



Programación Modular

MANUAL

Unidad 1

UNIVERSIDAD DE LAS FUERZAS ARMADAS ESPE

NRC: 17832

Introducción a la Programación

Ing.Edgar Solis

2024

ÍNDICE

Unidad 1.....	4
1.1 Introducción a la Programación de Computadores.....	4
1.1.1. Paradigmas de Programación.....	4
1.1.2 Compilación e intérprete.....	8
1.1.3 Lenguaje de programación.....	10
1.1.4 IDE vs editores.....	13
1.1.5 Concepto de Programa.....	16
2.1 Algoritmos.....	21
2.1.1 Definición y características de algoritmos.....	21
2.1.2 Diseño de Algoritmos Utilizando Técnicas de Representación.....	23
2.1.3 Prueba de Escritorio.....	31
3.1 Variables y Tipos de Datos.....	34
3.1.1 Tipos de datos primitivos.....	34
3.1.2 Variables.....	36
3.1.3 Constantes.....	39
4.1 Operadores y expresiones.....	42
4.1.1. Operadores de asignación.....	42
4.1.2. Operadores aritméticos.....	44
4.1.3 Operadores Relacionales.....	47
4.1.4 Operadores Lógicos.....	49
4.1.5 Precedencia de los operadores.....	52
4.1.6 Evaluación de expresiones.....	55
4.1.7 Conversión de tipos de datos.....	57
4.1.8 Gestión de errores (sintaxis, semánticos, tiempos de ejecución).....	59
5.1 Entrada y salida de datos.....	60
5.1.1. Entrada.....	61
5.1.2. Salida.....	63
6.1 Estructura Para - For.....	66
6.2 Estructura repetitivas Bucle - While.....	77
Unidad 2.....	133
1. Subprogramas : Funciones o Procedimientos.....	133
1.1. Definición de Subprograma.....	133
1.2. Declaración , implementación y llamada de Subprogramas.....	138
1.3. Argumentos y Parámetros.....	142
1.4 Argumentos por posición:.....	143
1.5 Argumentos por nombre:.....	144
1.6 Ámbito de las Variables.....	145
1.7 Funciones de librerías o módulos.....	148
1.8 Recursividad.....	150
1.9 Creación de librerías o módulos.....	153
2. Arreglos.....	157
2.1. Arreglos unidimensionales.....	157
2.2. Arreglos bidimensionales.....	160

2.3. Arreglos multidimensionales.....	163
2.4. Paso de Arreglos a funciones.....	165
2.5. Algoritmos de Ordenación (selección,intercambio y burbuja).....	167
2.6. Algoritmos de búsqueda.....	171
Unidad 3.....	202
Cadenas y/o Strings.....	202
1. Concepto.....	202
2. Declaración e inicialización de variables.....	204
3. Entrada/Salida.....	207
4. Asignación.....	208
5. Longitud y concatenación.....	211
6. Comparación.....	214
7. Conversión.....	216
8. Inversión.....	218
9. Sub cadenas.....	220
10. Búsqueda.....	224
11. Cadenas y/o strings como parámetros de funciones.....	227
Introducción a tipos de datos abstractos(TDA).....	232
12. Declaración.....	232
13. Definición de variables.....	236
14. Acceso.....	239
15. Almacenamiento de información.....	242
16. Lectura de información.....	246
17. Recuperación de la información.....	249
Entrada y salida por archivos.....	252
18. Archivos de texto.....	252
19. Archivos Binarios.....	255
BIBLIOGRAFÍA.....	259
SOLUCIONARIO.....	261

Unidad 1

1.1 Introducción a la Programación de Computadores

1.1.1. Paradigmas de Programación

Es un conjunto de principios y directrices que define un enfoque particular para diseñar, estructurar y escribir código. Cada paradigma impone una forma única de pensar sobre cómo debería desarrollarse el software y cómo interactúan sus componentes.

Tipos de paradigmas de programación

1. Programación imperativa: Este es uno de los paradigmas de programación más antiguos y fundamentales. En la programación imperativa, se describen detalladamente los pasos que debe seguir el programa para alcanzar un estado deseado. Los lenguajes de programación como C y Pascal son ejemplos clásicos de este paradigma.
2. Programación declarativa: Esta se centra en describir el resultado deseado sin especificar los pasos detallados para llegar allí. Dos subcategorías comunes son la programación funcional y la lógica. Lenguajes como Haskell y Prolog son representativos de estas subcategorías, respectivamente.
3. Programación orientada a objetos (OOP): Los programas se estructuran alrededor de «objetos», que son instancias de clases que encapsulan datos y métodos. Este enfoque se basa en conceptos como la encapsulación, la herencia y el polimorfismo. Lenguajes como Java, Python y C++ son conocidos por seguir este paradigma.
4. Programación orientada a aspectos (AOP): Es un paradigma que permite modularizar aspectos transversales a través de la aplicación, como el registro o la seguridad, que no encajan fácilmente en un paradigma OOP tradicional. AspectJ es un ejemplo.

5. Programación funcional: Se centra en tratar las computaciones como evaluaciones de funciones matemáticas y evita el cambio de estado y datos mutables. Lenguajes como Lisp, Haskell y Erlang son conocidos por seguir este paradigma.
6. Programación lógica: En este paradigma, la lógica formal se utiliza para expresar reglas y relaciones que gobiernan el problema en cuestión. Prolog es un ejemplo clásico.
7. Programación reactiva: Se centra en construir sistemas que reaccionan automáticamente a cambios en su entorno, mediante la propagación de cambios. RxJava y ReactiveX son bibliotecas que permiten la programación reactiva en lenguajes como Java.
8. Programación basada en eventos: Este es otro de los paradigmas de programación; aquí el flujo del programa está determinado por eventos, como clics del ratón o entradas de usuario. JavaScript en el contexto de desarrollo web es un ejemplo común.

Autoevaluación 1

1. ¿Cuál de los siguientes enunciados describe mejor el paradigma de programación orientada a objetos?

- a) Utiliza funciones como unidades principales de organización del código.
- b) Se centra en la manipulación directa de la memoria del sistema.
- c) Organiza el código en torno a la interacción entre objetos que encapsulan datos y comportamientos.
- d) Se basa en la ejecución secuencial de instrucciones paso a paso.

2. ¿Cuál de los siguientes enunciados describe mejor el paradigma de programación funcional?

- a) Se centra en la manipulación directa de la memoria del sistema.
- b) Organiza el código en torno a la interacción entre objetos que encapsulan datos y comportamientos.
- c) Utiliza funciones como unidades principales de organización del código, tratando la programación como una serie de evaluaciones de funciones matemáticas.
- d) Se basa en la ejecución secuencial de instrucciones paso a paso.

3. ¿Cuál de las siguientes afirmaciones describe mejor el paradigma de programación declarativa?

- a) Organiza el código en torno a la interacción entre objetos que encapsulan datos y comportamientos.
- b) Utiliza funciones como unidades principales de organización del código, tratando la programación como una serie de evaluaciones de funciones matemáticas.
- c) Se centra en la manipulación directa de la memoria del sistema.
- d) Especifica qué resultados se desean y no cómo obtenerlos, dejando la implementación de los detalles al sistema.

4. ¿Cuál de las siguientes afirmaciones describe mejor el paradigma de programación lógica?

- a) Organiza el código en torno a la interacción entre objetos que encapsulan datos y comportamientos.

- b) Utiliza funciones como unidades principales de organización del código, tratando la programación como una serie de evaluaciones de funciones matemáticas.
- c) Se centra en la manipulación directa de la memoria del sistema.
- d) Define relaciones y reglas lógicas para expresar la relación entre los datos y permite realizar consultas lógicas.

5. ¿Cuál de las siguientes afirmaciones describe mejor el paradigma de programación imperativa?

- a) Organiza el código en torno a la interacción entre objetos que encapsulan datos y comportamientos.
- b) Utiliza funciones como unidades principales de organización del código, tratando la programación como una serie de evaluaciones de funciones matemáticas.
- c) Se centra en la manipulación directa de la memoria del sistema.
- d) Se centra en el uso de instrucciones que modifican el estado del programa, controlando el flujo de ejecución a través de secuencias de comandos.

La importancia de entender los paradigmas de programación

1. Adaptabilidad a diferentes problemas: Cada paradigma tiene sus fortalezas y debilidades, y entender varios paradigmas le permite a los programadores seleccionar el enfoque más adecuado para resolver problemas específicos.
2. Flexibilidad en el desarrollo de software: La comprensión de múltiples paradigmas de programación le brinda a los desarrolladores la flexibilidad de elegir herramientas y lenguajes que mejor se adapten a los requisitos de un proyecto particular.

3. Facilita el aprendizaje continuo: Los paradigmas evolucionan con el tiempo, y comprender varios paradigmas facilita el aprendizaje continuo y la adaptación a nuevas tecnologías y metodologías.

Los paradigmas de programación son las filosofías que guían la forma en la que los desarrolladores crean software. Al entender y apreciar estos paradigmas, los programadores pueden ampliar sus horizontes y convertirse en profesionales más versátiles y efectivos.

La elección del paradigma adecuado para un proyecto específico es fundamental para el éxito del desarrollo de software en un mundo tecnológicamente diverso y en constante cambio.

1.1.2 Compilación e intérprete

Compilación :

Un compilador es una herramienta que toma el código fuente escrito en un lenguaje de programación (como C, C++, Java) y lo traduce a un lenguaje de bajo nivel o código máquina que la computadora puede entender directamente.

Ventajas:

El programa compilado generalmente se ejecuta más rápido, ya que ya está traducido a código de máquina.

Se pueden realizar optimizaciones durante el proceso de compilación.

Intérprete

Un intérprete es un programa que lee y ejecuta el código fuente directamente. Puede traducir y ejecutar cada instrucción del programa de manera secuencial.

Ventajas:

Mayor portabilidad, ya que el código fuente puede ejecutarse en cualquier sistema que tenga el intérprete instalado.

Facilita el desarrollo interactivo y la depuración, ya que se pueden probar fragmentos de código sin necesidad de compilar todo el programa.

Autoevaluación 2

1. ¿Cuál es la fase en la que un compilador analiza la estructura del programa fuente?

- a) Optimización.
- b) Síntesis.
- c) Análisis léxico.
- d) Generación de código.

2. ¿Cuál de las siguientes afirmaciones es verdadera acerca de los intérpretes?

- a) La traducción a código máquina se realiza antes de la ejecución.
- b) Los intérpretes son generalmente más eficientes que los compiladores.
- c) Los errores léxicos y sintácticos se detectan durante la ejecución.
- d) El código fuente se traduce completamente antes de ejecutarse.

3. ¿Cuál es la principal diferencia entre un compilador y un intérprete?

- a) Un compilador traduce el código fuente a código máquina, mientras que un intérprete ejecuta el código directamente.

- b) Ambos son términos intercambiables y no hay diferencia.
- c) Un compilador ejecuta el código directamente, mientras que un intérprete traduce el código a código máquina.
- d) No hay diferencia significativa entre ellos.

4. ¿Cuál es una ventaja específica del uso de un intérprete en términos de portabilidad del código fuente?

- a) El código fuente se traduce a código máquina.
- b) Permite la ejecución en cualquier sistema que tenga el intérprete instalado.
- c) Facilita el desarrollo interactivo.
- d) Permite la optimización durante el proceso de compilación.

5. ¿Cuál es una ventaja del uso de un compilador en comparación con un intérprete?

- a) Mayor portabilidad del código fuente.
- b) Facilita el desarrollo interactivo.
- c) El programa compilado generalmente se ejecuta más rápido.
- d) Permite probar fragmentos de código sin compilar.

1.1.3 Lenguaje de programación

Se lo conoce así a aquel lenguaje destinado a la construcción de otros programas informáticos. Su nombre se debe a que comprende un lenguaje formal que está diseñado para organizar algoritmos y procesos lógicos que serán luego llevados a cabo por un

ordenador o sistema informático, permitiendo controlar así su comportamiento físico, lógico y su comunicación con el usuario humano.

Dicho lenguaje está compuesto por símbolos y reglas sintácticas y semánticas, expresadas en forma de instrucciones y relaciones lógicas, mediante las cuales se construye el código fuente de una aplicación o pieza de software determinada. Así, puede llamarse también lenguaje de programación al resultado final de estos procesos creativos.

La implementación de lenguajes de programación permite el trabajo conjunto y coordinado, a través de un conjunto afín y finito de instrucciones posibles, de diversos programadores o arquitectos de software, para lo cual estos lenguajes imitan, al menos formalmente, la lógica de los lenguajes naturales.

No deben confundirse, sin embargo, con los distintos tipos de lenguaje informático. Estos últimos representan una categoría mucho más amplia, en donde están contenidos los lenguajes de programación y muchos otros protocolos informáticos, como el HTML de páginas web.

Autoevaluación 5

1. ¿Qué es la microcomputadora?

- a) Es una computadora pequeña
- b) Es un programa de codificación
- c) Es una página web
- d) Se lo utiliza en los bucles for y while

2. ¿Qué es el software?

- a) Conjunto de programas que permite a la computadora no realizar tareas
- b) Conjunto de programas que permite a la computadora realizar tareas
- c) Conjunto de programas que permite a la computadora y a la tablet realizar tareas
- d) Conjunto de programas que permite al código realizar tareas

3. ¿Qué significan las siglas HTML?

- a) Es Hypertext Markup Language y es lo que compone la web.
- b) Es Hypertext Markup Language y es lo que puede componer la web.
- c) Es Hypertext Markup Language y es lo que vamos a ver en la web.
- d) Es Hypertext Markup Language y es lo que compone la web.

4. ¿Qué es la IA?

- a) Un robot con apariencia humana
- b) Debe ver con el if y else en python
- c) Es la disciplina y un conjunto de capacidades cognoscitivas e intelectuales expresadas por sistemas informáticos
- d) Disciplina y no es conjunto de capacidad cognitiva que expresa un fin informático.

5. ¿Qué es código de máquina?

- a) Sistema que interpreta circuitos y permite a la máquina analizarlos
- b) Sistema que interpreta circuitos y permite a la máquina analizarlos y autodestruirnos si no convencen
- c) Sistema que interpreta circuitos y permite a la máquina analizarlos para la ejecución de un programa
- d) Sistema que interpreta circuitos y permite a la máquina analizarlos en python.

1.1.4 IDE vs editores

Los IDEs (Entornos de Desarrollo Integrados) son aplicaciones completas que ofrecen un conjunto integral de herramientas para el desarrollo de software. Incluyen un editor de texto, compilador/intérprete, herramientas de depuración y otras utilidades integradas.

Los editores de texto, por otro lado, se centran principalmente en la edición de código. Ofrecen funcionalidades básicas como resaltado de sintaxis, autocompletado y búsqueda de texto, pero no incluyen herramientas de compilación o depuración. La elección entre un IDE y un editor de texto suele depender de las necesidades del proyecto, las preferencias personales del desarrollador y el flujo de trabajo específico requerido para el desarrollo de software.

1.2. Estrategias para solucionar problemas de programación

1.2.1. Fases de resolución de problemas

La resolución de problemas es un proceso esencial en la programación y en la vida cotidiana. Incluye diversas fases y técnicas para abordar desafíos de manera efectiva, además, no solo implica encontrar soluciones, sino también desarrollar la habilidad de elegir

la mejor estrategia para cada situación. La práctica constante mejora la capacidad de enfrentar desafíos de manera sistemática y eficaz. (Morillo et al., 2014)

Las fases de resolución de problemas según Morillo en Metodologías Analíticas y Heurísticas para la Solución del Problema de Programación se dividen en los siguientes pasos:

1.2.1.1. Entender el problema. Lo que implica comprender completamente los requisitos del problema identificar las entradas y salidas esperadas y por último definir los límites y restricciones del problema.

1.2.1.2. Planificación y diseño. Se basa en elegir la estrategia de resolución de problemas, diseñar la estructura general del programa y descomponer el problema en partes más pequeñas lo que se conoce como descomposición modular y finalmente seleccionar algoritmos y estructuras de datos apropiadas.

1.2.1.3. Codificación. Consta de traducir el diseño en código, escribir y organizar el código de manera clara y utilizar buenas prácticas de codificación.

1.2.1.4. Pruebas. Se trata de probar el programa con diferentes conjuntos de datos, verificar si el programa produce resultados correctos e identificar y corregir posibles errores.

1.2.1.5. Depuración. Localizar y corregir errores (bugs) en el código utilizando herramientas de depuración para facilitar el proceso.

1.2.1.6 Optimización. Mejorar el rendimiento del programa si es necesario optimizando algoritmos o estructuras de datos para lograr eficiencia.

1.2.1.7. Documentación. Crear documentación clara y completa en donde se explique el propósito, el funcionamiento y el uso del programa.

1.2.1.8. Implementación. Integrar el programa en el entorno o sistema deseado asegurando que todos los requisitos del sistema se cumplan.

1.2.1.9. Mantenimiento. Realizar actualizaciones y correcciones según sea necesario en donde se pueda mejorar o añadir funcionalidades según los cambios en los requisitos.

Estas fases no son necesariamente lineales y pueden involucrar iteraciones. A menudo, los programadores regresan a fases anteriores según sea necesario durante el proceso de desarrollo.

Autoevaluación 3

1. En la fase de "Comprender el Problema", ¿cuál es el propósito principal de realizar un análisis de requisitos?

- a) Identificar patrones en el código.
- b) Definir claramente el problema y sus necesidades.
- c) Implementar directamente la solución en código.
- d) Evaluar la eficacia del código.

2. Durante la fase de "Planificar la Solución", ¿por qué es crucial realizar un diseño de algoritmos antes de comenzar a codificar?

- a) Para analizar la solución en términos de eficacia.
- b) Para definir claramente el problema.
- c) Para traducir la estrategia de solución en código.
- d) Para asegurar una implementación estructurada y eficiente.

3. En la fase de "Implementar la Solución", ¿cuál es el propósito principal de realizar pruebas de unidad en el código?

- a) Dividir el problema en subproblemas.
- b) Evaluar la eficiencia del código.
- c) Verificar que cada componente funcione correctamente.
- d) Analizar el problema y sus requisitos.

4. Al evaluar la eficiencia en la fase correspondiente, ¿por qué se considera importante realizar pruebas de rendimiento en el código?

- a) Identificar patrones en el código.
- b) Medir el tiempo de ejecución y uso de recursos.
- c) Analizar la solución en términos de eficacia.
- d) Definir claramente el problema.

5. ¿Cuál de las siguientes técnicas se emplea en la fase de "Dividir y Conquistar" para abordar problemas complejos?

- a) Fuerza Bruta.
- b) Análisis de complejidad espacial.
- c) Descomponer un problema en subproblemas más manejables.
- d) Reconocimiento de Patrones.

1.1.5 Concepto de Programa

Un programa es un conjunto de instrucciones lógicas y secuenciales diseñadas para realizar tareas específicas en una computadora. Estas instrucciones, escritas en un lenguaje de programación, guían al sistema para ejecutar operaciones precisas, procesar datos y generar resultados. Un programa puede variar en complejidad, desde simples scripts hasta aplicaciones robustas, y su propósito puede abarcar desde realizar cálculos básicos hasta gestionar sistemas complejos. La programación implica la creación y modificación de programas para lograr los objetivos deseados, lo que es esencial en el desarrollo de software y en la automatización de procesos computacionales.

Autoevaluación

1. ¿Qué definición describe mejor un programa informático?

- a) Un dispositivo físico que procesa información.
- b) Un conjunto de instrucciones que se ejecutan en una computadora para realizar una tarea específica.
- c) Un sistema operativo que controla el hardware de una computadora.
- d) Un archivo de texto que contiene datos.

2. ¿Cuál de las siguientes afirmaciones es correcta acerca de los programas informáticos?

- a) Los programas solo pueden ser escritos en lenguaje máquina.
- b) Todos los programas informáticos son gratuitos.
- c) Los programas son conjuntos de algoritmos que no requieren ejecución en una computadora.

d) Los programas pueden ser desarrollados en diferentes lenguajes de programación.

3. ¿Cuál es la función principal de un compilador en el contexto de la programación informática?

- a) Ejecutar el programa.
- b) Traducir el código fuente a código máquina.
- c) Almacenar variables.
- d) Realizar operaciones aritméticas.

4. ¿Qué término se utiliza para referirse a un programa informático diseñado para realizar tareas específicas de forma automática y autónoma?

- a) Compilador.
- b) Software.
- c) Aplicación.
- d) Algoritmo.

1.2.2. Técnicas de resolución de problemas

Son enfoques y estrategias fundamentales para abordar desafíos de manera efectiva durante el desarrollo de software. Una de las principales metodologías es el análisis sistemático, que implica examinar a fondo el problema para comprender sus componentes esenciales y las relaciones entre ellos.

La descomposición de problemas en partes más manejables es otra técnica clave. Además, el uso de patrones de diseño, que son soluciones probadas y eficientes para problemas comunes, proporciona una guía estructurada para el diseño de software.

La resolución de problemas en programación es un proceso que implica varios pasos:

Definición del problema

Comprender claramente el problema que estás tratando de resolver.

Análisis del problema

Comprender los requisitos del problema, los datos de entrada y salida esperados y cualquier restricción que debas considerar.

Descomposición del problema

Dividir el problema en partes más pequeñas y manejables.

Diseño o desarrollo de un algoritmo

Diseñar el algoritmo que resolverá el problema de manera eficiente. Un algoritmo es una secuencia de pasos que describe la solución de manera clara y detallada.

Selección de estructuras de control

Las estructuras de control, como bucles y condicionales, son herramientas esenciales para guiar el flujo de ejecución de un programa.

Transformación del algoritmo en un programa (codificación).

1.2.2.7. Ejecución y validación del programa.

Es importante recordar que la resolución de problemas en programación es un proceso iterativo. Es normal encontrarse con desafíos y enfrentar errores a lo largo del camino. Sin embargo, cada error es una oportunidad de aprendizaje.

También una de las técnicas es un diagrama de flujo que es una representación gráfica de un algoritmo o proceso. Estos diagramas utilizan símbolos con significados bien definidos que representan los pasos del algoritmo y representan el flujo de ejecución mediante flechas que conectan los puntos de inicio y fin.

Autoevaluación 4

1. ¿Qué técnica se centra en la representación visual de un algoritmo utilizando cajas para representar pasos y flechas para indicar el flujo?

- a) Diagrama de flujo
- b) Árbol de decisión
- c) Mapa conceptual
- d) Grafo

2. ¿Cuál de los siguientes enfoques se utiliza para evitar la repetición de cálculos y mejorar la eficiencia al almacenar resultados previos?

- a) Programación dinámica
- b) Programación lineal
- c) Programación reactiva
- d) Programación paralela

3. ¿Qué técnica se utiliza para iterar sobre una colección de elementos sin tener que preocuparse por la implementación específica de la estructura de datos?

- a) Búsqueda binaria
- b) Iteración recursiva
- c) Iteradores
- d) Programación concurrente

4. ¿Cuál es el propósito principal de las pruebas unitarias en el desarrollo de software?

- a) Evaluar el rendimiento del programa
- b) Verificar que cada unidad de código funciona como se espera.
- c) Realizar pruebas de integración entre módulos.
- d) Realizar codificación

5. ¿Qué es la refactorización de código?

- a) Escribir código desde cero.
- b) Optimizar el código existente sin cambiar su funcionalidad.
- c) Documentar el código para futuras referencias.
- d) Son binarios

2.1 Algoritmos

2.1.1 Definición y características de algoritmos

Un algoritmo es un conjunto ordenado y finito de pasos o reglas definidas que describe cómo realizar una tarea o resolver un problema en particular. Los algoritmos son

esenciales en la informática y otras disciplinas para lograr la automatización y la ejecución eficiente de procesos.

2.1.1.1. Características

Finitud: Un algoritmo debe estar compuesto por un número finito de pasos. Debe finalizar después de ejecutar un número determinado de instrucciones.

Precisión: Cada paso del algoritmo debe ser claro, preciso y sin ambigüedades. Las instrucciones deben ser entendibles y no dar lugar a interpretaciones diversas.

Entrada: Un algoritmo puede recibir cero o más entradas. Estas entradas son los datos sobre los cuales el algoritmo opera para producir un resultado.

Salida: Un algoritmo debe generar al menos una salida. La salida es el resultado o solución del problema para el cual se diseñó el algoritmo.

Eficiencia: Los buenos algoritmos deben ser eficientes en términos de uso de recursos, como tiempo y espacio. Deben realizar la tarea de manera rápida y utilizando la menor cantidad posible de recursos.

Determinismo: Un algoritmo debe ser determinista, lo que significa que, dado un conjunto particular de entradas, siempre producirá la misma salida. No debe haber lugar para la aleatoriedad en un algoritmo determinista.

Generalidad: Un algoritmo debe ser lo suficientemente general como para aplicarse a una variedad de instancias del problema para el cual fue diseñado. Debe ser adaptable y no limitarse a casos específicos.

Descomposición: La descomposición es el principio de dividir un problema grande en subproblemas más pequeños y manejables. Cada subproblema se resuelve por separado utilizando un algoritmo específico.

Orden: Los pasos del algoritmo deben estar ordenados de manera lógica. La secuencia de operaciones debe seguir un flujo lógico y coherente.

Claridad: La descripción del algoritmo debe ser clara y comprensible para cualquier persona que esté familiarizada con el problema. La claridad facilita la implementación y la comprensión del algoritmo.

Autoevaluación 6

1. ¿Cómo se define un algoritmo?

- a) Un conjunto caótico de pasos.
- b) Un conjunto ordenado y finito de pasos o reglas.
- c) Un conjunto infinito de reglas.
- d) Un conjunto de pasos indefinidos.

2. ¿Cuál es una característica esencial del concepto de "finitud" en un algoritmo?

- a) Puede tener un número infinito de pasos.
- b) Debe tener un número finito de pasos.
- c) Puede tener pasos indefinidos.
- d) No es necesario que finalice.

3. ¿Qué significa que un algoritmo sea "Determinista"?

- a) Puede producir diferentes salidas para las mismas entradas.
- b) Siempre producirá la misma salida para un conjunto particular de entradas.
- c) Se basa en la aleatoriedad.
- d) No tiene un resultado predecible.

4. ¿Cuál de las siguientes afirmaciones es correcta con respecto a la "Descomposición" en un algoritmo?

- a) Es la idea de mantener un problema grande sin dividir.
- b) Consiste en combinar varios problemas pequeños en uno grande.

- c) Implica dividir un problema grande en subproblemas más pequeños y manejables.
- d) No tiene relación con la resolución de problemas.

5. ¿Por qué es importante la "Claridad" en la descripción de un algoritmo?

- a) Facilita la confusión.
- b) Complica la implementación.
- c) Ayuda en la implementación y comprensión del algoritmo.
- d) No tiene impacto en la comprensión del algoritmo.

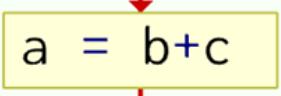
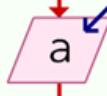
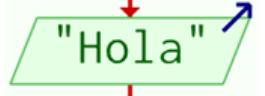
2.1.2 Diseño de Algoritmos Utilizando Técnicas de Representación

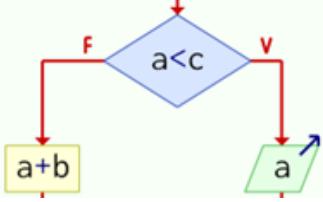
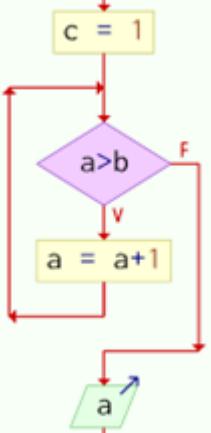
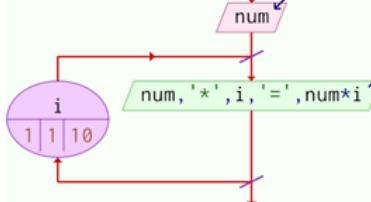
Son herramientas para modelar datos y operaciones de manera efectiva. Algunas técnicas son:

2.1.2.1. Diagramas de Flujo

Representaciones gráficas que usan formas geométricas para representar diferentes pasos o procesos, siendo que cada forma representa una acción específica (por ejemplo, rectángulos para procesos, rombos para decisiones, etc.), mientras que las flechas indican la secuencia de ejecución.

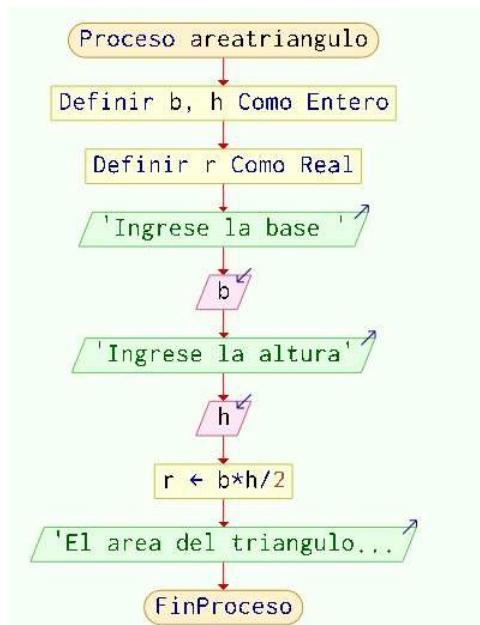
Una herramienta que puede ayudar a la creación de estos diagramas sería “PSeint”, de la cual nos vamos a basar en sus figuras para establecer los pasos y procesos.

Acciones o procesos	Datos de entrada	Datos de salida
 <p>Esta figura se usa para asignar datos, que pueden ser de diferente tipo, o procesos a una variable, a su vez definiendo la propia variable.</p>	 <p>Se usa esta figura cuando se quiere ingresar datos, a los cuales se les tiene que asignar una variable que los represente.</p> <p>Las variables por lo general se representan con letras minúsculas.</p>	 <p>Esta figura representa a los datos de salida, en si los datos que se imprimirán en pantalla</p>

Condicional	Bucle "While"	Bucle "For"
 <p>Esta figura es una condicional, en el rombo se ubica la condicional, si es falso o verdadero se ejecutan acciones diferentes.</p>	 <p>Para los bucles a trabajar se tiene que tener en cuenta que deben de constar de tres elementos:</p> <ul style="list-style-type: none"> . – Valor inicial . – Condición . – Incremento <p>El bucle "While" funciona ejecutando una misma acción indefinidamente siempre y cuando la condición sea verdadera, por lo tanto, el bucle se termina cuando la condición sea falsa.</p>	 <p>Los bucles "For" funcionan de manera semejante a un bucle "While", solo que a estos se le puede configurar la cantidad de veces que se repita el bucle.</p> <p>A estos por lo general se les da valor inicial, una condición u otro valor hasta donde se va a llegar, esto como rango, y por último la manera en que avanza valor a valor a lo largo de la lista.</p>

Ejemplo

- Realice un algoritmo con la técnica de diagrama de flujo que permita calcular el área de un triángulo, cuyas dimensiones sean valores que se ingrese por teclado.



2.1.2.2. Pseudocódigo

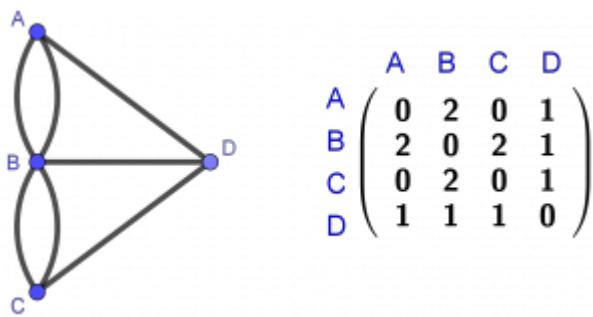
Usa lenguaje simple y estructurado que se parece al código fuente real, aunque no está relacionado a un lenguaje de programación específico, pero ayuda a expresar la lógica del algoritmo antes de implementarlo en un lenguaje de programación.

Un ejemplo para el mismo sería:

```
1 Algoritmo Tabla_de_multiplicar_del_1_al_10
2     Definir num Como Entero
3     Escribir 'Ingrese un número'
4     Leer num
5     Para i←1 Hasta 10 Con Paso 1 Hacer
6         Escribir num,'*',i,'=',num*i
7     FinPara
8 FinAlgoritmo
```

2.1.2.3. Tablas y Matrices

Utilizan tablas y matrices para organizar datos y visualizar relaciones entre diferentes elementos. Es útil para algoritmos que involucran datos tabulares o estructuras bidimensionales.



2.1.2.4. Grafos

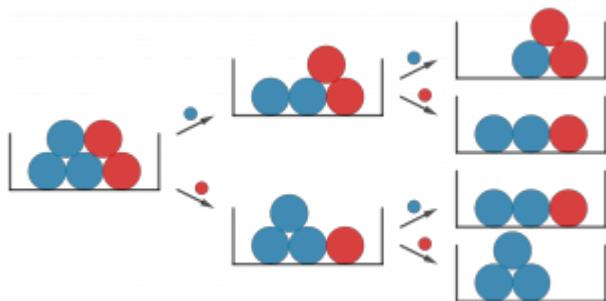
Representar el flujo de control o relaciones entre entidades utilizando grafos.

Los nodos representan estados o eventos, y las aristas indican las transiciones o relaciones entre ellos.



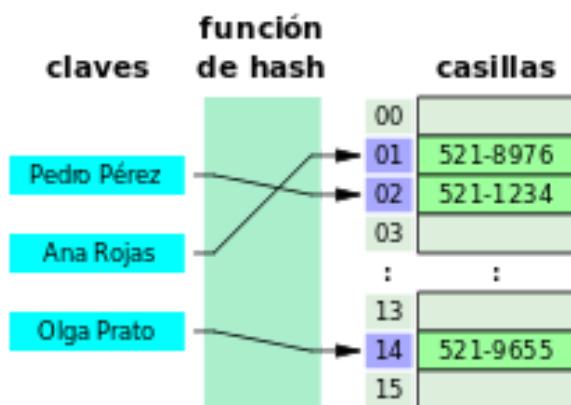
2.1.2.5. Árboles

Los árboles se utilizan para representar jerarquías y relaciones de parentesco entre elementos. Son útiles para estructuras de datos como árboles de búsqueda binaria o árboles de expresión.



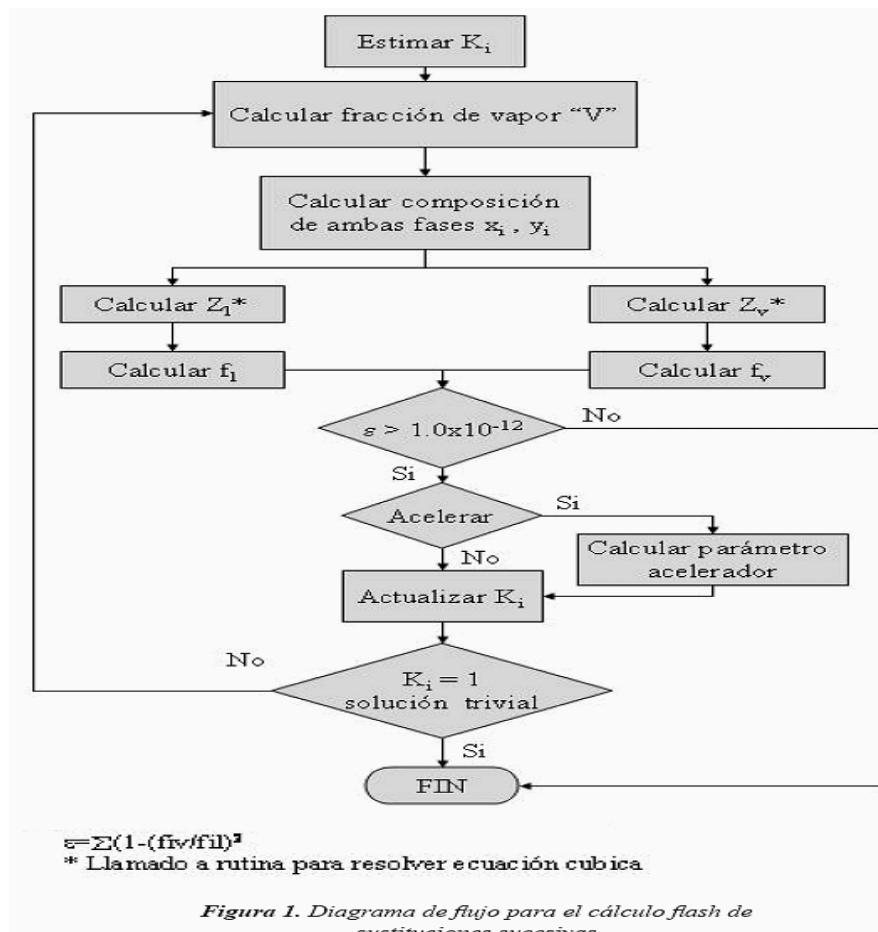
2.1.2.6. Modelado de Datos

Utiliza estructuras de datos para representar información de manera clara y eficiente. Por ejemplo, utilizar listas enlazadas, arreglos, pilas o colas según las necesidades del algoritmo.



2.1.2.7. Modelos Matemáticos

Desarrollar ecuaciones matemáticas que describen el comportamiento del algoritmo.



Autoevaluación 7

1. ¿Cuál de las siguientes figuras se utiliza para asignar datos o procesos a una variable, definiendo la propia variable?

- a) Datos de entrada
- b) Condicional
- c) Condicional
- d) Acciones o procesos

2. ¿Qué concepto se refiere a un lenguaje simple y estructurado que se asemeja al código fuente real, aunque no está vinculado a un lenguaje de programación?

a) Pseudocódigo

b) Sintaxis real

c) Código preliminar

d) Prototipo lógico

3. ¿Qué herramienta es útil para algoritmos que involucran datos tabulares o estructuras bidimensionales?

a) Diagrama de flujo

b) Gráfico de barras

c) Tablas y matrices

d) Diagrama de datos

4. ¿Cuál de las siguientes herramientas se utiliza para representar el flujo de control, donde los nodos representan estados o eventos, y las aristas indican transiciones o relaciones?

a) Matriz de datos

b) Mapa conceptual

c) Diagrama de flujo

d) Grafos

5. ¿Qué estructura se utiliza para representar jerarquías y relaciones de parentesco entre elementos, siendo útil para estructuras de datos?

a) Redes neuronales

b) Listas enlazadas

c) Árboles

d) Grafos de flujo

2.1.3 Prueba de Escritorio

Es un tipo de prueba algorítmica que consiste en la validación y verificación del algoritmo a través de las sentencias (procesos) que lo componen, para determinar sus resultados (salida) a partir de un conjunto determinado de elementos (entradas).

En otras palabras, se centra en analizar y determinar su validez antes de ejecutar el programa para detectar errores.

Consiste en:

1. Tabla que tenga el mismo número de columnas que variables
2. Tantas filas como instrucciones
3. Se llena, poniendo el valor correspondiente en cada celda de acuerdo a las instrucciones

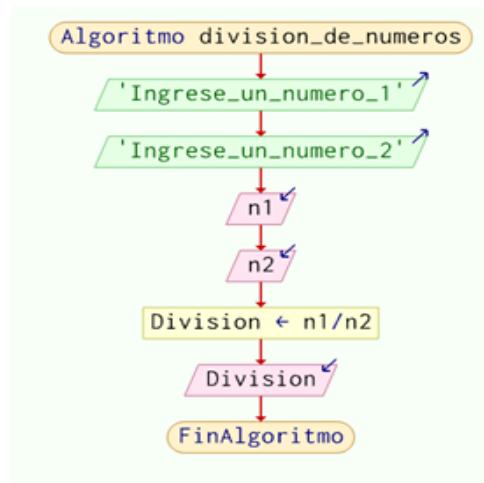
Dependiendo de la complejidad se deben realizar 3 o más pruebas sometidas a diferentes escenarios.

ENTRADA	PROCESO	SALIDA

Ejemplos:

- Solicitar dos números al usuario y obtener la división de dichos números e imprimir el resultado en pantalla.

Diagrama en PSeInt



Prueba de escritorio

ENTRADA	PROCESO			SALIDA
Ingreso de números	n1	n2	División<-n1/n2	Imprime en pantalla
35	35			
5		5		
			35/5	
				"El resultado es: " 7

Código en python

```
import os

n1 = int(input("Ingrese un numero1: "))

n2 = int(input("Ingrese un numero2: "))

Division = n1/n2

print ("El resultado es: ", Division)
```

Ejecución del código

```
<= RESTART: C:/Users/hp/Desktop/Semestr
Ingrese un numero1: 35
Ingrese un numero2: 5
El resultado es: 7.0

<= RESTART: C:/Users/hp/Desktop/Semestr
Ingrese un numero1: 89
Ingrese un numero2: 3
El resultado es: 29.666666666666668
-
```

Autoevaluación 8

1. ¿Con qué otro nombre se les llaman a las sentencias que lo componen?

- a) Procesos
- b) Entrada
- c) Salida
- d) Desarrollo

2. ¿Cuándo se ejecuta?

- a) Después de empezar el programa
- b) Antes de empezar el programa
- c) No es necesario hacerlo
- d) Antes y después

3. ¿De qué partes se compone?

- a) Inicio, desarrolló y fin
- b) Entrada, mecanismo y pantalla
- c) Entrada, procesos y salida

d) Entrada y salida

4. ¿Cuántas pruebas se debe realizar?

- a) 1 prueba
- b) 3 pruebas o más
- c) Las que sean necesarias
- d) No se necesita pruebas

5. Al formar la tabla, ¿Cuántas columnas debe haber por cada variable?

- a) Tantas por instrucciones dadas.
- b) Las que necesitemos.
- c) El mismo número de variables.
- d) Una más que las variables dadas.

3.1 Variables y Tipos de Datos

3.1.1 Tipos de datos primitivos

Los tipos de datos simples o primitivos significan que no están compuestos de otras estructuras de datos que se utilizan para representar valores en un programa.

Tabla 1

Tipos de Datos Primitivos con su descripción y memoria

Tipos de Datos	Descripción	Memoria
int	Cantidad entera con signo	4 bytes o una palabra (varía)

		según compilador).
short	Entero con signo más corto	(2 bytes)
long	Entero con signo más largo	(4 bytes o más)
unsigned int	Entero sin signo	Igual que int (4 byte)
char	Carácter	1 byte.
float	Almacena valores reales en punto flotante, de precisión simple	1 palabra (4 bytes).
double	Almacena valores reales en doble precisión.	2 palabras (8 bytes).
long double	Almacena valores reales en precisión extendida.	(Mayor de double)
bool	Valor booleano verdadero o falso	1 byte
void	Se utiliza para definir una función que no devuelve ningún valor o declarar punteros genéricos	

Autoevaluación 10

6. ¿Cuántos bytes se utilizan para almacenar un valor del tipo de dato int?

- a) 2 bytes
- b) 4 bytes
- c) 8 bytes
- d) Varía según el compilador

7. ¿Cuál es la función del tipo de dato float en programación?

- a) Almacenar valores enteros con signo
- b) Almacena valores reales en punto flotante de precisión simple
- c) Representa caracteres
- d) Define funciones que no devuelven ningún valor

8. ¿Cuál es la diferencia principal entre double y float en términos de precisión de los valores almacenados?

- a) Double tiene el doble de precisión que float
- b) Float tiene el doble de precisión que double
- c) Ambos tienen la misma precisión
- d) Double se utiliza para almacenar caracteres, mientras que float almacena valores reales

9. ¿Cuántos bytes se utilizan para almacenar un valor booleano (bool)?

- a) 1 byte
- b) 2 bytes
- c) 4 bytes
- d) Varía según el compilador

10. ¿Cuál es el propósito del tipo de dato void en programación?

- a) Almacena valores reales en doble precisión
- b) Representa caracteres
- c) Define funciones que no devuelven ningún valor

d) Almacenar valores enteros sin signo

3.1.2 Variables

Una variable es una entidad cuyo valor puede cambiar a lo largo de la ejecución del programa. Entonces podemos encontrar las siguientes variables en nuestro programa de codificación:

int: Para valores enteros

float: Para valores numéricos no enteros, o decimales

char: Para valores tipo carácter

bool: para valores booleanos, es decir verdadero o falso, o también 0 y 1

Ejemplos de aplicación en python:

Uno de los más utilizados en el programa python es el int, que como mencionamos antes sirve para declarar valores enteros por ejemplo podemos tener el siguiente caso cuando queremos ingresar una variable

```
import os  
  
int(input("ingrese un numero entero"))
```

En este caso hemos declarado la variable y hemos proyectado el mensaje que queremos que salga en la consola.

Autoevaluación 9

1. ¿Que es una variable?

a) Es un token que sirve para definir estructuras

- b) Es un espacio en la memoria donde se almacena algún valor
- c) Es un token de programación en C++
- d) Es un valor que no cambia

2. ¿Cuántas variables existen?

- a) 5 variables
- b) 6 variables
- c) 4 variables
- d) Múltiples variables

3. ¿Qué variables fueron mencionadas?

- a) int, float, char, bool
- b) int, float, bool
- c) entero, flotante, char
- d) double, float, unsigned

4. ¿Para qué sirve el int?

- a) Para valores enteros
- b) Para valores decimales
- c) Para valores irracionales
- d) Para valores imaginarios

5. ¿Cuál es la sintaxis para ingresar un dato en python?

- a) int(ent(""))
- b) int(input(""))
- c) ingresar(entero(""))
- d) input(int())

3.1.3 Constantes

Son valores que no cambian durante la ejecución de un programa. Estos valores se utilizan para representar información fija y no modificable, como configuraciones específicas, parámetros fundamentales o valores universales.

Características:

Inmutabilidad:

Tienen un valor fijo que no se altera.

Convenciones de Nomenclatura:

En muchos lenguajes de programación, las constantes se nombran utilizando letras mayúsculas y guiones bajos para separar palabras. Esto ayuda a distinguir las constantes de las variables y facilita la lectura del código.

Enfoque en la Legibilidad y Mantenimiento:

Al dar nombres descriptivos a las constantes, el código se vuelve más comprensible y menos propenso a errores. Además, si se necesita cambiar un valor constante en el futuro, sólo se debe hacer en un lugar, facilitando la actualización del programa.

Ejemplos de constantes en programación:

En C++, se puede definir una constante de la siguiente forma:

```
const int PI = 3.14159;
```

En Java, se define una constante utilizando la palabra clave «final».

Por ejemplo:

```
final int NUMERO MAXIMO = 100;
```

En este caso, el NÚMERO MÁXIMO es una constante y su valor es 100.

En Python, se puede definir una constante utilizando una variable con mayúsculas.

Por ejemplo:

```
PI = 3.14159
```

En PHP, se define una constante utilizando la función «define».

Por ejemplo:

```
define('PI', 3.14159);
```

En los ejemplos vistos, PI es una constante y su valor es 3.14159.

Autoevaluación 10

1. ¿Qué es una constante?

- a) Son valores fijos y no modificables en la ejecución de un programa.
- b) Son valores fijas y modificables en la ejecución.
- c) No son valores fijos y no modificables en la ejecución de un programa.
- d) Ninguna de las anteriores

2. ¿Cómo ayuda a distinguir las constantes de las variables?

- a) No nombra letras mayúsculas y guiones bajos.
- b) Se nombran utilizando letras mayúsculas y guiones bajos para separar palabras
- c) No se nombran caracteres .
- d) Solamente caracteres

3. ¿Cómo se define en Java para una constante?

- a) «final»
- b) (fin(
- c) «final-»
- d) “definir”

4. ¿Cómo se define en C + + para una constante?

- a) const int
- b) const
- c) int
- d) input

5. ¿Qué es inmutabilidad en constantes?

- a) Es un valor fijo que no se altera
- b) Es un valor fijo que si se altera
- c) Valor alterable
- d) Ninguna de las anteriores

4.1 Operadores y expresiones

4.1.1. Operadores de asignación

Son aquellos que nos permiten realizar una operación y almacenar su resultado en una variable y se realiza de derecha a izquierda; ejemplos:

Operador : Se encarga de asignar a la variable de la izquierda el contenido de la derecha.

Operador `+=` : Suma el valor de su derecha a la variable de la izquierda y lo asigna en la izquierda.

Operadores `-=` : Resta y guarda el resultado en la variable inicial.

Operador `*=` : Multiplica el valor de una variable por el de una expresión y el resultado lo guarda en la variable.

Operador `/=` : Divide el valor de una variable por el de una expresión y asigna el resultado de punto flotante en la variable.

Operador `%=` : Divide el valor de dos variables y almacena su resultado en la primera.

Operador : Realiza la comparación de bit a bit entre dos variables y su resultado lo asigna en la primera.

Operador `^=` : Eleva el resultado de una variable a una potencia y lo guarda en la variable.

Operador `<=` : Realiza un desplazamiento aritmético a la izquierda del valor de una variable y lo guarda en la variables.

Autoevaluación 11

1. ¿Qué es un operador de asignación en programación?

- a) Un operador que realiza comparaciones entre valores.

- b) Un operador que asigna un valor a una variable.
- c) Un operador que realiza operaciones matemáticas.
- d) Un operador para simbolizar cosas.

2. ¿Cuál es la diferencia principal entre el operador de asignación simple ("=") y el operador de igualdad ("==") en muchos lenguajes de programación?

- a) El operador "=" compara dos valores, mientras que "==" asigna un valor a una variable.
- b) El operador "=" asigna un valor a una variable, mientras que "==" compara dos valores.
- c) El operador "=" ingresa un valor a una variable, mientras que "==" compara valores.
- d) Ambos operadores realizan la misma función.

3. ¿Qué hace el operador de asignación compuesta "-=" en programación?

- a) Suma el valor de la variable a otro valor.
- b) Asigna el valor de la variable a otro.
- c) Resta el valor de la variable a otro valor.
- d) Asigna una comparación entre variables.

4. ¿Qué sucede si intentas utilizar un operador de asignación compuesta en una variable que aún no ha sido declarada en el programa?

- a) El programa dará un error.
- b) La variable se inicializará con un valor predeterminado.
- c) El programa funcionará correctamente.
- d) Se asigna un valor automáticamente.

5. ¿Por qué es importante considerar el tipo de datos al usar operadores de asignación en programación?

- a) Puede causar errores si se utiliza un operador de asignación incompatible con el tipo de datos de la variable.
- b) No es importante; los operadores de asignación funcionan igual para todos los tipos de datos.
- c) Solo importa en lenguajes de programación específicos.
- d) Porque puede cambiar la dirección de un diagrama de flujo

4.1.2. Operadores aritméticos

Los operadores aritméticos son símbolos utilizados en programación para realizar operaciones matemáticas en variables y valores. Aquí se presenta una breve definición de los operadores aritméticos comunes, junto con ejemplos en varios lenguajes de programación:

1. **Suma (+):** Se utiliza para sumar dos valores.

`a = 5 b = 3 resultado = a + b`

`# resultado ahora es 8`

2. **Resta (-):** Se utiliza para restar el segundo valor del primero.

`int a = 10; int b = 4; int resultado = a - b;`

`// resultado ahora es 6`

3. **Multiplicación (*):** Se utiliza para multiplicar dos valores.

`var x = 7; var y = 3; var resultado = x * y;`

```
// resultado ahora es 21
```

4. **División (/):** Se utiliza para dividir el primer valor por el segundo.

```
float a = 10.0f; float b = 2.5f; float resultado = a / b;
```

```
// resultado ahora es 4.0
```

5. **Módulo (%):** Devuelve el resto de la división del primer valor por el segundo.

```
a = 10 b = 3 resultado = a % b
```

```
# resultado ahora es 1
```

6. **Exponenciación/Potencia (o ^):** ** Eleva el primer valor a la potencia del segundo.

```
a = 2 b = 3 resultado = a**b
```

```
# resultado ahora es 8
```

Estos operadores son fundamentales para realizar cálculos matemáticos en programación y se utilizan en una variedad de contextos para manipular datos numéricos. Es importante recordar la jerarquía de operadores en expresiones matemáticas, o usar paréntesis para controlar el orden de evaluación si es necesario.

