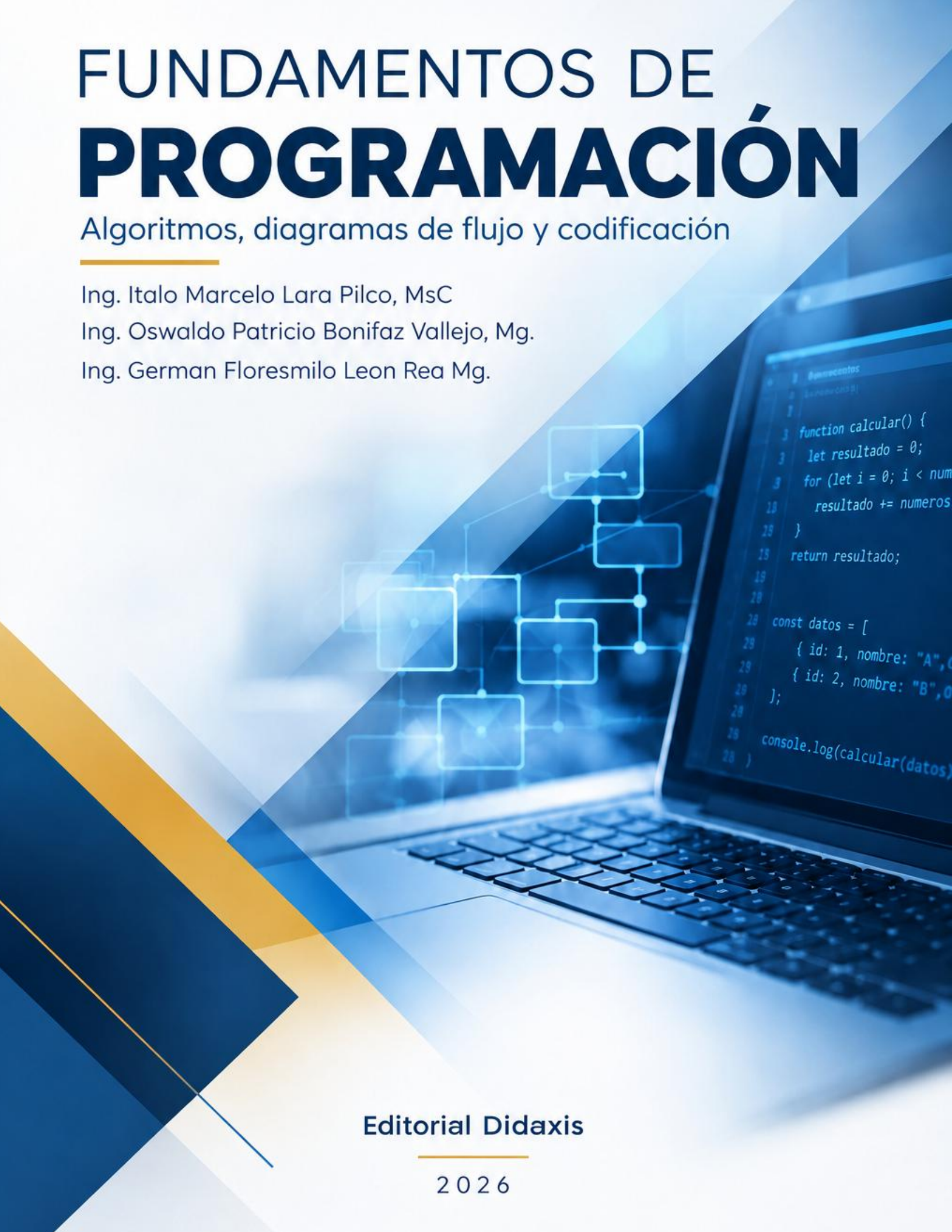
FUNDAMENTOS DE
PROGRAMACIÓN

Algoritmos, diagramas de flujo y codificación

Ing. Italo Marcelo Lara Pilco, MsC
Ing. Oswaldo Patricio Bonifaz Vallejo, Mg.
Ing. German Floresmilo Leon Rea Mg.



Editorial Didaxis

2026

Fundamentos de la programación: algoritmos, diagramas de flujo y codificación

Esta obra deberá ser citada de la siguiente manera:

Lara Pilco, I. M., Bonifaz Vallejo, O. P., & Leon Rea, G. F. (2026). Fundamentos de la programación. *Editorial Didaxis*. https://doi.org/10.64325/9kndwa97

pp.122 Tamaño (21.59 x 27.94 cm).

Incluye bibliografía.

© **Editorial Didaxis S.A.S**
@ **Instituto Superior Tecnológico Tena**

Autor(es): Lara Pilco Italo Marcelo, Bonifaz Vallejo Oswaldo Patricio y León Rea German Floresmilo
ISBN Digital: 978-9907-837-08-7
Número y año de edición: Primera edición, 2026.

Diagramación y diseño de cubierta: Editorial Didaxis S.A.S
Impresión y terminación: Editorial Didaxis S.A.S
Corrección de estilo: Andrea Eloisa Monroy Villón, Msc.
Tiraje: Online.

Prohibida la reproducción total o parcial de este con fines comerciales. Su utilización se puede realizar con carácter académico, siempre que se cite la fuente.

INSTITUTO SUPERIOR TECNOLÓGICO TENA
Tecnología, Innovación y Desarrollo

Didaxis
Editorial

Esta obra está bajo una Licencia BY 4.0 International (CC BY).

Autores



Italo Marcelo Lara Pilco
Ingeniero en Sistemas e Informática.
Magister en Docencia de Las Ciencias Informáticas.
Doctorante en la universidad Benito Juárez de México
Correo: italo.lara@itstena.edu.ec
ORCID: https://orcid.org/0009-0006-0717-2675



Oswaldo Patricio Bonifaz Vallejo
Ingeniero Electrónico y Computación
Magíster en Educación, Tecnología e Innovación, con mención honorífica.
Magíster en Evaluación y Auditoría de Sistemas Tecnológicos.
Docente-investigador del Ministerio de Educación del Ecuador
Correo: oswaldo.bonifaz@itstena.edu.ec
ORCID: https://orcid.org/0000-0003-2740-6345



German Floresmilo León Rea
Tecnólogo en Electricidad Mención Electromecánica, Ingeniero en Telecomunicaciones, Magíster en Tecnologías de la Información y la Comunicación (TIC), actualmente doctorante en Southwest State University, Kursk 305040, Russian Federation.
Correo: leon.german1987@hotmail.com
ORCID: https://orcid.org/0009-0005-4638-5959

Índice

Unidad 1. Metodología lógica para resolver problemas computacionales. 13

Diagrama de aprendizaje 13

Resumen 15

1. Introducción a la programación 15

1.1. Programación por computadora 16

1.2. Datos 17

1.2.1. Datos de tipo String **17**

1.2.2. Datos de tipo Integer **18**

1.2.3. Datos de tipo Float **19**

1.2.4. Datos de tipo Double **19**

1.2.5. Datos de tipo Booleano **20**

1.3 Pseudocódigo 21

Unidad 2. Estructuras básicas y técnicas para representar algoritmos. 27

Diagrama de aprendizaje 27

Resumen 29

2. Introducción a los algoritmos 30

2.1. Características 31

2.2. Clasificación 31

2.3. Diagramas de flujo 32

2.3.1. Introducción a los diagramas de flujo **32**

2.3.2. Características **32**

2.3.3. Simbología **33**

2.4. Estructuras Algorítmicas secuenciales 34

2.4.1. Introducción a las estructuras algorítmicas secuenciales **34**

2.4.2. Características **34**

..... 44

Unidad 3. 44

Lenguaje algorítmico y estructuras de control 44

Diagrama de aprendizaje 44

Resumen 46
3. Estructuras Algorítmicas Selectivas 46
3.1. Introducción a las Estructuras Algorítmicas Selectivas 46
3.1.1. Características **46**
3.1.2. Clasificación **46**
3.2. Estructuras Algorítmicas Cíclicas 59
3.2.1. Introducción a las Estructuras Algorítmicas Cíclicas **59**
Unidad 4: 80
Arreglos, listas y matrices 80
Diagrama de aprendizaje 80
Resumen 82
4. Listas 83
4.1 Introducción 83
4.1.1. Características **83**
4.2 Operaciones y recorridos en vectores 84
4.2.1. Introducción **84**
4.2.2. Características **84**
4.3. Matrices 85
4.3.1. Introducción **85**
4.3.2. Características **85**
Unidad 5: Programación estructurada y modular 93
Diagrama de aprendizaje 93
Resumen 95
5. Programación estructurada 95
5.1 Subprogramas, funciones y procedimientos 96
5.2 Bloques de secuencia, selección e iteración 98
5.3 Lenguajes de programación 99
Glosario 105
Referencias Bibliográficas 108
Anexos 111

PRÓLOGO

La asignatura Fundamentos de Programación constituye un pilar esencial en la formación académica de los estudiantes de la Carrera de Desarrollo de Software del Instituto Superior Tecnológico Tena, en un entorno tecnológicamente cambiante, comprender la lógica computacional, los algoritmos y la construcción estructurada de soluciones resulta imprescindible para responder con solvencia a los desafíos del ámbito profesional e industrial.

Esta guía general de estudio ha sido elaborada en estricta concordancia con el Programa de Estudio de la Asignatura (PEA), el propósito central es orientar de manera metódica y participativa el proceso de enseñanza-aprendizaje, sirviendo como una hoja de ruta conceptual y práctica que articula los fundamentos teóricos con su aplicación directa, de este modo, este recurso busca fortalecer la autonomía del estudiante, impulsar el pensamiento crítico y facilitar la apropiación progresiva de competencias clave para la resolución de problemas.

Los contenidos han sido estructurados de forma secuencial y lógica a lo largo de cinco unidades fundamentales:

Unidad 1: aborda la metodología lógica, los tipos de datos primarios y el pseudocódigo.

Unidad 2: introduce las características formales de los algoritmos y la representación visual mediante diagramas de flujo y estructuras secuenciales.

Unidad 3: explora a fondo las estructuras de control condicionales (simples, dobles y múltiples) y cíclicas (While, Do-While, For).

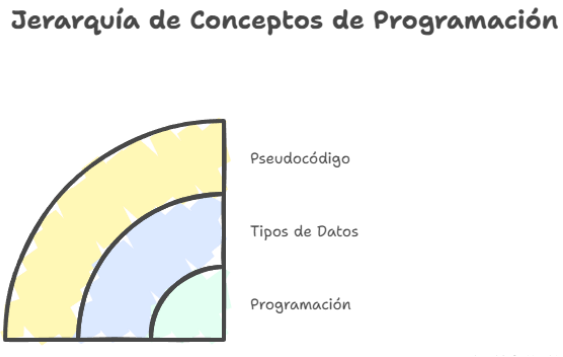
Unidad 4: organiza datos mediante arreglos, listas y matrices, y compara las colecciones según su comportamiento en Java y Python.

Unidad 5: integra programación estructurada y modular mediante bloques, funciones, procedimientos y pruebas de subprogramas.

Se motiva a los estudiantes a emplear de forma activa y constante este material Didáctico, aprovechando las síntesis, actividades de aprendizaje, solucionarios y evaluaciones incluidas en cada unidad, este recurso no solo apoyará su trabajo personal y colaborativo, sino que garantizará de forma efectiva el logro de los resultados de aprendizaje previstos para su perfil profesional.

Unidad 1. Metodología lógica para resolver problemas computacionales.

Diagrama de aprendizaje



Resumen

La programación plantea instrucciones para que un computador realice tareas. El estudiante analiza el problema, diseña el algoritmo, codifica y verifica la solución; la programación estructurada se desarrolla en la unidad 5 (Joint Task Force on Computer Science Curricula, 2024). La programación por computadora traduce algoritmos en lenguaje humano a lenguajes de programación como C, Java, y Python, utilizando compiladores o intérpretes para convertir el código fuente en instrucciones ejecutables. Los lenguajes de programación se dividen en tres tipos: Lenguaje máquina, de bajo nivel, de alto nivel. Los datos en programación son valores que un computador recibe a través de diferentes medios y que el programador procesa para construir soluciones o desarrollar algoritmos, existen varios tipos de datos:

- **String**: Representa cadenas de caracteres para almacenar información textual.
- **Integer**: Representa números enteros, positivos o negativos, con rango limitado según el lenguaje y la arquitectura del sistema.
- **Float**: Representa números con punto flotante, utilizado para ahorrar memoria en grandes matrices de números en coma flotante.
- **Double**: Números de punto flotante de doble precisión, utilizados en cálculos que requieren alto grado de precisión.
- **Booleano**: Representa dos valores, TRUE o FALSE, fundamentales para la toma de decisiones en algoritmos.

1. Introducción a la programación

La programación es el proceso de creación de instrucciones para que un ordenador realice tareas específicas. Estas instrucciones se agrupan en programas, que permiten automatizar procesos y resolver problemas de manera eficiente. La programación estructurada es considerada como la metodología clásica, la misma que se basa en el uso

de estructuras lógicas: secuencias, decisiones y repeticiones, promoviendo claridad y modularidad (Villalobos, 2014).

El desarrollo de software se sustenta en principios fundamentales como el análisis del problema, el diseño del algoritmo y su codificación en un lenguaje de programación, seguido de pruebas y mantenimiento. Comprender estos principios es esencial para garantizar soluciones computacionales.

En esta unidad se estudian el análisis del problema, el diseño del algoritmo, la codificación y la verificación; la programación estructurada se desarrolla de forma específica en la unidad 5.

1.1. Programación por computadora

La programación por computadora es el proceso mediante el cual se diseñan, codifican y ejecutan algoritmos para resolver problemas utilizando lenguajes de programación, como C, Java y Python. Estos lenguajes permiten expresar instrucciones mediante reglas sintácticas y semánticas definidas, facilitando la comunicación entre el programador y el sistema informático. Para ejecutar dichas instrucciones, se utilizan diferentes mecanismos de implementación que permiten transformar o procesar el código fuente. Según Sebesta (2016), los principales métodos de implementación de los lenguajes de programación son la compilación, la interpretación pura y la implementación híbrida.

Los lenguajes de programación también pueden clasificarse según su nivel de abstracción respecto al hardware:

- Lenguaje máquina: está constituido por instrucciones codificadas en formato binario que el procesador puede ejecutar directamente. Es específico de la

arquitectura de la máquina y resulta complejo de utilizar para el desarrollo de programas.

- Lenguaje de bajo nivel (ensamblador): utiliza símbolos mnemónicos para representar instrucciones de máquina. Mantiene una estrecha relación con la arquitectura del procesador y requiere un programa ensamblador que traduzca sus instrucciones a código máquina.
- Lenguajes de alto nivel: permiten expresar instrucciones utilizando estructuras sintácticas más cercanas al razonamiento humano. Facilitan el desarrollo, la comprensión y el mantenimiento de los programas, además de favorecer su portabilidad entre diferentes plataformas, dependiendo de los mecanismos de implementación disponibles (Sebesta, 2016).

Esta clasificación permite comprender la evolución de los lenguajes de programación y las diferencias entre los niveles de abstracción utilizados para desarrollar programas informáticos.

1.2. Datos

Los datos son valores o referentes que un computador recibe a través de diferentes medios y que el programador procesa para construir soluciones o desarrollar algoritmos, los datos son el inicio para producir la información, existen varios tipos de datos, los mismos que se describen a continuación.

1.2.1. Datos de tipo String

Las cadenas de caracteres permiten representar y almacenar información textual, como nombres, direcciones o descripciones. Joyanes Aguilar (2008) incluye el estudio de las cadenas de caracteres y sus operaciones entre los contenidos de programación y estructuras de datos. En esta guía se presentan ejemplos de su uso en Java y Python.

Ejemplo en Java	*Ejemplo en Python*

Código ejecutable
public class StringEjemplo {
public static void main(String[] args)
{
String nombre = "Juan";
System.out.println(nombre);
}
}
	mensaje = "Hola, Mundo"
	print(mensaje)

1.2.2. Datos de tipo Integer

Los datos de tipo entero, comúnmente identificados como `Integer` o `int`, permiten representar números positivos, negativos y el cero, sin componentes decimales. Su utilización es fundamental en operaciones aritméticas, contadores, índices y otras actividades propias del desarrollo de algoritmos. El rango de valores que pueden representar depende del lenguaje de programación y de las características de implementación del tipo de dato (Schildt, 2022).

Ejemplo en Java	*Ejemplo en Python*

Código ejecutable
public class IntegerEjemplo {
public static void main(String[] args)
{
int num = 5;
System.out.println(num);
}
}
	num = 5
	print(num)

1.2.3. Datos de tipo Float

Los valores Float son datos de tipo primitivo que almacenan números con punto flotante, como 3.5 o 12.75. En los lenguajes de programación, representan una coma flotante de precisión simple. Este tipo de dato se utiliza principalmente para ahorrar memoria en grandes matrices de datos numéricos, siendo notablemente menos preciso que el tipo de datos double (Oracle, 2023).

Ejemplo en Java	*Ejemplo en Python*

Código ejecutable
public class FloatEjemplo {
public static void main(String[] args)
{
float f = 35.5f;
System.out.println(f);
}
}
	f = 19.99
	print(f)

1.2.4. Datos de tipo Double

Los números de punto flotante de doble precisión, representados mediante el tipo de dato double, permiten almacenar y procesar valores numéricos con decimales. Permite representar valores de gran magnitud, por lo que resulta adecuado para operaciones matemáticas, cálculos científicos y aplicaciones que requieren mayor precisión que la ofrecida por los tipos de punto flotante de precisión simple. Sin embargo, debido a las limitaciones de la representación binaria, ciertos valores decimales no pueden almacenarse de manera exacta, lo que puede generar pequeñas diferencias de redondeo durante las operaciones aritméticas (Microsoft, s. f.).

Ejemplo en Java	*Ejemplo en Python*

Código ejecutable
public class DoubleEjemplo {
public static void main(String[] args)
{
double n1 = 8.9587979797797;
System.out.println(n1);
}
}
	n1 = 8.9587979797797
	print(n1)

1.2.5. Datos de tipo Booleano

Los datos booleanos son fundamentales para la toma de decisiones en los algoritmos, ya que permiten evaluar condiciones en bucles y estructuras de control, un dato booleano representa dos valores: TRUE o FALSE (Gaddis, 2023).

Ejemplo en Java	*Ejemplo en Python*

Código ejecutable
public class BooleanoEjemplo {
public static void main(String[] args)
{
boolean mayor_edad = false;
if (!mayor_edad) {
System.out.println("es niño");
}
}
}
	es_valido=True
	if es_valido:
	print("El valor es válido")

1.3 Pseudocódigo

El pseudocódigo es una herramienta fundamental en el diseño de algoritmos y programación, ya que permite describir procedimientos lógicos de forma clara y estructurada sin preocuparse por la sintaxis específica de un lenguaje de programación. Según el Instituto de Ingeniería de Software (2021), el pseudocódigo facilita la comunicación entre los miembros del equipo de desarrollo y ayuda a identificar posibles errores o mejoras en el algoritmo antes de su implementación. Además, al utilizar un lenguaje similar al natural, se puede concentrar en la lógica del problema, lo que mejora la comprensión y eficiencia del proceso de desarrollo.

