permitiendo un aprendizaje mucha más duradero.

En fin, este libro está orientado al aprendizaje de los fundamentos de programación para los futuros profesionales que inician en la programación es un texto que pretende incentivar al aprendizaje de la construcción de sistema informáticos estructurados desde el inicio, ayudando a desarrollar su capacidad de pensamiento lógico, elemento indispensable para construir sistemas informáticos completos en cualquier lenguaje de programación.

Gustavo X. Hidalgo Solórzano
Ingeniero en Sistemas Informáticos.
Dirección de Tecnologías de la información,
Comunicación y Procesos.
DTIC-ESPOCH

Introducción

Desde el inicio de la programación los desarrolladores de *software* han buscado formas y métodos para mejorar sus desarrollos de programas; este continuo aprendizaje ha venido evolucionando y llevándolos a niveles realmente altos.

Actualmente existen un sin número de herramientas y lenguajes para el desarrollo de *software*, unas más complejas que otras pero que al final persiguen el mismo objetivo, la construcción de un programa que permita realizar tareas y actividades en menos tiempo que un ser humano.

Actualmente la tecnología ha avanzado a pasos agigantados y todo lo que nos rodea tiene un *software* que permite controlar un dispositivo; en el mundo del automovilismo existen sistemas avanzados que permiten controlar varios segmentos de los vehículos, desde las plumas, los frenos, embragues, entre otros. Estos artefactos electrónicos tienen una mejor eficiencia y rendimiento si están controlados y manejados por un *software* o programa, por lo general están vinculados por medio de un computador central instalado en el mismo automóvil.

La programación es una disciplina de creación de programas, aplicativos o *software*, mediante la escritura de código o también llamada codificación. El programa o *software* es un conjunto de instrucciones codificadas en un lenguaje de programación que le indica a la computadora como resolver un determinado problema o tarea.

La programación de *software* actualmente es utilizada para resolver problemas y tareas en el campo informático, desde sistemas operativos, sitios *web*, video juegos y aplicaciones a medida hasta aplicaciones con inteligencia artificial, es decir, todo lo que el ser humano sea capaz de imaginar, se puede programar.

El aprender a programar no solo es escribir una serie de códigos para realizar un programa en el área que se requiera, sino también entender los términos que se utilizan al momento de desarrollar el *software*.

Un programador o desarrollador de software puede implementar la programación modular, que no es otra cosa que la implementación de código a través de la ejecución de métodos o funciones; este diseño se basa en el algoritmo divide y vencerás que consiste en que el problema original subdividirlo en subproblemas, por ejemplo si se desea identificar si el número

ingresado por el usuario es par o impar en vez de generar todo el código de forma secuencial, se generará a través de funciones; para este caso será una función a la que se le invocará cada vez que se solicite entonces, el problema se lo ha subdividido en otro algoritmo más pequeño que se encarga de una ejecución específica.

Es por tal motivo que en el Capítulo 1, se presenta un conjunto de generalidades y conceptos básicos comunes en la programación con ejemplos simples que pretende inducir al lector en el entorno conceptual del mundo de la programación de computadores; estos conceptos van a ser de mucha ayuda no solo para las herramientas que se usarán en este libro, sino que sirven para cualquier otra herramienta o lenguaje de programación que se esté usando en el área de computación.

El Capítulo 2 aborda el diseño y análisis de algoritmos que ayuda al lector a aclarar e interpretar ideas y da solución a un problema de una manera sistemática y lógica, además, permitirá una comprensión de cómo funciona la programación y ejecución de sentencias e instrucciones en un computador y de esta manera crear nuevas y creativas soluciones para problemas específicos. Estas representaciones que no son más que formas organizadas y ordenadas de diagramas para indicar al compilador de la herramienta o estructura programática, interprete y traduzca a un lenguaje de bajo nivel que entiende el

computador, de esa manera podrá ejecutar las tareas y actividades que describe el diagrama. En este mismo capítulo se podrá aprender y utilizar seudocódigos para implementar ejercicios más complejos.

Por otra parte, el Capítulo 3 afronta la temática de un lenguaje de programación de alto nivel como es Python; mediante ejercicios, tips y conceptos se tocará temas de variables, secuencias, condicionales, ciclos repetitivos o bucles, arreglos, lista y archivos.

Por último, el Capítulo 4 expondrá de manera práctica y conceptual, el uso y manipulación de un repositorio de datos permanentes como es el caso de “archivos”, manejados desde la perspectiva de Python ayudará a entender cómo un archivo se puede transformar en una herramienta poderosa a la hora de fabricar *software* o programas eficientes y dinámicos.

Cada capítulo es un avance en la lógica de programación, es por eso que el libro está diseñado de tal forma que el lector se involucre en una lectura secuencial, pero es también una ayuda para programadores un poco más expertos que desean introducirse a la programación en Python.

En los últimos años, la tecnología nos demostró lo importante que es en la vida cotidiana de los seres humanos, cuando azotó la pandemia del COVID 19 donde las personas se vieron en la necesidad de aislarse en sus domicilios por disposición estatal o personal; fue por medio de la tecnología que los individuos pudieron seguir trabajando, estudiando e incluso invirtiendo y negociando a alto nivel. Todo esto pudo ser posible porque alguien en un momento dado tuvo la decisión de escribir un programa que resolviera un problema o una tarea específica. Cómo lo dijo el fundador de Apple Sr. Steve Jobs, *"Las cosas no tienen que cambiar al mundo para ser importante"*.

CAPÍTULO 1

APRENDER A PROGRAMAR

Capítulo 1

Aprender a Programar

1. Conceptos básicos

El objetivo de este Capítulo es proporcionar al lector un medio que le permita conocer términos, conceptos, definiciones y elementos propios de la programación, declaraciones, representaciones y comprensión de los diferentes tipos de datos.

En el mundo de la programación de *software*, programas o sistema informáticos, existe un término que no se puede pasar por alto, este es conocido como “Lógica de programación” [LG]. La LG es un término que se utiliza a menudo para identificar la visualización de la solución que el programador le da a un determinado problema o tarea que desea resolver, es decir, es plasmar de forma escrita para solucionar el problema.

El dominar este conjunto de ideas le permite al programador expandir las posibilidades de resolver cuanto problema se le presente y de esa manera podrá plasmarlas en cualquier lenguaje de programación de su preferencia.

¿Por qué la lógica? Bueno, así como para aprender matemáticas se empieza aprendiendo a sumar y restar antes de aprender a integrar o derivar; en la programación se debe aprender, cómo realizar ejercicios de razonamiento lógico y resolución de problemas basados en ideas, luego se deberá plasmar esas ideas en un algoritmo o un diagrama de flujo hasta llegar a construir todo el sistema informatizado completo utilizando variables, ciclos, secuencias, condicionales, arreglos (vectores y matrices), librerías o bibliotecas.

Para el aprendizaje de la programación computacional estructurada, este libro contará con herramientas básicas como: los programas Diagrama de Flujo de Datos (DFD) que permite realizar algoritmos en forma gráfica, además de ejecutarlos y depurarlos, PSeudocódigo Intérprete (PSeInt) para el desarrollo de seudocódigo y como lenguaje de programación de alto nivel el *software* Python, así mismo como Entorno de Desarrollo Integrado (IDE) se utilizará el Visual Studio Code; todos los programas son de acceso gratuito y no requiere un pago económico para su uso o distribución.

1.1. Informática y Programación

La informática es una ciencia o disciplina que abarca un conjunto de conocimientos, métodos y técnicas que se utilizan para el procesamiento, manejo y gestión de información en los

equipos computacionales con el propósito de resolver problemas que en la vida cotidiana llevaría más tiempo y esfuerzo resolver.

La informática abarca una diversidad de campos para su correcta consolidación, como son:

a) ***Hardware***: son todos los componentes físicos que forman parte de un sistema, en el mundo de la informática existen una gran variedad de *hardware* que se utilizan para integrar un sistema informático. En el área computacional el *hardware* sería el ordenador o computadora, este equipo representa un elemento importante y fundamental para la mayoría de usuarios de los sistemas. Por otra parte, cuando nos referimos a telecomunicaciones el *hardware* sería los aparatos de red como el *switch*, *router*, cables de red, entre otros. El conjunto de estos elementos *hardware* son a los que se le denomina sistema informático.
b) ***Software***: es todo lo relacionado con la parte no tangible o que no se puede tocar del sistema informático, por lo general son: el sistema operativo, programas, aplicativos, servicios *web*, entre otros. Estos componentes intangibles son los que instalados sobre el *hardware* hacen funcionar el sistema. Todo *hardware* requiere de un *software* para funcionar, convirtiéndose en el elemento fundamental para la funcionalidad de un sistema informático.

c) **Algoritmos**: son un conjunto de instrucciones y estructuras ordenas que se utilizan para ejecutar alguna tarea o actividad que conlleve a resolver un problema o procedimiento. Los algoritmos están catalogados como una agenda programada que tiene que ejecutarse paso a paso para llegar a resolver el objetivo planteado. En la actualidad existen varias formas de representar los algoritmos entre ellos tenemos, los *seudocódigos* y los diagramas de flujos.

d) **Lenguajes de programación**: son escrituras basadas en palabras, signos, códigos y estructuras que sirven para escribir una serie de algoritmos que resuelven un problema, estas escrituras son el lenguaje de comunicación entre el programador y el computador, ya que es la forma como el computador puede entender las diversas instrucciones que el ser humano requiere que resuelva o ejecute.

e) **Repositorios de almacenamiento**: son estructuras que permiten almacenar o guardar datos o información de manera permanente. En la actualidad existen varios repositorios de almacenamiento, entre ellos tenemos los archivos y las bases de datos. Los archivos son repositorios planos que permiten almacenar información de manera secuencial o por línea, generalmente los archivos son almacenados permanentemente en el disco duro hasta que el usuario decida eliminarlo. Los archivos

son utilizados también como archivos ejecutables, esto permite desencadenar un conjunto de acciones que pretendan ejecutar instrucciones.

f) **Redes de comunicaciones**: es un conjunto de *hardware* que se utilizan para interconectar a dos o más computadores o dispositivos computacionales (impresoras, escáner, tablet, teléfonos, televisores). En el mundo de las telecomunicaciones existen varias redes de comunicación: red alámbrica (aquellas que necesitan de cables físicos conectados a los componentes de red para comunicarse entre computadores y/o equipos computacionales) e inalámbrica (aquellos que no requieren cables para comunicarse entre computadores y/o equipos computacionales).

g) **Inteligencia Artificial [IA]**: es una tendencia que en los últimos años ha evolucionado a pasos agigantados, al punto que actualmente existen muchas aplicaciones y tecnología que permiten aplicar de manera eficiente los aplicativos de inteligencia artificial. La IA es una rama de la programación que permite diseñar y crear algoritmos para sistemas informáticos o *software*, que conceden a las máquinas o equipos aprender a medida que los utilizan con la finalidad de realizar tareas que usualmente las ejecuta la inteligencia humana.

La programación es un concepto que está muy ligado con la informática, ya que es parte fundamental de la existencia de esta. La programación es una actividad que lleva a cabo aquellas personas que se dedican a escribir algoritmos en un determinado lenguaje de programación o herramienta programática. Antes de la computadora, existió la programación; es el corazón del sistema informático ya que es el lenguaje de comunicación que requiere el *hardware* para comunicarse y poder entregar un resultado.

h) **Desarrollo de *software***: son las actividades de diseño, análisis, implementación, pruebas y mantenimiento de un proyecto software o sistema informático, los desarrolladores de software son personas que se dedican a ejecutar las actividades antes mencionadas y que por lo general se les llaman “programadores” o “desarrolladores”.
i) **Estructura de datos**: son espacios de memoria que se utilizan para almacenar datos de manera temporal, permiten tener acceso y manipulación de la información.
j) **Depuradores**: son programas o aplicativos que contienen los lenguajes de programación para verificar y validar los errores de sintaxis que puedan tener un código determinado que se va a ejecutar.

La programación es una disciplina muy importante en la evolución de la informática, ya que gracias a ella se ha podido

avanzar en temas como: inteligencia artificial, análisis de datos, videojuegos, automatización de procesos, entre otros. Los programadores constantemente hacen esfuerzos para mejorar sus algoritmos para hacerlos más eficientes y versátiles.

1.2. La computadora

Es un equipo electrónico con la capacidad de procesar grandes cantidades de información mediante procesos de automatización eficiente de instrucciones contendidas en los programas y/o *software* que este ejecuta.

Las computadoras están compuestas por un conjunto de dispositivos o periféricos que permiten establecer una conexión eficiente y estable, entre ellos tenemos:

a) **Unidad Central de Procesamiento**: es considerado como la parte central o cerebro del computador, es el dispositivo encargado de realizar todo tipo de cálculo y/o procesamiento de la información que ha sido enviada por el usuario. Es aquel dispositivo que ejecuta paso a paso las instrucciones que le han sido encomendadas por medio de un programa, software o sistema informático, además permite gestionar de manera eficiente los recursos u otros dispositivos del computador.

b) **Memoria**: es un dispositivo de almacenamiento temporal (está almacenado siempre que el programa, *software* este en ejecución). En un computador existen dos tipos de memoria; la memoria *RAM* (o Memoria de Acceso Rápido) y la Memoria *ROM* (Memoria de Sólo Lectura).

   La memoria *RAM* es un tipo de almacenamiento temporal, la característica principal de este tipo de memoria es la rapidez para almacenar los datos, información e instrucciones de los usuarios o que necesita el computador. Una de las desventajas es sin lugar a duda el acceso aleatorio, esto quiere decir que este tipo de memoria almacena la información aleatoriamente en cualquier dirección o ubicación, provocando desorden y espacios sin usar.

c) **Dispositivos de Entrada/Salida**: son dispositivos electrónicos o periféricos destinados para el ingreso o salida de datos e información por parte del usuario u otra máquina. Los dispositivos de entrada son teclados, *mouse*, micrófono, cámaras; mientras que los de salida son: monitor, impresora, parlantes. Es importante conocer que estos dispositivos deben de estar permanentemente conectados a la computadora ya sea por cables o bluetooth.

d) **Disco duro**: son dispositivos de almacenamiento permanente, actualmente se manejan discos de estado solidos [SSD] que son más rápidos y eficientes para el manejo y almacenamiento de datos. La información contenida en estos dispositivos sólo se puede eliminar si el usuario así lo decide.

e) **Tarjeta madre o placa madre**: es una placa electrónica donde se conectan todos los componentes o dispositivos de un computador, esta tarjeta es el puente o canal por donde viajan los datos, información o instrucciones convertidos en *bits* y que son utilizados para manejar esos dispositivos.

f) **Tarjeta gráfica**: es otro tipo de placa electrónica, pero está destinada al procesamiento de gráficos e imágenes en pantalla; por cada dispositivo de salida gráfica como el monitor, se requiere contar con una tarjeta gráfica.

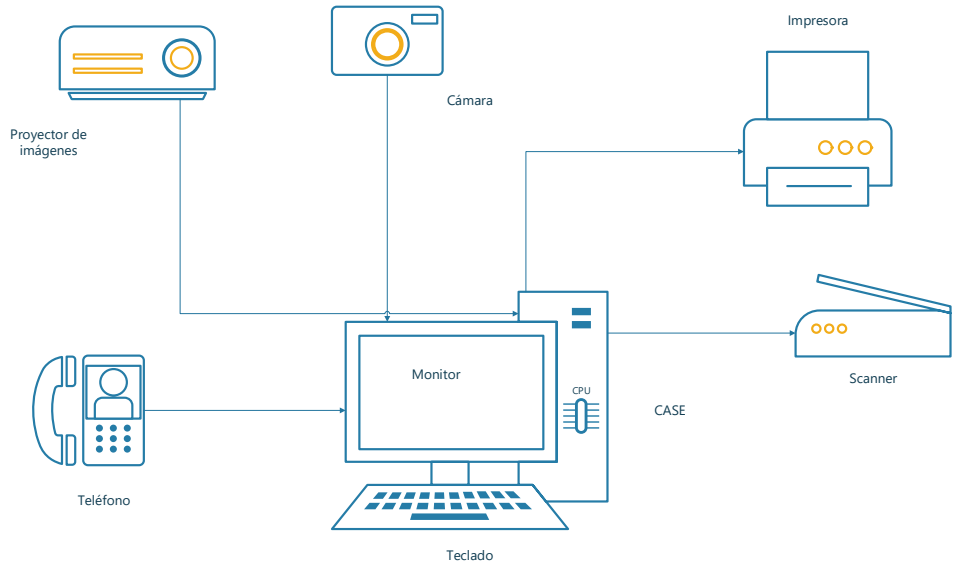
g) **Fuente de alimentación o fuente de poder**: es el dispositivo encargado de brindar la alimentación eléctrica necesaria para el funcionamiento de todos los dispositivos o periféricos del computador.

Las computadoras están diseñadas para realizar múltiples tareas y/ o actividades, desde cálculos matemáticos complejos pasando por reproducción de imágenes y gráficos hasta llegar a navegar por internet, ejecutar videojuegos, simulaciones y muchas funciones más.

Existen vatios tipos de computadoras destinadas a usos en diferentes ámbitos, las computadoras personales son las más conocidas y utilizadas a nivel mundial, son fabricadas para el ámbito empresarial y de oficina es decir para usuarios comunes; otro tipo de computadoras son los Servidores o Súper computadoras, son equipos destinados a trabajos forzados y robustos, están constantemente trabajando y brindan servicios a las computadoras personales y portátiles. El conjunto de súper computadoras son el elemento principal del internet. Las computadoras portátiles o laptop son computadoras personales, pero con la diferencia que pueden ser movilizadas a cualquier parte sin problema que se apaguen o se dañen.

Figura 1.1.

Periféricos del computador



Nota. Los autores, Microsoft Visio

La información es ingresada a la computadora por medio de los dispositivos de entrada, como son: teclado, cámara, *mouse*, micrófono. Luego el programa o sistema operativo que está en ejecución toma esta información y la envía la Unidad Central de Procesamiento [CPU] donde se realizará el cálculo y/o procesamiento de esa información, por último, la Unidad Central de Proceso (CPU) mostrará los resultados por medio de los dispositivos de salida como: monitor, impresora, parlantes.

1.3. Variable

Según Vegas (2021) “La variable es la unidad básica de almacenamiento de un programa” (p. 34).

Una variable es un espacio de memoria en la que se puede almacenar un valor y este puede ser utilizado, eliminado o modificado por el usuario en cualquier momento durante la ejecución del programa, software o sistema informático.

La variable es un elemento que nunca debe faltar en la codificación de un programa, ya que gracias a esta se puede, almacenar valores, realizar operaciones, representar estados o condiciones dentro del programa y reutilización de valores.

En un lenguaje común, imaginemos que tenemos una caja de cartón, de repente el dueño de esta caja quiere colocar dentro de ella una pelota (impresa el emogi de carita feliz), el resultado de esta acción es una caja que en su interior contiene una pelota; de la misma manera si el dueño de la caja desea quitar la pelota que estaba en la caja, esta quedará vacía. Ahora tomando la referencia anterior, la caja vendría a ser una variable y la pelota el valor.

Figura 1.2.

Ejemplo de variable

Nota. Los autores, Microsoft Word

En la caja se podrá almacenar un solo valor (Figura 1.2), para este caso solo se podrá almacenar una pelota. Si se requiere la pelota que se almacenó en la caja, sólo tiene que sacar de la caja y se podrá utilizar para jugar con ella, sin embargo, según Villalba et al., (2021) “una variable declarada en un programa es una abstracción de una celda o colección de celdas de la memoria del ordenador” (p. 94). Esto significa que cuando se declara una variable, el compilador del lenguaje de programación internamente crea un espacio en memoria y colocará el valor que específico, dicho valor podrá ser usado por el usuario en el momento que requiera.

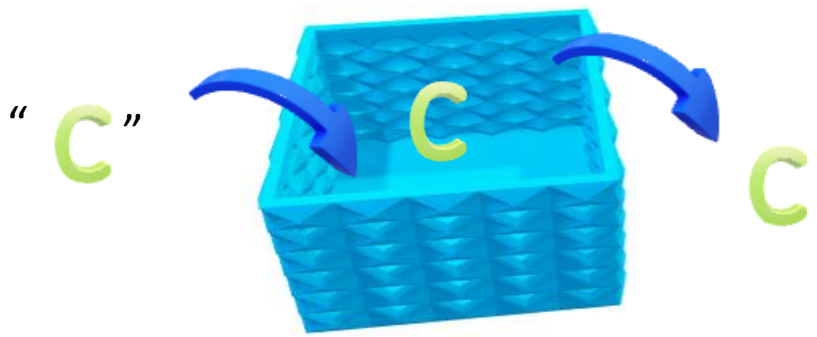
1.4. Tipos de variables

a) Variable tipo carácter

Las variables de datos tipo carácter, son aquellos valores que almacenan como letras o caracteres especiales.

Figura1.3.

Variable tipo carácter



Nota. Los autores, Microsoft Word

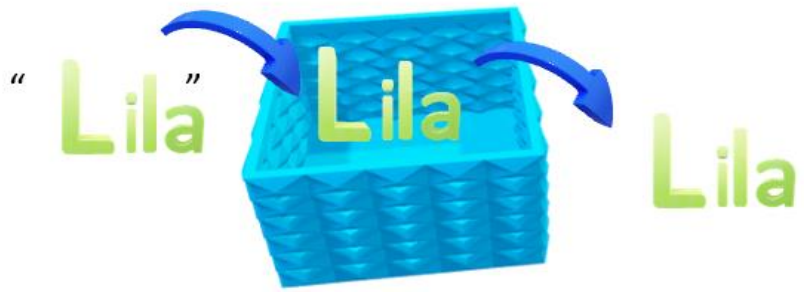
La ilustración anterior, representa el manejo de una variable tipo carácter, para indicar que ese tipo carácter se debe colocar entre comillas, esta acción provoca que el compilador del programa al momento de leer la variable, inmediatamente sabrá que se trata de un carácter.

b) Variable tipo cadena de caracteres

Es conocida como cadena o *string* y es una colección o conjunto de caracteres que forman una palabra, colección de letras o caracteres especiales. Estas cadenas deben de estar entre comillas simples o dobles para que el compilador sepa a ciencia cierta que se trata de ese tipo específico.

Figura 1.4.

Tipo cadena de caracteres



Nota. Los autores, Microsoft Word

En la Figura 1.4, se puede observar que se maneja en la caja una cadena de caracteres, la misma caja puede utilizarse para sobre escribir varias cadenas.

Figura 1.5.

Variable cadena “Lila”

Nota. Los autores, Microsoft Word

Figura 1.6.

Variable cadena “Sol” sobre escribe variable cadena “Lila”

Nota. Los autores, Microsoft Word

En la Figura 1.5, se almacena la cadena con el valor de “Lila” y luego por cuestiones de prueba se cambia el valor de la cadena a “Sol” como lo muestra la Figura 1.6, esta acción

provoca que la cadena con el valor “Lila” se elimine de la caja y en su lugar se coloque el valor “Sol”, es decir, siempre va a quedarse almacenado el último valor ingresado; pues como ya se había mencionado, en una variable sólo se podrá almacenar un valor del tipo específico, es decir, si se preparó la caja para almacenar cadena de caracteres, no podrá almacenar números; si requiere almacenar números tendrá que crear una nueva caja que le sirva para almacenar números.

c) Variable tipo número entero

Son variables con valores numéricos enteros de la familia de los naturales estas variables.

Figura 1.7.

Variable tipo número entero

Nota. Los autores, Microsoft Word

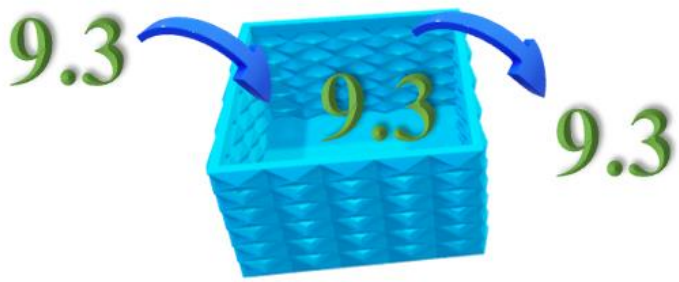
Para el caso de la Figura 1.7, la variable está almacenando un número, por lo que para este tipo de dato no se emplean las comillas dobles.

d) Variable tipo número decimal

Son variables con valores numéricos decimales de la familia de los reales; estas variables están dispuestas a manejar datos o valores con números con decimales.

Figura 1.8.

Variable tipo número decimal



Nota. Los autores, Microsoft Word

Las variables son importantes e indispensable en un programa, en vista de su importancia se requiere de una forma

que permita identificarla dentro del programa o *software* y así podrá manipularla sin correr el riesgo de equivocarse o incurrir en errores de implementación. Para identificar o nombrar una variable utilice una letra o palabra única, es decir, que no se repita, luego se asignará un valor correspondiente que puede ser numérico, carácter o booleano.

En la Tabla 1.1, se presenta un conjunto de ejemplos que representan la identificación y asignación de variables.

Tabla 1.1.

Ejemplos de variables

Expresión	Explicación
A=3	Variable que se le asigna un valor entero.
peso =453,5	Variable que se le asigna un valor decimal.
_Letra =”G”	Variable que se le asigna un carácter.
Mensaje = “expresa un valor”	Variable que se le asigna un texto o cadena de caracteres.

Nota. Los autores

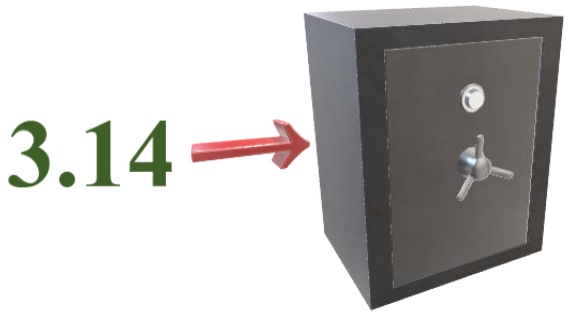
1.5. Constante

Se conoce como constante a la expresión de una letra o palabra común que se le asigna un valor, que puede ser

numérico o carácter; pero que en el transcurso del programa no cambia, es decir, desde el momento que se le asigna un valor este no se modifica durante toda la ejecución del programa o *software*.

Figura 1.9.

Ejemplo de una constante



Nota. Los autores, Microsoft Word

La Figura 1.9 representa gráficamente a una constante con una caja fuerte, pues durante toda la ejecución del programa permanecerá con el valor que se le asignó.

Según Ayala (2020) "… las variables y las constantes nos sirven para representar un valor de datos de entrada o de información. Dicho valor estará almacenado en la memoria del dispositivo" (p. 32).

Tabla 1.2.

Ejemplos de constantes

Expresión	Explicación
A=5	Constante que se le asigna un valor entero.
PI = 3,1416	Constante que se le asigna un valor decimal.
B=”F”	Constante que se le asigna un carácter.
_MENSAJE= “expresa un valor”	Constante que se le asigna un texto.

Nota. Los autores

1.6. Declaración

El término se utiliza para crear una variable o constante antes de su uso; el declarar significa reservar un espacio de memoria e identificar el tipo de dato que se le va a asignar, o lo que es lo mismo, qué tipo de valores puede recibir. En un lenguaje común es como indicar el idioma que va a hablar.

Algunos ejemplos de declaraciones:

1. ***A*** es de tipo entero.
2. ***PI*** es de tipo real.
3. ***B*** es de tipo carácter.
4. ***_MENSAJE*** es de tipo texto

Tipo de declaraciones

a) Declaraciones de tipo entero

suma es de tipo entero.
*identificador*es de tipo entero.
calculadora es de tipo entero.
número es de tipo entero.
n es de tipo entero.
num es de tipo entero.
jose de tipo entero.

En los ejemplos citados las variables están de color azul, y su declaración rige el tipo de datos que cada una puede recibir. Es importante que al identificar o dar nombre a la variable, esta lleve una concordancia con su objetivo. Pese a que esta recomendación no es una regla, se puede asumir como una sugerencia ya que es importante saber manejar las variables para evitar posibles errores a futuro.

Una prueba de ello es que la variable *Juan* pese a ser de tipo entero, no tiene concordancia con ese tipo de datos, pero se trae a colación como un ejemplo que se puede utilizar para manejar ese tipo de datos.

b) Declaraciones de tipo decimal

a es de tipo decimal
edad es de tipo decimal
estatura es de tipo decimal
peso es de tipo decimal
temperatura es de tipo decimal
precio es de tipo decimal
kilometros de tipo decimal

c) Declaraciones de tipo carácter

a es de tipo carácter
*caracter1*es de tipo carácter
texto es de tipo carácter
c es de tipo carácter
cont es de tipo carácter
V es de tipo carácter
inicial de tipo carácter

d) Declaraciones de tipo texto

vector es de tipo texto
cadena es de tipo texto
sentencia es de tipo texto
cosa es de tipo texto
nombre es de tipo texto
apellido es de tipo texto
ciudad de tipo texto

Tabla 1.3.

Nombre de una variable

Formas	Explicación
A	Una letra.
Abc	Combinación de letras.
Numero	Un texto específico, siempre y cuando no sea una palabra reservada.
Num	Abreviatura de una palabra.
D234c	Combinación de letras y números.
_nombre23	Una variable puede empezar con _ pero no con un número.

Nota. Se recomienda que al momento de generar una variable esta sea con letras minúsculas y no contenga signos especiales.

1.7. Condiciones

Una condición es una expresión lógica que evalúa si una determinada afirmación es verdadera o falsa; este tipo de condiciones permite al programa decidir qué ruta tomar en una situación donde exista varios caminos.

Tabla 1.4.

Ejemplos de condiciones lógicas

Expresión	Explicación
A es igual a 5	Si la variable *A* es igual al valor de 5
constante= 4,5	Si la variable *constante* es igual al valor de 4,5
B=”F”	Si la variable *B* es igual a la tecla presionada por el usuario “F“

Nota. Los autores

1.8. Ciclos

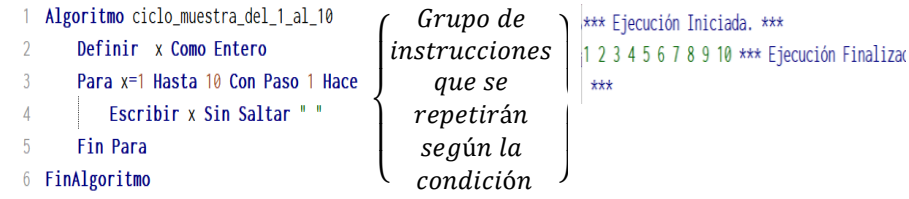
Un ciclo también es conocido como un bucle en el lenguaje del programador. Son estructuras o segmento de código que permiten repetir un cierto bloque de sentencias o instrucciones varias veces. Trejos y Muñoz (2021) indica que, “una sentencia repetitiva es una instrucción que permite que un conjunto de instrucciones se repita una cantidad de veces determinada” (p. 38).

Ejemplo:

Ciclo con condición lógica: la condición lógica es la expresión de comparación que indica el fin del ciclo, en este caso del for.

Figura 1.10.

