INSTITUTO SUPERIOR TECNOLÓGICO TENA
Tecnología, Innovación y Desarrollo

DESARROLLO DE SOFTWARE

Instrumento para facilitar el proceso de enseñanza-aprendizaje de la asignatura

GUÍA GENERAL DE ESTUDIO DE LA ASIGNATURA

20250023

FUNDAMENTOS DE PROGRAMACIÓN

Período académico

Primero

Octubre - 2025

ING. ITALO MARCELO LARA PILCO, MSC

INSTITUTO SUPERIOR TECNOLÓGICO TENA
Tecnología, Innovación y Desarrollo

¡Tecnología Innovación y Desarrollo!

GUIA GENERAL DE ESTUDIO DE LA ASIGNATURA – FUNDAMENTOS DE PROGRAMACIÓN

INSTITUTO SUPERIOR TECNOLÓGICO TENA

Carrera de Tecnología en Desarrollo de Software

ISTT DS Primera Edición – Tena, octubre 2025

SIN ISBN

Instituto Superior Tecnológico Tena

Km. 1 1/2 Vía Tena - Archidona

Tena, Ecuador

Este texto ha sido sometido a un proceso de evaluación por pares internos. El contenido se puede citar y reproducir, siempre que se reconozca los créditos correspondientes, refiriendo.

AUTOR - REDACCIÓN Y FORMULACIÓN DE CONTENIDOS

Ing. Italo Marcelo Lara Pilco., MsC

Profesor del Instituto Superior Tecnológico Tena

REVISIÓN DE PARES

Mg. Alvaro Santiago Toalombo Díaz
Mg. Henry Fabian Chango Chango
Mg. Martha Janina Duarte Mora
Mg. Danilo Alexander Zamora Núñez
Lcda. María Angélica Campoverde Encalada

Comisión de revisión técnica de guías de estudio del Instituto Superior Tecnológico Tena

APROBACIÓN

Mg. Danilo Alexander Zamora Núñez

Coordinador de Investigación, Desarrollo Tecnológico e Innovación

Impreso y hecho en Ecuador.

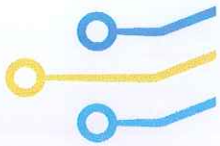
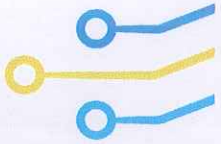


TABLA DE CONTENIDO

Contenido



GUIA GENERAL DE ESTUDIO DE LA ASIGNATURA

DATOS GENERALES DE LA ASIGNATURA							
Carrera	Tecnología en Desarrollo de Software			**Nombre asignatura**	Fundamentos de Programación		
Modalidad	Presencial			**Campo de Formación**	NA		
Jornada	Matutina/Nocturna			**Unidad de Organización Curricular**	Formación Profesional		
Período académico	Primero			**Código de la asignatura**	DSW - 103		
Distribución de horas en las actividades de aprendizaje				**N° Total de horas de la asignatura**	192		
N° de horas Docencia	**64**	**N° de horas Aprendizaje Práctico Experimental**				**N° de horas Autónomo**	**40**
		En contacto con docente	**48**	**Autónomo**	**40**		

PRERREQUISITOS Y CORREQUISITOS			
Prerrequisitos de la asignatura		**Correquisitos de la asignatura**	
Asignatura	**Código**	**Asignatura**	**Código**

DESCRIPCIÓN DE LA ASIGNATURA

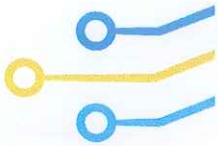
Se estudiarán conceptos básicos de programación, datos, operadores, estructuras de control entre otras, el manejo de algoritmos para entender y solucionar problemas cotidianos, se desarrollarán técnicas para adaptar los procesos estudiados en la asignatura a cualquier lenguaje de programación.

OBJETIVO GENERAL

Aplicar de forma autónoma la lógica de programación para el desarrollo de algoritmos sencillos utilizando una sintaxis adecuada.

CONTRIBUCIÓN DE LOS RESULTADOS DE APRENDIZAJE DE LA ASIGNATURA AL PERFIL DE EGRESO DE LA CARRERA

Resultados de aprendizaje de la asignatura	Resultados de aprendizaje del perfil de egreso de la carrera	Contribución (alta – media – baja)



Desarrolla aplicaciones de software, desde el análisis del problema y la planificación del proyecto, hasta la implementación, el mantenimiento, la prueba y la documentación. Desarrolla habilidades lógicas y de programación que permitan resolver problemas, tomar decisiones y ejercitar su buen juicio en situaciones organizacionales complejas.	Aplica técnicas de investigación en la búsqueda de nuevas formas de aplicación del desarrollo de software en los sectores industriales. Utiliza herramientas y tecnologías de programación para llevar a cabo tareas específicas en el campo de Desarrollo de Software. Aplica conceptos, técnicas, herramientas de programación, que contribuyan con la implementación de soluciones de software. Aplica habilidades de Tics, trabajo en equipo, gestión de proyectos, liderazgo y creatividad, para trabajar en ambientes colaborativos con profesionalismo y responsabilidad social.	Alta Media Media Alta

CONTENIDOS DE LA ASIGNATURA

Unidad 1 METODOLOGÍA LÓGICA PARA RESOLVER PROBLEMAS COMPUTACIONALES.