Para Larman (2003), una de las principales ventajas del pseudocódigo es su capacidad para abstraer detalles complejos y centrarse en la estructura y el flujo del algoritmo, destaca que el pseudocódigo es especialmente útil en las etapas iniciales del desarrollo, ya que permite visualizar la solución propuesta y evaluar su viabilidad sin necesidad de conocimientos profundos en un lenguaje de programación específico, Esto es crucial en proyectos grandes y complejos, donde la claridad y simplicidad pueden marcar la diferencia entre el éxito y el fracaso del proyecto.

Ejemplo

Suma de dos números naturales en Pseint.

```
Algoritmo Suma
    definir n1, n2, s Como Entero
    escribir "Ingrese el primer número"
    leer n1
    escribir "Ingrese el segundo número"
    leer n2
    s= n1+n2
    Imprimir "La suma es: " s
FinAlgoritmo
```

ACTIVIDADES DE APRENDIZAJE

-Elaborar una línea de tiempo que ilustre los momentos y conceptos más importantes que han dado forma a la programación tal como la conocemos hoy.

-Elaborar un mapa mental en la herramienta que usted considere (CANVA, Genially, GoConqr, etc), sobre el tema: Datos: numéricos, alfanuméricos, Lógicos (String, Integer, Float, Double, Booleano), Variables, Constantes.

Ejemplo en Java	*Ejemplo en Python*

SUMA DE DOS ENTEROS
Código ejecutable
import java.util.Scanner;
public class SumaDosEnterosRevision {
public static void main(String[] args) {
Scanner entrada = new Scanner(System.in);
int primero = entrada.nextInt();
int segundo = entrada.nextInt();
System.out.println("Suma: " + (primero + segundo));
entrada.close();
}
}
	SUMA DE DOS ENTEROS
	primero = int(input())
	segundo = int(input())
	print("Suma:", primero + segundo)

EVALUACIÓN

Fundamentos de Programación: Lógica, Lenguajes y Tipos de Datos

Nombre:		**Fecha:**	
Curso / Asignatura:		**Puntaje:**	/ 10

Instrucciones: Lea cada pregunta con atención y encierre en un círculo o marque con una X la letra correspondiente a la respuesta correcta. Cada pregunta tiene un único valor y una sola opción válida.

1. ¿Cuáles son las estructuras lógicas fundamentales en las que se basa la programación estructurada?

a) Variables, constantes y operadores.

b) Secuencias, decisiones y repeticiones.

c) Compiladores, intérpretes y código fuente.

d) Algoritmos, pseudocódigo y diagramas.

2. ¿Qué tipo de lenguaje de programación se caracteriza por utilizar símbolos nemotécnicos y requerir traducción al lenguaje máquina?

a) Lenguaje máquina.

b) Lenguaje de alto nivel.

c) Lenguaje de bajo nivel (Ensamblador).

d) Lenguaje natural.

3. ¿Cuál es una ventaja principal de los lenguajes de programación de alto nivel?

a) Es programación binaria directa y dependiente del hardware.

b) Es cercano al lenguaje natural, tiene menor tiempo de desarrollo y es independiente de la máquina.

c) No requiere compiladores ni intérpretes para ejecutarse.

d) Solo permite almacenar cadenas de texto sin formato.

4. El tipo de dato utilizado para almacenar información textual como nombres o direcciones se denomina:

a) Integer

b) Float

c) String

d) Booleano

5. ¿Qué diferencia principal existe entre el tipo de dato Float y el tipo Double?

a) Float almacena texto y Double almacena números enteros.

b) Float representa precisión simple y ahorra memoria, mientras que Double representa precisión doble para mayor exactitud.

c) Float se usa únicamente para tomar decisiones lógicas (True/False).

d) No existe ninguna diferencia, ambos tienen el mismo rango y consumo de memoria.

6. ¿Cuáles son los dos únicos valores que puede tomar un tipo de dato Booleano?

a) Positivo y Negativo.

b) Entero y Decimal.

c) TRUE o FALSE.

d) Texto y Número.

7. ¿Por qué los tipos de datos enteros (Integer) tienen un rango limitado de valores?

a) Porque no permiten valores negativos.

b) Debido al tamaño fijo de bits asignados según el lenguaje y la arquitectura del sistema.

c) Porque se convierten automáticamente en texto.

d) Porque solo se utilizan en pseudocódigo.

8. ¿Cuál es la función principal del pseudocódigo en el diseño de programas?

a) Ejecutar el programa directamente en la CPU sin compilador.

b) Convertir imágenes en código fuente ejecutable.

c) Describir procedimientos lógicos de forma clara sin preocuparse por la sintaxis estricta de un lenguaje.

d) Crear bases de datos automáticamente.

9. Según el texto, ¿en qué fase del desarrollo es especialmente útil el pseudocódigo?

a) En la fase final de mantenimiento y despliegue.

b) En las etapas iniciales de desarrollo para visualizar la solución y evaluar su viabilidad.

c) Solamente al momento de compilar a lenguaje máquina.

d) Durante la configuración de la tarjeta madre.

10. En el contexto de la programación, ¿qué son los 'datos'?

a) Instrucciones finales en código binario.

b) Los dispositivos físicos como el teclado o el monitor.

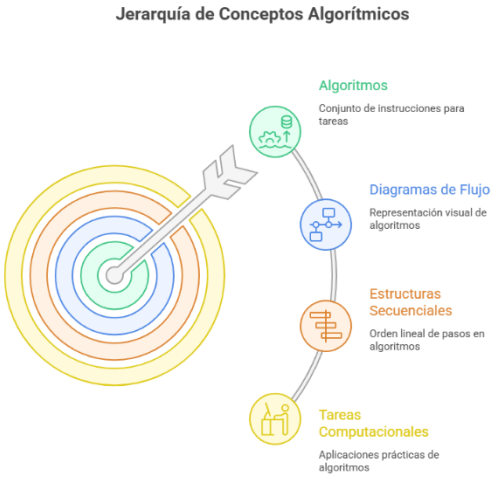
c) Valores o referentes que un computador recibe para ser procesados y construir soluciones o algoritmos.

d) Los manuales impresos de uso del software.

Unidad 2.

Estructuras básicas y técnicas para representar algoritmos.

Diagrama de aprendizaje



Resumen

Los algoritmos son una parte fundamental de la informática y la programación, también conocidos como estructuras de procesos, es una secuencia finita de instrucciones o pasos que se siguen para resolver un problema o realizar una tarea. Los algoritmos se pueden encontrar en todos los aspectos de la vida cotidiana, desde la preparación de recetas de cocina hasta la elaboración de sistemas robustos.

Un algoritmo bien organizado permite describir y revisar los pasos de una solución antes de implementarla. En esta guía se priorizan los algoritmos secuenciales, selectivos y cíclicos; las aplicaciones de inteligencia artificial se reservan para asignaturas posteriores (Sedgewick & Wayne, 2011).

Los diagramas de flujo es una representación gráfica de un proceso o algoritmo, los algoritmos utilizan símbolos estandarizados para mostrar las etapas del proceso, permitiendo a los usuarios entender y analizar la lógica detrás de las operaciones, su uso es fundamental tanto en la programación como en la gestión de procesos de negocio, ya que facilita la visualización de la secuencia de pasos que se deben seguir para alcanzar un objetivo específico.

Esta herramienta gráfica permite descomponer procesos complejos en componentes simples y manejables, lo que no solo mejora la comprensión, sino que también ayuda a identificar errores, redundancias y oportunidades de mejora, gracias a su claridad y versatilidad, los diagramas de flujo se aplican en múltiples disciplinas, desde la ingeniería de software hasta la administración empresarial, siendo útiles tanto para documentar sistemas existentes como para diseñar nuevos flujos operativos.

Las estructuras algorítmicas secuenciales son aquellas en las que las instrucciones se ejecutan una tras otra, en un orden lineal y determinado, cada acción sigue a la anterior sin desviaciones ni repeticiones, lo que permite que el flujo del algoritmo avance de manera continua desde el inicio hasta el fin. Esta forma de estructuración es ideal para tareas simples y directas, donde cada paso depende del resultado del anterior.

En este tipo de estructuras, la salida de una operación se convierte en la entrada de la siguiente, formando una cadena lógica de ejecución, se caracterizan por tener un único punto de entrada y salida, lo que facilita su comprensión, mantenimiento y depuración. gracias a su claridad, las estructuras secuenciales son ampliamente utilizadas en la enseñanza de programación básica y en el diseño de algoritmos para procesos deterministas.

2. Introducción a los algoritmos

Los algoritmos son una parte fundamental de la informática y la programación, también conocidos como estructuras de procesos, es una secuencia finita de instrucciones o pasos que se siguen para resolver un problema o realizar una tarea. Los algoritmos se pueden encontrar en todos los aspectos de la vida cotidiana, desde la preparación de recetas de cocina hasta la elaboración de cálculos matemáticos complejos. En términos de programación estas estructuras son esencial para desarrollar software eficiente y efectivo (Cormen et al., 2009).

Un algoritmo bien organizado permite describir y revisar los pasos de una solución antes de implementarla. En esta guía se priorizan los algoritmos secuenciales, selectivos y cíclicos; las aplicaciones de inteligencia artificial se reservan para asignaturas posteriores (Sedgewick & Wayne, 2011).

Existen diferentes tipos de algoritmos, como los algoritmos de búsqueda, de ordenación y de grafos, cada uno con sus propias características y aplicaciones; por ejemplo, los algoritmos de búsqueda se utilizan para encontrar elementos dentro de estructuras de datos, mientras que los algoritmos de ordenación organizan los datos en un orden específico, los algoritmos de grafos son esenciales en la teoría de redes y tienen aplicaciones en la optimización de rutas y la planificación de redes, sin embargo en esta guía nos basaremos en tres tipos de algoritmos como los secuenciales, selectivos y cíclicos (Sedgewick & Wayne, 2011).

2.1. Características

Las principales características de los algoritmos son:

Finito. - Un algoritmo debe tener un número finito de pasos. Esto significa que el proceso debe terminar después de realizar un número definido de pasos. La finitud garantiza que el algoritmo no se ejecutará indefinidamente (Cormen et al., 2009).

Definido y preciso. - Cada paso del algoritmo debe estar claramente definido. Las instrucciones deben ser precisas y sin ambigüedades para asegurar que el algoritmo se puede implementar correctamente (Knuth, 1997).

Entrada. - Un algoritmo debe aceptar cero o más entradas, que son valores externos necesarios para realizar el procesamiento. Estos datos iniciales son esenciales para el correcto funcionamiento del algoritmo (Knuth, 1997).

Salida. - Un algoritmo debe producir al menos una salida. Esta salida es el resultado esperado después de realizar todos los pasos del algoritmo y proporciona la solución al problema planteado (Aho et al., 1983).

Efectividad. - Las operaciones de un algoritmo deben ser básicas y realizables en un tiempo finito. Esto implica que cada operación puede ser llevada a cabo, al menos en principio, por una persona usando lápiz y papel (Knuth, 1997).

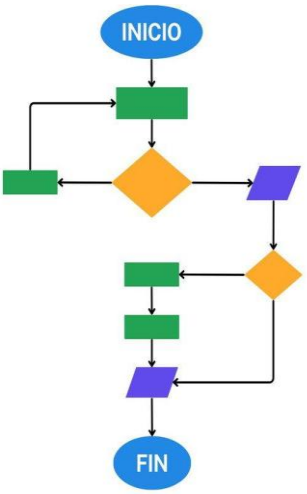
2.2. Clasificación

Los algoritmos son un conjunto de instrucciones que permiten resolver problemas específicos de manera sistemática y eficiente. Dependiendo de su estructura y funcionamiento, los algoritmos se pueden clasificar en tres grandes categorías: secuenciales, selectivos y cíclicos.

2.3. Diagramas de flujo

2.3.1. Introducción a los diagramas de flujo

Ilustración 1 Diagrama de Flujo



Nota. Los diagramas de flujo son la representación gráfica de un algoritmo.

Un diagrama de flujo es una representación gráfica de un proceso o algoritmo, utiliza símbolos estandarizados para mostrar las etapas del proceso, permitiendo a los usuarios entender y analizar la lógica detrás de las operaciones. Su uso es fundamental en la programación y en la gestión de procesos de negocio (Dennis et al., 2021).

2.3.2. Características

Los diagramas de flujo son una herramienta fundamental en la programación y en la gestión de procesos. Permiten representar gráficamente la secuencia de pasos en un proceso o algoritmo, facilitando la comprensión y análisis de la lógica subyacente. A continuación, se detallan sus principales características:

Claridad

Los diagramas de flujo proporcionan una visión clara y estructurada de un proceso, cada paso está representado de manera que es fácil de seguir y entender, lo que ayuda a eliminar malentendidos y confusiones.

Simplicidad

Estos diagramas descomponen procesos complejos en pasos manejables, esto no solo facilita la comprensión, sino que también permite identificar errores y áreas de mejora más fácilmente.

Flexibilidad

Son versátiles y pueden aplicarse a diversos campos, desde la ingeniería de software hasta la gestión de procesos empresariales, son útiles tanto para documentar sistemas existentes como para diseñar nuevos sistemas.

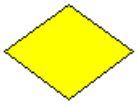
Identificación de Cuellos de Botella: al representar visualmente un proceso, los diagramas de flujo permiten identificar cuellos de botella y áreas ineficientes, esto facilita la optimización y mejora continua de los procesos (Rodríguez, 2024).

2.3.3. Simbología

Los diagramas de flujo utilizan una serie de símbolos estandarizados que permiten representar gráficamente los distintos componentes de un proceso, la **Ilustración 2** presenta los símbolos más utilizados.

Ilustración 2

Simbología de Los Diagramas de Flujo

	**Inicio/Final** Se utiliza para indicar el inicio y el final de un diagrama; de Inicio sólo puede salir una línea de flujo y al final sólo debe llegar una línea		**Decisión** Indica la comparación de dos datos y dependiendo del resultado lógico (falso o verdadero) se toma la decisión de seguir un camino del diagrama u otro
	**Entrada/Salida** Entrada/Salida de datos por cualquier dispositivo (scanner, lector de código de barras, micrófono, parlantes, etc.)		**Impresora/Documento.** Indica la presentación de uno o varios resultados en forma impresa
	**Entrada por teclado.** Entrada de datos por teclado. Indica que el computador debe esperar a que el usuario teclee un dato que se guardará en una variable o constante		**Pantalla** Instrucción de presentación de mensajes o resultados en pantalla
	**Acción/Proceso** Indica una acción o instrucción general que debe realizarse (operaciones aritméticas, asignaciones, etc.)		**Conector Interno** Indica el enlace de dos partes de un diagrama dentro de la misma página
	**Flujo/Flecas de Dirección** Indica el seguimiento lógico del diagrama. También indica el sentido de ejecución de las operaciones		**Conector Externo** Indica el enlace de dos partes de un diagrama en páginas diferentes

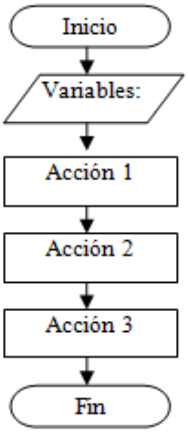
Nota. Cada símbolo representa un proceso del algoritmo; (Rodríguez, 2024).

2.4. Estructuras Algorítmicas secuenciales

2.4.1. Introducción a las estructuras algorítmicas secuenciales

Las estructuras secuenciales son aquellas en las que una acción (instrucción) sigue a otra de manera consecutiva. Las tareas se suceden de tal modo que la salida de una es la entrada de la siguiente, y así sucesivamente hasta el fin del proceso, Dennis et al. (2021).

Ilustración 3. Estructuras Algorítmicas Secuenciales



2.4.2. Características

Secuencia de Instrucciones: Las instrucciones se ejecutan en un orden específico, de arriba a abajo, una tras otra.

Un Solo Punto de Entrada y Salida: La ejecución comienza en un solo punto de entrada y continúa de manera secuencial hasta alcanzar un punto de salida.

Ejecución Unidireccional: La ejecución del programa sigue una dirección unidireccional, avanzando de una instrucción a la siguiente sin retroceder ni desviarse.

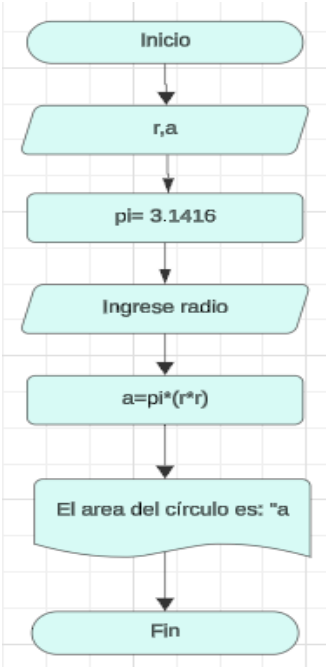
Fácil Lectura y Mantenimiento: La estructura secuencial es fácil de leer y comprender, facilitando el mantenimiento y la depuración del código.

Ejemplo 1: Desarrollar un algoritmo que calcule el área de un círculo.

Ejemplo en Java	*Ejemplo en Python*

Código ejecutable
package area_circulo;
import java.util.Scanner;
public class Area_circulo {
public static void main(String[] args)
{
Scanner sc=new Scanner (System.in);
double r,a;
double pi=3.1416;
System.out.println("Ingrese radio del círculo");
r=sc.nextDouble();
a=pi*(r*r);
System.out.println("El área del círculo es: "+a);
}
}
	r=float(input("Ingrese el radio del círculo \n"))
	pi=3.1416
	a=pi*(r*r)
	print("El área del círculo es: ",a)

Diagrama de Flujo

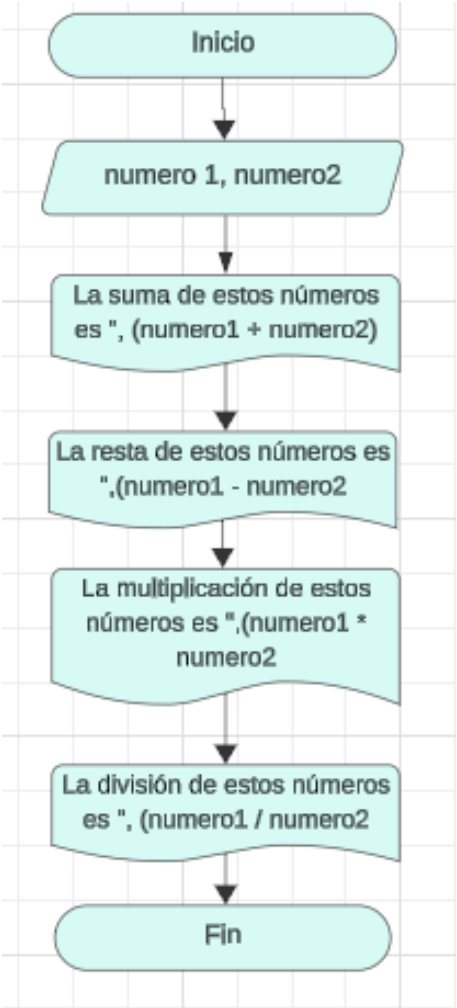


Ejemplo 2: Desarrollar un algoritmo que, dados dos números, muestre la suma, resta, división y multiplicación de ambos.