Autoevaluación 12

1. **¿Cuál es la función del operador de suma en programación y cómo se utiliza en un ejemplo con variables?**

- a) Realizar la multiplicación de dos valores.
- b) Restar el segundo valor del primero.
- c) Sumar dos valores.

d) Elevar el primer valor a la potencia del segundo.

2. Explique la operación realizada por el operador de módulo (%) y proporcione un ejemplo con variables.

a) Devuelve el resultado de la multiplicación.

b) Devuelve el cociente de la división.

c) Devuelve el resto de la división.

d) Realiza una exponenciación.

3. ¿Cuál es la función del operador de exponenciación/potencia (o ^) y cómo se utiliza en un ejemplo con variables?

a) Sumar dos valores.

b) Restar el segundo valor del primero.

c) Elevar el primer valor a la potencia del segundo.

d) Realizar una multiplicación.

4. Explique la importancia de la jerarquía de operadores en expresiones matemáticas y cómo se puede controlar el orden de evaluación.

a) La jerarquía no afecta la evaluación de expresiones matemáticas.

b) La jerarquía determina el orden de ejecución de las operaciones.

c) La jerarquía sólo es relevante en lenguajes de programación específicos.

d) La jerarquía sólo afecta a los operadores de suma y resta.

5. ¿Cómo se utiliza el operador de multiplicación en programación y cuál es su función principal? Proporcione un ejemplo en un lenguaje de programación.

- a) Se utiliza para realizar la división.
- b) Se utiliza para sumar dos valores.
- c) Se utiliza para multiplicar dos valores.
- d) Se utiliza para calcular el resto de una división.

4.1.3 Operadores Relacionales

Los operadores relacionales comparan datos numéricos, de serie de caracteres o lógicos. El resultado de la comparación, ya sea Verdadero (1) o falso (0), puede utilizarse para tomar una decisión referente al flujo del programa (consulte la sentencia IF).

La Tabla 1 lista los operadores relacionales:

·	EQ o =	Igualdad	$X = Y$
·	NE o #	Desigualdad	$X \neq Y$
·	>< o <>	Desigualdad	$X <> Y$
·	LT o <	Menor que	$X < Y$
·	GT o >	Mayor que	$X > Y$
·	LE o <= o == o #>	Menor que o igual a	$X \leq Y$
·	GE o >= o => o #<	Mayor que o igual a	$X \geq Y$

Cuando en una expresión se utilizan tanto operadores aritméticos como operadores relacionales, las operaciones aritméticas se realizan primero. Por ejemplo, la expresión:

$$X + Y < (T - 1) / Z$$

Las comparaciones de series se efectúa comparando los valores ASCII de los caracteres únicos de cada serie. La serie con el equivalente de código ASCII numérico más alto se considera mayor. Si todos los códigos ASCII son iguales, las series se consideran iguales.

Si las dos series tienen longitudes distintas, pero por lo demás la serie más corta es idéntica al principio de la serie más larga, la serie más larga se considera mayor.

Un espacio se evalúa como menor que cero. Los espacios iniciales y finales son significativos. Si dos series se pueden convertir en valores numéricos, la comparación siempre se realiza numéricamente.

Autoevaluación 13

¿Qué comparan los Operadores Relacionales?

- a) Números
- b) Datos numéricos
- c) Letras
- d) Conectores

1. El operador relacional “ EQ o = “ representa una relación de :

- a) Igualdad
- b) Desigualdad
- c) Menor que
- d) Mayor que

2. El operador relacional “ >< o <> “ representa una relación de :

- a) Igualdad
- b) Desigualdad
- c) Menor que
- d) Mayor que

3. El operador relacional “ NE o # “ representa una relación de :

- a) Igualdad
- b) Desigualdad
- c) Menor que
- d) Mayor que

4. El operador relacional “ GT o >“ representa una relación de :

- a) Igualdad
- b) Desigualdad
- c) Menor que
- d) Mayor que

4.1.4 Operadores Lógicos

Son las herramientas fundamentales que se guían en la lógica Booleana. Lo que va a permitir hacer es modificar o combinar los valores booleanos lo cual va a dar una respuesta de (Verdadero o Falso) lo cual ayudará a tomar decisiones en las condiciones lógicas.

Los operadores lógicos que se van a encontrar son:

- And (Y).- Se lo representará en las codificaciones con el signo de & &. Y si ambas respuestas son verdaderas en las condiciones dará (True) caso contrario dará (False)
- Or (O) Se lo representa en las codificaciones con el signo ||. Dara (True) si al menos una de las condiciones es verdadera y dará (False) si ambas no cumplen con la condición
- Not (No) Se lo representa en las codificaciones con el signo !. Va a dar el opuesto booleano, lo que quiere decir que si la condición es verdadera dará (False) y si es falsa dará (True)

Estos operadores son importantes para controlar el flujo de las condiciones que por lo general son (if, else, elif)

Autoevaluación 14

1. ¿Cómo se representa el operador lógico "And" en las codificaciones?

- a) &&
- b) ||
- c) !
- d) And

2. ¿Cuál es el resultado del operador lógico "Or" si al menos una de las condiciones es verdadera?

- a) False
- b) True

c) ||

d) Or

3. ¿Cuál es la función del operador lógico "Not" en las condiciones?

a) Invertir el valor booleano

b) Combinar valores booleanos

c) Representar el "And" lógico

d) Excluir opciones

4. ¿Cómo se representa el operador lógico "Or" en las codificaciones?

a) &&

b) ||

c) !

d) Or

5. ¿Qué resultado dará el operador lógico "And" si ambas condiciones son verdaderas?

a) True

b) False

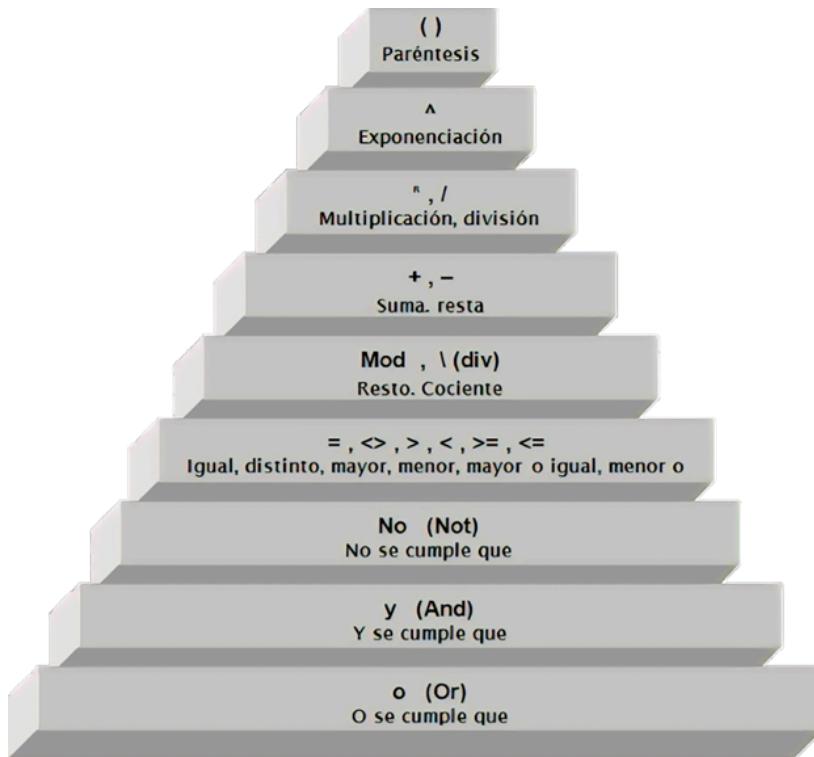
c) ||

d) And

4.1.5 Precedencia de los operadores

En programación, la precedencia de los operadores al igual que las matemáticas corresponde a la secuencia que se sigue una expresión aritmética, tal como ocurre cuando hacemos una operación compuesta.

De la misma manera en la que nosotros razonamos y operamos en un orden específico, la máquina también lo hace, por lo que les da prioridad a ciertas expresiones por sobre otras, es a esto a lo que llamamos como Precedencia de los Operadores pues al momento de programar, requerimos un orden específico de instrucciones para que la máquina lo opere sin mayor problema y al que nosotros podríamos llegar sin mayor problema, para entender mejor esto, a la hora de operar nos damos cuenta que -5×78 nos indica una resta o la multiplicación de un número negativo por un positivo? La computadora va a operar de la manera en la que se está multiplicando un número negativo, por ello lo ideal sería usar paréntesis para ordenar las expresiones y evitar confusiones en caso de ser necesarias. En el ejemplo anterior realmente no cambia mucho el multiplicar el -5 por el 78, pero en expresiones más complejas es necesario dar un orden correcto, especialmente sabiendo que la máquina simplemente se encarga de operar lo que se coloque en el código. Para ello es importante conocer la prioridad de estos operadores y la dirección en la que la computadora los opera.



Prioridad de Operadores

Pseudocódigo. (2013, noviembre 10). *Prioridad entre operadores. Orden de prelación [Imagen]*.

Ejemplo 1

$x = 5$

$y = 10$

$z = 20$

resultado = $x + y * z$

```
print(resultado) # Imprime 205 y no 150 porque la multiplicación tiene mayor  
precedencia que la suma.
```

Ejemplo 2

`x = 5`

`y = 10`

`z = 20`

`resultado = (x + y) * z`

`print(resultado)` # Imprime 300 porque los paréntesis cambian la precedencia de los operadores.

Autoevaluación 15

1. ¿Cuál de los siguientes operadores tiene la mayor precedencia en la mayoría de los lenguajes de programación?

- a) Operadores de asignación (por ejemplo, `=`)
- b) Operadores aritméticos (por ejemplo, `+`, `-`, `*`, `/`)
- c) Operadores de comparación (por ejemplo, `==`, `!=`, `<`, `>`)
- d) Operadores lógicos (por ejemplo, `&&`, `||`)

2. En la expresión `a = b + c * d`, ¿Qué operación se realiza primero en la mayoría de los lenguajes de programación?

- a) `b + c`
- b) `c * d`
- c) `a = b`
- d) Ninguna de las anteriores

3. En la expresión $a = b == c \& \& d$, ¿qué operación se realiza primero en la mayoría de los lenguajes de programación?

- a) $b == c$
- b) $c \&\& d$
- c) $a = b$
- d) $a \&\& b$

4. ¿Cómo puedes cambiar la precedencia de los operadores en una expresión?

- a) Usando el operador !
- b) Usando el operador ~
- c) Usando paréntesis ()
- d) No puedes cambiar la precedencia de los operadores Ninguna de las anteriores

5. En la expresión $a = b || c \&\& d$, ¿qué operación se realiza primero en la mayoría de los lenguajes de programación?

- a) $b || c$
- b) $c \&\& d$
- c) $a = b$
- d) Ninguna de las anteriores

4.1.6 Evaluación de expresiones

La evaluación de expresiones en programación es un proceso esencial en el cual las expresiones, compuestas por operadores, variables y constantes, son calculadas para

obtener un resultado. En Python, este proceso sigue reglas específicas de precedencia de operadores, permitiendo realizar operaciones matemáticas y lógicas de manera eficiente. Además, la inclusión de variables en expresiones facilita la adaptabilidad y flexibilidad del código, ya que los valores de estas variables son sustituidos durante la evaluación.

La comprensión de la evaluación de expresiones es crucial para programadores, ya que es fundamental para realizar cálculos precisos y manipulación de datos en programas Python, contribuyendo a la versatilidad de este lenguaje de programación.

Ejemplo 1: (Evaluación de expresión matemática)

Expresión

```
resultado = 5 + 3 * 2
```

Evaluación

Primero se realiza la multiplicación y luego la suma

*# $3 * 2 = 6$, luego $5 + 6 = 11$*

```
print(resultado) # Salida esperada: 11
```

Ejemplo 2: (Evaluación de expresiones con variables)

Expresiones con variables

```
a = 7
```

```
b = 3
```

```
c = a * b + 2
```

Imprimir el resultado

```
print(c) # Salida esperada: 23
```

4.1.7 Conversión de tipos de datos

La conversión de tipos de datos también conocida como "cast" o "type casting" es la acción de cambiar un tipo de dato a otro. El método para utilizar el cast depende de cada lenguaje de programación. Estas transformaciones son útiles y necesarias cuando se trabaja con diferentes tipos de datos y se necesita que un valor tenga un formato o estructura particular, de tal manera se pueda seguir con las operaciones específicas. Estas operaciones pueden ser generadas automáticamente por el lenguaje de programación, o explícitamente por el programador.

Ejemplo.

Para Python:

Conversión de cadena a entero

```
cadena número = "123"
```

```
numero = int(cadena número)
```

Conversión de entero a cadena

```
número = 456
```

```
cadena numero = str(numero)
```

Conversión de número a punto flotante

```
entero = 789
```

```
flotante = float(entero)
```

Autoevaluación 16

1. ¿Qué es el casting de datos?

- a) Un proceso para fundir objetos de metal.
- b) La creación de moldes para fabricar componentes electrónicos.
- c) Un método para convertir un tipo de dato en otro.
- d) Una función para realizar cálculos matemáticos.

2. En Python, ¿cuál de las siguientes afirmaciones describe mejor el casting implícito?

- a) Es necesario especificar explícitamente la conversión de tipos de datos.
- b) Es una característica exclusiva de Python 2.
- c) Sólo se aplica a lenguajes de programación específicos.
- d) La conversión de tipos de datos se realiza automáticamente por el intérprete.

3. En Python, ¿cuál es la función utilizada para el casting explícito de enteros?

- a) convertir entero()
- b) int()
- c) to_int()
- d) cast_int()

4. ¿Cuál es el riesgo asociado con el casting de datos?

- a) Pérdida de información o posible error si la conversión no es válida.
- b) No hay riesgos, ya que todas las conversiones son seguras.

- c) El casting siempre se realiza correctamente sin riesgos.
- d) Sólo se aplica a lenguajes de programación antiguos.

5. En Python, ¿en qué situación se utiliza comúnmente el casting de datos?

- a) Cuando se necesita fundir metales en una aplicación.
- b) Para convertir datos entre diferentes tipos en un programa.
- c) Sólo en situaciones de emergencia.
- d) Exclusivamente en operaciones de entrada y salida.

4.1.8 Gestión de errores (*sintaxis, semánticos, tiempos de ejecución*)

Errores de Sintaxis: Son equivalentes a errores gramaticales en un lenguaje de programación. Un ejemplo sería si un programador olvida colocar un punto y coma al final de una línea en ciertos lenguajes, eso se consideraría un error de sintaxis.

Errores Semánticos: Estos suceden cuando el código de un programador no coincide con la intención original. Un ejemplo sería si un programador usa accidentalmente un signo de suma en lugar de un signo de multiplicación, eso se consideraría un error semántico.

Errores de Tiempo de Ejecución: Estos errores suceden mientras el programa de un programador se está ejecutando. Un ejemplo común es cuando un programador intenta dividir un número por cero.

Para manejar estos errores, los programadores escriben código adicional para verificar si ha ocurrido un error y luego deciden qué hacer a continuación. Esto puede incluir corregir el problema, informar al usuario o detener el programa de manera segura.

Ejemplo 1

```
# Olvidamos poner los dos puntos al final de la declaración if, lo cual es requerido en Python

numero = 10

if numero > 5

    print("El número es mayor que 5")
```

Ejemplo 2

```
# Este es un ejemplo de un error de tiempo de ejecución en Python

# Intentamos dividir un número por cero, lo cual no está permitido

numero = 10

print(numero / 0)
```

5.1 Entrada y salida de datos

Cualquier programa informático debe comunicarse con los usuarios, tanto para recibir la información de entrada como para devolver los resultados de su ejecución. Esto es lo que se conoce con el nombre de procesos de entrada y salida de datos.

Un programa está compuesto por un bloque de declaraciones y otro de instrucciones. El bloque de instrucciones está formado por tres partes, podemos localizarlas según su función:

Entrada de datos: instrucciones que almacenan en la memoria interna datos procedentes de un dispositivo externo (teclado, ratón,...)

Proceso o algoritmo: instrucciones que modifican los objetos de entrada y, en ocasiones, crean otros nuevos.

Salida de resultados: conjunto de instrucciones que toman los datos finales de la memoria interna y los envían a los dispositivos externos (pantallas o dispositivos de almacenamiento).

5.1.1. Entrada

En lenguajes de programación, se utilizan funciones o métodos específicos para capturar la entrada.

Métodos comunes de entrada y salida:

- Teclado (entrada): Los programas solicitan al usuario que ingrese datos a través del teclado.
- Pantalla (salida): Los resultados se muestran en la pantalla para que el usuario los vea.
- Archivos (entrada/salida): Los programas pueden leer datos desde archivos y escribir resultados en archivos.
- Red (entrada/salida): La comunicación a través de redes permite la entrada y salida de datos a través de conexiones.

Manejo de errores:

Es esencial incluir mecanismos para manejar posibles errores durante la entrada y salida de datos, como entradas incorrectas del usuario o problemas al acceder a archivos.

Ejemplos de sintaxis:

1.- Python:

```
nombre = input("Ingrese su nombre: ")
```

2.- Java: Asegúrese de incluir la declaración import java.util.Scanner; al comienzo del programa.

```
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Ingrese su nombre: ");

        String nombre = scanner.nextLine();

        System.out.print("Ingrese su edad: ");

        int edad = scanner.nextInt();

    }

}
```

C++: incluir la declaración #include <iostream> al comienzo del programa para utilizar las funciones de entrada y salida de datos.

```
#include <iostream>
```

```
using namespace std;

int main() {

    string nombre;

    int edad;

    cout << "Ingrese su nombre: ";

    getline(cin, nombre);

    cout << "Ingrese su edad: ";

    cin >> edad;

    return 0;
}
```

5.1.2. Salida

La salida de datos es el último proceso, en el que se muestra información o resultados en un dispositivo o medio de comunicación. La forma en la que se realiza la salida varía dependiendo el lenguaje de programación que utilicemos, por ejemplo:

- En Python usamos la función "print()" para imprimir los resultados de la consola.

```
nombre = "Juan"
```

```
edad = 25

print("Nombre: ", nombre)

print("Edad: ", edad)
```

- En Java usamos la función "system.out.println()" para imprimir los resultados.

```
String nombre = "Juan";

int edad = 25;

System.out.println("Nombre: " + nombre);

System.out.println("Edad: " + edad);
```

- En C ++ usamos la función "std::cout" para imprimir los resultados.

```
#include <iostream>

using namespace std;

int main() {

    string nombre = "Juan";

    int edad = 25

    cout << "Nombre: " << nombre << endl;

    cout << "Edad: " << edad << endl;

    return 0;

}
```

- En JavaScript usamos la función "console.log()" para imprimir los resultados del navegador o del entorno Node.js.

```
let nombre = "Juan";  
  
let edad = 25;  
  
console.log("Nombre: ", nombre);  
  
console.log("Edad: ", edad);
```

La salida de los datos puede variar en complejidad dependiendo de qué se desea mostrar.

Aparte de imprimir valores directos, también se puede formatear la salida, imprimir en archivos, enviar datos a través de una red, etc.

Autoevaluación 17

1. ¿Qué es la entrada de datos en programación?

- a) El proceso de enviar información a una impresora.
- b) La acción de recibir información desde el usuario o algún dispositivo externo.
- c) El intercambio de datos entre variables dentro de un programa.
- d) La transmisión de datos a través de una red de computadoras.

2. ¿Cuál de las siguientes NO es una forma común de entrada de datos en programación?

- a) Teclado
- b) Ratón
- c) Pantalla
- d) Micrófono

3. ¿Qué función se utiliza en gran medida para recibir datos del usuario en lenguajes de programación como Python?

- a) output()
- b) input()
- c) read()
- d) get user_input()

4. ¿Cuál de las siguientes NO es una forma de salida de datos en programación?

- a) Impresión en consola
- b) Escritura en un archivo
- c) Envío de datos a través de una red
- d) Lectura de datos desde un sensor

5. ¿Qué representa el concepto de "stream" en la entrada/salida de datos en programación?

- a) Un flujo continuo de datos que se lee o escribe secuencialmente.
- b) Una estructura de datos en forma de árbol que organiza la entrada y salida.
- c) Una conexión de red para transmitir datos en tiempo real.
- d) Un tipo de variable utilizado para almacenar información de entrada y salida.

6.1 Estructura Para - For

Definición:

El bucle for es una estructura de control de flujo en programación que se utiliza para iterar sobre una secuencia de elementos. Puedes utilizarlo para recorrer elementos en una lista, tupla, cadena de caracteres, rango, u otros objetos iterables.T

Para qué sirve:

El bucle `for` se utiliza para realizar acciones repetitivas sobre cada elemento de una secuencia. Permite ejecutar un bloque de código un número específico de veces, donde cada ejecución está asociada a un elemento diferente de la secuencia.

Sintaxis:

La sintaxis general del bucle `for` en Python es la siguiente:

`for variable in iterable:`

 # Código a ejecutar en cada iteración

Variable: Es una variable que toma el valor de cada elemento en la secuencia durante cada iteración.

Iterable: Es la secuencia o colección sobre la cual se está iterando.

Ejemplo: Lista

```
frutas = ["manzana", "banana", "cereza"]
```

for fruta in frutas:

print(fruta)

Características del bucle `for`:

1. **Iteración sobre secuencias:** El bucle `for` se utiliza principalmente para iterar sobre secuencias de elementos, aunque también puede utilizarse con otros tipos de iterables.
2. **Finitud:** El bucle `for` se ejecuta hasta que la secuencia se agota. La cantidad de iteraciones está determinada por la longitud de la secuencia.
3. **Facilidad de uso:** Su sintaxis es clara y fácil de entender, lo que hace que sea una herramienta muy útil para tareas repetitivas y ciclos controlados.

4. **Compatible con múltiples tipos de datos:** Puede utilizarse con varios tipos de secuencias y objetos iterables, lo que lo hace versátil y flexible.
5. **Rango:** Es común utilizar la función `range()` con el bucle `for` para generar secuencias numéricas.

Finalidad del bucle for:

El bucle `for` se utiliza para automatizar tareas que requieren la repetición de un bloque de código para cada elemento en una secuencia. Esto puede incluir operaciones de procesamiento de datos, búsqueda, filtrado y muchas otras tareas que se benefician de la repetición estructurada.

Autoevaluación 19

1. ¿Cuál es la función principal del bucle for en programación según la definición proporcionada?

- a) Realizar operaciones aritméticas.
- b) Iterar sobre una secuencia de elementos.
- c) Ejecutar bloques de código de manera condicional.
- d) Definir estructuras de datos iterables.

2. ¿Para qué se utiliza el bucle for en Python, según su propósito principal?

- a) Realizar acciones únicas.
- b) Ejecutar un bloque de código de manera infinita.
- c) Automatizar acciones repetitivas sobre cada elemento de una secuencia.
- d) Definir estructuras de control de flujo.

3. ¿Cuál es la sintaxis general del bucle for en Python y qué representa la variable y el iterable en esta sintaxis?

- a) for item in lista: - "ítem" es la secuencia a iterar.
- b) for variable in secuencia: - "variable" toma el valor de cada elemento durante la iteración.
- c) for each elemento in rango: - elemento es la variable de control de iteración.
- d) for i from 1 to 10: - "i" es la variable de control y "1 to 10" es la secuencia.

4. ¿Cuáles son características del bucle for, según la información proporcionada?

- a) No es compatible con tipos de datos diferentes a listas.
- b) Ejecuta hasta que la secuencia sea infinita.
- c) Su sintaxis es compleja y difícil de entender.
- d) Se utiliza principalmente para iterar sobre secuencias, es finito, fácil de usar y compatible con varios tipos de datos.

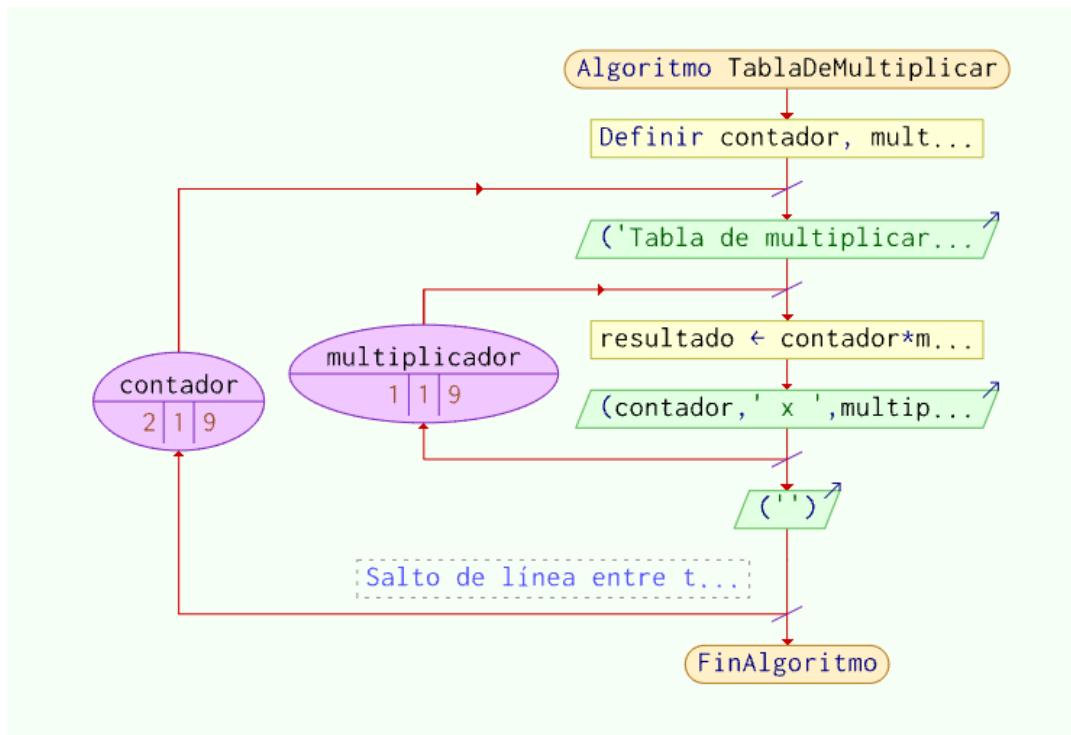
5. ¿Qué se destaca como una finalidad del bucle for, según la información proporcionada?

- a) Realizar tareas únicas en el código.
- b) Automatizar tareas que requieren la repetición de un bloque de código para cada elemento en una secuencia.
- c) Iterar sobre estructuras condicionales.
- d) Limitar su uso a la generación de secuencias numéricas.

Casos prácticos:

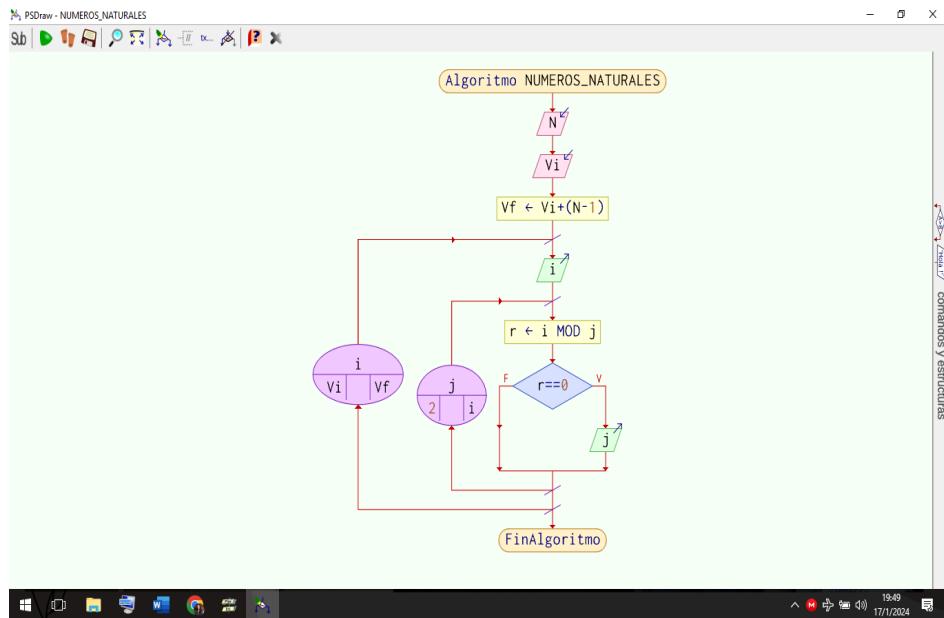
Ejemplo 1

- Hacer un programa que imprima una tabla de multiplicar del 2 al 9 mostrando sus valores multiplicados del 1 al 9 inclusive.



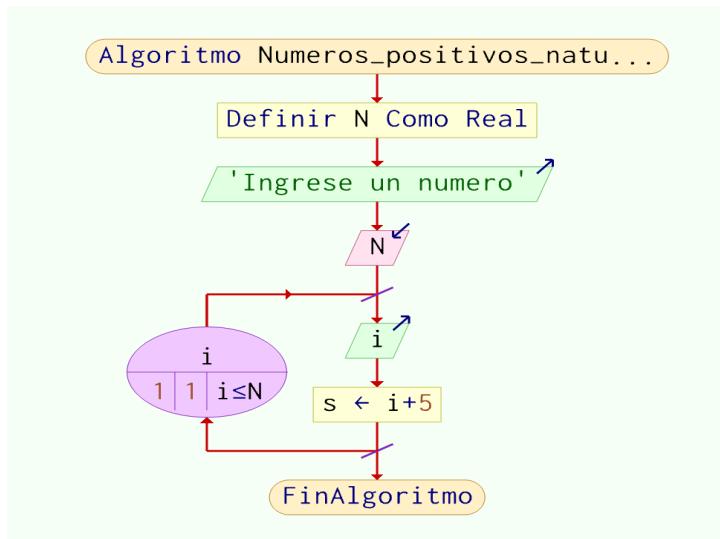
Ejemplo 2

- Generar los N primeros números naturales y a cada número natural descomponer en factores primos, considerando que los números naturales tendrán un valor inicial y un valor final.



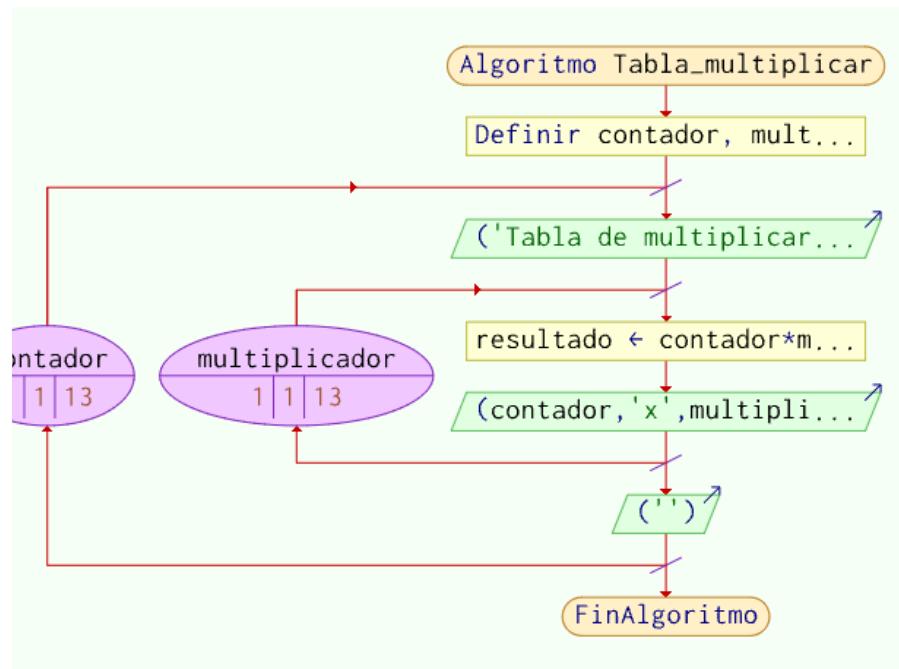
Ejemplo 3

- Imprima los n primeros números positivos naturales



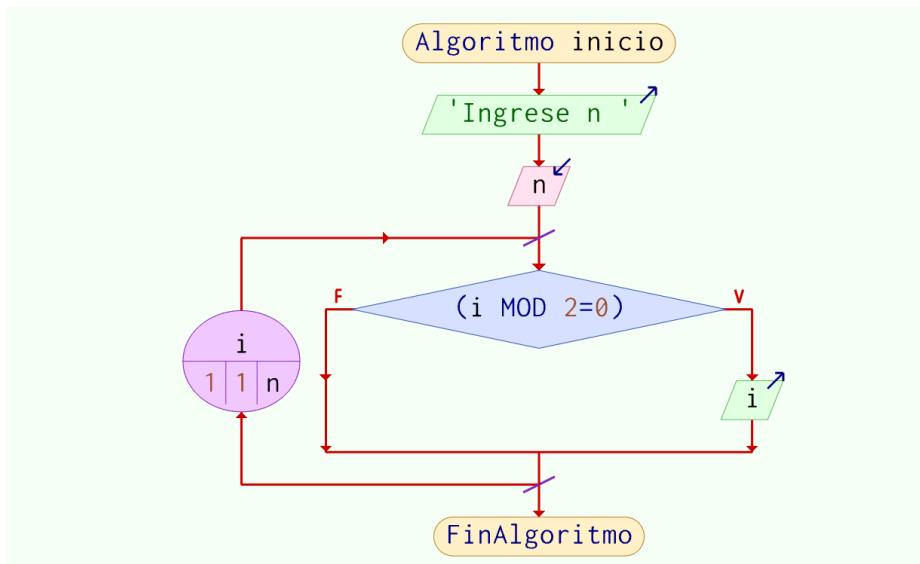
Ejemplo 4

- Imprimir la tabla de multiplicar de un número x, sabiendo que la tabla de multiplicar es del 1 al 12.



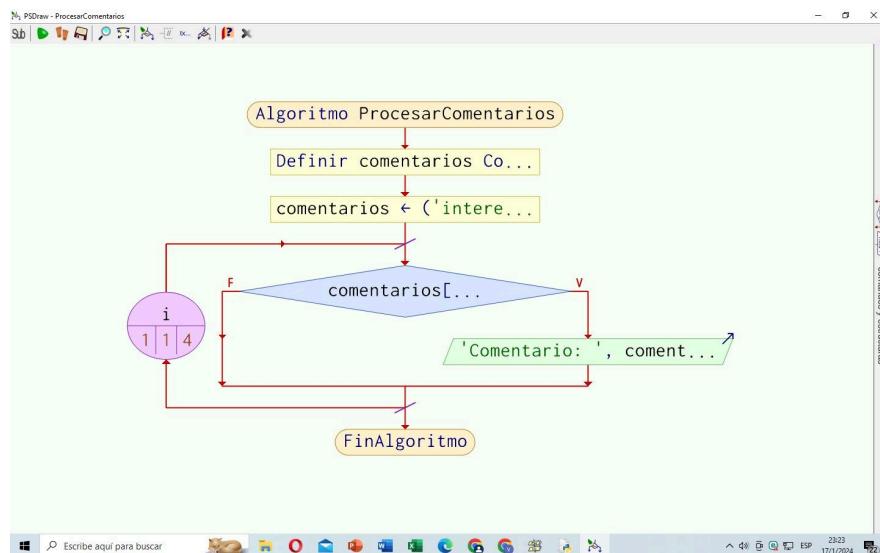
Ejemplo 5

- Se ingresa un número por teclado, realice un programa con la técnica de diagrama de flujo que permita obtener los “n” números pares desde 1 hasta el número ingresado en el teclado.



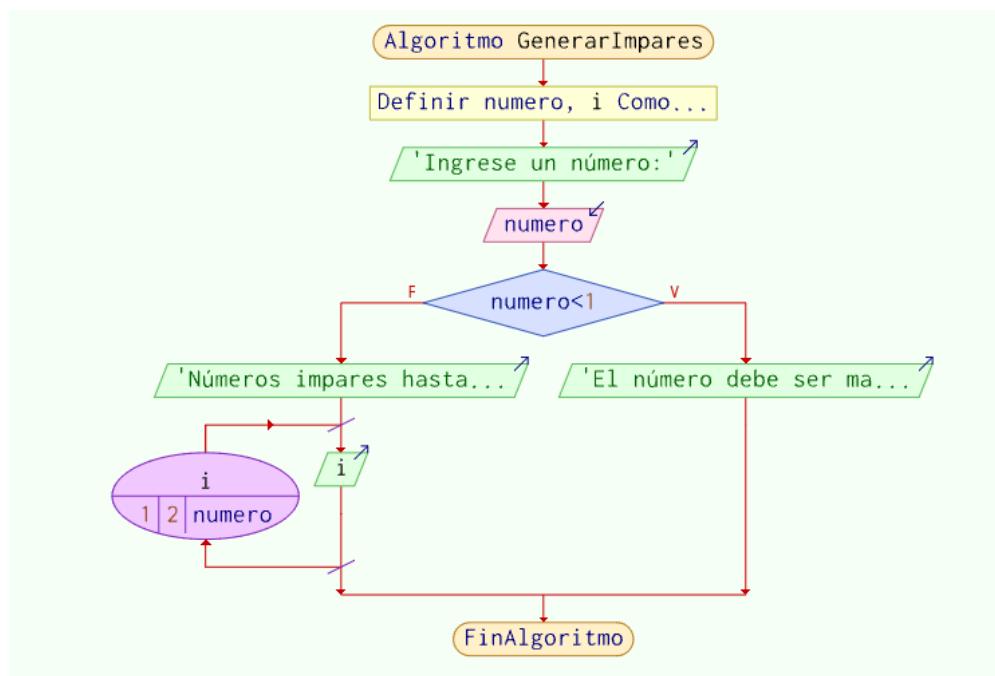
Ejemplo 6

- Cree una lista con elementos, luego cree un programa que imprima ciertos elementos de esa lista a consideración.



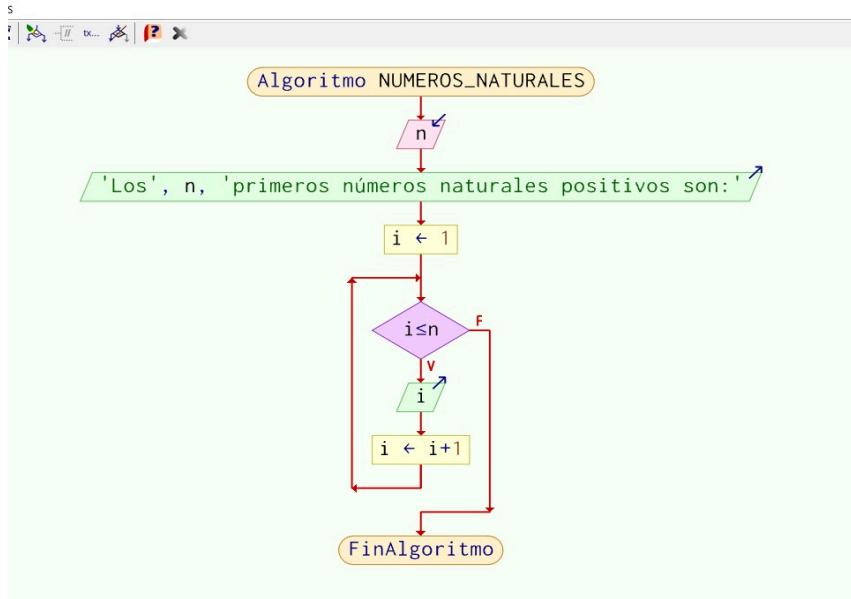
Ejemplo 7

- Se ingresa un número por teclado y genera los números impares hasta ese número:



Ejercicio 8

- Imprima los n primeros números positivos naturales



1.9 Estructura condicional

Es fundamental para tomar decisiones en la programación y controlar el flujo de ejecución del programa.

Tipos:

Simples: Existe un solo camino

Dobles: Tienen más de dos caminos.

Múltiples: Tienen una condición dentro de otra

Algunas características son:

1. La condición es una expresión que se evalúa como verdadera o falsa.
2. Se usan operadores relacionales.

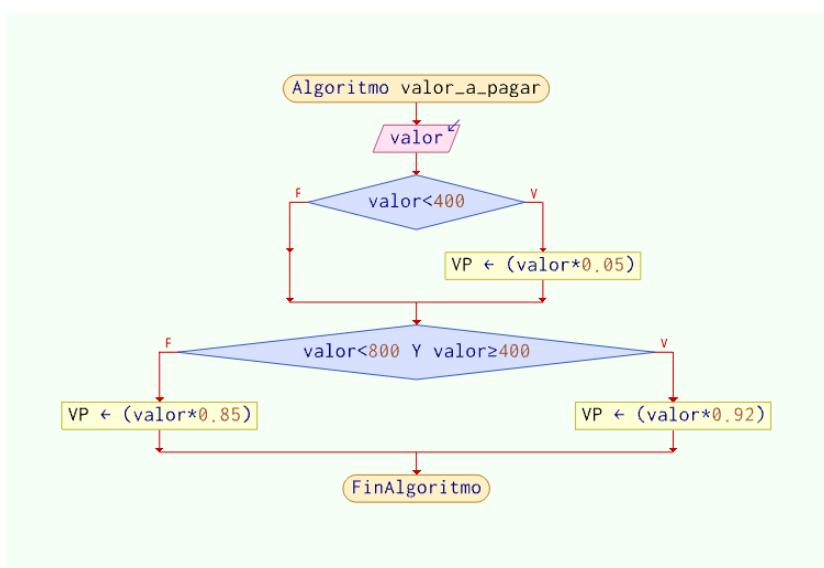
3. Tiene un bloque de instrucciones en el cual el código se ejecutará si se cumple la condición.

Sintaxis:

```
if condición 1:  
    # código a ejecutar si la condición 1 es verdadera  
  
elif condicion2:  
    # código a ejecutar si la condición 1 es falsa y la condición 2 es verdadera  
  
else:  
    #código a ejecutar si todas las condiciones son falsa
```

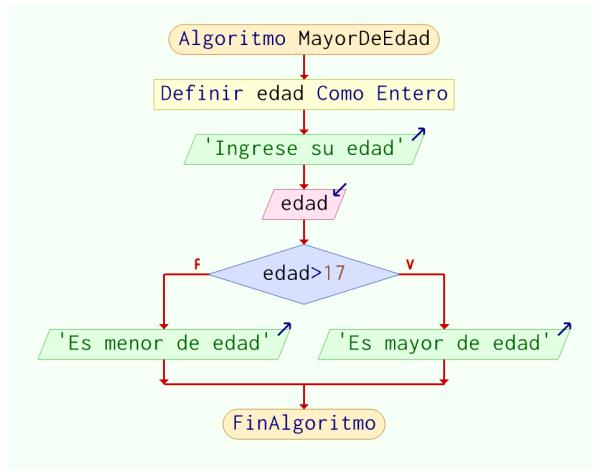
Ejemplo 1

- Ingresar por teclado el valor de un producto y luego determinar el valor a pagar.



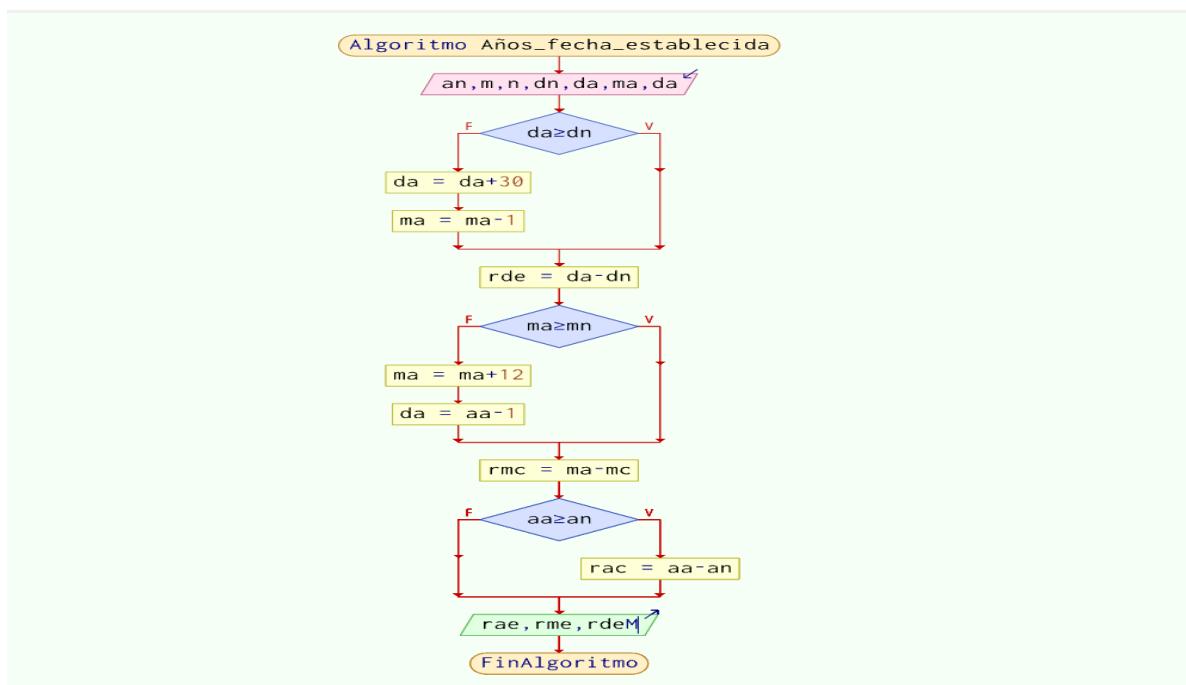
Ejemplo 2

- Se ingresa por teclado la edad de una persona, Determinar si es mayor de edad, sabiendo que una persona es mayor de edad a los 18 años.



Ejemplo 3

- Realice un diagrama de flujo para la determinación de los años a base de una fecha establecida



6.2 Estructura repetitivas Bucle - While

Definición:

Es una estructura que se repite mientras una condición específica sea verdadera ,se ejecuta repetidamente hasta que la condición sea falsa.

Características:

- Valor inicial: controlado por el contador.
- Condición: Indica cuantas veces se debe repetir.
- Incremento: Número de veces que aumenta el contador.
- Valor final: Fin del ciclo

Sintaxis:

while condición:

#Código a ejecutar mientras la condición sea verdadera.

Ejemplo:

contador=0

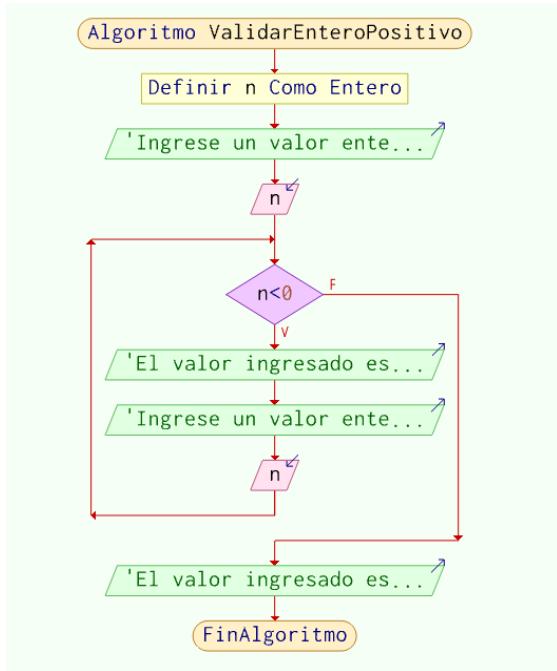
while contador<4:

 print (“contador”)

 contador +=1

Ejemplo 1

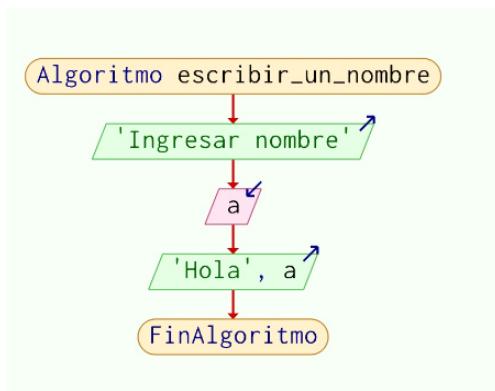
- Se ingresa por teclado un número, determinar si es menor a 0 e imprimir si no lo es, finalizar el programa.



ANEXOS

- 1) Hacer un programa donde se pida un nombre por teclado y se imprima "Hola ..el_nombre_ingresado"

Diagrama de flujo en PSeInt



Codificación de Python

```
nombre=str(input("Ingresa tu nombre \n"))
print ("Hola" , nombre)
```

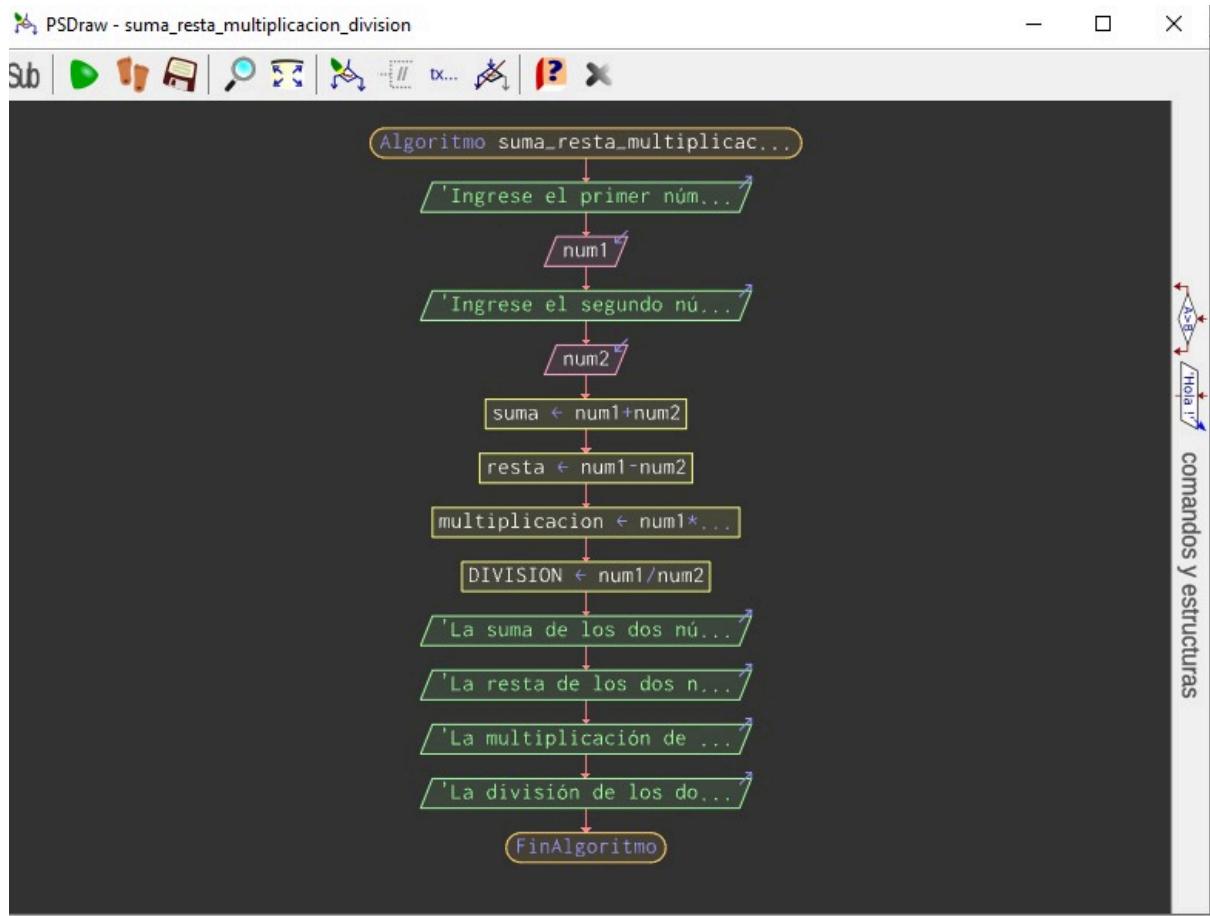
Ejecución del código

IDLE Shell 3.12.0

```
File Edit Shell Debug Options Window Help
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct 2 2023, 13:03:39) [MSC v.1935 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/Francisco/Desktop/JADE/python/nombre.py
Ingresa tu nombre
Francisco
Hola Francisco
```

2) Hacer un programa que solicite por teclado dos números y muestre la suma , la resta ,la multiplicación y la división de esos números.