Código ciclo en Python



a. *b.* *c.*

Nota. Python

1.9. Expresiones matemáticas

Existen ocasiones en las que el programa exige un conjunto de operaciones o cálculos matemáticos que se debe resolver a través de una fórmula o ecuación, las herramientas de programación cuentan con un conjunto de librerías o bibliotecas que permiten la utilización de signos o caracteres especiales para su ejecución, por ejemplo:

$$E = \frac{\sqrt{a^2 + b^2} + e^{-i\omega t}}{a}$$

Ecuación 1.1. Ejemplo de una expresión aritmética

La Ecuación 1.1. expresada en término legible para el computador sería:

E=(((a^2+b^2)^1/2)+e^(-i*ω*t))/a

Ecuación 1.2. Expresión aritmética representado en lenguaje de programación.

Tabla 1.5.

Representación de símbolos

Símbolo	Nombre	Expresión
+	*Suma*	+
-	*Resta*	-
x	*multiplicación*	*
÷	*división*	/
x^2	*exponente*	X^2

Símbolo	Nombre	Expresión
$\sqrt{x}$	*Raíz*	*X^1/2* *O según el lenguaje de programación*
( )	*paréntesis*	*()*
[ ]	*corchetes*	*[ ]*
{ }	*Llaves*	*{ }*

Nota. Los autores

Más ejemplos de ecuaciones:

1)

$$f = \frac{x^2 + 3x - 2}{4x + 7}$$

$$f = (x\text{^}2 + 3 * x - 2)/(4 * x + 7)$$

2)

$$f = \frac{\frac{x^3}{3} + x^2 + \sqrt{\frac{x}{2} + 1}}{x + \frac{x}{5}}$$

$$f = ((x\text{^}3)/3) + (x\text{^}2) + (x/2 + 1)\text{^}(1/2))/(x + (x/5))$$

3)

$$f = \sqrt{\sqrt{\sqrt{a^2 + b^2}}}$$

$$f = (((a\text{^}2 + b\text{^}2)\text{^}1/2)\text{^}1/2)\text{^}(1/2)$$

4)

$$f = x^5 + 6\sqrt{x + t + \frac{z}{7}}$$

$$f = x^5 + 6 * (x + t + z/7)\text{^}1/2$$

1.10. Operadores lógicos

Son símbolos utilizados para evaluar o consultar expresiones lógicas y obtener un resultado que por lo general es de tipo booleano (verdadero o falso), son muy utilizado en condiciones o ciclos ya que permiten realizar operaciones de evaluación lógica entre dos o más sentencias o condiciones.

Los operadores lógicos pueden formar combinaciones de expresiones más complejas, los operadores utilizados en programación son:

Tabla 1.6.

Operadores lógicos

Símbolo	Nombre	Significado
&	AND	y
||	OR	o
!	NOT	no

Nota. Los autores

Los operadores lógicos son un conjunto de operadores que se utilizan para expresiones de condición entre uno o más valores.

Algunos ejemplos de su uso:

1. *variable1*&*variable2*, en este caso la *variable1* y la *variable2* deben ser verdaderos para que dé verdadero.
2. *variable1* || *variable2*, en este caso cualquiera de las dos variables debe ser verdadero para cumplir la condición de verdadero.
3. ***!****variable1*, si la variable1 es verdadera al anteponer el signo de exclamación“**!**” dará como resultado falso.

Para poder trabajar con los operadores lógicos se debe tomar en cuenta las tablas de verdad:

Tabla 1.7.

Operador and

AND		
variable1	variable2	variable1 **and** variable 2
V	V	V
V	F	F
F	V	F
F	F	F

Nota. Los autores

Tabla 1.8.

Operador or

OR		
variable1	variable2	variable1 **or** variable 2
V	V	V
V	F	V
F	V	V
F	F	F

Nota. Los autores

Tabla 1.9

Operador not

NOT	
variable1	not variable 1
V	F
F	V

Nota. Los autores

Para realizar expresiones de consultas o condición se debe proceder a realizar esas condiciones con los operadores de relación y si son más de una las condiciones, unir las expresiones con los operadores lógicos.

1.11. Operadores de relación

Son semejantes a los operadores lógicos ya que ambos como resultado entregan un valor booleano, la diferencia radica en que los operadores de relación sirven para comparar una o más expresiones o valores según la relación que existan entre ellos. En un programa, estos operadores son fundamentales para la toma de decisiones y garantizar el correcto funcionamiento del programa.

Tabla 1.10.

Operadores de relación

Símbolo	Nombre
>	Mayor que
<	Menor que
>=	Mayor igual
<=	Menor igual
(=) o (==)	Igual
(! =) o (<>)	Diferente

Nota. Los autores

Ejemplos de uso:

a) *748>546*, en el ejemplo se aprecia claramente que la cantidad 748 es mayor que 546.
b) *4<5*, en el ejemplo se aprecia claramente que 4 es menor que 5.

c) *567>=566*, en el ejemplo para una condición lo que se evalúa en este caso es la condición de mayor: la cantidad 567 es mayor que 566 lo cual dará como evaluación de la condición verdadero.
d) *567<=567*, en el ejemplo para una condición lo que se evalúa en este caso es la condición de igualdad: la cantidad 567 es igual que 567 lo cual dará como evaluación de la condición verdadero.
e) *678 !=12*, en el ejemplo se está evaluando que la cantidad 678 es diferente a la cantidad de 12.
f) *678<> 23*, en el ejemplo se está evaluando que la cantidad 678 es diferente a la cantidad de 23.

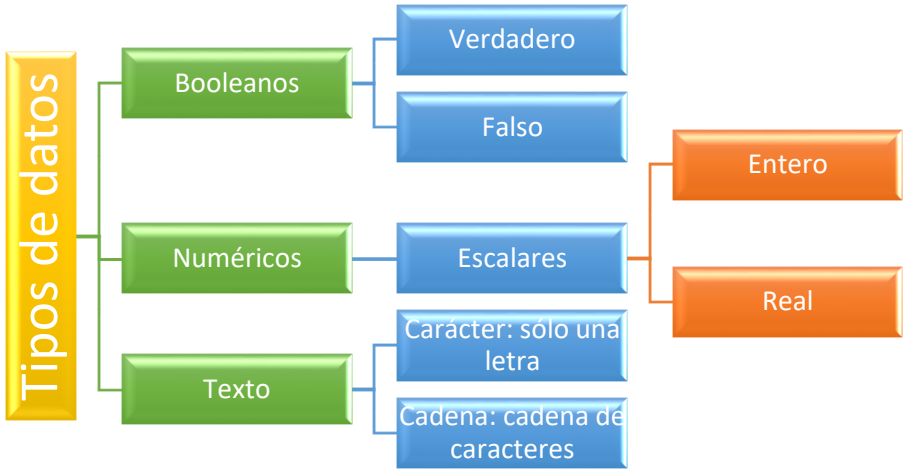
1.12. Estructura de los tipos de datos

Se utiliza al momento de definir o crear una variable o constante, es indicar qué tipo de valores recibirá, los cuales pueden ser: número, texto, carácter, entero, decimal, lógico (verdadero o falso), entre otros.

Para el caso de las aplicaciones que se utilizarán, los tipos de datos que utilizan los editores de PSeInt y DFD son los siguientes:

Figura 1.11.

Tipos de datos



Nota. Los autores

Ejemplos de los tipos de datos:

1. Booleano

   Asignar a variable como dato la estrella amarilla.

   variable:

   Se procede a realizar la consulta:

   Si la variable **=**

   Si se cumple dará como respuesta verdadera, caso contrario es falso.

2. Numérico entero

   variable: entero

   Como se ha declarado a la variable de tipo entero entonces los valores que podrá recibir son:

   variable= 123

   variable=345

   variable=6

   Si se le asigna a la variable de tipo entera un valor decimal, el sistema procederá a seleccionar sólo la parte entera del valor asignado.

3. Numérico real

   variable: real

   Como se ha declarado a la variable de tipo real entonces los valores que podrá recibir son:

   variable= 1,23

   variable=34,5

   variable=6,0

4. Texto carácter

   variable: carácter

   Como se ha declarado a la variable de tipo carácter entonces los valores que podrá recibir son:

   variable= “a”

   variable=”c”

   variable=”&”

5. Texto cadena

   variable: cadena

   Como se ha declarado a la variable de tipo cadena entonces los valores que podrá recibir son:

   variable= “María”

   variable=”José Veloz”

   variable=”345”

En el caso de los tipos de datos de texto lo que desee asignar se deberá colocar entre comillas.

Más ejemplos de tipos de datos:

a) Booleano

   V: Lógico

   V= Verdadero

   F: Lógico

   F=Falso

   Caminar: Lógico

   Caminar=Verdadero

   Caminar: Lógico

   Caminar=Falso

b) Entero:

Edad: Entero

Edad= 27

Número_Habitaciones: Entero

Número_Habitaciones= 36

Código: Entero

Código= 12689

Página: Entero

Página= 7

c) Real

Precio: Real

Precio=12,49

Distancia: Real

Distancia=2,5

Altura: Real

Altura= 26,78

Gramos: Real

Gramos= 1,2

d) Carácter

N: Carácter
N= ‘V’

espacio: Carácter
espacio=‘ ‘

N: Carácter
N=‘1’

guion: Carácter
guion=‘_’

e) Cadena

Nombre: Texto
Nombre= “Vanessa”

Apellido: Texto
Apellido= “Valverde”

Ciudad: Texto
Ciudad= “Riobamba”

Teléfono: Texto
Teléfono= “057234567”

1.13. Vector

Estructura lineal en la que se almacena un número finito de datos del mismo tipo. Por tanto, una variable declarada como un vector es una agrupación ordenada, limitada y homogénea de datos del mismo tipo. En memoria se almacenan los datos en posiciones consecutivas (Algar & Sevilla, 2019, p. 209).

*"*Los vectores o *arrays* son variables estructuradas como cadena de caracteres, donde cada elemento se almacena de forma consecutiva en un número de notaciones.*"* (Delgado, 2022, párr. 1)

*"*Un arreglo es un conjunto de elementos dispuestos uno a continuación de otro, donde cada elemento conserva su propio espacio, tamaño en bytes*"*. (Toro, 2020, párr. 4)

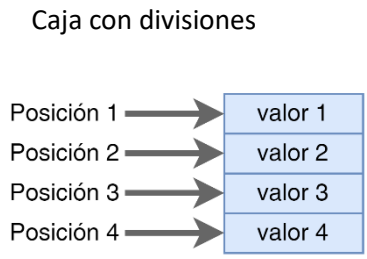
Un arreglo o vector es un conjunto o grupo de información almacenada en una variable, de manera secuencial y organizada por medio de índices llamados "posiciones". La información contenida en estas casillas debe ser del mismo tipo de dato. Es decir, tiene la misma función de la variable, pero, con la diferencia que puede almacenar varios valores bajo el mismo nombre.

Si se toma el ejemplo de la caja como variable vector, esta deberá someterse a una modificación, es decir, la misma caja debe de dividirse en secciones y cada sección debe contar con

una posición o índice, esto permitirá identificar el valor según su posición, por ejemplo:

Figura 1.12.

Representación de un vector



Nota. Los autores

En la Figura 1.12, se representa un vector en el cual se observa que se subdivide en posiciones, en cada posición del vector se almacenará un valor, ya se sea un número, carácter, cadena de caracteres, todo dependerá del tipo de dato que se le haya asignado en el momento de su declaración.

Sí un vector fue asignado como número, entonces sólo almacenará números.

El arreglo o vector ***nombre*** almacena caracteres de tipo char, es decir si el usuario ingresa ***Lorena*** en el arreglo o vector almacenaría letra por letra como se muestra a continuación:

Figura 1.13.

Vector de caracteres

nombre

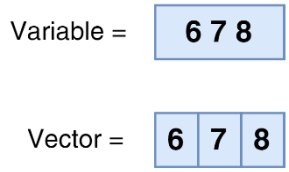
L	o	r	e	n	a
Posición 1	Posición 2	Posición 3	Posición 4	Posición 5	Posición 6

Nota. Los autores

La diferencia entre un arreglo y una variable radica en que la variable almacena un valor mientras que en el arreglo almacena varios valores. Ejemplo 1:

Figura 1.14.

Variable y vector de tipo número



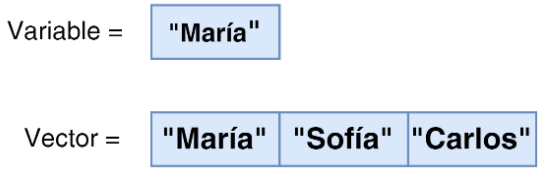
Nota. Los autores

En la Figura 1.14, se observa que en la variable sólo se puede almacenar una cifra, mientras que en el vector es como si

tuviera tres variables de tipo número en uno sólo, donde se ha procedido por un método a separar los números de la cifra, para almacenar en forma individual en cada casilla o posición; si el ejemplo del vector se quisiera realizar con variables, se tendría que definir 3 variables de tipo número.

Figura 1.15.

Variable y vector de tipo cadena.



Nota. Los autores

En la Figura 1.15, se aprecia una variable de tipo cadena que almacena un solo nombre “María”, mientras que en el vector se observa que es de tipo cadena y puede almacenar tres nombres “María”, “Sofía”, “Carlos”.

1.14. Matriz

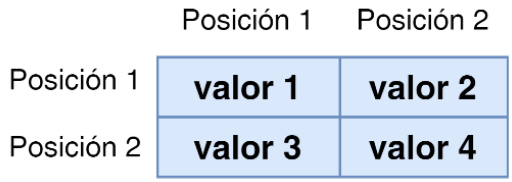
Es una estructura de datos parecida al vector, con la diferencia que esta es una colección de valores o datos

dispuestos en filas y columnas, es decir, es un arreglo en dos dimisiones. Son muy utilizadas para representación de datos tabulares como tablas, mapas e imágenes; así como también para realizar cálculos matemáticos, entre otros.

El concepto de matriz es el mismo que se emplea en las matemáticas como es el caso del Álgebra Lineal.

Figura 1.16.

Representación de una matriz

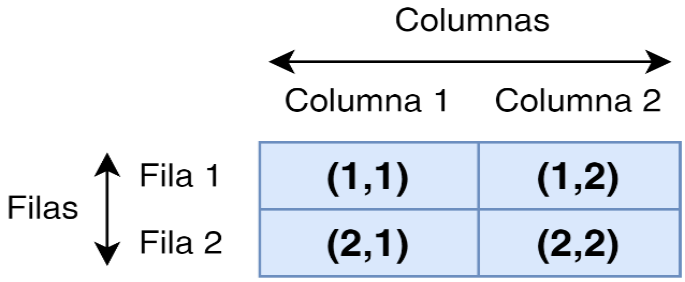


Nota. Los autores

En la Figura 1.16, se muestra un ejemplo de matriz donde a diferencia del vector, este contiene posiciones tanto en las filas como en las columnas; utilizando la definición del álgebra lineal se observa que se trabaja con filas y columnas donde el valor 1 está en la posición (1 fila, 1 columna); de igual manera, si la matriz se le declara como número, todos los elementos de la matriz deben ser número; en el siguiente ejemplo se apreciará las posiciones con las que se debe trabajar la matriz, para poder manipular su información.

Figura 1.17.

Representación de matriz.



Nota. Los autores

La matriz representada en la Figura 1.17, tiene dos filas y dos columnas por lo tanto es de 2X2; dentro de los paréntesis el primer número representa la fila y el segundo número separado por una coma, representa la columna (fila, columna) por lo que si se desea obtener un dato de la matriz, se debe especificar la posición indicando la fila y columna en la que está ubicada la información.

Ejemplo 1:

Figura 1.18.

Matriz de nombres.

NOMBRES	Columna 1	Columna 2
Fila 1	"María"	"Sofía"
Fila 2	"Carlos"	"Fabián"

Nota. Los autores

En la Figura 1.18, se aprecia una matriz 2X2 que almacena nombres; el nombre Fabián está en la posición fila dos, columna dos, cuya expresión sería (2,2).

La matriz almacena mayor cantidad de información, con una sola declaración.

Ejemplo 2:

Figura 1.19.

Ejemplo de matriz que almacena números.

NÚMEROS	Columna 1	Columna 2
Fila 1	**178**	**5**
Fila 2	**3545**	**45**

Nota. Los autores

En la Figura 1.19 se aprecia una matriz 2X2 que almacena números; el número cinco está en la posición fila uno, columna dos, cuya expresión sería (1,2).

Actividad

Realizar un mapa conceptual del Capítulo 1, donde se identifique igualdad, diferencias y orden de conceptos.

CAPÍTULO 2

INICIANDO EN LA PROGRAMACIÓN

Capítulo 2

Iniciando en la Programación

2. Lógica de programación

El Objetivo del Capítulo 2 es introducir al lector en el aprendizaje de la lógica de programación, para que pueda convertir sus ideas en realidad mediante la representación de una solución a través de la implementación de algoritmos, uso de condiciones, repeticiones, estructuras y funciones.

Para programar se debe de pensar cómo resolver un problema tomando en cuenta las restricciones de cada lenguaje de programación.

El codificar un programa es aplicar a través de un código lo que se extrae del mundo real a la computadora, a través de una serie de parámetros preestablecidos.

2.1. Algoritmo

La Real Academia Española [RAE] (2022) define al algoritmo como: “Conjunto ordenado y finito de operaciones que permite hallar la solución de un problema”(párr. 1).

Un algoritmo es describir paso a paso en forma ordenada y consecutiva que se debe realizar para resolver un problema, por ejemplo:

Algoritmo contestar el teléfono.

1. Inicio.
2. Dirigirse al teléfono.
3. Levantar la bocina.
4. Contestar la llamada.
5. Colgar la bocina.
6. Fin.

Las características de un algoritmo son:

1. Todo algoritmo empieza con la palabra Inicio.
2. Enumerar cada paso.
3. Describir cada paso con palabras muy sencillas y si es necesario indicar que paso especifico se debe seguir.
4. Todo algoritmo termina con la palabra Fin.
5. Los algoritmos son finitos.
6. Los algoritmos resuelven un problema.

Los algoritmos pueden tener ciclos; para indicar la repetición se debe indicar qué numeración sigue para continuar; los algoritmos son parecidos a los itinerarios de actividades el cual debe llevar un estricto orden para poder alcanzar los objetivos planteados, por ejemplo:

Algoritmo ir a la universidad.

1. Inicio.
2. Levantarse de la cama y colocarse las zapatillas.
3. Dirigirse al baño para asearse.
4. Dirigirse a la habitación para vestirse.
5. Salir de la habitación y preparar el desayuno.
6. Desayunar y lavar la vajilla.
7. Seleccionar los útiles escolares y salir de casa.
8. Dirigirse a la parada de los buses.
9. Sí el bus escolar pasa, tomar el bus escolar caso contrario tomar el bus que siga la ruta escolar.
10. Sí el bus que tomó es el escolar bajarse en la facultad de la escuela caso contrario bajarse cerca de la universidad y dirigirse a la escuela.
11. Fin.

Algoritmo que suma dos números ingresados por el usuario.

1. Inicio.
2. Declarar las variables *A,B* y suma de tipo *Entero*.
3. Mostrar en pantalla “Ingrese el valor del primer sumando”.
4. El valor ingresado ´por el usuario asignarlo a la variable *A*.
5. Mostrar en pantalla “Ingrese el valor del segundo sumando”.
6. El valor ingresado por el usuario asignarlo a la variable *B*.
7. Sumar el valor de la variable *A* con el valor de la variable *B* y asignarlo a la variable suma.
8. Mostrar en pantalla el valor de la variable suma.
9. Fin.

Los algoritmos se utilizan para comprender la lógica y la secuencia de lo que posteriormente se convertirá en un código de programación.

Algoritmo que muestra en pantalla si un número ingresado por el usuario es positivo.

1. Inicio.
2. Declarar la variable *N* como *Entero*.
3. Solicitar al usuario que ingrese un número entero y asignarlo a la variable *N*.

4. Si *N*>0 entonces mostrar en pantalla el número que ingreso es positivo, caso contrario mostrar el número es negativo.
5. Fin.

Algoritmo que multiplica tres veces un número ingresado por el usuario.

1. Inicio.
2. Declarar la variable *N*, Resultado de tipo *Real*.
3. Declarar la variable *contador* como *Entero*.
4. Mostrar en pantalla “Ingrese un número decimal”.
5. El valor ingresado por el usuario asignarlo a la variable *N*.
6. Asignar 1 a la variable *contador*.
7. Asignar 1 a la variable Resultado.
8. Si la variable *contador*<=3 entonces multiplicar Resultado * *N* y asignarlo en la variable Resultado e incrementar la variable contador en 1.
9. Si la variable *contador*<=3 repetir el paso 8, caso contrario continuar al paso 10.
10. Mostrar en pantalla el valor de la variable Resultado.
11. Fin.

2.2. Diagrama de Flujo

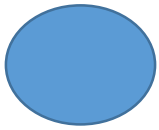
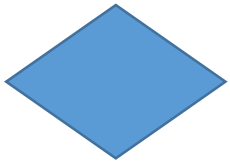
Un diagrama de flujo es la representación de un algoritmo a través de figuras geométricas, se puede decir que es la forma gráfica de un algoritmo. Existe una gran cantidad de formas, pero se representarán las más conocidas.

Formas de representar un algoritmo en un diagrama de flujo o DFD.

Tabla 2.1.

Símbolos de DFD secuenciales.

Símbolo	Uso
	Inicio y Fin del algoritmo.
	Asignación, declaraciones y procesos del algoritmo.
	Imprimir en pantalla Entrada de datos por cualquier medio.
	Imprimir por impresora.

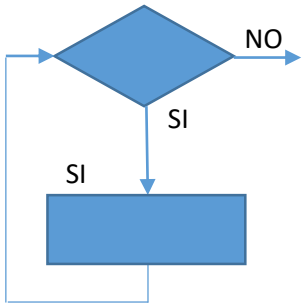
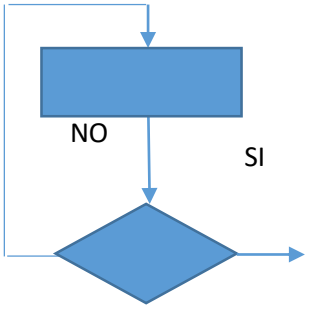
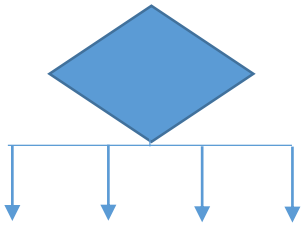
Símbolo	Uso
	Conector ayuda a la secuencia del DFD en la misma página, se puede colocar dentro del círculo una letra o número.
	Conector ayuda a la secuencia del DFD en otra página, se puede colocar dentro del círculo una letra o número.
	Realiza la llamada a una subproceso o procedimiento.
	Representa una condición.
	Representa la secuencia del DFD.
	Entrada por teclado.

Nota. Los autores, Microsoft Word

En la Tabla 2.2 se muestra diagramas de flujo para representar condicionales y ciclos.

Tabla 2.2.

Símbolos de DFD condicional y cíclica.

SÍMBOLO	USO
	Mientras: ingresará por el sí siempre y cuando se cumpla la condición.
	Repetir: se repetirá siempre y cuando la condición sea falsa.
	Casos: según la opción seleccionada en la condición seguirá el camino para continuar con la secuencia.
	Para: repetición automática.

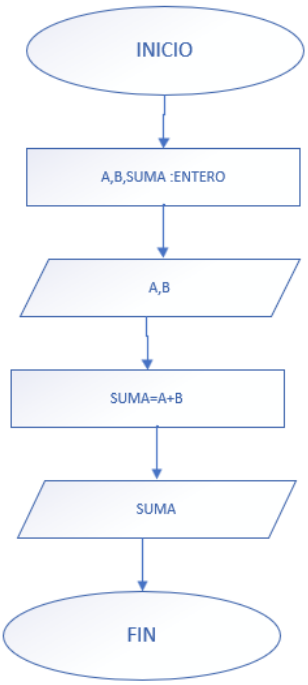
Nota. Los autores, Microsoft Word

En un diagrama de flujo tradicional se debe respetar las siguientes leyes:

1. Las líneas o flechas no deben cruzarse entre sí o las figuras.
2. Se debe graficar en secuencia de arriba hacia abajo o de izquierda a derecha.
3. Colocar texto claro y lo más conciso posible.

Ejemplo de un Diagrama de Flujo en la forma tradicional que suma dos números ingresados por el usuario.

Figura 2.1.
Diagrama de flujo suma dos números



Nota. Los autores, Microsoft Visio

El diagrama de flujo al ser un algoritmo tiene un inicio y un fin, las operaciones y declaraciones de variables están representadas por el paralelogramo y su secuencia está representada por las líneas de flujo.

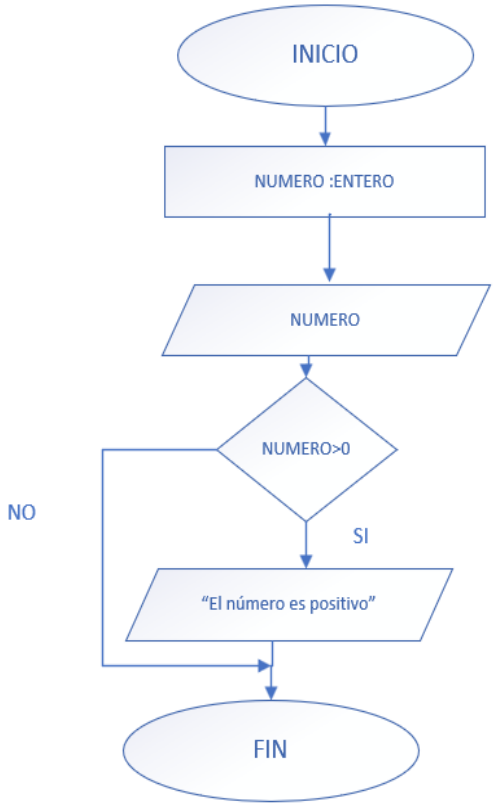
En la Figura 2.1 se inicia con la declaración de tres variables **A**,**B** y **suma** de tipo entero, luego se le solicita el ingreso de esos dos valores, los cuales serán ingresados por el usuario para ser sumados y asignados en la variable **suma**, la cual será mostrada su contenido en pantalla.

Para proceder realizar diagramas de flujo en una hoja, sin la ayuda de un *software*, se le puede realizar a través de las plantillas de diagramación.

Ejemplo Diagrama de Flujo que muestra en pantalla si un número ingresado por el usuario es positivo.

Figura 2.2.

Diagrama de flujo número positivo



Nota. Los autores, Microsoft Visio

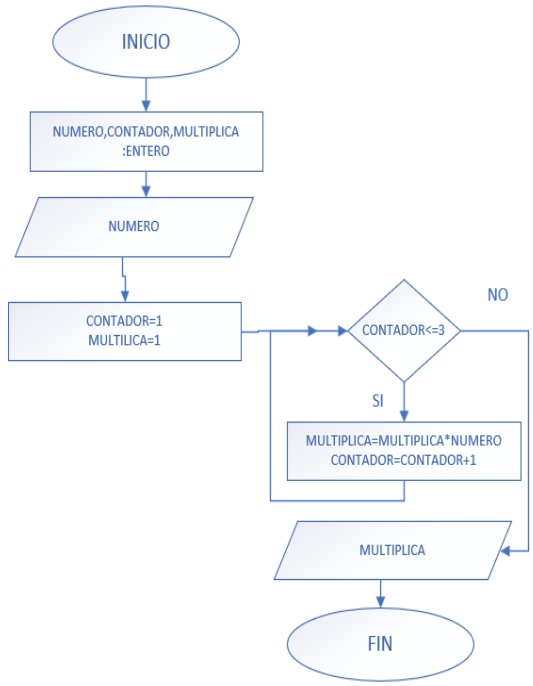
En el ejemplo de la Figura 2.2, se inicia con la declaración de una variable de nombre **número** de tipo entero, este valor es ingresado por el usuario y se procede a realizar una pregunta, sí el valor ingresado por el usuario en la variable **número** es mayor que cero, se mostrará en pantalla que el número es positivo, caso contrario tomará el camino del no y terminará el algoritmo.

En el algoritmo se ha procedido a trabajar con el rombo que representa una condición, donde según el análisis de la pregunta formulada en la condición podrá tomar sólo un camino el verdadero o el falso. Pero es importante aclarar que las estructuras condicionales sólo se ejecutarán una sola vez, pues no se debe confundir con los ciclos, por lo general suele ocurrir que lo confunde con el ciclo **mientras**.

Ejemplo de un Diagrama de Flujo tradicional que multiplica tres veces un número ingresado por el usuario.

Figura 2.3.

Diagrama de flujo multiplicar



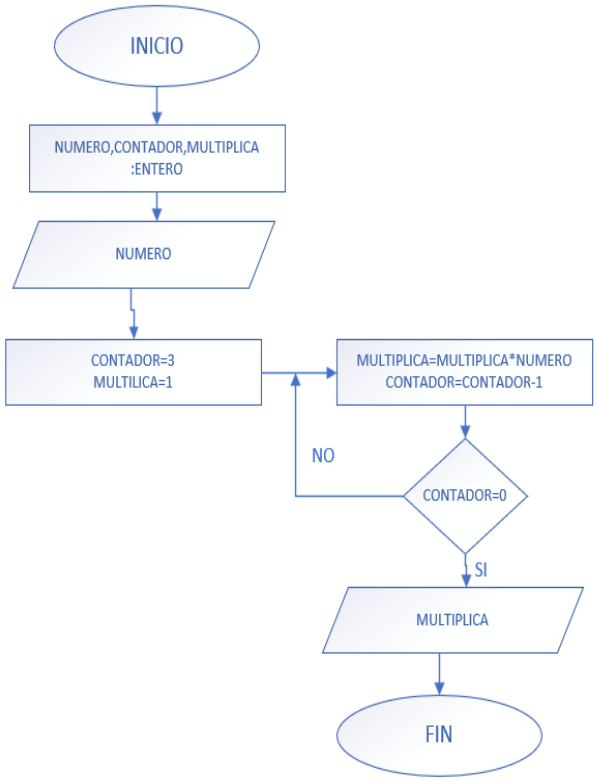
Nota. Los autores, Microsoft Visio

En el ejemplo de la figura 2.3, se ha procedido a declarar tres variables de tipo entero, **número**, **contador** y **multiplica**, el **número** es un valor leído por teclado pues es ingresado por el usuario, luego se procede a inicializar las variables **contador** y **multiplica** con el valor de uno, para ingresar a un ciclo conocido como **mientras** donde se pregunta por la condición si **contador** es menor o igual a tres, si esto es verdad, se procede a multiplicar el número por la variable **multiplica** que inicialmente contiene el valor neutro de la multiplicación y se le asigna a la misma variable para haga las veces de un acumulador para todos los procesos de multiplicación que tenga que ejecutar el ciclo, además a través de la variable **contador** se le irá incrementando al contador para poder controlar que el ciclo no se vuelva infinito, una vez que no se cumpla la condición se acabará el ciclo y continuará con el algoritmo , imprimiendo el valor final de la variable **multiplica** y finalizará el algoritmo.

El ciclo **mientras** siempre empezará con la pregunta de la condición, la que se representa con un rombo al igual que el condicional e ingresará al ciclo siempre que la condición sea verdadera, pero cada vez que se cumpla la condición se repetirá el ciclo, se debe tener muy en cuenta el control de la condición caso contrario se volverá un ciclo infinito.