Ejemplo en Java	*Ejemplo en Python*

Código ejecutable
package ejemplos;
import java.util.Scanner;
public class Operaciones_Matematicas {
public static void main(String[] args) {
Scanner scanner = new Scanner (System.in);
double numero1;
double numero2;
// Solicitud de datos al usuario
System.out.println("Escribe dos números para obtener el resultado de su suma, resta, multiplicación y división");
System.out.print("Dame el valor del primer número: ");
numero1 = scanner.nextDouble();
System.out.print("Dame el valor del segundo número: ");
numero2 = scanner.nextDouble();
// Realizamos Cálculos y mostramos en pantalla
System.out.println("La suma de estos números es " + (numero1 + numero2));
System.out.println("La resta de estos números es " + (numero1 - numero2));
System.out.println("La multiplicación de estos números es " + (numero1 * numero2));
if (numero2 != 0) { System.out.println("La división de estos números es " + (numero1 / numero2)); }
	#Solicitud de datos al usuario
	numero1 = float(input("Ingresa el primer número \n"))
	numero2 = float(input("Ingresa el segundo número \n"))
	#Realizamos Cálculos y mostramos en pantalla
	print("La suma de estos números es ", (numero1 + numero2))
	print("La resta de estos números es ",(numero1 - numero2))
	print("La multiplicación de estos números es ",(numero1 * numero2))
	if numero2 != 0:
	print("La división de estos números es", numero1 / numero2)
	else:
	print("No se puede dividir para cero.")

```
else { System.out.println("No se puede dividir para cero."); }
        }
    }
```



ACTIVIDADES DE APRENDIZAJE

1. Desarrollar un algoritmo secuencial que convierta un valor dado en grados Fahrenheit a grados Celsius.
2. Desarrollar un algoritmo secuencial que calcule la media de tres números introducidos por teclado.
3. Desarrollar un algoritmo secuencial que reciba una cantidad de minutos y lo convierta a horas y minutos.
4. Una tienda ofrece un descuento del 15% sobre el total de la compra y un cliente desea saber cuánto deberá pagar finalmente por su compra.
5. Un vendedor recibe un sueldo base más un 10% extra por comisión de sus ventas, el vendedor desea saber cuánto dinero obtendrá por concepto de comisiones por las tres ventas que realiza en el mes y el total que recibirá, tomando en cuenta su sueldo base y comisiones.

ÁREA DE UN RECTÁNGULO

Ejemplo en Java	*Ejemplo en Python*

Código ejecutable
import java.util.Scanner;
public class AreaRectanguloRevision {
public static void main(String[] args)
{
Scanner entrada = new Scanner(System.in);
double base = entrada.nextDouble();
double altura = entrada.nextDouble();
System.out.println("Área: " + (base * altura));
entrada.close();
}
}
	base = float(input())
	altura = float(input())
	print("Área:", base * altura)

ACTIVIDADES DE APRENDIZAJE

1. Desarrollar un algoritmo secuencial que convierta un valor dado en grados Fahrenheit a grados Celsius.
2. Desarrollar un algoritmo secuencial que calcule la media de tres números introducidos por teclado.
3. Desarrollar un algoritmo secuencial que reciba una cantidad de minutos y lo convierta a horas y minutos.
4. Una tienda ofrece un descuento del 15% sobre el total de la compra y un cliente desea saber cuánto deberá pagar finalmente por su compra.
5. Un vendedor recibe un sueldo base más un 10% extra por comisión de sus ventas, el vendedor desea saber cuánto dinero obtendrá por concepto de comisiones por las tres ventas que realiza en el mes y el total que recibirá, tomando en cuenta su sueldo base y comisiones.

EVALUACIÓN

Estructuras básicas y técnicas para representar algoritmos

Nombre:		**Fecha:**	
Curso / Asignatura:		**Puntaje:**	/ 10

Instrucciones: Lea cada pregunta con atención y encierre en un círculo o marque con una X la letra correspondiente a la respuesta correcta. Cada pregunta tiene un único valor y una sola opción válida.

1. ¿Cuál de las siguientes opciones define correctamente la característica de 'finitud' en un algoritmo?

a) El algoritmo debe aceptar cero o más entradas de datos.

b) El algoritmo debe tener un número definido de pasos y terminar en un tiempo determinado.

c) Cada instrucción del algoritmo debe ser ambigua para permitir flexibilidad.

d) El algoritmo debe ejecutarse de forma continua sin llegar a una conclusión.

2. ¿Qué símbolo de diagrama de flujo se utiliza para representar la toma de decisiones basada en una comparación lógica?

a) Rectángulo

b) Paralelogramo (Romboide)

c) Rombo

d) Óvalo / Elipse

3. ¿Cuál es la función del símbolo del paralelogramo (romboide) en la simbología estándar de diagramas de flujo?

a) Indicar el inicio o el final del algoritmo.

b) Representar la entrada o salida de datos a través de cualquier dispositivo.

c) Representar un proceso u operación aritmética.

d) Mostrar una instrucción de presentación en pantalla únicamente.

4. ¿Qué característica define a las estructuras algorítmicas secuenciales?

a) Las instrucciones se repiten indefinidamente hasta cumplir una condición.

b) Se evalúan múltiples caminos alternativos de ejecución.

c) Cada acción sigue a la anterior en un orden lineal sin desviaciones ni repeticiones.

d) Poseen múltiples puntos de entrada y de salida.

5. Según el documento, ¿cuál es uno de los principales beneficios de utilizar diagramas de flujo en la gestión de procesos?

a) Aumentar la complejidad de los códigos de programación.

b) Permitir identificar cuellos de botella y áreas ineficientes para optimizar procesos.

c) Sustituir completamente el código ejecutable en los lenguajes de programación.

d) Evitar la necesidad de definir variables de entrada.

6. Un algoritmo exige que sus operaciones sean básicas y realizables en la práctica. ¿A qué característica corresponde este enunciado?

a) Definido y preciso

b) Efectividad

c) Entrada

d) Salida

7. ¿Qué símbolo se utiliza específicamente para conectar dos partes de un diagrama de flujo que se encuentran en páginas diferentes?

a) Conector Interno (Círculo)

b) Conector Externo (Pentágono / Figura apuntando abajo)

c) Flecha de Dirección

d) Rectángulo de Proceso

8. ¿Cuáles son las tres grandes categorías en las que se clasifican los algoritmos según el texto?

a) Básicos, Intermedios y Avanzados

b) Secuenciales, Selectivos y Cíclicos

c) Visuales, Textuales y Gráficos

d) Numéricos, Alfanuméricos y Lógicos

9. En el ejemplo del cálculo del área de un círculo, ¿qué elementos representan la 'entrada' y la 'salida' respectivamente?

a) Entrada: la constante Pi; Salida: el radio.

b) Entrada: el radio (r); Salida: el área calculada (a).

c) Entrada: la fórmula matemática; Salida: el mensaje inicial.

d) Entrada: el área; Salida: la multiplicación.

10. ¿Por qué las estructuras secuenciales son ideales para procesos deterministas y para la enseñanza inicial de la programación?

a) Porque no requieren un punto final de ejecución.

b) Porque permiten saltar de forma aleatoria entre distintas instrucciones.

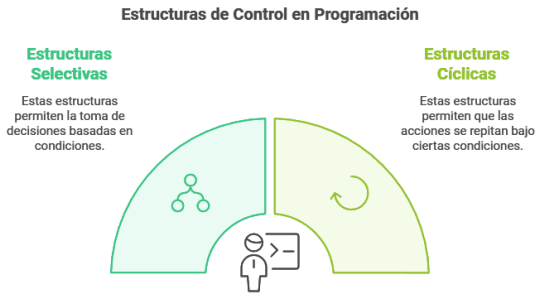
c) Porque tienen un único punto de entrada y salida, facilitando su lectura, mantenimiento y depuración.

d) Porque manejan automáticamente decisiones complejas en paralelo.

Unidad 3.

Lenguaje algorítmico y estructuras de control

Diagrama de aprendizaje



Resumen

La Unidad 3 se enfoca en el análisis y desarrollo de las estructuras de control. Estas estructuras permiten a los sistemas informáticos gestionar procesos que requieren toma de decisiones y controlar operaciones repetitivas de manera eficiente.

Las estructuras selectivas, también conocidas como estructuras condicionales, son componentes fundamentales en la programación ya que permiten tomar decisiones basadas en condiciones específicas, estas estructuras dirigen el flujo de ejecución del programa hacia diferentes rutas según el resultado de una condición evaluada.

Las estructuras repetitivas, también conocidas como bucles o estructuras de iteración, son construcciones en la programación que permiten ejecutar un conjunto de instrucciones múltiples veces, estas son fundamentales para realizar tareas repetitivas de manera eficiente y automatizada.

3. Estructuras Algorítmicas Selectivas

3.1. Introducción a las Estructuras Algorítmicas Selectivas

Las estructuras algorítmicas selectivas permiten que el flujo de ejecución de un programa se desvíe en función de condiciones específicas. Estas decisiones se basan en expresiones lógicas que resultan en verdadero o falso, determinando así qué acciones deben ejecutarse (Dennis et al., 2021).

3.1.1. Características

- **Condicionalidad**: Basadas en condiciones lógicas.
- **Flexibilidad**: Permiten múltiples caminos de ejecución.
- **Claridad**: Mejoran la legibilidad del código al explicitar las decisiones.

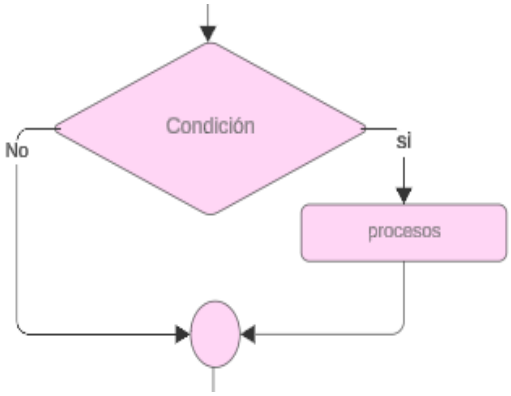
3.1.2. Clasificación

Las estructuras algorítmicas selectivas se clasifican en: Estructuras simples, dobles y múltiples.

Estructuras Selectivas Simples:

Las estructuras selectivas simples permiten controlar el flujo de ejecución de un algoritmo mediante la evaluación de una condición lógica. Cuando dicha condición es verdadera, se ejecuta un bloque específico de instrucciones; en caso contrario, el programa continúa con la siguiente instrucción. La sentencia if constituye una de las estructuras de selección más utilizadas en los lenguajes de programación (Franchitti, 2024).

Ilustración 4 Algoritmo Selectivo Simple



Nota. Estas estructuras realizan procesos solo por el camino del sí, y por camino del no van al fin del algoritmo.

Características

- **Simplicidad**: Solo evalúan una condición.
- **Ejecución Condicional**: Bloques de código se ejecutan solo si la condición es verdadera.

Ejemplos

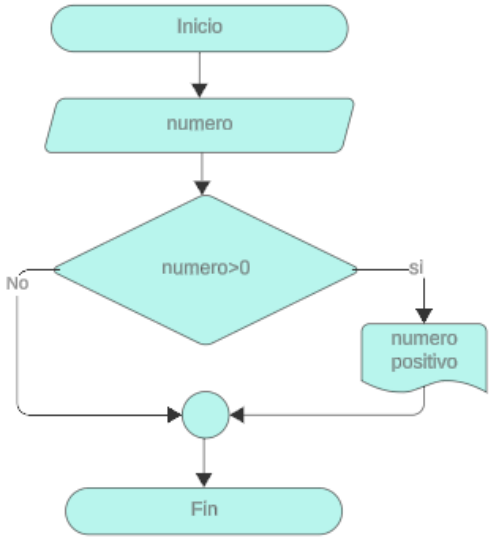
Ejemplo 1 (selección simple): ingresar un entero y mostrar un mensaje solo si es positivo. Pruebe 5, −2 y 0; solo 5 activa el mensaje.

Ejemplo en Java	*Ejemplo en Python*

Código ejecutable
package numero;
import java.util.Scanner;
public class Numero_positivo {
public static void main(String[] args) {
// Crear un objeto Scanner para leer la entrada del usuario
Scanner sc = new Scanner(System.in);
	numero=int(input("Ingrese un número"))
	#Estructura selectiva simple para verificar si el número es positivo
	if (numero > 0):
	print("El número es positivo.")

```
        System.out.println("Ingrese un número");
        int numero=sc.nextInt();
         // Estructura selectiva simple para verificar si el número es positivo
        if (numero > 0) {
            System.out.println("El número es positivo.");
        }
        // Fin del programa
        System.out.println("Programa finalizado.");
    }
}
```

Diagrama de Flujo

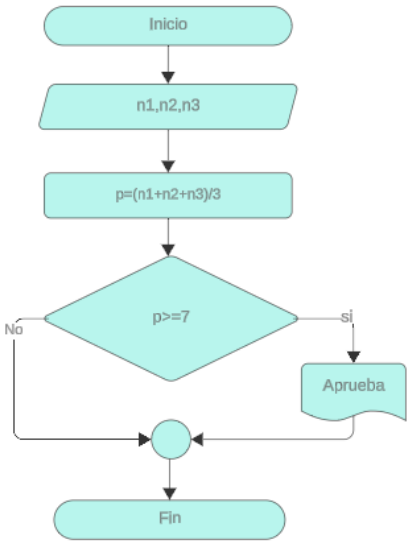


Ejemplo 2: Desarrollar un algoritmo que, en base a tres notas ingresadas por teclado, me permita obtener el promedio de un estudiante, si el promedio es mayor o igual a siete, imprimir un mensaje que diga, el estudiante aprueba.

Ejemplo en Java	*Ejemplo en Python*

Código ejecutable
package algoritmos_selectivos;
import java.util.Scanner;
public class Promedio {
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
// Solicitar al usuario que ingrese las tres notas
System.out.println("Ingrese la primera nota: ");
double nota1 = sc.nextDouble();
System.out.println("Ingrese la segunda nota: ");
double nota2 = sc.nextDouble();
System.out.println("Ingrese la tercera nota: ");
double nota3 = sc.nextDouble();
// Calcular el promedio
double promedio = (nota1 + nota2 + nota3) / 3;
// Estructura selectiva simple para determinar si el estudiante aprueba
if (promedio >= 7) {
System.out.println("El estudiante aprueba con un promedio de: " + promedio);
}
}
}
	# Solicitar al usuario que ingrese las tres notas
	nota1 = float(input("Ingrese la primera nota: \n"))
	nota2 = float(input("Ingrese la segunda nota: \n"))
	nota3 = float(input("Ingrese la tercera nota: \n"))
	# Calcular el promedio
	promedio = (nota1 + nota2 + nota3) / 3
	# Estructura selectiva simple para determinar si el estudiante aprueba
	if promedio >= 7:
	print("El estudiante aprueba con un promedio de:", promedio)

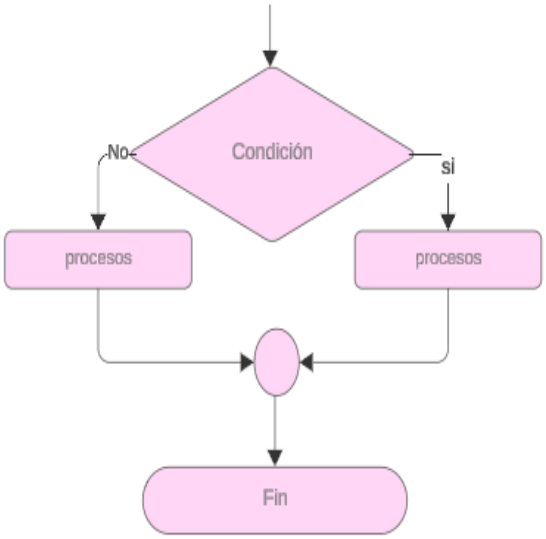
Diagrama de Flujo



Estructuras Algorítmicas Selectivas Dobles

Las estructuras selectivas dobles permiten controlar el flujo de ejecución de un algoritmo mediante la evaluación de una condición lógica que determina dos posibles caminos. Cuando la condición es verdadera, se ejecuta un bloque específico de instrucciones; cuando es falsa, se ejecuta un bloque alternativo. Esta estructura se implementa comúnmente mediante la sentencia if-else, que permite establecer decisiones excluyentes dentro de un programa (Franchitti, 2024).

Ilustración 5 Algoritmo Selectivo Dobles



Nota. Estas estructuras evalúan la condición, y de acuerdo al resultado, realizan procesos por el camino del sí y por camino del no.

Características

- **Bifurcación**: Proveen dos caminos de ejecución.
- **Condiciones Múltiples**: Evalúan al menos dos condiciones.

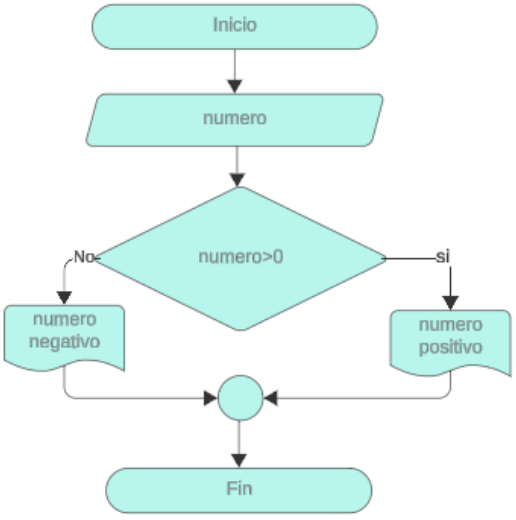
Ejemplos

Ejemplo 1: ingresar un entero y clasificarlo como positivo, negativo o cero; entradas de prueba: 5, −2 y 0.

Ejemplo en Java	*Ejemplo en Python*

Código ejecutable
package numero;
import java.util.Scanner;
public class Numero_positivo {
public static void main(String[] args) {
// Crear un objeto Scanner para leer la entrada del usuario
Scanner sc = new Scanner(System.in);
System.out.println("Ingrese un número");
int numero=sc.nextInt();
// Estructura selectiva simple para verificar si el número es positivo
if (numero > 0) {
System.out.println("El número es positivo.");
}
// el camino del no
else if (numero == 0) {
System.out.println("El número es cero.");
} else {
System.out.println("El número es negativo.");
}
}
}
	numero=int(input("Ingrese un número"))
	#Estructura selectiva simple para verificar si el número es positivo
	if (numero > 0):
	print("El número es positivo.")
	elif numero == 0:
	print("El número es cero.")
	else:
	print("El número es negativo.")

Diagrama de Flujo



Ejemplo 2: Desarrollar un algoritmo que, en base a tres notas ingresadas por teclado, me permita obtener el promedio de un estudiante, si el promedio es mayor o igual a siete, imprimir un mensaje que diga, el estudiante aprueba, caso contrario el estudiante reprueba.