Diagrama de flujo en PSeInt



Codificación en Python

```
# Programa en Python para realizar operaciones aritméticas  
# Solicitar al usuario que ingrese dos números  
num1 = float(input("Ingrese el primer número: "))  
num2 = float(input("Ingrese el segundo número: "))  
# Realizar las operaciones aritméticas  
suma = num1 + num2  
resta = num1 - num2  
multiplicacion = num1 * num2  
# Verificar si el segundo número no es cero antes de realizar la división  
if num2 != 0:  
    division = num1 / num2  
else:  
    division = "No se puede dividir por cero"
```

```

if num2 != 0:

    division = num1 / num2

    print("La división es:", division)

else:

    print("No se puede dividir entre cero.")

# Mostrar los resultados de las operaciones

print("La suma es:", suma)

print("La resta es:", resta)

print("La multiplicación es:", multiplicacion)

```

Ejecución del código

```

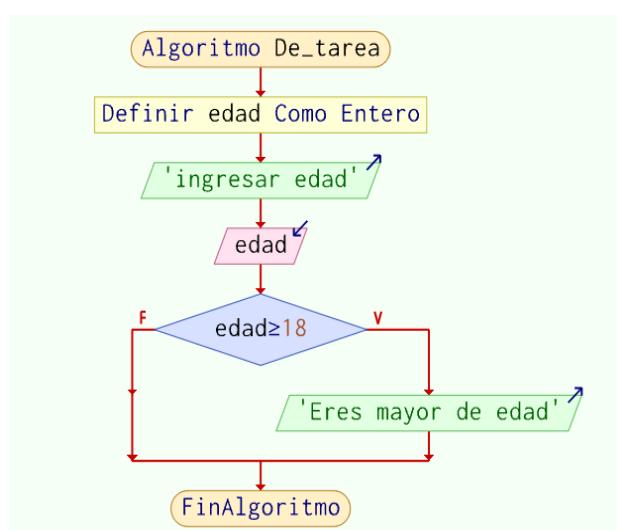
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct  2 2023, 13:03:39) [MSC v.1935 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.

= RESTART: C:/Users/IMAGEN-ACER/AppData/Local/Programs/Python/Python312/ghghgh.py
Ingrese el primer número: 8
Ingrese el segundo número: 8
La división es: 1.0
La suma es: 16.0
La resta es: 0.0
La multiplicación es: 64.0

```

3) Hacer un programa que solicite una edad y determine si es mayor de edad.

Diagrama de flujo en PSeInt



Codificación

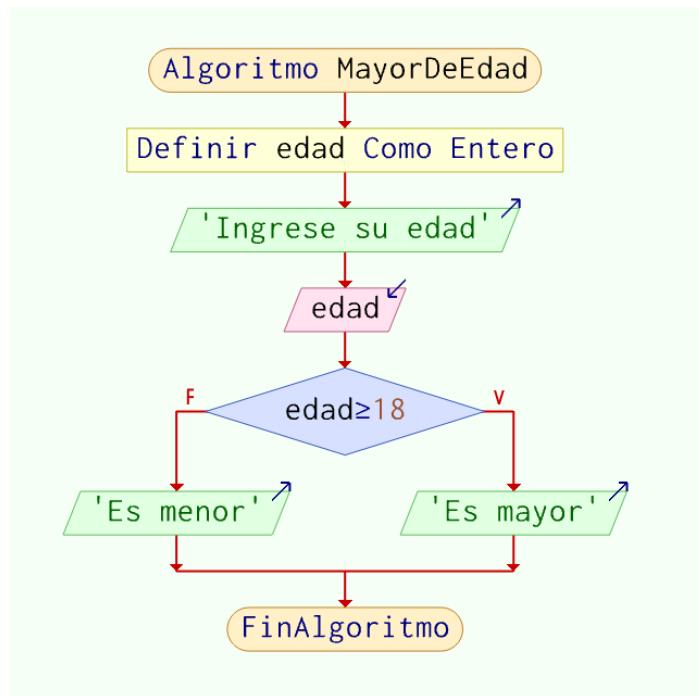
```
print("Por favor ingrese edad")  
edad= int(input())  
if edad>= 18:  
    print("Eres mayor de edad")
```

Ejecución del código

```
File Edit Shell Debug Options Window Help  
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct 2 2023, 13:03:39) [MSC v.1935 64 bit (AMD64)] on win32  
Type "help", "copyright", "credits" or "license()" for more information.  
>>>  
= RESTART: C:/Users/samue/Downloads/phytonn.Samuel/edad.py  
Por favor ingresar edad  
20  
Eres mayor de edad  
>>>
```

- 4) Hacer un programa que solicite una edad e imprima “Es mayor” si es mayor de edad , sino que imprima “Es menor”.

Diagrama de flujo en PSeInt



Codificación en Python

```
edad= int (input("Ingrese su edad"))  
if edad >= 18:
```

```

print ("Es mayor de edad")

else:

    print ("Es menor de edad")

```

Ejecución del código

```

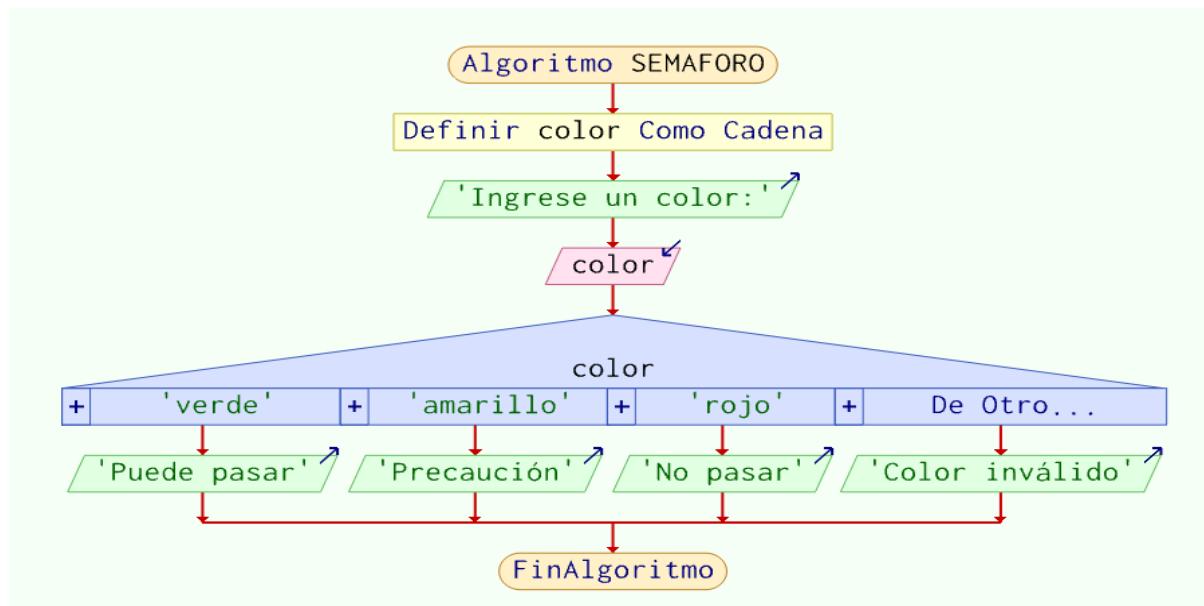
=====
RESTART: C:/Users/alemo/AppData/Local/Programs/Python/Python312/ejercicios python/ejercicio 4 para el manual.py =====
Ingrese su edad 18
Es mayor de edad

=====
RESTART: C:/Users/alemo/AppData/Local/Programs/Python/Python312/ejercicios python/ejercicio 4 para el manual.py =====
Ingrese su edad 7
Es menor de edad
|

```

- 5) Hacer un programa que solicite un color por teclado e imprima “Puede pasar” si el color ingresado es verde, “Precaución” si es amarillo, “No pasar” si es rojo o “Color invalido” si es cualquier otro.

Diagrama de Flujo en PSeInt



Codificación en python

```

import os

os.system("cls")

color = input("Ingrese un color:")

if color== "verde":

```

```

print("Puede pasar")

elif color== "amarillo":

    print("Precaución")

elif color== "rojo":

    print("No pasar")

else:

    print("Color inválido")

```

Ejecución de código

```

Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct  2 2023, 13:03:39) [MSC v.1935 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.

= RESTART: C:\Users\Ely\Documents\PARCIAL 2 PROGRAMACIÓN\ejercicio 5.py
Ingrese un color:amarillo
Precaución

===== RESTART: C:\Users\Ely\Documents\PARCIAL 2 PROGRAMACIÓN\ejercicio 5.py =====
Ingrese un color:verde
Puede pasar

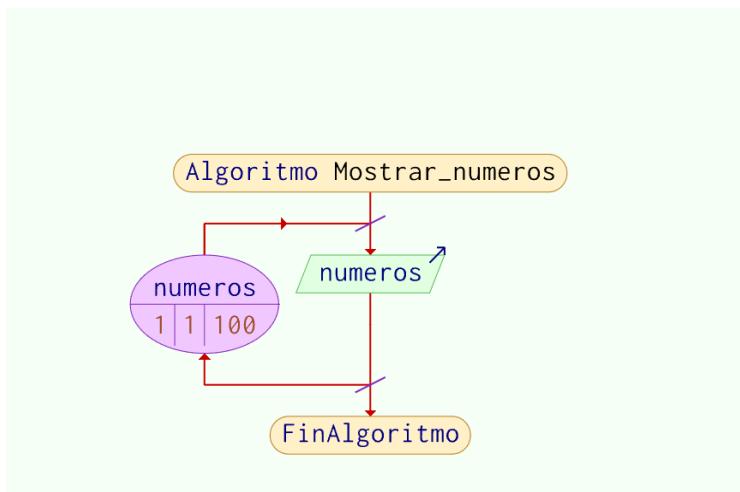
===== RESTART: C:\Users\Ely\Documents\PARCIAL 2 PROGRAMACIÓN\ejercicio 5.py =====
Ingrese un color:rojo
No pasar

===== RESTART: C:\Users\Ely\Documents\PARCIAL 2 PROGRAMACIÓN\ejercicio 5.py =====
Ingrese un color:rosado
Color invalido
|

```

6) Hacer un programa que cuente desde el 1 al 100 y los imprima uno a uno.

Diagrama de flujo en PSeInt



Codificación en Python

```
import os
```

```
for numeros in range(1,101,1):
    print(numeros)
```

Ejecución de código

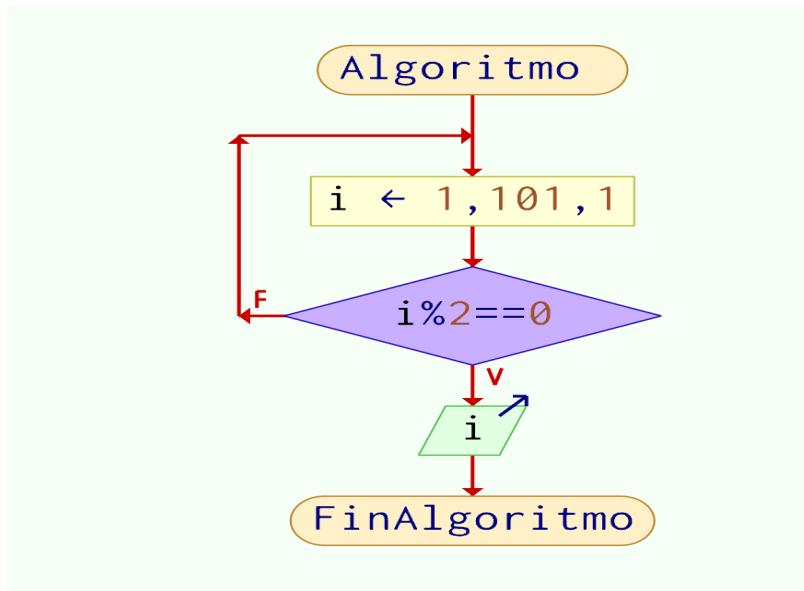
```
>>> = RESTART: C:\Users\Camila Ruiz\Desktop\Uni 9\Primer Semestre Biotecnología\Introducción a la Programación Parcial \Deber 1_30 Ejercicios\Ejercicio 6.py
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
```

```
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
```

```
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

7) Hacer un programa que cuente del 1 al 100 inclusive e imprima sólo los números pares

Diagrama de flujo en PSeInt



Código en Python

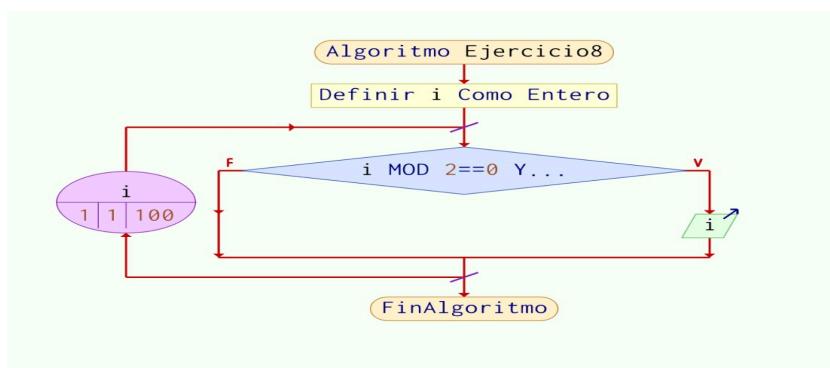
```
for i in range(1, 101):
    if i & 2 == 0:
        print(i)
```

Ejecución del código

```
>>> = RESTART: C:\Users\COMPANY02\Desktop\tareas\descargas documentos\Ejercicio_7_Pares1a100.py
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40
42
44
46
48
50
52
54
56
58
60
62
64
66
68
70
72
74
76
78
80
82
84
86
88
90
92
94
96
98
100
>>>
```

- 8) Hacer un programa que cuente del 1 al 100 inclusive e imprima los números que son divisibles por 2 y por 5.

Diagrama de flujo en PSeInt



Código en Python

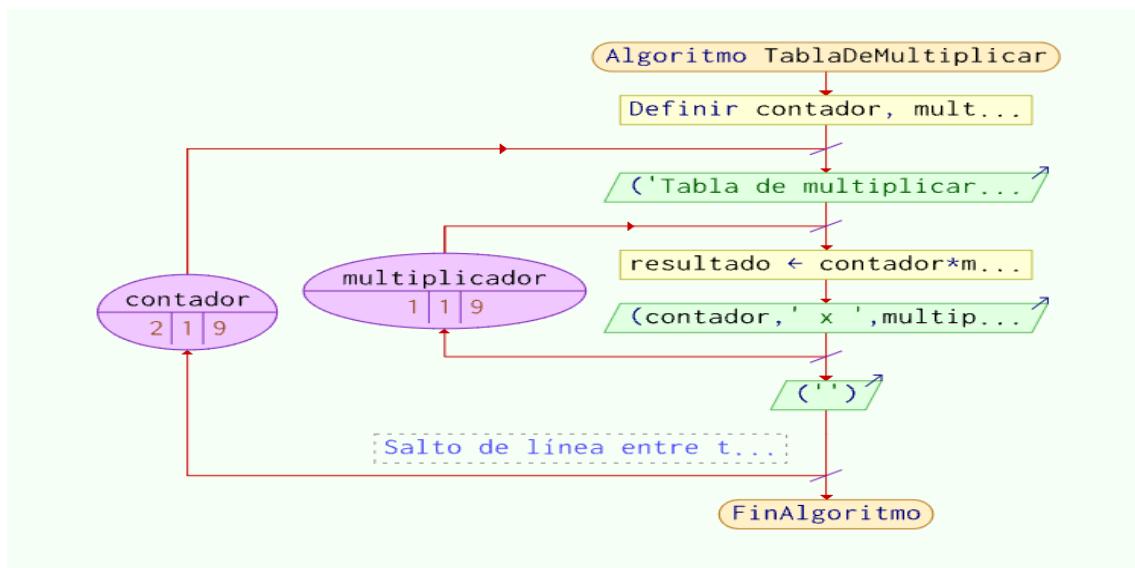
```
for i in range(1, 101):  
    if i % 2 == 0 and i % 5 == 0:  
        print(i)
```

Ejecución del código

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS  
PS C:\Users\Admin> & C:/Users/Admin/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/Admin/Downloads/Ejercicio Nº 8.py"  
10  
20  
30  
40  
50  
60  
70  
80  
90  
100  
PS C:\Users\Admin>
```

- 9) Hacer un programa que imprima una tabla de multiplicar del 2 al 9 mostrando sus valores multiplicados del 1 al 9 inclusive.

Diagrama de flujo en PSeInt



Código en Python

```
for i in range(2, 10):
```

```
print(f"\nTabla de multiplicar del {i}!\n")
```

```
for j in range(1, 10):
```

```
    resultado = i * j
```

```
    print(f"{i} x {j} = {resultado}")
```

Ejecución del código

Tabla de multiplicar del 2:	Tabla de multiplicar del 5:	Tabla de multiplicar del 8:
2 x 1 = 2	5 x 1 = 5	8 x 1 = 8
2 x 2 = 4	5 x 2 = 10	8 x 2 = 16
2 x 3 = 6	5 x 3 = 15	8 x 3 = 24
2 x 4 = 8	5 x 4 = 20	8 x 4 = 32
2 x 5 = 10	5 x 5 = 25	8 x 5 = 40
2 x 6 = 12	5 x 6 = 30	8 x 6 = 48
2 x 7 = 14	5 x 7 = 35	8 x 7 = 56
2 x 8 = 16	5 x 8 = 40	8 x 8 = 64
2 x 9 = 18	5 x 9 = 45	8 x 9 = 72
Tabla de multiplicar del 3:	Tabla de multiplicar del 6:	Tabla de multiplicar del 9:
3 x 1 = 3	6 x 1 = 6	9 x 1 = 9
3 x 2 = 6	6 x 2 = 12	9 x 2 = 18
3 x 3 = 9	6 x 3 = 18	9 x 3 = 27
3 x 4 = 12	6 x 4 = 24	9 x 4 = 36
3 x 5 = 15	6 x 5 = 30	9 x 5 = 45
3 x 6 = 18	6 x 6 = 36	9 x 6 = 54
3 x 7 = 21	6 x 7 = 42	9 x 7 = 63
3 x 8 = 24	6 x 8 = 48	9 x 8 = 72
3 x 9 = 27	6 x 9 = 54	9 x 9 = 81
Tabla de multiplicar del 4:	Tabla de multiplicar del 7:	
4 x 1 = 4	7 x 1 = 7	
4 x 2 = 8	7 x 2 = 14	
4 x 3 = 12	7 x 3 = 21	
4 x 4 = 16	7 x 4 = 28	
4 x 5 = 20	7 x 5 = 35	
4 x 6 = 24	7 x 6 = 42	
4 x 7 = 28	7 x 7 = 49	
4 x 8 = 32	7 x 8 = 56	
4 x 9 = 36	7 x 9 = 63	

10) Hacer un programa que muestre el siguiente dibujo.

```
*****
```

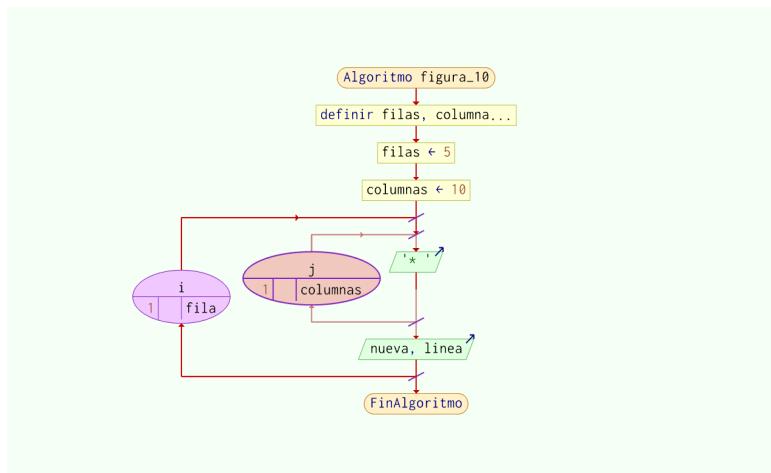
```
*****
```

```
*****
```

```
*****
```

```
*****
```

Diagrama de flujo en PSeInt



Código en Python

```
#10
fila=5
columna=10
i=int(input("ingrima fila"))
j=int(input("ingrima columna"))

for i in range(5):
    print("*" * 10, end="")
    print()
```

Ejecución del código

The screenshot shows the Python IDLE Shell interface. The code 'figura_10.py' is run, and the output displays a 5x10 grid of asterisks:

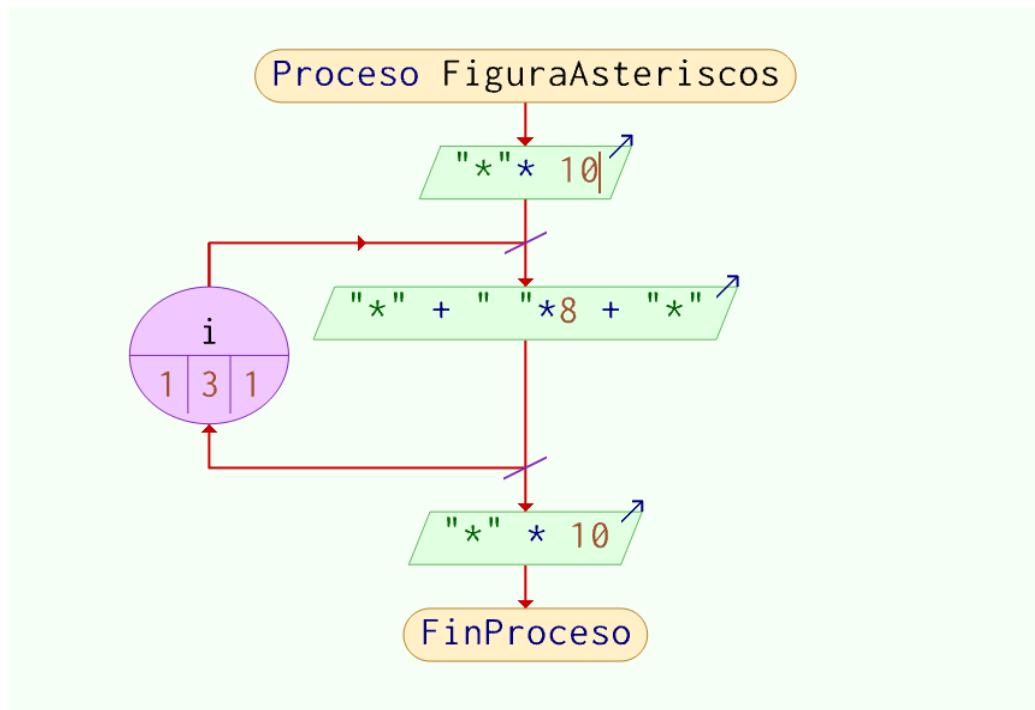
```
File Edit Shell Debug Options Window Help
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct 2 2023, 13:03:39) [MSC v.1935 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.

>>>
= RESTART: D:/Mis documentos/Downloads/figura_10.py
imprima fila5
imprima columnal0
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
```

11) Hacer un programa que muestre el siguiente dibujo

```
* * * * * * * * *
*           *
*           *
*           *
*           *
* * * * * * * * *
```

Diagrama de flujo en PSeInt



Código en Python

```
print('*' * 10)

for i in range(3):

    print('*' + ' ' * 8 + '*')

print('*' * 10)
```

Ejecución del código

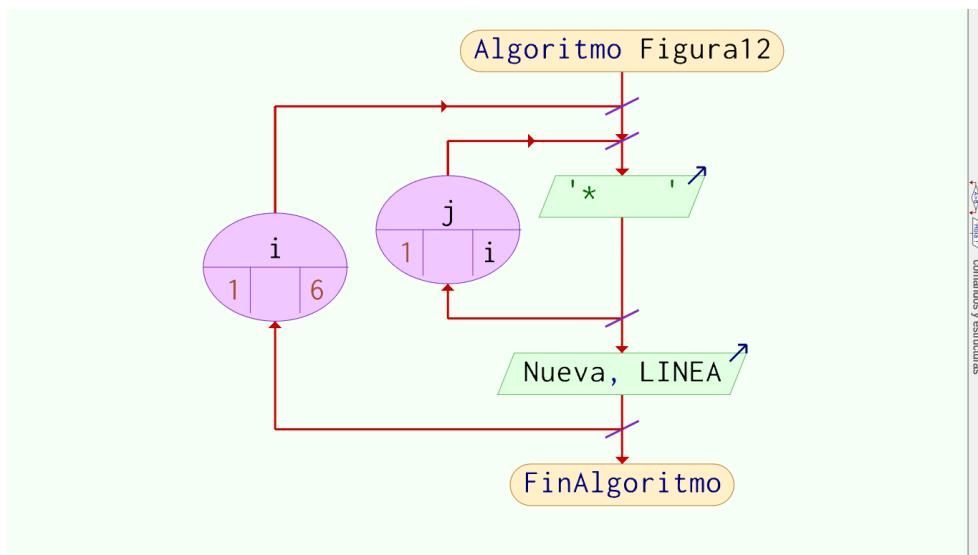
```
Python 3.11.7 (tags/v3.11.7:fa7a6f2, Dec  4 2023, 19:24:49) [MSC v.1937 64 bit  
(AMD64)] on win32  
Type "help", "copyright", "credits" or "license()" for more information.
```

```
= RESTART: C:/Users/usuario/OneDrive/Escritorio/Cuadrado.py  
*****  
*  
*  
*  
*  
*****
```

12) Hacer un programa que muestre el siguiente dibujo

```
*  
* *  
* * *  
* * * *  
* * * * *
```

Diagrama de flujo en PSeInt



Codificación en Python

```
#12  
  
for i in range(1, 6):  
  
    for j in range(i):  
  
        print("* ", end="")  
  
    print()
```

Ejecución del código

```

IDLE Shell 3.12.0
File Edit Shell Debug Options Window Help
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct 2 2023, 13:03:39) [MSC v.1935 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\Usuario\Documents\ESPE\PRIMER SEMESTRE\Programación\DEBERES\30 Ejercicios\Ejercicio 12.py
*
*
**
**
*
*
*

```

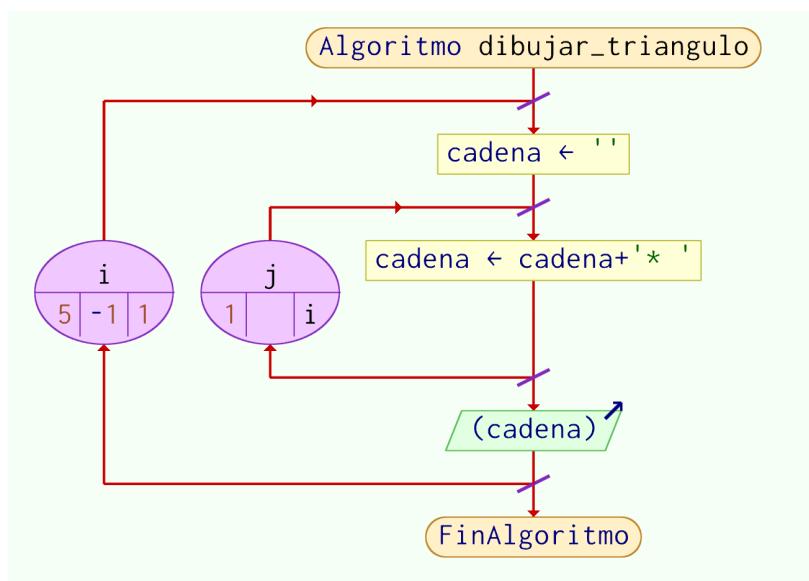
13) Idem anterior con este dibujo

```

*****
*****
*****
**
*

```

Diagrama de flujo en PSeInt



Codificación en Python

```

for i in range(5, 0, -1):
    print("* " * i)

```

Ejecución del código

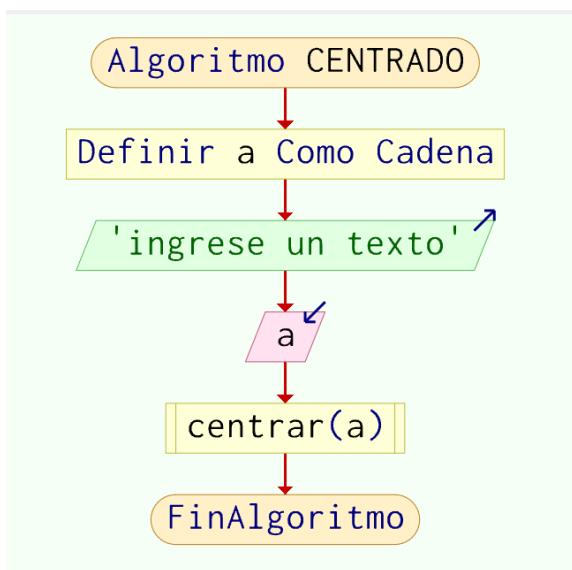
```

type "help", "copyright", "credits" or "license()" for more information.
>>> ===== RESTART: C:\Users\USER\AppData\Local\Programs\Python\Python312\13.py =====
* * * *
* * * *
* * *
* *
* 
>>> |

```

- 14) Crea un función Escribir Centrado, que reciba como parámetro un texto y lo escriba centrado en pantalla (suponiendo una anchura de 80 columnas;
 pista: deberás escribir $40 - \text{longitud}/2$ espacios antes del texto). Además subraya el mensaje utilizando el carácter =.

Diagrama de Flujo en PSeInt



Codificación de Python

```

import os

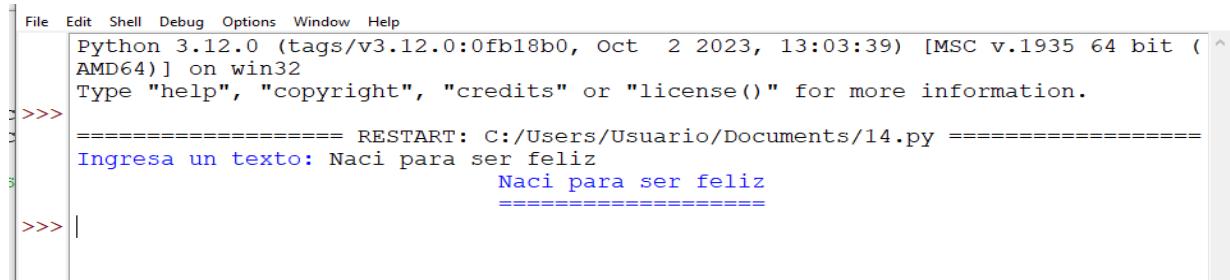
def centrar(tx):
    espacios = (80 - len(tx)) // 2
    print(' ' * espacios + tx)
    print(' ' * espacios + '*' * len(tx))

```

```
texto = input("Ingresa un texto: ")
```

```
centrar(texto)
```

Ejecución del programa



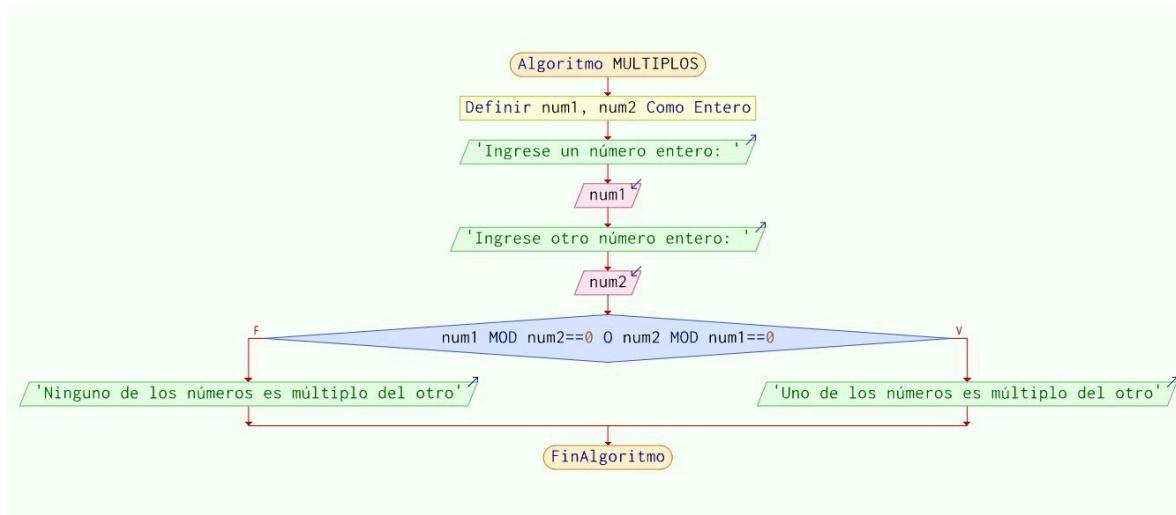
```
File Edit Shell Debug Options Window Help
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct 2 2023, 13:03:39) [MSC v.1935 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.

>>> ===== RESTART: C:/Users/Usuario/Documents/14.py =====
Ingresá un texto: Naci para ser feliz
Naci para ser feliz
=====

>>>
```

- 15) Crea un programa que pida dos números enteros al usuario y diga si alguno de ellos es múltiplo del otro. Crea una función es múltiplo que reciba los dos números, y devuelve si el primero es múltiplo del segundo.

Diagrama de flujo en PSeInt



Codificación en Python

```
import os

num1=int(input("Ingrese un número entero: "))

num2=int(input("Ingrese otro número entero: "))

if num1%num2==0:

    print("El primer número es múltiplo del segundo")
```

```

elif num2%num1==0:
    print("El segundo número es múltiplo del primero")
else:
    print("Ninguno es múltiplo")

```

Ejecución del Código

```

IDLE Shell 3.12.0
File Edit Shell Debug Options Window Help
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct 2 2023, 13:03:39) [MSC v.1935 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.

>>> ===== RESTART: C:\Users\MAQUINA 3\Documents\ESPE\ejercicio 15.py =====
Ingrese un número entero: 4
Ingrese otro número entero: 8
El segundo número es múltiplo del primero

>>> ===== RESTART: C:\Users\MAQUINA 3\Documents\ESPE\ejercicio 15.py =====
=====
Ingrese un número entero: 10
Ingrese otro número entero: 2
El primer número es múltiplo del segundo

>>> ===== RESTART: C:\Users\MAQUINA 3\Documents\ESPE\ejercicio 15.py =====
=====
Ingrese un número entero: 15
Ingrese otro número entero: 7
Ninguno es múltiplo

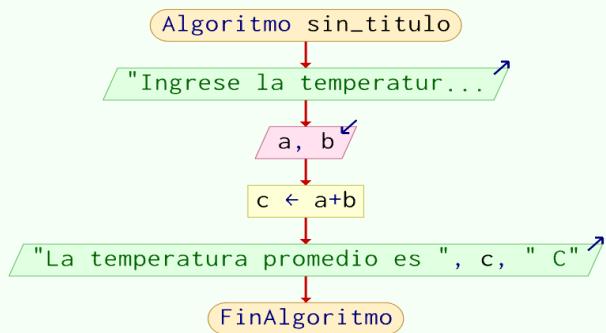
```

16)Crear una función que calcule la temperatura media de un día a partir de la temperatura máxima y mínima.

Crear un programa principal, que utilizando la función anterior, vaya pidiendo la temperatura máxima y mínima de cada día y vaya mostrando la media.

El programa pedirá el número de días que se van a introducir.

Diagrama flujo



Codificación en Python

```
print ("Ingrese la temperatura máxima y mínima")
```

```
a = float (input ())
```

```
b = float (input ())
```

```
c = (a+b)/2
```

```
print ("La temperatura promedio es: ", c, "°C")
```

Ejecución del código

Ingresar la temperatura máxima y mínima

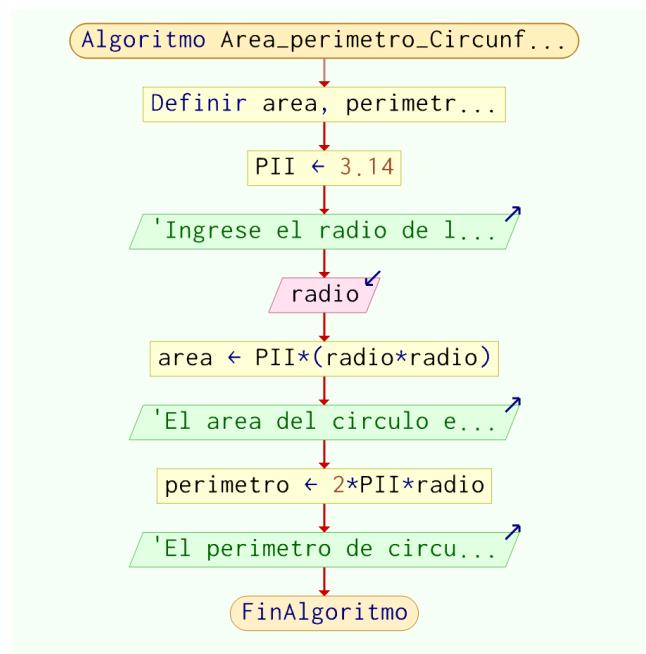
45

33

La temperatura promedia es 39.0 °C

- 19)** Diseñar una función que calcule el área y el perímetro de una circunferencia. Utiliza dicha función en un programa principal que lea el radio de una circunferencia y muestre su área y perímetro

Diagrama de Flujo



Codificación en Python

```
import os

r = int(input("Ingrese el radio"))

pi = 3.14

A = pi * r**2

print("El área es: ", A)

P = 2*pi*r

print("El perímetro de círculo es: ", P)
```

Ejecución del programa

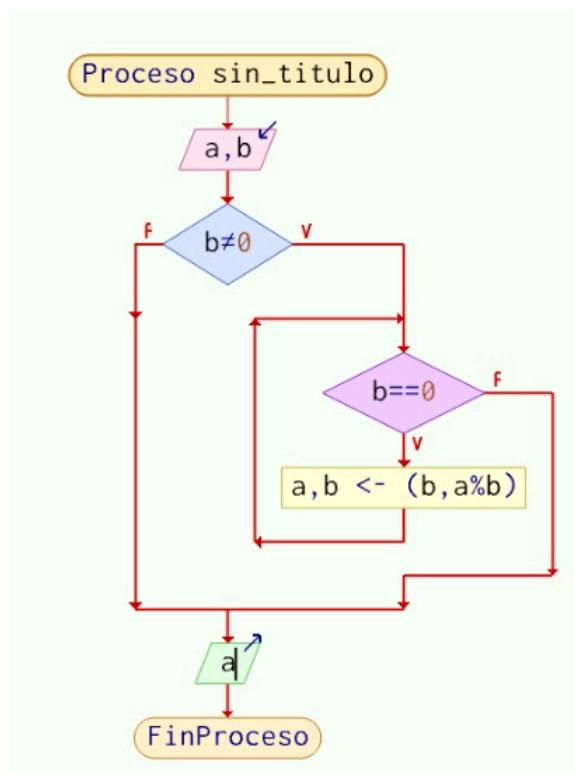
```
IDLE Shell 3.12.0
File Edit Shell Debug Options Window Help
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct 2 2023, 13:03:39) [MSC v.1935 64 bit
AMD64] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/Asus/Desktop/radio.py
Ingrese el radio de la circunferencia: 4
El área de la circunferencia es: 50.24
El perímetro de la circunferencia es: 25.12
>>>
```

22) Crear una función que calcule el MCD de dos números por el método de Euclides. El método de Euclides es el siguiente:

- Se divide el número mayor entre el menor.
- Si la división es exacta, el divisor es el MCD.
- Si la división no es exacta, dividimos el divisor entre el resto obtenido y se continúa de esta forma hasta obtener una división exacta, siendo el último divisor el MCD.

Crea un programa principal que lea dos números enteros y muestre el MCD.

Diagrama de flujo



Codificación

```
def mcd(a, b):  
    while b != 0:  
        a, b = b, a % b  
    return a  
  
def main():  
    num1 = int(input("Introduce el primer número: "))
```

```

num2 = int(input("Introduce el segundo número: "))

print("El MCD de", num1, "y", num2, "es", mcd(num1, num2))

```

main()

Ejecución del programa

```

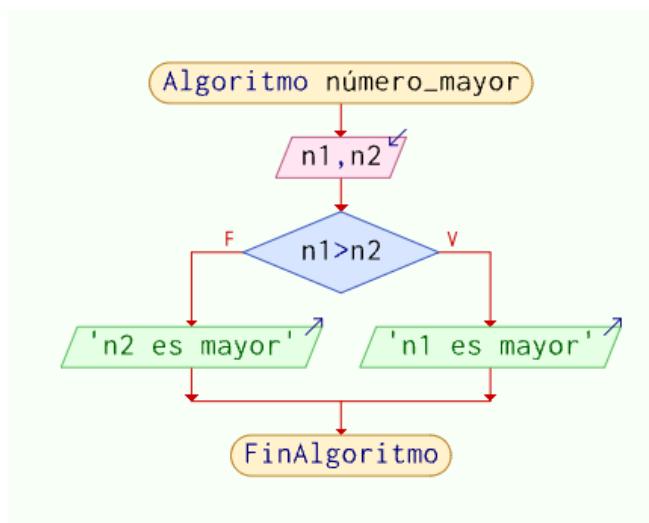
IDLE Shell 3.12.0
File Edit Shell Debug Options Window Help
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct 2 2023, 13:03:39) [MSC v.1935 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.

>>> ===== RESTART: C:/Users/PAME ZURITA/Downloads/trabajo programacion.py =====
Introduce el primer número: 1
Introduce el segundo número: 5
El MCD de 1 y 5 es 1
>>>

```

27) Definir una función max() que tome como argumento dos números y devuelva el mayor de ellos.

Diagrama de Flujo



Codificación en Python

```

def encontrar_maximo(numero1, numero2):

    if numero1 > numero2:
        return numero1

    else:
        return numero2

```

```

numero1 = float(input("Ingrese el primer número: "))

numero2 = float(input("Ingrese el segundo número: "))

resultado = encontrar_maximo(numero1, numero2)

print(f"El número máximo es: {resultado}")

```

Ejecución del programa

The screenshot shows the Python IDLE Shell interface. The title bar says "IDLE Shell 3.12.0". The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The main window displays the Python interpreter's welcome message and the execution of a script named "27.py". The script prompts for two numbers, receives inputs of 8 and 9, and prints the maximum value, which is 9.0.

```

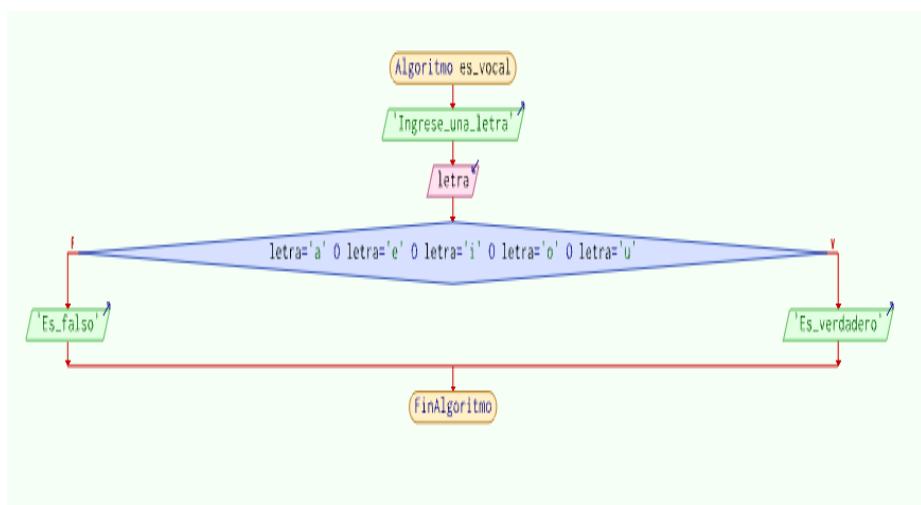
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct 2 2023, 13:03:39) [MSC v.1935 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.

>>> ===== RESTART: C:\Users\IMG_NUC\Downloads\27.py =====
Ingrese el primer número: 8
Ingrese el segundo número: 9
El número máximo es: 9.0
>>>

```

- 30) Escribir una función que tome un carácter y devuelva verdadero si es una vocal, de lo contrario devuelve falso.

Diagrama:



Codificación en Python

```
n=input("Escriba una letra: ")
```

```
if n in ("a","e","i","o","u"):
```

```
    print("True")
```

```
else:
```

```
    print("False")
```

Ejecución del Código

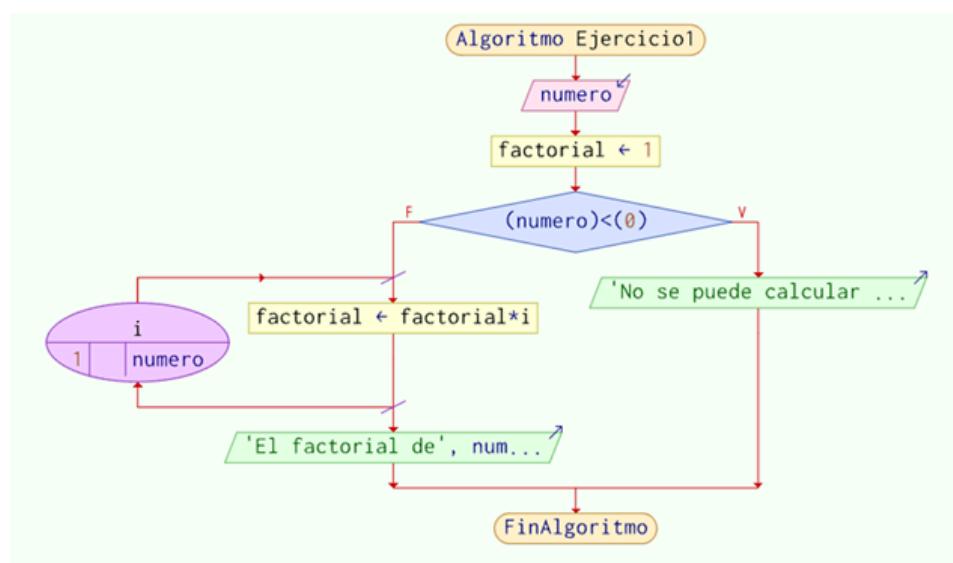
```
= RESTART: C:/Users/hp/Desktop/Semestre 1/Pro...
Escriba una letra: a
True
=====
RESTART: C:/Users/hp/Desktop/Semestre ...
Escriba una letra: c
False
```

EJERCICIOS PROPUESTOS 2

Ejercicio 1

Crea una aplicación que pida un número y calcule su factorial (La factorial de un número es el producto de todos los enteros entre 1 y el propio número y se representa por el número seguido de un signo de exclamación. ¡Por ejemplo $5! = 1 \times 2 \times 3 \times 4 \times 5 = 120$)

Diagrama de Flujo



Codificación en Python

```
numero = int(input("Ingrese un número para calcular su factorial: "))

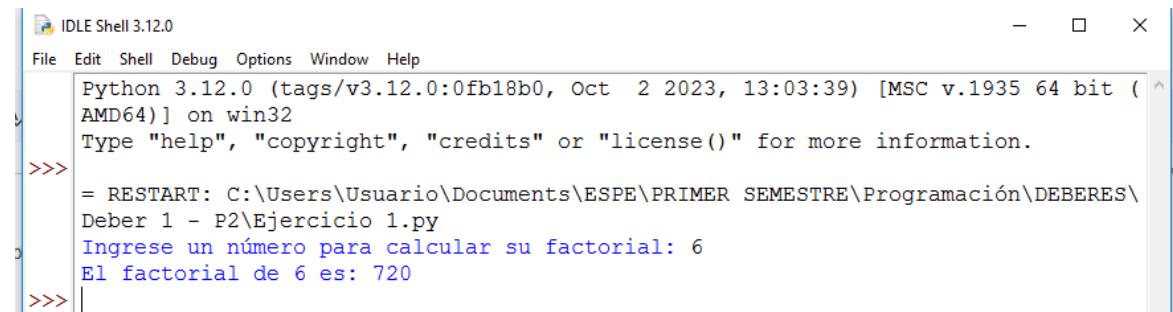
factorial = 1

if numero < 0:
    print("No se puede calcular el factorial de un número negativo.")

else:
    for i in range(1, numero + 1):
        factorial *= i

    print(f"El factorial de {numero} es: {factorial}")
```

Ejecución



The screenshot shows the IDLE Shell 3.12.0 interface. The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The main window displays the Python interpreter's welcome message and a session transcript:

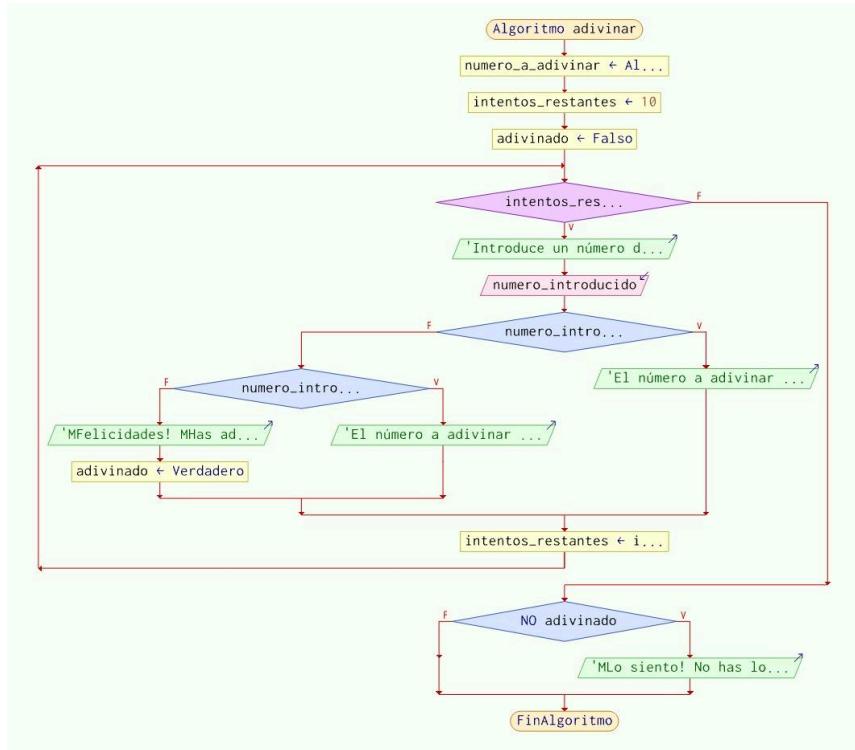
```
IDLE Shell 3.12.0
File Edit Shell Debug Options Window Help
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct  2 2023, 13:03:39) [MSC v.1935 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.

>>> = RESTART: C:\Users\Usuario\Documents\ESPE\PRIMER SEMESTRE\Programación\DEBERES\
Deber 1 - P2\Ejercicio 1.py
Ingrese un número para calcular su factorial: 6
El factorial de 6 es: 720
>>>
```

Ejercicio 2

Crea una aplicación que permita adivinar un número. La aplicación genera un número aleatorio del 1 al 100. A continuación, va pidiendo números y va respondiendo si el número a adivinar es mayor o menor que el introducido, además de los intentos que te quedan (tienes 10 intentos para acertarlo). El programa termina cuando se acierta el número (además te dice en cuantos intentos lo has acertado), si se llega al límite de intentos te muestra el número que había generado.

Diagrama de flujo.