Figura 2.4.

Ejemplo de diagrama con ciclo REPETIR



Nota. Los autores, Microsoft Visio

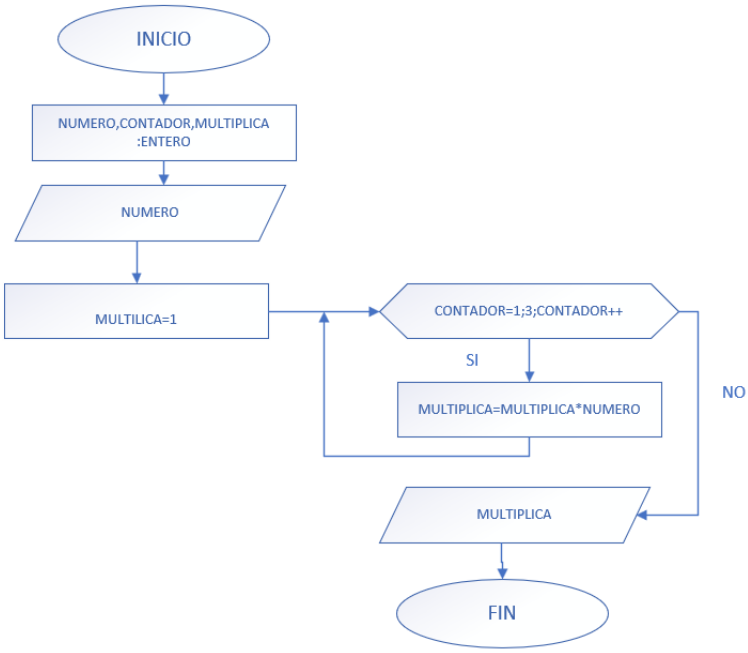
En el ejemplo de la Figura 2.4, se ha procedido a declarar tres variables de tipo entero, **número**, **contador** y **multiplica**, el **número** es un valor leído por teclado pues es ingresado por el usuario, luego se procede a inicializar las variables **contador** con el valor de tres y **multiplica** con el valor de uno, para ingresar a un ciclo conocido como **repetir**, donde se pregunta por la condición después de haber ejecutado el bloque de procesos, que en este caso es la multiplicación y el decremento del

contador, si **contador** no es igual a cero, se procede a volver a multiplicar y decremento el **contador**, cuando el **contador** sea igual a cero saldrá del ciclo y continuará con el algoritmo, imprimiendo el valor final de la variable **multiplica** y finalizará el algoritmo.

El ciclo r**epetir** siempre empezará con la ejecución del bloque de instrucciones que está dentro de la estructura para luego proceder a hacer la consulta de la condición, a diferencia del mientras que primero genera la condición.

Figura 2.5.

Diagrama de flujo con el ciclo PARA



Nota. Los autores, Microsoft Visio

En el ejemplo de la Figura 2.5, se ha procedido a declarar tres variables de tipo entero, **número**, **contador** y **multiplica**, el **número** es un valor leído por teclado pues es ingresado por el usuario, luego se procede a inicializar la variable **multiplica** con el valor de uno, para ingresar a un ciclo conocido como **para**, donde en la estructura se inicializa el **contador**, se le expresa la condición y se le indica el incremento del **contador**, si la condición se cumple procederá al proceso de la multiplicación y cuando la condición deje de cumplirse, saldrá del ciclo, para continuar con el algoritmo y mostrar en pantalla el valor de la variable **multiplica** para finalizar el algoritmo.

Los diagramas de flujo ofrecen muchas ventajas como son:

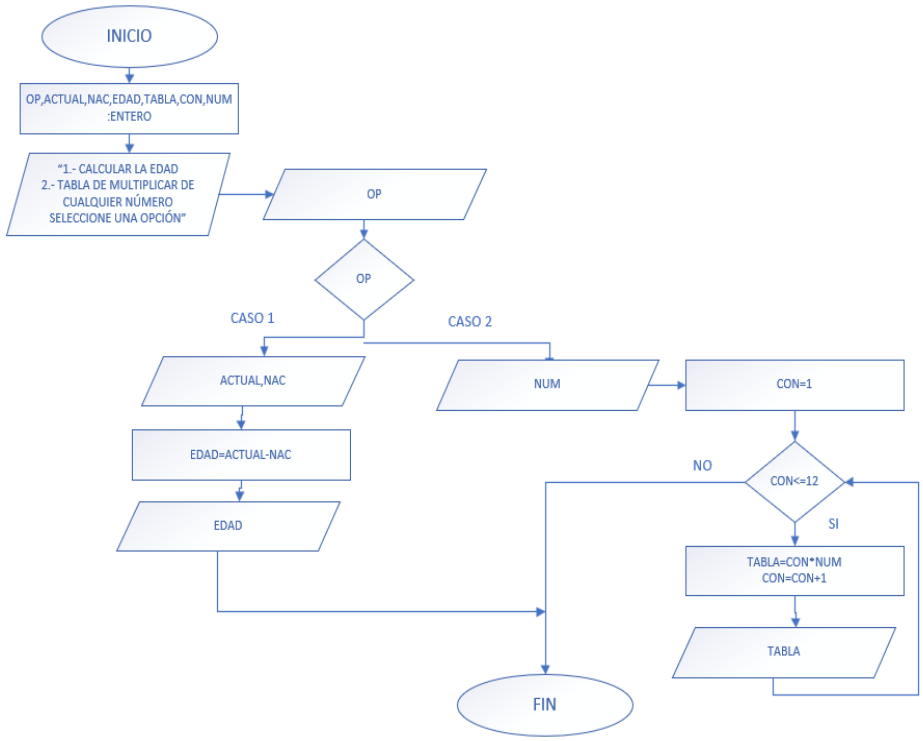
- Al ser representado en forma gráfica, se puede apreciar la secuencia que sigue el algoritmo y facilita seguir una secuencia para identificar su solución.
- Es más fácil implementar modificaciones o actualizaciones al algoritmo.
- Facilita su implementación a un código.
- Es muy importante al momento del desarrollo de un proyecto de desarrollo de software.

Diagrama de flujo con Selección

Diagrama de flujo que muestra al usuario un menú en pantalla de las cuales se tendrá que seleccionar sólo una opción.

Figura 2.6.

Diagrama de flujo de menú de opciones con CASE



Nota. Los autores, Microsoft Visio

En el ejemplo de la Figura 2.6, muestra un diagrama de flujo con el condicional case(caso), la sentencia case facilita a la hora de trabajar con menús; en el ejemplo planteado se procede con la declaración de variables de tipo entero, para proceder a mostrar por pantalla dos opciones de las cuales el usuario podrá escoger ingresando a través de teclado un número el cual puede ser uno o dos, que será almacenado en la variable **op**, la

sentencia case está representada a través de un rombo que tiene dos caminos, según el número digitado por el usuario.

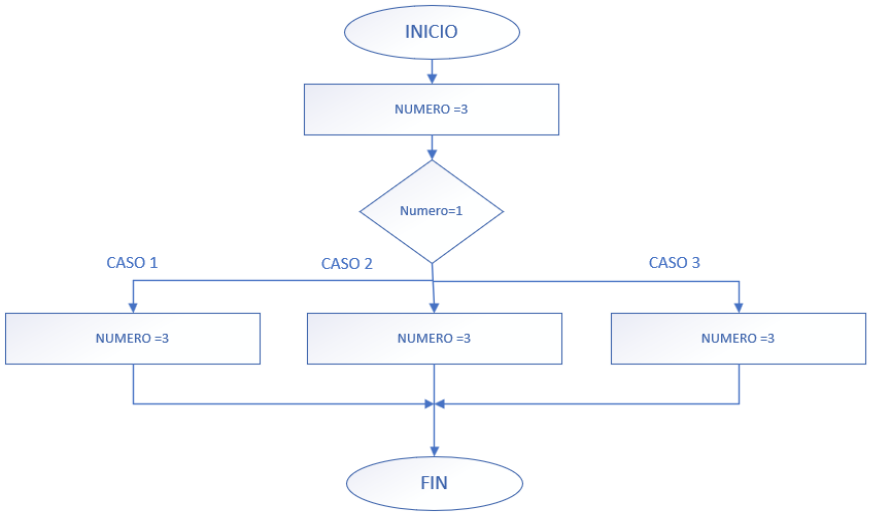
Si el usuario optó por la opción uno, la sentencia tomará el camino de la coincidencia, que en el diagrama de flujo es el caso 1, solicitando el año actual y el año de nacimiento, se procede al cálculo respectivo y se mostrará la edad por pantalla, para culminar con el algoritmo.

Si el usuario escoge la opción dos, que en el algoritmo está representado por el caso 2, se solicitará el número que se desea generar la tabla de multiplicar; se inicializará la variable **con** en uno para ingresar al ciclo **mientras**, donde se evalúa la condición si la condición es verdadera continuará con el ciclo, ejecutando el proceso de la multiplicación, incrementado el **contador** y mostrando en pantalla los resultados de la multiplicación, cuando la condición deje de cumplirse, se saldrá del ciclo para finalizar con el algoritmo.

La sentencia case en un condicional que trabaja internamente como varias preguntas de forma interna, por ejemplo:

Figura 2.7.

Representación de una sentencia CASE

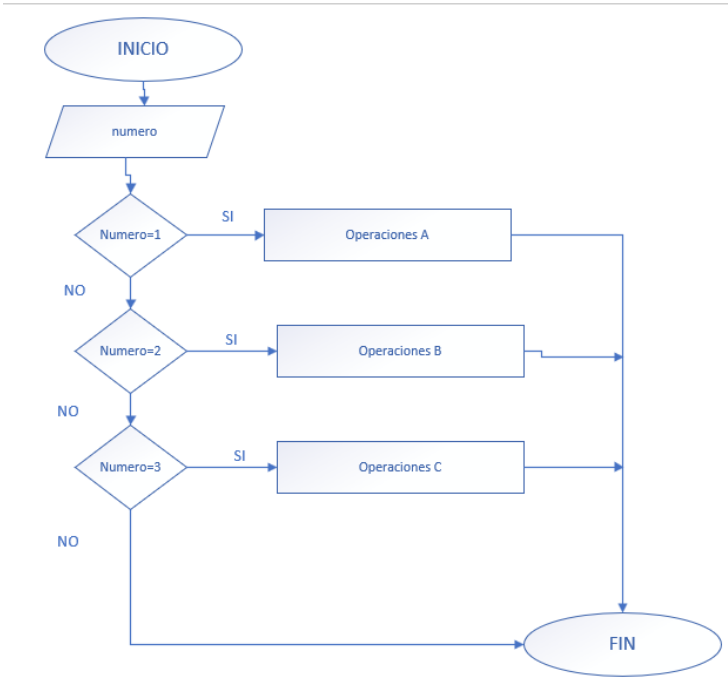


Nota. Los autores, Microsoft Visio

Ejemplo de trabajo interno del case, representado con un condicional como salida SI o NO.

Figura 2.8.

Representación de case con varias condiciones



Nota. Los autores, Microsoft Visio

Diagrama de flujo con Subproceso

Un subproceso o procedimiento es un pequeño algoritmo que ejecuta algo específico y que puede ser llamado o convocado las veces que sea necesarias, también se le conoce como procedimiento cuando no devuelve nada a la función al momento de terminar su ejecución.

Los procedimientos en diagramas de flujo se los representa con:

Figura 2.9.

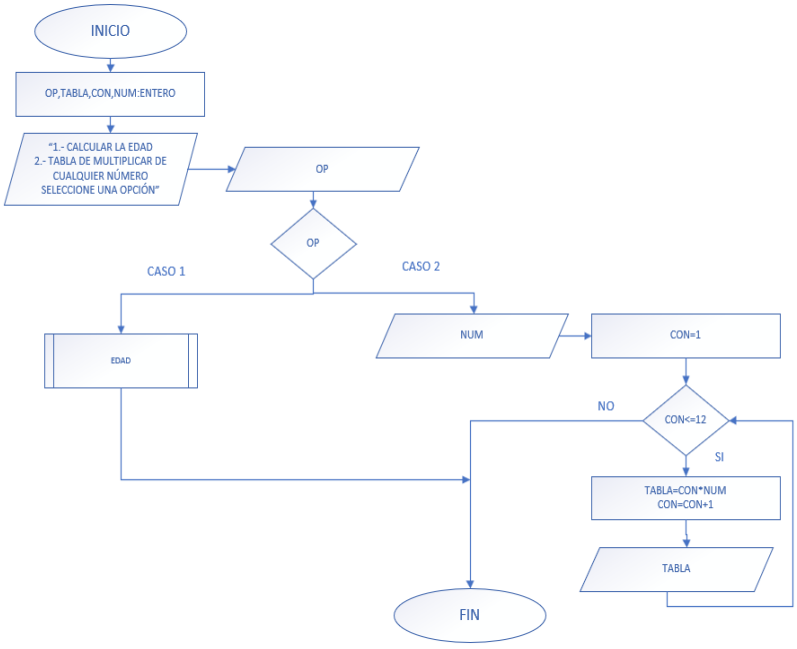
Representación de un Subproceso

Nota. Los autores, Microsoft Visio

Se realiza el diagrama de flujo de la Figura 2.5, pero con subproceso o procedimiento.

Figura 2.10.

Diagrama de flujo con subproceso



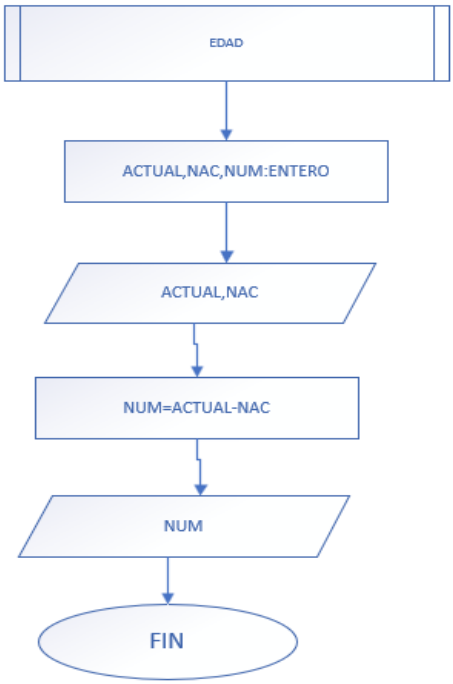
Nota. Los autores, Microsoft Visio

En el ejemplo de la Figura 2.10, cuando el algoritmo toma el camino del caso1, tiene que solicitar un llamado al procedimiento edad, el cual no recibe argumentos.

Los argumentos pueden ser valores, funciones, variables, constantes o algún tipo de estructura.

Figura 2.11.

Diagrama de flujo de un subproceso



Nota. Los autores, Microsoft Visio

En la Figura 2.11, se representa el algoritmo que pertenece al subproceso, que ejecuta el cálculo de la edad del usuario, para proceder a mostrarlo por pantalla y finalizar con el algoritmo, una vez culminado el subproceso continúo con el algoritmo que lo invocó.

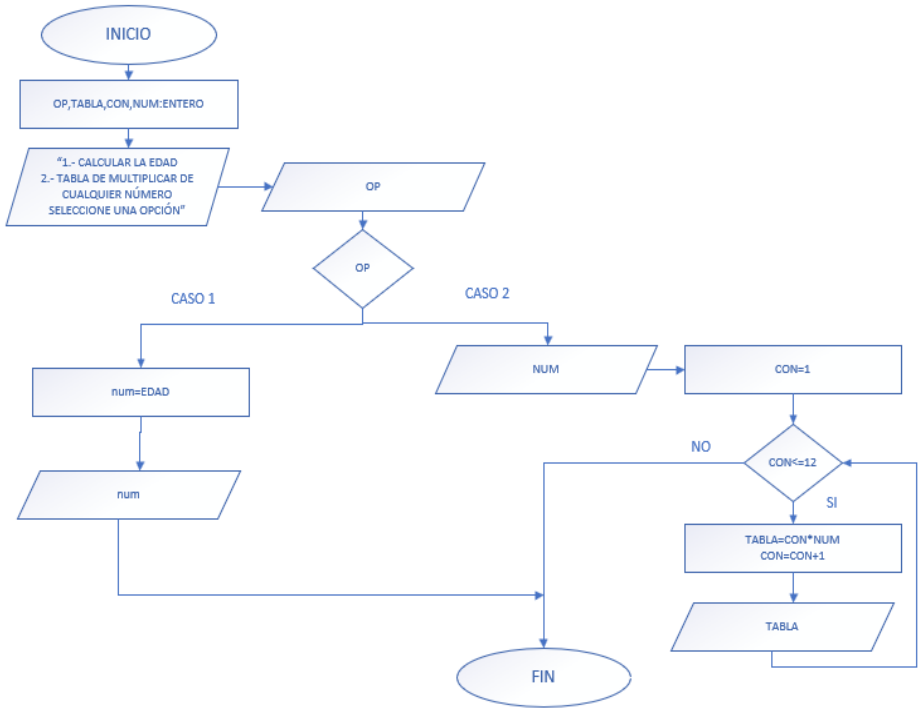
En los diagramas de flujo existe otra forma de representar los subprocesos que es a través de los rectángulos; se utiliza la gráfica del rectángulo cuando el subproceso devuelve algún valor y el algoritmo que lo invoca a través del rectángulo podrá realizar una asignación para recibir el valor.

El ejemplo de la Figura 2.12, se procederá a representar con un rectángulo de asignación o definición.

El llamado al subproceso se le ha representado a través de un rectángulo donde se ha procedido a realizar una asignación, para que, cuando se retorne de la ejecución del algoritmo edad, se proceda a almacenar el valor en la variable **num** y se muestre en pantalla.

Figura 2.12.

Diagrama de flujo con subproceso que devuelve un valor



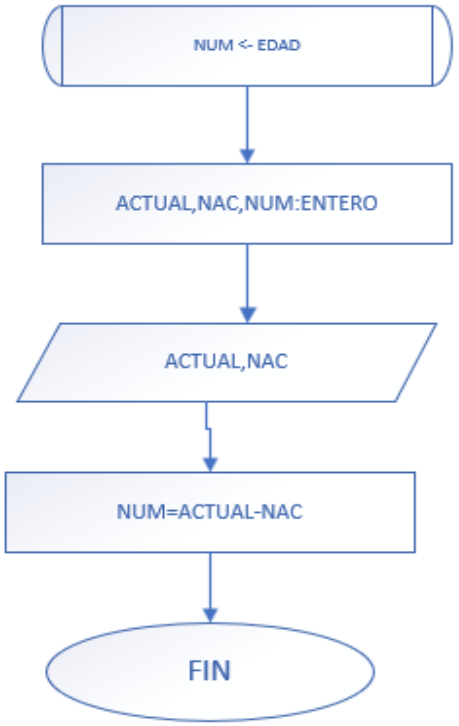
Nota. Los autores, Microsoft Visio

Cuando ocurre que el subproceso devuelve un valor al algoritmo que lo invoca, se le conoce a este proceso como función.

Representación del subproceso de la Figura 2.13, pero con retorno de un valor.

Figura 2.13.

Subproceso que devuelve valor

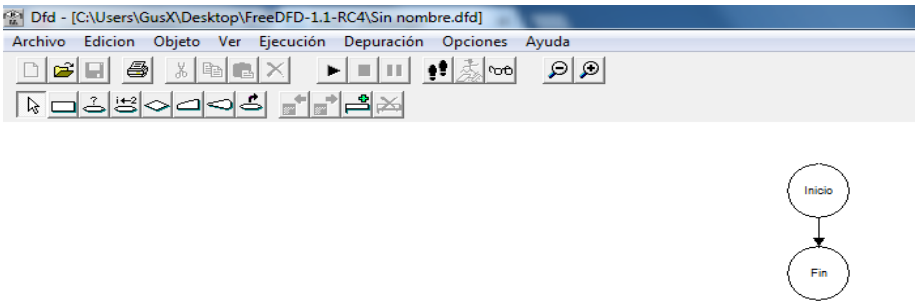


Nota. Los autores, Microsoft Visio

El diagrama de flujo de datos (DFD) es la representación gráfica del flujo de datos de un proceso o sistema. El diagrama de flujo de datos es una de las principales formas de crear un modelo de procesos de un sistema (European Knowledge Center for Information Technology, 2021, párr. 1).

Figura 2.14.

Editor DFD



Nota. Los autores, DFD

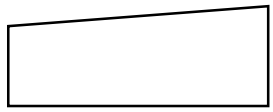
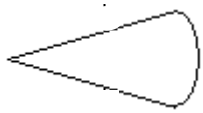
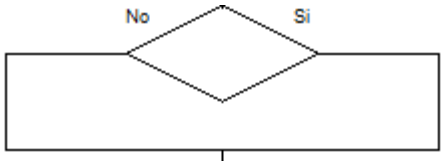
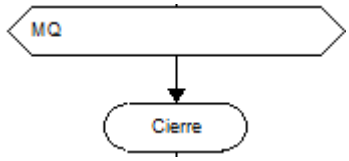
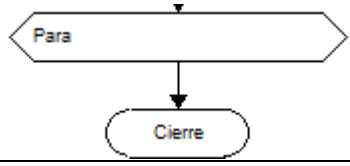
Para graficar un diagrama de flujo se utilizan figuras geométricas, que representarán cada uno de los pasos de un algoritmo.

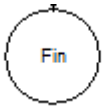
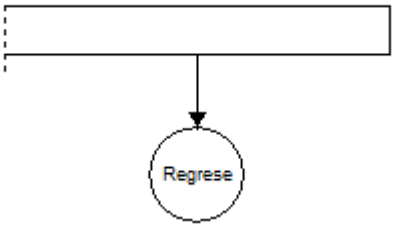
Cada figura está conectada a través de una flecha o línea que indica la secuencia del diagrama.

Tabla 2.3.

Figuras utilizadas en Editor DFD

Figuras	Utilización
	Representa el inicio del algoritmo.

Figuras	Utilización
	Representa la asignación o para realizar cálculos.
	Representa la lectura de los datos que digite el usuario.
	Representa la presentación en pantalla.
	Representa una condición.
	Representa el ciclo Mientras.
	Representa el ciclo Para.
	Representa el llamado a un SubPrograma.

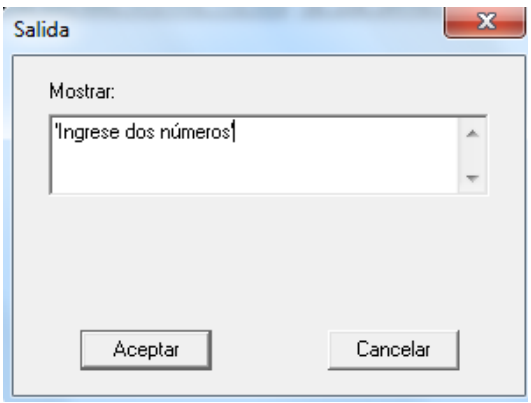
Figuras	Utilización
	Representa el final del algoritmo.
	Representa un SubPrograma.
	Representa la unión y secuencia entre figuras.

Nota. Los autores, DFD

Para mostrar en pantalla un mensaje al usuario este debe escribirse entre comillas simples (‘’).

Figura 2.15

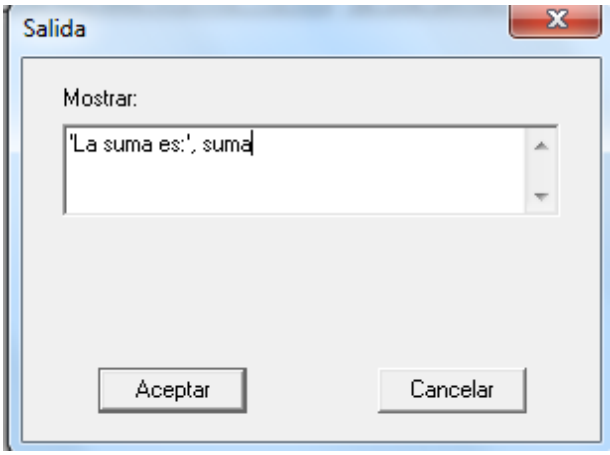
Ventana de salida, para mostrar en pantalla un mensaje.



Nota. Los autores, DFD

Para mostrar en pantalla el contenido de una variable, debe colocarse la variable antes o después de la comilla simple, separada por una coma o solo colocarse la variable sin el mensaje.

Figura 2.16
Ventana de salida, para mostrar en pantalla un mensaje y el contenido de una variable.

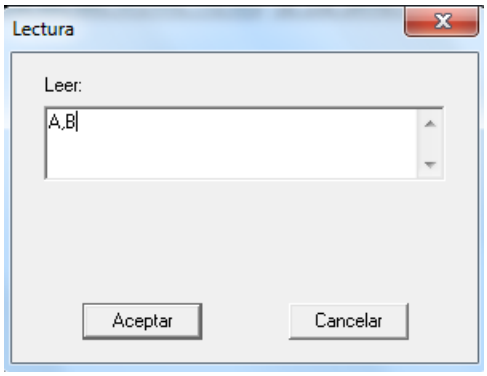


Nota. Los autores, DFD

Se puede leer un valor ingresado por el usuario o varios a la vez, solo se tiene que separar con una coma cada variable que va a recibir el valor.

Figura 2.17.

Ventana de lectura, para recibir dos valores en distintas variables.

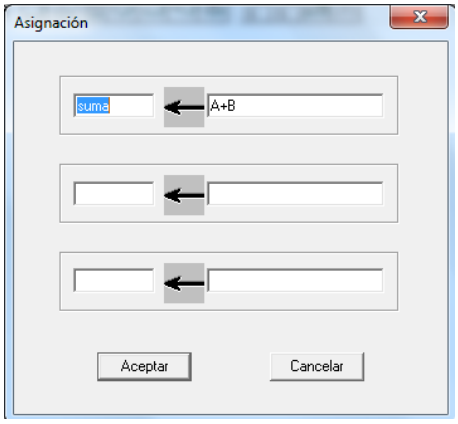


Nota. Los autores, DFD

DFD solo permite tres asignaciones por figura; si se desean realizar más de tres asignaciones, se colocará otra gráfica de asignación adicional o las que sea necesarias.

Figura 2.18.

Ventana de asignación



Nota. Los autores, DFD

Declaración de variables

En DFD no es necesario realizar una declaración, con sólo realizar una asignación de valores enteros, decimales, letra, cadena o booleana (verdadero o falso).

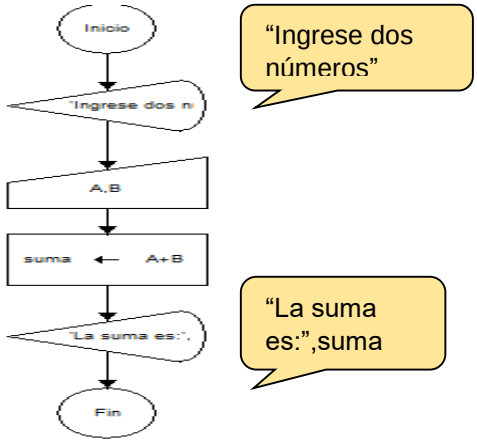
Constantes

No es necesario realizar una declaración para poder trabajar con constantes, con solo realizar la asignación es suficiente para trabajarla durante todo el diagrama.

Ejemplo de un Diagrama de Flujo que suma dos números ingresados por el usuario.

Figura 2.19.

Diagrama de flujo que suma dos números.



Nota. Los autores, DFD

En DFD se utilizan operadores lógicos y relaciones como:

Tabla 2.4.
Operadores aritméticos.

Símbolo	**Expresión**
+	Suma
-	Resta
*	Multiplicación
/	División
Mod	Residuo
^	Potencia
Sqrt	Raíz cuadrada
Abs	Valor absoluto
Random	Números aleatorios
Exp	Exponencial de e
Round	Redondeo
Trunc	Parte entera

Nota. Los autores

Tabla 2.5.

Operadores relacionales

Símbolo	Nombre
>	Mayor que
<	Menor que
>=	Mayor igual
<=	Menor igual
=	Igual
!=	Diferente

Nota. Los autores

Tabla 2.6.

Operadores lógicos

Símbolo	Significado
and	y
or	o
not	no

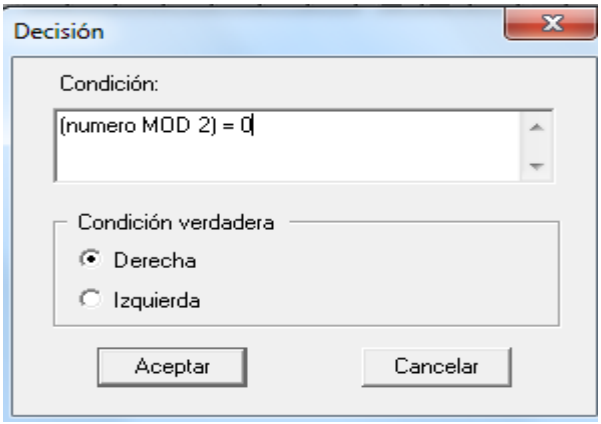
Nota. Los autores

Condición

Al colocar una condición esta tiene dos opciones ir por el Si o por el No; estos dos caminos se los puede intercambiar si se lo desea en el momento que se coloque la pregunta o condición dentro de la opción del rombo.

Figura 2.20.

Ventana de la condición

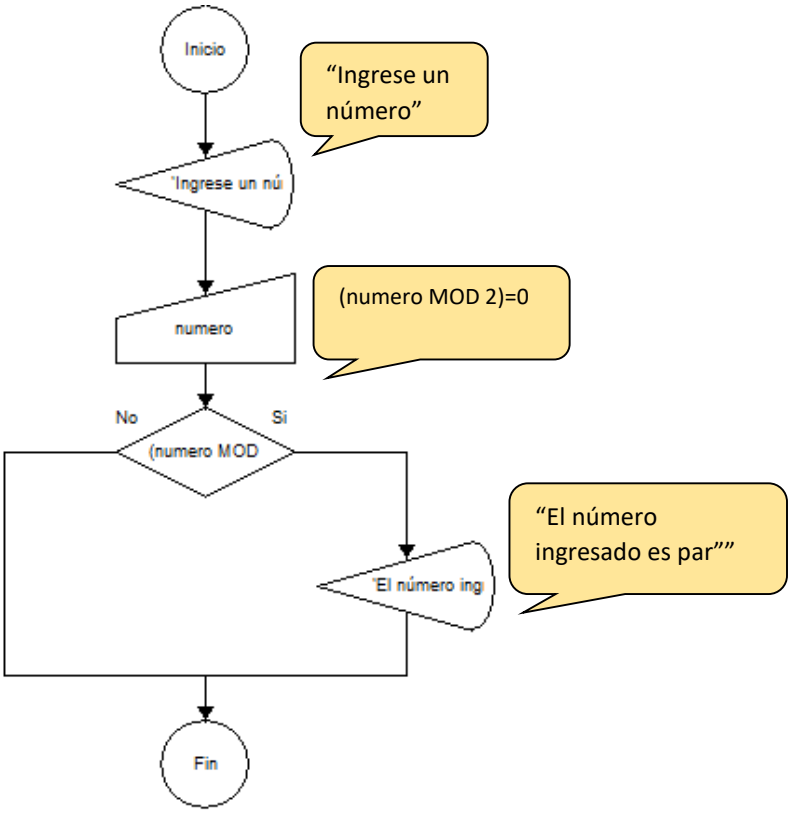


Nota. Los autores, DFD

Ejemplo: Diagrama de Flujo que muestra en pantalla si un número ingresado por el usuario es positivo.