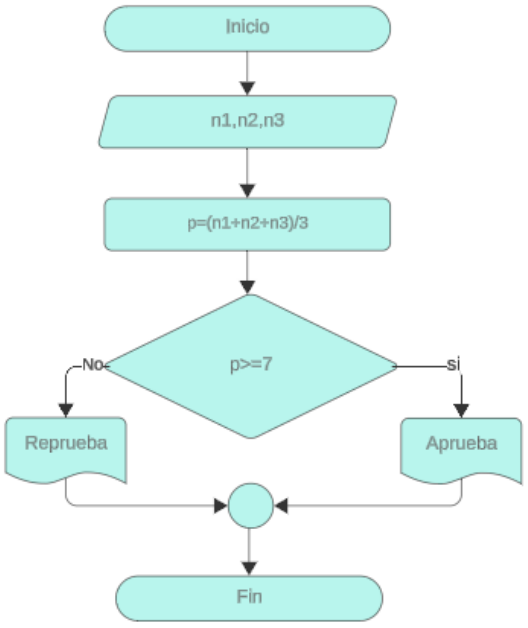
Ejemplo en Java	*Ejemplo en Python*

Código ejecutable
package algoritmos_selectivos;
import java.util.Scanner;
public class Promedio {
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
// Solicitar al usuario que ingrese las tres notas
System.out.println("Ingrese la primera nota: ");
double nota1 = sc.nextDouble();
	# Solicitar al usuario que ingrese las tres notas
	nota1 = float(input("Ingrese la primera nota: \n"))
	nota2 = float(input("Ingrese la segunda nota: \n"))
	nota3 = float(input("Ingrese la tercera nota: \n"))
	# Calcular el promedio
	promedio = (nota1 + nota2 + nota3) / 3
	# Estructura selectiva simple para determinar si el estudiante aprueba
	if promedio >= 7:

```
        System.out.println("Ingrese la segunda nota: ");
        double nota2 = sc.nextDouble();
        System.out.println("Ingrese la tercera nota: ");
        double nota3 = sc.nextDouble();
        // Calcular el promedio
        double promedio = (nota1 + nota2 + nota3) / 3;
        // Estructura selectiva simple para determinar si el estudiante aprueba
        if (promedio >= 7) {
            System.out.println("El estudiante aprueba con un promedio de: " + promedio);
        }
    else {
    System.out.println("El estudiante reprueba con un promedio de: " + promedio);
        }
      }
    }
```

```
    print("El estudiante aprueba con un promedio de:", promedio)
  else:
    print("El estudiante reprueba con un promedio de:", promedio)
```

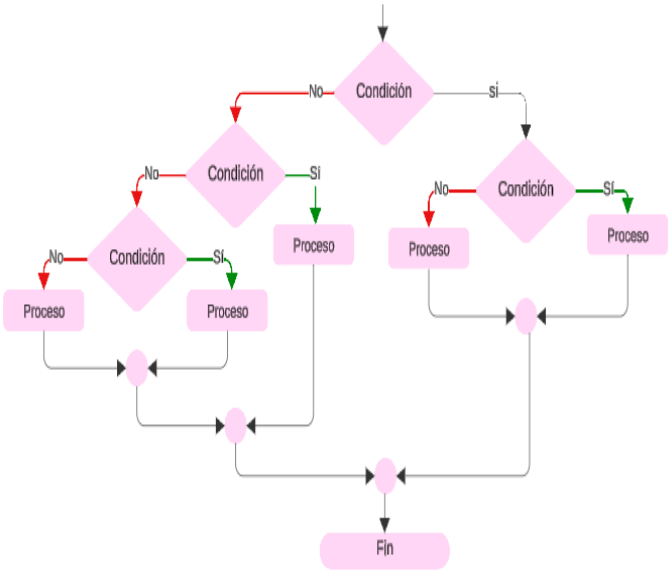
Diagrama de Flujo



Estructuras Algorítmicas Selectivas Múltiples

Las estructuras algorítmicas selectivas múltiples permiten controlar el flujo de ejecución de un programa mediante la selección de una alternativa entre varias opciones disponibles. Su funcionamiento se fundamenta en la evaluación de una expresión o de determinadas condiciones que permiten identificar el bloque de instrucciones que debe ejecutarse. En el lenguaje de programación Java, la sentencia switch constituye un ejemplo representativo de esta estructura, ya que permite comparar el valor de una expresión con diferentes alternativas definidas mediante etiquetas case. Asimismo, puede incluir una opción default para gestionar los casos que no coincidan con las alternativas establecidas (Oracle, 2025).

Ilustración 6 Algoritmos Selectivos Múltiples



Nota. Estas estructuras permiten evaluar más de dos condiciones y ejecutar diferentes bloques de código según el resultado de la evaluación.

Características

- **Diversidad de Caminos**: Permiten múltiples resultados posibles.
- **Clara Estructuración**: Mejoran la legibilidad y organización del código.

Ejemplos

Ejemplo 1: Desarrollar un algoritmo que me permita ingresar un número del 1 al 7 y según el número ingresado imprimir el día de la semana correspondiente.

Ejemplo en Java	*Ejemplo en Python*

Ejemplo en Java:

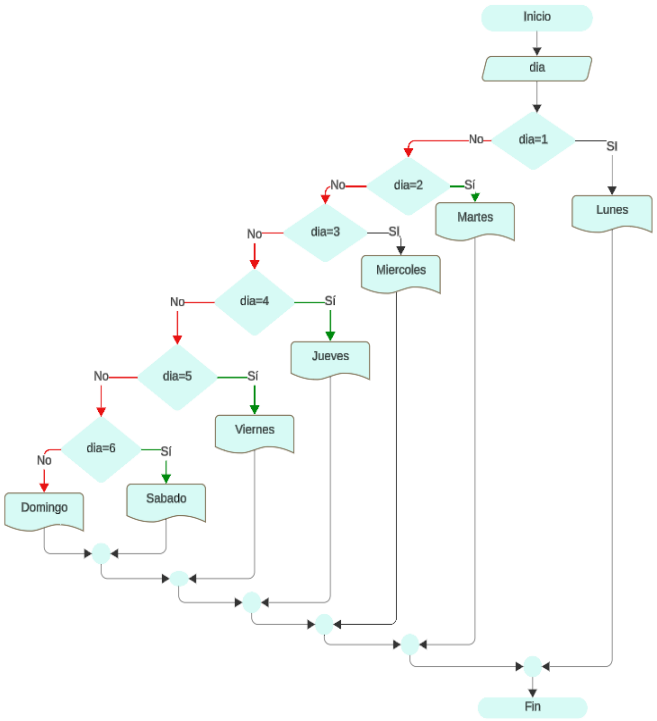
Código ejecutable

```
package algoritmos_selectivos;
import java.util.Scanner;
public class Dias_semana {
   public static void main(String[] args)
{
        Scanner scanner = new
Scanner(System.in);
        // Solicitar al usuario que ingrese
un número del 1 al 7
        System.out.println("Ingrese un
número del 1 al 7: ");
        int dia = scanner.nextInt();
        // Estructura selectiva múltiple
utilizando if-else if-else
        if (dia == 1){
        System.out.println("Lunes");
        }
        else if (dia == 2) {
        System.out.println("Martes");
        }
        else if (dia == 3) {
        System.out.println("Miércoles");
        }
        else if (dia == 4) {
        System.out.println("Jueves");
        }
        else if (dia == 5) {
        System.out.println("Viernes");
        }
        else if (dia == 6) {
        System.out.println("Sábado");
        }
        else if (dia == 7) {
```

```
# Solicitar al usuario que ingrese un
número del 1 al 7
dia=int(input("Ingrese un número del 1
al 7: \n"))
#Estructura selectiva múltiple
utilizando if-else if-else
if (dia == 1):
   print("Lunes")
elif(dia == 2) :
   print("Martes")
elif(dia == 3) :
   print("Miercoles")
elif(dia == 4) :
   print("Jueves")
elif(dia == 5) :
   print("viernes")
elif(dia == 6) :
   print("Sabado")
elif(dia == 7) :
   print("Domingo")
else:
   print("Número no válido. Ingrese un
número del 1 al 7.")
```

```
            System.out.println("Domingo");
            }
            else {
            System.out.println("Número no válido. Ingrese un número del 1 al 7.");
            }
            // Fin del programa System.out.println("Programa finalizado.");
        }
    }
```

Diagrama de Flujo



Ejemplo 2: Desarrollar un algoritmo que me permita ingresar una calificación numérica, según la calificación le mostrará la letra correspondiente en una escala de calificaciones típica (A, B, C, D, F).

Ejemplo en Java	*Ejemplo en Python*

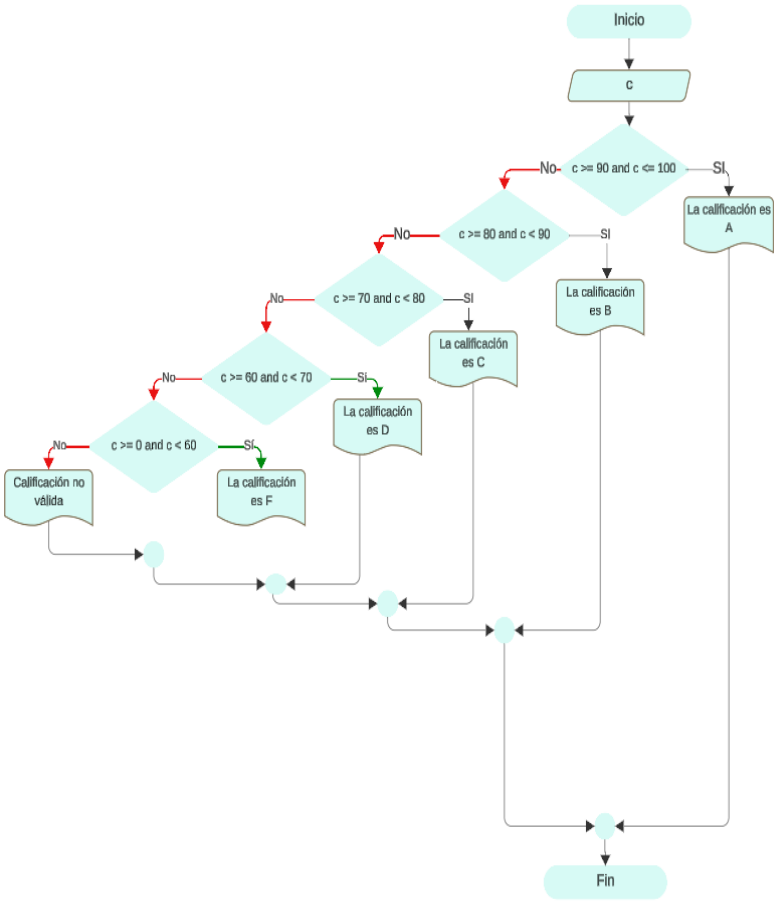
Código ejecutable

```
package algoritmos_selectivos;
import java.util.Scanner;
public class Calificación_numérica {
   public static void main(String[] args)
{
      Scanner scanner = new
Scanner(System.in);
      // Solicitar al usuario que ingrese
una calificación numérica
      System.out.println("Ingrese una
calificación numérica (0-100): ");
      int c = scanner.nextInt();
      // Estructura selectiva múltiple
utilizando if-else if-else
      if (c >= 90 && c <= 100) {
         System.out.println("La
calificación es A."); }
      else if (c >= 80 && c< 90) {
         System.out.println("La
calificación es B."); }
      else if (c >= 70 && c < 80) {
         System.out.println("La
calificación es C."); }
      else if (c >= 60 && c< 70) {
         System.out.println("La
calificación es D."); }
      else if (c>= 0 && c < 60) {
         System.out.println("La
calificación es F."); }
      else {
```

```
#Solicitar al usuario que ingrese una
calificación numérica
c=int(input("Ingrese una calificación
numérica (0-100): \n "))
#Estructura selectiva múltiple
utilizando if-else if-else
if (c >= 90 and c <= 100):
   print("La calificación es A.")
elif (c >= 80 and c < 90):
   print("La calificación es B.")
elif (c >= 70 and c < 80):
   print("La calificación es C.")
elif (c >= 60 and c < 70):
   print("La calificación es D.")
elif (c >= 0 and c < 60):
   print("La calificación es F.")
else:
   print("Calificación no válida. Ingrese
un número entre 0 y 100.")
```

```
System.out.println("Calificación no válida. Ingrese un número entre 0 y 100.");
            }
        }
    }
```

Diagrama de Flujo



3.2. Estructuras Algorítmicas Cíclicas

3.2.1. Introducción a las Estructuras Algorítmicas Cíclicas

Las estructuras algorítmicas cíclicas, también conocidas como estructuras repetitivas o de iteración, permiten ejecutar un bloque de instrucciones de manera reiterada mientras se cumpla una condición lógica o hasta alcanzar un número determinado de repeticiones. Estas estructuras son fundamentales en el desarrollo de algoritmos, ya que facilitan la automatización de procesos repetitivos, optimizan la organización del código y evitan la duplicación innecesaria de instrucciones. Entre las estructuras cíclicas más utilizadas se encuentran los ciclos while, do-while y for, que permiten controlar el flujo de ejecución según las características del problema que se desea resolver (Franchitti, 2024).

Características

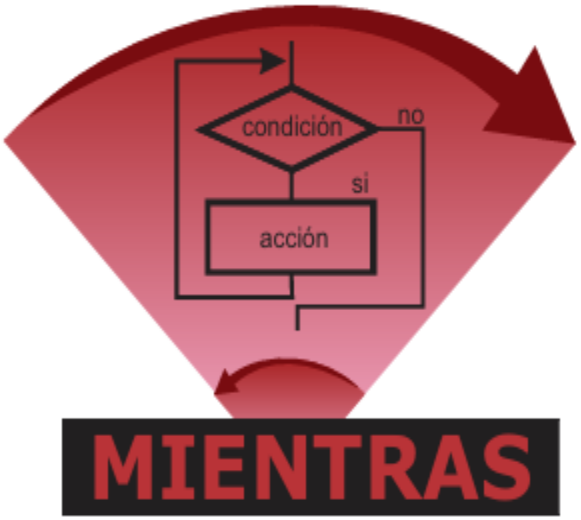
- **Repetición**: Ejecutan múltiples iteraciones.
- **Condicionalidad**: Basadas en una condición lógica.
- **Eficiencia**: Optimizan la ejecución de tareas repetitivas.

Clasificación

Las estructuras algorítmicas cíclicas se clasifican en: Estructuras While, Do While y For.

Estructuras Algorítmicas Cíclicas While

Ilustración 7 Estructuras Algorítmicas Cíclicas While



Nota. La estructura While evalúa la condición al inicio del algoritmo. *(Formación en Ambientes Virtuales de Aprendizaje, s.f)*

Las estructuras algorítmicas cíclicas while permiten ejecutar repetidamente un bloque de instrucciones mientras se cumpla una condición lógica. Esta condición se evalúa antes de cada iteración; si es verdadera, se ejecutan las instrucciones contenidas en el ciclo y, si es falsa, el ciclo finaliza. Debido a que la condición se verifica antes de ejecutar las instrucciones, es posible que el bloque de código no se ejecute ninguna vez cuando la condición inicial sea falsa (Franchitti, 2024).

Características

- **Evaluación Previa**: La condición se evalúa antes de la ejecución del bloque.
- **Ejecución Condicional**: Solo se ejecuta si la condición es verdadera.

Ejemplos

Ejemplo 1: Desarrollar un algoritmo que permita ingresar números positivos y calcula la suma de estos números. El bucle continúa hasta que el usuario ingresa un número negativo.

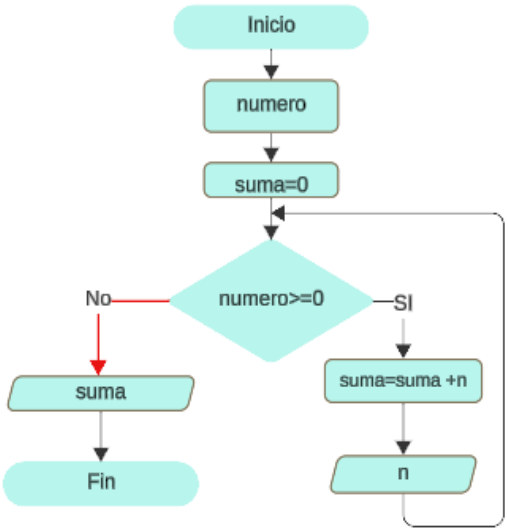
Ejemplo en Java	*Ejemplo en Python*

Código ejecutable
import java.util.Scanner;
 	
public class MenuWhile {
public static void main(String[] args) {
Scanner entrada = new Scanner(System.in);
int opcion = -1; // Inicializar la opción con un valor distinto de 0
// Bucle while que se repite mientras el usuario no elija salir
while (opcion != 0) {
// Mostrar la opción disponible
System.out.println("0. Salir");
	# Solicitar al usuario que ingrese números positivos
	numero = int(input(
	"Ingrese un número positivo (o un número negativo para terminar):\n"
	))
	suma = 0
	# Bucle while para sumar los números ingresados mientras sean no negativos
	while numero >= 0:
	suma += numero # Sumar el número ingresado al total
	# Solicitar al usuario que ingrese otro número
	numero = int(input(
	"Ingrese otro número positivo (o un número negativo para terminar):\n"

```
            // Solicitar al usuario que ingrese una opción
            System.out.print("Elija 0 para terminar: ");
            opcion = entrada.nextInt();
        }
        // Cerrar el lector de datos cuando finalice el programa
        entrada.close();
    }
}
}
```

```
    ))
# Mostrar la suma total de los números ingresados
print(f"La suma de los números positivos ingresados es: {suma}")
```

Diagrama de Flujo



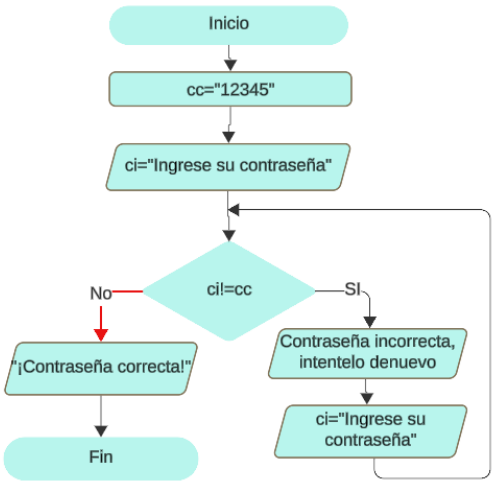
Ejemplo 2: Desarrollar un algoritmo que le permita al usuario ingresar una contraseña y continúa solicitándola hasta que ingrese la correcta.