Codificación

```

import os
import random

numero_a_adivinar = random.randint(1, 100)
intentos = 10

print("¡Bienvenido a Adivina el Número!")
print("Tienes 10 intentos para adivinar un número del 1 al 100.")

while intentos > 0:
    numero = int(input("Ingresa un número: "))

    if numero == numero_a_adivinar:
        print("¡Felicitaciones! ¡Has adivinado el número en", 11 - intentos, "intentos!")
        break

    if numero < numero_a_adivinar:

```

```

print("El número a adivinar es mayor. ¡Inténtalo de nuevo!")

if numero > numero_a_adivinar:
    print("El número a adivinar es menor. ¡Inténtalo de nuevo!")

intentos -= 1

print("Te quedan", intentos, "intentos.")

if intentos == 0:
    print("¡Lo siento! Has agotado tus 10 intentos.")
    print("El número que debías adivinar era:", numero_a_adivinar)

```

Ejecución

```

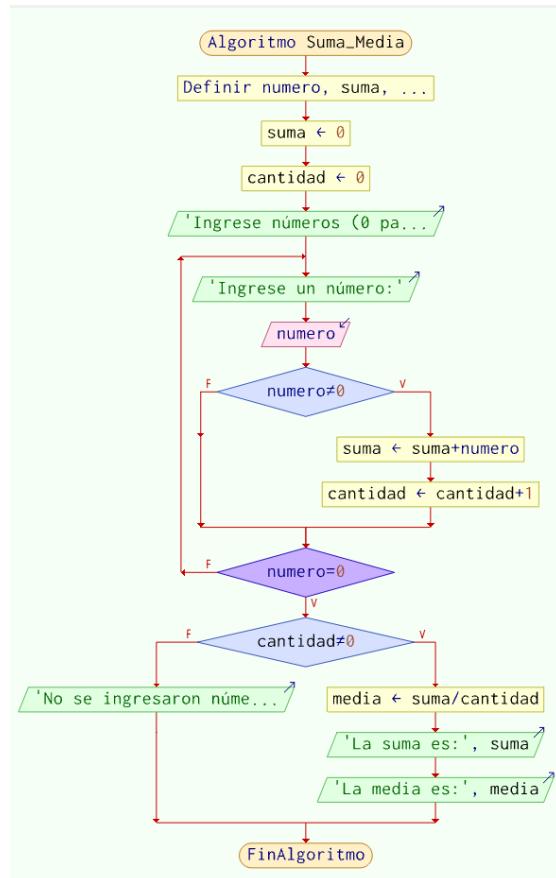
¡Bienvenido a Adivina el Número!
Tienes 10 intentos para adivinar un número del 1 al 100
.
Ingrresa un número: 5
El número a adivinar es mayor. ¡Inténtalo de nuevo!
Te quedan 9 intentos.
Ingrresa un número: 10
El número a adivinar es mayor. ¡Inténtalo de nuevo!
Te quedan 8 intentos.
Ingrresa un número: 50
El número a adivinar es mayor. ¡Inténtalo de nuevo!
Te quedan 7 intentos.
Ingrresa un número: 100
El número a adivinar es menor. ¡Inténtalo de nuevo!
Te quedan 6 intentos.
Ingrresa un número: 90
El número a adivinar es menor. ¡Inténtalo de nuevo!
Te quedan 5 intentos.
Ingrresa un número: 80
El número a adivinar es mayor. ¡Inténtalo de nuevo!
Te quedan 4 intentos.
Ingrresa un número: 85
El número a adivinar es menor. ¡Inténtalo de nuevo!
Te quedan 3 intentos.
Ingrresa un número: 82
El número a adivinar es mayor. ¡Inténtalo de nuevo!
Te quedan 2 intentos.
Ingrresa un número: 83
El número a adivinar es mayor. ¡Inténtalo de nuevo!
Te quedan 1 intentos.
Ingrresa un número: 84
¡Felicitaciones! ¡Has adivinado el número en 10 intentos!

```

Ejercicio 3

Algoritmo que pida números hasta que se introduzca un cero. Debe imprimir la suma y la media de todos los números introducidos.

Diagrama de flujo



Codificación en Python

```
suma = 0
```

```
cantidad = 0
```

```
print("Ingrese números (0 para terminar):")
```

```
while True:
```

```
    numero = float(input("Ingrese un número: "))
```

```
    if numero != 0:
```

```
suma += numero

cantidad += 1

else:

break

if cantidad != 0:

media = suma / cantidad

print("La suma es:", suma)

print("La media es:", media)

else:

print("No se ingresaron números válidos.")
```

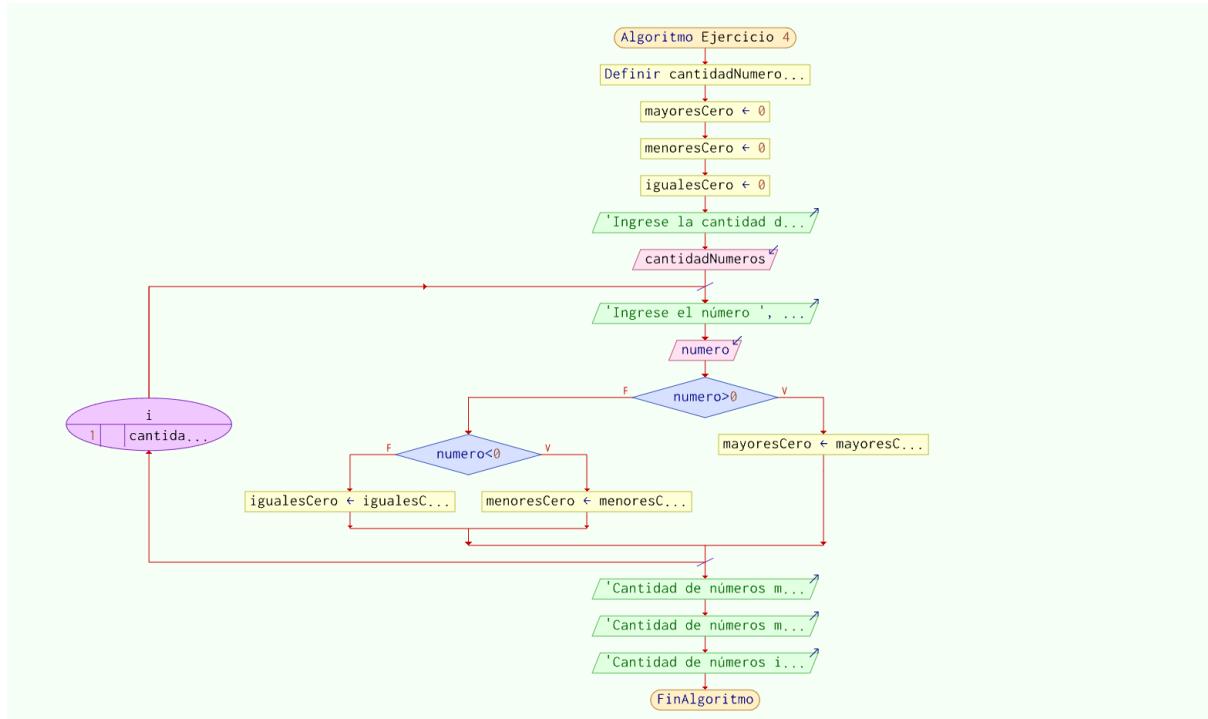
Ejecución

```
===== RESTART: C:\Users\Asus\Desktop\suam.py =====
Ingrese números (0 para terminar):
Ingrese un número: 7
Ingrese un número: 8
Ingrese un número: 9
Ingrese un número: 4
Ingrese un número: 0
La suma es: 28.0
La media es: 7.0
```

Ejercicio 4

Realizar un algoritmo que pida números (se pedirá por teclado la cantidad de números a introducir). El programa debe informar de cuantos números introducidos son mayores que 0, menores que 0 e iguales a 0.

Diagrama



Codificación en python:

```
mayores_cero = 0
```

```
menores_cero = 0
```

```
iguales_cero = 0
```

```
cantidad_numeros = int(input("Ingrese la cantidad de números a introducir: "))
```

```
for i in range(cantidad_numeros):
```

```
    numero = float(input(f"Ingrese el número {i+1}: "))
```

```
if numero > 0:
```

```
    mayores_cero += 1
```

```
elif numero < 0:
```

```
    menores_cero += 1
```

```
else:
```

```
iguales_cero += 1
```

```
print("Cantidad de números mayores que 0:", mayores_cero)
print("Cantidad de números menores que 0:", menores_cero)
print("Cantidad de números iguales a 0:", iguales_cero)
```

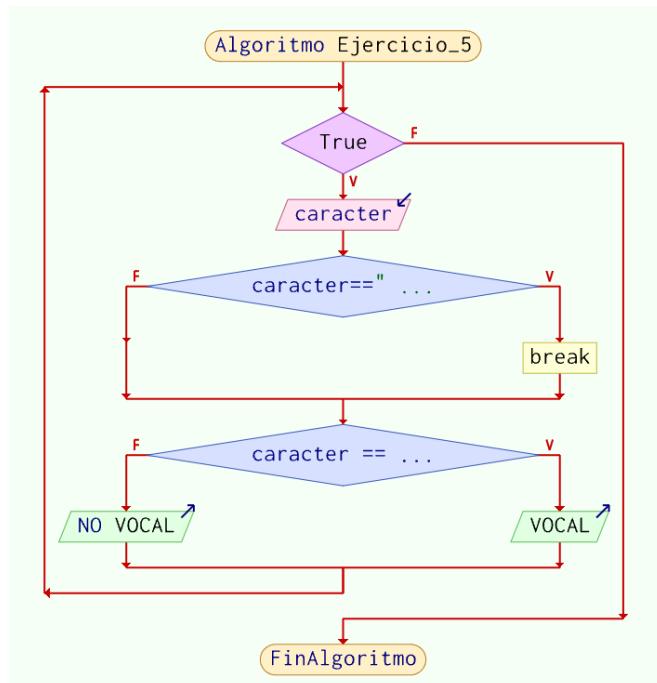
Ejecución:

```
File Edit Shell Debug Options Window Help
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct 2 2023, 13:03:39) [MSC v.1935 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\Camila Ruiz\Desktop\Uni 🌐\Primer Semestre Biotecnología\Introducción a la Programación 🎓🔗🔗🔗\Parcial 2\Deber lasos repetitivos\Ejercicios 4.py
¿Cuántos números vas a introducir?: 4
Número 1: 5
Número 2: 4
Número 3: 3
Número 4: 5
Números positivos: 4
Números negativos: 0
Números igual a 0: 0
```

Ejercicio 5

Algoritmo que pida caracteres e imprima 'VOCAL' si son vocales y 'NO VOCAL' en caso contrario, el programa termina cuando se introduce un espacio.

Diagrama



Codificación en python:

```
import os

while True:

    caracter = input("Ingresa un carácter (presiona espacio para terminar): ")

    if caracter == ' ':

        print("Programa terminado.")

        break

    if caracter.lower() in ['a', 'e', 'i', 'o', 'u','A', 'E', 'I', 'O', 'U']:

        print("VOCAL")

    else:

        print("NO VOCAL")
```

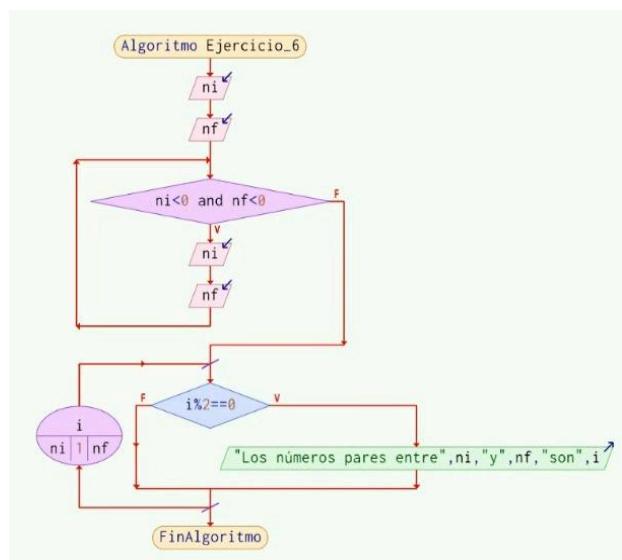
Ejecución:

```
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct 2 2023, 13:03:39) [MSC v.1935 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.

= RESTART: C:\Users\Ely\Downloads\ejercicio 5 manual.py
Ingrese un carácter (presione enter para salir): E
VOCAL
Ingrese un carácter (presione enter para salir): A
VOCAL
Ingrese un carácter (presione enter para salir): M
NO VOCAL
Ingrese un carácter (presione enter para salir): H
NO VOCAL
Ingrese un carácter (presione enter para salir):
```

Escribir un programa que imprima todos los números pares entre dos números que se le pida al usuario

Codificación



Codificación en python:

```
ni=int(input("ingrese un numero entero positivo"))

nf=int(input("ingrese un numero entero positivo"))

while ni<0 and nf<0:

    ni=int(input("ingrese un numero entero positivo"))

    nf=int(input("ingrese un numero entero positivo"))

for i in range (ni,nf+1,1):

    if i%2==0:

        print("los numeros pares entre",ni,"y",nf,"son",i)
```

Ejecución:

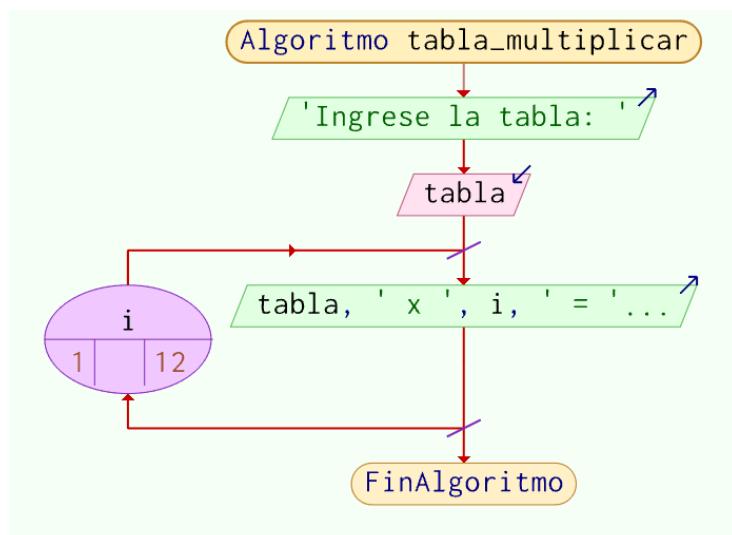
```
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct 2 2023, 13:03:39) [MSC v.1935 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.

= RESTART: C:\Users\Ely\Downloads\ejercicio 6 manual (1).py
ingrese un numero entero positivo4
ingrese un numero entero positivo24
los numeros pares entre 4 y 24 son 4
los numeros pares entre 4 y 24 son 6
los numeros pares entre 4 y 24 son 8
los numeros pares entre 4 y 24 son 10
los numeros pares entre 4 y 24 son 12
los numeros pares entre 4 y 24 son 14
los numeros pares entre 4 y 24 son 16
los numeros pares entre 4 y 24 son 18
los numeros pares entre 4 y 24 son 20
los numeros pares entre 4 y 24 son 22
los numeros pares entre 4 y 24 son 24
```

Ejercicio 7

Realizar un algoritmo que muestre la tabla de multiplicar de un número introducido por teclado.

Diagrama de flujo



Codificación

```
tabla = int(input("Ingrese la tabla: "))

for i in range(1, 13):

    resultado = tabla * i

    print(f"{tabla} x {i} = {resultado}")
```

Ejecución

```
Ingrese la tabla: 6
6 x 1 = 6
6 x 2 = 12
6 x 3 = 18
6 x 4 = 24
6 x 5 = 30
6 x 6 = 36
6 x 7 = 42
6 x 8 = 48
6 x 9 = 54
6 x 10 = 60
6 x 11 = 66
6 x 12 = 72
```

Ejercicio 8

Escribe un programa que pida el límite inferior y superior de un intervalo. Si el límite inferior es mayor que el superior lo tiene que volver a pedir. A continuación, se van introduciendo números hasta que introducimos el 0. Cuando termine el programa dará las siguientes informaciones:

- La suma de los números que están dentro del intervalo (intervalo abierto).
- Cuántos números están fuera del intervalo.
- Informa si hemos introducido algún número igual a los límites del intervalo.

Diagrama de flujo



Codificación

lim_inf = 0

lim_sup = 0

num = 1

suma_dentro_intervalo = 0

cont_fuera_intervalo = 0

```
igual_limites = False

while lim_inf >= lim_sup:
    lim_inf = int(input("Introduce el límite inferior del intervalo: "))
    lim_sup = int(input("Introduce el límite superior del intervalo: "))
    if lim_inf >= lim_sup:
        print("ERROR: El límite inferior debe ser menor que el superior.")

while num != 0:
    num = int(input("Introduce un número (0 para salir): "))
    if num > lim_inf and num < lim_sup:
        suma_dentro_intervalo += num
    else:
        cont_fuera_intervalo += 1
    if num == lim_inf or num == lim_sup:
        igual_limites = True

print("La suma de los números dentro del intervalo es", suma_dentro_intervalo)
print("La cantidad de números fuera del intervalo es", cont_fuera_intervalo)

if igual_limites:
    print("Se ha introducido algún número igual a los límites del intervalo.")
else:
    print("No se ha introducido ningún número igual a los límites del intervalo.")
```

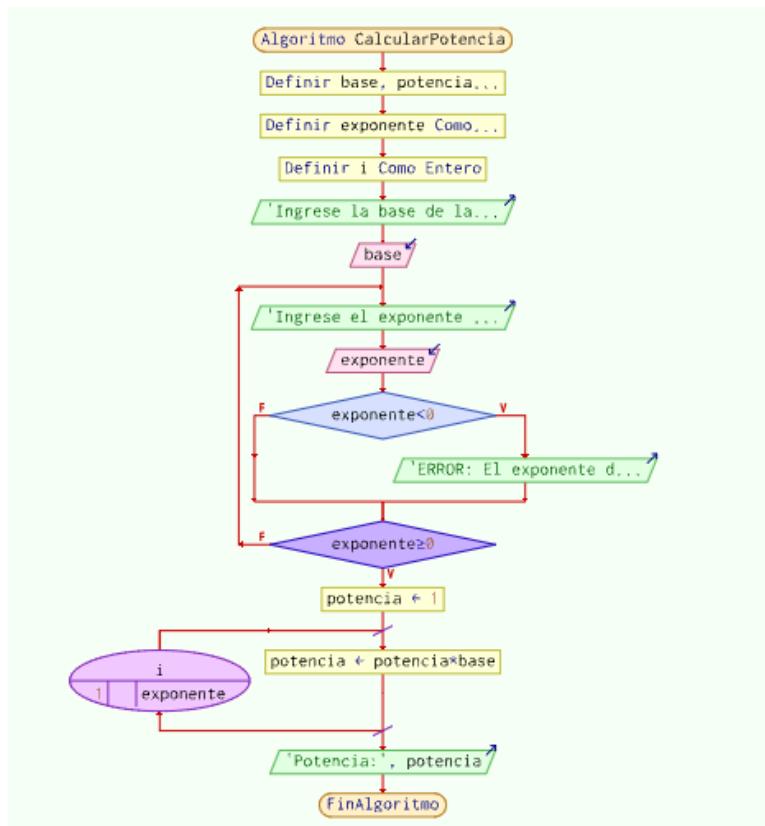
Ejecución

```
===== RESTART: C:/Users/SERVER/Downloads/Pregunta 8.py =====
Introduce el límite inferior del intervalo: 34
Introduce el límite superior del intervalo: 42
Introduce un número (0 para salir): 3
Introduce un número (0 para salir): 0
La suma de los números dentro del intervalo es 0
La cantidad de números fuera del intervalo es 2
No se ha introducido ningún número igual a los límites del intervalo.
```

Ejercicio 9

Escribe un programa que, dados dos números, uno real (base) y un entero positivo (exponente), saque por pantalla el resultado de la potencia. No se puede utilizar el operador de potencia.

Diagrama de flujo



Codificación

```
base = float(input("Ingrese la base de la potencia: "))

exponente = -1
```

```

while exponente < 0:

    exponente = int(input("Ingrese el exponente de la potencia: "))

    if exponente < 0:

        print("ERROR: El exponente debe ser positivo")

```

```

potencia = 1

for i in range(1, exponente + 1):

    potencia *= base

print("Potencia:", potencia)

```

Ejecución

.

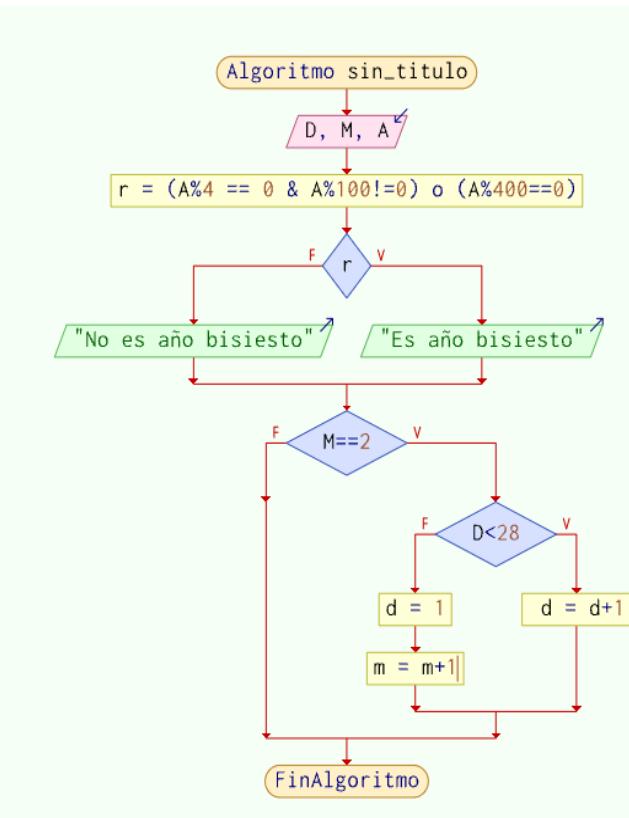
```

>>> =
      = RESTART: C:/Users/SERVER/Downloads/Pregunta 9.py
      Ingrese la base de la potencia: 3
      Ingrese el exponente de la potencia: 2
      Potencia: 9.0

```

Ejercicio 10

Haz un programa que al ingresar una fecha indique si el año es bisiesto o no.



Codificación

```
D = int(input("Ingrese el día (D): "))

M = int(input("Ingrese el mes (M): "))

A = int(input("Ingrese el año (A): "))

r = (A % 4 == 0 and A % 100 != 0) or (A % 400 == 0)

if r:

    print("Es año bisiesto")

else:

    print("No es año bisiesto")

if M == 2:

    if D < 28:

        D += 1

    else:

        D = 1

    M += 1
```

Ejecución

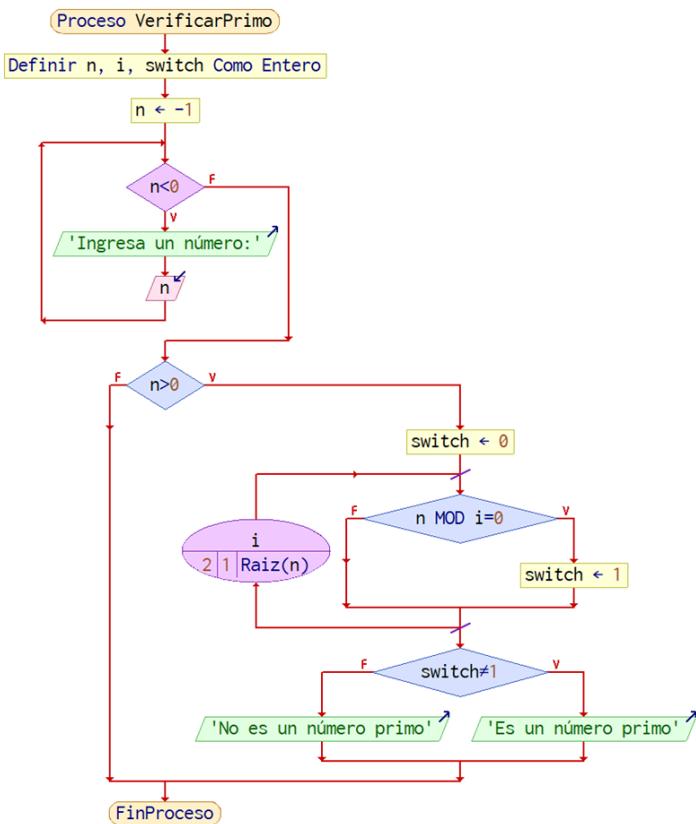
```
Ingrese el dia (D): 12
Ingrese el mes (M): 2
Ingrese el año (A): 2000
Es año bisiesto
```

Ejercicio 11

Escribe un programa que diga si un número introducido por teclado es o no primo.

Un número primo es aquel que sólo es divisible entre él mismo y la unidad. Nota: Es suficiente probar hasta la raíz cuadrada del número para ver si es divisible por algún otro número.

Diagrama de flujo



Codificación en Python

#Ejercicio 11

n=-1

while n<0:

```
n = int(input("Ingresar un número:"))
```

if n>0:

```
    switch = 0
```

```
for i in range(2, int(n**0.5) + 1):
```

```
    if n % i == 0:
```

```

switch = 1

break

if switch!=1:

    print("Es un número primo")

else:

    print("No es un número primo")

```

Ejecución

```

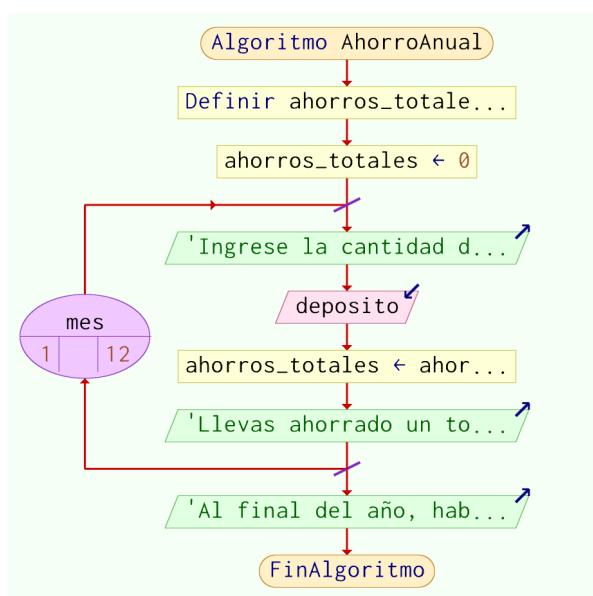
Ingresá un número:2
Es un número primo
PS C:\Users\usuario\OneDrive\Escritorio\Programación\Tarea programación>
gramación\Tarea programación'; & 'c:\Users\usuario\AppData\Local\Microso
vscode\extensions\ms-python.debugpy-2024.0.0-win32-x64\bundled\libs\debu
C:\Users\usuario\OneDrive\Escritorio\Programación\Tarea programación\#Ej
Ingresá un número:3
Es un número primo
PS C:\Users\usuario\OneDrive\Escritorio\Programación\Tarea programación>
7
PS C:\Users\usuario\OneDrive\Escritorio\Programación\Tarea programación>
gramación\Tarea programación'; & 'c:\Users\usuario\AppData\Local\Microso
vscode\extensions\ms-python.debugpy-2024.0.0-win32-x64\bundled\libs\debu
C:\Users\usuario\OneDrive\Escritorio\Programación\Tarea programación\#Ej
Ingresá un número:9
No es un número primo

```

Ejercicio 12

Realizar un algoritmo para determinar cuánto ahorrará una persona en un año, si al final de cada mes deposita cantidades variables de dinero; además, se quiere saber cuánto lleva ahorrado cada mes.

Diagrama de flujo



Codificación en python:

```
ahorros_totales = 0

for mes in range(1, 13):

    deposito = int(input("Ingrese la cantidad depositada para el mes " + str(mes) + ":"))

    ahorros_totales += deposito

    print("Llevas ahorrado un total de: " + str(ahorros_totales))

print("Al final del año, habrás ahorrado un total de: " + str(ahorros_totales))
```

Ejecución del código:

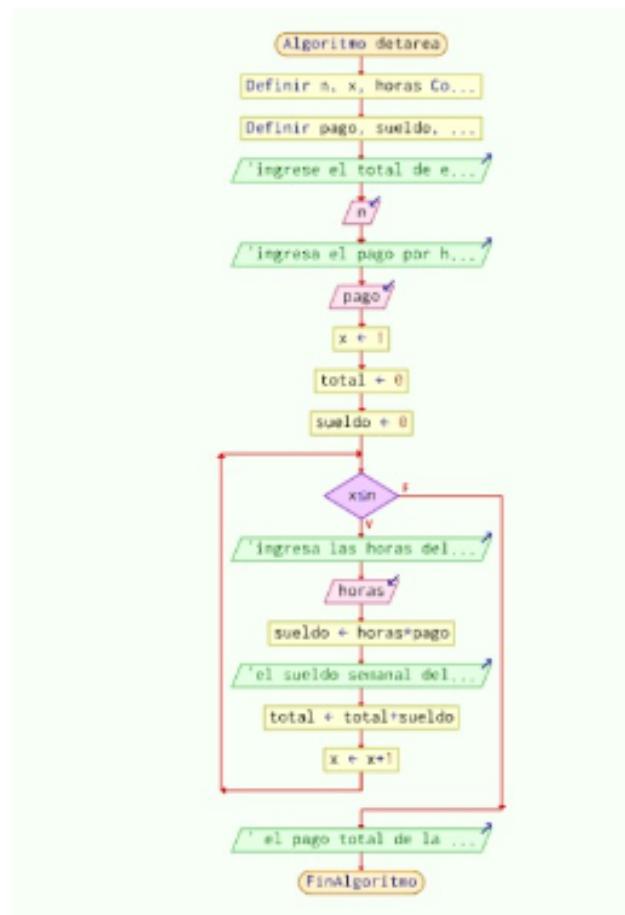
```
Ingrese la cantidad depositada para el mes 1: 120
Llevas ahorrado un total de: 120
Ingrese la cantidad depositada para el mes 2: 120
Llevas ahorrado un total de: 240
Ingrese la cantidad depositada para el mes 3: 120
Llevas ahorrado un total de: 360
Ingrese la cantidad depositada para el mes 4: 120
Llevas ahorrado un total de: 480
Ingrese la cantidad depositada para el mes 5: 120
Llevas ahorrado un total de: 600
Ingrese la cantidad depositada para el mes 6: 120
Llevas ahorrado un total de: 720
Ingrese la cantidad depositada para el mes 7: 120
Llevas ahorrado un total de: 840
Ingrese la cantidad depositada para el mes 8: 120
Llevas ahorrado un total de: 960
Ingrese la cantidad depositada para el mes 9: 120
Llevas ahorrado un total de: 1080
Ingrese la cantidad depositada para el mes 10: 120
Llevas ahorrado un total de: 1200
Ingrese la cantidad depositada para el mes 11: 120
Llevas ahorrado un total de: 1320
Ingrese la cantidad depositada para el mes 12: 120
Llevas ahorrado un total de: 1440
Al final del año, habrás ahorrado un total de: 1440
```

Ejercicio 16

Una empresa les paga a sus empleados con base en las horas trabajadas en la semana.

Realice un algoritmo para determinar el sueldo semanal de N trabajadores y, además, calcule cuánto pagó la empresa por los N empleados.

Diagrama:



Código y ejecución:

```
import os

trabajadores = int(input("¿Cuántos trabajadores tiene la empresa? "))

sueldoxhora = float(input("Sueldo por hora: "))

hacumuladas = 0

for trabajador in range(1, trabajadores + 1):

    htrabajador = 0

    dias = int(input(f"¿Cuántos días ha trabajado el trabajador {trabajador}? "))

    for dia in range(1, dias + 1):

        horas = int(input(f"¿Cuántas horas ha trabajado el trabajador {trabajador} el día {dia}?"))

        htrabajador += horas

    print(f"El sueldo semanal del trabajador {trabajador} es: {htrabajador * sueldoxhora} pesos")
```

```
print(f"El trabajador {trabajador} tiene un sueldo: {htrabajador * sueldoxhora}")
```

```
hacumuladas += htrabajador
```

```
print(f"El pago a los {trabajadores} trabajadores es: {hacumuladas * sueldoxhora}")
```

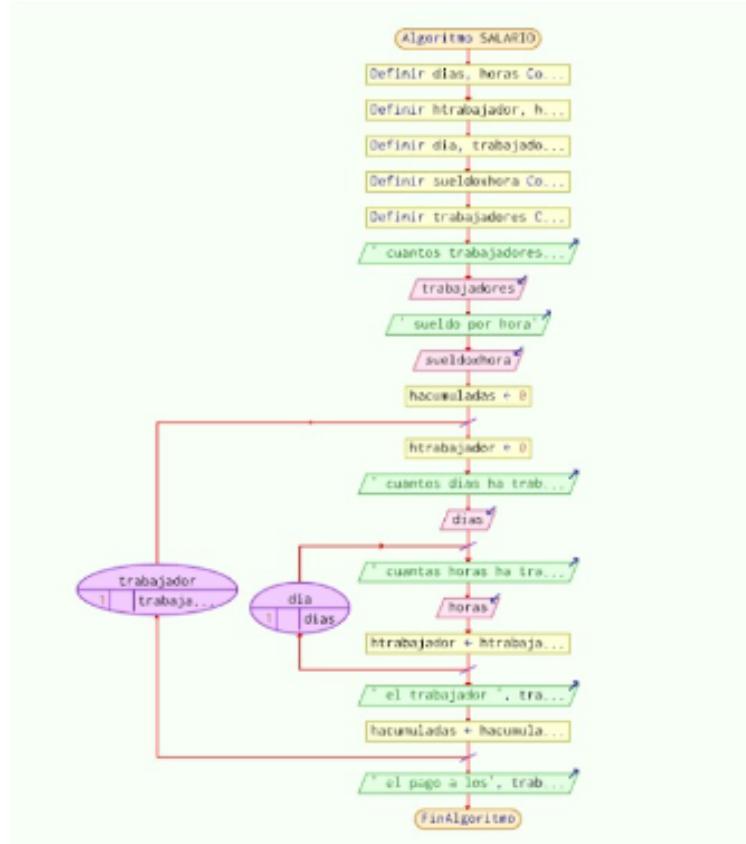
Ejecución

```
==== RESTART: C:\Users\hp\Desktop\Semestre 1\Programacion\Deber\Ej.16 DEBER.py ===
¿Cuántos trabajadores tiene la empresa? 2
Sueldo por hora: 6
¿Cuántos días ha trabajado el trabajador 1? 9
¿Cuántas horas ha trabajado el trabajador 1 el dia 1? 4
¿Cuántas horas ha trabajado el trabajador 1 el dia 2? 5
¿Cuántas horas ha trabajado el trabajador 1 el dia 3? 5
¿Cuántas horas ha trabajado el trabajador 1 el dia 4? 7
¿Cuántas horas ha trabajado el trabajador 1 el dia 5? 8
¿Cuántas horas ha trabajado el trabajador 1 el dia 6? 2
¿Cuántas horas ha trabajado el trabajador 1 el dia 7? 1
¿Cuántas horas ha trabajado el trabajador 1 el dia 8? 8
¿Cuántas horas ha trabajado el trabajador 1 el dia 9? 7
El trabajador 1 tiene un sueldo: 282.0
¿Cuántos días ha trabajado el trabajador 2? 2
¿Cuántas horas ha trabajado el trabajador 2 el dia 1? 5
¿Cuántas horas ha trabajado el trabajador 2 el dia 2? 6
El trabajador 2 tiene un sueldo: 66.0
El pago a los 2 trabajadores es: 348.0
```

Ejercicio 17

Una empresa les paga a sus empleados con base en las horas trabajadas en la semana. Para esto, se registran los días que trabajó y las horas de cada día. Realice un algoritmo para determinar el sueldo semanal de N trabajadores y además calcule cuánto pagó la empresa por los N empleados.

Diagrama:



Código y ejecución:

```

import os

n = int(input("Ingrese el total de empleados: "))

pago = float(input("Ingrese el pago por hora: "))

x = 1
total = 0

while x <= n:

    horas = int(input(f"Ingrese las horas del trabajador {x}: "))

    sueldo = horas * pago

    print(f"El sueldo semanal del trabajador {x} es: ${sueldo}")

    total += sueldo

    x += 1

print(f"El pago total de la empresa es: ${total}")

```

```

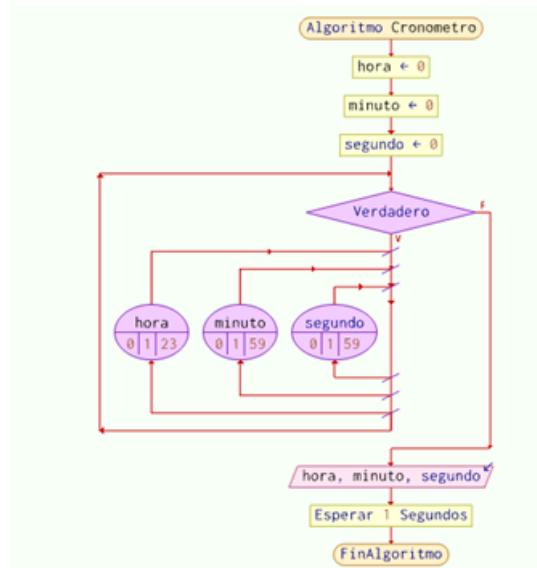
= RESTART: C:\Users\hp\Desktop\Semestre 1\Programacion\Deber\Ej.17 DEBER.PY
Ingrese el total de empleados: 3
Ingrese el pago por hora: 8
Ingrese las horas del trabajador 1: 5
El sueldo semanal del trabajador 1 es: $40.0
Ingrese las horas del trabajador 2: 6
El sueldo semanal del trabajador 2 es: $48.0
Ingrese las horas del trabajador 3: 10
El sueldo semanal del trabajador 3 es: $80.0
El pago total de la empresa es: $168.0

```

Ejercicio 18

Hacer un programa que muestre un cronómetro, indicando las horas, minutos y segundos.

Diagrama



Código y ejecución:

```
import time
```

```
hora = 0
```

```
minuto = 0
```

```
segundo = 0
```

```

while True:

    for hora in range(23):

        for minuto in range(59):

            for segundo in range(59):

                print(f'{hora:02}:{minuto:02}:{segundo:02}', end="\r")

                time.sleep(1)

```

```

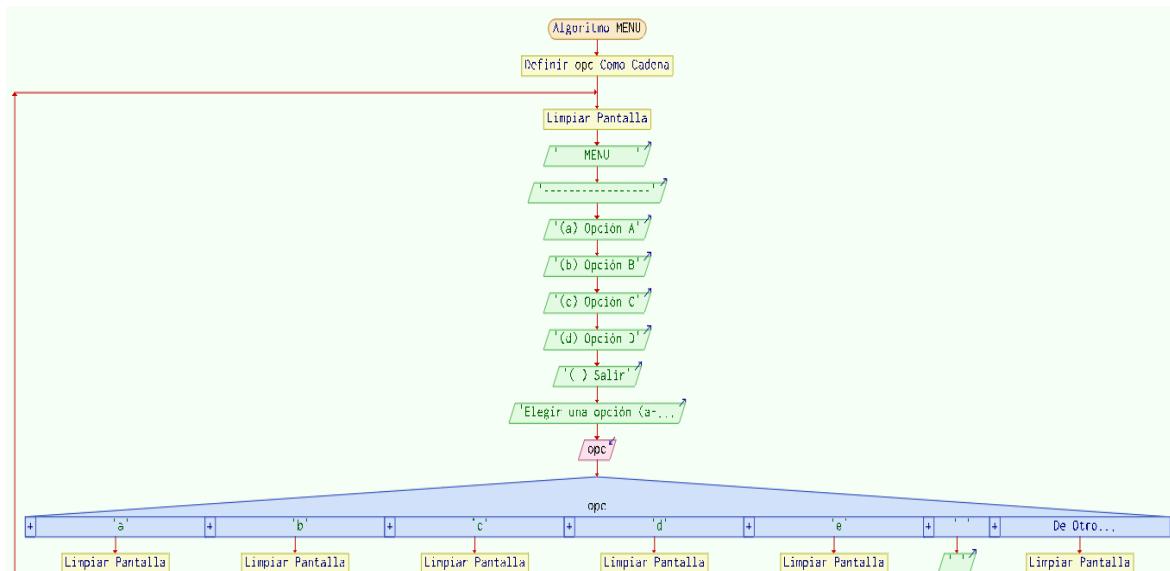
AMD64] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> ===== RESTART: C:\Users\hp\Desktop\Semestre 1\Programacion\Ej.18\Ej..py =====
00:00:0000:00:0100:00:0200:00:0300:00:0400:00:0500:00:0600:00:0700:00:08
00:00:0900:00:1000:00:1100:00:1200:00:1300:00:1400:00:1500:00:1600:00:1
700:00:1800:00:1900:00:2000:00:21      00:00:22

```

Ejercicio 19

Realizar un ejemplo de menú, donde podemos escoger las distintas opciones hasta que seleccionamos la opción de “Salir”

Diagrama



Codificación en Phyton:

```
opcion = ""  
  
while opcion != '':  
  
    contador = 0  
  
    print("Menú:")  
  
    print("a. Opción A")  
  
    print("b. Opción B")  
  
    print("c. Opción C")  
  
    print("d. Opción D")  
  
    print("e. Opción E")  
  
    print("espacio. Salir")  
  
    opcion = input("Seleccione una opción: ")  
  
    if opcion == 'a' or opcion == 'A':  
  
        print("Ha seleccionado la Opción A")
```

```
elif opcion == 'b' or opcion == 'B':  
  
    print("Ha seleccionado la Opción B")  
  
elif opcion == 'c' or opcion == 'C':  
  
    print("Ha seleccionado la Opción C")  
  
elif opcion == 'd' or opcion == 'D':  
  
    print("Ha seleccionado la Opción D")  
  
elif opcion == 'e' or opcion == 'E':  
  
    print("Ha seleccionado la Opción E")  
  
elif opcion == ' ':  
  
    print("Saliendo del programa...")  
  
else:  
  
    print("Opción no válida. Inténtelo de nuevo.")
```

Ejecución

```

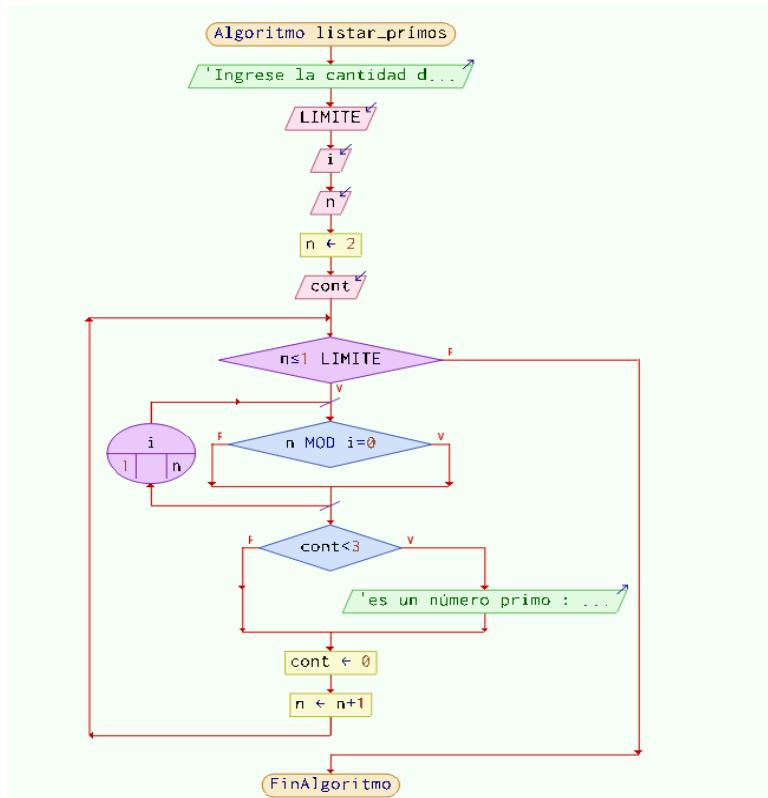
IDLE Shell 3.12.1
File Edit Shell Debug Options Window Help
Python 3.12.1 (tags/v3.12.1:2305ca5, Dec 7 2023, 22:03:25) [MSC v.1937 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> ===== RESTART: C:\Users\Pinpu\Documents\DEBER.SAMUEL.MONTE\18.py =====
Menú:
a. Opción A
b. Opción B
c. Opción C
d. Opción D
e. Opción E
espacio. Salir
Seleccione una opción: a
Ha seleccionado la Opción A
Menú:
a. Opción A
b. Opción B
c. Opción C
d. Opción D
e. Opción E
espacio. Salir
Seleccione una opción: b
Ha seleccionado la Opción B
Menú:
a. Opción A
b. Opción B
c. Opción C
d. Opción D
e. Opción E
espacio. Salir
Seleccione una opción:
Saliendo del programa...

```

Ejercicio 20

Mostrar en pantalla los N primero número primos. Se pide por teclado la cantidad de números primos que queremos mostrar.

Diagrama



Codificación en python:

```

n = int(input("Ingrese la cantidad de números primos que desea imprimir (máximo 100): "))

if 1 <= n <= 100:

    numeros_primos = []

    i = 2

    while n > 0 and i <= 100:

        es_primo = True

        j = 2

        while j <= int(i**0.5):

            if i % j == 0:

                es_primo = False

                break

            j += 1

        if es_primo:

            print(i, end=' ')

            n -= 1

        i += 1

print()

else:

    print("Por favor, ingrese un valor entre 1 y 100.")

```

Ejecución

```

fit Shell Debug Options Window Help
python 3.12.1 (tags/v3.12.1:2305ca5, Dec  7 2023, 22:03:25) [MSC v.1937 64 bit (AMD64)] on win32
type "help", "copyright", "credits" or "license()" for more information.

```

```

RESTART: C:\Users\Pimpu\Documents\DEBER.SAMUEL.MONTE\20.py
ingrese la cantidad de números primos que desea imprimir (máximo 100): 5
3 5 7 11

```





Programación Modular

MANUAL

Unidad 2

UNIVERSIDAD DE LAS FUERZAS ARMADAS ESPE

NRC: 17832

Introducción a la Programación

Ing.Edgar Solis

05 de Febrero del 2024

Unidad 2

1. Subprogramas : Funciones o Procedimientos

1.1. Definición de Subprograma

También conocidos como procedimientos, funciones o subrutinas, son porciones de código que realizan tareas específicas dentro de un programa más grande. Estas secciones de código pueden ser llamadas y ejecutadas desde otras partes del programa principal.

Los subprogramas ayudan a organizar y modularizar el código, facilitando así la comprensión, mantenimiento y reutilización del software. Aquí algunas de las características clave:

- **Reutilización de código:** Los subprogramas permiten escribir una vez una porción de código y utilizarla en múltiples lugares del programa principal o en diferentes programas
- **Modularidad:** Los subprogramas facilitan la división de un programa complejo en partes más pequeñas y manejables.
- **Facilita el mantenimiento:** Al dividir un programa en subprogramas más pequeños, el mantenimiento se vuelve más sencillo.
- **Claridad y legibilidad del código:** El uso de subprogramas puede hacer que el código sea más claro y legible al organizar tareas específicas en funciones o procedimientos con nombres descriptivos.
- **Parámetros:** Los subprogramas pueden aceptar parámetros, lo que permite la personalización de su comportamiento.
- **Retorno de valores:** En el caso de las funciones, la capacidad de devolver un valor resulta fundamental. Esto permite que un subprograma calcule un resultado y lo comparta con el resto del programa.

Ejemplos:

Función para calcular el área de un rectángulo:

```
def calcular_area_rectangulo(base, altura):  
    area = base * altura  
    return area  
  
# Llamada al subprograma  
  
base = 6  
  
altura = 4  
  
resultado = calcular_area_rectangulo(base, altura)  
  
print("El área del rectángulo con base", base, "y altura", altura, "es:", resultado)
```

Función que recibe el radio de un círculo como parámetro y devuelve su área:

```
def calcular_area_circulo(radio):  
    pi = 3.14159  
  
    area = pi * radio * 2  
  
    return area  
  
# Llamada al subprograma  
  
radio = 5  
  
resultado = calcular_area_circulo(radio)  
  
print("El área del círculo con radio", radio, "es:", resultado)
```

Autoevaluación 1

1. ¿Qué es un subprograma en programación?

- a) Una variable global
- b) Una función o procedimiento
- c) Un comentario en el código
- d) Un bucle for

2. ¿Cuál es la principal función de un subprograma?

- a) Declarar variables
- b) Realizar cálculos matemáticos
- c) Agrupar y reutilizar código
- d) Imprimir mensajes en la consola

3. ¿Cuál es una característica clave de los subprogramas?

- a) No pueden aceptar parámetros
- b) Solo pueden devolver valores numéricos
- c) Pueden ser llamados y reutilizados en el programa principal
- d) Siempre deben estar ubicados al final del código

4. ¿Qué es un parámetro en el contexto de los subprogramas?

- a) Una variable local
- b) Un valor constante
- c) Una entrada proporcionada al subprograma
- d) Una condición de bucle

5. ¿Qué beneficio ofrece el uso de subprogramas en el desarrollo de software?

- a) Mayor complejidad en el código
- b) Menor modularidad
- c) Facilita la reutilización y mantenimiento del código

- d) Aumenta la dificultad de depuración

1.1.1. Tipos de subprogramas

a. Funciones:

- Devuelven un valor como resultado.
- Pueden aceptar parámetros.
- Su ejecución implica el cálculo o procesamiento de datos.

Ejemplo:

```
def suma(a, b):  
    return a + b  
  
resultado = suma(3, 5)
```

b. Procedimientos:

- Realizan una tarea sin devolver un valor explícito.
- Pueden aceptar parámetros.
- Su ejecución implica realizar una serie de acciones.

Ejemplo:

```
def imprimir_saludo(nombre):  
    print(f"Hola, {nombre}!")  
  
imprimir_saludo("Juan")
```

Además de estas dos categorías principales, hay variantes o combinaciones de funciones y procedimientos, como los siguientes:

- **Métodos**

Son subprogramas asociados a objetos en la programación orientada a objetos (POO). Se invocan sobre instancias específicas de una clase y pueden tener acceso a los atributos del objeto.

- **Subrutinas**

Término general que se utiliza para referirse a funciones y procedimientos en algunos contextos.

- **Bloques de código anónimos:**

Algunos lenguajes permiten definir bloques de código sin nombre (como funciones anónimas o lambdas) que pueden ser utilizados en expresiones o pasados como argumentos.

La elección entre el uso de funciones o procedimientos generalmente depende de si se espera o no un valor de retorno. Ambos tipos de subprogramas son herramientas valiosas para mejorar la claridad, modularidad y reutilización del código.(Gomis, 2018)

Autoevaluación 2

- 1. ¿Cuál es la principal diferencia entre una función y un procedimiento en programación?**
 - Las funciones devuelven un valor, los procedimientos no.
 - Los procedimientos devuelven un valor, las funciones no.
 - Ambos siempre devuelven un valor.
 - Ambos nunca devuelven un valor.
- 2. ¿Qué tipo de subprograma se asocia comúnmente con la programación orientada a objetos?**
 - Funciones
 - Procedimientos
 - Métodos
 - Subrutinas
- 3. ¿Qué hace que una función lambda sea diferente de una función convencional?**
 - Las funciones lambda no pueden aceptar parámetros.
 - Las funciones lambda pueden devolver múltiples valores.
 - Las funciones lambda son funciones anónimas.
 - Las funciones lambda no pueden contener bucles.

4. ¿Qué caracteriza a una subrutina en programación?