Figura 2.21.

Diagrama de Flujo con condición.



Nota. Los autores, DFD

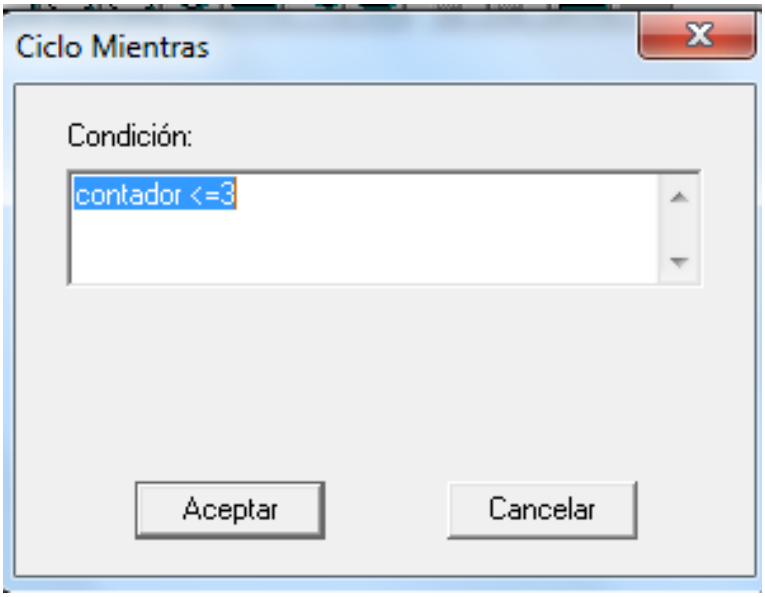
Ciclos Repetitivos

DFD ofrece dos tipos de ciclos el **Mientras** y el **Para.**

En el ciclo **Mientras** (MQ) al dar clic en la gráfica se mostrará la ventana donde se coloca la condición que deberá ser verdadera para que ingrese al ciclo.

Figura 2.22.

Diagrama de Flujo con condición

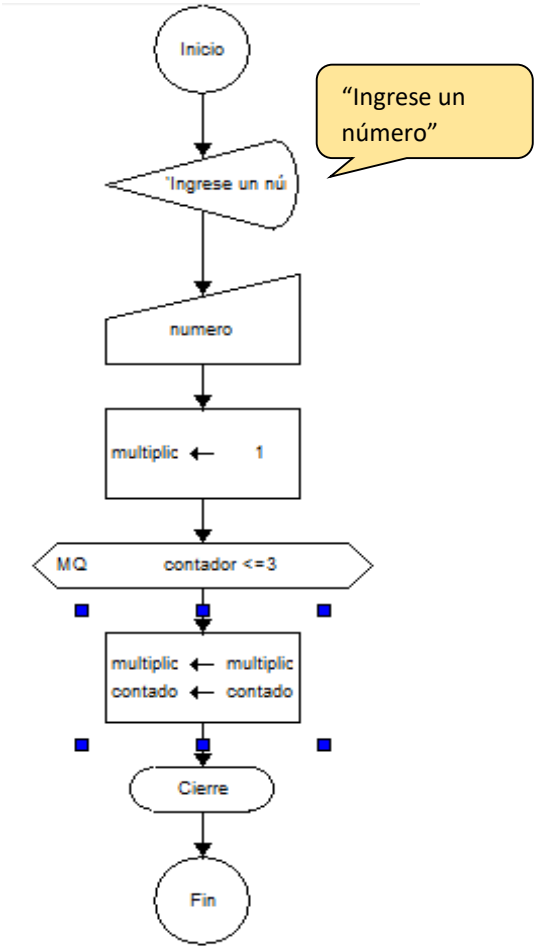


Nota. Los autores, DFD

Ejemplo de un Diagrama de Flujo que multiplica tres veces un número ingresado por el usuario.

Figura 2.23.

Diagrama de flujo con Mientras.

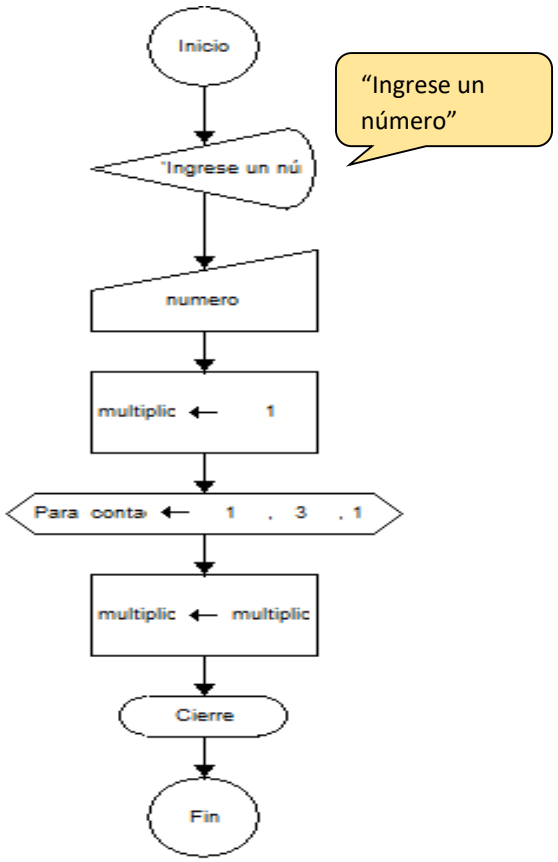


Nota. Los autores, DFD

En el ciclo **Para** al dar clic en la gráfica se mostrará la ventana donde se da valor al contador, se indica el límite, el incremento o decremento.

Figura 2.24.

Diagrama de flujo con Mientras



Nota. Los autores, DFD

La implementación de un ciclo de repetición dependerá de la estructura del problema a resolver, pero se puede utilizar el ciclo **Mientras**, cuando no se conozca cuantas veces se va a iterar o cuando la condición pueda cambiar e implementar un ciclo **Para** cuando ya se establece el número de veces a iterar.

Subprograma

En un diagrama de flujo primero se debe crear el programa principal que es de donde se procederá a realizar la llamada al subprograma. El subprograma ejecuta las instrucciones que se le hayan especificado y vuelve al programa principal para continuar o dirigirse al fin.

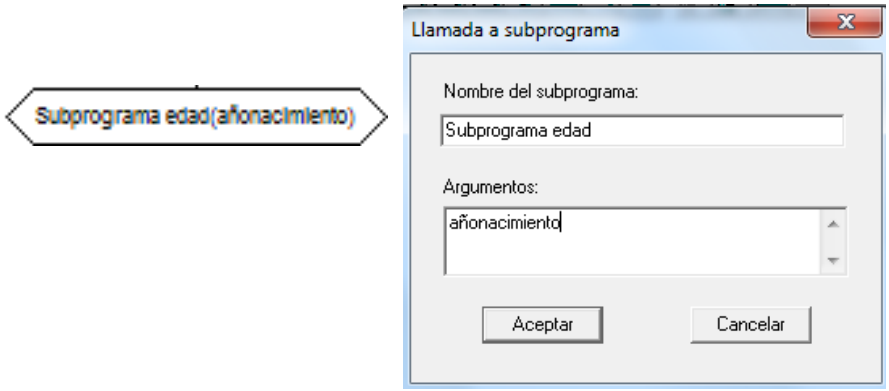
Un Subprograma ayuda a que el programa principal no se cargue con muchas instrucciones. Existen 2 formas de realizar una llamada a un Subprograma, con parámetros y sin parámetros.

Con parámetros:

Se coloca el nombre y se indica las variables que se van a enviar conocidas como parámetros.

Figura 2.25.

Llamado de Subprograma con parámetro.

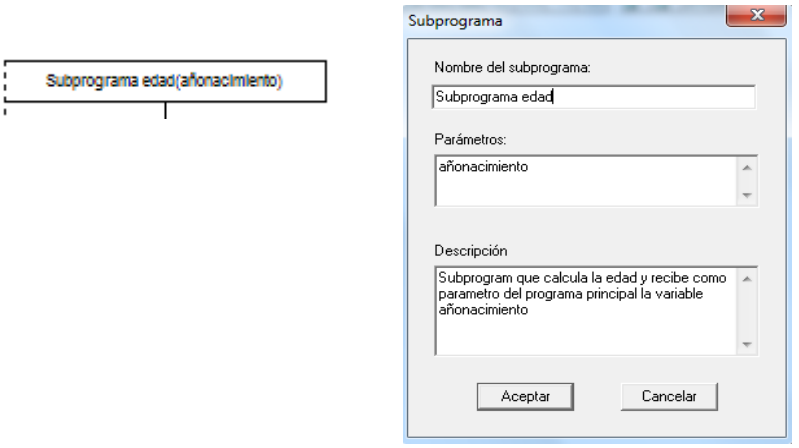


Nota. Los autores, DFD

El Subprograma se debe crear de la siguiente forma:

Figura 2.26.

Subprograma con parámetro

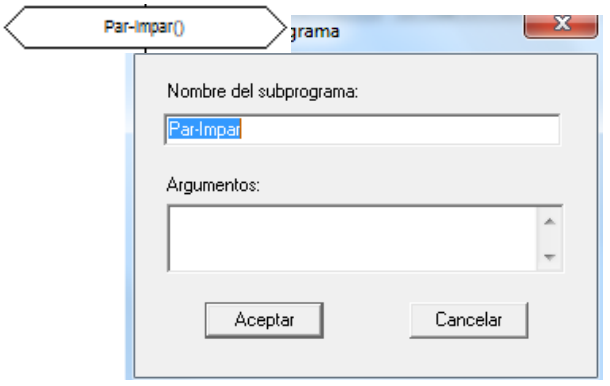


Nota. Los autores, DFD

Llamado de Subprograma sin parámetro:

Figura 2.27.

Llamado de Subprograma sin parámetro

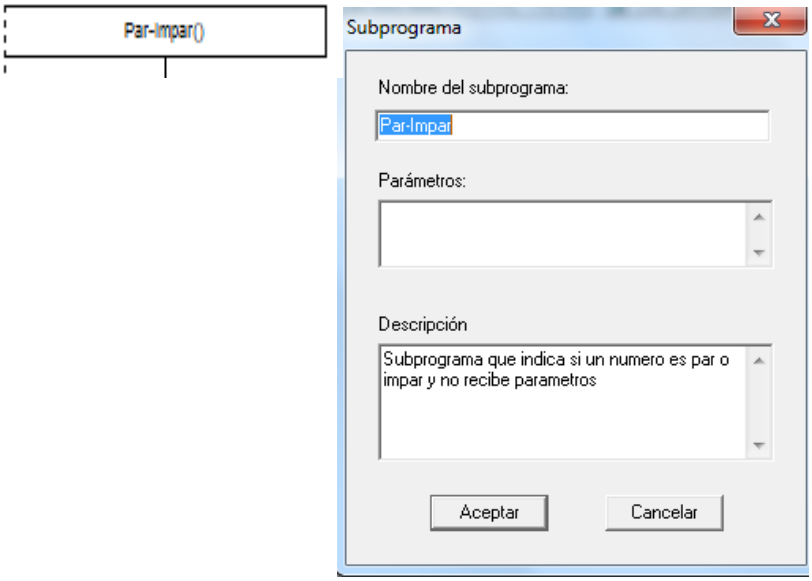


Nota. Los autores, DFD

El Subprograma se realiza de la siguiente forma:

Figura 2.28.

Subprograma sin parámetro

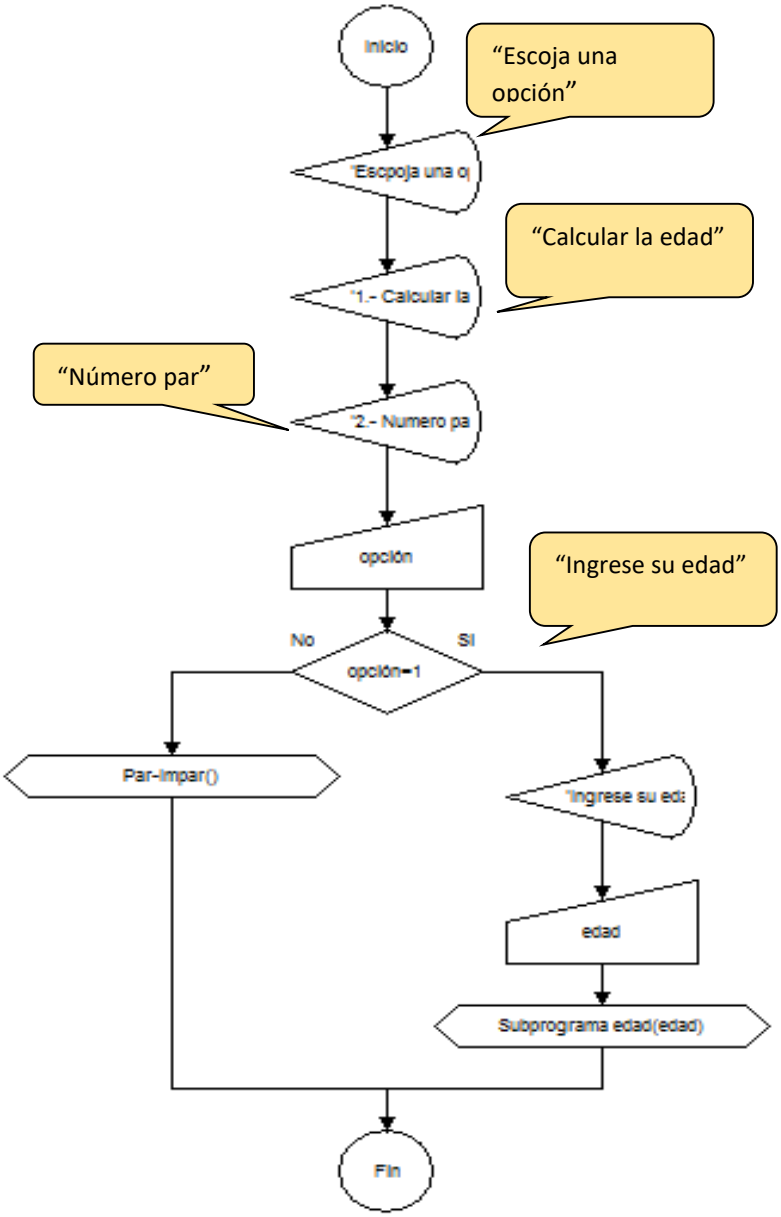


Nota. Los autores, DFD

Ejemplo de un subprograma que muestra un menú al usuario según las opciones que seleccione, ejecuta el subprograma que corresponda.

Figura 2.29.

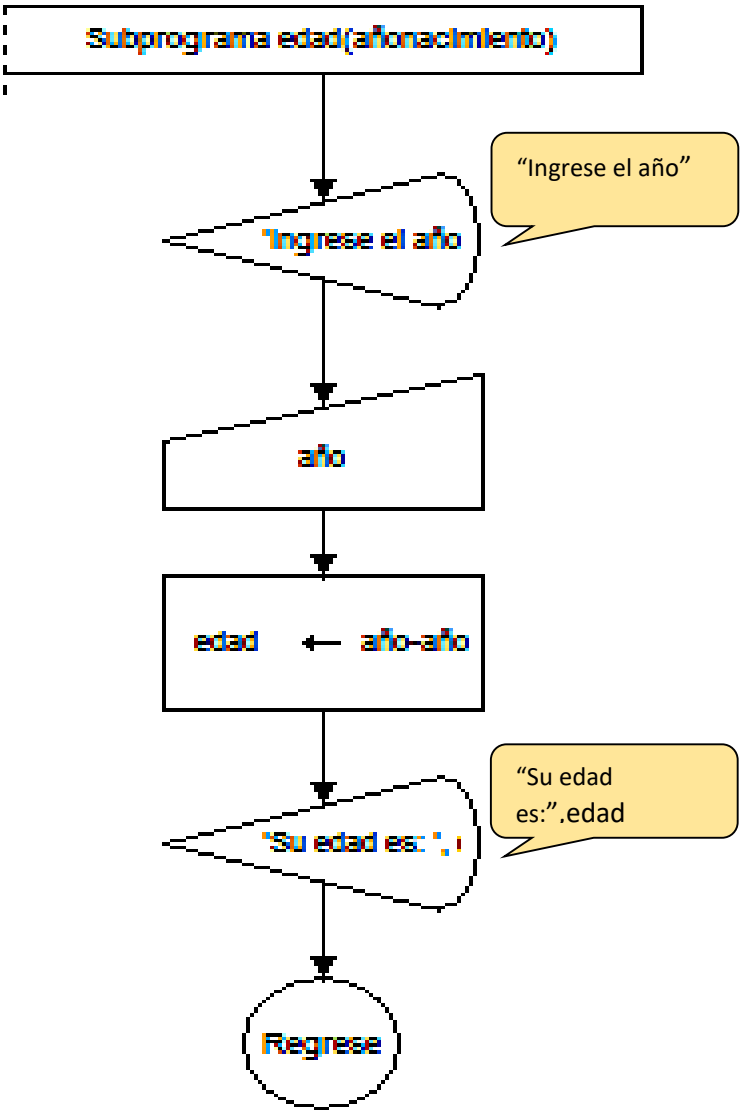
Diagrama con dos llamadas a Subprograma



Nota. Los autores, DFD

Figura 2.30.

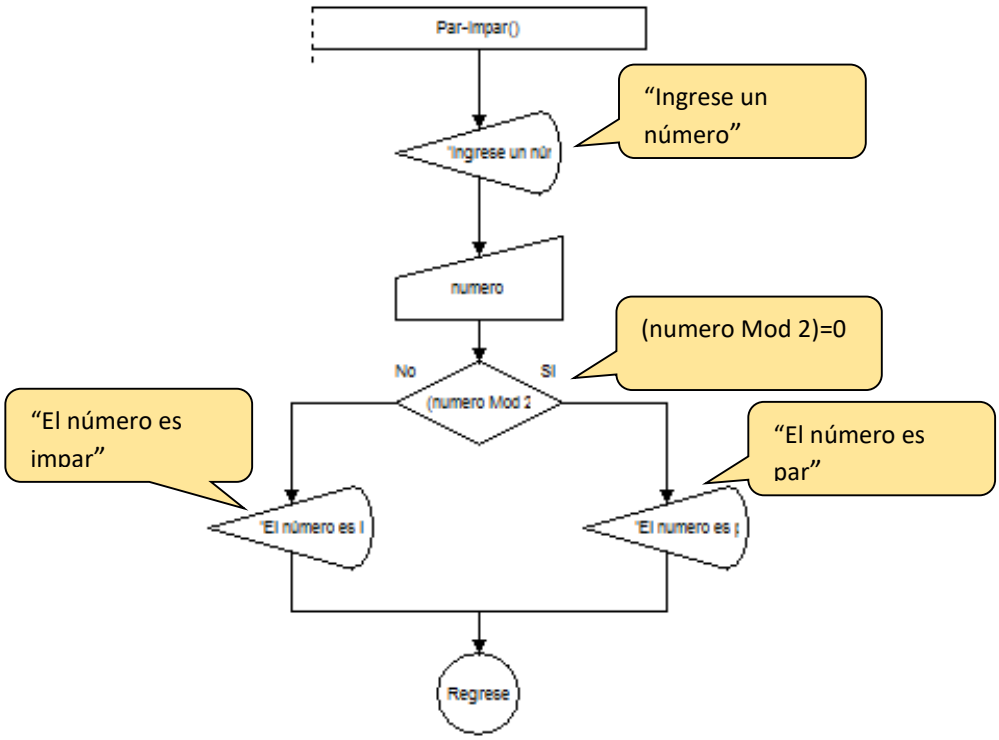
Subprograma que calcula la edad



Nota. Los autores, DFD

Figura 2.31.

Subprograma verifica si un número es par o impar



Nota. Los autores, DFD

2.3. Seudocódigo

Robledano (2019) indica que "El **pseudocódigo** es una forma de expresar los distintos pasos que va a realizar un programa, de la forma más parecida a un lenguaje de programación".

Es una sentencia de código muy sencilla de usar para programadores principiantes, en el cual se utilizan palabras reservadas (palabras pre programadas para uso exclusivo del sistema) para ejecución del algoritmo sin tener muchos inconvenientes con la estricta sintaxis de un lenguaje de programación.

“La herramienta PSeInt es un intérprete de un lenguaje de programación basado en pseudocódigo” (Beúnes y Vargas, 2019, p. 5).

“PSeint, una herramienta gratuita y de código abierto con la que el alumnado no solo descubre que es la lógica de la programación; también aprende conceptos básicos” (de Miguel, 2023, párr. 1).

Uno de los programas gratuitos que facilita el uso del seudocódigo es PSeInt. Es un programa fácil de utilizar y no se necesita licencia, con solo digitar en el buscador de su preferencia la palabra PseInt, automáticamente saldrán muchos lugares donde podrá descargar totalmente gratis.

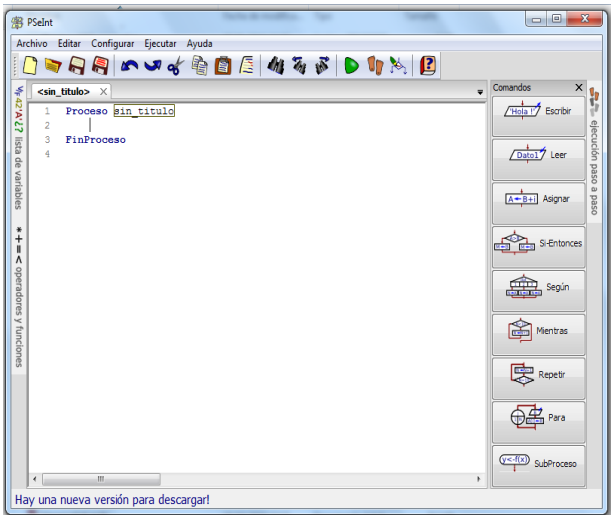
Programa para aprender seudocódigo y así entender de mejor manera lo que realiza un algoritmo, además muestra cómo se vería el seudocódigo en diagrama de flujo.

Un seudocódigo es una escritura de texto con símbolos especiales propios de cada lenguaje de programación al ser un algoritmo tiene un inicio y un fin y resuelve algún problema.

PSeInt no es un lenguaje de programación es decir, no crea programas; una vez de haber transcrito el algoritmo en seudocódigo, se le puede pasar al lenguaje de programación que se desee, PSeInt ayuda a entender la lógica de programación.

Figura 2.32.

Pantalla para escribir el seudocódigo.



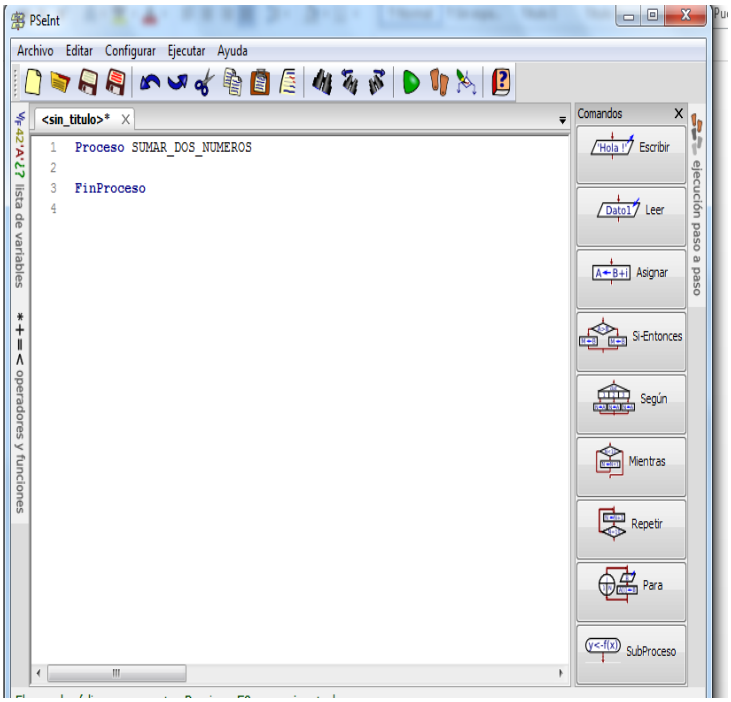
Nota. Los autores, PSeInt

Como es la representación del algoritmo ya el programa coloca el inicio y el fin del seudocódigo, en el inicio se colocará el nombre del seudocódigo que se va a escribir.

Para escribir el nombre del seudocódigo no se debe dejar espacios en blanco, si se desea separar palabras se utilizará un guión bajo(_).

Figura 2.33.

Seudocódigo con su respectivo nombre.



Nota. Los autores, PSeInt

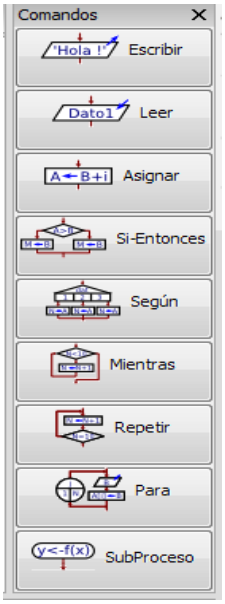
Comandos

El menú de comandos tiene herramientas que pueden ser utilizadas con un solo clic o si no se desea utilizar el menú, se puede escribir directamente el código.

El menú de comando da las siguientes opciones:

Figura 2.34.

Ventana de Comandos

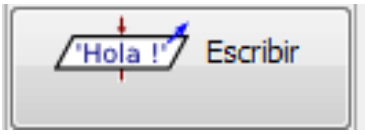


Nota. Los autores,PSeInt

Escribir: sirve para mostrar en pantalla un mensaje o un valor.

Figura 2.35.

Comando Escribir



Nota. Los autores, PSeInt

Para mostrar un texto en pantalla se debe colocar el texto entre comillas.

Figura 2.36.

Algoritmo de inicio

```
1 Algoritmo Mostrar_en_Pantalla
2       Escribir "Hola a todos"
3 FinAlgoritmo
```

Nota. Los autores, PSeInt

En caso de desear colocar un texto y el valor de una variable, el texto estará entre comillas y la variable o valor se colocará fuera de las comillas y separado por una coma seria de la siguiente forma.

Figura 2.37.

Algoritmo muestra valor de constante

```
1 Algoritmo Mostrar_en_Pantalla
2     Escribir "El valor de PI es: ",PI
3 FinAlgoritmo
```

Nota. Los autores, PSeInt

Otra forma:

Figura 2.38.

Otra forma de imprimir una constante

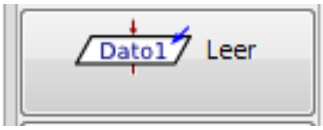
```
1 Algoritmo Mostrar_en_Pantalla
2     Escribir PI," Este es el valor de PI "
3 FinAlgoritmo
```

Nota. Los autores, PSeInt

Leer: recibe en una variable lo ingresado mediante teclado por el usuario.

Figura 2.39.

Comando Leer



Nota. Los autores, PSeInt

Por ejemplo:

Figura 2.40.

Algoritmo de lectura

```
1 Algoritmo Mostrar_en_Pantalla
2     Leer A
3 FinAlgoritmo
```

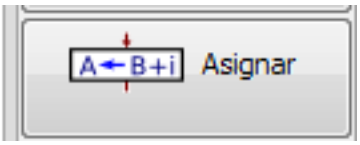
Nota. Los autores, PSeInt

Si el usuario ingresa el valor de 1 se guardará en la variable **A.**

Asignar: es el símbolo (=) de una ecuación representado a través de (<-).

Figura 2.41.

Comando Asignar



Nota. Los autores, PSeInt

Por ejemplo.

Figura 2.42.

Algoritmo de asignación

```
1  Algoritmo Mostrar_en_Pantalla
2      suma←5+4
3  FinAlgoritmo
```

Nota. Los autores, PSeInt

Al no ser un lenguaje de programación se puede configurar como flexible y poder utilizar tranquilamente el signo igual.

Figura 2.43.

Algoritmo Asignación con signo igual

```
1  Algoritmo Mostrar_en_Pantalla
2      Suma=5+4
3  FinAlgoritmo
```

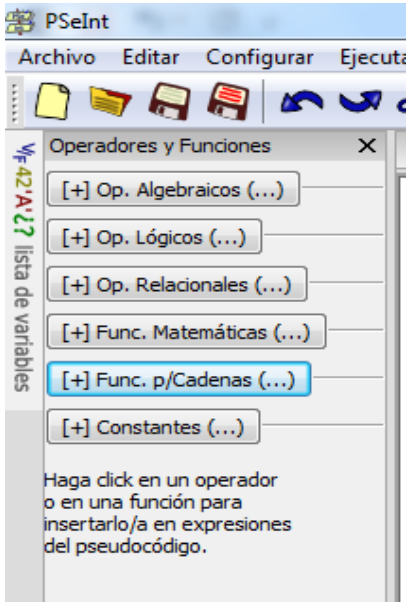
Nota. Los autores, PSeInt

Operadores y funciones

Es un menú que ayuda a colocar con un solo clic un operador o función si no se desea utilizar el menú se puede escribir directamente.

Figura 2.44.

Ventana Operadores y Funciones

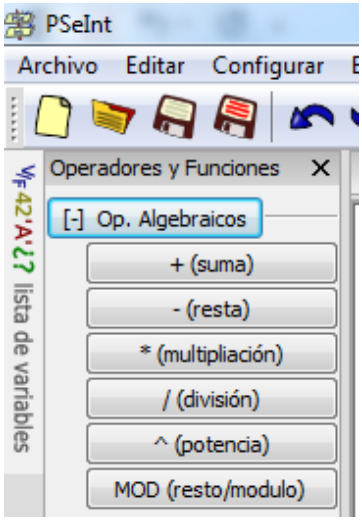


Nota. Los autores, PSeInt

Operador Algebraicos: contiene los operadores matemáticos comúnmente utilizados.

Figura 2.45.

Ventana Operadores Algebraicos



Nota. Los autores, PSeInt

Mod: El operador Mod ayuda a obtener el residuo de una división. Por ejemplo

Figura 2.46.

Aplicando módulo

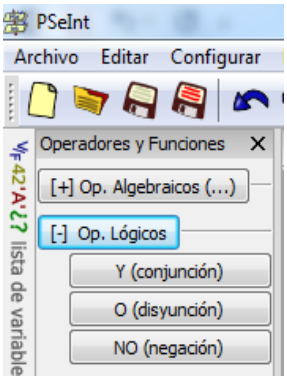
```
1 Algoritmo Ejemplo_mod
2     residuo = 4 MOD 5
3 FinAlgoritmo
```

Nota. Los autores, PSeInt

Operadores Lógicos: sirven para colocar restricciones.

Figura 2.47.

Ventana Operadores Lógicos

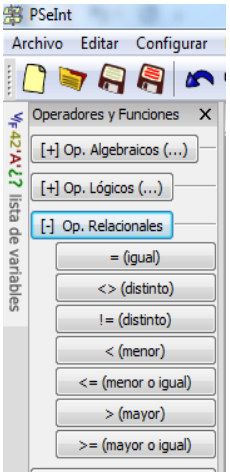


Nota. Los autores, PSeInt

Operadores Relacionales: sirven para realizar una pregunta o ejecutar una condición.