Ejemplo en Java	*Ejemplo en Python*

Código ejecutable
package algoritmos_ciclicos_while;
import java.util.Scanner;
public class Contrasena {
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
// Clave fija usada únicamente para demostrar el ciclo while.
// No se debe usar este método para proteger cuentas o información real.
String claveDemostrativa = "demo";
String claveIngresada;
// Solicitar la clave al usuario
System.out.print("Ingrese la clave de demostración: ");
claveIngresada = scanner.nextLine();
// Repetir la solicitud mientras la clave ingresada sea incorrecta
while (!claveIngresada.equals(claveDemostrativa)) {
System.out.println("Clave incorrecta. Inténtelo de nuevo.");
System.out.print("Ingrese la clave de demostración: ");
claveIngresada = scanner.nextLine();
}
// Informar que la clave ingresada es correcta
System.out.println("¡Clave correcta!");
// Cerrar el lector de datos
	# Clave fija para demostrar el funcionamiento del ciclo while.
	# No usar este método para proteger cuentas o información real.
	clave_demostrativa = "demo"
	# Solicitar al usuario que ingrese la clave
	clave_ingresada = input("Ingrese la clave de demostración: ")
	# Repetir la solicitud mientras la clave ingresada sea incorrecta
	while clave_ingresada != clave_demostrativa:
	print("Clave incorrecta. Inténtelo de nuevo.")
	clave_ingresada = input("Ingrese la clave de demostración: ")
	# Mostrar el mensaje cuando la clave ingresada sea correcta
	print("¡Clave correcta!")

```
        scanner.close();
    }
}
```

Diagrama de Flujo



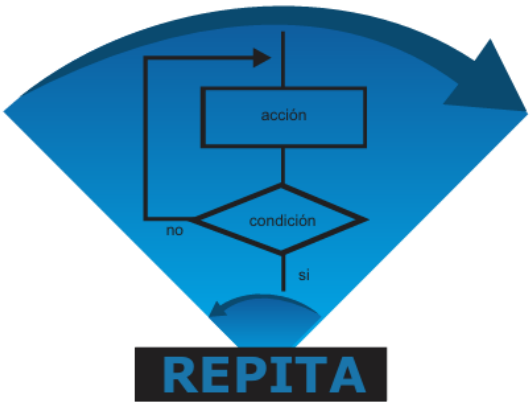
Estructuras Algorítmicas Cíclicas Do-While

Las estructuras Do-while es diferente a la estructura While ya que la condición se evalúa después de la primera iteración asegurándose de esta manera de que el bloque de código se ejecute al menos una vez (Dennis et al., 2021).

Características

- **Evaluación Posterior**: La condición se evalúa después de la ejecución del bloque.

Ilustración 8 Estructuras Algorítmicas Cíclicas Do-While



Nota. La estructura D-While ejecuta los procesos por lo menos una vez. *(Formación en Ambientes Virtuales de Aprendizaje, s.f)*

- **Garantía de Ejecución**: El bloque se ejecuta al menos una vez.

Ejemplos

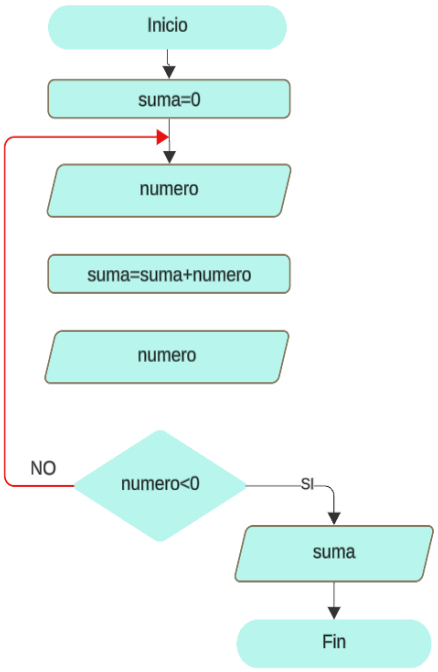
Ejemplo 1: Desarrollar un algoritmo que permita ingresar números positivos y calcula la suma de estos números. El bucle continúa hasta que el usuario ingresa un número negativo.

Ejemplo en Java	*Ejemplo en Python*

Código ejecutable
package algoritmos_do_while;
import java.util.Scanner;
public class NumerosPositivos {
public static void main(String[] args) {
// Crear un lector para recibir números ingresados por teclado
Scanner scanner = new Scanner(System.in);
int numero;
int suma = 0;
// Ejecutar el bloque al menos una vez y solicitar un número
do {
System.out.print(
"Ingrese un número positivo (o uno negativo para terminar): "
);
numero = scanner.nextInt();
// Sumar únicamente los números mayores que cero
if (numero > 0) {
suma += numero;
}
// Repetir mientras el usuario ingrese un número no negativo
} while (numero >= 0);
	suma = 0
	
	# Solicitar el primer número
	numero = int(input(
	"Ingrese un número positivo (o uno negativo para terminar):\n"
	))
	# Repetir mientras el número ingresado no sea negativo
	while numero >= 0:
	# Sumar únicamente los números mayores que cero
	if numero > 0:
	suma += numero
	# Solicitar otro número
	numero = int(input(
	"Ingrese otro número positivo (o uno negativo para terminar):\n"
	))
	# Mostrar la suma cuando el usuario ingrese un número negativo
	print("La suma de los números positivos ingresados es:", suma)

```
            // Mostrar la suma de los números positivos ingresados
            System.out.println("La suma de los números positivos es: " + suma);
            // Cerrar el lector de datos
            scanner.close();
        }
    }
```

Diagrama de Flujo



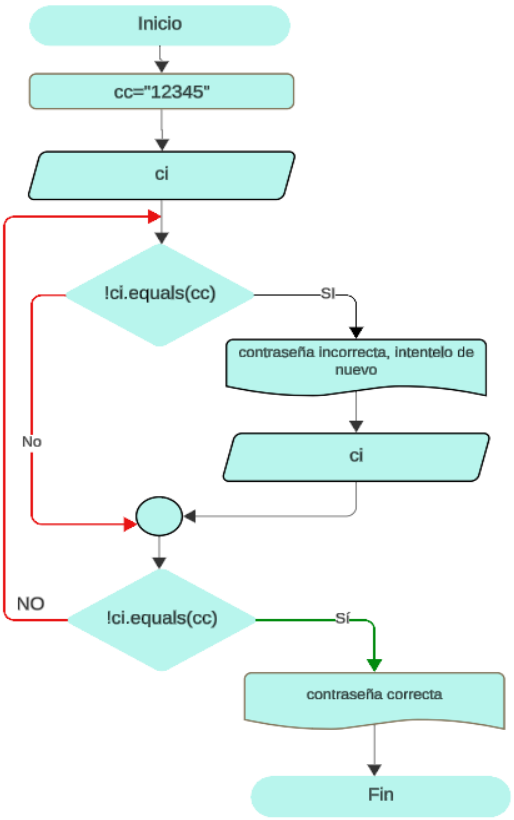
Ejemplo 2: Desarrollar un algoritmo que le permita al usuario ingresar una contraseña y continúa solicitándola hasta que ingrese la correcta.

Ejemplo en Java	*Ejemplo en Python*

Código ejecutable
package algoritmos_do_while;
import java.util.Scanner;
public class Contrasena {
public static void main(String[] args) {
// Crear un lector para recibir texto ingresado por teclado
Scanner scanner = new Scanner(System.in);
// Clave fija utilizada únicamente para demostrar el ciclo do-while
String claveDemostrativa = "demo";
String claveIngresada;
// Solicitar la clave al menos una vez
do {
System.out.print("Ingrese la clave de demostración: ");
claveIngresada = scanner.nextLine();
// Avisar si la clave ingresada no coincide con la clave demostrativa
if (!claveIngresada.equals(claveDemostrativa)) {
System.out.println("Clave incorrecta. Inténtelo de nuevo.");
}
// Repetir la solicitud mientras la clave sea incorrecta
} while (!claveIngresada.equals(claveDemostrativa));
// Mostrar un mensaje cuando la clave ingresada sea correcta
	# Clave fija utilizada únicamente para demostrar el funcionamiento del ciclo.
	# No usar este método para proteger cuentas o información real.
	clave_demostrativa = "demo"
	# Repetir la solicitud hasta que se ingrese la clave correcta
	while True:
	clave_ingresada = input("Ingrese la clave de demostración: ")
	
	# Si la clave coincide, salir del ciclo
	if clave_ingresada == clave_demostrativa:
	break
	# Informar que la clave ingresada no es correcta
	print("Clave incorrecta. Inténtelo de nuevo.")
	# Mostrar un mensaje cuando finalice el ciclo
	print("¡Clave correcta!")

```
            System.out.println("¡Clave
correcta!");
            // Cerrar el lector de datos
            scanner.close();
        }
    }
```

Diagrama de Flujo



Estructuras Algorítmicas Cíclicas For

Las estructuras algorítmicas cíclicas, también denominadas estructuras repetitivas o de iteración, permiten ejecutar un bloque de instrucciones de manera reiterada, según el cumplimiento de una condición lógica o un número determinado de repeticiones. Estas estructuras son fundamentales en el desarrollo de algoritmos, ya que facilitan la automatización de tareas que requieren ejecutarse varias veces, evitando la duplicación innecesaria de instrucciones. Su aplicación depende de las condiciones y características del problema que se desea resolver (Franchitti, 2024)

Ilustración 9 Estructuras Algorítmicas Cíclicas For



Nota. Esta estructura es ideal cuando se conoce el número de interacciones de debe cumplir el algoritmo, *(Formación en Ambientes Virtuales de Aprendizaje, s.f)*

Características

- **Control de Iteración**: Proveen un control preciso sobre el número de iteraciones.
- **Eficiencia**: Suelen ser más eficientes para ciertas tareas repetitivas.

Ejemplos

Ejemplo 1: Desarrollar un algoritmo que calcule el área de un círculo.

Ejemplo en Java	*Ejemplo en Python*

Código ejecutable
package pkgfor;
import java.util.Scanner;
public class Area_circulo {
public static void main(String[] args)
{
	# Importar la biblioteca math para usar el valor de PI
	import math
	# Solicitar al usuario el número de círculos n
	num_circulos = int(input("Ingrese el número de círculos que desea calcular: "))

```
        Scanner scanner = new Scanner(System.in);
        int numCirculos;
        System.out.print("\n Ingrese el número de círculos que desea calcular: ");
        numCirculos = scanner.nextInt();
        for (int i = 1; i <= numCirculos; i++) {
            System.out.print("\n Ingrese el radio del círculo " + i + ": ");
            double radio = scanner.nextDouble();
            double area = Math.PI * Math.pow(radio, 2);
            System.out.println("El área del círculo " + i + " es: " + area);
        }
        scanner.close();
    }
}
```

```
# Bucle para calcular el área de cada círculo
for i in range(1, num_circulos + 1):
    radio = float(input(f"Ingrese el radio del círculo {i}: "))
    area = math.pi * radio ** 2
    print(f"El área del círculo {i} es: {area}")
```

Ejemplo 2: Desarrollar un algoritmo que presente los 10 primeros números.

Ejemplo en Java	*Ejemplo en Python*

Código ejecutable

```
public class NumerosDelUnoAlDiez {
    public static void main(String[] args) {
        // Bucle para imprimir los 10 primeros números
        for (int i = 1; i <= 10; i++) {
            System.out.println(i);
```

Código ejecutable

```
# Bucle para imprimir los 10 primeros números
for i in range(1, 11):
    print(i)
```

```
        }
    }
}
```

ACTIVIDADES DE APRENDIZAJE

Desarrollar los siguientes algoritmos (Java, Python)

Estructuras algorítmicas selectivas Simples.

- ✓ Desarrollar un algoritmo que en base al año de nacimiento de una persona me permita obtener su edad; y determinar si la persona es Adulta.
- ✓ Desarrollar un algoritmo que me permita obtener el promedio de un estudiante; cuando se haya obtenido el promedio, determinar si el estudiante: Aprueba, entendiéndose que para aprobar su promedio debe ser >=7.
- ✓ Desarrolle un algoritmo para sumar dos números, y determinar si el resultado es par.
- ✓ De terminar el Índice de Masa Corporal (imc) a partir de la fórmula p/e, y con ese resultado ver si necesita dieta.

Clasificación	IMC (Kg/m2)	Riesgo
Normal	18.5 – 24.9	Promedio
Sobrepeso	25 – 29.9	Aumentado
Obesidad grado I	30 – 34.9	Moderado
Obesidad grado II	35 – 39.9	Severo
Obesidad grado III	Más de 40	Muy severo

Fuente: OMS (Organización Mundial de la Salud)

- ✓ Elaborar un algoritmo que me permita ingresar el nombre de una persona, si el número de caracteres del nombre ingresado es igual a 6, imprima los 20 primeros números pares.

Estructuras algorítmicas selectivas dobles

- ✓ Obtener el Promedio de (Calcular con tres notas), si su promedio es mayor o igual a 7 mostrar un mensaje que diga el estudiante aprueba, caso contrario Un mensaje que diga el estudiante reprueba.
- ✓ Una empresa decide incrementar el sueldo de sus empleados en un 15%, el requisito es que el trabajador tenga entre 5 a 10 años de servicio; calcule el sueldo actual de los empleados que tendrán este beneficio.
- ✓ Elaborar un algoritmo que me permita sumar dos números, si el número es par; obtener el área de un triángulo, caso contrario obtener el área y perímetro de un rectángulo.
- ✓ Elaborar un algoritmo que me permita ingresar el nombre de una persona, si el número de caracteres del nombre ingresado es igual a 6, imprima los 20 primeros números pares, caso contrario imprimir los 20 primeros números impares.
- ✓ Elaborar un algoritmo que me permita seleccionar una figura geométrica (Círculo o Cuadrado), dependiendo de la figura geométrica seleccionada, obtener:

Círculo con un radio que usted crea conveniente

- Diámetro
- Perímetro
- Área

Cuadrado

- Área
- Perímetro

Estructuras algorítmicas selectivas múltiples

- ✓ Desarrollar un algoritmo que me permita obtener el promedio de un estudiante; cuando se haya obtenido el promedio, determinar si el estudiante:
  - Aprueba (>=7), si el estudiante aprueba, determinar si tiene derecho a una beca, sabiendo que para acceder a una su promedio debe ser >=9.5
  - Suspenso (>=4 y >7), si el estudiante se encuentra suspenso calcular cual es la nota mínima que debe sacarse en el examen para aprobar.
  - Reprueba (<4), si el estudiante aprueba determinar si tiene derecho a una, se queda suspenso o reprueba entendiéndose que para aprobar su promedio deber ser >=7, caso contrario reprueba.
- ✓ Ingrese el año de nacimiento de una persona, con el dato ingresado calcular que edad tiene actualmente; de acuerdo a la edad obtenida determinar si es:
  - bebe (edad entre 0 a 2)
  - niño (edad entre 3 a 12)
  - Adolescente (edad entre 13 a 17)
  - Joven (edad entre 18 a 35)
  - Adulto (edad entre 36 a 60)
  - Adulto mayor (61 años en adelante)
- ✓ Elaborar un algoritmo que me permita ingresar la cédula de una persona, con el dato ingresado determinar de qué provincia es el ciudadano ecuatoriano.
- ✓ Determinar el Índice de Masa Corporal (imc) a partir de la fórmula p/e^2 y con ese resultado ver el riesgo según la tabla que se muestra.

Clasificación	IMC (Kg/m2)	Riesgo
Normal	18.5 – 24.9	Promedio
Sobrepeso	25 – 29.9	Aumentado
Obesidad grado I	30 – 34.9	Moderado
Obesidad grado II	35 – 39.9	Severo
Obesidad grado III	Más de 40	Muy severo

Fuente: OMS (Organización Mundial de la Salud)

- ✓ Elaborar un algoritmo que me permita ingresar los países de América del Sur y de acuerdo al país ingresado mostrar su capital y la moneda que circulan.

Estructuras algorítmicas cíclicas While

- ✓ Desarrollar un algoritmo cíclico While que me permita obtener el promedio de N número de estudiantes; cuando se haya obtenido el promedio, determinar si el estudiante:
  - Aprueba (>=7), si el estudiante aprueba, determinar si tiene derecho a una beca, sabiendo que para acceder a una su promedio debe ser >=9.5
  - Suspenso (>=4 y <7), si el estudiante se encuentra suspenso calcular cual es la nota mínima que debe sacarse en el examen para aprobar.
  - Reprueba (<4)
  - Obtener el promedio general en base a los promedios obtenidos.
- ✓ Ingresar un conjunto de números, sumar los números pares y multiplicar los números impares hasta que la suma sea mayor que 50 o el producto mayor que 150.
- ✓ Una Persona va de compras N veces a un supermercado donde le obsequian un chocolate por cada vez que compra más de $50,00 dólares. Se desea saber cuántos chocolates ha ganado en las N veces que compro.

- ✓ Escribir un programa que solicite la carga de un valor positivo y nos muestre desde 1 hasta el valor ingresado de uno en uno. Ejemplo: Si ingresamos 30 se debe mostrar en pantalla los números del 1 al 30.
- ✓ Desarrollar un programa que permita la carga de 10 valores por teclado y nos muestre posteriormente la suma de los valores ingresados y su promedio.

Estructuras algorítmicas cíclicas Do-While

- ✓ Desarrollar los ejercicios de la estructura While

Estructuras algorítmicas cíclicas For

- ✓ Desarrollar un algoritmo que me permita obtener la tabla de multiplicar que el usuario ingrese por teclado
- ✓ Imprimir los 50 números impares
- ✓ Desarrollar un algoritmo que me permita ingresar el año de nacimiento de 10 personas, según en año de nacimiento determinar su edad, y que indique cuantos son mayores y menores de edad.
- ✓ Desarrollar un algoritmo que me permita ingresar letras hasta que se ingrese una vocal. Imprimir el número de letras ingresadas (sin contar la vocal) Considerar que solo se van a ingresar solo letras minúsculas.
- ✓ Desarrollar un algoritmo que me permita ingresar por teclado 10 números, el programa debe indicar cuales de estos números son pares y cuáles no.

CLASIFICAR UNA NOTA

Ejemplo en Java	***Ejemplo en Python***

Código ejecutable
import java.util.Scanner;
public class ClasificarNotaRevision {
public static void main(String[] args)
{
Scanner entrada = new
Scanner(System.in);
	nota = float(input())
	if nota < 0 or nota > 10:
	print("Fuera de rango")
	elif nota >= 7:
	print("Aprueba")
	else:
	print("No aprueba")

```
            double nota =
entrada.nextDouble();
            if (nota < 0 || nota > 10) {
                System.out.println("Fuera de
rango");
            } else if (nota >= 7) {
                System.out.println("Aprueba");
            } else {
                System.out.println("No
aprueba");
            }
            entrada.close();
        }
    }
```

EVALUACIÓN

Lenguaje algorítmico y estructuras de control

Nombre:		**Fecha:**	
Curso / Asignatura:		**Puntaje:**	/ 10

Instrucciones: Lea cada pregunta con atención y encierre en un círculo o marque con una X la letra correspondiente a la respuesta correcta. Cada pregunta tiene un único valor y una sola opción válida.

1. ¿Cuál es la función principal de las estructuras de control selectivas o condicionales en programación?

a) Repetir un bloque de código un número determinado de veces.

b) Dirigir el flujo de ejecución hacia diferentes rutas según el resultado de una condición lógica.

c) Declarar variables numéricas y realizar operaciones aritméticas complejas.

d) Finalizar inmediatamente la ejecución del programa.