- a) Puede devolver un valor y aceptar parámetros.
- b) Siempre devuelve un valor pero no acepta parámetros.
- c) Siempre acepta parámetros pero no devuelve un valor.
- d) No puede devolver un valor ni aceptar parámetros.

5. ¿Cuál es uno de los beneficios principales de utilizar subprogramas en programación?

- a) Incrementan la complejidad del código.
- b) Facilitan el mantenimiento y la reutilización del código.
- c) Sólo son útiles en programas pequeños.
- d) Sólo se utilizan en casos específicos de programación.

1.2. Declaración , implementación y llamada de Subprogramas

La declaración de un subprograma define su firma, indicando el nombre, los parámetros y el tipo de retorno. La implementación detalla el código interno del subprograma. Luego, la llamada se realiza desde otra parte del programa principal para ejecutar el código del subprograma.

Declaración:

- La declaración se refiere a la especificación de la firma de la función o subprograma.
- Incluye información como el nombre de la función, los tipos y nombres de los parámetros (si los hay), y el tipo de valor de retorno (si la función devuelve un valor).
- La declaración proporciona una interfaz para el uso de la función, estableciendo cómo debe ser invocada.

Sintaxis:

Declaración de una función:

```
def mi_funcion(parametro):  
  
    resultado = parametro * i  
  
    return resultado
```

Implementación:

- La implementación es la fase en la que se define el código interno del subprograma.
- Aquí es donde escribes las instrucciones y lógica que se ejecutarán cuando la función sea llamada.
- La implementación es lo que determina el comportamiento real de la función.

Sintaxis:

```
def sumar(a, b):  
  
    """
```

Función para sumar dos números.

Parámetros:

a (float): Primer número a sumar.

b (float): Segundo número a sumar.

Retorna:

float: La suma de los dos números.

"""

```
resultado = a + b
```

```
return resultado
```

Llamada de Subprogramas:

- La llamada de subprogramas se refiere al acto de invocar o utilizar la función en el código principal.

- Se realiza proporcionando los valores necesarios para los parámetros (si los hay) y recibiendo, si es el caso, el valor de retorno de la función.

- La llamada permite reutilizar el código encapsulado en la función sin necesidad de repetir su implementación.

Sintaxis:

```
def saludar(nombre):
```

"""

Función para saludar a alguien.

Parámetros:

nombre (str): El nombre de la persona a saludar.

Retorna:

str: El saludo personalizado.

"""

```
mensaje = "¡Hola, " + nombre + "! ¿Cómo estás?"
```

```
return mensaje
```

Estas fases son fundamentales para organizar y modularizar el código, permitiendo la creación de programas más estructurados y mantenibles.

Autoevaluación 3

- 1. ¿Qué aspecto de un subprograma se define durante la fase de declaración?**
 - a. Código interno
 - b. Nombre y parámetros
 - c. Tipo de retorno
 - d. Sintaxis
- 2. ¿Cuál es el propósito de la declaración de un subprograma?**
 - a. Definir el código interno
 - b. Especificar la firma y la interfaz
 - c. Realizar la llamada
 - d. Proporcionar la sintaxis de implementación
- 3. ¿En qué fase se define el código interno del subprograma?**
 - a. Declaración
 - b. Llamada
 - c. Implementación
 - d. Sintaxis

4. ¿Qué información incluye la declaración de un subprograma?

- a. Código interno y sintaxis
- b. Nombre, parámetros y tipo de retorno
- c. Llamadas y sintaxis
- d. Implementación y declaración

5. ¿Cuál es el propósito de la llamada de subprogramas?

- a. Definir la interfaz
- b. Proporcionar la sintaxis
- c. Invocar o utilizar la función
- d. Especificar el tipo de retorno

1.3. Argumentos y Parámetros

Parámetros:

Un *parámetro* representa un valor que el procedimiento espera que se pase al llamarlo. La declaración del procedimiento define sus parámetros.

Cuando se define un procedimiento Function o Sub, se especifica una *lista de parámetros* entre paréntesis inmediatamente después del nombre del procedimiento. En cada parámetro se especifica un nombre, un tipo de datos y un mecanismo de paso. También se puede indicar que un parámetro es opcional. Esto significa que el código de llamada no tiene que pasarle un valor.

El nombre de cada parámetro actúa como una *variable local* del procedimiento. Use el nombre del parámetro de la misma manera que cualquier otra variable

Argumentos:

Un *argumento* representa el valor que se pasa a un parámetro de procedimiento al llamar a este último. El código de llamada proporciona los argumentos cuando llama al procedimiento.

Cuando se llama a un procedimiento Function o Sub, se incluye una *lista de argumentos* entre paréntesis inmediatamente después del nombre del procedimiento. Cada argumento corresponde al parámetro que se encuentra en la misma posición en la lista.

A diferencia de la definición de parámetro, los argumentos no tienen nombres. Cada argumento es una expresión que puede contener cero o más variables, constantes y literales. El tipo de datos de la expresión evaluada normalmente debe coincidir con el tipo de datos definido para el parámetro correspondiente y, en cualquier caso, debe poder convertirse al tipo de parámetro.

Sintaxis

La sintaxis utilizada para la declaración de la lista de parámetros formales es similar a la utilizada en la declaración de cualquier identificador

1.4 Argumentos por posición:

```
def resta(a, b):
    return a - b
resta(30, 10) # argumento 30 => posición 0 => parámetro a
                # argumento 10 => posición 1 => parámetro b
```

1.5 Argumentos por nombre:

```
resta(b=30, a=10)
```

Autoevaluación 4

1. ¿Qué representa un parámetro, en un procedimiento en programación?

- a. Un valor pasado al llamar el procedimiento
- b. Una variable local del procedimiento
- c. Una lista de argumentos
- d. Un mecanismo de paso opcional

2. ¿Qué se especifica al definir un procedimiento Function o Sub en cuánto a los parámetros?

- a. Solo el nombre del parámetro
- b. Solo el tipo de datos del parámetro
- c. Nombre, tipo de dato y mecanismo de paso del parámetro
- d. Solo el mecanismo de paso del parámetro

3. ¿Cuál es la función de un argumento?

- a. Representar un valor esperado en la declaración
- b. Actuar como una variable local
- c. Ser un tipo de dato definido para el parámetro
- d. Representar el valor pasado

4. ¿Qué caracteriza a los argumentos por posición, en la llamada?

- a. Tiene nombres asignados

- b. Se especifican entre paréntesis
- c. Corresponden a la posición en la lista de parámetros
- d. Pueden contener cero o más variables

5. ¿En qué consisten los argumentos por nombre al llamar a un procedimiento?

- a. No es una forma válida de llamar a un parámetro
- b. Los nombres de los argumentos no importan
- c. Corresponden a la posición en la lista de parámetros
- d. Se especifica el nombre y el tipo de dato de cada argumento

1.6 Ámbito de las Variables

Dentro de la programación se conoce a las variables como un valor que se puede ir modificando a lo largo de un proceso; pues en muchos lenguajes de programación estas deben ser declaradas antes de poder utilizarlas, pues es un sinsentido usar algo que ni siquiera existe dentro del contexto en que se planea utilizar, e incluso esta misma fuera de su margen de trabajo no se puede utilizar.

Por eso desde que se declara la variable (creación), y su utilización en el bloque que se ha declarado, sin embargo, surge la siguiente pregunta con respecto a la variable ¿Hasta qué punto existe?

Pues dentro de lenguajes como lo son Python o C++, las variables existen siempre y cuando continúen en el bloque de trabajo en el que estas han sido asignadas, con ello un buen ejemplo sería el siguiente:

Ejemplo 1

```
# Aquí 'mi_variable' no existe
```

```
mi_variable = "¡Hola, mundo!" # Aquí 'mi_variable' existe  
  
print(mi_variable) # Aquí 'mi_variable' existe.  
  
mi_variable = None # Aquí 'mi_variable' todavía existe, pero su valor es None.  
  
print(mi_variable) # Aquí 'mi_variable' existe, pero su valor es None.  
  
del mi_variable # Aquí 'mi_variable' ya no existe.  
  
# print(mi_variable) # Descomenta esta línea y obtendrás un error porque 'mi_variable' ya  
no existe.
```

Ejemplo 2

```
# Aquí 'a', 'b', 'c' y 'resultado' no existen  
  
a = 5 # Aquí 'a' existe.  
  
b = 10 # Aquí 'b' existe.  
  
c = 2 # Aquí 'c' existe.  
  
# Aquí 'a', 'b' y 'c' existen.  
  
resultado = a * b / c # Aquí 'resultado' existe.  
  
print(resultado) # Aquí 'resultado' existe. Imprime 25.0.  
  
del resultado # Aquí 'resultado' ya no existe.  
  
# print(resultado) # Descomenta esta línea y obtendrás un error porque 'resultado' ya no  
existe.
```

Considerando lo anterior, hay lenguajes como JavaScript en la que se tiene que tener muy en cuenta estos aspectos de las variables que por lo general son fácilmente

identificables en otros lenguajes como Python o C++, pues siguen guardando la variable incluso después del bloque, es debido a esto que se consideran dos conceptos:

- **Variables Globales:** Son aquellas que se involucran a lo largo de todo el programa, por lo que se pueden utilizar en cualquier parte del mismo.
- **Variables Locales:** Se definen como aquellas que se encuentran focalizadas en un bloque específico de trabajo, por lo que fuera de este no se pueden utilizar.

Autoevaluación 5

1. ¿Qué es una variable en programación?

- a. Un valor que no puede cambiar.
- b. Un valor que puede modificarse a lo largo de un proceso.
- c. Una función que se puede llamar en cualquier momento.
- d. Un tipo de dato que solo puede ser numérico.

2. ¿Cuándo existe una variable en lenguajes como Python o C++?

- a. Solo después de que se declara y antes de que se utilice.
- b. Solo mientras se está ejecutando el bloque de código en el que se declaró.
- c. Desde el inicio hasta el final del programa, independientemente de dónde se declare.
- d. Solo cuando se asigna un valor.

3. ¿Qué es una variable global?

- a. Una variable que solo puede utilizarse en la función en la que se declaró.

- b. Una variable que se puede utilizar en cualquier parte del programa.
- c. Una variable que solo puede tener valores numéricos.
- d. Una variable que solo puede tener valores de cadena.

4. ¿Qué es una variable local?

- a. Una variable que solo puede utilizarse en la función en la que se declaró.
- b. Una variable que se puede utilizar en cualquier parte del programa.
- c. Una variable que solo puede tener valores numéricos.
- d. Una variable que solo puede tener valores de cadena.

5. En JavaScript, ¿qué sucede con las variables después de que se ejecuta su bloque de código?

- a. Las variables se eliminan inmediatamente después de que se ejecuta su bloque de código.
- b. Las variables se guardan y se pueden utilizar en cualquier parte del programa.
- c. Las variables se convierten en variables globales.
- d. Las variables se convierten en constantes y no pueden modificarse.

1.7 Funciones de librerías o módulos

En la programación, las funciones de librerías o módulos son conjuntos de código prescrito que puedes utilizar para realizar tareas específicas sin tener que volver a escribir ese código. Estas funciones pueden incluir operaciones matemáticas, manipulación de archivos, acceso a bases de datos, interfaz de

usuario, y mucho más. Las librerías o módulos pueden ser estándar (incluidos con el lenguaje de programación) o externos (desarrollados por terceros).

Autoevaluación 6

- 1. ¿Cuál es el propósito principal de las funciones de librerías o módulos en programación?**
 - a) Generar código aleatorio.
 - b) Realizar tareas específicas sin tener que volver a escribir ese código.
 - c) Depurar errores en el código.
 - d) Escribir código desde cero en cada proyecto.

- 2. ¿Qué librería de Python se utiliza principalmente para operaciones numéricas y matriciales?**
 - a) Pandas
 - b) Numpy
 - c) Matplotlib
 - d) Seaborn

- 3. ¿Cuál de las siguientes librerías se utiliza para manipulación y análisis de datos en Python?**
 - a) Numpy
 - b) Pandas
 - c) Matplotlib

d) Flask

4. ¿Para qué se utiliza principalmente la librería Scikit-learn en Python?

a) Desarrollo web

b) Aprendizaje automático y minería de datos

c) Visualización de datos

d) Manipulación de archivos

5. ¿Cuál de las siguientes afirmaciones es verdadera sobre las librerías o módulos en programación?

a) Siempre deben ser desarrolladas por el mismo equipo que creó el lenguaje de programación.

b) No se pueden utilizar para realizar operaciones matemáticas.

c) Pueden ser estándar (incluidos con el lenguaje de programación) o externos (desarrollados por terceros).

d) Son útiles sólo para la depuración de código.

1.8 Recursividad

La recursividad es una técnica utilizada en programación para resolver problemas que implican la repetición de un proceso, como la búsqueda o el cálculo matemático. En lugar de una solución iterativa, la recursión utiliza funciones que se llaman a sí mismas para encontrar la solución final.

En Python, podemos implementar la recursividad utilizando funciones recursivas, que son funciones que se llaman a sí mismas directa o indirectamente.

Estas funciones se detienen cuando alcanzan un caso base, es decir, una condición que les indica que deben dejar de llamarse a sí mismas y devolver un valor.

Un ejemplo común de uso de la recursividad es la función factorial. El factorial de un número n se define como el producto de todos los números enteros positivos desde 1 hasta n .

Para describir una función recursiva se debe cumplir tres condiciones:

1.6.1. Debe haber al menos un caso base de parada.

1.6.2. Inducción: Paso recursivo que provoca una llamada recursiva. Debe ser correcto para distintos parámetros de entrada.

1.6.3. Convergencia: Cada paso recursivo debe acercarse a un caso base. Se describe el problema en términos de problemas más sencillos.

- El caso base: Es cualquier expresión donde le dice a la función cuando dejar de llamarse a sí misma, si no tiene un caso base la recursión podría ser un bucle infinito.

1.6.4. Uso de funciones recursivas

Las llamadas funciones de recursivas pueden usarse para actividades que son recursivas por naturaleza y que tienen las siguientes características:

- Cuando los problemas están más cercanos a la descripción matemática.
- Cuando su lectura es más fácil de analizar.
- En estructuras que se adaptan mejora estructuras de datos recursivas.
- Los algoritmos recursivos ofrecen soluciones estructuradas modulares y simples.

Autoevaluación 7

1. ¿Qué es la recursividad en Python?

- a) Un bucle que se repite hasta que se cumple una condición.
- b) Una técnica donde una función se llama a sí misma.
- c) Un método para evitar el uso de funciones.
- d) Una operación de asignación de variables.

2. ¿Cuál es la condición base en una función recursiva?

- a) Es una condición que se cumple al principio de la función.
- b) Es una condición que determina cuántas veces se llama la función recursiva.
- c) Es una condición que se utiliza para detener la recursión.
- d) Es una condición que se utiliza para iniciar la recursión.

3. ¿Cuál es un ejemplo típico de un problema que se puede resolver de manera recursiva en Python?

- a) Suma de números pares.
- b) Búsqueda binaria en una lista ordenada.
- c) Ordenamiento de una lista.
- d) Concatenación de cadenas.

4. ¿Cuál es un posible inconveniente de utilizar la recursividad en Python?

- a) Mayor eficiencia en el uso de la memoria.
- b) La recursividad no es compatible con Python.

- c) Posibilidad de causar un desbordamiento de pila si no se maneja correctamente.
- d) Facilidad de implementación en comparación con los bucles iterativos.

5. ¿Cuál es el caso base en la recursividad para calcular el factorial de un número en Python?

- a) Cuando el número es cero (0).
- b) Cuando el número es uno (1).
- c) Cuando el número es positivo.
- d) Cuando el número es mayor que uno (1).

1.9 Creación de librerías o módulos

Crear una librería en Python implica organizar y empaquetar un conjunto de módulos o funcionalidades para su uso en otros proyectos.

Los pasos para realizar una librería son los siguientes:

1.- Planificación:

Define claramente el propósito y las funcionalidades principales de tu librería. Piensa en cómo quieras que otros desarrolladores la utilicen y en qué problemas específicos resolverá.

2.- Estructura del Proyecto:

Organiza tu proyecto en una estructura coherente. Puedes seguir la estructura típica de un paquete de Python:

```
mi_libreria/
    ├── mi_libreria/
    |   ├── __init__.py
    |   ├── modulo1.py
    |   └── modulo2.py
```

```
|   └── ...
|
|   └── tests/
|       ├── test_modulo1.py
|       ├── test_modulo2.py
|       └── ...
|
└── setup.py
    └── README.md
```

mi_libreria/: Carpeta principal del proyecto.

mi_libreria/_init_.py: Archivo para inicializar el paquete.

mi_libreria/modulo1.py, mi_libreria/modulo2.py: Módulos que contienen funciones y clases.

tests/: Carpeta para pruebas unitarias.

setup.py: Archivo para configurar la instalación de la librería.

README.md: Documentación del proyecto.

3.- Código:

Desarrolla las funciones y clases dentro de los módulos en la carpeta *mi_libreria/*.

Asegúrate de documentar cada función o clase para que otros desarrolladores comprendan cómo usarlas.

4.-Pruebas Unitarias:

Escribe pruebas unitarias en la carpeta *tests/* para garantizar la calidad y corrección de tu código. Utiliza herramientas como unittest o pytest.

5.- Empaquetado:

Crea un archivo *setup.py* en la raíz del proyecto para describir cómo se debe instalar tu librería. Aquí tienes un ejemplo básico:

```
from setuptools import setup, find_packages

setup(
    name='mi_libreria',
```

```
version='0.1',  
packages=find_packages(),  
install_requires=[  
    # Lista de dependencias si las hay  
,  
)
```

6.- Documentación:

Escribe una documentación clara en el archivo README.md. Explica la funcionalidad de tu librería, cómo instalarla y cómo los desarrolladores pueden empezar a utilizarla.

7.- Pruebas y Validación:

Realiza pruebas exhaustivas de tu librería y valida su funcionalidad en diferentes entornos y escenarios.

8.- Publicación:

Puedes publicar tu librería en el Índice de Paquetes de Python (PyPI) para que otros puedan instalarla fácilmente usando pip. Necesitarás crear una cuenta en PyPI y seguir sus instrucciones.

9.- Instalación:

Después de publicar en PyPI, otros desarrolladores pueden instalar tu librería usando el siguiente comando:

10.- Mantenimiento:

Asegúrate de mantener tu librería actualizada y responde a cualquier problema o contribución de la comunidad. Puedes utilizar herramientas como git y plataformas como GitHub para gestionar el desarrollo colaborativo.

1. ¿Cuál es el propósito principal de crear un archivo `__init__.py` en una librería de Python?

- a) Incluir la documentación del proyecto.
- b) Inicializar el paquete y permitir la importación de módulos.
- c) Configurar las dependencias del proyecto.
- d) Definir las pruebas unitarias

2. ¿En qué parte de la estructura del proyecto se encuentran las pruebas unitarias en una librería de Python?

- a) En la carpeta `mi_libreria/`.
- b) En la carpeta `tests/`.
- c) En el archivo `setup.py`.
- d) En el archivo `README.md`.

3. ¿Cuál es el propósito del archivo `setup.py` en el desarrollo de una librería de Python?

- a) Ejecutar las pruebas unitarias.
- b) Definir el código principal de la librería.
- c) Inicializar el paquete.
- d) Documentar el proyecto.

4. ¿Cómo se puede instalar una librería de Python después de publicarla en PyPI?

- a) Descargando un archivo ZIP desde GitHub.
- b) Copiando y pegando el código fuente en un proyecto.
- c) Utilizando el comando `pip install nombre_libreria`.
- d) Creando un entorno virtual desde cero.

5. ¿Cuál es un aspecto clave al mantener una librería de Python?

- a) Publicarla en PyPI solo una vez.
- b) Ignorar problemas reportados por la comunidad.

- c) Mantenerla actualizada y responder a contribuciones.
- d) Cambiar la estructura del proyecto frecuentemente.

2. Arreglos

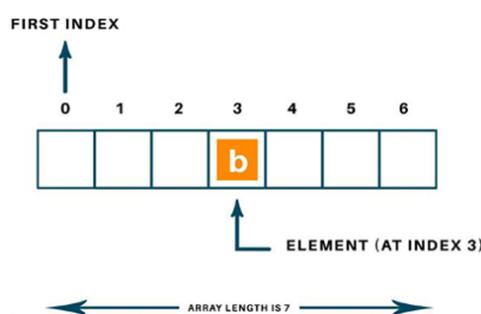
2.1. Arreglos unidimensionales

Unidimensional, significa que tiene solo una dimensión, que se representa como una como una línea, solo que aquí la representamos con una fila para facilitar la comprensión.

Un arreglo es un conjunto finito de variables del mismo tipo, que comparten el mismo nombre. Sin embargo, tienen diferente índice, que vendría siendo su posición dentro del conjunto.

Por lo que un arreglo unidimensional vendría siendo un conjunto representado por una fila en la cual se encuentran los elementos en posiciones indicadas.

Siendo usados para almacenar y manipular conjuntos de datos de manera eficiente.



Cuando en Python se declara el “array” se lo hace indicando su nombre y los de elementos. Por ejemplo:

```
a = [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

El “índice”, que sería la posición dentro del conjunto, permite al usuario manipular los elementos de manera individual. Por ejemplo:

Ejemplos

1. Realiza un programa que multiplique dos elementos y sumen 7.

Programa

```
a = [1, 2, 3, 4, 5, 6, 7, 8, 9]  
print ((a[3]*a[7])+7)
```

Ejecución

39

2. Realiza un programa que dé como resultado el área de una caja si el largo es de 10 m, la altura 30 m y el ancho de 20 m.

Programa

```
a = [10, 30, 20]  
print ("El área es de ", a[1] * a[2] * a[3], " metros cúbicos")
```

Ejecución

El área es de 60000 metros cúbicos.

Autoevaluación 9

1. **¿Cómo se representa un arreglo unidimensional en términos de dimensiones?**
 - Como un conjunto de filas y columnas
 - Como una línea
 - Como un conjunto de capas
 - Como una matriz
2. **¿Cómo se define un arreglo unidimensional en Python indicando su tamaño?**
 - list = [1, 2, 3]
 - array(1, 2, 3)

- c. arr[1, 2, 3]
- d. array = [1][2][3]

3. ¿Cuál es la función del índice en un arreglo unidimensional en Python?

- a. Representa la dimensión del arreglo
- b. Indica el tipo de datos del arreglo
- c. Determina la posición del elemento en el conjunto
- d. Define el nombre del arreglo

4. ¿Cómo se accede a un elemento específico en un arreglo unidimensional en Python?

- a. array[elemento]
- b. array.elemento
- c. array[index]
- d. array(posición)

5. ¿Cuál es el propósito principal de un arreglo unidimensional en programación?

- a. Almacenar datos de forma aleatoria
- b. Representar datos en formato de tabla
- c. Almacenar y manipular conjuntos de datos de manera eficiente
- d. Visualizar datos en un gráfico de barras

2.2. Arreglos bidimensionales

Las matrices bidimensionales son estructuras de datos versátiles que se utilizan con diversos fines en programación, y comprender la indexación y cómo crearlas y manipularlas puede ayudar a resolver problemas de programación complejos de forma eficaz.

Las matrices bidimensionales son estructuras de datos que utilizan dos índices para identificar los elementos, lo que permite operaciones básicas como

acceder a elementos concretos y modificarlos. Se utilizan filas y columnas para colocar elementos en una matriz unidimensional, y son útiles para el cálculo numérico y el tratamiento de imágenes.

Las matrices bidimensionales son herramientas versátiles que se utilizan para la encriptación, los cálculos científicos y los tableros de juego.

En Python, los arreglos bidimensionales son una estructura de datos que permite almacenar datos en una tabla de filas y columnas. Estos arreglos son comúnmente implementados utilizando listas de listas. Cada lista interna representa una fila de la tabla y contiene elementos que representan las columnas.

Aquí hay un ejemplo de cómo se pueden crear y trabajar con arreglos bidimensionales en Python:

Código:

```
tablero_tres_en_raya = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]
```

```
tablero_tres_en_raya[0][1]
```

```
tablero_tres_en_raya[1][1]
```

```
tablero_tres_en_raya[2][2]
```

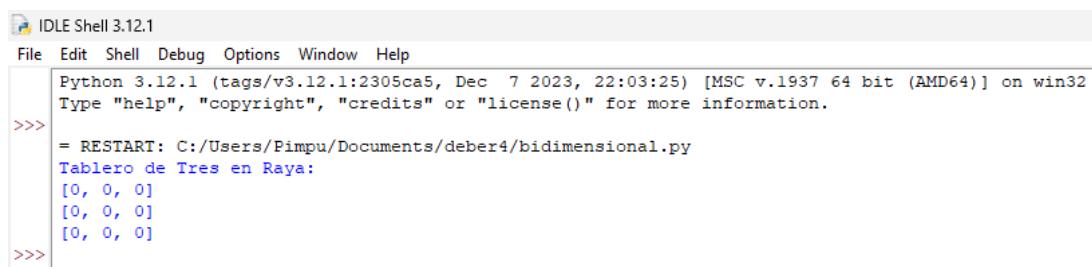
```
print("Tablero de Tres en Raya:")
```

```
for fila in tablero_tres_en_raya:
```

```
    print(fila)
```

Utilizando arreglos bidimensionales es posible crear simuladores de juegos como el de tres en raya.

Ejecución:



```
IDLE Shell 3.12.1
File Edit Shell Debug Options Window Help
Python 3.12.1 (tags/v3.12.1:2305ca5, Dec  7 2023, 22:03:25) [MSC v.1937 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/Pimpu/Documents/deber4/bidimensional.py
Tablero de Tres en Raya:
[0, 0, 0]
[0, 0, 0]
[0, 0, 0]
>>>
```

Los arreglos bidimensionales son útiles para almacenar datos tabulares, como matrices numéricas, tablas de datos y cuadros de juego. Python proporciona una flexibilidad considerable en la manipulación de arreglos bidimensionales gracias a la flexibilidad de las listas. Además, el módulo numpy es ampliamente utilizado para trabajar con arreglos multidimensionales en Python, ya que proporciona una amplia gama de funciones y operaciones optimizadas para el procesamiento numérico.

Autoevaluación 10

- 1. ¿Cómo se crea un arreglo bidimensional en Python?**
 - a) Utilizando la función array() del módulo numpy.
 - b) Declarando una lista de listas.
 - c) Aplicando la función matrix() del módulo numpy.
 - d) Utilizando la función grid() del módulo numpy.
- 2. ¿Cuál es la sintaxis para acceder al elemento en la segunda fila y tercera columna de un arreglo bidimensional en Python?**

a) array[2, 3]

b) array[1][2]

c) array[3][2]

d) array[2][3]

3. ¿Cómo se inicializa un arreglo bidimensional de tamaño 3x3 con todos sus elementos como cero?

a) [[0] * 3] * 3

b) [[0] * 3]

c) [0] * 3

d) [] * 3]

4. ¿Qué método se puede utilizar para recorrer todos los elementos de un arreglo bidimensional en Python?

a) for i in range(len(array))

b) for i in array:

c) for i in range(len(array[0]))

d) for i in range(len(array)):

5. ¿Cuál es la ventaja de utilizar arreglos bidimensionales en Python?

a) Mayor complejidad en el manejo de datos.

b) Estructura más organizada para almacenar datos en forma de tabla.

c) Mayor dificultad para acceder a elementos específicos.

d) Imposibilidad de realizar operaciones matemáticas con los datos almacenados.

2.3. **Arreglos multidimensionales**

Los arreglos multidimensionales son estructuras de datos que contienen elementos organizados en varias dimensiones, como filas y columnas en una matriz, o incluso con más dimensiones, como cubos o tensores. Estos arreglos son esenciales en la programación para representar datos estructurados, como imágenes, audio, datos científicos y más.

Por ejemplo, en Python, la librería NumPy proporciona una implementación eficiente de arreglos multidimensionales, permitiendo realizar operaciones matemáticas avanzadas y manipulaciones de datos con facilidad. Los arreglos multidimensionales son útiles en una amplia gama de aplicaciones, desde el procesamiento de imágenes y señales hasta el análisis de datos y la simulación numérica.

Un arreglo multidimensional tiene más de una columna. Podemos considerar un arreglo multidimensional como una hoja de cálculo de Excel, tiene columnas y filas. Cada columna puede ser considerada como una dimensión.

ejemplo:

```
import numpy as np  
  
# Crear un arreglo multidimensional de 2x3  
  
arr = np.array([[1, 2, 3], [4, 5, 6]])  
  
print(arr)
```

En este ejemplo, creamos un arreglo de tamaño 2×3 , lo que significa que tiene dos filas y tres columnas. Los elementos del arreglo son enteros del 1 al 6.

La salida del código será:

`[[1 2 3]`

`[4 5 6]]`

Autoevaluación 11

1. ¿Qué es un arreglo multidimensional en programación?

- a) Una estructura de datos que solo puede contener números enteros.
- b) Una estructura de datos que puede contener elementos organizados en varias dimensiones.
- c) Una función en Python para manipular cadenas de texto.
- d) Un tipo de bucle utilizado para iterar sobre una lista en Python.

2. ¿Cuál de las siguientes librerías de Python es ampliamente utilizada para trabajar con arreglos multidimensionales?

- a) Matplotlib
- b) Pandas
- c) NumPy
- d) Scikit-learn

3. ¿Cuál de las siguientes NO es una dimensión común en un arreglo multidimensional?

- a) Altura
- b) Ancho
- c) Tiempo

d) Peso

4. ¿Qué operación se utiliza para acceder a un elemento específico en un arreglo multidimensional en Python?

a) get()

b) access()

c) []

d) ()

5. ¿Qué método se utiliza para crear un arreglo multidimensional en NumPy?

a) array()

b) matrix()

c) multidimensional()

d) tensor()

2.4. Paso de Arreglos a funciones

Declaración , implementación y llamada de Subprogramas

La declaración de un subprograma define su firma, indicando el nombre, los parámetros y el tipo de retorno. La implementación detalla el código interno del subprograma. Luego, la llamada se realiza desde otra parte del programa principal para ejecutar el código del subprograma.

Declaración:

- La declaración se refiere a la especificación de la firma de la función o subprograma.

- Incluye información como el nombre de la función, los tipos y nombres de los parámetros (si los hay), y el tipo de valor de retorno (si la función devuelve un valor).
- La declaración proporciona una interfaz para el uso de la función, estableciendo cómo debe ser invocada.

Sintaxis general de declaración:

```
tipo_de_retorno nombre_de_la_funcion(tipo_parametro nombre_parametro, ...);
```

Ejemplo de declaración de función:

```
int suma(int a, int b);
```

Implementación:

- La implementación es la fase en la que se define el código interno del subprograma.
- Aquí es donde escribes las instrucciones y lógica que se ejecutarán cuando la función sea llamada.
- La implementación es lo que determina el comportamiento real de la función.

Sintaxis:

Cómo definir el código en el cuerpo de una función.

Sintaxis de implementación de función

```
tipo_de_retorno nombre_de_la_funcion(tipo_parametro nombre_parametro, ...)  
// Cuerpo de la función  
}
```

Ejemplo de implementación de función:

```
int suma(int a, int b) {  
  
    return a + b;  
  
}
```

2.5. *Algoritmos de Ordenación (selección,intercambio y burbuja)*

Son un conjunto de instrucciones que toman un arreglo o lista como entrada y organizan los elementos en un orden particular.

Selección (Selection Sort):

Principio de Funcionamiento: Este algoritmo divide la lista en dos partes: la parte ordenada y la parte no ordenada. En cada iteración, busca el elemento mínimo en la parte no ordenada y lo intercambia con el primer elemento de la parte no ordenada.

Eficiencia: A pesar de ser fácil de entender, no es eficiente para listas grandes, ya que tiene una complejidad cuadrática $O(n^2)$.

Ejemplo:

```
def ord_seleccion(arreglo):  
  
    for i in range(len(arreglo) - 1):  
  
        menor = i  
  
        for j in range(i + 1, len(arreglo)):  
  
            if arreglo[j] < arreglo[menor]:
```

```
menor = j

if menor != i:

    arreglo[menor], arreglo[i] = arreglo[i], arreglo[menor]

a = [22, 25, 12, 64, 11]

ord_seleccion(a)

print(a)
```

Intercambio (Bubble Sort):

Principio de Funcionamiento: Este algoritmo compara repetidamente pares adyacentes de elementos y los intercambia si están en el orden incorrecto. Este proceso se repite hasta que la lista esté ordenada.

Eficiencia: Es simple pero ineficiente para listas grandes debido a su complejidad cuadrática $O(n^2)$. Además, tiende a ser más lento que otros algoritmos de ordenación en la práctica.

Ejemplo:

```
def bubbleSort(arr):

    n = len(arr)

    for i in range(n):

        for j in range(0, n-i-1):

            if arr[j] > arr[j+1] :

                arr[j], arr[j+1] = arr[j+1], arr[j]
```

```
arr = [2, 4, 21, 16, 14, 10]
```

```
bubbleSort(arr)
```

```
print(arr)
```

Intercambio (Insertion Sort):

Principio de Funcionamiento: Similar al proceso de ordenar cartas en una mano. Se toma un elemento de la lista y se inserta en la posición correcta dentro de la parte ya ordenada de la lista.

Eficiencia: También tiene una complejidad cuadrática $O(n^2)$, pero puede ser más eficiente en ciertos casos que Selection Sort o Bubble Sort, especialmente para listas pequeñas o casi ordenadas.

Autoevaluación 12

1. ¿Cuál de los siguientes algoritmos de ordenación divide la lista en dos partes, ordenada y no ordenada, y busca el elemento mínimo en la parte no ordenada en cada iteración?

- A) Bubble Sort
- B) Insertion Sort
- C) Selection Sort
- D) Merge Sort

2. En el algoritmo de burbuja, ¿cómo se comparan y ordenan los elementos?

- A) Se compara cada elemento con el primero y se intercambian si son diferentes.
- B) Se compara cada elemento con el siguiente y se intercambian si están en el orden incorrecto.
- C) Se compara cada elemento con el último y se intercambian si son mayores.
- D) Se compara cada elemento con el mínimo y se intercambian si son menores.

3. ¿Cuál de los siguientes algoritmos de ordenación es conocido por su proceso de "burbujeo" donde los elementos más grandes se desplazan hacia el final de la lista?

- A) Selection Sort
- B) Insertion Sort
- C) Bubble Sort
- D) QuickSort

4. ¿Qué característica comparten los algoritmos de Selection Sort, Bubble Sort y Insertion Sort en términos de eficiencia?

- A) Complejidad logarítmica $O(\log n)$
- B) Complejidad lineal $O(n)$
- C) Complejidad cuadrática $O(n^2)$
- D) Complejidad constante $O(1)$

5. En el algoritmo de Insertion Sort, ¿cómo se construye la lista ordenada?

- A) Comparando elementos adyacentes y realizando intercambios.
- B) Seleccionando el elemento mínimo en cada iteración.
- C) Construyendo la lista uno a uno, colocando cada elemento en su posición correcta.
- D) Utilizando una estructura de datos de cola.

2.6. **Algoritmos de búsqueda.**

Los algoritmos de búsqueda son fundamentales en el campo de la programación y la ciencia de la computación. Estos algoritmos se utilizan para encontrar elementos específicos dentro de una colección de datos, como una lista, un arreglo o un grafo, entre otros. Hay varios tipos de algoritmos de búsqueda, cada uno con sus propias características y aplicaciones:

- **Búsqueda Lineal (o Secuencial):** Este algoritmo verifica cada elemento de la colección de datos en orden secuencial hasta encontrar el elemento deseado o hasta que todos los elementos hayan sido revisados. Es simple pero puede ser ineficiente para grandes conjuntos de datos.
- **Búsqueda Binaria:** Este algoritmo solo se puede aplicar a colecciones de datos ordenadas. Divide repetidamente la lista en dos mitades y compara el elemento buscado con el elemento en la mitad de la lista. Si no coincide, se elimina la mitad donde el elemento buscado no puede estar, y la búsqueda continúa en la mitad restante. Es eficiente para grandes conjuntos de datos ordenados.

- **Búsqueda de Profundidad (Depth-First Search - DFS):** Es un algoritmo utilizado para recorrer o buscar en estructuras de datos como grafos o árboles. Comienza en un nodo inicial y explora tanto como sea posible a lo largo de cada rama antes de retroceder. Puede ser implementado recursivamente.
- **Búsqueda de Amplitud (Breadth-First Search - BFS):** Es un algoritmo similar a DFS pero en lugar de explorar en profundidad, explora todos los nodos vecinos al mismo nivel antes de avanzar al siguiente nivel. Se suele implementar utilizando una cola.
- **Algoritmos de Búsqueda Heurística:** Estos algoritmos, como A* (A-star), utilizan información adicional sobre el problema para guiar la búsqueda hacia una solución óptima de manera más eficiente. Utilizan heurísticas para estimar cuánto falta para llegar al objetivo.

Estos son solo algunos ejemplos de algoritmos de búsqueda comunes. La elección del algoritmo adecuado depende del tipo de datos, del problema específico y de los requisitos de eficiencia. Los algoritmos de búsqueda son esenciales en muchas aplicaciones, desde la búsqueda en bases de datos hasta la planificación de rutas en sistemas de navegación.

Autoevaluación 13

1. **¿Qué tipo de algoritmo de búsqueda se utiliza comúnmente para buscar elementos en grandes conjuntos de datos ordenados?**
 - a) Búsqueda Binaria
 - b) Búsqueda Lineal
 - c) Búsqueda de Profundidad
 - d) Búsqueda de Amplitud

2. ¿Qué algoritmo de búsqueda es más eficiente para encontrar una solución óptima en un grafo o árbol?

- a) Búsqueda Lineal
- b) Búsqueda de Amplitud
- c) Búsqueda de Profundidad
- d) Algoritmos de Búsqueda Heurística

3. ¿Cómo se llama el algoritmo de búsqueda que explora todos los nodos vecinos al mismo nivel antes de avanzar al siguiente nivel?

- a) Búsqueda Binaria
- b) Búsqueda de Profundidad
- c) Búsqueda de Amplitud
- d) Búsqueda Lineal

4. ¿Qué tipo de algoritmo de búsqueda utiliza información adicional sobre el problema para guiar la búsqueda hacia una solución óptima de manera más eficiente?

- a) Búsqueda de Profundidad
- b) Búsqueda Binaria
- c) Búsqueda de Amplitud
- d) Algoritmos de Búsqueda Heurística

5. ¿Qué algoritmo de búsqueda es adecuado para buscar en colecciones de datos ordenadas y solo se puede aplicar a estos conjuntos?

- a) Búsqueda de Profundidad
- b) Búsqueda Lineal
- c) Búsqueda Binaria
- d) Búsqueda de Amplitud

ANEXOS

Ejercicio 1

Crea una aplicación que pida un número y calcule su factorial (La factorial de un número es el producto de todos los enteros entre 1 y el propio número y se representa por el número seguido de un signo de exclamación. ¡Por ejemplo $5! = 1 \times 2 \times 3 \times 4 \times 5 = 120$)

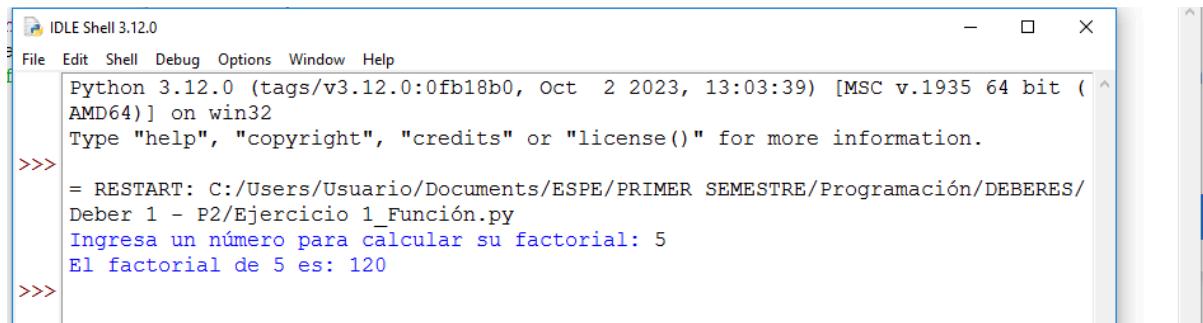
Codificación en Python

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)

def main():
    try:
        num = int(input("Ingresa un número para calcular su factorial: "))
        if num < 0:
            print("Por favor, ingresa un número entero no negativo.")
        else:
            result = factorial(num)
            print(f"El factorial de {num} es: {result}")
    except ValueError:
        print("Por favor, ingresa un número entero válido.")

if __name__ == "__main__":
    main()
```

Ejecución



The screenshot shows the IDLE Shell 3.12.0 interface. The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The main window displays Python version information and a command-line session. The session starts with a restart message, followed by the user's input of '5' and the program's output 'El factorial de 5 es: 120'.

```
IDLE Shell 3.12.0
File Edit Shell Debug Options Window Help
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct  2 2023, 13:03:39) [MSC v.1935 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.

>>> = RESTART: C:/Users/Usuario/Documents/ESPE/PRIMER SEMESTRE/Programación/DEBERES/
Deber 1 - P2/Ejercicio 1_Función.py
Ingrasa un número para calcular su factorial: 5
El factorial de 5 es: 120
```

Ejercicio 2

Crea una aplicación sin funciones que permita adivinar un número. La aplicación genera un número aleatorio del 1 al 100. A continuación, va pidiendo números y va respondiendo si el número a adivinar es mayor o menor que el introducido, además de los intentos que te quedan (tienes 10 intentos para acertarlo). El programa termina cuando se acierta el número (además te dice en cuantos intentos lo has acertado), si se llega al límite de intentos te muestra el número que había generado.

Codificación en python

```
import os
import random

def adivinar_numero():
    numero_a_adivinar = random.randint(1, 100)
    intentos = 10

    print("Bienvenido al juego de adivinar el número!")
    print("Tienes 10 intentos para adivinar un número del 1 al 100.")

    while intentos > 0:
        intento = int(input("Introduce un número: "))
```

```
if intento < numero_a_adivinar:  
    print("El número a adivinar es mayor.")  
  
elif intento > numero_a_adivinar:  
    print("El número a adivinar es menor.")  
  
else:  
    print("¡Felicitaciones! ¡Has adivinado el número en", 11 - intentos, "intentos!")  
    return  
  
  
intentos -= 1  
  
  
print("Lo siento, has agotado tus intentos. El número era:", numero_a_adivinar)  
  
  
adivinar_numero()
```

Ejecución

```
>>> | Introduce un número del 1 al 100: 56  
| El número a adivinar es menor.  
| Introduce un número del 1 al 100: 55  
| El número a adivinar es menor.  
| Introduce un número del 1 al 100: 40  
| El número a adivinar es mayor.  
| Introduce un número del 1 al 100: 45  
| El número a adivinar es menor.  
| Introduce un número del 1 al 100: 44  
| El número a adivinar es menor.  
| Introduce un número del 1 al 100: 41  
| ¡Felicitaciones! ¡Has adivinado el número en 6 intentos!
```

Ejercicio 3

Algoritmo que pida números hasta que se introduzca un cero. Debe imprimir la suma y la media de todos los números introducidos.

Codificación en Python

```
def main():

    numeros = [ ]

    while True:

        numero = int(input("Introduce un número (introduce 0 para salir): "))

        if numero == 0:

            break

        numeros.append(numero)

    if numeros:

        suma = sum(numeros)

        media = suma / len(numeros)

        print(f"La suma de los números es: {suma}")

        print(f"La media de los números es: {media}")

    else:

        print("No se ingresaron números.")

if __name__ == "__main__":
    main()
```

Ejecución

```
===== RESTART: C:/Users/Asus/Desktop/SumaMedia.py =====
Introduce un número (introduce 0 para salir): 5
Introduce un número (introduce 0 para salir): 7
Introduce un número (introduce 0 para salir): 8
Introduce un número (introduce 0 para salir): 9
Introduce un número (introduce 0 para salir): 0
La suma de los números es: 29
La media de los números es: 7.25
```

Ejercicio 4

Realizar un algoritmo que pida números (se pedirá por teclado la cantidad de números a introducir). El programa debe informar de cuantos números introducidos son mayores que 0, menores que 0 e iguales a 0.

Codificación en Python:

```
def contar_numeros():

    cantidad = int(input("Introduce la cantidad de números a ingresar: "))

    numeros = []

    for i in range(cantidad):

        numero = float(input(f"Ingrese el número {i + 1}:"))

        numeros.append(numero)

    return numeros


def contar_ocurrencias(numeros):

    mayores_cero = sum(1 for num in numeros if num > 0)

    menores_cero = sum(1 for num in numeros if num < 0)

    igual_cero = sum(1 for num in numeros if num == 0)

    return mayores_cero, menores_cero, igual_cero


def imprimir_resultados(mayores_cero, menores_cero, igual_cero):

    print("Cantidad de números mayores que 0:", mayores_cero)
    print("Cantidad de números menores que 0:", menores_cero)
    print("Cantidad de números iguales a 0:", igual_cero)


def main():
```

```

numeros = contar_numeros()

mayores_cero, menores_cero, igual_cero = contar_ocurrencias(numeros)

imprimir_resultados(mayores_cero, menores_cero, igual_cero)

if __name__ == "__main__":
    main()

```

Ejecución:

```

File Edit Shell Debug Options Window Help
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct 2 2023, 13:03:39) [MSC v.1935 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> ===== RESTART: C:/Users/PAME ZURITA/Downloads/ejercicio 4.py =====
Introduce la cantidad de números a ingresar: 2
Ingrese el número 1: 1
Ingrese el número 2: 2
Cantidad de números mayores que 0: 2
Cantidad de números menores que 0: 0
Cantidad de números iguales a 0: 0
>>>

```

Ejercicio 5

Algoritmo que pida caracteres e imprima ‘VOCAL’ si son vocales y ‘NO VOCAL’ en caso contrario, el programa termina cuando se introduce un espacio.

Codificación en python:

```

def es_vocal(caracter):

    vocales = "aeiouAEIOU"

    return caracter in vocales

def verificar_vocal():

    while True:

        entrada = input("Introduce un carácter (o presiona Enter para salir): ")

        if entrada == "":

            print("¡Adiós!")

            break

        if len(entrada) == 1:

            if es_vocal(entrada):

```

```

print("VOCAL")

else:

    print("NO VOCAL")

else:

    print("Por favor, ingresa solo un caracter.")

if __name__ == "__main__":
    verificar_vocal()

```

Ejecución:

```

IDLE Shell 3.12.0
File Edit Shell Debug Options Window Help
Python 3.12.0 (tags/v3.12.0-0fb18b0, Oct 2 2023, 13:03:39) [MSC v.1935 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> = RESTART: C:/Users/PAME ZURITA/AppData/Local/Programs/Python/Python312/ejercicio_5.py
Introduce un caracter (o presiona Enter para salir): a
VOCAL
Introduce un caracter (o presiona Enter para salir):
Adiós!
>>>

```

Ejercicio 6

Escribir un programa que imprima todos los números pares entre dos números que se le pida al usuario

Codificación en python:

```

def obtener_numeros():

    while True:

        try:

            num1 = int(input("Ingrese el primer número: "))

            num2 = int(input("Ingrese el segundo número: "))

            return num1, num2

        except ValueError:

            print("Por favor, ingrese números enteros válidos.")

```

```

def imprimir_pares(num1, num2):

    print(f"Números pares entre {num1} y {num2}:")

    for i in range(num1, num2 + 1):

```

```

if i % 2 == 0:
    print(i)

def main():
    num1, num2 = obtener_numeros()
    imprimir_pares(num1, num2)

if __name__ == "__main__":
    main()

```

Ejecución:

```

File Edit Shell Debug Options Window Help
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct 2 2023, 13:03:39) [MSC v.1935 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> = RESTART: C:/Users/PAME ZURITA/AppData/Local/Programs/Python/Python312/ejercicio_6.py
Ingrese el primer número: 1
Ingrese el segundo número: 2
Números pares entre 1 y 2:
2
>>>

```

Ejercicio 7

Realizar un algoritmo que muestre la tabla de multiplicar de un número introducido por teclado.

Codificación

```

def imprimir_tabla_multiplicar(tabla):
    for i in range(1, 13):
        resultado = tabla * i
        print(f"{tabla} x {i} = {resultado}")

tabla = int(input("Ingrese la tabla: "))

imprimir_tabla_multiplicar(tabla)

```

Ejecución

```
Ingrese la tabla: 6
6 x 1 = 6
6 x 2 = 12
6 x 3 = 18
6 x 4 = 24
6 x 5 = 30
6 x 6 = 36
6 x 7 = 42
6 x 8 = 48
6 x 9 = 54
6 x 10 = 60
6 x 11 = 66
6 x 12 = 72
```

Ejercicio 8

Escribe un programa que pida el límite inferior y superior de un intervalo. Si el límite inferior es mayor que el superior lo tiene que volver a pedir. A continuación, se van introduciendo números hasta que introducimos el 0. Cuando termine el programa dará las siguientes informaciones:

- La suma de los números que están dentro del intervalo (intervalo abierto).
- Cuántos números están fuera del intervalo.
- Informa si hemos introducido algún número igual a los límites del intervalo.

Codificación

```
def obtener_limites():

    lim_inf = 0

    lim_sup = 0

    while lim_inf >= lim_sup:

        lim_inf = int(input("Introduce el límite inferior del intervalo: "))

        lim_sup = int(input("Introduce el límite superior del intervalo: "))

        if lim_inf >= lim_sup:

            print("ERROR: El límite inferior debe ser menor que el superior.")

    return lim_inf, lim_sup
```

```
def procesar_numeros(lim_inf, lim_sup):
    suma_dentro_intervalo = 0
    cont_fuera_intervalo = 0
    igual_limites = False
    num = 1

    while num != 0:
        num = int(input("Introduce un número (0 para salir): "))

        if num > lim_inf and num < lim_sup:
            suma_dentro_intervalo += num
        else:
            cont_fuera_intervalo += 1

        if num == lim_inf or num == lim_sup:
            igual_limites = True

    return suma_dentro_intervalo, cont_fuera_intervalo, igual_limites

def main():
    lim_inf, lim_sup = obtener_limites()
    suma_dentro_intervalo, cont_fuera_intervalo, igual_limites =
    procesar_numeros(lim_inf, lim_sup)

    print("La suma de los números dentro del intervalo es", suma_dentro_intervalo)
    print("La cantidad de números fuera del intervalo es", cont_fuera_intervalo)

    if igual_limites:
        print("Se ha introducido algún número igual a los límites del intervalo.")
```

```

else:

    print("No se ha introducido ningún número igual a los límites del intervalo.")

if __name__ == "__main__":
    main()

```

Ejecución

```

=====
RESTART: C:/Users/SERVER/Downloads/Pregunta 8.py =====
Introduce el límite inferior del intervalo: 34
Introduce el límite superior del intervalo: 42
Introduce un número (0 para salir): 3
Introduce un número (0 para salir): 0
La suma de los números dentro del intervalo es 0
La cantidad de números fuera del intervalo es 2
No se ha introducido ningún número igual a los límites del intervalo.

```

Ejercicio 9

Escribe un programa que, dados dos números, uno real (base) y un entero positivo (exponente), saque por pantalla el resultado de la potencia. No se puede utilizar el operador de potencia.

Codificación

```

def obtener_datos_potencia():

    base = float(input("Ingrese la base de la potencia: "))

    exponente = -1

    while exponente < 0:

        exponente = int(input("Ingrese el exponente de la potencia: "))

        if exponente < 0:

            print("ERROR: El exponente debe ser positivo")

    return base, exponente

def calcular_potencia(base, exponente):

    potencia = 1

    for _ in range(exponente):

        potencia *= base

```

```
        return potencia

def main():

    base, exponente = obtener_datos_potencia()

    resultado_potencia = calcular_potencia(base, exponente)

    print(f"La potencia de {base} elevado a {exponente} es: {resultado_potencia}")

if __name__ == "__main__":
    main()
```

Ejecución

```
= RESTART: C:/Users/SERVER/Downloads/Pregunta 9con funciones.py
Ingrese la base de la potencia: 3
Ingrese el exponente de la potencia (debe ser positivo): 2
La potencia de 3.0 elevado a 2 es: 9.0
```

Ejercicio 10

Algoritmo que muestre la tabla de multiplicar de los números 1,2,3,4 y 5.