Figura 2.48.

Ventana Operadores Relacionales

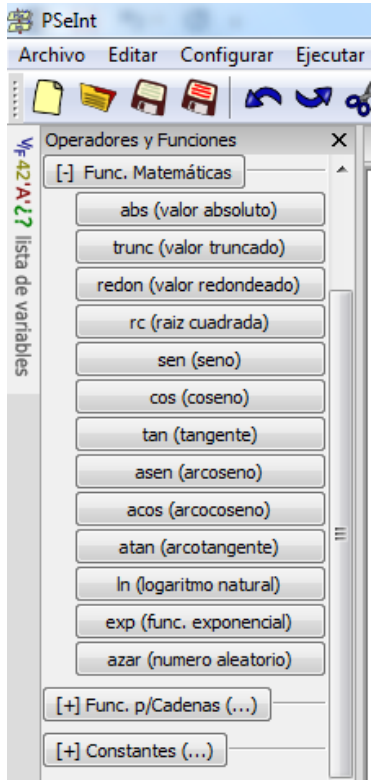


Nota. Los autores, PSeInt

Funciones Matemáticas: muestra las palabras claves que se utilizan para ejecutar una función matemática que se tenga que realizar durante el seudocódigo.

Figura 2.49.

Ventana Funciones Matemáticas



Nota. Los autores, PSeInt

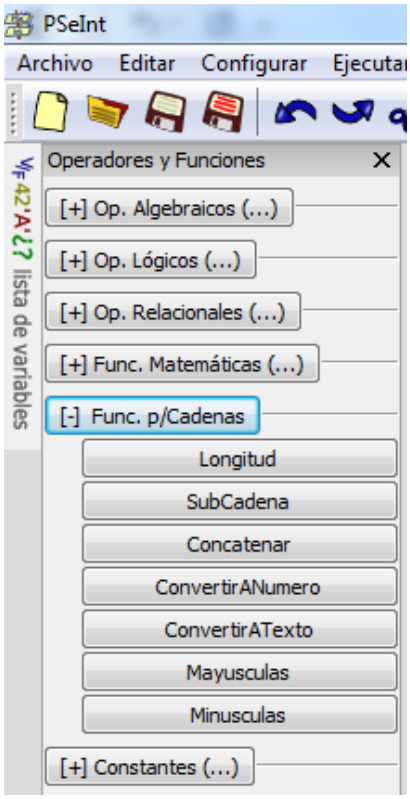
PSeInt ofrece la facilidad de que, si el programador no recuerda las funciones matemáticas, este las podrá seleccionar del menú de operadores y funciones, seleccionado del menú o

escribiéndola; por ejemplo,**Trunc** que sirve para seleccionar solo la parte entera de un número decimal, es una función muy útil y comúnmente utilizada por los programadores a la hora de trabajar con números.

Funciones para Cadenas: sirven para manipular un texto son funciones predefinidas que ayudan a manipular un texto fácilmente.

Figura 2.50.

Ventana Funciones para Cadenas

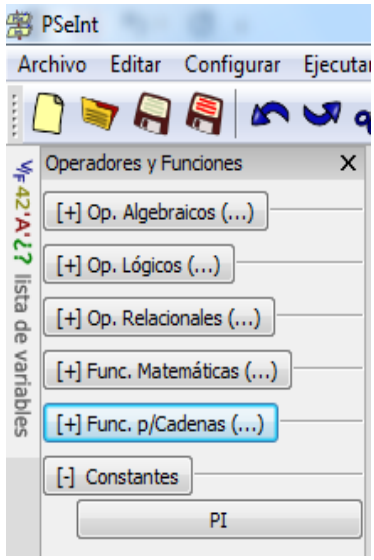


Nota. Los autores, PSeInt

Constante: es un valor ya definido en este caso PseInt, la única constante que facilita es el valor de PI.

Figura 2.51.

Ventana Constantes



Nota. Los autores, PSeInt

Declaración de variables

Figura 2.52.

Algoritmo de definición

```
1 Algoritmo Sumar_dos_números
2     Definir A,B Como Entero
3 FinAlgoritmo
```

Nota. Los autores, PSeInt

Como todo algoritmo se debe declarar la o las variables a utilizar en el desarrollo del programa. Se declara con la palabra **definir** y para indicar de qué tipo de dato es se utiliza la palabra **como**, el auto completado del programa le indicará las opciones de tipo de dato que podrá seleccionar.

Si en el idioma se coloca flexible no será necesario la declaración.

El texto de color azul son las palabras claves del programa, es decir palabras propias del sistema también conocidas como palabras reservadas.

En el ejemplo se coloca como nombre **SUMAR_DOS_NUMEROS**, y se declara las variables **A** y **B** de tipo entero, si existiera más variables del mismo tipo se utilizaría la coma para aumentar otra variable, si la variable no es de tipo entero sino decimal se colocaría otra declaración:

Figura 2.53.

Algoritmo con dos tipos de definiciones

```
1 Algoritmo Sumar_dos_números
2     Definir A, B Como Entero
3     Definir C Como Real
4 FinAlgoritmo
```

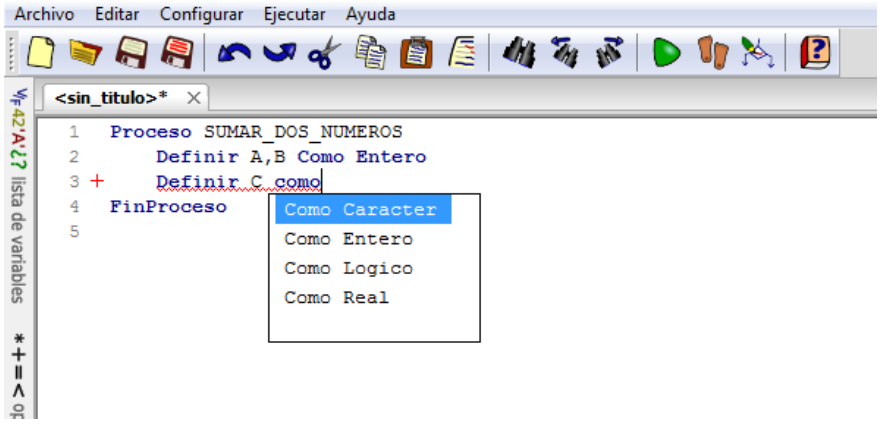
Nota. Los autores, PSeInt

Tipos de datos

En PSeInt existen solo 4 tipos de datos que se muestran a través de un menú para ser seleccionados o si no se lo desea solamente escribirlo.

Figura 2.54.

Menú con los tipos de datos



Nota. Los autores, PSeInt

Los tipos de datos de PSeInt son Caracter, Entero, Lógico y Real.

Carácter: representa un símbolo, número, letra o un texto debe ir entre comillas; ejemplo:

Figura 2.55.

Algoritmo manejo de caracteres

```
1 Algoritmo ejemplo_caracter
2     Definir C Como Caracter
3     c ←" mamá"
4 FinAlgoritmo
```

Nota. Los autores, PSeInt

Al declarar la variable **C** como carácter se le puede asignar la palabra “mamá” recordando siempre que el texto debe ir entre comillas.

Entero: serán todos los numero positivos, negativos y el cero sin punto decimal.

Figura 2.56.

Algoritmo manejo de enteros

```
1 Algoritmo ejemplo_entero
2     Definir C Como Entero
3     c ← 1563
4 FinAlgoritmo
```

Nota. Los autores, PSeInt

Lógico: será verdadero o falso.

Figura 2.57.

Algoritmo manejo de booleanos

```
1 Algoritmo ejemplo_lógico
2     Definir C Como Logico
3     c ← verdadero
4 FinAlgoritmo
```

Nota. Los autores, PSeInt

Real: serán todos los numero positivos, negativos, el cero con y sin punto decimal.

Figura 2.58.

Algoritmo manejo de reales

```
1 Algoritmo Ejemplo_Real
2     Definir C Como Real
3     c ← 5.67
4 FinAlgoritmo
```

Nota. Los autores, PSeInt

Constante: es un valor predefinido que no cambia a menos que el usuario lo realice internamente en el código. Ejemplo.

Figura 2.59.

Algoritmo manejo de constante

```
1  Algoritmo Ejemplo_de_constante
2      Definir constante como entero
3      constante←120
4  FinAlgoritmo
5
```

Nota. Los autores, PSeInt

En PSeInt el programa da la opción de colocar directamente la constante PI, solo se debe seleccionar del menú de Operadores y Funciones.

Figura 2.60.

Menú de Operadores y Funciones

Nota. Los autores, PSeInt

Ejemplo de seudocódigo:

Figura 2.61.

Seudocódigo Suma de dos números

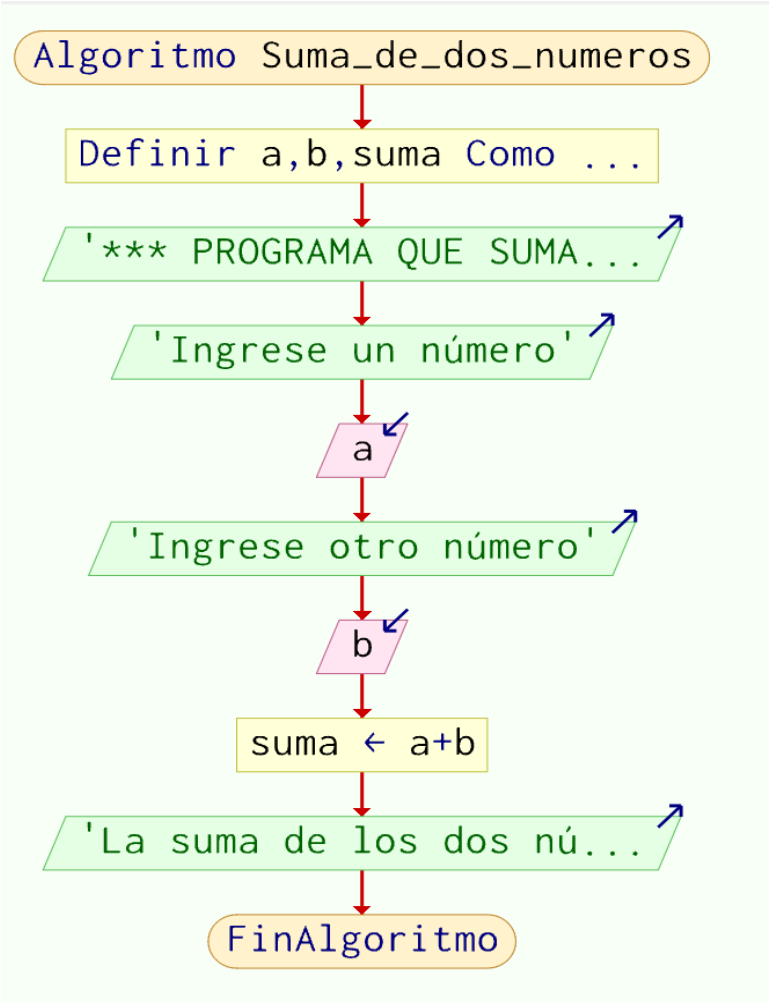
```
1  Algoritmo Suma_de_dos_numeros
2      definir A, B, suma como entero
3      Escribir "*** PROGRAMA QUE SUMA DOS NUMEROS ENTEROS INGRESADOS POR EL USUARIO ***"
4      Escribir "Ingrese un número"
5      Leer A
6      Escribir "Ingrese otro número"
7      Leer B
8      suma←a + b
9      Escribir "La suma de los dos números enteros son: ", suma
10 FinAlgoritmo
```

Nota. Los autores, PSeInt

El programa también facilita la representación del seudocódigo a diagrama de flujo como se muestra en la Figura 2.62.

Figura 2.62.

Diagrama de flujo PSeInt que suma dos números



Nota. Los autores, PSeInt

En el diagrama de flujo se puede observar flechas en las gráficas que representan entrada y salida, la flecha hacia fuera, representa salida o impresión por pantalla y la flecha hacia adentro, representa ingreso de información.

Condición Lógica

Una condición Lógica es realizar una pregunta con la que obtendrá una respuesta verdadera o falsa, para realizar la pregunta se utiliza el comando Si-Entonces.

Figura 2.63.

Comando Si-Entonces

Nota. Los autores, PSeInt

Ejemplo:

Seudocódigo que muestra en pantalla si un número ingresado por el usuario es positivo.

Figura 2.64.

Seudocódigo número positivo

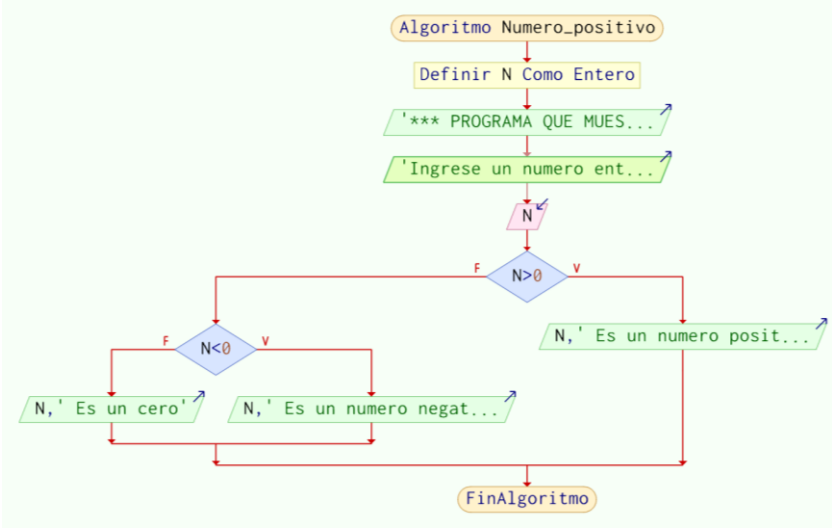
```
Algoritmo Numero_positivo
    definir N como entero
    Escribir "*** PROGRAMA QUE MUESTRA SI EL NUMERO INGRESADO ES POSITIVO ***
    Escribir "Ingrese un numero entero"
    Leer N
    Si N > 0 Entonces
        Escribir N," Es un numero positivo"
    Sino
        Si N < 0 Entonces
            Escribir N," Es un numero negativo"
        Sino
            Escribir N," Es un cero"
        Fin Si
    Fin Si

FinAlgoritmo
```

Nota. Los autores, PSeInt

Representación en diagrama de flujo:

Figura 2.65.

Diagrama de flujo en PSeInt número positivo



Nota. Los autores, PSeInt

Ciclos Repetitivos

En PSeInt existen 3 ciclos repetitivos el Mientras, Repetir y el Para.

Ciclo Mientras: es una instrucción con la que comienza con una condición, si la condición es verdadera, ingresa al ciclo caso contrario, se saldrá del mismo. El ciclo utiliza un contador, el que hay que incrementarlo en forma manual es decir, a través de líneas de programación lo que hace que el ciclo no sea automático.

Figura 2.66.

Comando Mientras

Nota. Los autores, PSeInt

Ciclo Repetir: es una instrucción con la que comienza con la ejecución de la instrucciones y al final pregunta por la condición, si la condición es falsa, regresa a ejecutar el ciclo, caso contrario se saldrá del mismo.

El ciclo al igual que el ciclo mientras, utiliza un contador, el mismo que hay que incrementarlo en forma manual; es decir, a través de líneas de seudocódigo lo que hace que el ciclo no sea automático.

Figura 2.67.

Comando Repetir

Nota. Los autores, PSeInt

Ciclo Para: es una instrucción que no necesita que se inicialice el contador fuera de la instrucción, la inicialización, la condición y el incremento o decremento según sea el caso se lo realiza en la misma línea de instrucción para de ahí continuar con el conjunto de códigos que tenga que realizar, es por eso que se lo conoce como ciclo automático.

El ciclo Para en PSeInt solo acepta valores enteros, no decimales ni caracteres o valores lógicos.

Figura 2.68.

Comando Para

Nota. Los autores, PSeInt

Ejemplo de ciclo Mientras, validar usuario y contraseña.

Figura 2.69.

Ejemplo de Mientras validar usuario

```
Algoritmo contraseña_ejemplo
    Definir usuario,contraseña Como Caracter
    Escribir "Bienvenidos al sistema"
    Escribir 'Ingrese el usuario'
    Leer usuario
    Escribir 'Ingrese la contraseña'
    Leer contraseña

    Mientras (usuario≠ pepito)=(contraseña≠"123") Hacer
        Escribir "ingrese usuario"
        Leer usuario
        Escribir "Ingrese contraseña"
        Leer contraseña
    FinMientras
    Escribir "El usuario y contraseña son correctos "

FinAlgoritmo
```

Nota. Los autores, PSeInt

Ejemplo de ciclo **Mientras**, el ciclo se repetirá siempre que el usuario digite el texto OK .

Figura 2.70.

Ejemplo de Mientras hasta que se digite OK

```
Algoritmo de_mientras
    Definir variable Como Caracter
    Definir contador como entero
    Escribir "Si desea incrementar el contador digite OK "
    Leer variable
    variable← Mayusculas(variable)
    Mientras variable = "OK" Hacer
        contador ← contador+1
        Escribir "Si desea continuar digite OK"
        Leer variable
        variable← Mayusculas(variable)
    Fin Mientras
    Escribir "El número de veces que se repitió el ciclo fue: ", contador
FinAlgoritmo
```

Nota. Los autores, PSeInt

Ejemplo de ciclo mientras, el ciclo se repetirá siempre que el contador sea menor que 100.

Figura 2.71.

Ejemplo de Mientras contador menor que 100

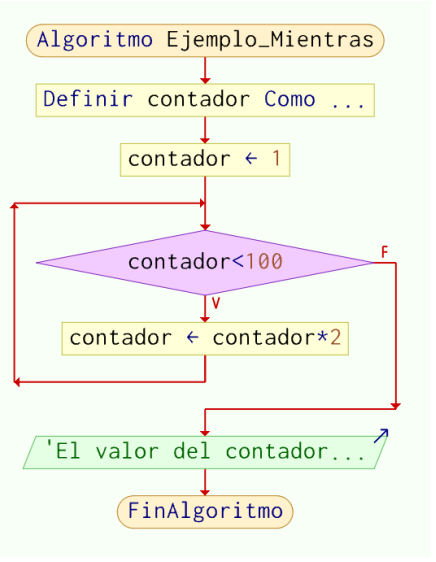
```
1  Algoritmo Mientras_ejemplo
2      Definir contador como entero
3      Contador←1
4      Mientras contador < 100 Hacer
5          contador ← contador*2
6      Fin Mientras
7      Escribir "El valor del contador es: ", contador
8  FinAlgoritmo
```

Nota. Los autores, PSeInt

Diagrama de flujo:

Figura 2.72.

Diagrama de flujo en PSeInt Mientras contador menor que 100



Nota. Los autores, PSeInt

Ejemplo de ciclo Repetir, el ciclo se repetirá siempre que el usuario digite el texto OK .

Figura 2.73.

Ejemplo ciclo Repetir hasta digitar OK

```
1  Proceso Ejemplo_Repetir
2      Definir variable Como Caracter
3      Definir contador como entero
4      Escribir "Si desea incrementar el contador digite OK "
5      Leer variable
6      variable← Mayusculas(variable)
7      Repetir
8          contador ← contador+1
9          Escribir "Si desea continuar digite OK"
10         Leer variable
11         variable← Mayusculas(variable)
12     Hasta Que variable ≠ "OK"
13     Escribir "El numero de veces que se repitio el ciclo fue : ", contador
14 FinProceso
```

Nota. Los autores, PSeInt

Ejemplo de ciclo repetir, el ciclo se repetirá siempre que el número ingresado por el usuario no sea mayor o igual a cero.

Figura 2.74.

Repetir hasta que se ingrese un número positivo

```
1  Algoritmo Repetir_numero
2      definir num, suma Como Entero
3      Repetir
4          Escribir "Ingrese un número positivo"
5          Leer num
6          Si num≥0 Entonces
7              Escribir 'ingrese,5,numeros'
8              suma =0
9              Para contador←1 Hasta 5 Con Paso 1 Hacer
10                 Escribir 'ingrese el numero',contador,' :
11                 Leer num
12                 suma = suma+num
13             FinPara
14             num=0
15         SiNo
16             Escribir "Ingrese un número positivo"
17             Leer num
18         Fin Si
19     Hasta Que num≥0
20     Escribir "La suma de los 5 números es ", suma
21 FinAlgoritmo
```

Nota. Los autores, PSeInt

En el ejemplo de la Figura 2.73, se solicita al usuario un número, si el número ingresado por el usuario es negativo el algoritmo solicitará otro número, si el número es positivo o igual a cero, se solicitará cinco números al usuario, se los sumará y a la vez se romperá el ciclo **mientras**, para proceder a mostrar la suma de los cinco números ingresados.

Ejemplo de ciclo repetir, el ciclo se repetirá siempre que el contador no sea mayor que 100.

Figura 2.75.

Ejemplo ciclo repetir contador no mayor a 100

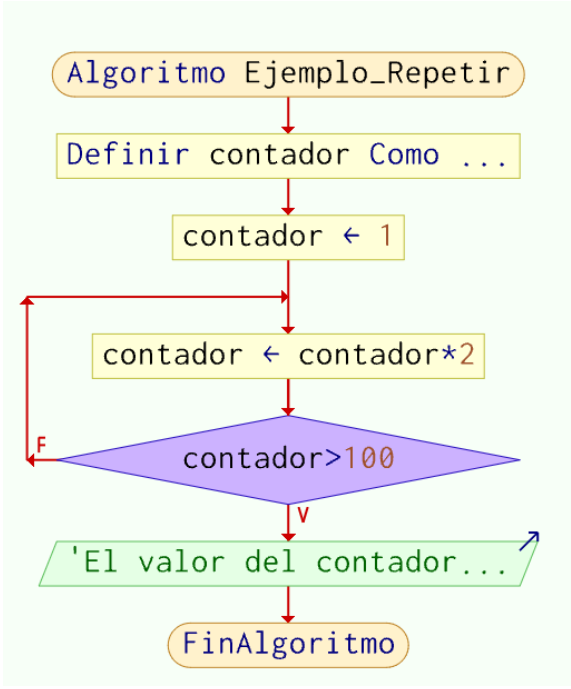
```
1  Proceso Ejemplo_Repetir
2      Definir contador como entero
3      Contador←1
4      Repetir
5          contador ← contador*2
6      Hasta Que contador > 100
7      Escribir "El valor del contador es : ", contador
8  FinProceso
```

Nota. Los autores, PSeInt

Diagrama de flujo:

Figura 2.76.

Diagrama de flujo en PSeInt ciclo repetir contador no mayor a 100



Nota. Los autores, PSeInt

Ejemplo de ciclo **Para**, el ciclo incrementa el contador en 2, finaliza cuando sea mayor que 100.

Figura 2.77.

Ejemplo ciclo Para incrementa contador en 2

```
1 Proceso Ejemplo_Para
2     Definir contador como entero
3     Para contador←2 Hasta 100 Con Paso 2 Hacer
4         Escribir "El valor del contador es: ", contador
5     Fin Para
6 FinProceso
```

Nota. Los autores,PSeInt

Ejemplo de ciclo **Para**, el ciclo quede decrementar el contador en 2, finaliza cuando sea igual a cero.

Figura 2.78.

Ejemplo ciclo Para contador decremento en 2

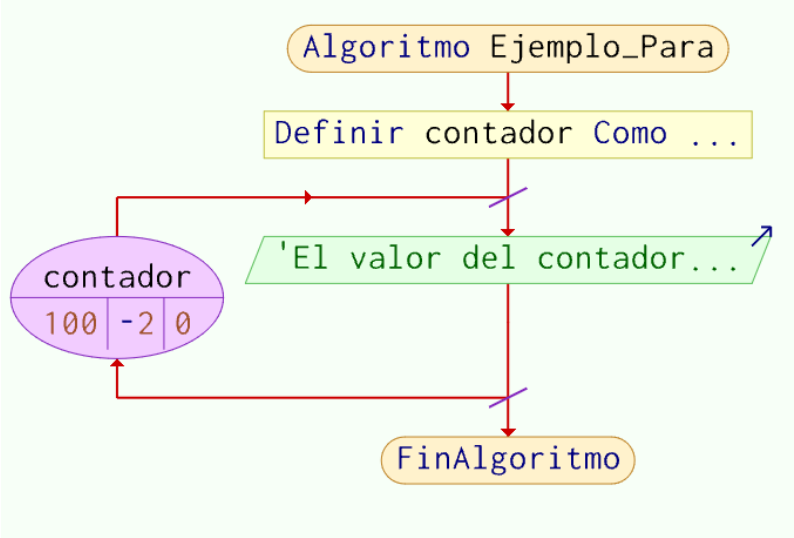
```
1 Proceso Ejemplo_Para
2     Definir contador como entero
3     Para contador←100 Hasta 0 Con Paso -2 Hacer
4         Escribir "El valor del contador es: ", contador
5     Fin Para
```

Nota. Los autores, PSeInt

Diagrama de flujo:

Figura 2.79.

Diagrama de flujo en PSeInt ciclo Para contador decrementa en 2



Nota. Los autores, PSeInt

En PSeInt existe una ligera modificación del ciclo **for** o **Para**, al colocar la gráfica hacia el lado derecho del diagrama, pero su función se mantiene. En el óvalo para representar el **Para** se identifica el **contador** indicando primero el límite al que tiene que llegar, seguido del decremento y finaliza indicando con qué valor inicia.

Ejemplo de ciclo **Para**, si el número ingresado por el usuario es primo.

Figura 2.80.

Ejemplo ciclo Para- número primo

```
Algoritmo numeros_primos
    Definir num1 como entero
    Escribir "Algoritmo que identifica si el numero ingresado es primo"
    Escribir "Ingrese un numero"
    leer num1
    suma=0
    Para contador←1 Hasta num1 Con Paso 1 Hacer
        Si (num1%contador)=0 Entonces
            suma=suma+1
        Fin Si
    Fin Para
    Si suma=2 Entonces
        Escribir "el numero es primo"
    SiNo
        Escribir "el numero no es primo"
    Fin Si

FinAlgoritmo
```

Nota. Los autores, PSeInt

Estructura de Selección

PSeInt solo ofrece una estructura de selección con el nombre de Según.

Según: es una estructura que presenta algunas opciones enumeradas, ejecutará una del grupo de instrucciones que haya escogido el usuario.

Se pueden agregar más opciones si es necesario, así como también se las puede eliminar.

Figura 2.81.

Comando Según

Nota. Los autores, PSeInt

Ejemplo de la instrucción:

Seudocódigo donde se muestra un menú para que el usuario escoja una de las opciones a través del ingreso de un número que se muestra en la pantalla.

Figura 2.82.

Ejemplo del uso de un Según de menú

```
Proceso Ejemplo_Segun
    Definir OP, CONTADOR como entero
    Definir variable como caracter
    Escribir "Escoja una de las opciones:"
    Escribir "1.- Números pares hasta el 100"
    Escribir "2.- Primera letra"
    Escribir "3.- Longitud del texto"
    Leer OP
    Segun OP Hacer
        1:          Para CONTADOR←0 Hasta 100 Con Paso 2 Hacer
                       Escribir CONTADOR
                    Fin Para
        2:  Escribir "Ingrese el texto"
            Leer variable
            Escribir SubCadena(variable,1,1)
        3:  Escribir "Ingrese el texto"
            Leer variable
            Escribir Longitud(variable)
        De Otro Modo:
            Escribir "No selecciono ninguna de las opciones"
    Fin Segun
FinProceso
```

Nota. Los autores, PSeInt

Otro ejemplo de seudocódigo aplicando según a través de opciones.

Figura 2.83.

Ejemplo Según menú de números

```
Algoritmo Menu
    Escribir "seleccione una opcion"
    Escribir "1.- Par-Inpar"
    Escribir "2.- Suma de los numeros"
    Escribir "3.- Resta de los numeros"
    Escribir "4.- Mayor de 3 numeros"
    Leer op

    Segun op Hacer
        1:
            Escribir " selecione la opcion Par-Inpar"
            Escribir "Ingrese el numero"
            Leer num1
            Si (num1)=0 Entonces
                Escribir "escribir numeroingresado es par"
            SiNo
                Escribir "El numero ingresado es inpar"
            FinSi
        2:
            Escribir "seleccione la suma de dos numeros"
            Escribir "ESCRIBIR EL PRIMER NUMERO"
            Leer a
            Escribir "INGRESE EL SEGUNDO NUMERO"
            Leer b
            c=a+b
            Escribir "EL RESULTADO DE LA SUMA ES = ",c
        3:
            Escribir "seleccione la suma de dos numeros"
            Escribir "ESCRIBIR EL PRIMER NUMERO"
            Leer a
            Escribir "INGRESE EL SEGUNDO NUMERO"
            Leer b
            c=a-b
            Escribir "EL RESULTADO DE LA SUMA ES = " ,c
        4:
            Escribir "seleccione la opcion el mayor de dos numeros"
            Escribir "INGRASA EL PRIMER NUMERO"
            Leer a
            Escribir "INGRESA EL SEGUNDO NUMERO"
            Leer b
            SI a>b Entonces
                Escribir "El mayor es " ,a
            SiNo
                Escribir "El mayor es " ,b
            FinSi

        De Otro Modo:
            Escribir "No selecciono ninguna opcion"

    Fin Segun
FinAlgoritmo
```

Nota. Los autores, PSeInt

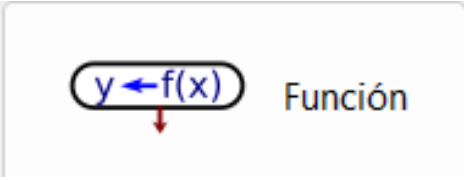
Función

Una función es el llamado a otro conjunto de instrucciones, una vez ejecutada las instrucciones detalladas en la función regresa al proceso original.

La función se ejecutará cada vez que se la llame.

Figura 2.84.

Comando Función



Nota. Los autores, PSeInt

La finalidad de una función es ejecutar una tarea específica, se puede generar varias funciones las mismas que pueden ser llamadas por el algoritmo principal o por otras funciones.

En la estructura de la función si devuelve algún valor, se colocará el nombre de la variable de retorno caso contrarió sólo se elimina ese paso, siempre se coloca el nombre de la función la opción de argumentos también es opcional, se describirá todos

los argumentos que sea necesarios, si la función no recibe argumentos se dejará los paréntesis con espacio en blanco.

Figura 2.85.

Estructura de una función

```
1  Funcion variable_de_retorno <- Nombre ( Argumentos )
2
3  Fin Funcion
4
```

Nota. Los autores, PSeInt

Ejemplo de Función. El ejercicio realizado en **Según** se lo pasa a **Función** colocando los tres casos que se pueden utilizar.

Figura 2.86.