2. ¿Qué caracteriza fundamentalmente a una estructura selectiva simple (como la instrucción if)?

a) Posee caminos alternativos tanto para la opción verdadera como para la falsa.

b) Evalúa procesos solo por el camino del "Sí" (verdadero); si es falso, va directo al fin de la estructura.

c) Permite evaluar tres o más condiciones simultáneamente.

d) Se ejecuta en un bucle infinito.

3. ¿Cuál es la diferencia principal entre una estructura selectiva doble y una estructura selectiva simple?

a) La estructura doble ofrece dos caminos posibles (uno si la condición es verdadera y otro si es falsa).

b) La estructura doble no utiliza expresiones lógicas.

c) La estructura simple solo se puede programar en lenguaje Java.

d) La estructura doble repite las instrucciones dos veces.

4. ¿Qué estructura selectiva es ideal para evaluar más de dos caminos o condiciones, como las opciones de un menú o los días de la semana?

a) Estructura Selectiva Simple.

b) Estructura Cíclica While.

c) Estructura Selectiva Múltiple (ej. switch o if-else if).

d) Estructura Repetitiva For.

5. ¿Cuál es la característica principal de la estructura cíclica While (Mientras)?

a) Evalúa la condición al final de la iteración.

b) Evalúa la condición antes de cada iteración, ejecutando el bloque solo si es verdadera.

c) Se ejecuta siempre un número fijo de 10 veces.

d) No requiere condición lógica de salida.

6. ¿Qué diferencia fundamental existe entre la estructura cíclica Do-While y la estructura While?

a) Do-While garantiza que el bloque de código se ejecute al menos una vez, ya que la condición se evalúa al final.

b) While se ejecuta siempre al menos una vez y Do-While cero veces.

c) Do-While solo funciona con números enteros.

d) No existe ninguna diferencia, ambas se comportan idénticamente.

7. ¿En qué situación es más recomendable utilizar la estructura cíclica For (Para)?

a) Cuando se desconoce totalmente cuándo finalizará el bucle.

b) Cuando se requiere repetir una operación hasta que se seleccione una opción de salida.

c) Cuando se conoce de antemano el número exacto de iteraciones que se deben realizar.

d) Únicamente para verificar si un número es par o impar.

8. Si un estudiante obtiene las notas 8, 7 y 9, y el algoritmo evalúa promedio >= 7 para determinar si aprueba, ¿cuál es el resultado del promedio y de la condición?

a) Promedio = 8; Condición Falsa.

b) Promedio = 8; Condición Verdadera (Imprime 'El estudiante aprueba').

c) Promedio = 24; Condición Falsa.

d) Promedio = 7; Condición Falsa.

9. En el contexto de estructuras condicionales, ¿qué valores evalúa o devuelve una expresión lógica?

a) Solo valores enteros (1, 2, 3...).

b) Verdadero (True) o Falso (False).

c) Texto o cadenas de caracteres únicamente.

d) Números decimales o flotantes.

10. ¿Qué estructura permite repetir un menú hasta que el usuario seleccione la opción 0 para salir?

a) Estructura Selectiva Simple (if).

b) Estructura Cíclica (While o Do-While).

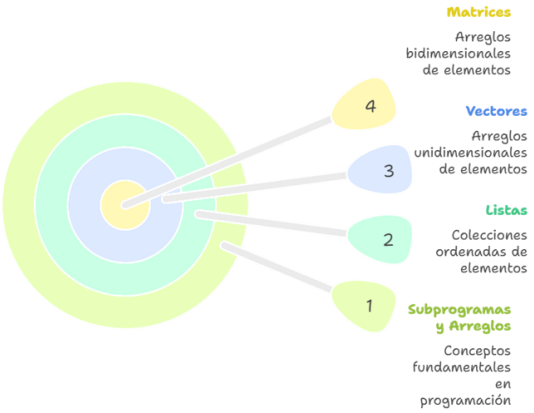
c) Estructura Selectiva Múltiple (switch).

d) Secuencia directa sin condicionales ni bucles.

Unidad 4:

Arreglos, listas y matrices

Diagrama de aprendizaje



Resumen

La Unidad 4 aborda tres estructuras fundamentales en la programación estructurada: listas, vectores y matrices. Estas herramientas permiten organizar, almacenar y manipular datos de forma eficiente, facilitando el desarrollo de algoritmos y la resolución de problemas computacionales complejos.

- Las **listas** se caracterizan por su flexibilidad y capacidad dinámica, lo que las convierte en estructuras ideales para gestionar colecciones de datos cuyo tamaño varía durante la ejecución del programa. Su implementación puede variar según el lenguaje, incluyendo listas enlazadas, dobles o circulares, cada una con ventajas específicas en términos de rendimiento y accesibilidad.
- Los **vectores**, o arreglos unidimensionales, permiten almacenar elementos homogéneos en posiciones contiguas de memoria, son esenciales para representar secuencias ordenadas y realizar operaciones como búsqueda, ordenamiento y análisis estadístico. Su uso es común en aplicaciones que requieren rapidez y eficiencia en el acceso a datos.
- Las matrices organizan valores en filas y columnas. En esta guía se trabajan recorridos y operaciones sencillas; la forma de declarar y recorrerlas depende del lenguaje. Java permite arreglos multidimensionales con filas de longitudes distintas; las listas anidadas de Python también pueden tener longitudes diferentes (Oracle, s. f.; Python Software Foundation, 2025).

En conjunto, estas estructuras permiten al programador diseñar soluciones robustas, escalables y adaptadas a distintos contextos de aplicación, desde el análisis de datos hasta el desarrollo de sistemas inteligentes.

4. Listas

4.1 Introducción

Las listas son estructuras de datos fundamentales en programación, utilizadas para almacenar colecciones ordenadas de elementos, a diferencia de los arreglos tradicionales, las listas permiten una gestión dinámica de la memoria, lo que significa que pueden crecer o reducirse durante la ejecución del programa. Esta flexibilidad las convierte en herramientas esenciales para resolver problemas donde el número de elementos no está definido previamente, en lenguajes como Python, las listas son estructuras versátiles que admiten distintos tipos de datos y operaciones como inserción, eliminación y búsqueda de elementos (González, 2020).

4.1.1. Características

Las listas pueden ser implementadas de diversas formas, como listas enlazadas, listas dobles o listas circulares, cada una con características específicas que optimizan el rendimiento según el tipo de aplicación, por ejemplo, las listas enlazadas permiten una inserción eficiente sin necesidad de mover elementos, lo que resulta útil en algoritmos que requieren modificaciones frecuentes, su uso se extiende a áreas como la inteligencia artificial, la gestión de bases de datos y el desarrollo de interfaces gráficas (Dennis et al., 2021).

Ejemplo 1. Desarrollar un algoritmo que me permita listar un conjunto de frutas.

Ejemplo en Java	*Ejemplo en Python*

Código ejecutable
import java.util.ArrayList;
public class ListaEjemplo {
public static void main(String[] args) {
// Crear una lista dinámica para almacenar nombres de frutas
ArrayList<String> frutas = new ArrayList<>();
	Código ejecutable
	# Lista de frutas
	frutas = ["manzana", "banana", "naranja"]
	frutas.append("mango") # Agregar elemento
	frutas.remove("banana") # Eliminar elemento

```
        // Agregar frutas a la lista
        frutas.add("manzana");
        frutas.add("banana");
        frutas.add("naranja");
        // Eliminar "banana" de la lista
        frutas.remove("banana");
        // Agregar "mango" al final de la lista
        frutas.add("mango");
        // Mostrar el contenido actual de la lista
        System.out.println(frutas);        // [manzana, naranja, mango]
    }
}
```

```
print(frutas)  # ['manzana', 'naranja', 'mango']
```

4.2 Operaciones y recorridos en vectores

4.2.1. Introducción

Los vectores, también conocidos como arreglos unidimensionales, son estructuras de datos que almacenan elementos del mismo tipo en posiciones contiguas de memoria, cada elemento se accede mediante un índice, lo que permite una manipulación rápida y directa. En programación estructurada, los vectores son esenciales para representar secuencias de datos como puntuaciones, temperaturas o registros numéricos, su uso es común en algoritmos de ordenamiento, búsqueda y análisis estadístico (Universidad Nacional de Rosario, 2020).

4.2.2. Características

Una característica importante de los vectores es que su tamaño puede ser fijo o dinámico, dependiendo del lenguaje de programación utilizado, en lenguajes como C, el tamaño se define en tiempo de compilación, mientras que en otros como Java o Python, se puede modificar durante la ejecución. Esta capacidad de adaptación permite que los

vectores sean utilizados en aplicaciones que requieren eficiencia y rapidez en el acceso a datos, como simulaciones científicas y procesamiento de señales (UNLaM, 2020).

Ejemplo: Mostar la edad de un conjunto de personas.

Ejemplo en Java	*Ejemplo en Python*

Código ejecutable
public class VectorEjemplo {
public static void main(String[] args)
{
int[] edades = {23, 45, 18, 30};
for (int edad : edades) {
System.out.println(edad);
}
}
}
	# Vector de edades
	edades = [23, 45, 18, 30]
	for edad in edades:
	print(edad)

4.3. Matrices

4.3.1. Introducción

Las matrices son arreglos bidimensionales que permiten almacenar datos en forma de filas y columnas. Son ampliamente utilizadas en programación para representar tablas, imágenes, grafos y sistemas de ecuaciones. Cada elemento de una matriz se accede mediante dos índices, lo que facilita la organización y manipulación de grandes volúmenes de información. En el ámbito académico, las matrices son fundamentales para el desarrollo de algoritmos de álgebra lineal, transformaciones geométricas y análisis numérico (Universidad Nacional del Sur, 2020).

4.3.2. Características

El uso de matrices también se extiende a áreas como la inteligencia artificial, donde se emplean para representar redes neuronales y operaciones de convolución en aprendizaje profundo, en estos contextos, la eficiencia en el manejo de matrices es crucial, por lo que se utilizan bibliotecas especializadas que optimizan el rendimiento computacional.

Además, las matrices pueden ser estáticas o dinámicas, y su implementación varía según el lenguaje de programación, lo que requiere una comprensión profunda de su estructura y funcionamiento (González, 2020).

Ejemplo. Mostar la edad de un conjunto de personas.

Ejemplo en Java	*Ejemplo en Python*

public class MatrizEjemplo {
public static void main(String[] args) {
// Crear una matriz de 2 filas y 2 columnas
int[][] matriz = {
{1, 2},
{3, 4}
};
// Recorrer cada fila de la matriz
for (int i = 0; i < matriz.length; i++) {
// Recorrer cada elemento de la fila actual
for (int j = 0; j < matriz[i].length; j++) {
// Mostrar el elemento y dejar un espacio
System.out.print(matriz[i][j] + " ");
}
 	
// Saltar a la línea siguiente al terminar cada fila
System.out.println();
}
}
}
	# Matriz 2x2
	matriz = [[1, 2], [3, 4]]
	for fila in matriz:
	for elemento in fila:
	print(elemento, end=" ")
	print()

Ejemplo en Java:

ACTIVIDADES DE APRENDIZAJE

- Listas (Java y Python)

1. Eliminar duplicados: Desarrolla un programa en Java y Python que reciba una lista de nombres con posibles duplicados y retorne una nueva lista sin elementos repetidos.

2. Lista de tareas: Crea una aplicación en Java y Python que permita al usuario agregar tareas a una lista, eliminar tareas completadas y mostrar las tareas pendientes.

3. Ordenar lista de palabras: Escribe un programa en Java y Python que tome una lista de palabras ingresadas por el usuario y las ordene alfabéticamente. Muestra el resultado final.

- Vectores (Java y Python)

1. Promedio de calificaciones: Desarrolla un programa en Java y Python que almacene las calificaciones de 5 estudiantes en un vector y calcule el promedio general.

2. Buscar un número: Escribe un programa en Java y Python que permita al usuario ingresar un número y verifique si ese número está presente en un vector de 10 elementos.

3. Invertir vector: Crea un programa en Java y Python que tome un vector de números enteros y lo imprima en orden inverso.

- Matrices (Java y Python)

1. Suma de dos matrices: Desarrolla un programa en Java y Python que permita sumar dos matrices de 3x3 y mostrar la matriz resultante.

2. Matriz identidad: Escribe un programa en Java y Python que genere una matriz identidad de tamaño 4x4.

3. Transposición de matriz: Crea un programa en Java y Python que tome una matriz de 3x2 y retorne su transpuesta (de 2x3).

Ejemplo: PROMEDIO DE UNA COLECCIÓN

Ejemplo en Java	*Ejemplo en Python*

public class PromedioArregloRevision
{
public static void main(String[] args)
{
double[] notas = {8.0, 7.0, 9.0};
double suma = 0;
for (double nota : notas) { suma +=
nota; }
System.out.println("Promedio: " +
(suma / notas.length));
}
}
	notas = [8.0, 7.0, 9.0]
	promedio = sum(notas) / len(notas)
	print("Promedio:", promedio)

EVALUACIÓN

Arreglos, listas y matrices

Nombre:		**Fecha:**	
Curso / Asignatura:		**Puntaje:**	/ 10

Instrucciones: Lea cada pregunta con atención y encierre en un círculo o marque con una X la letra correspondiente a la respuesta correcta. Cada pregunta tiene un único valor y una sola opción válida.

1. ¿Cuál es la característica principal que diferencia a las listas de los arreglos o vectores tradicionales?

a) Las listas solo pueden almacenar valores de tipo entero.

b) Permiten una gestión dinámica de la memoria, pudiendo crecer o reducirse durante la ejecución.

c) Tienen un tamaño fijo definido obligatoriamente en tiempo de compilación.

d) Los elementos de una lista no pueden ser eliminados una vez agregados.

2. En el contexto de las listas enlazadas, ¿cuál es una ventaja clave mencionada en el documento?

a) Ocupan menos espacio en disco que cualquier otra estructura.

b) Permiten una inserción eficiente de elementos sin necesidad de desplazar los demás.

c) No permiten elementos duplicados bajo ninguna circunstancia.

d) Solo pueden implementarse en lenguaje Python.

3. En el lenguaje Python, ¿qué método se utiliza para agregar un nuevo elemento al final de una lista?

a) add()

b) push()

c) append()

d) insert_last()

4. ¿Cómo se define un vector o arreglo unidimensional?

a) Una estructura que almacena elementos de distintos tipos en posiciones aleatorias de memoria.

b) Una tabla bidimensional organizada en filas y columnas.

c) Una estructura que almacena elementos del mismo tipo en posiciones contiguas de memoria.

d) Una colección jerárquica de subprogramas.

5. Respecto a los vectores, ¿en qué lenguaje el tamaño del vector se define típicamente en tiempo de compilación?

a) Python

b) C

c) JavaScript

d) Java con ArrayList

6. ¿Qué mecanismo se utiliza para acceder a un elemento específico dentro de un vector?

a) Un par de coordenadas (fila, columna).

b) Un índice numérico.

c) La clave asociada al elemento.

d) Un puntero de cabecera únicamente.

7. ¿Qué es una matriz en el ámbito de las estructuras de datos?

a) Un subprograma que no retorna valores.

b) Un arreglo unidimensional de cadenas de texto.

c) Un arreglo bidimensional que organiza los datos en forma de filas y columnas.

d) Una lista enlazada simple de números enteros.

8. ¿Cuántos índices se requieren obligatoriamente para acceder a un elemento individual dentro de una matriz?

a) Un solo índice.

b) Dos índices (uno para la fila y otro para la columna).

c) Tres índices.

d) No requiere índices, solo el nombre de la variable.

9. En una matriz, ¿qué representan los dos índices usados para acceder a un elemento?

a) El primero identifica la fila y el segundo la columna.

b) Ambos identifican la columna.

c) El primero identifica la matriz y el segundo el tipo de dato.

d) No se usan índices en una matriz.

10. En los diagramas de código presentados para recorrer una matriz (en Java o Python), ¿qué tipo de estructura de control se utiliza comúnmente?

a) Un único condicional IF.

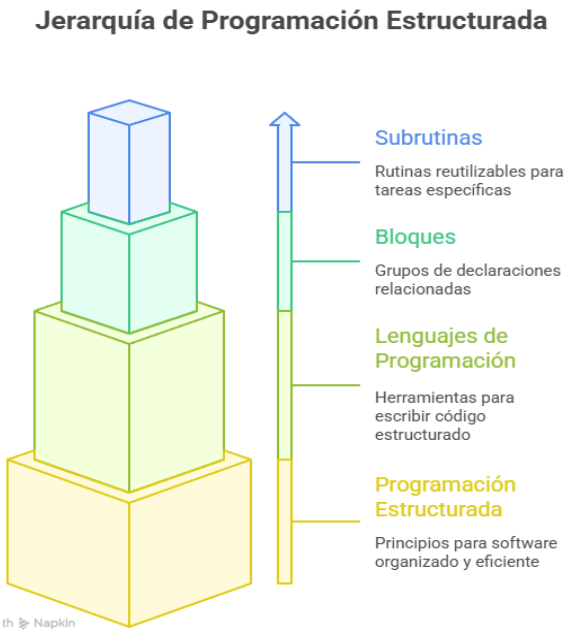
b) Bucles (FOR) anidados.

c) Un bloque TRY-CATCH.

d) Un interruptor SWITCH-CASE.

Unidad 5: Programación estructurada y modular

Diagrama de aprendizaje



Resumen

La programación estructurada es un paradigma de desarrollo de software que promueve la organización lógica y jerárquica del código mediante el uso de estructuras de control como secuencias, decisiones e iteraciones, evitando instrucciones desordenadas, su objetivo principal es mejorar la claridad, la legibilidad y el mantenimiento de los programas, dividiéndolos en bloques funcionales y subrutinas reutilizables. Este enfoque facilita la resolución de problemas complejos al permitir que los desarrolladores construyan algoritmos de manera ordenada, modular y predecible, sentando las bases para paradigmas posteriores como la programación orientada a objetos.

5. Programación estructurada

La programación estructurada surgió durante las décadas de 1960 y 1970 como respuesta a la creciente complejidad de los programas informáticos y a la necesidad de desarrollar métodos que facilitaran su comprensión, organización y mantenimiento. Este paradigma se fundamenta en el uso de tres estructuras básicas de control: **secuencia, selección y repetición**, que permiten organizar el flujo de ejecución de un programa de manera lógica, ordenada y predecible (Böhm & Jacopini, 1966).

En este contexto, Dijkstra (1972) destaca la importancia de aplicar una metodología disciplinada en la construcción de programas, mediante estructuras que faciliten el razonamiento sobre su funcionamiento. La programación estructurada favorece la legibilidad, el modularidad y la verificación de las instrucciones, al tiempo que reduce la dependencia de saltos incondicionales que pueden dificultar la comprensión del código.

Un ejemplo básico de aplicación de la programación estructurada es el cálculo del área de un triángulo mediante una función y una secuencia de instrucciones.