Codificación

```
print ("Las tablas de 1 hasta 5 son: ")

def tablas ():
    a = int(input())
    for i in range (1, 11, 1):
        b = a * i
        print (a, " * ", i, " = ", b)

tablas ()
```

Compilación

Las tablas de 1 hasta 5 son:

1	2	3	4	5
1 * 1 = 1	2 * 1 = 2	3 * 1 = 3	4 * 1 = 4	5 * 1 = 5
1 * 2 = 2	2 * 2 = 4	3 * 2 = 6	4 * 2 = 8	5 * 2 = 10
1 * 3 = 3	2 * 3 = 6	3 * 3 = 9	4 * 3 = 12	5 * 3 = 15
1 * 4 = 4	2 * 4 = 8	3 * 4 = 12	4 * 4 = 16	5 * 4 = 20
1 * 5 = 5	2 * 5 = 10	3 * 5 = 15	4 * 5 = 20	5 * 5 = 25
1 * 6 = 6	2 * 6 = 12	3 * 6 = 18	4 * 6 = 24	5 * 6 = 30
1 * 7 = 7	2 * 7 = 14	3 * 7 = 21	4 * 7 = 28	5 * 7 = 35
1 * 8 = 8	2 * 8 = 16	3 * 8 = 24	4 * 8 = 32	5 * 8 = 40
1 * 9 = 9	2 * 9 = 18	3 * 9 = 27	4 * 9 = 36	5 * 9 = 45
1 * 10 = 10	2 * 10 = 20	3 * 10 = 30	4 * 10 = 40	5 * 10 = 50

Ejercicio 11

Escribe un programa que diga si un número introducido por teclado es o no primo.

Un número primo es aquel que sólo es divisible entre él mismo y la unidad. Nota: Es suficiente probar hasta la raíz cuadrada del número para ver si es divisible por algún otro número.

Codificación en python:

```
#Ejercicio 11 (Funciones)
```

```
def es_primo(n):
    if n < 2:
        return False
    for i in range(2, int(n**0.5) + 1):
```

```

if n % i == 0:

    return False

return True

n = int(input("Ingresa un número:"))

if es_primo(n):

    print("Es un número primo")

else:

    print("No es un número primo")

```

Ejecución:

```

Ingresá un número:2
Es un número primo
PS C:\Users\usuario\OneDrive\Escritorio\Programación\Tarea programación>
gramación\Tarea programación'; & 'c:\Users\usuario\AppData\Local\Microso
vscode\extensions\ms-python.debugpy-2024.0.0-win32-x64\bundled\libs\debu
C:\Users\usuario\OneDrive\Escritorio\Programación\Tarea programación\#Ej
Ingresá un número:3
Es un número primo
PS C:\Users\usuario\OneDrive\Escritorio\Programación\Tarea programación>
7
PS C:\Users\usuario\OneDrive\Escritorio\Programación\Tarea programación>
gramación\Tarea programación'; & 'c:\Users\usuario\AppData\Local\Microso
vscode\extensions\ms-python.debugpy-2024.0.0-win32-x64\bundled\libs\debu
C:\Users\usuario\OneDrive\Escritorio\Programación\Tarea programación\#Ej
Ingresá un número:9
No es un número primo

```

Ejercicio 12

Realizar un algoritmo para determinar cuánto ahorrará una persona en un año, si al final de cada mes deposita cantidades variables de dinero; además, se quiere saber cuánto lleva ahorrado cada mes.

Codificación en python:

```

def calcular_ahorros():

    ahorros_totales = 0

    for mes in range(1, 13):

```

```

deposito=int(input("Ingrese la cantidad depositada para el mes "+ str(mes) + ":"))

ahorros_totales += deposito

print("Llevas ahorrado un total de: " + str(ahorros_totales))

print("Al final del año, habrás ahorrado un total de: " + str(ahorros_totales))

calcular_ahorros()

```

Ejecución del código:

```

Ingrese la cantidad depositada para el mes 1: 120
Llevas ahorrado un total de: 120
Ingrese la cantidad depositada para el mes 2: 120
Llevas ahorrado un total de: 240
Ingrese la cantidad depositada para el mes 3: 120
Llevas ahorrado un total de: 360
Ingrese la cantidad depositada para el mes 4: 120
Llevas ahorrado un total de: 480
Ingrese la cantidad depositada para el mes 5: 120
Llevas ahorrado un total de: 600
Ingrese la cantidad depositada para el mes 6: 120
Llevas ahorrado un total de: 720
Ingrese la cantidad depositada para el mes 7: 120
Llevas ahorrado un total de: 840
Ingrese la cantidad depositada para el mes 8: 120
Llevas ahorrado un total de: 960
Ingrese la cantidad depositada para el mes 9: 120
Llevas ahorrado un total de: 1080
Ingrese la cantidad depositada para el mes 10: 120
Llevas ahorrado un total de: 1200
Ingrese la cantidad depositada para el mes 11: 120
Llevas ahorrado un total de: 1320
Ingrese la cantidad depositada para el mes 12: 120
Llevas ahorrado un total de: 1440
Al final del año, habrás ahorrado un total de: 1440

```

Ejercicio 13

Una empresa tiene el registro de las horas que trabaja diariamente un empleado durante la semana (seis días) y requiere determinar el total de éstas, así como el sueldo que recibirá por las horas trabajadas.

Codificación en python

```

def ingresar_horas_trabajadas(dias):

    horas_trabajadas = []

    for i in range(dias):

        horas = float(input(f"Ingrese las horas trabajadas el día {i+1}:"))

        horas_trabajadas.append(horas)

```

```

    return horas_trabajadas

def calcular_total_horas(horas_trabajadas):
    return sum(horas_trabajadas)

def calcular_sueldo(total_horas, salario_por_hora):
    return total_horas * salario_por_hora

def main():
    dias_trabajados = 6

    horas_trabajadas = ingresar_horas_trabajadas(dias_trabajados)

    salario_por_hora = 10 # Ejemplo: $10 por hora

    total_horas_trabajadas = calcular_total_horas(horas_trabajadas)

    sueldo = calcular_sueldo(total_horas_trabajadas, salario_por_hora)

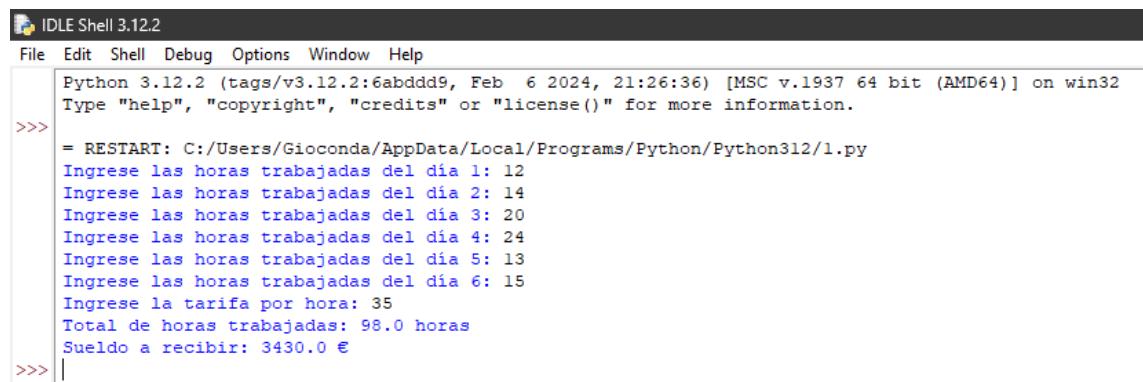
    print(f"Total de horas trabajadas durante la semana: {total_horas_trabajadas}")

    print(f"Sueldo correspondiente: ${sueldo}")

if __name__ == "__main__":
    main()

```

Ejecución del código



The screenshot shows the Python IDLE Shell 3.12.2 interface. The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The shell window displays the following output:

```

Python 3.12.2 (tags/v3.12.2:6abddd9, Feb  6 2024, 21:26:36) [MSC v.1937 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.

>>> = RESTART: C:/Users/Gioconda/AppData/Local/Programs/Python/Python312/1.py
Ingrese las horas trabajadas del dia 1: 12
Ingrese las horas trabajadas del dia 2: 14
Ingrese las horas trabajadas del dia 3: 20
Ingrese las horas trabajadas del dia 4: 24
Ingrese las horas trabajadas del dia 5: 13
Ingrese las horas trabajadas del dia 6: 15
Ingrese la tarifa por hora: 35
Total de horas trabajadas: 98.0 horas
Sueldo a recibir: 3430.0 €
>>> |

```

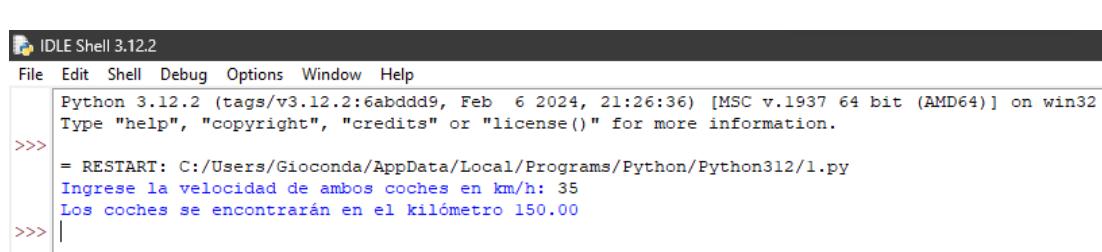
Ejercicio 14

Una persona se encuentra en el kilómetro 70 de una carretera, otra se encuentra en el km 150, los coches tienen sentido opuesto y tienen la misma velocidad. Realizar un programa para determinar en qué kilómetro de esa carretera se encontrarán.

Codificación en python.

```
def calcular_punto_encuentro(km1, km2):  
    distancia_entre_personas = abs(km1 - km2)  
    return max(km1, km2) - distancia_entre_personas / 2  
  
def main():  
    # Posiciones iniciales de las personas en la carretera  
    km_persona1 = 70  
    km_persona2 = 150  
    punto_encuentro = calcular_punto_encuentro(km_persona1, km_persona2)  
    print(f"Las personas se encontrarán en el kilómetro {punto_encuentro}.")  
  
if __name__ == "__main__":  
    main()
```

ejecución



The screenshot shows the Python IDLE Shell interface. The title bar reads "IDLE Shell 3.12.2". The menu bar includes "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The main window displays the following text:
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb 6 2024, 21:26:36) [MSC v.1937 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> = RESTART: C:/Users/Gioconda/AppData/Local/Programs/Python/Python312/1.py
Ingrese la velocidad de ambos coches en km/h: 35
Los coches se encontrarán en el kilómetro 150.00
>>> |

Ejercicio 15

Una persona adquirió un producto para pagar en 20 meses. El primer mes pagó 10 €, el segundo 20 €, el tercero 40 € y así sucesivamente. Realizar un algoritmo para determinar cuánto debe pagar mensualmente y el total de lo que pagó después de los 20 meses.

Código en python.

```
def calcular_pago_mensual(meses):

    pago_mensual = 0

    for mes in range(meses):

        pago_mensual += 10 * (2 ** mes)

    return pago_mensual


def calcular_total_pagado(meses):

    total_pagado = 0

    for mes in range(meses):

        total_pagado += 10 * (2 ** mes)

    return total_pagado


def main():

    meses = 20


    pago_mensual = calcular_pago_mensual(meses)

    total_pagado = calcular_total_pagado(meses)

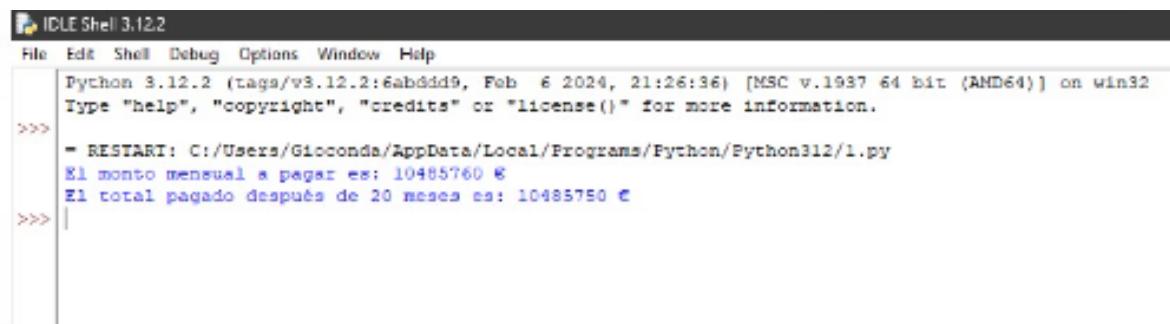

    print(f"El pago mensual es de {pago_mensual} €.")

    print(f"El total pagado después de {meses} meses es de {total_pagado} €.")


if __name__ == "__main__":

    main()
```

Ejecución



The screenshot shows the Python IDLE Shell interface. The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The shell window displays the following text:

```
File Edit Shell Debug Options Window Help
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb 6 2024, 21:26:36) [MSC v.1937 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.

>>> = RESTART: C:/Users/Gioconda/AppData/Local/Programs/Python/Python312/1.py
El monto mensual a pagar es: 10485760 €
El total pagado después de 20 meses es: 10485750 €
>>>
```

Ejercicio 16

Una empresa les paga a sus empleados con base en las horas trabajadas en la semana. Realice un algoritmo para determinar el sueldo semanal de N trabajadores y, además, calcule cuánto pagó la empresa por los N empleados.

Código y ejecución:

```
def calcular_sueldo(horas, tarifa):
    return horas * tarifa

def main():
    n = int(input("Número de empleados: "))
    total_pago = 0

    for i in range(1, n + 1):
        horas = float(input(f"Horas trabajadas por empleado {i}: "))
        tarifa = float(input(f"Tarifa por hora para empleado {i}: "))

        sueldo = calcular_sueldo(horas, tarifa)
        total_pago += sueldo

    print(f"Sueldo de empleado {i}: ${sueldo}")
```

```
print(f"\nPago total de la empresa por {n} empleados: ${total_pago}")

main()
```

```
File Edit Shell Debug Options Window Help
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct 2 2023, 13:03:39) [MSC v.1935 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.

>>> = RESTART: C:/Users/Usuario/OneDrive/Documentos/Ejercicios PYTHON/16 FUNCIONES.py
Número de empleados: 3
Horas trabajadas por empleado 1: 1
Tarifa por hora para empleado 1: 2
Sueldo de empleado 1: $2.0
Horas trabajadas por empleado 2: 3
Tarifa por hora para empleado 2: 32
Sueldo de empleado 2: $96.0
Horas trabajadas por empleado 3: 23
Tarifa por hora para empleado 3: 3
Sueldo de empleado 3: $69.0

Pago total de la empresa por 3 empleados: $167.0
>>>
```

Ejercicio 17

Una empresa les paga a sus empleados con base en las horas trabajadas en la semana. Para esto, se registran los días que trabajó y las horas de cada día. Realice un algoritmo para determinar el sueldo semanal de N trabajadores y además calcule cuánto pagó la empresa por los N empleados.

Código

```
return horas_trabajadas * tarifa_por_hora

def calcular_sueldo_total(lista_sueldos):
    return sum(lista_sueldos)

n = int(input("Ingrese el número de empleados: "))
sueldos = []

for i in range(n):
    horas_trabajadas = float(input(f"Ingrese las horas trabajadas {i+1}: "))
```

```

tarifa_por_hora = float(input(f"Ingrese la tarifa por hora {i+1}: "))

sueldo_semanal = calcular_sueldo_semanal(horas_trabajadas, tarifa_por_hora)
sueldos.append(sueldo_semanal)

sueldo_total = calcular_sueldo_total(sueldos)

print("\n--- Resultados ---")

for i, sueldo in enumerate(sueldos):
    print(f"El sueldo semanal del empleado {i+1} es ${sueldo:.2f}")

print(f"\nLa empresa pagó un total de ${sueldo_total:.2f} a {n} empleados.")

```

Ejecución

```

= RESTART: C:/Users/alemo/AppData/Local/Programs/Python/Python312/ejercicios python/Ejercicios deber grupal/ejercicio 17 manual.py
Ingrese el número de empleados: 3
Ingrese las horas trabajadas 1: 5
Ingrese la tarifa por hora 1: 7
Ingrese las horas trabajadas 2: 6
Ingrese la tarifa por hora 2: 7
Ingrese las horas trabajadas 3: 8
Ingrese la tarifa por hora 3: 7

--- Resultados ---
El sueldo semanal del empleado 1 es $35.00
El sueldo semanal del empleado 2 es $42.00
El sueldo semanal del empleado 3 es $56.00

La empresa pagó un total de $133.00 a 3 empleados.
>|

```

Ejercicio 18

Hacer un programa que muestre un cronómetro, indicando las horas, minutos y segundos.

Código y ejecución:

```
import time

def cronometro():

    segundos=0
    minutos=0
    horas = 0

    while True:

        print(f"{horas:02d}:{minutos:02d}:{segundos:02d}", end="\r")
        time.sleep(1)
        segundos= segundos+1

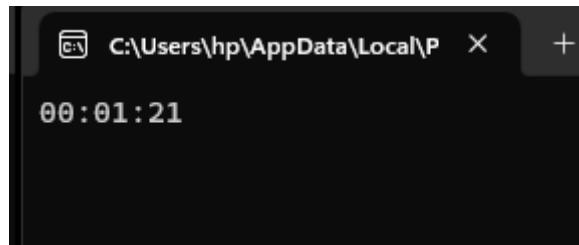
        if segundos == 60:
            segundos=0
            minutos=minutos+ 1

        if minutos == 60:
            minutos=0
            horas= horas+1

    try:
        cronometro()
    except KeyboardInterrupt:
        print("\nCronómetro detenido.")
```

```
( Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct 2 2023, 13:03:39) [MSC v.1935 64 bit
AMD64]) on win32
Type "help", "copyright", "credits" or "license()" for more information.

= RESTART: C:\Users\hp\Desktop\Ej..py
00:00:0000:00:0100:00:0200:00:0300:00:0400:00:0500:00:0600:00:0700:00:08
00:00:0900:00:1000:00:1100:00:1200:00:1300:00:1400:00:1500:00:1600:00:1
700:00:1800:00:19    00:00:20 |
```



Ejercicio 19

Realizar un ejemplo de menú, donde podemos escoger las distintas opciones hasta que seleccionamos la opción de “Salir”.

Código y ejecución:

```
def clear_screen():

    print("\n" * 50)

def display_menu():

    print("Menú:")

    print("a. Opción A")

    print("b. Opción B")

    print("c. Opción C")

    print("d. Opción D")

    print("e. Opción E")
```

```
print("e. Salir")

def process_option(option):

    if option.lower() == 'a':
        print("Ha seleccionado la Opción A")

    elif option.lower() == 'b':
        print("Ha seleccionado la Opción B")

    elif option.lower() == 'c':
        print("Ha seleccionado la Opción C")

    elif option.lower() == 'd':
        print("Ha seleccionado la Opción D")

    elif option.lower() == 'e':
        print("Ha seleccionado la Opción E")

    elif option.lower() == 'e':
        print("Saliendo del programa...")

    else:
        print("Opción no válida. Inténtelo de nuevo.")

def main():
```

```

opcion = ""

while opcion.lower() != 'e':

    clear_screen()

    display_menu()

    opcion = input("Seleccione una opción: ")

    process_option(opcion)

if __name__ == "__main__":

```

```

*IDLE Shell 3.12.1*
File Edit Shell Debug Options Window Help
Python 3.12.1 (tags/v3.12.1:2305ca5, Dec  7 2023, 22:03:25) [MSC v.1937 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/Pimpu/Documents/ejercicioshhfdhfd/ejercicio19.py
Squeezed text (50 lines).

Menú:
a. Opción A
b. Opción B
c. Opción C
d. Opción D
e. Opción E
e. Salir
Seleccione una opción: a
Ha seleccionado la Opción A
Squeezed text (50 lines).

Menú:
a. Opción A
b. Opción B
c. Opción C
d. Opción D
e. Opción E
e. Salir
Seleccione una opción: b
Ha seleccionado la Opción B

```

Ejercicio 20

Mostrar en pantalla los N primeros números primos. Se pide por teclado la cantidad de números primos que queremos mostrar.

Codificación y ejecución:

```

def es_primo(num):

    if num <= 1:

        return False

```

```

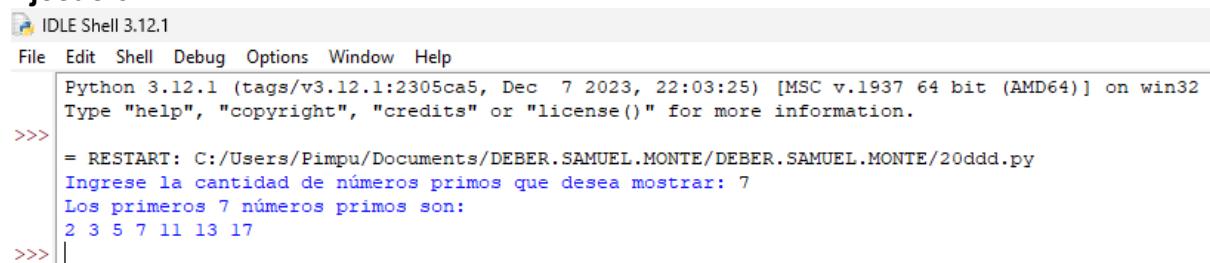
for i in range(2, int(num**0.5) + 1):
    if num % i == 0:
        return False
return True

def mostrar_primos(cantidad):
    primos_encontrados = 0
    numero = 2
    while primos_encontrados < cantidad:
        if es_primo(numero):
            print(numero, end=" ")
            primos_encontrados += 1
        numero += 1

try:
    cantidad = int(input("Ingrese la cantidad de números primos que desea mostrar: "))
    if cantidad <= 0:
        print("Por favor, ingrese un número entero positivo.")
    else:
        print(f"Los primeros {cantidad} números primos son:")
        mostrar_primos(cantidad)
except ValueError:
    print("Por favor, ingrese un número entero válido.")

```

Ejecución



The screenshot shows the Python IDLE Shell interface. The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The shell window displays the following output:

```

Python 3.12.1 (tags/v3.12.1:2305ca5, Dec  7 2023, 22:03:25) [MSC v.1937 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/Pimpu/Documents/DEBER.SAMUEL.MONTE/DEBER.SAMUEL.MONTE/20ddd.py
Ingrese la cantidad de números primos que desea mostrar: 7
Los primeros 7 números primos son:
2 3 5 7 11 13 17
>>> |

```



Programación Estructurada

MANUAL

Unidad 3

UNIVERSIDAD DE LAS FUERZAS ARMADAS ESPE

NRC: 17832

Introducción a la Programación

Ing. Edgar Solis

07 de Marzo del 2024

Unidad 3

Cadenas y/o Strings

1. Concepto

En programación, una "cadena" (también conocida como "string" en inglés) es un tipo de dato que representa una secuencia de caracteres. Los caracteres pueden incluir letras, números, espacios en blanco, símbolos y otros caracteres especiales. Las cadenas son utilizadas para almacenar y manipular texto en programas de computadora.

Las cadenas suelen estar delimitadas por comillas simples (' '), comillas dobles (" "), o en algunos lenguajes de programación, comillas invertidas (`).

Por ejemplo:

- 'Hola, mundo!'
- "Ejemplo de cadena"

Estos son algunos ejemplos de cadena con comillas invertidas

En la mayoría de los lenguajes de programación, las cadenas son consideradas como tipos de datos primitivos o nativos, y ofrecen una serie de funciones y métodos para realizar operaciones comunes, como la concatenación (unión de dos o más cadenas), la extracción de subcadenas, la búsqueda de caracteres, entre otras.

Características de las cadenas

- **Secuencia de caracteres:** Una cadena es una secuencia ordenada de caracteres, que pueden incluir letras, números, espacios en blanco, y otros símbolos.
- **Índices y acceso:** Los caracteres dentro de una cadena pueden ser accedidos individualmente mediante índices. En muchos lenguajes de programación, los

índices comienzan desde 0. Por ejemplo, el primer carácter de una cadena tiene el índice 0, el segundo tiene el índice 1, y así sucesivamente.

- **Operaciones de manipulación:** Las cadenas ofrecen una variedad de operaciones para manipularlas, como concatenación (unir dos cadenas), subdivisión (dividir una cadena en subcadenas más pequeñas), búsqueda (buscar subcadenas o caracteres específicos), reemplazo (sustituir partes de una cadena por otras), entre otras.

Autoevaluación 1

1. ¿Qué es una cadena en programación?

- a. Una secuencia de números
- b. Una secuencia de caracteres
- c. Una secuencia de palabras
- d. Una secuencia de operaciones

2. ¿Cuál de las siguientes afirmaciones sobre las cadenas es verdadera?

- a. Las cadenas son mutables, lo que significa que se pueden modificar directamente.
- b. Las cadenas son inmutables, lo que significa que no se pueden modificar directamente.
- c. Las cadenas solo pueden contener letras y números.
- d. Las cadenas son tipos de datos numéricos.

3. ¿Cómo se delimitan las cadenas en muchos lenguajes de programación?

- a. Con corchetes []
- b. Con paréntesis ()
- c. Con comillas simples ''
- d. Con llaves { }

4. ¿Qué operación se utiliza para unir dos cadenas en una sola?

- a. Separación
- b. Unificación
- c. Concatenación
- d. Agregación

5. ¿Cómo se accede al primer carácter de una cadena en la mayoría de los lenguajes de programación?

- a. Con el índice 0
- b. Con el índice 1
- c. Con el índice -1
- d. No se puede acceder al primer carácter

2. Declaración e inicialización de variables

La declaración e inicialización de variables es un concepto fundamental en programación. Al declarar una variable, le estamos indicando al programa que reserve un espacio en la memoria para almacenar un valor. La inicialización, por otro lado, consiste en asignar un valor inicial a la variable.

Al declarar una variable, debemos especificar su tipo y darle un nombre único y significativo. Los tipos de variables pueden ser enteros, cadenas de texto, booleanos, entre otros.

La inicialización de una variable se realiza asignándole un valor mediante el signo igual (=). Esto permite que la variable tenga un valor inicial con el cual trabajar.

Ejemplos

1. Ejemplo de declaración e inicialización de una variable entera:

En java

```
int edad = 25;
```

2. Ejemplo de declaración e inicialización de una variable de tipo cadena de texto:

En python

```
nombre = "Juan";
```

3. Ejemplo de declaración e inicialización de una variable booleana:

En javascript

```
var esMayorDeEdad = true;
```

4. Ejemplo de declaración e inicialización de una variable de tipo flotante:

En plaintext

```
float precio = 9.99;
```

Autoevaluación 2

1. ¿Qué es la declaración de variables?

- a) Reservar espacio en la memoria para almacenar un valor.
- b) Asignar un valor inicial a la variable.
- c) Especificar el tipo y el nombre de la variable.
- d) Ninguna de las anteriores.

2. ¿Qué es la inicialización de variables?

- a) Reservar espacio en la memoria para almacenar un valor.
- b) Asignar un valor inicial a la variable.
- c) Especificar el tipo y el nombre de la variable.
- d) Ninguna de las anteriores.

3. ¿Cuál es el signo utilizado para la inicialización de variables?

- a) +
- b) -
- c) *
- d) =

4. ¿Cuál es el objetivo de la declaración e inicialización de variables?

- a) Crear y asignar valores a variables en un programa.
- b) Reservar espacio en la memoria para almacenar un valor.
- c) Asignar un valor inicial a la variable.
- d) Especificar el tipo y el nombre de la variable.

5. ¿Por qué es importante asignar un nombre único y significativo a una variable?

- a) Para que ocupe menos espacio en la memoria.
- b) Para evitar conflictos y hacer el código más legible.
- c) Para que el programa se ejecute más rápido.
- d) Para que la variable pueda cambiar de tipo dinámicamente.

3. Entrada/Salida

La manipulación de cadenas de texto (strings) es una tarea común y esencial. Este lenguaje de programación proporciona una serie de métodos integrados para la entrada y salida de strings.

Entrada de Strings: La entrada de strings en Python se realiza principalmente a través de la función `input()`. Esta función captura la entrada del usuario como una cadena de texto. Por ejemplo:

```
texto = input("Introduce un texto: ")  
print("El texto introducido es: ", texto)
```

Salida de Strings: La salida de strings en Python se realiza principalmente a través de la función `print()`. Esta función imprime la cadena de texto en la consola. Por ejemplo:

```
texto = "Hola, Mundo!"  
print(texto)
```

Python trata las cadenas de texto como secuencias de caracteres, lo que permite acceder a los elementos individuales de la cadena mediante su índice.

Autoevaluación 3

1. ¿Qué función se utiliza para capturar la entrada del usuario en Python?

- a) `output()`
- b) `enter()`
- c) `input()`
- d) `capture()`

2. ¿Cómo se imprime una cadena de texto en la consola en Python?

- a) `console("Hola Mundo!")`
- b) `echo("Hola Mundo!")`
- c) `print("Hola Mundo!")`
- d) `output("Hola Mundo!")`

3. ¿Cómo se accede al primer carácter de una cadena de texto en Python?

- a) texto[1]
- b) texto[0]
- c) texto[-1]
- d) texto[2]

4. ¿Cómo se accede al último carácter de una cadena de texto en Python?

- a) texto[1]
- b) texto[0]
- c) texto[-1]
- d) texto[2]

5. ¿Qué tipo de dato devuelve la función input() en Python?

- a) int
- b) float
- c) str
- d) bool

4. Asignación

La asignación es un concepto central en la programación que implica almacenar un valor en una variable. En términos más simples, es como colocar un objeto en una caja con etiqueta para poder referirse a él más tarde. Este proceso es esencial en la manipulación y gestión de datos en cualquier programa.

En la mayoría de los lenguajes de programación, la asignación se realiza utilizando el operador de asignación "=" o su equivalente en el lenguaje específico. Por ejemplo, en Python, se utiliza el operador "=" para asignar un valor a una variable, como en `x = 10`. Esto asigna el valor 10 a la variable x.

La asignación puede involucrar diferentes tipos de datos, como números enteros, números de punto flotante, cadenas de texto, booleanos, listas, diccionarios, y otros tipos de datos más complejos dependiendo del lenguaje de programación utilizado.

Una vez que se asigna un valor a una variable, ese valor se puede utilizar en el programa. Por ejemplo, si más tarde necesitas utilizar el valor almacenado en la variable `x`, simplemente puedes hacer referencia a ella en tu código.

Es importante tener en cuenta que la asignación no es lo mismo que la igualdad matemática. En programación, el signo `"+"` no representa igualdad matemática, sino que indica una operación de asignación donde el valor a la derecha del operador se asigna a la variable a la izquierda del operador.

La asignación también puede implicar operaciones más complejas, como la asignación de expresiones aritméticas o la asignación de valores basados en condiciones lógicas. Por ejemplo, en muchos lenguajes de programación, puedes realizar asignaciones condicionales utilizando el operador ternario, como en `x = (condicion) ? valor_verdadero : valor_falso`.

Ejemplo:

```
# Asignación de valores a variables  
x = 10  
y = "Hola, mundo!"  
z = True  
  
# Asignación de valores basados en una expresión aritmética  
a = 5  
b = 3  
suma = a + b
```

```
# Asignación condicional utilizando el operador ternario  
condicion = True  
  
resultado = "Verdadero" if condicion else "Falso"  
  
# Imprimir los valores asignados  
  
print("x es:", x)  
  
print("y es:", y)  
  
print("z es:", z)  
  
print("La suma de a y b es:", suma)  
  
print("El resultado es:", resultado)
```

Autoevaluación 4

1. ¿Qué es la asignación en programación?

- a) Almacenar un valor en una variable
- b) Realizar operaciones aritméticas
- c) Comparar dos valores
- d) Eliminar una variable

2. ¿Qué operador se utiliza comúnmente para realizar asignaciones en muchos lenguajes de programación?

- a) "+"
- b) "="
- c) "=="
- d) "->"

3. ¿Cuál de los siguientes NO es un tipo de dato que puede ser asignado a una variable?

- a) Números enteros
- b) Funciones
- c) Listas
- d) Cadenas de texto

4. ¿Qué representa el signo "=" en programación en el contexto de asignación?

- a) Igualdad matemática
- b) Asignación de valores
- c) Comparación de valores
- d) Ninguna de las anteriores

5. ¿Cuál es una forma común de realizar asignaciones condicionales en muchos lenguajes de programación?

- a) Utilizando la palabra clave "if"
- b) Utilizando la palabra clave "else"
- c) Utilizando el operador ternario
- d) Utilizando la palabra clave "for"

5. Longitud y concatenación

La longitud se refiere al número preciso de caracteres que contiene una cadena o strin. Dichos caracteres pueden ser letras, números o símbolos. Es una medida que proporciona información sobre su tamaño o extensión en término de caracteres.

En Python se puede obtener la longitud de cadena utilizando la función 'len()', la cual devuelve un número entero que representa la cantidad caracteres en la cadena

Ejemplo:

```
cadena = "El cielo está nublado"

longitud = len(cadena)

print( "La longitud de la cadena es:", longitud)

# Salida: 21
```

La concatenación es una operación que implica combinar dos o más cadenas para formar una cadena más extensa, construir mensajes o textos complejos , A parte de combinar cadenas, también puede introducir nuevos caracteres como espacios o comas.

En Python se puede concatenar cadenas con el operador ‘+’ o con el método ‘join()’

Ejemplo:

- Con el operador ‘+’

```
cadena1 = "¡Hola"
```

```
cadena2 = "mundo!"
```

```
concatenacion = cadena1 + " " + cadena2
```

```
print(concatenacion)
```

```
# Salida: ¡Hola mundo!
```

- Con el método ‘join’

```
palabras = ["Esto", "es", "una", "frase"]
```

```
frase = " ".join(palabras)
```

```
print(frase)
```

```
# Salida: Esto es una frase
```

Autoevaluación 5

1. ¿Qué es la longitud de una cadena en Python?

- a) La cantidad de espacios que ocupa la cadena en memoria
- b) El número exacto de caracteres que contiene la cadena

c) El número de palabras dentro de la cadena

d) La posición del último carácter en la cadena

2. ¿Cómo se puede obtener la longitud de una cadena en Python?

a) Utilizando la función length()

b) Con el método count()

c) Mediante la función size()

d) Utilizando la función len()

3. ¿Cuál es la salida del siguiente código en Python?

```
cadena = "¡Hola"
```

```
cadena += " mundo!"
```

```
print(cadena)
```

a) ¡Hola mundo!

b) ¡Hola+mando!

c) ¡Hola , mundo!

d) Hola mundo!

4. ¿Cuál es la forma preferida y más eficiente de concatenar las palabras de una lista en Python?

a) Utilizando el método concat()

b) Utilizando el método append()

c) Utilizando el método join()

d) Utilizando el método extend()

5. ¿Cuál es la función de la función join() en Python?

a) Dividir una cadena en una lista de palabras

b) Concatenar cadenas de manera efectiva

c) Contar el número de caracteres en una cadena

d) Dividir una cadena en subcadenas basadas en un delimitador

6. Comparación

En Python, las cadenas de texto, o strings, son un tipo de dato que se encuentra en todas partes. Estas cadenas pueden ser creadas utilizando comillas simples o dobles y manipuladas de diversas maneras. Una de las operaciones más habituales que se realizan con las cadenas es la comparación

La comparación de cadenas en Python se realiza utilizando varios operadores. El operador == se utiliza para verificar si dos cadenas son idénticas. Si lo son, devuelve True; si no, devuelve False. Aquí se muestra un ejemplo:

```
nombre = "Ana"  
nombre == "Ana" # Esto devolverá: True
```

Por otro lado, si se quiere verificar si dos cadenas son diferentes, se puede utilizar el operador !=. Este operador devuelve True si las cadenas son diferentes y False si son iguales.

```
color = "azul"  
color != "verde" # Esto devolverá: True
```

Es importante mencionar que estos operadores también consideran la longitud de las cadenas. Una cadena se considera mayor que otra si tiene más caracteres. Además,

distinguen entre mayúsculas y minúsculas, considerando que las mayúsculas son menores que las minúsculas en ASCII.

Autoevaluación 6

1. ¿Cuál de las siguientes opciones compara correctamente dos cadenas en Python?

- a) cadena1 == cadena2
- b) cadena1.equals(cadena2)
- c) cadena1.compare(cadena2)
- d) cadena1.isEqual(cadena2)

2. ¿Cómo se puede verificar si una cadena es un subconjunto de otra cadena en Python?

- a) cadena1.contains(cadena2)
- b) cadena1.isSubset(cadena2)
- c) cadena1 in cadena2
- d) cadena1.subset(cadena2)

3. ¿Cuál de las siguientes opciones devuelve True si la cadena1 es lexicográficamente mayor que la cadena2?

- a) cadena1 > cadena2
- b) cadena1 >= cadena2
- c) cadena1 < cadena2
- d) cadena1 <= cadena2

4. ¿Cómo se puede convertir una cadena a minúsculas en Python?

- a) cadena.lower()
- b) cadena.toLowerCase()
- c) cadena.toLower()
- d) cadena.minusculas()

5. ¿Cómo se puede dividir una cadena en Python?

- a) cadena.split()
- b) cadena.divide()
- c) cadena.break()
- d) cadena.separate()

7. Conversión

La conversión en Python se refiere a la transformación de un tipo de dato a otro. Esto puede implicar cambiar entre tipos numéricos (por ejemplo, de entero a flotante), entre tipos de datos (por ejemplo, de cadena a lista), o entre estructuras de datos (por ejemplo, de lista a tupla).

Tipos de conversión:

Conversión de cadena a entero (int): Se utiliza la función int() para convertir una cadena que representa un número entero en un entero.

Conversión de cadena a flotante (float): Se utiliza la función float() para convertir una cadena que representa un número flotante en un flotante.

Conversión de entero a cadena: Se utiliza la función str() para convertir un entero en una cadena.

Conversión de flotante a cadena: Se utiliza la función str() para convertir un flotante en una cadena.

Ejemplos:

Convertir un entero a una cadena:

```
entero = 42
```

```
cadena = str(entero)
```

Convertir una cadena a un entero:

```
cadena = "42"
```

```
entero = int(cadena)
```

Convertir una lista a una tupla:

```
lista = [1, 2, 3]
```

```
tupla = tuple(lista)
```

Autoevaluación 7

1. ¿Qué función se utiliza para convertir un entero a una cadena?

- a) int()
- b) str()
- c) float()
- d) list()

2. ¿Cuál es la función para convertir una cadena a un entero?

- a) int()
- b) str()
- c) float()
- d) list()

3. ¿Qué se utiliza para convertir una lista a una tupla?

- a) list()
- b) tuple()
- c) set()
- d) dict()

4. ¿Cuál de las siguientes opciones convierte un flotante a un entero?

- a) int()
- b) str()
- c) float()
- d) list()

5. ¿Qué función se utiliza para convertir una cadena a flotante?

- a) int()
- b) str()

- c) float()
- d) list()

8. Inversión

En Python, la inversión de cadenas es un proceso que permite cambiar el orden de los caracteres en una cadena de texto, de manera que el último carácter se convierte en el primero, el penúltimo en el segundo, y así sucesivamente. Este proceso es comúnmente utilizado en diversas aplicaciones de programación.

Un método sencillo para invertir una cadena en Python es utilizando el slicing.

Python permite acceder a los elementos de una cadena a través de índices, y el slicing es una funcionalidad que permite obtener subcadenas a partir de una cadena original. Para invertir una cadena, se puede utilizar el slicing con un paso de -1.

Por ejemplo, si se tiene la cadena original `cadena = "Hola Mundo"`, se puede invertir utilizando el slicing de la siguiente manera:

Ejemplo 1:

```
cadena_invertida = cadena[::-1]
print(cadena_invertida) # Imprime: "odnuM aloH"
```

Ejemplo 2:

```
cadena_invertida = cadena[::-1]
print(cadena_invertida) # Imprime: "laineG se nohtyP"
```

En este código, `cadena[::-1]` genera una nueva cadena que comienza en el último carácter de la cadena original (`cadena[-1]`), y continúa hacia atrás hasta el primer carácter (`cadena[0]`). Es importante mencionar que este método crea una nueva cadena, ya que las cadenas en Python son inmutables, lo que significa que no se pueden modificar una vez creadas.

Autoevaluación 8

1. ¿Cómo se invierte una cadena en Python?

a) cadena.reverse()

b) cadena.invert()

c) cadena[::-1]

d) cadena.flip()

2. ¿Qué hace el slicing cadena[::-1] en Python?

a) Elimina todos los caracteres de la cadena.

b) Convierte todos los caracteres de la cadena a mayúsculas.

c) Invierte el orden de los caracteres en la cadena.

d) Repite la cadena en orden inverso.

3. ¿Las cadenas en Python son mutables o inmutables?

a) Mutables

b) Inmutables

c) Depende de la cadena

d) Ninguna de las anteriores

4. ¿Qué devuelve cadena[::-1] si cadena = "Hola Mundo"?

a) "Hola Mundo"

b) "odnuM aloH"

c) "Mundo Hola"

d) "aloH odnuM"

5. ¿Cómo se puede verificar si una cadena es un palíndromo en Python?

a) cadena == cadena.reverse()

b) cadena == cadena.invert()

c) cadena == cadena[::-1]

d) cadena == cadena.flip()

9. Sub cadenas

En Python, una subcadena se refiere a una parte de una cadena más grande.

Puedes obtener subcadenas utilizando la técnica de indexación y rebanado. Aquí tienes algunos ejemplos:

Indexación: Puedes acceder a caracteres individuales en una cadena mediante su índice, donde el primer carácter tiene un índice de 0.

Por ejemplo:

```
cadena = "Hola mundo"
```

```
print(cadena [0]) # H
```

```
print(cadena [1]) # o
```

Rebanado (slicing): Puedes obtener una parte de una cadena utilizando la sintaxis **[inicio:final]**. El resultado incluirá los caracteres desde el índice de inicio hasta el índice de final (sin incluir el carácter en el índice final).

Por ejemplo:

```
cadena = "Hola mundo"

print(cadena[0:4]) # Hola

print(cadena[5:]) # mundo
```

También puedes omitir los índices para indicar que deseas comenzar desde el principio o ir hasta el final:

Ejemplo:

```
print(cadena[:4])      # Hola

print(cadena[5:])      # mundo

print(cadena[:])       # Hola mundo (devuelve toda la cadena)
```

Paso en rebanado: Puedes especificar un paso para moverte a través de la cadena.

Ejemplo:

```
cadena = "Hola mundo"

print(cadena[::-2]) # Hl ud (imprime cada segundo carácter)
```

Reversión de cadena: Puedes revertir una cadena fácilmente utilizando un paso negativo.

Ejemplo:

```
cadena = "Hola mundo"

print(cadena[::-1]) # odnum aloH (imprime la cadena al revés)
```

Trabajar con subcadenas en Python ofrece una amplia gama de opciones para adaptarse a diversas necesidades. Dependiendo de lo que quieras lograr, puedes ajustar

estas técnicas para obtener la subcadena específica que necesitas. Desde acceder a caracteres individuales hasta extraer porciones más grandes de una cadena, Python proporciona una variedad de herramientas para manejar eficientemente la manipulación de cadenas.

Autoevaluación 9

1. ¿Qué es una subcadena en Python y por qué es útil en la programación?

- a) Una subcadena en Python es una cadena que contiene solo caracteres numéricos. Es útil en la programación para realizar operaciones matemáticas específicas.
- b) Una subcadena en Python es una cadena que contiene solo caracteres alfabéticos. Es útil en la programación para manipular y extraer partes específicas de una cadena más grande.
- c) Una subcadena en Python es una cadena que contiene solo caracteres especiales como símbolos y puntuación. Es útil en la programación para el formateo de texto y la presentación de datos.
- d) Una subcadena en Python es una cadena que contiene solo caracteres de control como tabulaciones y saltos de línea. Es útil en la programación para el manejo de la entrada y salida de datos en archivos.

2. ¿Cuál es la diferencia entre indexación y rebanado (slicing) al trabajar con subcadenas en Python?

- a) La indexación se refiere a la obtención de un único carácter de una cadena utilizando su posición, mientras que el rebanado (slicing) se refiere a la obtención de una parte más grande de una cadena utilizando rangos de índices.

- b) La indexación se utiliza para dividir una cadena en partes más pequeñas, mientras que el rebanado (slicing) se utiliza para acceder a caracteres individuales en una cadena.
- c) La indexación se utiliza para acceder a subcadenas en una cadena, mientras que el rebanado (slicing) se utiliza para realizar operaciones aritméticas en cadenas.
- d) La indexación se utiliza para agregar nuevos caracteres a una cadena, mientras que el rebanado (slicing) se utiliza para eliminar caracteres de una cadena.

3. ¿Qué hace la expresión cadena[::-1] en Python?

- a) Invierte la cadena, es decir, devuelve la cadena en orden inverso.
- b) Elimina todos los caracteres de la cadena que no sean letras.
- c) Divide la cadena en palabras individuales.
- d) Convierte todos los caracteres de la cadena en mayúsculas.

4. ¿Cuál es la diferencia entre cadena[2] y cadena[2:3] al trabajar con subcadenas en Python?

- a) cadena[2] devuelve un solo carácter en el índice 2, mientras que cadena[2:3] devuelve una subcadena que incluye el carácter en el índice 2.
- b) cadena[2] devuelve una subcadena que incluye el carácter en el índice 2, mientras que cadena[2:3] devuelve un solo carácter en el índice 2.
- c) Ambas expresiones devuelven subcadenas del mismo tamaño, pero cadena[2] incluye el carácter en el índice 2 y cadena[2:3] no lo incluye.
- d) Ambas expresiones devuelven subcadenas del mismo tamaño, pero cadena[2:3] incluye el carácter en el índice 2 y cadena[2] no lo incluye.

5. ¿Qué expresión en Python se puede utilizar para obtener una subcadena que incluya todos los caracteres desde el segundo hasta el último, omitiendo el primer carácter?

- a) cadena[1:]
- b) cadena[:-1]
- c) cadena[2:]
- d) cadena[1:-1]

10. Búsqueda

Dentro de Python, existen múltiples técnicas para localizar una subcadena dentro de una cadena. Estas técnicas están disponibles a través de métodos integrados proporcionados por String en Python. Estos métodos facilitan la búsqueda y manipulación de subcadenas dentro de una cadena principal.

Buscar una subcadena con find:

Esta función proporciona la opción de especificar posiciones iniciales y finales dentro de la cadena para limitar la búsqueda a un rango específico.

Función: `find ("subcadena"[,posicion_inical,posicion_final])`

Devuelve un número entero que indica la posición donde comienza la subcadena dentro de la cadena principal. Si no se encuentra la subcadena, devuelve -1.

Ejemplo:

```
cadena = "me gusta la mermelada"  
subcadena = "mermelada"  
posicion_inicial = 12  
posicion_final = 18  
indice = cadena.find(subcadena, posicion_inicial, posicion_final)  
if indice >= 0:
```

```
    print("La subcadena '", subcadena, "' comienza en la posición", indice)
else:
    print("La subcadena '", subcadena, "' no está dentro del rango especificado")
```

Buscar una subcadena con index:

Función: index(sub [, pos_inicial, pos_final])

El método busca una subcadena dentro de otra cadena, similar a find(). Sin embargo, si la subcadena no se encuentra, en lugar de devolver -1, index (), lanza un error que indica que el valor no ha sido encontrado “valueError”.

Ejemplo:

```
cadena = "El cielo es azul"
subcadena = "azul"
posicion_inicial = 10
posicion_final = 30
try:
    indice = cadena.index(subcadena, posicion_inicial, posicion_final)
    print("La subcadena '", subcadena, "' comienza en la posición", indice)
except ValueError:
    print("La subcadena '", subcadena, "' no se encontró dentro del rango
especificado")
```

Autoevaluación 10

- 1. ¿Qué no hace el método index(sub [, pos_inicial, pos_final]) en Python?**
 - a) Busca una subcadena dentro de otra cadena.
 - b) Devuelve el índice de la primera ocurrencia de la subcadena.
 - c) Lanza una excepción ValueError si la subcadena no se encuentra.
 - d) Concatena dos cadenas.
- 2. ¿Cuál es el propósito de los parámetros posicion_inicial y posicion_final en el método find()?**
 - a) Especificar la subcadena a buscar.
 - b) Definir la posición inicial y final de la búsqueda dentro de la cadena.
 - c) Controlar el formato de salida del resultado.
 - d) Convertir la cadena a minúsculas.
- 3. ¿Qué valor devuelve el método index() si la subcadena no se encuentra dentro del rango especificado?**
 - a) -1
 - b) 0
 - c) Lanza una excepción ValueError
 - d) None

4. ¿Qué valor devuelve el método find() si la subcadena no se encuentra dentro de la cadena?

- a) -1
- b) 0
- c) Lanza una excepción ValueError
- d) None

5. ¿Cuál es la diferencia principal entre los métodos find() e index() en Python?

- a) find() devuelve -1 si la subcadena no se encuentra, mientras que index() lanza una excepción ValueError.
- b) find() lanza una excepción ValueError si la subcadena no se encuentra, mientras que index() devuelve -1.
- c) No hay diferencia, ambos métodos hacen lo mismo.
- d) Ninguna de las anteriores.

11. Cadenas y/o strings como parámetros de funciones

Las cadenas en Python o strings son un tipo inmutable que permite almacenar secuencias de caracteres. Para crear una, es necesario incluir el texto entre comillas dobles, y a menudo es necesario trabajar con ellas como parámetros de funciones en Python.

Caracteres :	P	y	t	h	o	n
Índice :	0	1	2	3	4	5
Índice inverso :	-6	-5	-4	-3	-2	-1

- **Pasar cadenas como parámetros:**

Cuando defines una función en Python, puedes incluir parámetros que representen cadenas de texto. Estos parámetros pueden luego ser utilizados dentro de la función.

Ejemplo:

```
def saludar(nombre):
    mensaje = f"Hola, {nombre}!"
    print(mensaje)
saludar("Juan")
```

- **Retornar cadenas desde funciones:**

Las funciones también pueden devolver cadenas como resultado. Puedes usar el tipo de dato str como tipo de retorno de la función.

Ejemplo:

```
def invertir_palabra(palabra):
    """
    Esta función toma una palabra como parámetro y devuelve la palabra invertida.
    """
    palabra_invertida = palabra[::-1]
    return palabra_invertida

# Llamada a la función con diferentes palabras
palabra_original = "Python"
```

Esta función toma una palabra como parámetro y devuelve la palabra invertida.

.....

palabra_invertida = palabra[::-1]

return palabra_invertida

Llamada a la función con diferentes palabras

palabra_original = "Python"

```

palabra_invertida = invertir_palabra(palabra_original)

# Impresión de los resultados

print(f"Palabra original: {palabra_original}")

print(f"Palabra invertida: {palabra_invertida}")

```



- **Operaciones con cadenas:**

Las cadenas en Python son objetos que admiten diversas operaciones.

Algunas de ellas incluyen la concatenación, la obtención de longitud, la indexación y la división.