Según con Función

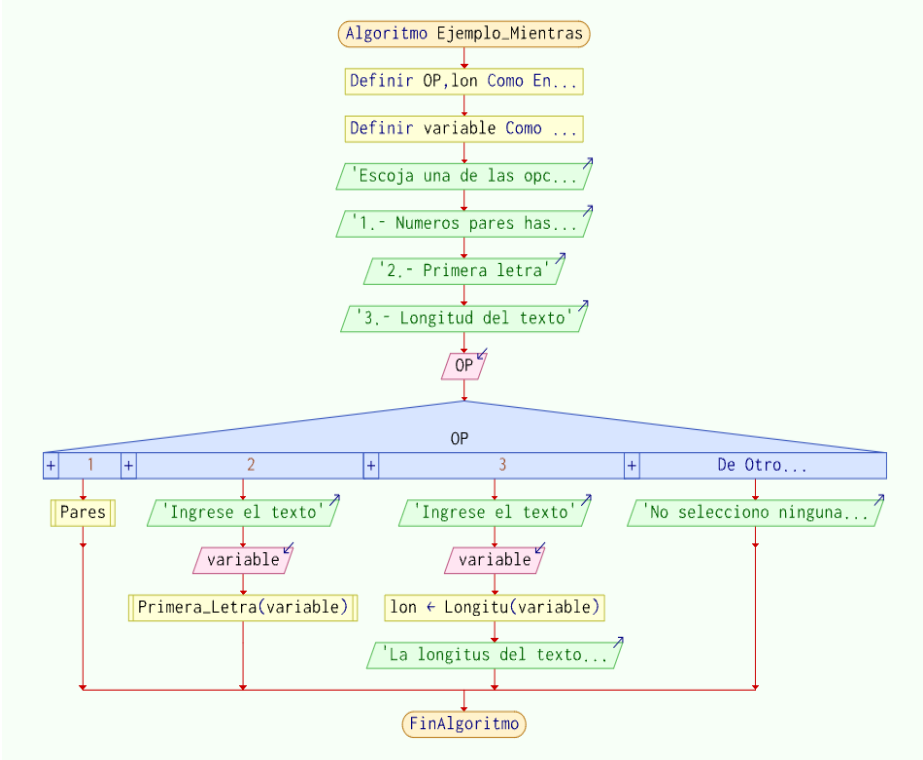
```
1
2  Funcion lon ← Longitu ( variable )
3          lon←Longitud(variable)
4  FinFuncion
5
6  Funcion Primera_Letra ( variable )
7          Escribir SubCadena(variable,1,1)
8  FinFuncion
9  Funcion   Pares
10         Para CONTADOR←0 Hasta 100 Con Paso 2 Hacer
11             Escribir CONTADOR
12         Fin Para
13 FinFuncion
14 Algoritmo  Ejemplo_Segun
15         Definir OP,lon como entero
16         Definir variable como caracter
17         Escribir "Escoja una de las opciones:"
18         Escribir "1.- Numeros pares hasta el 100"
19         Escribir "2.- Primera letra"
20         Escribir "3.- Longitud del texto"
21         Leer OP
22         Segun OP Hacer
23             1:
24                 Pares
25             2:
26                 Escribir "Ingrese el texto"
27                 Leer variable
28                 Primera_Letra(variable)
29             3:
30                 Escribir "Ingrese el texto"
31                 Leer variable
32                 lon← Longitu(variable)
33                 Escribir "La longitus del texto es: ", lon
34             De Otro Modo:
35                 Escribir "No selecciono ninguna de las opciones"
36         Fin Segun
37 FinAlgoritmo
38
```

Nota. Los autores, PSeInt

Diagrama de flujo:

Figura 2.87.

Diagrama de flujo en PSeInt - según con Función



Nota. Los autores, PSeInt

Existen tres formas de realizar un llamado a una función a continuación se mostrará cada una:

Primera Forma:

Figura 2.88.

Función con parámetro

```
2  Funcion lon ← Longitu ( variable )
3      lon←Longitud(variable)
4  FinFuncion
```

Nota. Los autores, PSeInt

La función tiene por nombre **Longitu** recibe como parámetro variable que contiene información que ha sido enviada del proceso principal. **Lon** es una variable que retorna al proceso principal con un valor.

Llamado:

Figura 2.89.

Llamado a una función

```
lon← Longitu(variable)
```

Nota. Los autores, PSeInt

En el algoritmo principal se tiene que realizar el llamado a la función para que se ejecute, además se envía un valor a la

función a través de **variable**, una vez ejecutado recibe en la variable **Lon** el valor que envía **Longitu**.

Segunda Forma:

Figura 2.90.

Otra forma de llamar a un Subproceso

```
6   Funcion Primera_Letra ( variable )
7       Escribir SubCadena(variable,1,1)
8   FinFuncion
```

Nota. Los autores, PSeInt

La función tiene por nombre **Primera_Letra** recibe como parámetro **variable** que contiene información que ha sido enviada del algoritmo principal. Muestra en pantalla el resultado y no devuelve nada al proceso principal.

Llamado:

Figura 2.91.

Llamado a Función

```
Primera_Letra(variable)
```

Nota. Los autores, PSeInt

En el algoritmo principal tiene que realizar el llamado a la función para que se ejecute, además se envía un valor a través de la **variable**, una vez ejecutado no recibe nada y continua con la ejecución de las instrucciones en el algoritmo principal.

Tercera Forma:

Figura 2.92.

Función pares

```
9   Funcion   Pares
10      Para CONTADOR←0 Hasta 100 Con Paso 2 Hacer
11          Escribir CONTADOR
12      Fin Para
13  FinFuncion
```

Nota. Los autores, PSeInt

La función tiene por nombre **Pares** no recibe parámetro. Muestra en pantalla el resultado y no devuelve nada al proceso principal.

Llamado:

Figura 2.93.

Llamado a subproceso sin parámetros

```
1:
    Pares
```

Nota. Los autores, PSeInt

En el algoritmo principal se tiene que realizar el llamado a la función para que se ejecute, una vez ejecutado no recibe nada y continua con la ejecución de las instrucciones en el algoritmo principal.

Vectores y matrices

Como se comentó en el primer capítulo, lo vectores y matrices se utilizan para el almacenamiento de varios valores en una sola variable, es decir, con el mismo identificador encontraremos varios valores.

En elemento de su implementación se maneja los índices, para representar la posición de los valores dentro de un vector, se utilizan los corchetes [ ] para indicar la dimensión o tamaño que va a contener ese arreglo, vector o matriz.

Las matrices o vectores son muy versátiles para contener valores en grandes cantidades, esto permite al programador poder gestionar dentro de su código varios cálculos y funcionalidades a partir de una sola variable.

Para poder inicializar una matriz se debe anteponer la palabra **Dimensión** y luego declarar el nombre o identificador de la estructura de almacenamiento, seguido por los corchetes y

dentro de estos el nombre de una variable común de tipo numérico, que indicará el tamaño o dimensión del vector o matriz.

Los vectores son estructuras de almacenamiento con mayor capacidad a diferencia de una variable.

Ejemplo de un vector con una dimensión definida por el usuario y que almacena nombres.

Figura 2.94.

Ejemplo de vector con nombres

```
Algoritmo Vector_nombre
    Escribir "Ingrese el numero de nombres que desee ingresar"
    Leer n
    Dimension nombres[n]
    Para contador= 1  Hasta n Con Paso 1 Hacer
        Escribir "Ingrese el nombre " , contador
        Leer nombre
        nombres[contador]=nombre
    Fin Para
    Para contador =1 Hasta n Con Paso 1 Hacer
        Escribir "El nomber almacenado en la poscicion " ,contador , " es:",nombres[contador]
    Fin Para

FinAlgoritmo
```

Nota. Los autores, Python

Como se muestra en la Figura 2.94, para manipular un vector se aconseja la utilización de ciclos como **for**, por la

facilidad que da al momento de manipular los índices del vector, pero también va a depender del seudocódigo que se desee implementar.

Ejemplo de un vector que almacena los elementos de otro vector pero en forma invertida.

Figura 2.95.

Ejemplo de vector invertido

```
Algoritmo Ejercicio
    Definir a,b,c,d,e Como Entero
    dimension A[5] , B[5]
    Para contador=1 Hasta 5 Con Paso 1 Hacer
        Escribir "Ingrese un numero"
        Leer  num
        A[contador]=num
    Fin Para
    I=5
    Para contador=1 Hasta 5 Con Paso 1 Hacer
        B[contador]=A[I]
        I=I-1
    Fin Para
    Para contador=1 Hasta 5 Con Paso 1 Hacer
        Escribir A[contador] , " y " , B[contador]
    Fin Para
FinAlgoritmo
```

Nota. Los autores, Python

A diferencia de los vectores que sólo manejan un índice, las matrices manejan dos índices, uno para las filas y otro para las columnas.

A igual que los vectores se recomienda su manipulación a través de los ciclos en especial el ciclo **for**, por la facilidad que presta en el momento de manipular las filas y las columnas. Por lo que se manejan dos contadores uno para que represente las filas y el otro las columnas.

Ejemplo de una matriz que almacena n elementos y es generada con n dimensiones definidas por el usuario.

Figura 2.96.

Ejemplo de matriz orden n

```
Algoritmo Matrices
    Escribir "Ingrese numero de filas"
    leer f
    Escribir "Ingrese numero de columnas"
    leer c
    Dimension  Matriz[f,c]
    Para fila=1 Hasta f Con Paso 1 Hacer
        Para columna=1 Hasta c Con Paso 1 Hacer
            Escribir "Ingrese un valor"
            leer a
            Matriz[fila,columna]=a
        Fin Para
    Fin Para
    Para fila=1 Hasta f Con Paso 1 Hacer
        Para columna=1 Hasta c Con Paso 1 Hacer
            Escribir " - ", Matriz[fila,columna]Sin Saltar
        Fin Para
        Escribir ""
    Fin Para
FinAlgoritmo
```

Nota. Los autores, Python

En la Figura 2.96, se observa dos contadores uno para las filas y otro para las columnas, además la instrucción **Sin Saltar**, indica que no dé un salto de línea y se mantenga en la misma hasta que se culminen todas las iteraciones del contador **columna**.

Ejemplo de suma de matrices.

Figura 2.97.

Ejemplo de suma de matrices

```
Algoritmo matrices2
    Escribir"Ingrese el numero de filas"
    Leer f
    dimension matriz[f,3]
    Para fila=1 Hasta f Con Paso 1 Hacer
        Para columna=1 Hasta 2 Con Paso 1 Hacer
            Escribir"Ingrese un valor",fila,",",columna
            leer matriz[fila,columna]
        Fin Para
    Fin Para
    Para fila=1 Hasta f Con Paso 1 Hacer
        Para columna=1 Hasta 2 Con Paso 1 Hacer
            Si columna=1 Entonces
                a=matriz[fila,columna]
            Fin Si
            Si columna=2 Entonces
                b=matriz[fila,columna]
            Fin Si
        Fin Para
        matriz[fila,3]=a+b
    Fin Para
    Para fila=1 Hasta f Con Paso 1 Hacer
        Escribir matriz[fila,1],"+",matriz[fila,2],"=",matriz[fila,3] sin saltar
        Escribir""
    Fin Para
FinAlgoritmo
```

Nota. Los autores, Python

En el ejemplo de la Figura 2.97, se realiza el llenado de ambas matrices para después proceder a sumarlas; en el ejemplo se demuestra que se puede trabajar con valores estáticos, es decir que no se modifican durante la ejecución del programa, lo cual también es válido, pues siempre todo dependerá del requerimiento que se solicite para el seudocódigo.

Ejercicios Propuestos

1. Realizar un diagrama de flujo que lea dos números y muestre cual es el mayor.

2. Realizar un Diagrama de flujo que resuelva la fórmula de Pitágoras.

3. Realizar un Diagrama de flujo que calcule el promedio de n notas ingresadas por el usuario.

4. Realizar un seudocódigo que lea tres números y muestre cual es el mayor, el intermedio y el menor.

5. Realizar un seudocódigo que muestre si el número ingresado por el usuario es divisible para cuatro.

6. Realizar una función que muestre los primeros 10 números pares a raíz del número ingresado por el usuario. Ej: si ingresa 15 será del 16 hasta 34, si ingresa 16 será hasta el 34.

7. Realizar un vector que almacene una lista de productos y otro vector que almacene el precio de dichos productos, mostrar el producto con su respectivo precio, la dimensión será generada por el usuario.

8. Realizar un vector con tres funciones una para inicializar el vector otra para llenar el vector para mostrar el vector.

9. Realizar la transpuesta de una matriz.

10. Generar un mes del calendario con matrices.

CAPÍTULO 3
GENERANDO CÓDIGO

Capítulo 3

Generando código

3. Programación estructurada

El objetivo del Capítulo 3, familiarizar al lector con un lenguaje de programación, de esta manera brindar habilidades técnicas lógicas, que permitan solucionar problemas a través de la implementación de código y convertirlos en programas computacionales simples para tareas específicas.

A finales de los años 1970 surgió una nueva forma de programar que no solamente daba lugar a programas fiables y eficientes, sino que además estaban escritos de manera que facilitaba su mejor comprensión, no sólo proveyendo ventajas durante la fase de desarrollo, sino también posibilitando una más sencilla modificación posterior.

Es un enfoque de diseño de código fuente para un programa basado en una estructura lógica y organizado para mejorar la claridad, mantenibilidad y eficiencia.

La programación estructura consiste en implementar código fuente basado en características como: secuencia de instrucciones que se ejecutan una después de otra y en orden, estructuras de selección para controlar la toma de decisiones basada en una condición (*if, else, switch*, entre otras.), interacciones o ciclos de repetición que son utilizadas frecuentemente para repetir ciertas secuencias de instrucciones que requieran ser repetidas en base a una condición determinada, modularidad mediante el uso de funciones/procedimientos y abstracción mediante el uso de variables y constantes con el propósito de hacer un código más legible y mantenible.

De esta manera se tiene un programa que ejecuta un conjunto de código secuencialmente.

Es una programación muy sencilla, pero al seguir una secuencia hace que para programas extensos se complique seguir con la ruta del mismo ya que se debe ir paso a paso el código descrito.

Las habilidades de programar se van adquiriendo con el tiempo a través de la práctica, la programación es como las matemáticas, mientras más ejercicios realice, más destrezas irá adquiriendo. Es una práctica que no debe estresar, lo que el futuro programador debe adquirir es paciencia y ser ordenado.

Lo que se debe hacer antes de generar código es conocer el problema, entenderlo e identificar una posible solución, recordando que programar es generar código, este código proviene de un algoritmo y que los algoritmos resuelven un problema, sino entiende el problema no podrá programar.

3.1. PYTHON

"Python es un lenguaje de programación potente y fácil de aprender. Tiene estructuras de datos de alto nivel eficientes y un enfoque simple pero efectivo para la programación orientada a objetos".(Python Software Foundation,2023, párr. 1)

Python es un lenguaje de programación multiparadigma (programación orientada a objetos, programación imperativa y programación funcional). Se lo puede aplicar en diferentes plataformas, es muy útil para el análisis de datos y se puede conectar con base de datos.

Python en la inteligencia artificial es un lenguaje muy adecuado pues maneja grandes cantidades de solicitudes por lo que tiene una gran potencia. Además, es un lenguaje de programación de código abierto, es interpretado primero por un *bytecode* para luego por el intérprete.

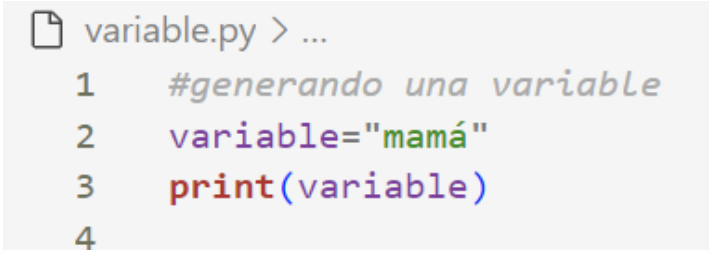
3.2. Variables

Son espacios reservados en memoria para almacenar un valor, son considerados como contenedores de valores que pueden cambiar de acuerdo a la naturaleza de la ejecución de un determinado programa. En *Python* no es necesario declarar una variable con asignarle el valor es suficiente para utilizarla. A estas acciones en *Python* se la conoce como “tipado dinámico”, esto significa que una variable puede contener un tipo de dato automático según el valor que el usuario a ingresado.

En *Python* se puede generar la variable en cualquier lugar del código. Cuando se implemente una variable, esta se inicializará con el tipo de dato especificado, por ejemplo:

Figura 3.1.

Generando una variable

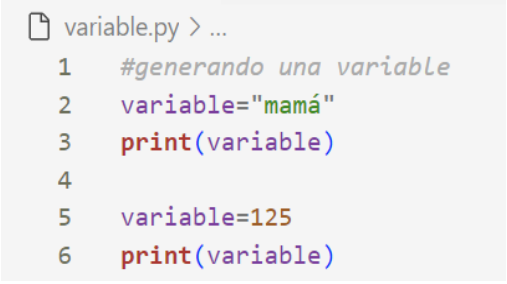


Nota. Los autores, Python

Las variables en *Python* tomarán el tipo de dato especificado es decir, a la misma variable se le puede asignar un valor.

Figura 3.2.

Utilizando la misma variable



Nota. Los autores, Python

En la Figura 3.2, se observa que se está utilizando la misma variable con diferentes tipos de datos, que es una de las facilidades que ofrece *Python*, al momento de tener extensas líneas de código y poder utilizar variables ya generadas con facilidad.

3.3. Constantes

El concepto **constante** no es utilizado en *Python* como en otros lenguajes de programación que aseguran que el valor no va a cambiar en la ejecución del programa. Pero se puede simular que una variable va a ser constante nombrándola con letras mayúsculas.

Al momento de generar código, se necesita realizar operaciones aritméticas, por lo que en la Tabla 3.1, se muestra los operadores aritméticos que utiliza *Python.*

Tabla 3.1.

Operadores Aritméticos.

Signo	Descripción
+	Suma
-	Resta
*	Multiplicación
/	División
//	Divide y da el resultado la parte entera
%	Devuelve el residuo de la división
**	Potencia

Nota. Los autores, Python

Para proceder en *Python* a utilizar una constante generada por el usuario se diferenciará con la variable utilizando letras mayúsculas.

Figura 3.3.

Constantes en Python

```
1   # Definir una "constante"
2   PI = 3.14159
```

Nota. Los autores, Python

3.4. Tipos de datos que utiliza Python

Al ser un lenguaje de programación de “tipado dinámico”, *Python* infiere automáticamente el tipo de datos de una variable en función del valor que se le asigna.

En la tabla 3.2, se muestran los tipos de datos más comunes del *software*.

Tabla 3.2.

Tipos de datos

Dato	Escritura
Entero	Int
Decimal	Float
Complejo	Complex
Cadena	Str
Booleano	Bool

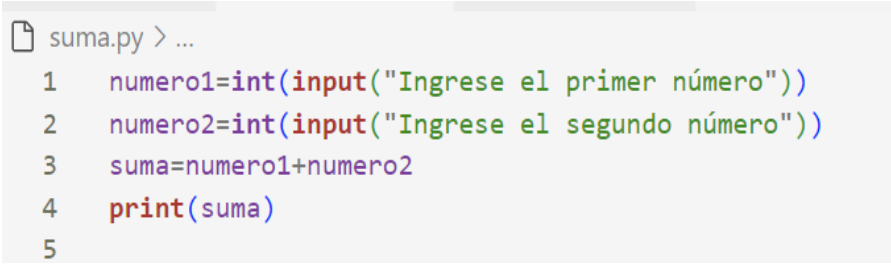
Nota. Los autores

Para poder obtener información del usuario se utiliza la función **input** que es para entrada y salida de datos de tipo texto y para mostrar la información la función **print** que es para salida de información.

Ejemplo, suma de dos números en *Python*.

Figura 3.4.

Operaciones



Nota. Los autores, Python

En la Figura 3.4, se observa que se ha utilizado tres variables **numero1** y **numero2** que reciben la información ingresada por el usuario y la variable **suma** que recibe el resultado de la operación de las variables anteriores.

En *Python* como en todo lenguaje de programación se utiliza los condicionales, que pueden ser simples o concatenados.

3.5. Condicional

En *Python* un condicional empezará con la sentencia **if** siempre las palabras claves se escriben con minúsculas y para indicar que una estructura de código pertenece a la sentencia se debe realizar la tabulación respectiva.

Ejemplo de una sentencia condicional simple:

Figura 3.5.

Sentencia if

simple.py > ...

```
1   contraseña=input("Ingrese la contraseña")
2   if contraseña=="Robot":
3       print("Bienvenido al sistema")
4
```

Nota. Los autores, Python

Para realizar un condicional concatenado se puede utilizar la sentencia **else**, que se aplicará cuando la sentencia **if** no sea verdadera.

Ejemplo: mostrar si un número es positivo.

Figura 3.6.

Sentencia if-else

positivo.py > ...

```
1   numero=int(input("Ingrese el número"))
2   if numero>0:
3       print(f"El número {numero} es positivo")
4   else:
5       print(f"El número {numero} es negativo")
6
```

Nota. Los autores, Python

Pero Python ofrece mayor facilidad a la hora de trabajar con condicionales como es la sentencia **elif,** sirve para comprobar múltiples condiciones de las cuales ingresará a las que sean válidas (verdaderas).

Ejemplo de múltiple condición:

Figura 3.7.

Sentencia elif

multiple.py > ...

```
1   edad=int(input("Ingrese la edad"))
2   if edad>18:
3       print(f"Usted es mayor de eadad, por tener {edad}")
4   elif edad<18 and edad >1:
5       print(f"Usted es menor de eadad, por tener {edad}")
6   else:
7       print("Todavia no cumple años")
```

Nota. Los autores, Python

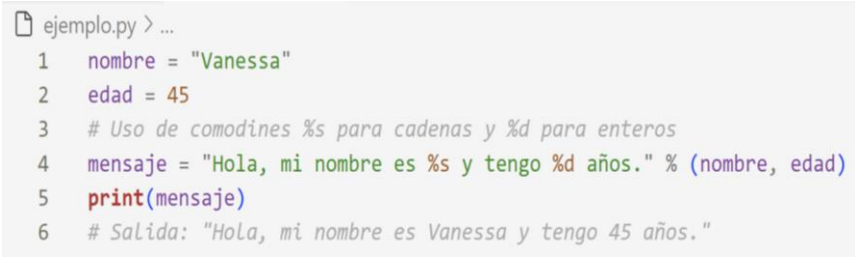
Los comodines se utilizan en combinación con la función **print** para formatear y mostrar datos en la salida. Los comodines son marcadores de posición que permiten insertar valores variables dentro de una cadena formateada. Hay varias formas de lograr esto en *Python*:

1.- Formateo con “%”: este comodín está dispuesto para formatear un determinado tipo de datos de una variable, de esta

manera se puede indicar al compilador que imprima junto a la cadena de salida las variables que fueron mentada en la línea de código. Ejemplo:

Figura 3.8.

Formato %



Nota. Los autores, Python

2.- Formato “format()”: también es considerado como un comodín, pero mejorado ya que no se requiere indicarle que tipo de datos se va a imprimir, para poder utilizar este comodín deberá utilizar dentro de la cadena las llaves { } donde se requiera insertar la variable.

Figura 3.9.

Formato format

Nota. Los autores, Python

3.- Formato f (f-strings): Es la forma más actual de formatear cadenas para imprimir y funciona de la misma manera que el formato **format,** a diferencia que lo único que se requiere es anteponer la letra **f** antes de la cadena a imprimir y colocar los nombres de las variables dentro de las llaves.

Figura 3.10.

Formato f

ejemplo.py > ...

```
1   nombre = "Vanessa"
2   edad = 45
3   # Uso del formato format()
4   mensaje = f"Hola, mi nombre es {nombre} y tengo {edad} años."
5   print(mensaje)
6   # Salida: "Hola, mi nombre es Vanessa y tengo   45 años."
```

Nota. Los autores, Python

Al utilizar **f** dentro de un **print** o de **input** se puede combinar el texto con la variable u operación que se desee colocar entre las llaves incluidas en la cadena de impresión. Los operadores lógicos en Python son **and, or** y **not** que pueden ser utilizados en condicionales simples o concatenados.

A partir de *Python* 3.10 podrá realizar estructuras selectivas **match** que resultan de mayor facilidad para cuando se requiere realizar un menú.

3.6. Sentencias repetitivas

En Python existen varias estructuras de repetición que sirven para provocar iteraciones o repeticiones en uno o varios bloques de código en el programa. Estas estructuras son útiles cuando se necesita realizar operaciones muchas veces o iterar sobre elementos de una secuencia.

En este punto es importante indicar el cuidado que hay que tener al momento de usar bucles **while**, ya que si la condición nunca se vuelve **False**, el bucle continuará ejecutándose indefinidamente, lo que puede llevar a un bucle infinito y hacer que el programa se bloquee.

Tabla 3.3.

Ciclos en Python

Sentencia Psint	Sentencia python
Mientras	While
Para	For

Nota. Los autores

Ejemplo de un menú con **while**:

Figura 3.11.

Sentencia match y while

menuw.py > ...

```
1  print("1.- Calcular la edad")
2  print("2.- Tabla de cualquier número")
3  op=int(input("Seleccione un opción"))
4  match op:
5      case 1:
6          fecha1=int(input("Ingrese su fecha de nacimiento"))
7          print(f" Su edad es {2023-fecha1}")
8      case 2:
9          num=int(input("Ingrese un número"))
10         i=1
11         while i<=12:
12             print(i, " X " ,num ," = ", i*num )
13             i=i+1
```

Nota. Los autores, Python

Como se muestra en la Figura 3.11, el **while** mantiene la lógica de sus tres líneas de acción: inicializar el contador, consultar por el contador que este sea verdadero y, modificar el contador, para controlar que no resulte un ciclo infinito.

Ejemplo de un ciclo implementado con **for**.

Figura 3.12.

Sentencia for recorre cadena

contar.py > ...

```
1  cadena=input("Ingrese el texto")
2  contador=0
3  for i in cadena:
4      contador=contador+1
5  print("El texto tiene ",contador )
```

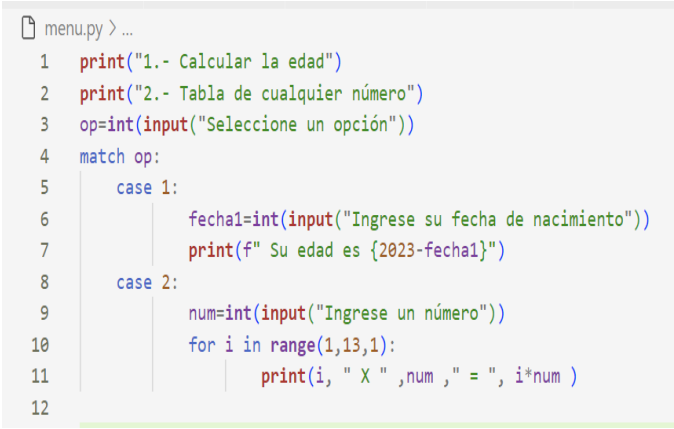
Nota. Los autores, Python

En el ejemplo de la Figura 3.12, el **for** va tomando carácter por carácter, con lo que maneja la cadena como una lista de caracteres, tema que se tratará más adelante en este capítulo.

Ejemplo de un menú con **for**:

Figura 3.13.

Sentencia match – for



Nota. Los autores, Python

En el caso de la Figura 3.13, se utiliza para el ciclo **for** que utiliza una función **range** que, lo que hace es generar una secuencia de números.

Range es una función que genera una secuencia de números enteros para realizar iteraciones controladas más eficientes sin necesidad de almacenarlos en memoria.

Tabla 3.4.

Algunos usos de range

Función	Explicación
range(5)	Iniciará un conteo desde el cero hasta el cuatro.
range(1,5,1)	Iniciará un conteo desde el uno, con un incremento de uno hasta el cuatro.
range(5+1)	Iniciará en cero, con un incremento de uno hasta el cinco.

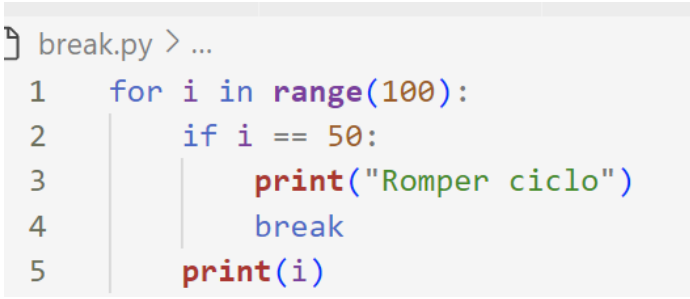
Nota. Los autores

3.7. Romper ciclos

Existen ocasiones que, como programadores, se necesita romper el ciclo ya sea este un **while** o un **for,** para este tipo de casos**,** *Python* ofrece dos alternativas el **break**, **continue** y el **pass**.

Figura 3.14.

Sentencia for con BREAK

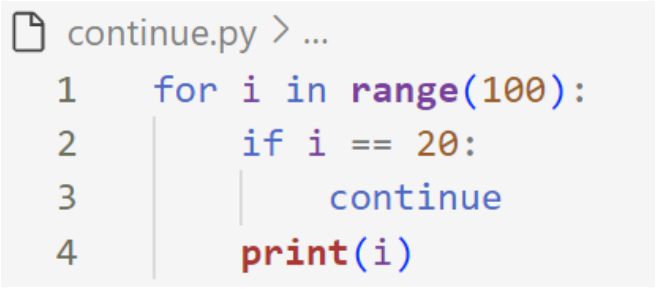


Nota. Los autores, Python

En la Figura 3.14, se puede observar que el ciclo va a ir mostrando lo valores que va tomando la variable **i**, esto lo repetirá hasta que la variable tome el valor de 50, cuando tome el valor ejecutará el **break**, por lo que saldrá del condicional y del ciclo con lo que finalizará el programa.

Figura 3.15.

Sentencia for con CONTINUE

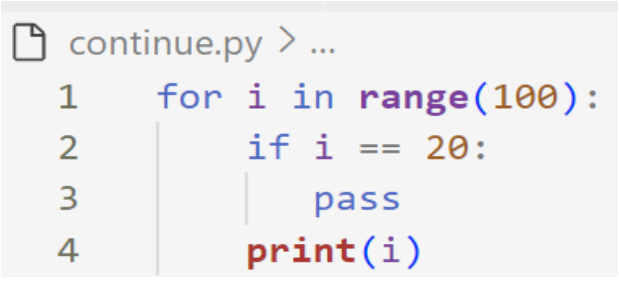


Nota. Los autores, Python

En la Figura 3.15 en la sentencia **if** se coloca el **continue**, que una vez ejecutado el programa mostrará todos los valores menos el 3, pero continuará con el ciclo.

Figura 3.16.

Sentencia for con PASS



Nota. Los autores, Python

En la Figura 3.16, dentro de la instrucción **if** se coloca la instrucción **pass,** la cual permite que el ciclo continúe y muestre todos los valores que inicialmente se le ha establecido.

3.8. Principales funciones de uso

Cuando se trabaja con cálculos resulta necesario el uso de funciones que facilitan este tipo de cálculos. A continuación se mostrarán las más conocidas:

Tabla 3.5.

Funciones

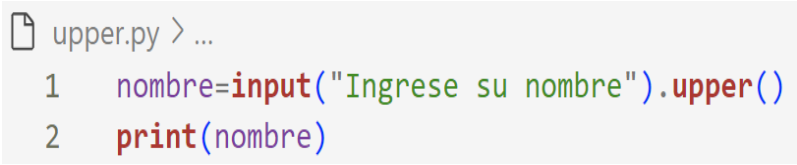
Estrcutura	Uso
abs	Valor absoluto
round	Redondear
n!	Factorial
exp	Exponencial
log	Logaritmo neperiano
max	Máximo
min	Mínimo
sqrt	Raíz cuadrada
random	Números aleatorios
upper	Cambia el texto a mayúscula.
lower	Cambia el texto a minúscula
title	Primera letra de cada palabra la transforma a mayúscula.