En el lenguaje Python, este procedimiento puede expresarse como:

***Ejemplo en Python*:**

Código ejecutable

```
def calcular_area_triangulo(base, altura):
    area = (base * altura) / 2
    return area
base = float(input("Ingrese la base del triángulo: "))
altura = float(input("Ingrese la altura del triángulo: "))
resultado = calcular_area_triangulo(base, altura)
print("El área del triángulo es:", resultado)
```

Este ejemplo muestra cómo un lenguaje estructurado como Python permite organizar el código en funciones reutilizables y controlar el flujo mediante estructuras condicionales. La claridad del código facilita su comprensión incluso para personas que recién se inician en la programación.

5.1 Subprogramas, funciones y procedimientos

¿Qué son las subrutinas?

Las subrutinas son bloques de código que encapsulan tareas específicas dentro de un programa. También se conocen como funciones o procedimientos, dependiendo del lenguaje de programación utilizado. Su propósito principal es modularizar el código, es decir, dividirlo en partes independientes que puedan ser reutilizadas, comprendidas y mantenidas con facilidad. Esta técnica es fundamental en la programación estructurada, ya que permite construir algoritmos más claros y eficientes (Gaddis, 2023)

¿Por qué son importantes?

- ✓ **Evitan la repetición de código**: se escribe una vez y se llama cuando se necesita.
- ✓ **Facilitan el mantenimiento**: si hay un error, se corrige en un solo lugar.

- ✓ **Mejoran la legibilidad**: el programa se entiende como una secuencia de tareas bien definidas.
- ✓ **Permiten la colaboración**: cada subrutina puede ser desarrollada por diferentes personas.

Ejemplos.

Ejemplo 1: Subrutina para calcular el área de un triángulo (Este ejemplo muestra cómo encapsular una fórmula matemática en una función que puede ser reutilizada en distintos contextos. La claridad del código mejora al separar la lógica de cálculo del flujo principal del programa).

***Ejemplo en Python*:**

Código ejecutable

```
def area_triangulo(base, altura):
    return (base * altura) / 2
    # Uso
    print(area_triangulo(10, 5))  # Resultado: 25.0
```

Ejemplo 2: Subrutina para verificar si un número es par (Este tipo de subrutina permite simplificar decisiones lógicas dentro del programa. Al encapsular la verificación en una función, se mejora la legibilidad y se facilita su reutilización en otros contextos).

***Ejemplo en Python*:**

Código ejecutable

```
def es_par(numero):
    return numero % 2 == 0
    # Uso
    if es_par(8):
        print("Es par")
    else:
        print("Es impar")
```

5.2 Bloques de secuencia, selección e iteración

¿Qué son los bloques?

En programación estructurada, los bloques son unidades lógicas que agrupan instrucciones relacionadas bajo una misma estructura de control. Estos bloques definen cómo se ejecuta el código: de forma secuencial, condicional o repetitiva. El uso de bloques permite construir algoritmos claros, predecibles y fáciles de mantener, evitando el uso de instrucciones desordenadas como goto. (Gaddis, 2023).

Tipos de bloques

- ✓ Existen tres tipos fundamentales de bloques en programación estructurada:
- ✓ Secuencia: instrucciones que se ejecutan una tras otra.
- ✓ Selección: decisiones que se toman según condiciones (if, else, switch).
- ✓ Iteración: repeticiones controladas mediante bucles (for, while, do-while).

Estos bloques permiten que el programa tenga un flujo lógico definido, facilitando su comprensión y depuración.

Ejemplos

Ejemplo 1: Bloque de secuencia (Este bloque ejecuta las instrucciones en orden, sin condiciones ni repeticiones. Es útil para tareas simples y lineales).

***Ejemplo en Python*:**

Código ejecutable

```
nombre = "Italo"
edad = 35
print("Nombre:", nombre)
print("Edad:", edad)
```

Ejemplo 2: Bloque de selección (Este bloque toma decisiones según una condición. Es esencial para validar entradas, controlar accesos o adaptar el comportamiento del programa).

Ejemplo en Python:

```
edad = int(input("Edad: "))
if edad >= 18:
   print("Mayor de edad")
else:
   print("Menor de edad")
```

5.3 Lenguajes de programación

¿Qué son los lenguajes de programación estructurados?

Los lenguajes de programación que permiten aplicar el paradigma de programación estructurada facilitan el desarrollo de programas mediante el uso de estructuras de control claramente definidas, como la secuencia, la selección y la repetición. Estas estructuras permiten organizar el flujo lógico de las instrucciones, favoreciendo la claridad, la legibilidad y el mantenimiento del código. Asimismo, la programación estructurada promueve la organización modular de los programas y reduce la necesidad de utilizar instrucciones de salto incondicional, como goto, que pueden dificultar la comprensión y el seguimiento de la ejecución del programa (Dijkstra, 1968; Gaddis, 2023).

Características clave

La programación estructurada presenta las siguientes características fundamentales:

- **Modularidad:** permite dividir un programa en módulos, subrutinas o funciones que facilitan la organización, reutilización y mantenimiento del código.
- **Control de flujo estructurado:** utiliza estructuras secuenciales, condicionales (if, if-else) y repetitivas (while, for) para controlar la ejecución de las instrucciones.
- **Organización de bloques de código:** agrupa instrucciones de manera lógica y jerárquica, facilitando la comprensión de las diferentes partes del programa.

- **Claridad y legibilidad:** promueve una organización coherente de las instrucciones que facilita la lectura, depuración y mantenimiento del código.
- **Reducción de saltos incondicionales:** favorece el uso de estructuras de control definidas en lugar de instrucciones como goto.

Estas características contribuyen a desarrollar programas más comprensibles, organizados y fáciles de mantener, especialmente cuando se trabaja con aplicaciones de mayor complejidad. La utilización de funciones, estructuras condicionales y ciclos repetitivos permite organizar el código en unidades lógicas, favoreciendo su comprensión y reutilización (Downey & Mayfield, 2020; Gaddis, 2023).

Ejemplos de lenguajes estructurados

A continuación, se presentan algunos lenguajes representativos del paradigma estructurado:

Tabla 1

Lenguajes de Programación Para la Programación Estructurada

Lenguaje	*Características principales*	*Aplicaciones comunes*
Pascal	Sintaxis clara, ideal para enseñanza	Educación, algoritmos básicos
C	Control de bajo nivel, eficiente	Sistemas operativos, software embebido
Ada	Seguridad y robustez	Aeronáutica, defensa
Python	Sintaxis legible, multiparadigma	Educación, ciencia de datos, automatización
Java	Orientado a objetos, portable, robusto	Aplicaciones empresariales, móviles (Android), sistemas distribuidos

ACTIVIDADES DE APRENDIZAJE

Actividad de aplicación: describa, diseñe y pruebe un subprograma pequeño en Java o Python con entradas y salidas definidas.

Ejemplo en Java: FUNCIÓN PARA EL ÁREA DE UN TRIÁNGULO

```
public class AreaTrianguloRevision {
    static double area(double base, double altura) {
        return base * altura / 2.0;
    }
    public static void main(String[] args) {
        System.out.println("Área: " + area(10.0, 5.0));
    }
}
```

Ejemplo en Python: FUNCIÓN PARA EL ÁREA DE UN TRIÁNGULO

```
def area_triangulo(base, altura):
    return base * altura / 2
print("Área:", area_triangulo(10, 5))
```

EVALUACIÓN

Programación estructurada y modular

Nombre:		**Fecha:**	
Curso / Asignatura:		**Puntaje:**	/ 10

Instrucciones: Lea cada pregunta con atención y encierre en un círculo o marque con una X la letra correspondiente a la respuesta correcta. Cada pregunta tiene un único valor y una sola opción válida.

1. ¿Cuál es el objetivo principal de la programación estructurada como paradigma de desarrollo?

a) Fomentar el uso de la instrucción goto para saltos directos en memoria.

b) Mejorar la claridad, legibilidad y mantenimiento del programa dividiéndolo en bloques y subrutinas.

c) Reducir la necesidad de documentar el código de los algoritmos.

d) Permitir únicamente la ejecución de programas en ensamblador de bajo nivel.

2. ¿Qué son las subrutinas y qué otro nombre recibe según el lenguaje de programación?

a) Son variables globales que almacenan texto; también llamadas cadenas.

b) Son bloques de código que encapsulan tareas específicas; también conocidas como funciones o procedimientos.

c) Son estructuras condicionales simples; también llamadas iteraciones.

d) Son compiladores especiales para traducir pseudocódigo.

3. ¿Cuál de los siguientes NO es un beneficio directo del uso de subrutinas?

a) Evitan la repetición innecesaria de código.

b) Facilitan el mantenimiento, ya que los errores se corrigen en un solo lugar.

c) Obligan a usar siempre la instrucción goto.

d) Permiten la colaboración al dividir el trabajo entre desarrolladores.

4. En programación estructurada, ¿cuáles son los tres tipos fundamentales de bloques?

a) Entrada, Proceso y Salida.

b) Secuencia, Selección e Iteración.

c) Variable, Constante y Tipo.

d) Lectura, Escritura y Asignación.

5. ¿Cómo se define un bloque de 'Secuencia'?

a) Un conjunto de instrucciones que toman decisiones según una condición lógica.

b) Un bloque que repite un ciclo hasta que una variable cambie de estado.

c) Instrucciones que se ejecutan una tras otra en orden lineal, sin condiciones ni repeticiones.

d) Una subrutina exclusiva del lenguaje Pascal.

6. Según el cuadro comparativo del documento, ¿qué lenguaje estructurado se caracteriza por su control de bajo nivel y alta eficiencia en sistemas operativos y software embebido?

a) Pascal

b) Python

c) C

d) Ada

7. De acuerdo con el documento, ¿para qué ámbito destaca principalmente el lenguaje Pascal?

a) Desarrollo de redes neuronales profundas.

b) Sintaxis clara, ideal para la enseñanza y educación.

c) Programación embebida en aeronáutica.

d) Creación de aplicaciones móviles para Android.

8. ¿En qué décadas surgió la programación estructurada como respuesta a la creciente complejidad del software?

a) Décadas de 1940 y 1950.

b) Décadas de 1960 y 1970.

c) Décadas de 1990 y 2000.

d) Décadas de 2010 y 2020.

9. Según Böhm y Jacopini (1966), ¿cuáles son las tres estructuras fundamentales de control en las que se basa la programación estructurada?

a) Variables, constantes y operadores.

b) Secuencia, selección y repetición.

c) Funciones, procedimientos y bibliotecas.

d) Compilación, interpretación y depuración.

10. En el ejemplo provisto en Python para calcular el área de un triángulo (def area_triangulo(base, altura): return (base * altura) / 2), ¿qué concepto de la unidad se ejemplifica?

a) Una subrutina que encapsula una fórmula matemática reutilizable.

b) Un bloque de iteración forzada.

c) Una instrucción goto indirecta.

d) Una matriz bidimensional de datos.

Glosario

Programación: Proceso de creación de instrucciones para que un ordenador realice tareas específicas, agrupadas en programas para automatizar procesos y resolver problemas de manera eficiente.

Programación Estructurada: Metodología clásica basada en el uso de estructuras lógicas de control: secuencias, decisiones e iteraciones, promoviendo la claridad, modularidad y mantenibilidad.

Dato: Valores o referentes conceptuales que recibe la computadora para ser procesados y construir soluciones algorítmicas.

String: Tipo de dato primitivo/clase que representa cadenas de caracteres (texto compuesto por letras, números y símbolos).

Integer: Tipo de dato numérico primitivo que representa números enteros (positivos o negativos sin parte decimal).

Float: Dato numérico primitivo de punto flotante en precisión simple, utilizado para ahorrar memoria en arreglos extensos.

Double: Tipo de dato numérico de punto flotante de doble precisión, utilizado en operaciones científicas y de alta exactitud.

Booleano (Boolean): Tipo de dato lógico fundamental que únicamente admite dos valores: TRUE (Verdadero) o FALSE (Falso).

Pseudocódigo: Herramienta de diseño de algoritmos que describe procedimientos lógicos en lenguaje cercano al natural sin preocuparse por la sintaxis estricta de un lenguaje de programación.

Lenguaje de Alto Nivel: Lenguaje independiente de la máquina y cercano al lenguaje natural humano que requiere compiladores o intérpretes para ejecutarse.

Algoritmo: Secuencia finita de instrucciones o pasos lógicos para resolver un problema o ejecutar una tarea específica.

Diagrama de Flujo: Representación gráfica de un algoritmo o proceso que utiliza símbolos estandarizados para mostrar las etapas de operación.

Finitud: Propiedad esencial de un algoritmo que indica que el proceso debe finalizar tras realizar un número definido de pasos.

Efectividad: Característica que establece que cada operación del algoritmo debe ser básica y realizable en un tiempo finito.

Estructura Algorítmica Secuencial: Estructura donde las instrucciones se ejecutan una tras otra de manera lineal y consecutiva, donde la salida de una acción es la entrada de la siguiente.

Estructura Selectiva Simple (If**):** Estructura condicional que evalúa una expresión lógica y ejecuta un proceso solo si la condición es verdadera.

Estructura Selectiva Doble (If-Else**):** Estructura que ofrece dos caminos de ejecución: uno para cuando la condición resulta verdadera y otro para cuando resulta falsa.

Estructura Selectiva Múltiple (Switch / If-Else If): Estructura condicional que evalúa múltiples alternativas o caminos posibles en función del valor de una expresión.

Estructura Cíclica While **(Mientras):** Bucle con evaluación previa de la condición lógica antes de cada iteración, ejecutando el bloque solo mientras esta sea verdadera.

Estructura Cíclica Do-While **(Repetir-Hasta):** Bucle con evaluación posterior de la condición al final de la iteración, garantizando la ejecución del bloque al menos una vez.

Estructura Cíclica For **(Para):** Bucle de control preciso utilizado idealmente cuando se conoce de antemano el número exacto de iteraciones a realizar.

Lista: Estructura de datos dinámica y flexible para almacenar colecciones ordenadas cuyo tamaño varía durante la ejecución del programa.

Vector (Arreglo Unidimensional): Estructura de datos que almacena elementos del mismo tipo en posiciones contiguas de memoria, accesibles mediante un único índice numérico.

Matriz (Arreglo Bidimensional): Estructura de datos organizada en filas y columnas, cuyos elementos individuales se acceden mediante dos índices (fila, columna).

Subrutina (Función / Procedimiento): Bloque de código independiente que encapsula una tarea específica para ser reutilizado dentro de un programa.

Bloque de Código: Conjunto de instrucciones agrupadas bajo una misma estructura de control (secuencia, selección o iteración) que define el flujo del programa.

Modularidad: Principio del desarrollo de software que consiste en descomponer un programa complejo en subprogramas o módulos independientes.

Referencias Bibliográficas

Aho, A. V., Hopcroft, J. E., & Ullman, J. D. (1983). *Estructuras de datos y algoritmos* (A. Vargas Villazón, J. Lozano Moreno, & G. Levine Gutiérrez, Trads.). Addison-Wesley Iberoamericana.

Böhm, C., & Jacopini, G. (1966). Flow diagrams, Turing machines and languages with only two formation rules. *Communications of the ACM, 9*(5), 366–371. https://doi.org/10.1145/355592.365646

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to algorithms* (3rd ed.). MIT Press.

Dennis, A., Wixom, B. H., & Roth, R. M. (2021). *Systems analysis and design* (8th ed.). Wiley.

Dijkstra, E. W. (1968). Go to statement considered harmful [Letter to the editor]. *Communications of the ACM, 11*(3), 147–148. https://doi.org/10.1145/362929.362947

Dijkstra, E. W. (1972). Notes on structured programming. En O.-J. Dahl, E. W. Dijkstra, & C. A. R. Hoare (Eds.), *Structured programming* (pp. 1–82). Academic Press. https://doi.org/10.5555/1243380.1243381

Downey, A. B., & Mayfield, C. (2020). *Think Java: How to think like a computer scientist* (2nd ed., Version 7.1.0). Green Tea Press. https://www.greenteapress.com/thinkjava7/html/index.html

Formación en Ambientes Virtuales de Aprendizaje. (s. f.). *Estructuras cíclicas*. *Estructuras cíclicas con LPP* [Objeto de aprendizaje]. https://progra1utc.files.wordpress.com/2015/10/estructurasciclicas.pdf

Franchitti, J.-C. (2024). *Introduction to computer science*. OpenStax. https://openstax.org/books/introduction-computer-science/pages/7-2-programming-language-constructs

Gaddis, T. (2023). *Starting out with programming logic and design* (6th ed.). Pearson.

González, M. (2020). *Fundamentos de programación con Python*. Alfaomega.

Instituto de Ingeniería de Software. (2021). *Guía práctica para el desarrollo de software*. Editorial Técnica.

Joint Task Force on Computer Science Curricula. (2024). *Computer science curricula 2023*. Association for Computing Machinery, IEEE Computer Society, & Association for the Advancement of Artificial Intelligence. https://csed.acm.org/wp-content/uploads/2025/11/CS2023-Report.htm

Joyanes Aguilar, L. (2008). *Fundamentos de programación* (4.ª ed.). McGraw-Hill Interamericana de España. https://books.google.com/books?id=nrNvPwAACAAJ

Knuth, D. E. (1997). *El arte de programar computadoras: Volumen 1. Algoritmos fundamentales* (3.ª ed.). Addison-Wesley.

Larman, C. (2003). *UML y patrones: Una introducción al análisis y diseño orientado a objetos y al proceso unificado* (B. Moros Valle, Trad.; 2.ª ed.). Pearson Educación.

Microsoft. (s. f.). *Floating-point numeric types (C# reference)*. Microsoft Learn. https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/double

Oracle. (s. f.). *Arrays*. The Java Tutorials. https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html

Oracle. (2023). *The Java language specification: Java SE 21 edition*. https://docs.oracle.com/javase/specs/jls/se21/html/jls-4.html

Oracle. (2025). *The Java language specification: Java SE 25 edition* (sección 14.11, "The switch statement"). https://docs.oracle.com/javase/specs/jls/se25/html/jls-14.html#jls-14.11

Python Software Foundation. (2025). *Data structures*. Python 3.14 documentation. https://docs.python.org/3.14/tutorial/datastructures.html

Rodríguez, M. (2024). *Símbolos del diagrama de flujo*. Tu Gimnasia Cerebral. https://tugimnasiacerebral.com/herramientas-de-estudio/diagrama-de-flujo-que-es-caracteristicas-y-como-hacerlo

Schildt, H. (2022). *Java: A beginner's guide* (9th ed.). McGraw Hill.

Sebesta, R. W. (2016). *Concepts of programming languages* (11th global ed.). Pearson Education.

Sedgewick, R., & Wayne, K. (2011). *Algorithms* (4th ed.). Addison-Wesley Professional.

Universidad Nacional de Rosario. (2020). *Estructura de datos: Vectores*. https://www.fceia.unr.edu.ar/estruc/2006/vector.htm

Universidad Nacional del Sur. (2020). *Tipos subindicados en Pascal*. http://www.lip.uns.edu.ar/pci/Tipos_Subinidicados_en_Pascal.PDF

UNLaM. (2020). *Vectores*. https://miel.unlam.edu.ar/data/contenido/3635/Vectores.pdf

Villalobos, R. M. (2014). *Fundamentos de programación con JAVA (100 algoritmos codificados)*. Editorial Macro.