Ejemplo

```

def operaciones_con_cadenas(cadena1, cadena2):

    # Concatenación de las dos cadenas

    concatenacion = cadena1 + " " + cadena2

    # Longitud de la cadena resultante

    longitud_resultante = len(concatenacion)

    # Conversión a mayúsculas

    mayusculas = concatenacion.upper()

```

```
# Imprimir resultados

print(f"Concatenación: {concatenacion}")

print(f"Longitud resultante: {longitud_resultante}")

print(f"Mayúsculas: {mayusculas}")

# Llamada a la función con dos cadenas

cadena_1 = "Hola"

cadena_2 = "Mundo"

operaciones_con_cadenas(cadena_1, cadena_2)
```

Autoevaluación 11

1. Pregunta: ¿Cómo se define una función en Python que toma una cadena como parámetro?

- a) definir_funcion(cadena):
- b) def funcion(cadena):
- c) crear_funcion(cadena):
- d)cadena_funcion(cadena):

2. Pregunta: ¿Cómo puedes concatenar dos cadenas en Python?

- a) concat(cadena1, cadena2)
- b) cadena1.join(cadena2)

- c) cadena1 + cadena2
- d) unir_cadenas(cadena1, cadena2)

3. Pregunta: ¿Cuál es la característica principal de las cadenas en Python en términos de modificación?

- a) Son mutables y se pueden modificar directamente.
- b) Son inmutables y no se pueden modificar directamente.
- c) Solo se pueden modificar mediante bucles for.
- d) Pueden modificarse en cualquier momento.

4. Pregunta: ¿Qué método se utiliza para dividir una cadena en Python?

- a) split()
- b) divide()
- c) separar()
- d) partir()

5. Pregunta: ¿Cómo puedes invertir una cadena en Python utilizando una función?

- a) cadena.reverse()
- b) invertir_cadena(cadena)
- c) cadena[::-1]

d) voltear_cadena(cadena)

Introducción a tipos de datos abstractos(TDA)

12. Declaración



En el contexto de la programación y estructuras de datos, un Tipo de Dato Abstracto (TDA) es un concepto que se refiere a una entidad matemática con operaciones bien definidas, pero sin especificar cómo se implementan esas operaciones. La declaración de un TDA se centra en la descripción de las operaciones que se pueden realizar sobre él, más que en los detalles internos de la implementación.

La declaración de un TDA incluye:

1. **Nombre del TDA:** Un nombre que representa el tipo de datos abstracto.
2. **Operaciones:** Las operaciones que se pueden realizar sobre el TDA, junto con sus firmas (nombres, tipos de parámetros y tipos de retorno). Estas operaciones definen el comportamiento del TDA.
3. **Invariantes:** Restricciones o condiciones que deben mantenerse en todas las instancias del TDA.
4. **Semántica:** El significado y comportamiento de cada operación en términos del TDA.

La declaración de un TDA proporciona una interfaz clara y abstrae los detalles internos de la implementación, permitiendo que diferentes implementaciones sean

intercambiables siempre que cumplan con la interfaz definida. Esta abstracción facilita el diseño modular y la reutilización del código, ya que los usuarios pueden utilizar el TDA sin necesidad de conocer los detalles internos de cómo se implementan las operaciones.

Ejemplo:

```
class Pila:
```

```
    def __init__(self):
```

```
        self.items = []
```

```
    def esta_vacia(self):
```

```
        return len(self.items) == 0
```

```
    def apilar(self, elemento):
```

```
        self.items.append(elemento)
```

```
    def desapilar(self):
```

```
        if not self.está_vacia():
```

```
            return self.items.pop()
```

```
    def ver_cima(self):
```

```
        if not self.está_vacia():
```

```
            return self.items[-1]
```

```
# Ejemplo de uso
```

```
mi_pila = Pila()  
  
mi_pila.apilar(1)  
  
mi_pila.apilar(2)  
  
mi_pila.apilar(3)  
  
print("Cima de la pila:", mi_pila.ver_cima())  
  
print("Desapilando:", mi_pila.desapilar())  
  
print("Cima de la pila después de desapilar:", mi_pila.ver_cima())
```

Este ejercicio implementa una **pila** utilizando una clase en Python. Una pila es una estructura de datos que sigue el principio de "último en entrar, primero en salir" (LIFO).

Autoevaluación 12

1. ¿Qué significa TDA?

- a) Tipo de datos abstractos
 - b) Datos de tipo boolean
 - c) Tipo de datos random
 - d) Una función que responde a mi necesidad

2. ¿ Que incluye el TDA ?

- a) Nombre del TDA, Operaciones, Invariantes, Semántica
 - b) Nombre del TDA, Invariantes, Funciones
 - c) Def, Import, Int, Input

d) Funciones varias

3. ¿ Cuántos tipos de datos existen ?

- a) 4
- b) 5
- c) 6
- d) 3

4. ¿ Que es la declaración TDA ?

- a) La declaración de un TDA se centra en la descripción de las operaciones que se pueden realizar sobre él, más que en los detalles internos de la implementación.
- b) La declaración de un TDA se centra en realizar un cadena de códigos complejos y sin salida
- c) La declaración TDA se basa en miles de datos encriptados en un mismo archivo
- d) La declaración TDA no es nada y solo sirve de relleno a cualquier código de python

5. ¿ En que no ayuda la declaración ?

- a) Esta abstracción facilita el diseño modular y la reutilización del código, ya que los usuarios pueden utilizar el TDA sin necesidad de conocer los detalles internos de cómo se implementan las operaciones
- b) Esta declaración nos ayuda a facilitar el movimiento de nuestro código en diferentes ramas, como C++, o java
- c) Esta declaración nos facilita el diseño de la página web que estemos realizando

- d) Esta declaración solo sirve para programación de alto nivel en semestres superiores

13. Definición de variables

En programación, una variable es un espacio de memoria asignado para almacenar un valor. Este valor puede cambiar durante la ejecución del programa, de ahí el término "variable".

Tipos de datos:

Las variables pueden contener diferentes tipos de datos, como números enteros, números de punto flotante, caracteres, cadenas de texto, booleanos, entre otros. Cada tipo de dato tiene un rango y una precisión específicos, y determina cómo se almacena y manipula la información en la memoria de la computadora.

Declaración de variables:

Antes de usar una variable en un programa, generalmente se debe declarar, es decir, se debe especificar su tipo y nombre. La declaración de variables reserva espacio en la memoria para almacenar los valores correspondientes y asigna un nombre a ese espacio de memoria para que pueda ser referenciado y manipulado por el programa.

Inicialización de variables:

Después de declarar una variable, a menudo se inicializa dándole un valor inicial. Esto es opcional en algunos lenguajes de programación, pero en otros es obligatorio antes de que se pueda utilizar la variable en operaciones posteriores.

Alcance de las variables:

El alcance de una variable se refiere a la parte del programa donde esa variable es válida y accesible. Puede ser local, lo que significa que solo está disponible dentro de un bloque de código específico, o global, lo que significa que está disponible en todo el programa.

Ámbito de las variables:

El ámbito de una variable se refiere a la parte del programa donde esa variable puede ser referenciada y utilizada. Esto está determinado por el alcance de la variable y las reglas de visibilidad del lenguaje de programación utilizado.

Asignación de valores a variables:

Una vez que una variable ha sido declarada e inicializada, se pueden asignar valores a esa variable en diferentes partes del programa utilizando operadores de asignación específicos.

Reutilización de variables:

En algunos casos, las variables pueden ser reutilizadas para almacenar diferentes valores durante la ejecución del programa. Esto puede ayudar a optimizar el uso de la memoria y mejorar la legibilidad y mantenibilidad del código.

Ejemplo de definición de variables

Declaración e inicialización de variables

número_entero = 10

número_decimal = 3.14

texto = "Hola, mundo!"

Declaración de una variable sin inicializar

```
otro_numero = None  
# Asignación de un nuevo valor a una variable  
numero_entero = 20
```

Autoevaluación 13

1. ¿Qué es una variable en programación?

 - a) Un valor constante que no cambia durante la ejecución del programa.
 - b) Un espacio de memoria reservado para almacenar un valor que puede cambiar durante la ejecución del programa.
 - c) Una función que realiza cálculos matemáticos.
 - d) Una estructura de control que permite tomar decisiones en un programa.
2. ¿Qué se entiende por "declarar una variable"?

 - a) Asignar un valor a una variable.
 - b) Definir una variable con su tipo de dato y nombre, reservando espacio en memoria.
 - c) Cambiar el valor de una variable.
 - d) Eliminar una variable de la memoria.
3. ¿Qué es el "ámbito de una variable"?

 - a) La parte del programa donde la variable es válida y accesible.
 - b) El valor inicial de una variable.

c) La cantidad de memoria reservada para almacenar una variable.

d) La forma en que se almacena una variable en la memoria.

4. ¿Qué significa "inicializar una variable"?

a) Cambiar el valor de una variable.

b) Asignarle un valor inicial a una variable cuando se declara.

c) Declarar una variable en un programa.

d) Eliminar el valor de una variable.

5. ¿Qué tipo de datos puede almacenar una variable?

a) Solo valores numéricos.

b) Solo caracteres.

c) Cualquier tipo de dato, dependiendo del lenguaje de programación.

d) Sólo valores booleanos.

14. Acceso

El acceso a los Tipos de Datos Abstractos (TDA) se refiere a cómo interactuamos

con ellos en nuestro código.

Encapsulación

Es un concepto que oculta la implementación interna de una estructura de datos,

exponiendo sólo lo necesario para interactuar con ella. Su importancia es que permite controlar el acceso de datos internos, mejorando la seguridad y estructura del código.

Algunas formas comunes de acceder al TDA:

Creación de Instancias:

Para usar un TDA, primero debemos crear una instancia de él. Por ejemplo, si estamos trabajando con una lista, creamos una lista vacía con `mi_lista = []`.

Operaciones Básicas:

Los TDA ofrecen operaciones específicas. Por ejemplo:

En una lista, podemos agregar elementos con `mi_lista.append(valor)` o acceder a un elemento con `mi_lista[indice]`.

En una pila, podemos apilar elementos con `mi_pila.push(valor)` o desapilar con `mi_pila.pop()`.

En una cola, podemos encolar elementos con `mi_colas.enqueue(valor)` o desencolar con `mi_colas.dequeue()`.

Iteración:

Podemos recorrer los elementos de un TDA utilizando bucles como `for` o `while`. Por ejemplo:

```
for elemento in mi_lista:
```

```
    print(elemento)
```

Acceso público a miembros

- Facilita el acceso desde fuera de la estructura: Puede agilizar el desarrollo, pero potencialmente disminuye la seguridad y control.
- Precauciones y problemas de seguridad: La exposición de miembros internos puede llevar a un mal uso o manipulación indebida.

Acceso privado a miembros

- Miembros privados en un ADT: Solo son accesibles desde dentro de la propia estructura.
- Restringe el acceso a los miembros internos: Mayor control sobre cómo se accede y se modifica la información.

Autoevaluación 14

1. ¿Qué es la encapsulación en relación con los TDA?

- Ocultar la implementación interna de una estructura de datos.
- Exponer todos los detalles de implementación.
- Permitir acceso ilimitado a los miembros internos.
- No afectar la seguridad del código.

2. ¿Cuál es la importancia de la encapsulación en los TDA?

- Mejorar la seguridad y estructura del código.
- Aumentar la complejidad del software.
- Facilitar el acceso público a los miembros.
- Reducir la modularidad.

3. ¿Qué operación se utiliza para crear una instancia de un TDA?

- `mi_lista.append(valor)`
- `mi_pila.push(valor)`
- `mi_cola.enqueue(valor)`
- Creamos una instancia con `mi_lista = []`

4. ¿Cuál es el principio detrás de las pilas?

- a. “Primero en entrar, primero en salir” (FIFO).
- b. “Último en entrar, primero en salir” (LIFO).
- c. “Primero en entrar, último en salir” (FILO).
- d. “Último en entrar, último en salir” (LILO).

5. ¿Qué ventaja ofrece el acceso privado a los miembros en un TDA?

- a. Mayor control sobre cómo se accede y modifica la información.
- b. Facilita el acceso desde fuera de la estructura.
- c. Aumenta la seguridad del código.
- d. Permite la exposición de miembros internos.

15. Almacenamiento de información

El almacenamiento de información en funciones en Python es un concepto fundamental para la programación modular y la reutilización de código. En Python, una función es un bloque de código reutilizable que realiza una tarea específica. Puedes almacenar información dentro de una función de varias maneras, incluyendo variables locales, argumentos de función, y estructuras de datos como listas, tuplas, diccionarios, y conjuntos.(Walker, 2018)

1. Variables locales. Dentro de una función, puedes declarar variables locales para almacenar información temporal. Estas variables solo son accesibles dentro de la función y se eliminan de la memoria cuando la función termina de ejecutarse. Las variables locales se definen simplemente asignando un valor a un nombre de variable dentro de la función. (Fernández, 2013)

```
def mi_funcion():
```

```
    variable_local = 10
```

```
    print(variable_local)
```

```
mi_funcion()
```

2. Argumentos de función. Puedes pasar información a una función mediante argumentos. Estos argumentos pueden ser utilizados dentro de la función para realizar cálculos o ejecutar operaciones. Los argumentos pueden ser cualquier tipo de dato en Python. (Walker, 2018)

```
def suma(a, b):  
  
    resultado = a + b  
  
    return resultado  
  
print(suma(5, 3))
```

3. Estructuras de datos. Las estructuras de datos como listas, tuplas, diccionarios y conjuntos pueden ser utilizadas para almacenar y manipular información dentro de una función. Estas estructuras de datos pueden ser creadas dentro de la función o pasadas como argumentos. (Fernández, 2013)

Listas:

```
def duplicar_elementos(lista):  
  
    for i in range(len(lista)):  
  
        lista[i] *= 2  
  
    return lista  
  
lista_original = [1, 2, 3, 4]  
  
print(duplicar_elementos(lista_original))
```

Tuplas:

```
def obtener_valor(tupla, indice):  
  
    return tupla[indice]  
  
tupla_datos = (10, 20, 30, 40)  
  
print(obtener_valor(tupla_datos, 2))
```

Diccionarios:

```
def obtener_valor(diccionario, clave):  
  
    return diccionario[clave]  
  
datos = {"nombre": "Juan", "edad": 30, "ciudad": "Madrid"}  
  
print(obtener_valor(datos, "nombre"))
```

Conjuntos:

```
def union_conjuntos(conjunto1, conjunto2):  
  
    return conjunto1.union(conjunto2)  
  
conjunto_a = {1, 2, 3}  
  
conjunto_b = {3, 4, 5}  
  
print(union_conjuntos(conjunto_a, conjunto_b))
```

Almacenar información en funciones en Python es útil para modularizar el código y hacerlo más fácil de entender y mantener. Dependiendo de los requisitos del programa,

puedes elegir la mejor manera de almacenar y manipular la información dentro de las funciones.

Autoevaluación 15

1. ¿Cuál es una forma de almacenar información dentro de una función en Python?

- a) Variables globales
- b) Argumentos de función
- c) Constantes
- d) Clases y objetos

2. ¿Qué estructura de datos se utiliza comúnmente para almacenar y manipular información dentro de una función en Python?

- a) Arrays
- b) Tuplas
- c) Estructuras
- d) Pilas

3. ¿Qué tipo de variables son accesibles solo dentro de la función en la que se definen en Python?

- a) Variables estáticas
- b) Variables privadas
- c) Variables locales
- d) Variables globales

4. ¿Cuál de las siguientes afirmaciones sobre los argumentos de función en Python es correcta?

- a) Los argumentos sólo pueden ser de tipo entero.
- b) Los argumentos son opcionales en todas las funciones.
- c) Los argumentos pueden ser de cualquier tipo de datos.
- d) Los argumentos no pueden ser pasados por referencia.

5. ¿Qué estructura de datos de Python se utiliza para almacenar pares clave-valor y puede ser utilizada para almacenar información dentro de una función?

- a) Listas
- b) Tuplas
- c) Diccionarios
- d) Conjuntos

16. Lectura de información

La lectura de información en programación se refiere al proceso de obtener datos de diferentes fuentes para su posterior procesamiento por parte del programa. Esto puede incluir datos proporcionados por el usuario durante la ejecución del programa, datos almacenados en archivos en el sistema de archivos del ordenador, información almacenada en bases de datos, datos obtenidos de servicios web externos a través de solicitudes HTTP, así como datos provenientes de dispositivos físicos como sensores o dispositivos externos conectados al sistema.

Este proceso es fundamental ya que los datos son la materia prima con la que los programas trabajan para realizar cálculos, tomar decisiones o generar resultados. La capacidad de leer datos de manera eficiente y precisa es esencial para que los programas funcionen correctamente y puedan cumplir con sus objetivos.



Ejemplos

Entrada del usuario:

Un programa de calculadora que solicita al usuario ingresar dos números y luego realiza una operación matemática con esos números.

Archivos:

Un programa de procesamiento de texto que lee un archivo de texto y cuenta la cantidad de palabras que contiene.

Bases de datos:

Un programa de gestión de inventario que consulta una base de datos para obtener información sobre los productos disponibles y su cantidad en stock.

Un sistema de gestión de usuarios que consulta una base de datos de usuarios para verificar las credenciales de inicio de sesión de un usuario.

Servicios web:

Un programa de pronóstico del tiempo que hace una solicitud a una API de pronóstico del tiempo y muestra la temperatura actual y las condiciones climáticas.

Sensores y dispositivos externos:

Un programa de monitoreo de la salud que lee datos de un sensor de frecuencia cardíaca para rastrear el ritmo cardíaco de una persona.

Autoevaluación 19

- 1. ¿Cuál de las siguientes opciones describe mejor la lectura de información en programación?**
 - a) El proceso de procesar datos para su posterior almacenamiento.
 - b) Obtener datos de diferentes fuentes para su posterior procesamiento por parte del programa.
 - c) La manipulación de datos dentro del programa para generar resultados.
 - d) El proceso de visualización de datos en la interfaz de usuario.

- 2. ¿Cuál de las siguientes fuentes no se mencionó como una fuente común de lectura de información en programación?**
 - a) Entrada del usuario
 - b) Sensores y dispositivos externos
 - c) Software antivirus
 - d) Archivos

- 3. ¿Cuál de las siguientes afirmaciones es verdadera sobre la lectura de archivos en programación?**
 - a) Los archivos solo contienen datos de texto.
 - b) Los archivos solo se pueden leer, no escribir.
 - c) Los archivos pueden contener una variedad de datos, como texto, datos estructurados y datos binarios.
 - d) Los archivos solo se pueden leer si están en el formato correcto.

- 4. ¿Cuál de las siguientes tareas no implica la lectura de información en programación?**
 - a) Leer un archivo de configuración al inicio del programa.
 - b) Consultar una base de datos para obtener información actualizada.
 - c) Imprimir resultados en la pantalla para que el usuario los vea.

d) Obtener datos de un sensor para monitorear una variable ambiental.

5. ¿Cuál de las siguientes opciones describe mejor por qué es importante la lectura de información en programación?

- a) Para aumentar la complejidad del código.
- b) Para reducir la cantidad de datos necesarios para el programa.
- c) Para que el programa pueda obtener datos necesarios para realizar sus tareas.
- d) Para disminuir la velocidad de ejecución del programa.

17. Recuperación de la información

Recuperación de datos es una herramienta que sirve para recuperar los archivos perdidos de un medio de almacenamiento, bien por un fallo del disco duro, bien porque se han borrado de manera accidental o bien por otros motivos.

Un programa de recuperación de datos está construido con características únicas que trabajan alrededor del dispositivo de almacenamiento para revelar la secuencia exacta de los números binarios y reorganizarlos para exponer los archivos eliminados. De esta forma, podrás recuperar los archivos perdidos. Un buen programa de recuperación de datos debe ser fiable y recuperar los archivos en su estado original en poco tiempo. Además, debe ser capaz de recuperar una amplia gama de tipos de archivos de los distintos dispositivos de almacenamiento disponibles.

En Python, la recuperación de datos generalmente se refiere al proceso de obtener información almacenada en algún tipo de estructura de datos, como listas, diccionarios, archivos, bases de datos, entre otros. Dependiendo del contexto, la recuperación de datos puede involucrar la lectura de archivos, consultas a bases de datos, manipulación de datos en memoria, entre otras operaciones. Por ejemplo, si tienes una lista de elementos en Python y deseas recuperar un elemento específico de esa lista, puedes hacerlo utilizando su índice, de la siguiente manera:

```
mi_lista = [10, 20, 30, 40, 50]
```

```
elemento_recuperado = mi_lista[1]  
print("Elemento recuperado:", elemento_recuperado)
```

En este ejemplo, se recupera el segundo elemento de la lista (20) utilizando el índice 1.

La recuperación de datos puede ser un concepto más amplio dependiendo del contexto específico de la aplicación, como recuperar datos de una base de datos SQL, leer archivos CSV o Excel, realizar solicitudes HTTP para obtener datos de una API web, entre otras operaciones.

Otro ejemplo se puede tener en un diccionario, de la siguiente manera:

```
productos = {  
    'manzanas': 2.5,  
    'plátanos': 1.8,  
    'uvas': 3.0,  
    'naranjas': 2.2,  
    'peras': 2.8  
}  
  
precio_uvas = productos['uvas']  
print("El precio de las uvas es:", precio_uvas)
```

En este ejemplo, el diccionario de productos contiene nombres de frutas como claves y sus respectivos precios como valores. Se recupera el precio de las uvas accediendo al diccionario mediante la clave 'uvas'.

Autoevaluación 16

1. ¿Cómo se accede a un valor específico en un diccionario en Python?

- a. Utilizando la función get()
- b. Mediante corchetes []
- c. Con la función search()
- d. Usando la función value()

2. ¿Cuál es el resultado de intentar acceder a una clave que no existe en un diccionario utilizando la función get()?

- a. Se produce un error
- b. Retorna el valor None
- c. Retorna un mensaje de advertencia
- d. Retorna una cadena vacía

3. ¿Qué método se utiliza para obtener todas las claves de un diccionario en Python?

- a. keys()
- b. get_keys()
- c. all()
- d. values()

4. ¿Cuál es la forma correcta de verificar si una clave existe en un diccionario en Python?

- a. Utilizando la función exists()
- b. Con la palabra clave in
- c. Mediante la función has_key()
- d. Usando el método contains()

5. ¿Cuál es la complejidad temporal (en el peor caso) de la operación de búsqueda de una clave en un diccionario de Python?

- a. O(1)
- b. O(n)
- c. O(log n)
- d. O(n log n)

Entrada y salida por archivos

18. Archivos de texto

Un archivo de texto contiene un conjunto de caracteres estructurados en distintas líneas. Es un formato de archivo ampliamente utilizado como pueden ser:

- 1) El código fuente de un script en Python se almacena en un archivo de texto (igual que cualquier otro lenguaje de programación)
- 2) Archivos HTML, CSS, XML se almacenan en archivos de texto.
- 3) Archivos JSON etc.

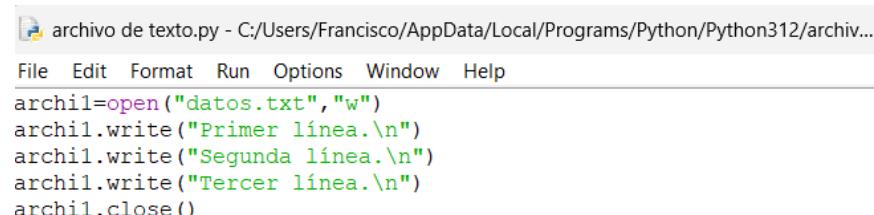
Creación de un archivo de texto y almacenamiento de datos.

Como es una actividad tan común en todo programa el lenguaje Python incluye por defecto todas las funcionalidades para trabajar con archivos de texto.

Ejemplo:

Crear un archivo de texto llamado 'datos.txt', almacenar tres líneas de texto. Abrir luego el archivo creado con un editor de texto.

Codificación de Python:



```
archivo de texto.py - C:/Users/Francisco/AppData/Local/Programs/Python/Python312/archiv...
File Edit Format Run Options Window Help
archi1=open("datos.txt","w")
archi1.write("Primer línea.\n")
archi1.write("Segunda línea.\n")
archi1.write("Tercer línea.\n")
archi1.close()
```

```
archi1=open("datos.txt","w")
```

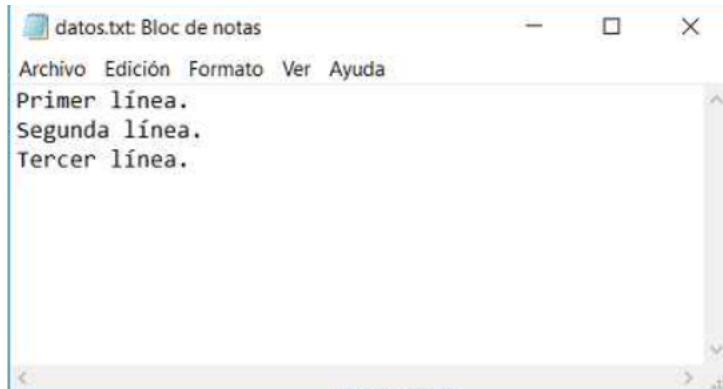
```
archi1.write("Primer línea.\n")
```

```
archi1.write("Segunda línea.\n")
```

```
archi1.write("Tercer línea.\n")
```

```
archi1.close()
```

Luego de ejecutar este programa (no muestra nada por pantalla) debemos abrir con un editor de texto el archivo de texto que se acaba de crear llamado 'datos.txt':



Para crear un archivo de texto debemos llamar a la función open y pasar dos parámetros, el primero indica el nombre del archivo a crear y el segundo un string con el carácter "w" (la "w" write indica crear el archivo de texto):

```
archi1=open("datos.txt","w")
```

Si el archivo 'datos.txt' ya existe luego se crea uno nuevo y se borra el actual.

El archivo se crea en la misma carpeta donde se está ejecutando el script de Python, si necesitamos que se cree en otra carpeta podemos indicar el path del mismo:

```
archi1 = open("c:/administracion/datos.txt","w")
```

Si indicamos un path inexistente se genera un error.

Para grabar caracteres en el archivo de texto utilizamos el método 'write' y le pasamos un string a grabar:

```
archi1.write("Primer línea.\n")
```

Mediante la sintaxis \n indicamos que debe almacenarse un salto de línea en el archivo de texto.

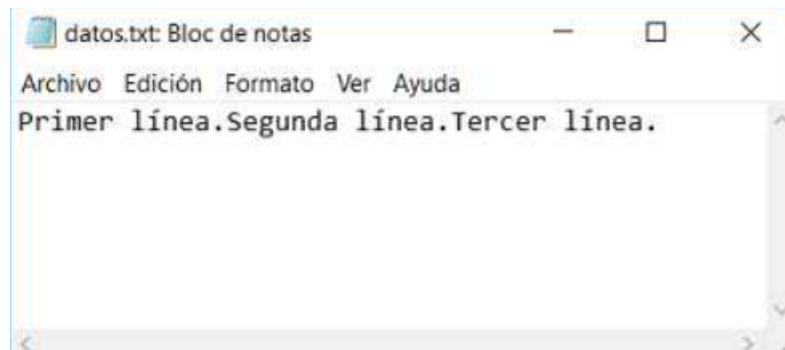
Si no incluimos los respectivos \n:

```
archi1.write("Primer línea.")
```

```
archi1.write("Segunda línea.")
```

```
archi1.write("Tercer línea.")
```

El archivo de texto tiene una sola línea:



Luego de trabajar con el archivo de texto debemos liberarlo para que otro programa pueda acceder a su contenido:

```
archi1.close()
```

Autoevaluación 17

1. ¿Qué función se utiliza para abrir un archivo en Python?

- a) read()
- b) open()
- c) write()
- d) create()

1. ¿Cuál es el modo correcto para abrir un archivo en modo de escritura?

- a) 'r'
- b) 'w'

c) 'a'

d) 'x'

2. ¿Qué método se utiliza para leer todo el contenido de un archivo de texto?

a) read()

b) readline()

c) readlines()

d) todas las anteriores

3. ¿Qué método se utiliza para escribir una cadena de texto en un archivo?

a) write()

b) writelines()

c) append()

d) add()

4. ¿Qué ocurre si intentas abrir un archivo en modo de escritura que ya existe?

a) Se agrega el nuevo contenido al final del archivo

b) Se sobrescribe el contenido existente

c) Se produce un error

d) No ocurre nada

19. Archivos Binarios

Los archivos binarios son archivos que contienen datos en un formato no legible para los humanos, ya que están compuestos por secuencias de bytes. A diferencia de los archivos de texto, que utilizan caracteres legibles, los archivos binarios pueden contener información no solo de texto, sino también datos multimedia, ejecutables, etc.

Funciones comunes utilizadas para trabajar con archivos binarios en Python:

1. `open()`: Abre un archivo en un determinado modo (lectura, escritura, binario, etc.).

```
```python
```

```
with open('archivo.bin', 'rb') as archivo:
```

```
...
...
```

2. `read(size)` : Lee una cantidad específica de bytes desde el archivo.

```
```python  
datos = archivo.read(100)  
...  
````
```

3. `write(data)` : Escribe datos en el archivo.

```
```python  
archivo.write(datos_nuevos)  
...  
````
```

4. `seek(offset, whence)` : Mueve el puntero de lectura/escritura a una posición específica.

```
```python  
archivo.seek(0, 0) # Mueve al principio del archivo  
...  
````
```

5. `tell()` : Devuelve la posición actual del puntero en el archivo.

```
```python  
posicion = archivo.tell()  
...  
````
```

6. `close()` : Cierra el archivo después de su uso.

```
```python  
archivo.close()  
...  
````
```

Estas funciones proporcionan las operaciones básicas para trabajar con archivos binarios en Python. La combinación de estas funciones te permite leer, escribir y manipular datos binarios de manera efectiva.

Un ejemplo común de archivo binario es un archivo de imagen, como un archivo JPEG o PNG. Estos archivos almacenan información de píxeles y otros datos relacionados de una manera binaria, lo que los hace no legibles directamente en forma de texto.

Este es un ejemplo simple y claro para leer y abrir un archivo binario:

```
Abre un archivo binario en modo lectura
```

```
with open('ejemplo_imagen.jpg', 'rb') as archivo_binario:
 # Lee los datos binarios del archivo
 datos_binarios = archivo_binario.read()