Nota. Los autores

Por ejemplo:

Figura 3.17.

Función upper



Nota. Los autores, Python

3.9. Funciones

Una función es un bloque de código que permite realizar una actividad o tarea específica de acuerdo al entorno en la que se implementó, estos bloques funcionales pueden ser utilizados en cualquier instancia o parte del programa según la necesidad del mismo.

Como en cualquier lenguaje de programación, una función va a resolver algo en concreto. Para declarar una función en *Python* se utiliza la palabra reservada **def** acompañado de su nombre y como toda estructura tendrá la respectiva tabulación del bloque que contiene.

Figura 3.18.

Declaración de una función

```
ejemplo.py > ...
1   def nombre_de_la_funcion(parametros):
2       # Cuerpo de la función
3       return resultado # Devolver un valor
```

Nota. Los autores, Python

Las funciones fueron creadas bajo el concepto divide y vencerás, lo cual ayuda a la programación estructurada en revisar código más sencillo y dio paso a la reutilización de código.

Python ofrece muchas funciones sencillas que pueden ayudar a resolver un problema concreto, pero en este libro la idea es aprender a programar por lo que las funciones serán generadas.

Recordando conceptos:

Una función o un procedimiento pueden tener o no argumentos, pero una función siempre retornará algo, en los casos de los procedimientos estos sólo se ejecutan.

Ejemplo generar una función(procedimiento) que invierta un número de dos cifras:

Figura 3.19.

Función (procedimiento)

fun.py > ...

```
1   def invertir (numero):
2       if numero < 10 or numero > 99:
3           return "El número debe contener dos cifras."
4       else:
5           num1 = numero % 10
6           numero = numero // 10
7           invertido = (num1*10) + numero
8       print (f"El numero invertido es:{invertido} ")
9
10
11  numero = int(input("Ingrese una cifra de dos números: "))
12  invertir(numero)
```

Nota. Los autores, Python

En el ejemplo de la Figura 3.19, la función hace las veces de un procedimiento, pues no devuelve ningún valor sólo se ejecuta.

Ejemplo: generar una función que invierta un número de dos cifras:

Figura 3.20.

Función invertir

fun.py > ...

```
1   def invertir (numero):
2       if numero < 10 or numero > 99:
3           return "El número debe contener dos cifras."
4       else:
5           num1 = numero % 10
6           numero = numero // 10
7           invertido = (num1*10) + numero
8
9       return invertido
10
11  numero = int(input("Ingrese una cifra de dos números: "))
12  print (f"El numero invertido es:{invertir(numero)} ")
```

Nota. Los autores, Python

En el ejemplo de la Figura 3.20, se implementa la función la función **return**, que a través de la función **return,** retorna el número invertido.

En *Python* tanto una función como un procedimiento, se los nombra con la palabra clave **def,** pero existen otros tipos de lenguajes de programación que si realizan esta diferencia con distintas palabras calves, por lo que es importante recordar conceptos tratados en unidades anteriores, pues como objetivo de este libro es enseñar a programar y que estos conceptos le sirvan al lector en cualquier lenguaje que se proponga a aprender.

Ejemplo generar dos funciones que serán llamadas a través de un menú:

Figura 3.21.

Funciones edad y tabla

funcion.py > ...

```
1   def edad():
2       fecha1=int(input("Ingrese su fecha de nacimiento"))
3       return(2023-fecha1)
4
5   def tabla():
6       num=int(input("Ingrese un número"))
7       for i in range(1,13,1):
8           print(i, " X " ,num ," = ", i*num )
9
10
11  print("1.- Calcular la edad")
12  print("2.- Tabla de cualquier número")
13  op=int(input("Seleccione un opción"))
14  match op:
15      case 1:
16              print(f" Su edad es {edad()}")
17
18      case 2:
19              tabla()
```

Nota. Los autores, Python

En la Figura 3.21, se muestra dos ejemplos de funciones, la primera función **edad** no recibe argumentos, pero lo interesante es que devuelve a través de la función **return** la edad calculada en la función a donde fue llamada. La segunda función **tabla** tampoco recibe argumentos, la diferencia está en que esta función trabaja como un procedimiento ya que sólo se ejecuta y no retorna nada.

Una función puede llamar a otra función ya sea sólo ejecutándose o retornando.

Figura 3.22.

Llamado de una función a través de otra función

Nota. Los autores, Python

En la Figura 3.22, se han realizado dos funciones, la función **mostrar** recibe como argumento la **edad** ingresada por el usuario y esta a su vez llama a la función **repetir** que hace las veces de un procedimiento, pues sólo se ejecuta y no devuelve nada.

La función **repetir** mostrará todos los años que ha cumplido el usuario hasta su edad actual, una vez realizado su trabajo retornará a la función **mostrar** quien retornará la edad que recibió como argumento, asignándolo a la variable **num** para que el programa principal proceda a mostrarla en pantalla. La función **return** siempre devolverá un valor, si en el caso no se desea devolver ningún valor devolverá el valor predeterminado None.

En *Python*, las bibliotecas o librerías son colecciones de módulos y funciones predefinidas que facilitan y amplían las capacidades del lenguaje. Estas librerías están diseñadas para abordar tareas específicas y proporcionar funcionalidades adicionales que no están incluidas en el núcleo del lenguaje.

Ejemplo de solución de la fórmula general con funciones en Python:

Figura 3.23.

Ejemplo de solución de la fórmula general

formula.py > ...

```python
import math

def formula(a,b,c):
    d=(b**2)-(4*a*c)
   # print(d)
    if d>=0:
        if d>0:
            d=pow( d, 1/2)
            x1=(-b+d)/(2*a)
            x2=(-b-d)/(2*a)
            return(x1,x2)
        else:
            d=pow( d, 1/2)
            x=(-b+d)/(2*a)
            return(x)
    else:
        print("No se tiene raices negativas")

a=int(input("Ingrese a"))
b=int(input("Ingrese b"))
c=int(input("Ingrese c"))
if a!=0:
    print("El valor de las raiz/raices es ",formula(a,b,c))
else:
    print("No existen divisiones para cero")
```

Nota. Los autores, Python

En la Figura 3.23, se ha generado la función **formula**, que recibe tres argumentos y en caso de existir raíces retornará las raíces correspondientes, se ha procedido a realizar un **import** de la librería **math.**

Math es una librería que incorpora funciones matemáticas, en el caso del ejemplo se la importa para poder usar **pow** y poder obtener la raíz cuadrada.

Ejemplo del juego piedra papel o tijera importa la librería **random:**

Figura 3.24.

Import random

ejemplo.py > ...

```
1   import random
2   compu=random.randint(1,3)
3   print("JUEGO PIEDRA PAPEL O TIJERA")
4   match compu:
5       case 1:
6               usuario=input("Escoja una opción (Piedra/Papel/Tijera)")
7               if usuario=="Piedra":
8                       print("Empate")
9               if usuario=="Papel":
10                      print("Gana usuario")
11              if usuario=="Tijera":
12                      print("Gana computadora")
13      case 2:
14              usuario=input("Escoja una opción (Piedra/Papel/Tijera)")
15              if usuario=="Piedra":
16                      print("Gana computadora")
17              if usuario=="Papel":
18                      print("Empate")
19              if usuario=="Tijera":
20                      print("Gana usuario")
21      case 3:
22              usuario=input("Escoja una opción (Piedra/Papel/Tijera)")
23              if usuario=="Piedra":
24                      print("Gana usuario")
25              if usuario=="Papel":
26                      print("Gana computadora")
27              if usuario=="Tijera":
28                      print("Empate")
29      case _:
30              print("No seleccionó ninguna de las opciones")
```

Nota. Los autores, Python

Random es una librería que genera números aleatorios en el caso del ejemplo de la figura 3.24, se utilizó **randint** que genera números aleatorios enteros.

3.10. Listas

Las **listas** son colecciones ordenadas de valores que pueden almacenar diferentes tipos de datos y valores como numéricos, cadenas, booleanos, e incluso otras listas.

En el Capítulo 2 se explicó acerca de las estructuras de almacenamiento como vectores y matrices, en *Python* se los mencionará como listas.

Python ofrece nuevas formas de almacenamiento como son las listas, las mismas que no necesitan predefinirse, ni indicar que sólo van a almacenar un tipo de datos, se utiliza la sintaxis de corchetes [ ], y los valores de la lista se separan por comas.

Figura 3.25.

Generando una lista

ejemplo.py > ...

```
1    # Creación de una lista
2    numeros = [1, 2, 3, 4, 5]
```

Nota. Los autores, Python

Las **listas** son extremadamente versátiles y se utilizan en una variedad de entornos funcionales, como almacenar valores, representar matrices o vectores, manipulación de valores de archivos o bases de datos, entre otras cosas.

Ejemplo del manejo de una cadena de *Python* como **lista**:

Figura 3.26.

Cadenas

vocales.py > ...

```python
cadena=input("ingrese una cadena")
contador=0
for i in cadena:
    if "a"==i or "A"==i:
        contador=contador+1
    if "e"==i or "E"==i:
        contador=contador+1
    if "i"==i or "I"==i:
        contador=contador+1
    if "o"==i or "O"==i:
        contador=contador+1
    if "u"==i or "U"==i:
        contador=contador+1
print(f"El texto tuvo{contador}")
```

Nota. Los autores, Python

En la Figura 3.26, el **for** recorre la cadena tomando de ella carácter por carácter, este es el principio básico del manejo de **listas**, cada carácter corresponde a una posición. El ejemplo al recorrer cada carácter si existe coincidencia procede a contabilizar y así se le indica al usuario cuantas vocales (coincidencias) existió en la cadena.

Figura 3.27.

Manejo de listas

Nota. Los autores, Python

En la lista planteada en la Figura 3.27, se puede observar que contiene como elementos una cadena, un número y un carácter.

Ejemplo de llenado e impresión de una lista.

Figura 3.28.

Manejo de listas con funciones

lista.py > ...

```
1   def llenar(n,lista):
2       for i in range(n):
3           num=int(input(f"ingrese el {i+1} valor"))
4           lista.append(num)
5       return lista
6   def mostrar(lista):
7       print(lista)
8
9   lista=[]
10  n=int(input("cuantos datos desea ingresar "))
11  mostrar(llenar(n,lista))
```

Nota. Los autores, Python

Se inicia con la generación de una **lista** y se solicita al usuario el número de valores que se va a ingresar, se lo almacena en la variable **n**, luego se procede a realizar un llamado a la función **mostrar**, la cual invoca a la función **llenar** que recibe la variable **n** más la **lista** a través del ciclo **for** se procede al llenado de la lista con valores ingresados por el usuario con la propiedad **append**, la función llenar retorna la **lista** a la función **mostrar**, la cual a través de **print** mostrará en pantalla la **lista**.

Ejemplo, solicitar al usuario una cantidad de números y mostrar el número de valor más bajo, remplazar ese lugar con ceros y sacar el promedio con la misma cantidad de números.

Figura 3.29.

Listas dinámicas

leccion2B.py > ...

```
7   num = int(input("Cuantos números desea ingresar"))
8   numeros=[]
9   for i in range (num):
10      numeros.append(int(input(f"Ingrese la {i+1} cantidad")))
11
12  mini=numeros[0]
13  print(mini)
14
15  for i in (numeros):
16      print(i)
17      if i<=mini:
18          mini=i
19  print ("El numero más bajo es:",mini)
20  print(numeros)
21  j=0
22  for i in (numeros):
23      if i==mini:
24          numeros[j]=0
25      j=j+1
26  print(numeros)
27
28  suma = 0
29  for i in numeros:
30    suma=i+suma
31  print ("El promedio es",suma/num)
```

Nota. Los autores, Python

En el ejemplo de la Figura 3.29, se le solicita al usuario la cantidad de números que desea ingresar, luego se genera una **lista** que será recorrida por un **for** y como estructura tendrá el llenado de la **lista**. Se asume que el valor mínimo es el primer valor de la lista y se procede a mostrarla para que el usuario verifique el llenado de la misma, por ejemplo:

Número[1,2,3]

Se procede a recorrer la lista y como estructura interna tiene un **if,** el cual consulta si el número que se asumió como mínimo almacenado en la variable **min** es en realidad el valor menor, caso contrario se actualiza el nuevo valor menor a la variable hasta terminar el recorrido de la **lista.** Luego se procede a recorrer la **lista** para encontrar una igualdad del valor mínimo contenido en la variable **min** comparándolo con cada valor de la **lista**, cuando se encuentre la igualdad en esa posición de la lista se asignará con un valor de cero.

Se procede a recorrer la **lista** para sumar todos los valores y sacar el promedio, el cual se mostrará en pantalla.

Los índices de posición se utilizan para conocer el lugar exacto donde remplazar con ceros en la **lista**. Para agregar elementos a una lista se utiliza la función **append** esta función siempre agregará al final de la lista es decir en la última posición.

Lista que contienen listas

En *Python* las estructuras pueden contener otras estructuras, es por eso que a los siguientes ejemplos se los va a generar con **listas** conteniendo otras listas, a lo que se le conoce en la programación tradicional como matrices.

Figura 3.30.

Listas que contienen listas

matriz.py > ...

```
matriz=[
    ["Maria","José","Roberto"],
    ["PEPE","Carlos","Figueroa"],
    ["Manuel","Rosario","Sofia"],
]

for fila in matriz:
    print(fila)
```

Nota. Los autores, Python

La Figura 3.30, se implementa una **lista** que contiene otras **listas**, específicamente contiene tres listas, cada una de estas listas contiene tres elementos que son cadenas.

El índice de la lista nombrada **matriz** corresponderá a la fila de la **matriz** que se desea generar, mientras que el índice de la lista interna corresponderá a la columna de la **matriz** que se desea generar y es así como usando **listas** dentro de otra lista se puede mostrar una matriz.

Ejemplo de llenado de una lista como matriz:

Figura 3.31.

Generando listas anidadas

matrizll.py > ...

```
1   fila=int(input("Ingrese el numero de filas"))
2   col=int(input("Ingrese el numero de columnas"))
3   matriz=[]
4   for i in range(fila):
5       vector=[]
6       for j in range(col):
7           a=int(input(f"Ingrese el valor en la celda {i+1},{j+1}= "))
8           vector.append(a)
9       matriz.append(vector)
10
11  for fila in matriz:
12      print(fila)
13
```

Nota. Los autores, Python

Para el llenado de una **lista** que contiene otras **listas**, se procede a implementar una **lista** de nombre **matriz** que es la que contendrá a las demás **listas**, cada una de estas será un elemento de la lista general **matriz**, la **lista**(vector) es la que contendrá los elementos que formará parte de cada una de las columnas de la **lista** general **matriz**, es decir que **i** representa fila y **j** representa columnas. Una vez llenado el vector se procede a

ser agregada a la **matriz** como su primer elemento, **i** incrementará con lo que procede a estar en la segunda fila, una vez más se llenará el **vector** que en cuanto este esté completo formará el segundo elemento de la **matriz** y así sucesivamente.

Ejemplo: generar una matriz en la que las dos primeras columnas contendrá valores ingresados por el usuario, y en la tercera columna contendrá la suma de las dos primeras columnas.

Figura 3.32.

Sumando elementos dentro de la lista

identidad.py > ...

```
8   filas=int(input("Ingrese el numero de filas"))
9   matriz=[]
10
11  for i in range(filas):
12      vector=[]
13      for j in range(3):
14          if j==2:
15              vec= vector[0]+vector[1]
16              vector.append(vec)
17          else:
18              vector.append(int(input(f"Ingrese el valor en la posición {i+1}, {j+1}")) )
19      matriz.append(vector)
20
21  print()
22  for fila in matriz:
23      print(fila)
```

Nota. Los autores, Python

En la Figura 3.32, se solicita al usuario que indique cuantas filas desea generar en la **matriz** que se va a trabajar, este valor ingresado por el usuario será colocado en la variable de nombre **filas**, luego se procede a crear una lista con nombre **matriz**.

Para proceder al llenado de la **matriz** que se desea generar, se plantea que el número de columnas que va tener la matriz será de tres, primero con la instrucción de **for** se procederá a controlar el número de filas solicitado por el usuario, entonces la variable **i** representa a las filas, seguido de otra instrucción **for** con la variable **j** que representa a las columnas de la lista de nombre **matriz**.

Se ha procedido a generar una segunda **lista** con el nombre **vector**, pues para generar una lista en forma de una matriz se utilizará la lista de nombre **matriz** que contendrá como elemento otras listas que en el caso del ejercicio tiene de nombre **vector**.

El proceso de llenado es el siguiente:

- Generar la lista **matriz.**
- Comenzar el recorrido controlando la generación de la fila.
- Generar una nueva lista de nombre **vector.**
- Recorrer **j** hasta tres.

vector[ ] vector sin elemento

Recorrido de j	Agregar dato a vector	Suma de valores del vector por posición
0	vector[2,]	
1	vector[3,]	
2	vector[5] ←	vector[0] + vector[1]

vector[2,3,5] vector lleno con tres posiciones

- Una vez llenado el **vector** proceder a pasarlo como elemento de la lista **matriz** con la propiedad **append**.

  vector[2,3,5]
  matriz [[2,3,5],] lista con su primer elemento otra lista

- Consultar al **for** de la variable **i** si se mantiene el recorrido caso contrario se saldrá del ciclo

- Proceder a mostrar la lista **matriz**, la cual será recorrida elemento por elemento (fila por fila).

  Suponiendo que el valor de filas que ingresó el usuario ha sido tres, la lista **matriz** queda con tres elementos de la siguiente forma.

```
matriz[  [2,3,5],
         [,3,5,8],
         [6,10,16]
]
```

Ejemplo de la Figura 3.33, pero con una tercera matriz.

Figura 3.33.

Manejo de listas como matrices

identidad2.py > suma

```python
def suma(matriz,filas):
    matriz3 = []
    for i in range(filas):
        vector3 = []
        for j in range(3):
            if j==2:
                suma = matriz[i][0] + matriz[i][1]
                vector3.append(suma)
            else:
                m=matriz[i][j]
                print(m)
                vector3.append(m)
        matriz3.append(vector3)
    return matriz3

filas=int(input("Ingrese el numero de filas"))
matriz=[]
print("Llenado de la primera matriz")
for i in range(filas):
    vector1=[]
   # matriz=[int(input(f"Ingrese el valor en la celda {i+1},{j+1}= ")) for j in range(col)]
    for j in range(3):
        if j==2:
            vector1.append(0)
        else:
            a=int(input(f"Ingrese el valor en la celda {i+1},{j+1}= "))
            vector1.append(a)
    matriz.append(vector1)

print(matriz)
a1=suma(matriz,filas)

print()
for fila in a1:
   print(fila)
```

Nota. Los autores, Python

El proceso de llenado es el siguiente:

- Generar la lista **matriz.**
- Comenzar el recorrido controlando la generación de la fila.
- Generar una nueva lista de nombre **vector.**
- Recorrer **j** hasta tres, pero dentro del **for** existe una condición cuando la variable **j** tome el valor de dos la posición dos que corresponde al **vector** se le asignará el valor de cero.

  vector[ ] vector sin elemento.

Recorrido de j	Agregar dato a vector
0	vector[2,]
1	vector[3,]
2	vector[0]

vector[2,3,0] vector lleno con tres posiciones.

- Una vez llenado el **vector** proceder a pasarlo como elemento de la lista **matriz** con la propiedad **append**.

  vector[2,3,0]
  matriz [[2,3,0],] lista con su primer elemento otra lista.

- Consultar al **for** de la variable **i** si se mantiene el recorrido caso contrario se saldrá del ciclo.
- Proceder a mostrar la lista **matriz**, la cual será recorrida elemento por elemento (fila por fila).
  Suponiendo que el valor de filas que ingresó el usuario ha sido tres, la lista **matriz** queda con tres elementos de la siguiente forma.

```
matriz[  [2,3,0],
         [,3,5,0],
         [6,10,0]
]
```

- Se genera la variable **a1** que se le asignará lo que devuelve la función **suma**, la cual recibe dos argumentos, la **matriz** y la variable **filas**.
- En la función se procede con el llenado de dos nuevas listas **matriz3** y **vector3**, utilizando la misma condición de **j** cuando sea igual a dos.
- La función devuelve la nueva lista llenada **matriz3**, la cual es asignada en **a1**.
- Se procede a recorrer la **matriz** y se muestra por pantalla.

```
matriz[  [2,3,5],
         [,3,5,8],
         [6,10,16]
        ]
```

Ejercicios Propuestos

1. Realizar un programa que simule el diagnóstico de un doctor con tres opciones de diagnóstico, se analizará tres posibles síntomas y se le dará un pequeño diagnóstico.

2. Realizar un programa que solicite al usuario un número y muestre si este es divisible para tres.

3. Realizar un programa que genere una tabla de multiplicar con funciones hasta el número 12, el número será ingresado por el usuario.

4. Realizar la serie de Fibonacci con funciones.

5. Realizar un programa que almacene una cantidad de notas, generar tres funciones, la primera función para el ingreso de las notas, la segunda función para calcular el promedio y la tercera función para mostrar si el estudiante aprobó, la calificación mínima para aprobar será mayor o igual de 7.

6. Generar una lista con n notas y otra con los respectivos nombres de los estudiantes, mostrar cada estudiante con su nota.

7. Generar una matriz en Python de un mes del calendario generado dinámicamente.

8. Realizar la suma de matrices, con funciones.

9. Generar una matriz y mostrar su transpuesta.

CAPÍTULO 4

UTILIZANDO ARCHIVOS EN PYTHON

Capítulo 4

Utilizando archivos en Python

4. Manejo de archivos con Python

El objetivo del Capítulo 4 es aprender a escribir, leer, modificar datos de manera persistente, de esta forma se podrá comprender mejor cómo recuperar y almacenar información por un largo tiempo, a través de ejemplos el manejo de archivos en la vida real.

Hasta el momento se ha venido trabajando con representaciones de almacenamiento de valores en memoria con las variables; esto si bien es cierto, ayuda a manipular la información de manera eficiente, solo estará disponible mientras el programa o aplicativo esté en ejecución, sabiendo que si el programa deja de ejecutarse, esta información simplemente se perderá o no estará disponible para una próxima ejecución.

Para mantener almacenada la información luego que el programa o aplicación se reinicie o se ejecute, existen varios contenedores de información que permiten que la información

perdure a través del tiempo o circunstancias inusuales del sistema.

Los archivos son una de las estructuras de almacenamiento permanente más utilizados en las diferentes herramientas o lenguajes de desarrollo e implementación de programas o aplicativos.

Python permite crear, modificar, borrar y leer un archivo; para el manejo de archivos se debe considerar la ruta. En la Tabla 4.1, se muestran los diferentes comandos para trabajar con archivos.

La ruta de un archivo es un parámetro importante al momento de manipular un archivo, sin este parámetro simplemente no se podría trabajar con ellos; hay varias formas de establecer una ruta, pero las más conocidas son las rutas relativas y absolutas.

Cuando un archivo ha sido creado en la misma carpeta donde se encuentra el proyecto bastará con nombrarlo desde el código para poder manipularlo, este es un ejemplo de una ruta relativa. A continuación, se exponen varios ejemplos.

Según Trespaderne y Mazaeda (s.f.) "Un fichero es un conjunto de datos relacionados entre sí, que son almacenados

de forma permanente en dispositivos tales como discos duros, memorias flash, etc. y que pueden ser tratados, hasta cierto punto, como una unidad*”* (párr. 3).

4.1. Rutas de archivos

Los archivos son estructuras de almacenamiento permanente que tienen que estar alojados en un lugar específico dentro del registro del sistema operativo, esta ubicación es el camino que tiene que recorrer el usuario para llegar al mismo, por lo general es conocida como “ruta” o “path”.

Los archivos son abstracciones del sistema operativo que ocupan un lugar en el *hardware* de la computadora, ya que se almacenan directamente en el disco duro, los archivos estarán permanentemente guardados en esa ruta hasta que el usuario los elimine.

Tabla 4.1.

Manejo de rutas

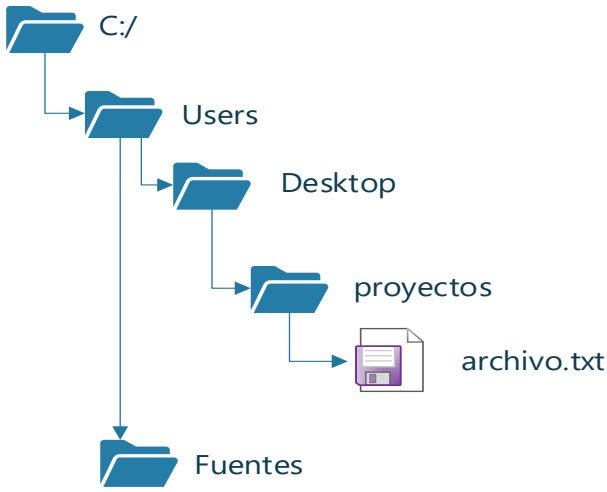
Tipos	Rutas
Ruta relativa	carpeta\archivo.txt
Ruta absoluta	C:\Users\Desktop\proyectos\archivo.txt

Nota. Los autores

El árbol de directorio de la ruta absoluta del ejemplo anterior quedaría de la siguiente manera:

Figura 4.1.

Árbol de directorio de la ruta absoluta



Nota. Los autores, Microsoft Visio

4.2. Tipo de archivos

El tipo de archivo en un lenguaje de programación como *Python*, radica en la forma en que se vaya a almacenar los datos en dicho archivo, desde esa perspectiva cada programador decide el tipo de datos que va a utilizar en su programa o *software* que esta implementado.

Dependiendo esa decisión los archivos pueden ser binarios o de texto.

a) **Archivos binarios**: son archivos que contienen daos o información en formato de *byte*, número, cadena de *byte*; es decir, datos en su representación original no legibles para las personas.

   Este tipo de archivo son utilizados por los programadores cuando estos requieren que sus programas preserven la exactitud o el formato original de los datos. Por lo general eso sucede con archivos de audio o imágenes.

b) **Archivos de texto**: al contrario de los archivos binarios los archivos de textos se almacenan en un formato legible para las personas, son estructuras de almacenamiento permanente que se utilizan para registrar información del usuario permanentemente.

Cuando hablamos de archivos es necesario tener en cuenta que siempre se va requerir realizar dos funciones básicas, la apertura y cierre de archivos.

1. **Apertura de archivos**: es la función que permite asociar el archivo con una variable que provea la fuente de datos que va a ser leída o escrita, es decir, el modo de acceso. Existe una función predeterminada que *Python* utiliza para

la apertura de archivos y se conoce como **open**. Esta función tiene la siguiente sintaxis.

Figura 4.2.

Función open

```
open("ruta/nombre del archivo","modo",buff)
```

Fuente: Los autores, Python

- **ruta/nombre del archivo**: es la ubicación del archivo en el disco duro (esto va a depender si es una ruta relativa o ruta absoluta) es una cadena de caracteres que contiene el nombre del archivo al que queremos acceder.
- **modo**: el modo de acceso al archivo, esto sirve para determinar si el archivo ha sido abierto para leer, escribir, agregar información del usuario.
- **buff**: es el valor de almacenamiento de **buffer**, si el valor es 0 no se realiza ningún almacenamiento de **buffer** mientras que si el valor es 1 la instrucción ejecuta un **buffer** de línea. Po lo general no es necesario colocarlo en la instrucción ya que por defecto tomará el valor de 1.

2. **Cierre de archivo**: Una vez que el programa haya terminado de ejecutar la variable que representa el

archivo, se debe poner en conocimiento al sistema operativo para que este proceda a resguardarlo y cerrar, esta acción impedirá el desbordamiento de memoria por ejecución infinita. Además, el sistema operativo está diseñado para soportar una cierta cantidad de archivos abiertos simultáneamente en memoria, si no se cierra los archivos que no se están utilizando este no tendrá más espacio para abrir nuevos archivos o aquellos que se requiere abrir en un momento dado.

Para cerrar un archivo, basta con invocar la función **close**, su sintaxis es la siguiente.

Figura 4.3.

Función close

```
archivo.close()
```

Nota. Los autores, Python

4.3. Modos de acceso a los archivos

Modos de acceso o de apertura son operaciones de archivos importante a la hora de la manipulación y control de la información que se almacenará en los archivos, con los modos de acceso el programa podrá actuar según corresponda ya sea para leer, escribir, añadir, etc.

Tabla 4.2.

Modos de apertura

Modo	Acceso
r	Accede al archivo solo para leerlo y se ubica en la posición cero. Si no existe mensaje de error.
r+	Actualiza el archivo. Si no existe error.
w	Permite crear un archivo desde cero y si existe lo sobre escribe.
w+	Actualiza el archivo.
x	Crea un archivo, pero si existe no realiza nada.
a	Si el archivo no existe lo crea para escritura. Si existe se posesiona en la última línea.
a+	Si existe el archivo lo coloca al final.
r+	Accederá a un archive para escribir y leer.
close	Para cerrar un fichero.
x	Para crear ficheros.

Nota. Los autores

En *Python* para abrir un archivo se utiliza la propiedad ***open*.** Al trabajar con archivos, siempre es una buena práctica cerrar el archivo después de su uso. Además, se debe tener cuidado de no sobrescribir o eliminar datos importantes al escribir en un archivo. Usar los modos de apertura adecuadamente según sus necesidades para evitar accidentes no deseados.

El siguiente código representa la implementación de la apertura y lectura de un archivo.

Figura 4.4.

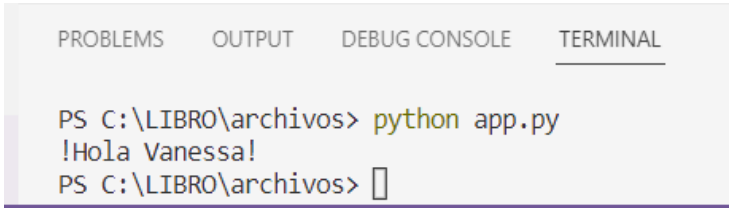
Abriendo y leyendo un archivo con rutas relativas

Nota. Los autores, Python

En la Figura 4.4, se implementa una variable de nombre **ruta_relativa** que contiene toda la dirección o ubicación donde está el archivo, luego mediante la variable **archivo** invocamos a la función **open** para proceder a abrir el **archivo.txt**, el archivo abierto está contenido en la variable **archivo** ya que al aplicar el modo de acceso **r**, permite leer el contenido del archivo desde la posición cero, es decir, desde el inicio del archivo; luego utilizando la función **print** y **read** el código procede a mostrar el contenido del archivo en pantalla.

Figura 4.5.

Resultado de la lectura del archivo



Nota. Los autores, Python

El manejo de rutas absolutas es parecido, la diferencia radica en que hay que colocar la ubicación completa del archivo, así como lo muestra el siguiente código.

Figura 4.6.

Abriendo y leyendo un archivo con rutas absolutas

app.py

```
# Accediendo a un archivo con ruta absolutas
ruta_absoluta = "C:\\LIBRO\\archivo.txt"
archivo = open(ruta_absoluta,"r")
print(archivo.read())
archivo.close()
```

Nota. Los autores, Python

En *Python*, hay varios métodos y funciones que se utilizan para el manejo de archivos. Estos métodos permiten leer, escribir y manipular archivos de texto y binarios.