Anexos

Solucionario de las evaluaciones

UNIDAD 1

Pregunta 1: ¿Cuáles son las estructuras lógicas fundamentales en las que se basa la programación estructurada?

Respuesta correcta: b) Secuencias, decisiones y repeticiones.
Explicación: La programación estructurada es la metodología clásica que se basa principalmente en tres estructuras lógicas: secuencias, decisiones y repeticiones, promoviendo claridad y modularidad en el desarrollo.

Pregunta 2: ¿Qué tipo de lenguaje de programación se caracteriza por utilizar símbolos nemotécnicos y requerir traducción al lenguaje máquina?

Respuesta correcta: c) Lenguaje de bajo nivel (Ensamblador).
Explicación: El lenguaje de bajo nivel (Ensamblador) emplea símbolos nemotécnicos para facilitar la programación, pero necesita ser traducido para que la máquina pueda ejecutarlo.

Pregunta 3: ¿Cuál es una ventaja principal de los lenguajes de programación de alto nivel?

Respuesta correcta: b) Es cercano al lenguaje natural, tiene menor tiempo de desarrollo y es independiente de la máquina.
Explicación: Los lenguajes de alto nivel (como Java, C o Python) son cercanos al lenguaje humano, acortan los tiempos de desarrollo y no dependen de la arquitectura específica de una máquina.

Pregunta 4: ¿El tipo de dato utilizado para almacenar información textual como nombres o direcciones se denomina?

Respuesta correcta: c) String
Explicación: El tipo de dato String representa cadenas de caracteres (texto escrito que contiene letras, números y símbolos) ideal para registrar nombres, direcciones y descripciones.

Pregunta 5: ¿Qué diferencia principal existe entre el tipo de dato Float y el tipo Double?

Respuesta correcta: b) Float representa precisión simple y ahorra memoria, mientras que Double representa precisión doble para mayor exactitud.
Explicación: Float es un dato de precisión simple que ahorra memoria en grandes matrices, mientras que Double ofrece doble precisión para cálculos numéricos o científicos que exigen mayor grado de exactitud.

Pregunta 6: ¿Cuáles son los dos únicos valores que puede tomar un tipo de dato Booleano?

Respuesta correcta: c) TRUE o FALSE.
Explicación: Un dato booleano representa exclusivamente dos valores: TRUE (Verdadero) o FALSE (Falso), siendo fundamentales para evaluar condiciones y controlar el flujo de un algoritmo.

Pregunta 7: ¿Por qué los tipos de datos enteros (Integer) tienen un rango limitado de valores?

Respuesta correcta: b) Debido al tamaño fijo de bits asignados según el lenguaje y la arquitectura del sistema.
Explicación: Los datos enteros están delimitados en rango por la cantidad de bits asignados en la memoria, dependiendo del lenguaje de programación y de la arquitectura de la máquina.

Pregunta 8: ¿Cuál es la función principal del pseudocódigo en el diseño de programas?

Respuesta correcta: c) Describir procedimientos lógicos de forma clara sin preocuparse por la sintaxis estricta de un lenguaje.
Explicación: El pseudocódigo permite representar la lógica de la solución antes de implementarla.

Pregunta 9: Según el texto, ¿en qué fase del desarrollo es especialmente útil el pseudocódigo?

Respuesta correcta: b) En las etapas iniciales de desarrollo para visualizar la solución y evaluar su viabilidad.
Explicación: El pseudocódigo es crucial en las etapas iniciales, ya que facilita visualizar el flujo de la solución propuesta, identificar errores tempranos y abstraer la lógica antes de programar.

Pregunta 10: En el contexto de la programación, ¿qué son los 'datos'?

Respuesta correcta: c) Valores o referentes que un computador recibe para ser procesados y construir soluciones o algoritmos.
Explicación: Los datos son valores o referentes numéricos, textuales o lógicos que ingresan al computador y son procesados por el programador como materia prima para generar información y algoritmos.

UNIDAD 2

Pregunta 1: ¿Cuál de las siguientes opciones define correctamente la característica de 'finitud' en un algoritmo?

Respuesta correcta: B) El algoritmo debe tener un número definido de pasos y terminar en un tiempo determinado.
Explicación: La finitud garantiza que el proceso no se ejecute de manera indefinida, finalizando tras realizar un número definido de pasos.

Pregunta 2: ¿Qué símbolo de diagrama de flujo se utiliza para representar la toma de decisiones basada en una comparación lógica?

Respuesta correcta: C) Rombo
Explicación: El rombo indica la comparación de dos datos y, según el resultado lógico (verdadero o falso), determina el camino a seguir en el diagrama.

Pregunta 3: ¿Cuál es la función del símbolo del paralelogramo (romboide) en la simbología estándar de diagramas de flujo?

Respuesta correcta: B) Representar la entrada o salida de datos a través de cualquier dispositivo.
Explicación: El paralelogramo representa la entrada/salida general de datos en el sistema (por ejemplo, escáner, teclado o lectura externa).

Pregunta 4: ¿Qué característica define a las estructuras algorítmicas secuenciales?

Respuesta correcta: C) Cada acción sigue a la anterior en un orden lineal sin desviaciones ni repeticiones.

Explicación: En la estructura secuencial, las instrucciones se ejecutan consecutivamente en un orden predeterminado (de arriba hacia abajo), donde la salida de un paso es la entrada del siguiente.

Pregunta 5: Según el documento, ¿cuál es uno de los principales beneficios de utilizar diagramas de flujo en la gestión de procesos?

Respuesta correcta: B) Permitir identificar cuellos de botella y áreas ineficientes para optimizar procesos.

Explicación: Al ofrecer una representación visual del proceso, los diagramas de flujo facilitan detectar ineficiencias, errores y cuellos de botella para su continua mejora.

Pregunta 6: Un algoritmo exige que sus operaciones sean básicas y realizables en la práctica. ¿A qué característica corresponde este enunciado?

Respuesta correcta: B) Efectividad

Explicación: La efectividad establece que cada operación del algoritmo debe ser realizable en un tiempo finito mediante operaciones básicas.

Pregunta 7: ¿Qué símbolo se utiliza específicamente para conectar dos partes de un diagrama de flujo que se encuentran en páginas diferentes?

Respuesta correcta: B) Conector Externo (Pentágono / Figura apuntando abajo)

Explicación: El conector externo permite dar continuidad al flujo de un diagrama cuando este se extiende a páginas distintas.

Pregunta 8: ¿Cuáles son las tres grandes categorías en las que se clasifican los algoritmos según el texto?

Respuesta correcta: B) Secuenciales, Selectivos y Cíclicos

Explicación: Dependiendo de su estructura y funcionamiento, la guía clasifica los algoritmos en tres grandes grupos: secuenciales, selectivos y cíclicos.

Pregunta 9: En el ejemplo del cálculo del área de un círculo, ¿qué elementos representan la 'entrada' y la 'salida' respectivamente?

Respuesta correcta: B) Entrada: el radio (r); Salida: el área calculada (a). *Explicación:* El valor que debe proveer el usuario para realizar el cálculo es el radio (entrada), y el resultado que el programa entrega al finalizar es el área (salida).

Pregunta 10: ¿Por qué las estructuras secuenciales son ideales para procesos deterministas y para la enseñanza inicial de la programación?

Respuesta correcta: C) Porque tienen un único punto de entrada y salida, facilitando su lectura, mantenimiento y depuración.

Explicación: Su flujo unidireccional y sin bifurcaciones hace que su comprensión sea lógica y sencilla, con un único punto de inicio y un único punto de cierre.

UNIDAD 3

Pregunta 1: ¿Cuál es la función principal de las estructuras de control selectivas o condicionales en programación?

Respuesta correcta: b) Dirigir el flujo de ejecución hacia diferentes rutas según el resultado de una condición lógica.

Explicación: Las estructuras selectivas evalúan expresiones lógicas (verdadero o falso) para determinar qué bloque de acciones debe ejecutarse.

Pregunta 2: ¿Qué caracteriza fundamentalmente a una estructura selectiva simple (como la instrucción if)?

Respuesta correcta: b) Evalúa procesos solo por el camino del "Sí" (verdadero); si es falso, va directo al fin de la estructura.

Explicación: En las estructuras selectivas simples, el bloque de código únicamente se procesa si la condición resulta verdadera. Si es falsa, continúa hacia el final sin realizar otra acción.

Pregunta 3: ¿Cuál es la diferencia principal entre una estructura selectiva doble y una estructura selectiva simple?

Respuesta correcta: a) La estructura doble ofrece dos caminos posibles (uno si la condición es verdadera y otro si es falsa).

Explicación: La estructura doble (if-else) bifurca el flujo: si la condición se cumple realiza un proceso, y si no se cumple (camino del No) realiza otro.

Pregunta 4: ¿Qué estructura selectiva es ideal para evaluar más de dos caminos o condiciones, como las opciones de un menú o los días de la semana?

Respuesta correcta: c) Estructura Selectiva Múltiple (ej. switch o if-else if).

Explicación: Las estructuras múltiples evalúan más de dos condiciones y permiten derivar el programa en varios bloques de código estructurados.

Pregunta 5: ¿Cuál es la característica principal de la estructura cíclica While (Mientras)?

Respuesta correcta: b) Evalúa la condición antes de cada iteración, ejecutando el bloque solo si es verdadera.

Explicación: El bucle While hace una evaluación previa. Si la condición inicial es falsa desde el principio, el bloque interno no llega a ejecutarse ninguna vez.

Pregunta 6: ¿Qué diferencia fundamental existe entre la estructura cíclica Do-While y la estructura While?

Respuesta correcta: a) Do-While garantiza que el bloque de código se ejecute al menos una vez, ya que la condición se evalúa al final.

Explicación: En la estructura Do-While, la condición lógica se comprueba al final de la primera iteración, asegurando al menos una ejecución inicial.

Pregunta 7: ¿En qué situación es más recomendable utilizar la estructura cíclica For (Para)?

Respuesta correcta: c) Cuando se conoce de antemano el número exacto de iteraciones que se deben realizar.

Explicación: La estructura For es ideal cuando se cuenta con un número prefijado o determinado de iteraciones.

Pregunta 8: Si un estudiante obtiene las notas 8, 7 y 9, y el algoritmo evalúa promedio >= 7 para determinar si aprueba, ¿cuál es el resultado del promedio y de la condición?

Respuesta correcta: b) Promedio = 8; Condición Verdadera (Imprime 'El estudiante aprueba').

Explicación: $(8 + 7 + 9) / 3 = 8$. Como $8 \ge 7$ es verdadero, la condición se cumple y se imprime el mensaje correspondiente.

Pregunta 9: En el contexto de estructuras condicionales, ¿qué valores evalúa o devuelve una expresión lógica?

Respuesta correcta: b) Verdadero (True) o Falso (False).

Explicación: Las decisiones en algoritmos se basan en expresiones lógicas que devuelven exclusivamente resultados booleanos: Verdadero o Falso.

Pregunta 10: ¿Qué estructura permite repetir un menú hasta que el usuario seleccione la opción 0 para salir?

Respuesta correcta: b) Estructura Cíclica (While o Do-While).

Explicación: el ciclo continúa mientras la opción no sea 0. La condición de salida queda visible y el ejemplo no incorpora contraseñas fijas.

UNIDAD 4

Pregunta 1: ¿Cuál es la característica principal que diferencia a las listas de los arreglos o vectores tradicionales?

Respuesta correcta: B) Permiten una gestión dinámica de la memoria, pudiendo crecer o reducirse

durante la ejecución.

Explicación: A diferencia de los arreglos tradicionales de tamaño fijo, las listas gestionan la memoria de forma dinámica, lo que permite modificar su tamaño (añadir o eliminar elementos) mientras se ejecuta el programa.

Pregunta 2: En el contexto de las listas enlazadas, ¿cuál es una ventaja clave mencionada en el documento?

Respuesta correcta: B) Permiten una inserción eficiente de elementos sin necesidad de desplazar los demás.

Explicación: Las listas enlazadas facilitan la inserción de elementos de forma eficiente al ajustar los punteros o enlaces, sin tener que mover físicamente el resto de los elementos en la memoria.

Pregunta 3: En el lenguaje Python, ¿qué método se utiliza para agregar un nuevo elemento al final de una lista?

Respuesta correcta: C) append()

Explicación: Tal como se muestra en los ejemplos de código del documento (`frutas.append("mango")`), en Python se utiliza el método `append()` para anexar un nuevo elemento al final de la lista.

Pregunta 4: ¿Cómo se define un vector o arreglo unidimensional?

Respuesta correcta: C) Una estructura que almacena elementos del mismo tipo en posiciones contiguas de memoria.

Explicación: Un vector es un arreglo unidimensional que guarda datos homogéneos (del mismo tipo) contiguamente en memoria, accediendo a cada elemento mediante un único índice.

Pregunta 5: Respecto a los vectores, ¿en qué lenguaje el tamaño del vector se define típicamente en tiempo de compilación?

Respuesta correcta: B) C

Explicación: El documento señala explícitamente que en lenguajes como C el tamaño del vector se define en tiempo de compilación (fijo), a diferencia de lenguajes dinámicos o de más alto nivel.

Pregunta 6: ¿Qué mecanismo se utiliza para acceder a un elemento específico dentro de un vector?

Respuesta correcta: B) Un índice numérico.

Explicación: Cada elemento de un vector unidimensional está posicionado en la secuencia contigua y se accede directamente a él mediante su índice numerado (usualmente desde 0).

Pregunta 7: ¿Qué es una matriz en el ámbito de las estructuras de datos?

Respuesta correcta: C) Un arreglo bidimensional que organiza los datos en forma de filas y columnas.

Explicación: Las matrices se definen como arreglos bidimensionales estructurados de manera tabular mediante filas y columnas.

Pregunta 8: ¿Cuántos índices se requieren obligatoriamente para acceder a un elemento individual dentro de una matriz?

Respuesta correcta: B) Dos índices (uno para la fila y otro para la columna).

Explicación: Al ser una estructura bidimensional, para ubicar una celda específica en una matriz es indispensable proporcionar dos índices: el índice de la fila y el de la columna.

Pregunta 9: ¿En qué área avanzada de la informática se utilizan ampliamente las matrices para representar redes neuronales y operaciones de convolución?

Respuesta correcta: a) El primero identifica la fila y el segundo la columna.

Explicación: el primer índice selecciona la fila y el segundo la columna de la matriz.

Pregunta 10: En los diagramas de código presentados para recorrer una matriz (en Java o Python), ¿qué tipo de estructura de control se utiliza comúnmente?

Respuesta correcta: B) Bucles (FOR) anidados.

Explicación: Tanto en los ejemplos de Java como de Python provistos en el documento, se emplean dos bucles anidados (un ciclo externo para recorrer filas y uno interno para recorrer columnas).

UNIDAD 5

Pregunta 1: ¿Cuál es el objetivo principal de la programación estructurada como paradigma de desarrollo?

Respuesta correcta: B) Mejorar la claridad, legibilidad y mantenimiento del programa dividiéndolo en bloques y subrutinas.

Explicación: El texto explicita que la programación estructurada busca mejorar la claridad, legibilidad y mantenimiento del código organizándolo en bloques funcionales y subrutinas, evitando instrucciones desordenadas.

Pregunta 2: ¿Qué son las subrutinas y qué otro nombre recibe según el lenguaje de programación?

Respuesta correcta: B) Son bloques de código que encapsulan tareas específicas; también conocidas como funciones o procedimientos.

Explicación: Las subrutinas son bloques que encapsulan tareas específicas para modularizar el código y son comúnmente denominadas funciones o procedimientos.

Pregunta 3: ¿Cuál de los siguientes NO es un beneficio directo del uso de subrutinas?

Respuesta correcta: C) Obligan a usar siempre la instrucción goto.

Explicación: La programación estructurada y las subrutinas evitan precisamente el uso de instrucciones desordenadas como goto. Los otros tres puntos sí son ventajas explícitas del uso de subrutinas.

Pregunta 4: En programación estructurada, ¿cuáles son los tres tipos fundamentales de bloques?

Respuesta correcta: B) Secuencia, Selección e Iteración.

Explicación: El documento especifica claramente los tres tipos de bloques estructurales: Secuencia (instrucciones consecutivas), Selección (decisiones if/switch) e Iteración (bucles for/while).

Pregunta 5: ¿Cómo se define un bloque de 'Secuencia'?

Respuesta correcta: C) Instrucciones que se ejecutan una tras otra en orden lineal, sin condiciones ni repeticiones.

*Explicación***:** El bloque de secuencia ejecuta sus instrucciones ordenadamente en línea recta, una detrás de otra, sin requerir tomar decisiones condicionales ni iterar.

Pregunta 6: Según el cuadro comparativo del documento, ¿qué lenguaje estructurado se caracteriza por su control de bajo nivel y alta eficiencia en sistemas operativos y software embebido?

Respuesta correcta: C) C

*Explicación***:** En la Tabla 1 de Lenguajes de Programación Estructurada se detalla que el lenguaje C ofrece control de bajo nivel y alta eficiencia, siendo aplicado comúnmente en sistemas operativos y software embebido.

Pregunta 7: De acuerdo con el documento, ¿para qué ámbito destaca principalmente el lenguaje Pascal?

Respuesta correcta: B) Sintaxis clara, ideal para la enseñanza y educación.

*Explicación***:** La tabla resalta que Pascal se caracteriza por poseer una sintaxis sumamente clara, lo que lo convierte en un lenguaje ideal para la enseñanza y la educación inicial de algoritmos.

Pregunta 8: ¿En qué décadas surgió la programación estructurada como respuesta a la creciente complejidad del software?

Respuesta correcta: B) Décadas de 1960 y 1970.

*Explicación***:** La sección 5.4 señala expresamente que la programación estructurada surgió en las décadas de 1960 y 1970 para combatir la complejidad de los programas informáticos.

Pregunta 9: Según Böhm y Jacopini (1966), ¿cuáles son las tres estructuras fundamentales de control en las que se basa la programación estructurada?

Respuesta correcta: B) Secuencia, selección y repetición.

Explicación: La programación estructurada se fundamenta en tres estructuras básicas de control: secuencia, selección y repetición. Estas permiten organizar el flujo de ejecución de un programa de manera lógica, ordenada y predecible, facilitando la comprensión, el mantenimiento y la verificación del código.

Pregunta 10: En el ejemplo provisto en Python para calcular el área de un triángulo (`def area_triangulo(base, altura): return (base * altura) / 2`), ¿qué concepto de la unidad se ejemplifica?

Respuesta correcta: A) Una subrutina que encapsula una fórmula matemática reutilizable.

*Explicación***:** El documento explica que este código demuestra cómo encapsular una fórmula matemática dentro de una función/subrutina que puede ser reutilizada en distintos contextos del programa.

Esta obra se publicó en primera edición en el 2026.

DOI: 10.64325/9kndwa97
ISBN: 978-9907-837-08-7



Esta obra se edita bajo una Licencia Creative Commons BY.4.0