 # Puedes realizar operaciones adicionales con los datos binarios, dependiendo de tus
 necesidades

 # Por ejemplo, podrías procesar la imagen, extraer información, etc.
```

### Autoevaluación 18

- 1. ¿Qué función se utiliza para abrir un archivo en Python?**
  - a. `load()`
  - b. `open()`
  - c. `readfile()`
  - d. `fileopen()`
- 2. ¿Cómo se lee una cantidad específica de bytes desde un archivo binario?**
  - a. `read\_line()`
  - b. `read\_bytes()`
  - c. `readall()`
  - d. `read()`
- 3. ¿Cuál es la función para escribir datos en un archivo binario en Python?**
  - a. `save()`
  - b. `write()`
  - c. `append()`
  - d. `insert()`
- 4. ¿Qué función se utiliza para mover el puntero de lectura/escritura a una posición específica en un archivo binario?**
  - a. `shift()`
  - b. `move()`
  - c. `seek()`
  - d. `position()`

**5. ¿Qué función devuelve la posición actual del puntero en un archivo binario?**

- a. `current\_position()`
- b. `tell()`
- c. `position()`
- d. `get\_pointer()`

## BIBLIOGRAFÍA

Gomis, P. (2018). *Fundamentos de Programación en Python (Spanish)*.

Cabanes, P. N. (s.f) C++ -9.8. *Recursividad-Aprendeaprogramar.com*. Aprendeaprogramar.com. Recuperado el 7 de febrero de 2024, de: <https://www.aprendeaprogramar.com/cursos/verApartado.php?id=1690>

Llamas, L. (2020, 30 de octubre). Programación: Ámbito de las variables. Luis Llamas

Gonzalez, L. (2020, septiembre 1). *Librería NumPy*. Aprende IA. recuperado de: <https://aprendeia.com/libreria-de-python-numpy-machine-learning/>

*Ejercicio 1: Estructura arreglo unidimensional en python.* (2019, Marzo 3). Youtube.

[https://www.youtube.com/watch?v=RblYaoyKsEg&t=62s&ab\\_channel=EDWARDDA](https://www.youtube.com/watch?v=RblYaoyKsEg&t=62s&ab_channel=EDWARDDA)  
NIELVIDALGARCIA

GERMEC, M. (2022). *Python Tutorial (Codes)*. Scribd.

<https://es.scribd.com/document/638695265/Python-Tutorial>

ProgramacionATS. (2016, Abril 21). *Programación en C++ || Arreglos || Concepto de Arreglo Unidimensional*. Youtube.

[https://www.youtube.com/watch?v=l-kOjxvgyDQ&ab\\_channel=Programaci%C3%B3nATS](https://www.youtube.com/watch?v=l-kOjxvgyDQ&ab_channel=Programaci%C3%B3nATS)

Youtube. (2022, Diciembre 4). *Arreglo Unidimensionales en Python Nivel Fácil*.

[https://www.youtube.com/watch?v=P7fipvJ52rw&t=183s&ab\\_channel=Codificando](https://www.youtube.com/watch?v=P7fipvJ52rw&t=183s&ab_channel=Codificando)

Amorin, D. (2022, agosto 15). *Ordenamiento por Selección en Python*. Diego Amorin.

<https://diegoamorin.com/ordenamiento-seleccion-python/>

BlackeyeB. (2023, mayo 1). *Algoritmos de ordenación explicados con ejemplos en JavaScript, Python, Java y C++*. freecodecamp.org.

<https://www.freecodecamp.org/espanol/news/algoritmos-de-ordenacion-explicados-con-ejemplos-en-javascript-python-java-y-c/>

Walker, J. S. (2018). *Python: La Guía Definitiva para Principiantes para Dominar Python*. Babelcube Inc.

FERNANDEZ, A. (2013). *Python 3 al descubierto—2a ed.* Alfaomega Grupo Editor.

Tagliaferri, L. (2020, 15 diciembre). Cómo convertir tipos de datos en Python 3.

DigitalOcean.

<https://www.digitalocean.com/community/tutorials/how-to-convert-data-types-in-python-3>

Aguilar, I. Z. L. J., & Martinez, I. Z. (2005). Programacion en C. MCGRAW-HILL INTERAMERICANA-MUA.

AHO, V., ALFRED, H., JOHN, U., & JEFFREY. (1998). ESTRUCTURAS DE DATOS Y ALGORITMOS. MEXICO, D.F.: IMPRESIONES ALDINA, S.A.

Delgado, H. (2020). Bucles ejemplos y sintaxis - Ciclos While, Do While y For

Escuela Universitaria de Ingeniería Técnica Industrial de Gijón. (2002). Entrada y salida de datos.

[http://di002.edv.uniovi.es/~alberto\\_mfa/computadores2001/apuntes/Tema%2011.pdf](http://di002.edv.uniovi.es/~alberto_mfa/computadores2001/apuntes/Tema%2011.pdf)

Informática básica: ¿Qué es un programa o aplicación? (s. f.). GCFGloba.org.

<https://edu.gcfglobal.org/es/informatica-basica/que-es-un-programa-o-aplicacion/1/>

Joyanes Aguilar, L. (1998). Fundamentos de Programación, Algoritmos y Estructuras de Datos. McGraw-Hill-3-es

## SOLUCIONARIO

### Introducción a la Programación de Computadores

Paradigmas de Programación

### Autoevaluación 1

**1. ¿Cuál de los siguientes enunciados describe mejor el paradigma de programación orientada a objetos?**

- a) Utiliza funciones como unidades principales de organización del código.
- b) Se centra en la manipulación directa de la memoria del sistema.
- c) Organiza el código en torno a la interacción entre objetos que encapsulan datos y comportamientos.
- d) Se basa en la ejecución secuencial de instrucciones paso a paso.

**2. ¿Cuál de los siguientes enunciados describe mejor el paradigma de programación funcional?**

- a) Se centra en la manipulación directa de la memoria del sistema.
- b) Organiza el código en torno a la interacción entre objetos que encapsulan datos y comportamientos.
- c) Utiliza funciones como unidades principales de organización del código, tratando la programación como una serie de evaluaciones de funciones matemáticas.
- d) Se basa en la ejecución secuencial de instrucciones paso a paso.

**3. ¿Cuál de las siguientes afirmaciones describe mejor el paradigma de programación declarativa?**

- a) Organiza el código en torno a la interacción entre objetos que encapsulan datos y comportamientos.
- b) Utiliza funciones como unidades principales de organización del código, tratando la programación como una serie de evaluaciones de funciones matemáticas.
- c) Se centra en la manipulación directa de la memoria del sistema.
- d) Especifica qué resultados se desean y no cómo obtenerlos, dejando la implementación de los detalles al sistema.

**4. ¿Cuál de las siguientes afirmaciones describe mejor el paradigma de programación lógica?**

- a) Organiza el código en torno a la interacción entre objetos que encapsulan datos y comportamientos.
- b) Utiliza funciones como unidades principales de organización del código, tratando la programación como una serie de evaluaciones de funciones matemáticas.
- c) Se centra en la manipulación directa de la memoria del sistema.
- d) Define relaciones y reglas lógicas para expresar la relación entre los datos y permite realizar consultas lógicas.**

**5. ¿Cuál de las siguientes afirmaciones describe mejor el paradigma de programación imperativa?**

- a) Organiza el código en torno a la interacción entre objetos que encapsulan datos y comportamientos.
- b) Utiliza funciones como unidades principales de organización del código, tratando la programación como una serie de evaluaciones de funciones matemáticas.
- c) Se centra en la manipulación directa de la memoria del sistema.
- d) Se centra en el uso de instrucciones que modifican el estado del programa, controlando el flujo de ejecución a través de secuencias de comandos.**

## Compilación e Intérprete

### Autoevaluación 2

**¿Cuál es la fase en la que un compilador analiza la estructura del programa fuente?**

- a) Optimización.

b) Síntesis.

c) Análisis léxico.

d) Generación de código.

**¿Cuál de las siguientes afirmaciones es verdadera acerca de los intérpretes?**

a) La traducción a código máquina se realiza antes de la ejecución.

b) Los intérpretes son generalmente más eficientes que los compiladores.

c) Los errores léxicos y sintácticos se detectan durante la ejecución.

d) El código fuente se traduce completamente antes de ejecutarse.

**¿Cuál es la principal diferencia entre un compilador y un intérprete?**

a) Un compilador traduce el código fuente a código máquina, mientras que un intérprete ejecuta el código directamente.

b) Ambos son términos intercambiables y no hay diferencia.

c) Un compilador ejecuta el código directamente, mientras que un intérprete traduce el código a código máquina.

d) No hay diferencia significativa entre ellos.

**¿Cuál es una ventaja específica del uso de un intérprete en términos de portabilidad del código fuente?**

a) El código fuente se traduce a código máquina.

b) Permite la ejecución en cualquier sistema que tenga el intérprete instalado.

c) Facilita el desarrollo interactivo.

- d) Permite la optimización durante el proceso de compilación.

**¿Cuál es una ventaja del uso de un compilador en comparación con un intérprete?**

- a) Mayor portabilidad del código fuente.
- b) Facilita el desarrollo interactivo.
- c) El programa compilado generalmente se ejecuta más rápido.
- d) Permite probar fragmentos de código sin compilar.

### **IDE vs editores**

#### **Autoevaluación 3**

- 1. ¿Qué ofrecen los IDEs que los editores de texto no suelen proporcionar?**
  - a) Resaltado de sintaxis
  - b) Compilador/intérprete
  - c) Autocompletado
  - d) Búsqueda de texto
- 2. ¿Cuál de las siguientes herramientas se encuentra comúnmente en un IDE?**
  - a) Editor de texto
  - b) Compilador
  - c) Resaltado de sintaxis
  - d) Autocompletado
- 3. ¿Qué característica es más típica de los editores de texto en lugar de los IDEs?**

- a) Herramientas de depuración
- b) Resaltado de sintaxis
- c) Búsqueda de texto
- d) Compilador integrado

**4. ¿Qué factor suele influir en la elección entre un IDE y un editor de texto?**

- a) Resaltado de sintaxis
- b) Preferencias personales del desarrollador
- c) Herramientas de depuración
- d) Autocompletado

**5. ¿Cuál de las siguientes no es una característica común de los editores de texto?**

- a) Resaltado de sintaxis
- b) Compilador integrado
- c) Autocompletado
- d) Búsqueda de texto

#### **Autoevaluación 4**

**1. ¿Qué técnica se centra en la representación visual de un algoritmo utilizando cajas para representar pasos y flechas para indicar el flujo?**

- a) Diagrama de flujo
- b) Árbol de decisión
- c) Mapa conceptual
- d) Grafo

**2. ¿Cuál de los siguientes enfoques se utiliza para evitar la repetición de cálculos y mejorar la eficiencia al almacenar resultados previos?**

a) Programación dinámica

b) Programación lineal

c) Programación reactiva

d) Programación paralela

**3. ¿Qué técnica se utiliza para iterar sobre una colección de elementos sin tener que preocuparse por la implementación específica de la estructura de datos?**

a) Búsqueda binaria

b) Iteración recursiva

c) Iteradores

d) Programación concurrente

**4. ¿Cuál es el propósito principal de las pruebas unitarias en el desarrollo de software?**

a) Evaluar el rendimiento del programa

b) Verificar que cada unidad de código funciona como se espera.

c) Realizar pruebas de integración entre módulos.

d) Realizar codificación

**5. ¿Qué es la refactorización de código?**

a) Escribir código desde cero.

b) Optimizar el código existente sin cambiar su funcionalidad.

c) Documentar el código para futuras referencias.

d) Son binarios

Autoevaluación 5

### **Estrategias para solucionar problemas de programación**

Fases en la resolución de problemas programación aplicando las fases

**1. En la fase de "Comprender el Problema", ¿cuál es el propósito principal de realizar un análisis de requisitos?**

- a. Identificar patrones en el código.
- b. Definir claramente el problema y sus necesidades.**
- c. Implementar directamente la solución en código.
- d. Evaluar la eficacia del código.

**2. Durante la fase de "Planificar la Solución", ¿por qué es crucial realizar un diseño de algoritmos antes de comenzar a codificar?**

- a. Para analizar la solución en términos de eficacia.
- b. Para definir claramente el problema.
- c. Para traducir la estrategia de solución en código.
- d. Para asegurar una implementación estructurada y eficiente.**

**3. En la fase de "Implementar la Solución", ¿cuál es el propósito principal de realizar pruebas de unidad en el código?**

- a. Dividir el problema en subproblemas.
- b. Evaluar la eficiencia del código.
- c. Verificar que cada componente funcione correctamente.**
- d. Analizar el problema y sus requisitos.

**4. Al evaluar la eficiencia en la fase correspondiente, ¿por qué se considera importante realizar pruebas de rendimiento en el código?**

- a. Identificar patrones en el código.

- b. Medir el tiempo de ejecución y uso de recursos.
- c. Analizar la solución en términos de eficacia.
- d. Definir claramente el problema.

**5. ¿Cuál de las siguientes técnicas se emplea en la fase de "Dividir y Conquistar" para abordar problemas complejos?**

- a. Fuerza Bruta.
- b. Análisis de complejidad espacial.
- c. Descomponer un problema en subproblemas más manejables.
- d. Reconocimiento de Patrones.

## **Autoevaluación 6**

**1. ¿Qué es la microcomputadora?**

- a) Es una computadora pequeña
- b) Es un programa de codificación
- c) Es una página web
- d) Se lo utiliza en los bucles for y while

**2. ¿Qué es el software?**

- a) Conjunto de programas que permite a la computadora no realizar tareas
- b) Conjunto de programas que permite a la computadora realizar tareas
- c) Conjunto de programas que permite a la computadora y a la tablet realizar tareas
- d) Conjunto de programas que permite al código realizar tareas

### **3. ¿Qué significan las siglas HTML?**

- a) Es Hypertext Markup Language y es lo que no compone la web.
- b) Es Hypertext Markup Language y es lo que puede componer la web.
- c) Es Hypertext Markup Language y es lo que vamos a ver en la web.
- d) Es Hypertext Markup Language y es lo que compone la web.**

### **4. ¿Qué es la IA?**

- a) Un robot con apariencia humana
- b) Debe ver con el if y else en python
- c) Es la disciplina y un conjunto de capacidades cognoscitivas e intelectuales expresadas por sistemas informáticos**
- d) Disciplina y no es conjunto de capacidad cognitiva que expresa un fin informático.

### **5. ¿Qué es código de máquina?**

- a) Sistema que interpreta circuitos y permite a la máquina analizarlos
- b) Sistema que interpreta circuitos y permite a la máquina analizarlos y autodestruirnos si no convencen
- c) Sistema que interpreta circuitos y permite a la máquina analizarlos para la ejecución de un programa**
- d) Sistema que interpreta circuitos y permite a la máquina analizarlos en python

## **Algoritmos**

Definición y características de Algoritmos.

### **Autoevaluación 6**

#### **1. ¿Cómo se define un algoritmo?**

- a) Un conjunto caótico de pasos.
- b) Un conjunto ordenado y finito de pasos o reglas.
- c) Un conjunto infinito de reglas.
- d) Un conjunto de pasos indefinidos.

#### **2. ¿Cuál es una característica esencial del concepto de "finitud" en un algoritmo?**

- a) Puede tener un número infinito de pasos.
- b) Debe tener un número finito de pasos.
- c) Puede tener pasos indefinidos.
- d) No es necesario que finalice.

#### **3. ¿Qué significa que un algoritmo sea "Determinista"?**

- a) Puede producir diferentes salidas para las mismas entradas.
- b) Siempre producirá la misma salida para un conjunto particular de entradas.
- c) Se basa en la aleatoriedad.
- d) No tiene un resultado predecible.

#### **4. ¿Cuál de las siguientes afirmaciones es correcta con respecto a la "Descomposición" en un algoritmo?**

- a) Es la idea de mantener un problema grande sin dividir.
- b) Consiste en combinar varios problemas pequeños en uno grande.
- c) Implica dividir un problema grande en subproblemas más pequeños y manejables.

d) No tiene relación con la resolución de problemas.

**5. ¿Por qué es importante la "Claridad" en la descripción de un algoritmo?**

- a) Facilita la confusión.
- b) Complica la implementación.
- c) Ayuda en la implementación y comprensión del algoritmo.
- d) No tiene impacto en la comprensión del algoritmo.

**Diseño de Algoritmos utilizando técnicas de representación - Prueba de escritorio**

**Autoevaluación 7**

**1. ¿Cuál de las anteriores figuras se utiliza para asignar datos o procesos a una variable, definiendo la propia variable?**

- a) Datos de entrada
- b) Condicional
- c) Condicional
- d) Acciones o procesos**

**2. ¿Qué concepto se refiere a un lenguaje simple y estructurado que se asemeja al código fuente real, aunque no está vinculado a un lenguaje de programación?**

- a) Pseudocódigo**
- b) Sintaxis real
- c) Código preliminar

d) Prototipo lógico

**3. ¿Qué herramienta es útil para algoritmos que involucran datos tabulares o estructuras bidimensionales?**

a) Diagrama de flujo

b) Gráfico de barras

c) Tablas y matrices

d) Diagrama de datos

**4. ¿Cuál de las siguientes herramientas se utiliza para representar el flujo de control, donde los nodos representan estados o eventos, y las aristas indican transiciones o relaciones?**

a) Matriz de datos

b) Mapa conceptual

c) Diagrama de flujo

d) Grafos

**5. ¿Qué estructura se utiliza para representar jerarquías y relaciones de parentesco entre elementos, siendo útil para estructuras de datos?**

a) Redes neuronales

b) Listas enlazadas

c) Árboles

d) Grafos de flujo

## **Variables y tipos de datos**

Tipos de datos primitivo

## **Autoevaluación 8**

**1. ¿Con qué otro nombre se les llaman a las sentencias que lo componen?**

a) Procesos

b) Entrada

c) Salida

d) Desarrollo

**2. ¿Cuándo se ejecuta?**

a) Después de empezar el programa

b) Antes de empezar el programa

c) No es necesario hacerlo

d) Antes y después

**3. ¿De qué partes se compone?**

a) Inicio, desarrolló y fin

b) Entrada, mecanismo y pantalla

c) Entrada, procesos y salida

d) Entrada y salida

**4. ¿Cuántas pruebas se debe realizar?**

- a) 1 prueba
- b) 3 pruebas o más
- c) Las que sean necesarias
- d) No se necesita pruebas

**5. Al formar la tabla, ¿Cuántas columnas debe haber por cada variable?**

- a) Tantas por instrucciones dadas.
- b) Las que necesitemos.
- c) El mismo número de variables.
- d) Una más que las variables dadas.

**Autoevaluación 10**

**1. ¿Cuántos bytes se utilizan para almacenar un valor del tipo de dato int?**

- a) 2 bytes
- b) 4 bytes
- c) 8 bytes
- d) Varía según el compilador

**2. ¿Cuál es la función del tipo de dato float en programación?**

- a) Almacenar valores enteros con signo
- b) Almacena valores reales en punto flotante de precisión simple

- c) Representa caracteres
- d) Define funciones que no devuelven ningún valor

**3. ¿Cuál es la diferencia principal entre double y float en términos de precisión de los valores almacenados?**

- a) Double tiene el doble de precisión que float
- b) Float tiene el doble de precisión que double
- c) Ambos tienen la misma precisión
- d) Double se utiliza para almacenar caracteres, mientras que float almacena valores reales

**4. ¿Cuántos bytes se utilizan para almacenar un valor booleano (bool)?**

- a) 1 byte
- b) 2 bytes
- c) 4 bytes
- d) Varía según el compilador

**5. ¿Cuál es el propósito del tipo de dato void en programación?**

- a) Almacena valores reales en doble precisión
- b) Representa caracteres
- c) Define funciones que no devuelven ningún valor
- d) Almacenar valores enteros sin signo

**Constantes**

**1. ¿Qué es una constante?**

- a. Son valores fijos y no modificables en la ejecución de un programa.
- b. Son valores fijas y modificables en la ejecución.
- c. No son valores fijos y no modificables en la ejecución de un programa.
- d. Ninguna de las anteriores

**2. ¿Cómo ayuda a distinguir las constantes de las variables?**

- a. No nombra letras mayúsculas y guiones bajos.
- b. Se nombran utilizando letras mayúsculas y guiones bajos para separar palabras
- c. No se nombran caracteres .
- d. Solamente caracteres

**3. ¿Cómo se define en Java para una constante?**

- a. «final»
- b. (fin)
- c. «final-»
- d. “definir”

**4. ¿Cómo se define en C + + para una constante?**

- a. const int
- b. const
- c. int
- d. input

**5. ¿Qué es inmutabilidad en constantes?**

- a. Es un valor fijo que no se altera
- b. Es un valor fijo que si se altera
- c. Valor alterable
- d. Ninguna de las anteriores

## Operadores de Asignación

### Autoevaluación 11

#### 1. ¿Qué es un operador de asignación en programación?

- a) Un operador que realiza comparaciones entre valores.
- b) Un operador que asigna un valor a una variable.
- c) Un operador que realiza operaciones matemáticas.
- d) Un operador para simbolizar cosas.

#### 2. ¿Cuál es la diferencia principal entre el operador de asignación simple ("=") y el operador de igualdad ("==") en muchos lenguajes de programación?

- a) El operador "=" compara dos valores, mientras que "==" asigna un valor a una variable.
- b) El operador "=" asigna un valor a una variable, mientras que "==" compara dos valores.
- c) El operador "=" ingresa un valor a una variable, mientras que "==" compara valores.
- d) Ambos operadores realizan la misma función.

#### 3. ¿Qué hace el operador de asignación compuesta "-=" en programación?

- a) Suma el valor de la variable a otro valor.
- b) Asigna el valor de la variable a otro.
- c) Resta el valor de la variable a otro valor.
- d) Asigna una comparación entre variables.

#### 4. ¿Qué sucede si intentas utilizar un operador de asignación compuesta en una variable que aún no ha sido declarada en el programa?

- a) El programa dará un error.
- b) La variable se inicializará con un valor predeterminado.
- c) El programa funcionará correctamente.
- d) Se asigna un valor automáticamente.

**5. ¿Por qué es importante considerar el tipo de datos al usar operadores de asignación en programación?**

- a) Puede causar errores si se utiliza un operador de asignación incompatible con el tipo de datos de la variable.
- b) No es importante; los operadores de asignación funcionan igual para todos los tipos de datos.
- c) Solo importa en lenguajes de programación específicos.
- d) Porque puede cambiar la dirección de un diagrama de flujo

## Operadores Aritméticos

### Autoevaluación 12

**1. ¿Cuál es la función del operador de suma en programación y cómo se utiliza en un ejemplo con variables?**

- a) Realizar la multiplicación de dos valores.
- b) Restar el segundo valor del primero.
- c) Sumar dos valores.

d) Elevar el primer valor a la potencia del segundo.

**2. Explique la operación realizada por el operador de módulo (%) y proporcione un ejemplo con variables.**

a) Devuelve el resultado de la multiplicación.

b) Devuelve el cociente de la división.

c) Devuelve el resto de la división.

d) Realiza una exponenciación.

**3. ¿Cuál es la función del operador de exponenciación/potencia (o ^) y cómo se utiliza en un ejemplo con variables?**

a) Sumar dos valores.

b) Restar el segundo valor del primero.

c) Elevar el primer valor a la potencia del segundo.

d) Realizar una multiplicación.

**4. Explique la importancia de la jerarquía de operadores en expresiones matemáticas y cómo se puede controlar el orden de evaluación.**

a) La jerarquía no afecta la evaluación de expresiones matemáticas.

b) La jerarquía determina el orden de ejecución de las operaciones.

c) La jerarquía sólo es relevante en lenguajes de programación específicos.

d) La jerarquía sólo afecta a los operadores de suma y resta.

**5. ¿Cómo se utiliza el operador de multiplicación en programación y cuál es su función principal? Proporcione un ejemplo en un lenguaje de programación.**

- a) Se utiliza para realizar la división.
- b) Se utiliza para sumar dos valores.
- c) Se utiliza para multiplicar dos valores.**
- d) Se utiliza para calcular el resto de una división.

### **Operadores Relacionales**

**1. ¿Qué comparan los Operadores Relacionales?**

- a) Números
- b) Datos numéricos**
- c) Letras
- d) Conectores

**2. El operador relacional “ EQ o = “ representa una relación de :**

- a) Igualdad**
- b) Desigualdad
- c) Menor que
- d) Mayor que

**3. El operador relacional “ >< o <> “ representa una relación de :**

- a) Igualdad

b) Desigualdad

c) Menor que

d) Mayor que

**4. El operador relacional “ NE o # “ representa una relación de :**

a) Igualdad

b) Desigualdad

c) Menor que

d) Mayor que

**5. El operador relacional “ GT o >“ representa una relación de :**

a) Igualdad

b) Desigualdad

c) Menor que

d) Mayor que

## Operadores Lógicos

### Autoevaluación 14

**1. ¿Cómo se representa el operador lógico "And" en las codificaciones?**

a) &&

b) ||

c) !

d) And

**2. ¿Cuál es el resultado del operador lógico "Or" si al menos una de las condiciones es verdadera?**

a) False

b) True

c) ||

d) Or

**3. ¿Cuál es la función del operador lógico "Not" en las condiciones?**

a) Invertir el valor booleano

b) Combinar valores booleanos

c) Representar el "And" lógico

d) Excluir opciones

**4. ¿Cómo se representa el operador lógico "Or" en las codificaciones?**

a) &&

b) ||

c) !

d) Or

**5. ¿Qué resultado dará el operador lógico "And" si ambas condiciones son verdaderas?**

a) True

b) False

c) ||

d) And

## Precedencia de los operadores

### Autoevaluación 15

**1. ¿Cuál de los siguientes operadores tiene la mayor precedencia en la mayoría de los lenguajes de programación?**

a) Operadores de asignación (por ejemplo, =)

b) Operadores aritméticos (por ejemplo, +, -, \*, /)

c) Operadores de comparación (por ejemplo, ==, !=, <, >)

d) Operadores lógicos (por ejemplo, &&, ||)

**2. En la expresión  $a = b + c * d$ , ¿Qué operación se realiza primero en la mayoría de los lenguajes de programación?**

a)  $b + c$

b)  $c * d$

c)  $a = b$

d) Ninguna de las anteriores

**3. En la expresión  $a = b == c \& \& d$ , ¿qué operación se realiza primero en la mayoría de los lenguajes de programación?**

a) `b == c`

b) `c && d`

c) `a = b`

d) `a && b`

**4. ¿Cómo puedes cambiar la precedencia de los operadores en una expresión?**

a) Usando el operador !

b) Usando el operador ~

c) Usando paréntesis ()

d) No puedes cambiar la precedencia de los operadores Ninguna de las anteriores

**5. En la expresión `a = b || c && d`, ¿qué operación se realiza primero en la mayoría de los lenguajes de programación?**

a) `b || c`

b) `c && d`

c) `a = b`

d) Ninguna de las anteriores

**Conversión de tipos de datos**

**Autoevaluación 16**

**1. ¿Qué es el casting de datos?**

a) Un proceso para fundir objetos de metal.

b) La creación de moldes para fabricar componentes electrónicos.

c) Un método para convertir un tipo de dato en otro.

d) Una función para realizar cálculos matemáticos.

**2. En Python, ¿cuál de las siguientes afirmaciones describe mejor el casting implícito?**

a) Es necesario especificar explícitamente la conversión de tipos de datos.

b) Es una característica exclusiva de Python 2.

c) Sólo se aplica a lenguajes de programación específicos.

d) La conversión de tipos de datos se realiza automáticamente por el intérprete.

**3. En Python, ¿cuál es la función utilizada para el casting explícito de enteros?**

a) convertir entero()

b) int()

c) to\_int()

d) cast\_int()

**4. ¿Cuál es el riesgo asociado con el casting de datos?**

a) Pérdida de información o posible error si la conversión no es válida.

b) No hay riesgos, ya que todas las conversiones son seguras.

c) El casting siempre se realiza correctamente sin riesgos.

d) Sólo se aplica a lenguajes de programación antiguos.

**5. En Python, ¿en qué situación se utiliza comúnmente el casting de datos?**

- a) Cuando se necesita fundir metales en una aplicación.
- b) Para convertir datos entre diferentes tipos en un programa.**
- c) Sólo en situaciones de emergencia.
- d) Exclusivamente en operaciones de entrada y salida

## **Entrada y salida de datos**

### **Autoevaluación 17**

- 1. ¿Qué es la entrada de datos en programación?**
  - a. El proceso de enviar información a una impresora.
  - b. La acción de recibir información desde el usuario o algún dispositivo externo.**
  - c. El intercambio de datos entre variables dentro de un programa.
  - d. La transmisión de datos a través de una red de computadoras.
- 2. ¿Cuál de las siguientes NO es una forma común de entrada de datos en programación?**
  - a. Teclado
  - b. Ratón
  - c. Pantalla**
  - d. Micrófono
- 3. ¿Qué función se utiliza en gran medida para recibir datos del usuario en lenguajes de programación como Python?**
  - a. output()
  - b. input()**
  - c. read()
  - d. get user\_input()
- 4. ¿Cuál de las siguientes NO es una forma de salida de datos en programación?**

- a. Impresión en consola
- b. Escritura en un archivo
- c. Envío de datos a través de una red
- d. **Lectura de datos desde un sensor**

**5. ¿Qué representa el concepto de "stream" en la entrada/salida de datos en programación?**

- a. **Un flujo continuo de datos que se lee o escribe secuencialmente.**
- b. Una estructura de datos en forma de árbol que organiza la entrada y salida.
- c. Una conexión de red para transmitir datos en tiempo real.
- d. Un tipo de variable utilizado para almacenar información de entrada y salida.

## Estructuras de Control

### Autoevaluación 19

**1. ¿Cuál es la función principal del bucle for en programación según la definición proporcionada?**

- a) Realizar operaciones aritméticas.
- b) Iterar sobre una secuencia de elementos.
- c) Ejecutar bloques de código de manera condicional.
- d) **Definir estructuras de datos iterables.**

**2. ¿Para qué se utiliza el bucle for en Python, según su propósito principal?**

- a) Realizar acciones únicas.
- b) Ejecutar un bloque de código de manera infinita.
- c) **Automatizar acciones repetitivas sobre cada elemento de una secuencia.**

d) Definir estructuras de control de flujo.

**3. ¿Cuál es la sintaxis general del bucle for en Python y qué representa la variable y el iterable en esta sintaxis?**

a) for item in lista: - "ítem" es la secuencia a iterar.

b) for variable in secuencia: - "variable" toma el valor de cada elemento durante la iteración.

c) for each elemento in rango: - elemento es la variable de control de iteración.

d) for i from 1 to 10: - "i" es la variable de control y "1 to 10" es la secuencia.

**4. ¿Cuáles son características del bucle for, según la información proporcionada?**

a) No es compatible con tipos de datos diferentes a listas.

b) Ejecuta hasta que la secuencia sea infinita.

c) Su sintaxis es compleja y difícil de entender.

d) Se utiliza principalmente para iterar sobre secuencias, es finito, fácil de usar y compatible con varios tipos de datos.

**5. ¿Qué se destaca como una finalidad del bucle for, según la información proporcionada?**

a) Realizar tareas únicas en el código.

b) Automatizar tareas que requieren la repetición de un bloque de código para cada elemento en una secuencia.

c) Iterar sobre estructuras condicionales.

d) Limitar su uso a la generación de secuencias numéricas.

## **Solucionario Manual 2**

### **Subprogramas : Funciones o Procedimientos**

#### **Definición de Subprograma**

##### **1. ¿Qué es un subprograma en programación?**

- a) Una variable global
- b) Una función o procedimiento
- c) Un comentario en el código
- d) Un bucle for

##### **2. ¿Cuál es la principal función de un subprograma?**

- a) Declarar variables
- b) Realizar cálculos matemáticos
- c) Agrupar y reutilizar código
- d) Imprimir mensajes en la consola

##### **3. ¿Cuál es una característica clave de los subprogramas?**

- a) No pueden aceptar parámetros
- b) Solo pueden devolver valores numéricos
- c) Pueden ser llamados y reutilizados en el programa principal
- d) Siempre deben estar ubicados al final del código

##### **4. ¿Qué es un parámetro en el contexto de los subprogramas?**

- a) Una variable local
- b) Un valor constante
- c) Una entrada proporcionada al subprograma
- d) Una condición de bucle

**5. ¿Qué beneficio ofrece el uso de subprogramas en el desarrollo de software?**

- a) Mayor complejidad en el código
- b) Menor modularidad
- c) Facilita la reutilización y mantenimiento del código
- d) Aumenta la dificultad de depuración

**Tipos de subprogramas**

**1. ¿Cuál es la principal diferencia entre una función y un procedimiento en programación?**

- a) Las funciones devuelven un valor, los procedimientos no.
- b) Los procedimientos devuelven un valor, las funciones no.
- c) Ambos siempre devuelven un valor.
- d) Ambos nunca devuelven un valor.

**2. ¿Qué tipo de subprograma se asocia comúnmente con la programación orientada a objetos?**

- a) Funciones
- b) Procedimientos

c) Métodos

d) Subrutinas

**3. ¿Qué hace que una función lambda sea diferente de una función convencional?**

a) Las funciones lambda no pueden aceptar parámetros.

b) Las funciones lambda pueden devolver múltiples valores.

c) Las funciones lambda son funciones anónimas.

d) Las funciones lambda no pueden contener bucles.

**4. ¿Qué caracteriza a una subrutina en programación?**

a) Puede devolver un valor y aceptar parámetros.

b) Siempre devuelve un valor pero no acepta parámetros.

c) Siempre acepta parámetros pero no devuelve un valor.

d) No puede devolver un valor ni aceptar parámetros.

**5. ¿Cuál es uno de los beneficios principales de utilizar subprogramas en programación?**

a) Incrementan la complejidad del código.

b) Facilitan el mantenimiento y la reutilización del código.

c) Sólo son útiles en programas pequeños.

d) Sólo se utilizan en casos específicos de programación.

## **Declaración , implementación y llamada de Subprogramas**

**1. ¿Qué aspecto de un subprograma se define durante la fase de declaración?**

- a. Código interno
- b. Nombre y parámetros**
- c. Tipo de retorno
- d. Sintaxis

**2. ¿Cuál es el propósito de la declaración de un subprograma?**

- a. Definir el código interno
- b. Especificar la firma y la interfaz**
- c. Realizar la llamada
- d. Proporcionar la sintaxis de implementación

**3. ¿En qué fase se define el código interno del subprograma?**

- a. Declaración
- b. Llamada
- c. Implementación**
- d. Sintaxis

**4. ¿Qué información incluye la declaración de un subprograma?**

- a. Código interno y sintaxis
- b. Nombre, parámetros y tipo de retorno**
- c. Llamadas y sintaxis
- d. Implementación y declaración

**5. ¿Cuál es el propósito de la llamada de subprogramas?**

- a. Definir la interfaz

- b. Proporcionar la sintaxis
- c. Invocar o utilizar la función
- d. Especificar el tipo de retorno

## Argumentos y Parámetros

1. ¿Qué representa un parámetro, en un procedimiento en programación?
  - a. Un valor pasado al llamar el procedimiento
  - b. Una variable local del procedimiento
  - c. Una lista de argumentos
  - d. Un mecanismo de paso opcional
2. ¿Qué se especifica al definir un procedimiento Function o Sub en cuanto a los parámetros?
  - a. Solo el nombre del parámetro
  - b. Solo el tipo de datos del parámetro
  - c. Nombre, tipo de dato y mecanismo de paso del parámetro
  - d. Solo el mecanismo de paso del parámetro
3. ¿Cuál es la función de un argumento?
  - a. Representar un valor esperado en la declaración
  - b. Actuar como una variable local
  - c. Ser un tipo de dato definido para el parámetro
  - d. Representar el valor pasado

**4. ¿Qué caracteriza a los argumentos por posición, en la llamada?**

- a. Tiene nombres asignados
- b. Se especifican entre paréntesis
- c. Corresponden a la posición en la lista de parámetros
- d. Pueden contener cero o más variables

**5. ¿En qué consisten los argumentos por nombre al llamar a un procedimiento?**

- a. No es una forma válida de llamar a un parámetro
- b. Los nombres de los argumentos no importan
- c. Corresponden a la posición en la lista de parámetros
- d. Se especifica el nombre y el tipo de dato de cada argumento

**Ámbito de las variables**

**1. ¿Qué es una variable en programación?**

- a. Un valor que no puede cambiar.
- b. Un valor que puede modificarse a lo largo de un proceso.
- c. Una función que se puede llamar en cualquier momento.
- d. Un tipo de dato que solo puede ser numérico.

**2. ¿Cuándo existe una variable en lenguajes como Python o C++?**

- a. Solo después de que se declara y antes de que se utilice.
- b. Solo mientras se está ejecutando el bloque de código en el que se declaró.

c. Desde el inicio hasta el final del programa, independientemente de dónde se declare.

d. Solo cuando se asigna un valor.

**3. ¿Qué es una variable global?**

a. Una variable que solo puede utilizarse en la función en la que se declaró.

**b. Una variable que se puede utilizar en cualquier parte del programa.**

c. Una variable que solo puede tener valores numéricos.

d. Una variable que solo puede tener valores de cadena.

**4. ¿Qué es una variable local?**

**a. Una variable que solo puede utilizarse en la función en la que se declaró.**

b. Una variable que se puede utilizar en cualquier parte del programa.

c. Una variable que solo puede tener valores numéricos.

d. Una variable que solo puede tener valores de cadena.

**5. En JavaScript, ¿qué sucede con las variables después de que se ejecuta su bloque de código?**

**a. Las variables se eliminan inmediatamente después de que se ejecuta su bloque de código.**

b. Las variables se guardan y se pueden utilizar en cualquier parte del programa.

c. Las variables se convierten en variables globales.

d. Las variables se convierten en constantes y no pueden modificarse.

## **Funciones de librerías o módulos**

**1. ¿Cuál es el propósito principal de las funciones de librerías o módulos en programación?**

- a) Generar código aleatorio.
- b) Realizar tareas específicas sin tener que volver a escribir ese código.**
- c) Depurar errores en el código.
- d) Escribir código desde cero en cada proyecto.

**2. ¿Qué librería de Python se utiliza principalmente para operaciones numéricas y matriciales?**

- a) Pandas
- b) Numpy**
- c) Matplotlib
- d) Seaborn

**3. ¿Cuál de las siguientes librerías se utiliza para manipulación y análisis de datos en Python?**

- a) Numpy
- b) Pandas**
- c) Matplotlib
- d) Flask

**4. ¿Para qué se utiliza principalmente la librería Scikit-learn en Python?**

- a) Desarrollo web
- b) Aprendizaje automático y minería de datos**
- c) Visualización de datos
- d) Manipulación de archivos

**5. ¿Cuál de las siguientes afirmaciones es verdadera sobre las librerías o módulos en programación?**

- a) Siempre deben ser desarrolladas por el mismo equipo que creó el lenguaje de programación.
- b) No se pueden utilizar para realizar operaciones matemáticas.
- c) Pueden ser estándar (incluidos con el lenguaje de programación) o externos (desarrollados por terceros).**
- d) Son útiles sólo para la depuración de código.

**Recursividad**

**1.- ¿Qué es la recursividad en Python?**

- a) Un bucle que se repite hasta que se cumple una condición.
- b) Una técnica donde una función se llama a sí misma.**
- c) Un método para evitar el uso de funciones.
- d) Una operación de asignación de variables.

**2.- ¿Cuál es la condición base en una función recursiva?**

- a) Es una condición que se cumple al principio de la función.

b) Es una condición que determina cuántas veces se llama la función recursiva.

c) Es una condición que se utiliza para detener la recursión.

d) Es una condición que se utiliza para iniciar la recursión.

**3.- ¿Cuál es un ejemplo típico de un problema que se puede resolver de manera recursiva en Python?**

a) Suma de números pares.

b) Búsqueda binaria en una lista ordenada.

c) Ordenamiento de una lista.

d) Concatenación de cadenas.

**4.- ¿Cuál es un posible inconveniente de utilizar la recursividad en Python?**

a) Mayor eficiencia en el uso de la memoria.

b) La recursividad no es compatible con Python.

c) Posibilidad de causar un desbordamiento de pila si no se maneja correctamente.

d) Facilidad de implementación en comparación con los bucles iterativos.

**5.- ¿Cuál es el caso base en la recursividad para calcular el factorial de un número en Python?**

a) Cuando el número es cero (0).

b) Cuando el número es uno (1).

c) Cuando el número es positivo.

- d) Cuando el número es mayor que uno (1).

### **Creación de librerías o módulos**

**1.- ¿Cuál es el propósito principal de crear un archivo `__init__.py` en una librería de Python?**

- a) Incluir la documentación del proyecto.
- b) Inicializar el paquete y permitir la importación de módulos.**
- c) Configurar las dependencias del proyecto.
- d) Definir las pruebas unitarias

**2.- ¿En qué parte de la estructura del proyecto se encuentran las pruebas unitarias en una librería de Python?**

- a) En la carpeta `mi_libreria/`.
- b) En la carpeta `tests/`.**
- c) En el archivo `setup.py`.
- d) En el archivo `README.md`.

**3.- ¿Cuál es el propósito del archivo `setup.py` en el desarrollo de una librería de Python?**

- a) Ejecutar las pruebas unitarias.
- b) Definir el código principal de la librería.
- c) Inicializar el paquete.**
- d) Documentar el proyecto.

**4.- ¿Cómo se puede instalar una librería de Python después de publicarla en PyPI?**

- a) Descargando un archivo ZIP desde GitHub.
- b) Copiando y pegando el código fuente en un proyecto.
- c) Utilizando el comando `pip install nombre_libreria`.**
- d) Creando un entorno virtual desde cero.

**5.- ¿Cuál es un aspecto clave al mantener una librería de Python?**

- a) Publicarla en PyPI solo una vez.**

- b) Ignorar problemas reportados por la comunidad.
- c) Mantenerla actualizada y responder a contribuciones.
- d) Cambiar la estructura del proyecto frecuentemente.

## Arreglos

### Arreglos unidimensionales

**1.- ¿Cómo se representa un arreglo unidimensional en términos de dimensiones?**

- a. Como un conjunto de filas y columnas
- b. Como una línea
- c. Como un conjunto de capas
- d. Como una matriz

**2.- ¿Cómo se define un arreglo unidimensional en Python indicando su tamaño?**

- a. list = [1, 2, 3]
- b. array(1, 2, 3)
- c. arr[1, 2, 3]
- d. array = [1][2][3]

**3.- ¿Cuál es la función del índice en un arreglo unidimensional en Python?**

- a. Representa la dimensión del arreglo
- b. Indica el tipo de datos del arreglo
- c. Determina la posición del elemento en el conjunto
- d. Define el nombre del arreglo

**4.- ¿Cómo se accede a un elemento específico en un arreglo unidimensional en Python?**

- a. array[elemento]
- b. array.elemento

c. array[index]

d. array(posición)

**5.- ¿Cuál es el propósito principal de un arreglo unidimensional en programación?**

a. Almacenar datos de forma aleatoria

b. Representar datos en formato de tabla

c. Almacenar y manipular conjuntos de datos de manera eficiente

d. Visualizar datos en un gráfico de barras

## Arreglos bidimensionales

**1.- ¿Cómo se crea un arreglo bidimensional en Python?**

a) Utilizando la función array() del módulo numpy.

b) Declarando una lista de listas.

c) Aplicando la función matrix() del módulo numpy.

d) Utilizando la función grid() del módulo numpy.

**2.- ¿Cuál es la sintaxis para acceder al elemento en la segunda fila y tercera columna de un arreglo bidimensional en Python?**

01. array[2, 3]

**02. array[1][2]**

03. array[3][2]

04. array[2][3]

**3.- ¿Cómo se inicializa un arreglo bidimensional de tamaño 3x3 con todos sus elementos como cero?**

a) [[0] \* 3] \* 3

b) [[0] \* 3]

c) `[0] * 3`

d) `[] * 3]`

**4.- ¿Qué método se puede utilizar para recorrer todos los elementos de un arreglo bidimensional en Python?**

a) `for i in range(len(array))`

b) `for i in array:`

c) `for i in range(len(array[0]))`

d) `for i in range(len(array)):`

**5.- ¿Cuál es una ventaja de utilizar arreglos bidimensionales en Python?**

a) Mayor complejidad en el manejo de datos.

b) Estructura más organizada para almacenar datos en forma de tabla.

c) Mayor dificultad para acceder a elementos específicos.

d) Imposibilidad de realizar operaciones matemáticas con los datos almacenados.

**Arreglos multidimensionales**

**1. ¿Qué es un arreglo multidimensional en programación?**

a) Una estructura de datos que solo puede contener números enteros.

b) Una estructura de datos que puede contener elementos organizados en varias dimensiones.

c) Una función en Python para manipular cadenas de texto.

d) Un tipo de bucle utilizado para iterar sobre una lista en Python.

**2. ¿Cuál de las siguientes librerías de Python es ampliamente utilizada para trabajar con arreglos multidimensionales?**

a) Matplotlib

b) Pandas

c) NumPy

d) Scikit-learn

**3. ¿Cuál de las siguientes NO es una dimensión común en un arreglo multidimensional?**

a) Altura

b) Ancho

c) Tiempo

d) Peso

**4. ¿Qué operación se utiliza para acceder a un elemento específico en un arreglo multidimensional en Python?**

a) get()

b) access()

c) []

d) ()

**5. ¿Qué método se utiliza para crear un arreglo multidimensional en NumPy?**

a) array()

b) matrix()

c) multidimensional()

d) tensor()

### Paso de Arreglos a funciones

**1. ¿Qué aspecto de un subprograma se define durante la fase de declaración?**

a. Código interno

b. Nombre y parámetros

c. Tipo de retorno

d. Sintaxis

**2. ¿Cuál es el propósito de la declaración de un subprograma?**

a. Definir el código interno

b. Especificar la firma y la interfaz

c. Realizar la llamada

d. Proporcionar la sintaxis de implementación

**3. ¿En qué fase se define el código interno del subprograma?**

a. Declaración

b. Llamada

c. Implementación

d. Sintaxis

**4. ¿Qué información incluye la declaración de un subprograma?**

a. Código interno y sintaxis

b. Nombre, parámetros y tipo de retorno

c. Llamadas y sintaxis

d. Implementación y declaración

**5. ¿Cuál es el propósito de la llamada de subprogramas?**

a. Definir la interfaz

b. Proporcionar la sintaxis

c. Invocar o utilizar la función

d. Especificar el tipo de retorno

**Algoritmos de Ordenación (selección, intercambio y burbuja)**

**1. ¿Cuál de los siguientes algoritmos de ordenación divide la lista en dos partes, ordenada y no ordenada, y busca el elemento mínimo en la parte no ordenada en cada iteración?**

A) Bubble Sort

B) Insertion Sort

C) Selection Sort

D) Merge Sort

**2. En el algoritmo de burbuja, ¿cómo se comparan y ordenan los elementos?**

A) Se compara cada elemento con el primero y se intercambian si son diferentes.

B) Se compara cada elemento con el siguiente y se intercambian si están en el orden incorrecto.

C) Se compara cada elemento con el último y se intercambian si son mayores.

D) Se compara cada elemento con el mínimo y se intercambian si son menores.

**3. ¿Cuál de los siguientes algoritmos de ordenación es conocido por su proceso de "burbujeo" donde los elementos más grandes se desplazan hacia el final de la lista?**

A) Selection Sort

B) Insertion Sort

**C) Bubble Sort**

D) QuickSort

**4. ¿Qué característica comparten los algoritmos de Selection Sort, Bubble Sort y Insertion Sort en términos de eficiencia?**

A) Complejidad logarítmica  $O(\log n)$

B) Complejidad lineal  $O(n)$

**C) Complejidad cuadrática  $O(n^2)$**

D) Complejidad constante  $O(1)$

**5. En el algoritmo de Insertion Sort, ¿cómo se construye la lista ordenada?**

A) Comparando elementos adyacentes y realizando intercambios.

B) Seleccionando el elemento mínimo en cada iteración.

**C) Construyendo la lista uno a uno, colocando cada elemento en su posición correcta.**

D) Utilizando una estructura de datos de cola.

### **Algoritmos de búsqueda.**

**1.- ¿Qué tipo de algoritmo de búsqueda se utiliza comúnmente para buscar elementos en grandes conjuntos de datos ordenados?**

- a) Búsqueda Binaria
- b) Búsqueda Lineal
- c) Búsqueda de Profundidad
- d) Búsqueda de Amplitud

**2.- ¿Qué algoritmo de búsqueda es más eficiente para encontrar una solución óptima en un grafo o árbol?**

- a) Búsqueda Lineal
- b) Búsqueda de Amplitud
- c) Búsqueda de Profundidad
- d) Algoritmos de Búsqueda Heurística

**3.- ¿Cómo se llama el algoritmo de búsqueda que explora todos los nodos vecinos al mismo nivel antes de avanzar al siguiente nivel?**

- a) Búsqueda Binaria
- b) Búsqueda de Profundidad
- c) Búsqueda de Amplitud
- d) Búsqueda Lineal

**4.- ¿Qué tipo de algoritmo de búsqueda utiliza información adicional sobre el problema para guiar la búsqueda hacia una solución óptima de manera más eficiente?**

- a) Búsqueda de Profundidad
- b) Búsqueda Binaria

- c) Búsqueda de Amplitud
- d) Algoritmos de Búsqueda Heurística

**5.- ¿Qué algoritmo de búsqueda es adecuado para buscar en colecciones de datos ordenadas y solo se puede aplicar a estos conjuntos?**

- a) Búsqueda de Profundidad
- b) Búsqueda Lineal
- c) Búsqueda Binaria
- d) Búsqueda de Amplitud

## Cadenas y/o Strings

### **Autoevaluación 1**

**1. ¿Qué es una cadena en programación?**

- a) Una secuencia de números
- b) Una secuencia de caracteres
- c) Una secuencia de palabras
- d) Una secuencia de operaciones

**2. ¿Cuál de las siguientes afirmaciones sobre las cadenas es verdadera?**

- a) Las cadenas son mutables, lo que significa que se pueden modificar directamente.
- b) Las cadenas son inmutables, lo que significa que no se pueden modificar directamente.
- c) Las cadenas solo pueden contener letras y números.

- d) Las cadenas son tipos de datos numéricos.

**3. ¿Cómo se delimitan las cadenas en muchos lenguajes de programación?**

- a) Con corchetes [ ]
- b) Con paréntesis ( )
- c) Con comillas simples ''
- d) Con llaves { }

**4. ¿Qué operación se utiliza para unir dos cadenas en una sola?**

- a) Separación
- b) Unificación
- c) Concatenación
- d) Agregación

**5. ¿Cómo se accede al primer carácter de una cadena en la mayoría de los lenguajes de programación?**

- a) Con el índice 0
- b) Con el índice 1
- c) Con el índice -1
- d) No se puede acceder al primer carácter

**Declaración e inicialización de variables**

## **Autoevaluación 2**

**1. ¿Qué es la declaración de variables?**

- a) Reservar espacio en la memoria para almacenar un valor.
- b) Asignar un valor inicial a la variable.
- c) Especificar el tipo y el nombre de la variable.
- d) Ninguna de las anteriores.

**2. ¿Qué es la inicialización de variables?**

- a) Reservar espacio en la memoria para almacenar un valor.
- b) Asignar un valor inicial a la variable.
- c) Especificar el tipo y el nombre de la variable.
- d) Ninguna de las anteriores.

**3. ¿Cuál es el signo utilizado para la inicialización de variables?**

- a) +
- b) -
- c) \*
- d) =

**4. ¿Cuál es el objetivo de la declaración e inicialización de variables?**

- a) Crear y asignar valores a variables en un programa.
- b) Reservar espacio en la memoria para almacenar un valor.

- c) Asignar un valor inicial a la variable.
- d) Especificar el tipo y el nombre de la variable.

**5. ¿Por qué es importante asignar un nombre único y significativo a una variable?**

- a) Para que ocupe menos espacio en la memoria.
- b) **Para evitar conflictos y hacer el código más legible.**
- c) Para que el programa se ejecute más rápido.
- d) Para que la variable pueda cambiar de tipo dinámicamente.

**Entrada y salida**

**Autoevaluación 3**

**1. ¿Qué función se utiliza para capturar la entrada del usuario en Python?**

- a) output()
- b) enter()
- c) **input()**
- d) capture()

**2. ¿Cómo se imprime una cadena de texto en la consola en Python?**

- a) console("Hola Mundo!")
- b) echo("Hola Mundo!")
- c) **print("Hola Mundo!")**
- d) output("Hola Mundo!")

**3. ¿Cómo se accede al primer carácter de una cadena de texto en Python?**

a) texto[1]

b) **texto[0]**

c) texto[-1]

d) texto[2]

**4. ¿Cómo se accede al último carácter de una cadena de texto en Python?**

a) texto[1]

b) texto[0]

c) **texto[-1]**

d) texto[2]

**5. ¿Qué tipo de dato devuelve la función input() en Python?**

a) int

b) float

c) **str**

d) bool

## Asignación

### *Autoevaluación 4*

**1. ¿Qué es la asignación en programación?**

a) Almacenar un valor en una variable

- b) Realizar operaciones aritméticas
- c) Comparar dos valores
- d) Eliminar una variable

2. **¿Qué operador se utiliza comúnmente para realizar asignaciones en muchos lenguajes de programación?**

- a) "+"
- b) "="**
- c) "=="
- d) "->"

3. **¿Cuál de los siguientes NO es un tipo de dato que puede ser asignado a una variable?**

- a) Números enteros
- b) Funciones**
- c) Listas
- d) Cadenas de texto

4. **¿Qué representa el signo "=" en programación en el contexto de asignación?**

- a) Igualdad matemática
- b) Asignación de valores**
- c) Comparación de valores
- d) Ninguna de las anteriores

5. **¿Cuál es una forma común de realizar asignaciones condicionales en muchos lenguajes de programación?**

- a) Utilizando la palabra clave "if"**
- b) Utilizando la palabra clave "else"
- c) Utilizando el operador ternario
- d) Utilizando la palabra clave "for"

**Longitud y concatenación**

## **Autoevaluación 5**

### **1. ¿Qué es la longitud de una cadena en Python?**

- a) La cantidad de espacios que ocupa la cadena en memoria
- b) El número exacto de caracteres que contiene la cadena
- c) El número de palabras dentro de la cadena
- d) La posición del último carácter en la cadena

### **2. ¿Cómo se puede obtener la longitud de una cadena en Python?**

- a) Utilizando la función length()
- b) Con el método count()
- c) Mediante la función size()
- d) Utilizando la función len()

### **3. ¿Cuál es la salida del siguiente código en Python?**

```
cadena = "¡Hola"
```

```
cadena += " mundo!"
```

```
print(cadena)
```

- a) ¡Hola mundo!

- b) ¡Hola+mundo!

- c) ¡Hola , mundo!

- d) Hola mundo!

**4. ¿Cuál es la forma preferida y más eficiente de concatenar las palabras de una lista en Python?**

- a) Utilizando el método concat()
- b) Utilizando el método append()
- c) Utilizando el método join()
- d) Utilizando el método extend()

**5. ¿Cuál es la función de la función join() en Python?**

- a) Dividir una cadena en una lista de palabras
- b) Concatenar cadenas de manera efectiva
- c) Contar el número de caracteres en una cadena
- d) Dividir una cadena en subcadenas basadas en un delimitador

### **Comparación**

#### ***Autoevaluación 6***

**1. ¿Cuál de las siguientes opciones compara correctamente dos cadenas en Python?**

- a) cadena1 == cadena2
- b) cadena1.equals(cadena2)
- c) cadena1.compare(cadena2)
- d) cadena1.isEqual(cadena2)

**2. ¿Cómo se puede verificar si una cadena es un subconjunto de otra cadena en Python?**

- a) cadena1.contains(cadena2)
- b) cadena1.isSubset(cadena2)
- c) cadena1 in cadena2
- d) cadena1.subset(cadena2)

**3. ¿Cuál de las siguientes opciones devuelve True si la cadena1 es lexicográficamente mayor que la cadena2?**

- a) cadena1 > cadena2
- b) cadena1 >= cadena2
- c) cadena1 < cadena2
- d) cadena1 <= cadena2

**4. ¿Cómo se puede convertir una cadena a minúsculas en Python?**

- a) cadena.lower()
- b) cadena.toLowerCase()
- c) cadena.toLowerCase()
- d) cadena.minusculas()

**5. ¿Cómo se puede dividir una cadena en Python?**

- a) cadena.split()

b) cadena.divide()

c) cadena.break()

d) cadena.separate()

### **Conversión**

#### **Autoevaluación 7**

1. ¿Qué función se utiliza para convertir un entero a una cadena?

a) int()

b) str()

c) float()

d) list()

2. ¿Cuál es la función para convertir una cadena a un entero?

a) int()

b) str()

c) float()

d) list()

3. ¿Qué se utiliza para convertir una lista a una tupla?

a) list()

b) tuple()

c) set()

d) dict()

4. ¿Cuál de las siguientes opciones convierte un flotante a un entero?

a) int()

b) str()

c) float()

d) list()

5. ¿Qué función se utiliza para convertir una cadena a flotante?

a) int()

b) str()

c) float()

d) list()

### **Inversión**

#### **Autoevaluación 8**

##### **1. ¿Cómo se invierte una cadena en Python?**

a) cadena.reverse()

b) cadena.invert()

c) cadena[::-1]

d) cadena.flip()

##### **2. ¿Qué hace el slicing cadena[::-1] en Python?**

a) Elimina todos los caracteres de la cadena.

b) Convierte todos los caracteres de la cadena a mayúsculas.

c) Invierte el orden de los caracteres en la cadena.

d) Repite la cadena en orden inverso.

##### **3. ¿Las cadenas en Python son mutables o inmutables?**

a) Mutables

b) Inmutables

c) Depende de la cadena

d) Ninguna de las anteriores

**4. ¿Qué devuelve cadena[::-1] si cadena = "Hola Mundo"?**

a) "Hola Mundo"

b) "odnuM aloH"

c) "Mundo Hola"

d) "aloH odnuM"

**5. ¿Cómo se puede verificar si una cadena es un palíndromo en Python?**

a) cadena == cadena.reverse()

b) cadena == cadena.invert()

c) cadena == cadena[::-1]

d) cadena == cadena.flip()

**Sub cadenas**

***Autoevaluación 9***

**1. ¿Qué es una subcadena en Python y por qué es útil en la programación?**

a) Una subcadena en Python es una cadena que contiene solo caracteres numéricos. Es útil en la programación para realizar operaciones matemáticas específicas.

b) Una subcadena en Python es una cadena que contiene solo caracteres alfabéticos. Es útil en la programación para manipular y extraer partes específicas de una cadena más grande.

- c) Una subcadena en Python es una cadena que contiene solo caracteres especiales como símbolos y puntuación. Es útil en la programación para el formateo de texto y la presentación de datos.
- d) Una subcadena en Python es una cadena que contiene solo caracteres de control como tabulaciones y saltos de línea. Es útil en la programación para el manejo de la entrada y salida de datos en archivos.

**2. ¿Cuál es la diferencia entre indexación y rebanado (slicing) al trabajar con subcadenas en Python?**

- a) La indexación se refiere a la obtención de un único carácter de una cadena utilizando su posición, mientras que el rebanado (slicing) se refiere a la obtención de una parte más grande de una cadena utilizando rangos de índices.
- b) La indexación se utiliza para dividir una cadena en partes más pequeñas, mientras que el rebanado (slicing) se utiliza para acceder a caracteres individuales en una cadena.
- c) La indexación se utiliza para acceder a subcadenas en una cadena, mientras que el rebanado (slicing) se utiliza para realizar operaciones aritméticas en cadenas.
- d) La indexación se utiliza para agregar nuevos caracteres a una cadena, mientras que el rebanado (slicing) se utiliza para eliminar caracteres de una cadena.

**3. ¿Qué hace la expresión cadena[::-1] en Python?**

- a) Invierte la cadena, es decir, devuelve la cadena en orden inverso.
- b) Elimina todos los caracteres de la cadena que no sean letras.
- c) Divide la cadena en palabras individuales.

d) Convierte todos los caracteres de la cadena en mayúsculas.

**4. ¿Cuál es la diferencia entre cadena[2] y cadena[2:3] al trabajar con subcadenas en Python?**

a) cadena[2] devuelve un solo carácter en el índice 2, mientras que cadena[2:3] devuelve una subcadena que incluye el carácter en el índice 2.

b) cadena[2] devuelve una subcadena que incluye el carácter en el índice 2, mientras que cadena[2:3] devuelve un solo carácter en el índice 2.

c) Ambas expresiones devuelven subcadenas del mismo tamaño, pero cadena[2] incluye el carácter en el índice 2 y cadena[2:3] no lo incluye.

d) Ambas expresiones devuelven subcadenas del mismo tamaño, pero cadena[2:3] incluye el carácter en el índice 2 y cadena[2] no lo incluye.

**5. ¿Qué expresión en Python se puede utilizar para obtener una subcadena que incluya todos los caracteres desde el segundo hasta el último, omitiendo el primer carácter?**

a) cadena[1:]

b) cadena[:-1]

c) cadena[2:]

d) cadena[1:-1]

**Búsqueda**

**Autoevaluación 10**

**1. ¿Qué no hace el método index(sub [, pos\_inicial, pos\_final]) en Python?**

- a) Busca una subcadena dentro de otra cadena.
- b) Devuelve el índice de la primera ocurrencia de la subcadena.
- c) Lanza una excepción ValueError si la subcadena no se encuentra.
- d) Concatena dos cadenas.**

**2. ¿Cuál es el propósito de los parámetros posicion\_inicial y posicion\_final en el método find()?**

- a) Especificar la subcadena a buscar.
- b) Definir la posición inicial y final de la búsqueda dentro de la cadena.**
- c) Controlar el formato de salida del resultado.
- d) Convertir la cadena a minúsculas.

**3. ¿Qué valor devuelve el método index() si la subcadena no se encuentra dentro del rango especificado?**

- a) -1
- b) 0
- c) Lanza una excepción ValueError**
- d) None

**4. ¿Qué valor devuelve el método find() si la subcadena no se encuentra dentro de la cadena?**

- a) -1**
- b) 0
- c) Lanza una excepción ValueError
- d) None

**5. ¿Cuál es la diferencia principal entre los métodos find() e index() en Python?**

- a) `find()` devuelve -1 si la subcadena no se encuentra, mientras que `index()` lanza una excepción `ValueError`.
- b) `find()` lanza una excepción `ValueError` si la subcadena no se encuentra, mientras que `index()` devuelve -1.
- c) No hay diferencia, ambos métodos hacen lo mismo.
- d) Ninguna de las anteriores.

### **Cadenas y/o strings como parámetros de funciones**

#### **Autoevaluación 11**

1. **Pregunta: ¿Cómo se define una función en Python que toma una cadena como parámetro?**  
  

a) `definir_funcion(cadena):`

**b) `def funcion(cadena):`**

c) `crear_funcion(cadena):`

d) `cadena_funcion(cadena):`
2. **Pregunta: ¿Cómo puedes concatenar dos cadenas en Python?**  
  

a) `concat(cadena1, cadena2)`

b) `cadena1.join(cadena2)`

**c) `cadena1 + cadena2`**

d) `unir_cadenas(cadena1, cadena2)`
3. **Pregunta: ¿Cuál es la característica principal de las cadenas en Python en términos de modificación?**  
  

a) Son mutables y se pueden modificar directamente.

**b) Son inmutables y no se pueden modificar directamente.**

c) Solo se pueden modificar mediante bucles for.

d) Pueden modificarse en cualquier momento.

**4. Pregunta: ¿Qué método se utiliza para dividir una cadena en Python?**

**a) split()**

b) divide()

c) separar()

d) partir()

**5. Pregunta: ¿Cómo puedes invertir una cadena en Python utilizando una función?**

a) cadena.reverse()

b) invertir\_cadena(cadena)

**c) cadena[::-1]**

d) voltear\_cadena(cadena)

### **Introducción a tipos de datos abstractos(TDA)**

#### **Autoevaluación 12**

**1. ¿ Qué significa TDA ?**

**a) Tipo de datos abstractos**

b) Datos de tipo boolean

c) Tipo de datos random

d) Una función que responde a mi necesidad

**2. ¿ Que incluye el TDA ?**

- a) Nombre del TDA, Operaciones, Invariantes, Semántica
- b) Nombre del TDA, Invariantes, Funciones
- c) Def, Import, Int, Input
- d) Funciones varias

### 3. ¿ Cuántos tipos de datos existen?

- a) 4
- b) 5
- c) 6
- d) 3

### 4. ¿ Que es la declaración TDA ?

- a) La declaración de un TDA se centra en la descripción de las operaciones que se pueden realizar sobre él, más que en los detalles internos de la implementación.
- b) La declaración de un TDA se centra en realizar un cadena de códigos complejos y sin salida
- c) La declaración TDA se basa en miles de datos encriptados en un mismo archivo
- d) La declaración TDA no es nada y solo sirve de relleno a cualquier código de python

### 5. ¿En que no ayuda la declaración?

- a) Esta abstracción facilita el diseño modular y la reutilización del código, ya que los usuarios pueden utilizar el TDA sin necesidad de conocer los detalles internos de cómo se implementan las operaciones
- b) Esta declaración nos ayuda a facilitar el movimiento de nuestro código en diferentes ramas, como C++, o java

- c) Esta declaración nos facilita el diseño de la página web que estemos realizando
- d) Esta declaración solo sirve para programación de alto nivel en semestres superiores

### **Definición de variables**

#### **Autoevaluación 13**

##### **1. ¿Qué es una variable en programación?**

- a) Un valor constante que no cambia durante la ejecución del programa.
- b) Un espacio de memoria reservado para almacenar un valor que puede cambiar durante la ejecución del programa.
- c) Una función que realiza cálculos matemáticos.
- d) Una estructura de control que permite tomar decisiones en un programa.

##### **2. ¿Qué se entiende por "declarar una variable"?**

- a) Asignar un valor a una variable.
- b) Definir una variable con su tipo de dato y nombre, reservando espacio en memoria.

- c) Cambiar el valor de una variable.
- d) Eliminar una variable de la memoria.

**3. ¿Qué es el "ámbito de una variable"?**

- a) La parte del programa donde la variable es válida y accesible.
- b) El valor inicial de una variable.
- c) La cantidad de memoria reservada para almacenar una variable.
- d) La forma en que se almacena una variable en la memoria.

**4. ¿Qué significa "inicializar una variable"?**

- a) Cambiar el valor de una variable.
- b) Asignarle un valor inicial a una variable cuando se declara.
- c) Declarar una variable en un programa.
- d) Eliminar el valor de una variable.

**5. ¿Qué tipo de datos puede almacenar una variable?**

- a) Solo valores numéricos.
- b) Solo caracteres.
- c) Cualquier tipo de dato, dependiendo del lenguaje de programación.
- d) Sólo valores booleanos.

## **Acceso**

### **Autoevaluación 14**

**1. ¿Qué es la encapsulación en relación con los TDA?**

- a) Ocultar la implementación interna de una estructura de datos.
- b) Exponer todos los detalles de implementación.
- c) Permitir acceso ilimitado a los miembros internos.
- d) No afectar la seguridad del código.

**2. ¿Cuál es la importancia de la encapsulación en los TDA?**

- a) Mejorar la seguridad y estructura del código.
- b) Aumentar la complejidad del software.
- c) Facilitar el acceso público a los miembros.
- d) Reducir el modularidad.

**3. ¿Qué operación se utiliza para crear una instancia de un TDA?**

- a) mi\_lista.append(valor)
- b) mi\_pila.push(valor)
- c) mi\_cola.enqueue(valor)
- d) Creamos una instancia con mi\_lista = []

**4. ¿Cuál es el principio detrás de las pilas?**

- a) “Primero en entrar, primero en salir” (FIFO).
- b) “Último en entrar, primero en salir” (LIFO).
- c) “Primero en entrar, último en salir” (FILO).
- d) “Último en entrar, último en salir” (LILO).

**5. ¿Qué ventaja ofrece el acceso privado a los miembros en un TDA?**

- a) Mayor control sobre cómo se accede y modifica la información.
- b) Facilita el acceso desde fuera de la estructura.
- c) Aumenta la seguridad del código.

- d) Permite la exposición de miembros internos.

### **Almacenamiento de información**

#### **Autoevaluación 15**

**1. ¿Cuál es una forma de almacenar información dentro de una función en Python?**

- a) Variables globales
- b) Argumentos de función**
- c) Constantes
- d) Clases y objetos

**2. ¿Qué estructura de datos se utiliza comúnmente para almacenar y manipular información dentro de una función en Python?**

- a) Arrays
- b) Tuplas
- c) Estructuras**
- d) Pilas

**3. ¿Qué tipo de variables son accesibles solo dentro de la función en la que se definen en Python?**

- a) Variables estáticas
- b) Variables privadas
- c) Variables locales**
- d) Variables globales

**4. ¿Cuál de las siguientes afirmaciones sobre los argumentos de función en Python es correcta?**

- a) Los argumentos sólo pueden ser de tipo entero.
- b) Los argumentos son opcionales en todas las funciones.
- c) Los argumentos pueden ser de cualquier tipo de datos.
- d) Los argumentos no pueden ser pasados por referencia.

**5. ¿Qué estructura de datos de Python se utiliza para almacenar pares clave-valor y puede ser utilizada para almacenar información dentro de una función?**

- a) Listas
- b) Tuplas
- c) Diccionarios
- d) Conjuntos

### **Lectura de información**

#### **Autoevaluación 16**

**1. ¿Cómo se accede a un valor específico en un diccionario en Python?**

- a. Utilizando la función get()
- b. Mediante corchetes []
- c. Con la función search()
- d. Usando la función value()

**2. ¿Cuál es el resultado de intentar acceder a una clave que no existe en un diccionario utilizando la función get()?**

- a. Se produce un error
- b. Retorna el valor None**
- c. Retorna un mensaje de advertencia
- d. Retorna una cadena vacía

**3. ¿Qué método se utiliza para obtener todas las claves de un diccionario en Python?**

- a. keys()**
- b. get\_keys()
- c. all()
- d. values()

**4. ¿Cuál es la forma correcta de verificar si una clave existe en un diccionario en Python?**

- a. Utilizando la función exists()
- b. Con la palabra clave in**
- c. Mediante la función has\_key()
- d. Usando el método contains()

**5. ¿Cuál es la complejidad temporal (en el peor caso) de la operación de búsqueda de una clave en un diccionario de Python?**

- a. O(1)
- b. O(n)
- c. O(log n)
- d. O(n log n)

### **Entrada y salida por archivos**

#### **Autoevaluación 17**

**1. ¿Qué función se utiliza para abrir un archivo en Python?**

- a) read()
- b) open()
- c) write()
- d) create()

**2. ¿Cuál es el modo correcto para abrir un archivo en modo de escritura?**

- a) 'r'
- b) 'w'
- c) 'a'
- d) 'x'

**3. ¿Qué método se utiliza para leer todo el contenido de un archivo de texto?**

- a) `read()`
- b) `readline()`
- c) `readlines()`
- d) todas las anteriores

**4. ¿Qué método se utiliza para escribir una cadena de texto en un archivo?**

- a) `write()`
- b) `writelines()`
- c) `append()`
- d) `add()`

**5. ¿Qué ocurre si intentas abrir un archivo en modo de escritura que ya existe?**

- a) Se agrega el nuevo contenido al final del archivo
- b) Se sobrescribe el contenido existente
- c) Se produce un error
- d) No ocurre nada

## Archivos Binarios

### **Autoevaluación 18**

**1. ¿Qué función se utiliza para abrir un archivo en Python?**

- a. `load()`
- b. `open()`
- c. `readfile()`

d. fileopen()

2. ¿Cómo se lee una cantidad específica de bytes desde un archivo binario?

a. read\_line()

b. read\_bytes()

c. readall()

**d. read()**

3. ¿Cuál es la función para escribir datos en un archivo binario en Python?

a. save()

**b. write()**

c. append()

d. insert()

4. ¿Qué función se utiliza para mover el puntero de lectura/escritura a una posición específica en un archivo binario?

a. `shift()`

b. `move()`

**c. `seek()`**

d. `position()`

5. ¿Qué función devuelve la posición actual del puntero en un archivo binario?

a. `current\_position()`

**b. `tell()`**

c. `position()`

d. `get\_pointer()`

## Lectura de información

### **Autoevaluación 19**

1. ¿Cuál de las siguientes opciones describe mejor la lectura de información en programación?

- a) El proceso de procesar datos para su posterior almacenamiento.
- b) Obtener datos de diferentes fuentes para su posterior procesamiento por parte del programa.

c) La manipulación de datos dentro del programa para generar resultados.

d) El proceso de visualización de datos en la interfaz de usuario.

**2. ¿Cuál de las siguientes fuentes no se mencionó como una fuente común de lectura de información en programación?**

- a) Entrada del usuario
- b) Sensores y dispositivos externos
- c) Software antivirus
- d) Archivos

**3. ¿Cuál de las siguientes afirmaciones es verdadera sobre la lectura de archivos en programación?**

- a) Los archivos solo contienen datos de texto.
- b) Los archivos solo se pueden leer, no escribir.
- c) Los archivos pueden contener una variedad de datos, como texto, datos estructurados y datos binarios.
- d) Los archivos solo se pueden leer si están en el formato correcto.

**4. ¿Cuál de las siguientes tareas no implica la lectura de información en programación?**

- a) Leer un archivo de configuración al inicio del programa.
- b) Consultar una base de datos para obtener información actualizada.
- c) Imprimir resultados en la pantalla para que el usuario los vea.
- d) Obtener datos de un sensor para monitorear una variable ambiental.

**5. ¿Cuál de las siguientes opciones describe mejor por qué es importante la lectura de información en programación?**

- a) Para aumentar la complejidad del código.
- b) Para reducir la cantidad de datos necesarios para el programa.
- c) Para que el programa pueda obtener datos necesarios para realizar sus tareas.
- d) Para disminuir la velocidad de ejecución del programa.