4.4. Métodos para el manejo de archivos

Para el manejo de archivos Python proporciona un conjunto de métodos que ayudan a simplificar el trabajo con archivos. A continuación, se presenta un conjunto de métodos para la optimización del manejo de archivos.

Tabla 4.3.

Métodos del manejo de archivos

Propiedades	Usos
readline()	Lee sólo una línea del archivo, cada vez que se ejecute se guardará la última posición.
readlines()	Lee múltiples líneas.
write	Permite escribir el texto al archivo.
seek(0)	Coloca el puntero del archivo al principio del archivo de texto.
Close()	Cierra el archivo.
read(size)	Lee y devuelve una cantidad size de caracteres o bytes del archivo.
writelines(lines)	Escribe una lista de cadenas de texto lines en el archivo.
tell()	Devuelve la posición actual del cursor en el archivo.
flush()	Escribe todos los datos almacenados en el buffer al archivo.

Nota. Los autores, Python

Los métodos para manejo de archivos, son funciones pre establecidas en *Python* que ayudan a gestionar de mejor forma el contenido de los archivos, a continuación, se presentará algunos ejemplos que permitirán aplicar de manera práctica las propiedades antes mencionadas.

Propiedad *readline():* como ya se mencionó en la Tabla 4.3, esta función permite leer una línea completa de un archivo.

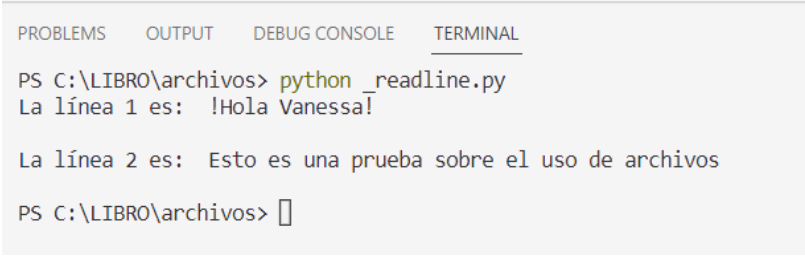
Figura 4.7.

Método readline

Nota. Los autores, Python

Figura 4.8.

Salida del método readline



Nota. Los autores, Python

El código ejemplo, esta implementado para entender cómo funciona el método **readline**, para ello se requiere abrir el archivo en modo lectura **r**, antes de ello se ha creado un archivo con información. Luego se invoca el método **readline**, asigna la línea

extraídas en la variable **línea**_1 y con el mismo procedimiento a la variable **linea_2** para luego ser presentadas en pantalla.

Propiedad *readlines()*: esta propiedad permite leer todas las líneas del archivo y los devuelve como una lista de cadenas, cada cadena en la lista representa una línea del archivo, incluido el carácter de nueva línea **\n** al final de cada línea.

Esta propiedad es útil cuando se requiere trabajar con todas las líneas del archivo a la vez o cuando es necesario realizar operaciones específicas en cada línea, ya que proporciona una lista que contiene todas las líneas del archivo de tal manera que pueda manipular fácilmente.

Figura 4.9.

Método readlines

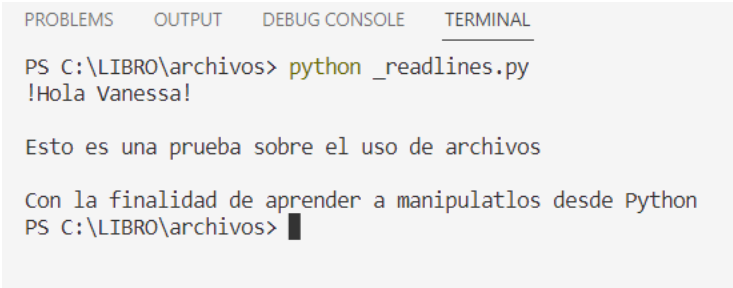
_readlines.py > ...

```
1   # manejo de la propiedad readlines()
2   ruta = "archivo.txt"
3   with open(ruta,"r") as archivo:
4       lineas = archivo.readlines()
5
6   for linea in lineas:
7       print(linea)
8   archivo.close()
```

Nota. Los autores, Python

Figura 4.10.

Salida del método readline()



Nota. Los autores, Python

El código ejemplo, esta implementado para entender cómo funciona el método **readlines***,* para ello se requiere abrir el archivo en modo lectura **r**, antes de ello se ha creado un archivo con información. Luego se invoca el método **readlines** y asigna las líneas extraídas en la variable **lineas** para luego ser presentadas en pantalla.

Propiedad *write*: esta propiedad es utilizada para escribir datos o texto en un archivo. Cada llamada a **write** escribe los datos proporcionados en el archivo. Nota que el carácter de nueva línea **\n** se utiliza para separar líneas.

Es importante indicar que, si el archivo contiene datos, este sobrescrita esos datos mientras utilice el método **w** (**write**), en caso que se requiera añadir datos al archivo existente, se deberá utilizar el método de acceso **a** (**append**).

Figura 4.11.

Método write

```python
# manejo de la propiedad write()
ruta = "archivo.txt"
with open(ruta,"a") as archivo:
    archivo.write("\nAdemás de realizar algunas modificaciones")
    archivo.write("\nEl archivo seguira creciendo poco a poco")

with open(ruta,"r") as archivo:
    lineas = archivo.readlines()

for linea in lineas:
    print(linea)
archivo.close()
```

Nota. Los autores, Python

Figura 4.12.

Salida del método write

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\LIBRO\archivos> python _write.py
!Hola Vanessa!

Esto es una prueba sobre el uso de archivos

Con la finalidad de aprender a manipulatlos desde Python

Además de realizar algunas modificaciones

El archivo seguira creciendo poco a poco
PS C:\LIBRO\archivos>
```

Nota. Los autores, Python

El código ejemplo, esta implementado para entender cómo funciona el método **write***,* para ello se requiere abrir el archivo en modo agregar **a**, luego invocamos el método y procederemos a agregar dos líneas más al archivo, para después abrir nuevamente el archivo pero ahora con modo lectura **r**para extraer las líneas del archivo y presentarlo en pantalla.

Propiedad *seek*: esta propiedad es utilizada para cambiar la posición del puntero de lectura/escritura en un archivo, el puntero es un indicador que dentro del archivo indica la ubicación en el archivo para realizar alguna operación o cálculo sobre los datos que están almacenados en el archivo.

Esta propiedad cuanta con dos parámetros necesarios para su ejecución: **offset** y **whence**.

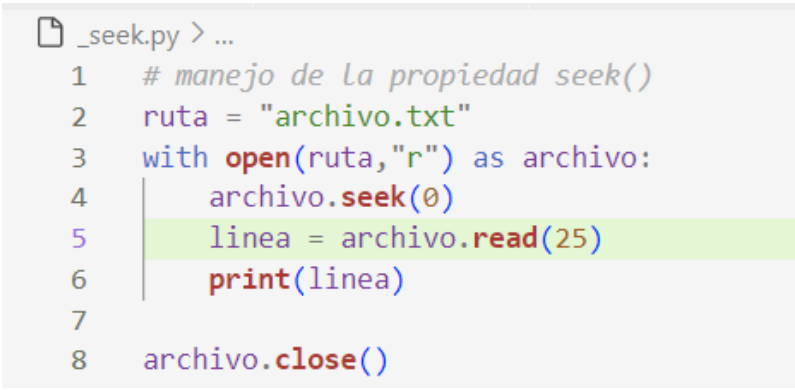
offset, es el valor entero que representa el desplazamiento en bytes desde la referencia indicada.

whence, es un valor entero que indica la referencia desde donde se aplicará el desplazamiento. Puede tomar uno de los siguientes valores:

0: (por defecto): desplazamiento desde el inicio del archivo.
1: desplazamiento desde la posición actual del puntero.
2: desplazamiento desde el final del archivo.

Figura 4.13.

Método seek



Fuente: Los autores, Python

Figura 4.14.

Salida del método seek

Nota. Los autores, Python

El código ejemplo, esta implementado para entender cómo funciona el método **seek**, para ello se requiere abrir el archivo en modo lectura **r**, luego invocando el método se puede cambiar la posición del puntero del archivo, en este caso se iniciará desde

el inicio, por último, se leerá las primeras 25 palabras y las presentará en pantalla.

Propiedad *read(size):* es utilizado para leer una cantidad específica de caracteres dentro de un archivo, el argumento **size** indica la cantidad máxima de caracteres o bytes que se leerán del archivo. Es importante indicar que, si este parámetro es mayor al tamaño del archivo, entonces el método leerá el contenido completo del archivo.

Figura 4.15.

Método read

Nota. Los autores, Python

Figura 4.16.

Salida del método seek

Nota. Los autores, Python

El código ejemplo, está implementado para entender cómo funciona el método **read**, para ello se requiere abrir el archivo en modo lectura **r** y luego en una variable colocar el resultado de extraer las 50 primeras letras contadas desde el índice igual a 0 o desde el principio del archivo.

Propiedad *writelines(lines):* este método es utilizado para escribir varias líneas dentro de un archivo, recibe como argumento una lista de caracteres y almacena cada elemento de la lista como una línea en el archivo.

Figura 4.17.

Método writelines

_writelines.py > ...

```
1   lista = ['Nombre: Vanessa\n','Apellido: Valverde\n','Cédula: 12013553432\n']
2   with open("archivo_1.txt","w") as archivo:
3       archivo.writelines(lista)
4
5   with open("archivo_1.txt","r") as archivo:
6       lineas = archivo.readlines()
7
8   for linea in lineas:
9       print(linea)
```

Nota. Los autores, Python

Figura 4.18.

Salida del método writelines

Nota. Los autores, Python

El código anterior es un ejemplo de como se implementa el método **writelines** en un programa, para poder utilizar este método se requiere tener una estructura tipo **lista**, ya que el contenido de cada posición va a corresponder a una línea en el archivo. El método **writelines**, recibe como argumento esta **lista** y la gestiona enviándola al archivo posición por posición y almacenándola de manera secuencial línea por línea.

Propiedad *tell():* este método está diseñado para obtener la posición actual del cursor del archivo, esta posición representa el byte o carácter donde se encuentra el próximo dato que se leerá con el cursor.

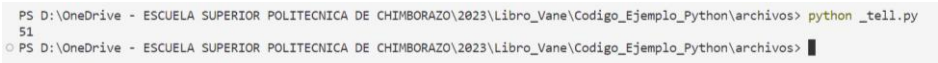
Figura 4.19.

Método tell

Nota. Los autores, Python

Figura 4.20.

Método tell



Nota. Los autores, Python

Como lo indica la ejecución del código la siguiente posición que el cursor va a leer es la 51, el método **tell** proporciona la siguiente posición que deberá tomar el cursor y lo almacena en la variable **posicion**, para luego imprimir por pantalla.

Propiedad *flush():* Se utiliza para escribir todos los datos almacenados en el **búfer** de un archivo en el disco sin cerrar el archivo. Cuando se escribe datos en un archivo, los datos no se escriben inmediatamente en el disco, sino que se almacenan en un **búfer** temporal en memoria.

Figura 4.21.

Método flush

_flush.py > ...

```
1  ruta = "archivo.txt"
2  with open(ruta,"w") as archvo:
3      archvo.write('Esta línea se incrementa al final del archivo')
4      archvo.flush()
5
```

Nota. Los autores, Python

En este punto los datos escritos en el archivo ya están almacenados en el disco duro del computador.

Se requiere que en un archivo binario se escriba la leyenda “HOLA VANE” en código binario y lego se presente en pantalla en su estado natural.

Figura 4.22.

Escribiendo leyenda en archivo binario

```
_binario.py > ...
1  archivo_binario = b'01001000 01001111 01001100 01000001 00100000 01010110 01000001 01001110 01000101'
2
3  with open("archivo_binario.bin","wb") as archivo:
4      archivo.write(archivo_binario)
5
6  with open("archivo_binario.bin","rb") as archivo:
7      contenido = archivo.read()
8
9  print(contenido)
```

Nota. Los autores, Python

En una variable **archivo_binario** se asigna la cadena binaria que corresponde a “HOLA VANE”, con la función **open**.

Ejercicios Propuestos

1. Abrir un archivo de texto.

2. Mostrar n líneas del archivo de texto.

3. Sobre escribir un archivo de texto existente.

4. Genere un archivo y mueva el puntero a la posición 1.

5. Genere un archivo y muestre todas las líneas del archivo.

6. Escriba una cadena dentro del archivo.

7. Genere un archivo con varias líneas y lea la primera línea.

8. Genere un archivo y consulte si el archivo está cerrado.

9. Genere un archivo y retorne la posición actual del puntero.

10. Lea un archivo y ciérrelo.

Problemas Resueltos

1. **Realizar un programa en Python que solicite al usuario un número de n cifras y que sume los números pares y los números impares.**

ncifras.py > ...

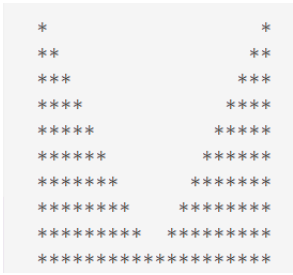
```
numero=input("Ingrese un número")
lista=list(numero)
listan=list(map(int, lista))
suma=0
sumai=0
for i in listan:
    if i%2==0:
        suma=i+suma
    else:
        sumai=i+sumai

print(f"La suma de los valores pares es:{suma} ")
print(f"La suma de los valores impares es:{sumai} ")
```

2. **Realizar un programa en Python que genere una matriz de n números, imprima la matriz a través de la función mostrar y sume los números pares e impares a través de una función buscar.**

matrizparimpar.py > ...

```python
def mostrar(matriz):
    for fila in matriz:
        print(fila)

def buscar(filas,columnas, matriz):
    sumapar=0
    sumaimpar=0
    for i in range(filas):
        for j in range(columnas):
            if matriz[i][j]%2==0:
                sumapar=sumapar+matriz[i][j]
            else:
                sumaimpar=sumaimpar+matriz[i][j]
    return(sumapar,sumaimpar)

filas=int(input("Ingrese el número de filas") )
columnas=int(input("Ingrese el número de columnas"))
matriz=[]
for i in range(filas):
    lista=[]
    for j in range(columnas):
        lista.append(int(input(f"Ingrese el valor en la posición {i}, {j}")))
    matriz.append(lista)

mostrar(matriz)
print(f"La suma de números pares e impares es: {buscar(filas,columnas,matriz)} , respectivamente")
```

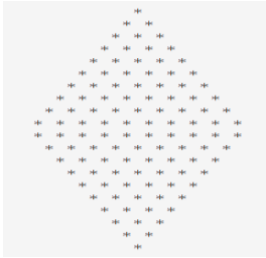
3. **Realizar un programa en Python que genere dos triángulos con una altura h y una base b cuya salida sea.**



triangulo.py > ...

```python
base=int(input("Ingrese la base"))
altura=base

i=0
k=base
while i<=base:
        for _ in range(i):
            print("*",end="")

        if i!=base:
            for _ in range(k):
                print(end="  ")

        for _ in range(i):
            print( "*",end="")
        print()
        i=i+1
        k=k-1
```

4. **Realizar un programa en Python con dos procedimientos que genere la siguiente figura.**



círculo.py > ...

```
1   def parte1(k,i):
2       for _ in range(k+1):
3           for _ in range(k):
4               print(end=" ")
5           for _ in range(i):
6               print("*",end=" ")
7
8           print()
9           i=i+1
10          k=k-1
11
12
13  def parte2(k,i):
14      for _ in range(k+1):
15          for _ in range(i):
16              print(end=" ")
17          for _ in range(k):
18              print("*",end=" ")
19
20          print()
21          i=i+1
22          k=k-1
23
24  diametro=int(input("Ingrese el diámetro"))
25  radio=diametro/2
26  k=int(radio)
27  i=0
28  parte1(k,i)
29  parte2(k,i)
```

5. **Realizar un programa en Python que genere el triángulo de pascal.**

```
[1]
[1, 1]
[1, 2, 1]
[1, 3, 3, 1]
[1, 4, 6, 4, 1]
```

teiangulo_pascal.py > ...

```python
filas=int(input("Ingrese un número impar"))

lista=[]
lista1=[]
lista2=[]

for i in range(3):
    if i==0:
        lista1.append(1)
        lista.append(lista1)
    if i==1:
        lista2.append(1)
        lista2.append(1)
        lista.append(lista2)
    if i>1:
        for i in range(1,filas):
            elemento = [1]
            cant=len(lista[i])-1
            for j in range(cant):
                suma=lista[i][j] + lista[i][j+1]
                elemento.append(suma)
            elemento.append(1)
            lista.append(elemento)

for fila in lista:
    print(fila)
```

Aplicando la potencia:

teiangulo_pascal.py > ...

```python
filas=int(input("Ingrese un número impar"))
lista=[]

for i in range(filas):
    lista.append(11**i)

for fila in lista:
    print(fila)
```

6. **Realizar un programa en Python que genere la matriz diagonal de orden n.**

```
[0, 2, 0, 0, 0]
[0, 0, 8, 0, 0]
[0, 0, 0, 7, 0]
[0, 0, 0, 0, 0]
```

matrizidiagonal.py > ...

```python
import random
orden=int(input("Ingrese el orden de la matriz"))
matriz=[]
k=0

for i in range(orden):
    fila=[]
    for j in range(orden):
        if j==k:
            fila.append(random.randint(0,9))
        else:
            fila.append(0)

    matriz.append(fila)
    k=k+1

for fila in matriz:
    print(fila)
```

7. **Realizar un programa en Python que identifique si la matriz ingresada por el usuario es simétrica.**

simétrica.py > ...

```python
def matrizoriginal(orden):
    matriz=[]

    for i in range(orden):
        fila=[]
        for j in range(orden):
            fila.append(int(input(f"Ingrese el valor en la posición {i},{j}: ")))
        matriz.append(fila)
    return(matriz)

def matriztrans(matriz,orden):
    matriz1=[]
    for i in range(orden):
        fila1=[]
        for j in range(orden):
            fila1.append(0)
        matriz1.append(fila1)
    for i in range(orden):
        for j in range(orden):
            matriz1[j][i]=matriz[i][j]
    return(matriz1)

def mostrar(matriz,matriz1):

    print("MATRIZ ORIGINAL")
    for fila in matriz:
        print(fila)
    print("MATRIZ TRANSPUESTA")
    for fila in matriz1:
        print(fila)

def consulta(matriz,matriz1):
    if matriz==matriz1:
        print("La matriz es simétrica")
    else:
        print("La matriz no es simétrica")

orden=int(input("Ingrese el orden de la matriz"))
matriz=matrizoriginal(orden)
matriz1=matriztrans(matriz,orden)
mostrar(matriz,matriz1)
consulta(matriz,matriz1)
```

8. Realizar un programa que coloque el autor de la frase en cada línea del archivo generado.

archivo.py > ...

```python
archivomio = open ('mi_primer_archivo.txt','w')
archivomio.write(f"Los ordenadores son inutiles. Solo pueden darte respuestas \n")
archivomio.write("Controlar la complejidad es la esencia de la programacion \n")

archivomio = ("mi_primer_archivo.txt")

with open(archivomio,"r") as archivo:
    filas= archivo.readlines()

archivomio =('mi_primer_archivo.txt')

with open(archivomio,"w") as archivo:
    for fila in filas:
        cita=input("Ingrese la cita")
        adicional= f"{fila.strip() } + {cita} \n"
        archivo.write(adicional)

archivomio = open("mi_primer_archivo.txt")
print(archivomio.readlines())
```

En el archivo:

mi_primer_archivo.txt

```
Los ordenadores son inutiles. Solo pueden darte respuestas +  Pablo Picaso
Controlar la complejidad es la esencia de la programacion +  Brian Kernigan
```

9. **Realizar un programa en Python que genere una tabla en un archivo de texto. La tabla contendrá dos columnas una con las carreras ofertadas de una institución y la otra columna tendrá la cantidad de estudiantes de cada carrera, la información será ingresada por el usuario.**

tablaarchivo.py > ...

```
num=int(input("Cuantos carreras desea registrar"))
carrera=[]
cantidad=[]
for i in range(num):
    carrera.append(input(f"Ingrese la carrera {i+1}: "))
    cantidad.append(input(f"Ingrese la cantidad de estudiantes de esa carrera {i+1}: "))

archivo = open ("tabla.txt",'w')
archivo = "tabla.txt"
with open(archivo, 'w') as archivo:
    archivo.write("CARRERA\t          CANTIDAD\n")
    for carrera,cantidad in zip(carrera,cantidad):
        archivo.write(f"{carrera}          {cantidad}\n")

archivo = open("tabla.txt")
print(archivo.readlines())
```

Salida:

```
Cuantos carreras desea registrar2
Ingrese la carrera 1: INDUSTRIAL
Ingrese la cantidad de estudiantes de esa carrera 1: 120
Ingrese la carrera 2: AUTOMOTRIZ
Ingrese la cantidad de estudiantes de esa carrera 2: 180
['CARRERA\t          CANTIDAD\n', 'INDUSTRIAL          120\n', 'AUTOMOTRIZ          180\n']
```

En el archivo:

tabla.txt

1	CARRERA	CANTIDAD
2	INDUSTRIAL	120
3	AUTOMOTRIZ	180

10. Realizar un programa en Python que introduzca texto generado por el usuario a un archivo .txt, y que el usuario pueda agregar una nueva línea a dicho texto.

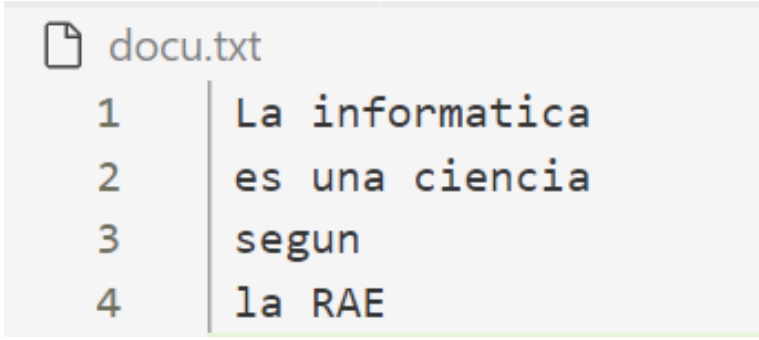
agregartexto.py > ...

```
1   documento = open ('docu.txt','w')
2   l=int(input("Cantidad de líneas a agregar"))
3   for i in range(l):
4       frase=input("Ingrese el texto en la línea {i+1}")
5       documento.write(f"{frase} \n")
6
7   documento = 'docu.txt'
8   texto= input("Ingrese el nuevo texto")
9   linea=int(input("Despues de que línea desea agregar"))
10
11  with open(documento, 'r') as archivo:
12          lineas = archivo.readlines()
13
14          if len(lineas) >= linea:
15              lineas.insert(linea, texto + '\n')
16
17          with open(documento, 'w') as archivo:
18              archivo.writelines(lineas)
19
20  documento = open("docu.txt")
21  print(documento.readlines())
```

Salida:

```
Cantidad de líneas a agregar3
Ingrese el texto en la línea {i+1} La informatica
Ingrese el texto en la línea {i+1} es una ciencia
Ingrese el texto en la línea {i+1} la RAE
Ingrese el nuevo texto segun
Despues de que línea desea agregar2
[' La informatica \n', ' es una ciencia \n', ' segun\n', ' la RAE \n']
```

En el archivo:



Conclusiones

- Los conceptos básicos de informática en el Capítulo 1, son necesarios para que el lector se encuadre en las diferentes terminologías necesarias para futuros usos en los diferentes lenguajes de programación.

- La lógica de programación implementada en el Capítulo 2, motiva a que el lector comience con un proceso de razonamiento para dar solución a diferentes problemas, que fueron representados a través de algoritmos en sus diferentes representaciones.

- El desarrollo de código en el lenguaje de programación en el Capítulo 3, permite al lector ampliar, a través de los ejemplos planteados en el libro, destrezas de generación de código.

- El manejo de archivos en el lenguaje de programación del Capítulo 4, permite al usuario almacenar de manera permanente la información y fortalecer destrezas para el desarrollo de programas.

- DFD es un programa dirigido a principiantes, fácil de usar y muy intuitivo, aunque sus gráficas difieren de los diagramas de flujo tradicionales no se constituye en un problema, al ser un programa sencillo no se puede implementar algoritmos complejos.

- PSeInt es un programa para personas ya con un poco de conocimiento en conceptos básicos de algoritmos, sencillo de utilizar para la implementación de seudocódigo, con n componente para el desarrollo de diagramas de flujo, prepara al estudiante para el desarrollo de código en un leguaje de programación.

- Python es un lenguaje de programación que permite la implementación de código para el desarrollo de aplicaciones como páginas web, análisis de datos, inteligencia artificial, entre otros, es un software que se recomienda para principiantes por ser uy amigable, es por ese motivo la secuencia del presente libro.

- DFD en una herramienta para iniciar aprendiendo la secuencia de algoritmos, mientras que PSeInt es una herramienta de práctica para enseñar a implementar seudocódigo y así continuaron la programación ambas herramientas ayudan a desarrollar la lógica de programación, si ya se desea asar a un lenguaje de programación lo recomendable es Python por tener na sintaxis limpia y fácil de leer, además existe mucha documentación y una gran comunidad.

Glosario

B

Bits: unidadmínima de información.

Bluetooth: redes inalámbricas personales.

C

Caracteres especiales: símbolos que representan acciones, puntuaciones, preguntas, ente otros.

Char: representa al tipo de dato carácter.

Ciclo: es un segmento de código que se repite un cierto número de veces.

Compilador: intérprete de un lenguaje de programación a un lenguaje que entiende el procesador.

Constante: no cambia su valor.

CPU: Unidad Central de Proceso.

D

Depuradores: aplicativos que se utilizan para detectar errores de sintaxis y ejecutar el código.

E

Emogi: imagen o pictograma utilizado para representar una idea o emoción.

H

Hardware: parte física del computador.

M

Mouse: dispositivo de entrada.

R

RAM: Memoria de Acceso Rápido.

ROM: Memoria de Sólo Lectura.

Router: conecta redes.

S

Sistema: entorno donde se ejecuta un proceso determinado.

Sistemas Operativos: grupo de programas que permite manejar al computador.

Servidores: máquina que intercambia información en una red.

Seudocódigos: descripción de un algoritmo.

Software: parte no física del computador.

String: representa al tipo de dato cadena.

Switch: conecta módulos.

P

Palabra reservada: texto que utiliza el software para un comando específico.

Python: lenguaje de programación.

Variable: en cualquier momento puede cambiar de valor.

Web: World Wide Web

Referencias

Algar Díaz, M. J. y Fernández de Sevilla Vellón, M. (2019). *Introducción práctica a la programación con Python*. Servicio de Publicaciones. Universidad de Alcalá. https://elibro.net/es/ereader/espoch/124259? page=209.

Ayala San Martín, G. (2020). *Algoritmos y programación: mejores prácticas.* San Andrés Cholula, Puebla, Fundación Universidad de las Américas Puebla (UDLAP). https://elibro.net/es/ereader/espoch/180290?page=32.

Beúnes, J. y Vargas A. (2019). *La introducción de la herramienta didáctica PSeInt en el proceso de enseñanza aprendizaje: una propuesta para Álgebra Lineal.* http://scielo.sld.cu/pdf/trf/v15n1/2077-2955-trf-15-01-147.pdf

Delgado H.(2022). *Arrays, arreglos, cadenas o vectores en C - Ejemplos y uso.* https://disenowebakus.net/arrays.php

de Miguel, R. (2023). *PSeint: descubrimos los secretos de esta herramienta para iniciarse en la programación por código.* https://www.educaciontrespuntocero.com/tecnologia/pseint-programacion/

European Knowledge Center for Information Technology. (2021). *Diagrama de flujo de datos (DFD).* Consultado el 24 de julio de 2023, TIC Portal. https://www.ticportal.es/glosario-tic/diagrama-flujo-datos-dfd

Martín Villalba, C. Urquía Moraleda, A. y Rubio González, M. Á. (2021). *Lenguajes de programación.* UNED - Universidad Nacional de Educación a Distancia. https://elibro.net/es/ereader/espoch/184827? page=94.

Python Software Foundation. (2023). *Tutorial.* https://docs.python.org/3/tutorial/index.html

Real Academia Española (2022). *Diccionario de la lengua española* https://dle.rae.es/algoritmo

Robledano A. (2019). *Qué es pseudocódigo.* *https://openwebinar s.net/blog/que-es-pseudocodigo/*

Toro, J.(2020). *Arreglos unidimensionales*. https://joseltoro .blogspot.com/2020/06/arreglos-unidimensionales.html

Trejos Buriticá, O. y Muñoz Guerrero, L. (Il.) (2021). *Introducción a la Programación con Python. 1.* RA-MA Editorial. https://elibro.net/es/ereader /espoch/230298?page=38.

Trespaderne F. y Mazaeda R.(sf). *Introduccióna Python*. https://www2.eii.uva.es/fund_inf/python/notebooks/10_Ficheros/Ficheros.html.

Vegas Gertrudix, J. M. (2021). *Java 17: Fundamentos prácticos. de programación. 1.* RA-MA Editorial. https://elibro.net/es/ereader /espoch/222664?page=34

Fundamentos de Programación con DFD-PSeInt-Python. Es un libro diseñado para personas que no conocen de programación, se inicia desde conceptos básicos de informática, para continuar con algoritmos hasta generar pequeños códigos a través de un lenguaje de programación. Como práctica en el Capítulo 2, se utilizó el programa DFD y PSeInt para la elaboración de diagramas de flujo y seudocódigo; en los Capítulos 3 y 4, el lenguaje de programación Python.

Autores

Vanessa Lorena Valverde González. Magister en Informática Educativa, Ingeniera en Sistemas Informáticos, Analista en Sistemas Informáticos por la Escuela Superior Politécnica de Chimborazo; Master Universitario en Ingeniería de Software y Sistemas Informáticos por la Universidad de la Rioja. Ha ejercido la docencia por 12 años; forma parte del Grupo de Investigación Ciencia del Mantenimiento CIMANT.

Julio Eduardo Cajamarca Villa. Ingeniero electrónico por la Universidad Politécnica Salesiana, Master Universitario en Ingeniería Electromecánica por la Universidad Politécnica de Madrid, Master Universitario en Industria 4.0 por la Universidad de la Rioja. Ha ejercido la docencia por 8 años.

Gabriel Vinicio Moreano Sánchez se forma como Ingeniero en Electrónica y Control por parte de la Escuela Politécnica Nacional (Quito - Ecuador) donde obtiene el título con mención de excelencia "Cum Laude", institución donde también alcanza la suficiencia en el idioma inglés (2012); continúa su formación profesional en la Universidad Politécnica de Madrid (UPM) donde consigue el título de Máster Universitario en Automática y Robótica en el año 2017; en el mismo año logra el título de Máster Universitario en Diseño y Gestión de Proyectos Tecnológicos de la Universidad Internacional de la Rioja (UNIR). Actualmente cursa el programa doctoral en Ingeniería en la Pontifica Universidad Católica de Perú desde marzo 2021

ISBN: 978-9942-636-32-4

9789942636324