- 1.1 Introducción a la programación.
- 1.2 Programación por computadora.
- 1.3 Datos
  - 1.3.1 Datos de tipo String
  - 1.3.2 Datos de tipo Integer
  - 1.3.3 Datos de tipo Float
  - 1.3.4 Datos de tipo Double
  - 1.3.5 Datos de tipo Booleano
- 1.4 Pseudocódigo

Unidad 2 ESTRUCTURAS BÁSICAS Y TÉCNICAS PARA REPRESENTAR ALGORITMOS

- 2.1 Algoritmos
  - 2.1.1 Introducción, características y clasificación
- 2.2 Diagramas de flujo
  - 2.2.1 Introducción a los diagramas de flujo
  - 2.2.2 Características
  - 2.2.3 Simbología
- 2.3 Estructuras Algorítmicas secuenciales
  - 2.3.1 Introducción a las estructuras algorítmicas secuenciales
  - 2.3.2 Características
  - 2.3.3 ejemplos

Unidad 3 LENGUAJE ALGORÍTMICO Y ESTRUCTURAS DE CONTROL

- 3.1 Estructuras Algorítmicas selectivas
  - 3.1.1 Introducción a las estructuras Algorítmicas selectivas.
  - 3.1.2 Características
  - 3.1.3 Clasificación
    - 3.1.3.1 Estructuras Algorítmicas selectivas simples
      - 3.1.3.1.1 Introducción
      - 3.1.3.1.2 Características
      - 3.1.3.1.3 Ejercicios

3.1.3.2 Estructuras Algorítmicas selectivas dobles
3.1.3.2.1 Introducción
3.1.3.2.2 Características
3.1.3.2.3 Ejercicios
3.1.3.3 Estructuras Algorítmicas selectivas múltiples
3.1.3.3.1 Introducción
3.1.3.3.2 Características
3.1.3.3.3Ejercicios
3.2 Estructuras Algorítmicas cíclicas
3.2.1 Introducción a las estructuras Algorítmicas cíclicas
3.2.2 Características
3.2.3 Clasificación
3.2.3.1 Estructuras Algorítmicas cíclicas While
3.2.3.1.1 Introducción
3.2.3.1.2 Características
3.2.3.1.3 Ejercicios
3.2.3.2 Estructuras Algorítmicas cíclicas Do-While
3.2.3.2.1 Introducción
3.2.3.2.2 Características
3.2.3.2.3 Ejercicios
3.2.3.3 Estructuras Algorítmicas cíclicas For
3.2.3.3.1 Introducción
3.2.3.3.2 Características
3.2.3.3.3Ejercicios

Unidad 4: SUBPROGRAMAS Y ARREGLOS.

4.1 Listas
4.1.1 Introducción
4.1.2 Características
4.1.3 Ejercicios
4.2 Vectores
4.2.1 Introducción
4.2.2 Características
4.2.3 Ejercicios
4.3 Matrices
4.3.1 Introducción
4.3.2 Características
4.3.3 Ejemplos

Unidad 5: PROGRAMACIÓN ESTRUCTURADA.

5.1 Subrutinas
5.2 Bloques
5.3 Lenguajes de programación
5.4 Programación estructurada

ESTRATEGIAS METODOLÓGICAS Y RECURSOS DIDÁCTICOS

ESTRATEGIAS METODOLÓGICAS	HABILIDADES BLANDAS	FINALIDAD
Activas para la enseñanza y aprendizaje	**Valores vinculados a la autonomía del sujeto**: confianza, crítica y autocrítica, honestidad, integridad	• Generar confianza/ Promover el pensamiento crítico. • Permite a los estudiantes cumplir un rol activo dentro de su formación.

		• Construye una sociedad participante.
Aprendizaje y trabajo cooperativo	**Valores elementales de convivencia y civilidad:** crítica y autocrítica, tolerancia, empatía, respeto, justicia, lealtad, paciencia	• Promover un ambiente de colaboración/ trabajo en equipo/ Saber escuchar/Promover el pensamiento crítico/ fomentar el liderazgo/ adaptabilidad. • Mantener una comunicación abierta con el equipo/ tolerancia a los errores, aceptar y aprender de las críticas. • Fomentar el sentido de pertenencia
Aprendizaje individual	**Valores vinculados a la autonomía del sujeto**: responsabilidad, honestidad, integridad, efectividad, autonomía	• Facilitar la asimilación del contenido por parte del estudiante/ Plantear preguntas para promover la comunicación efectiva /Promover el pensamiento crítico • Lectura comprensiva para fijar contenidos/ Promover el pensamiento crítico

RECURSOS DIDÁCTICOS	
MATERIALES CONVENCIONALES	*Material impreso: libros, folletos, fotocopias, periódicos, etc.*
	Tableros didácticos: pizarra
MATERIALES AUDIOVISUALES	*Imágenes fijas proyectables (fotos): diapositivas y fotografías.*
	Materiales audiovisuales (vídeo): películas y vídeos
NUEVAS TECNOLOGÍAS	*Programas informáticos: procesador de palabras, hojas de cálculo, presentaciones*
	Servicios telemáticos: páginas web, plataforma EVA, correo electrónico, chats (WhatsApp), Google Drive, Genially,Pseint,Netbeans

BIBLIOGRAFÍA

Bibliografía Básica de la Asignatura:	**Físico**	**Digital**
Vega, J. (2022). Java 17 Fundamentos de Programación (segunda edición). Edu. Colombia. ISBN: 978-958-792-410-7. Número de inventario en biblioteca: ISTT-DS-0239	X	
Villalobos, R. (2014). *Java 17 Fundamentos de Programación* (segunda edición). MACRO. Peru. ISBN: 978-612-304-235-6. Número de inventario en biblioteca: ISTT-DS-0018	X	

Bibliografía de consulta de la Asignatura:	**Físico**	**Digital**
Joyanes, L. (2008). *Fundamentos de Programación, Algoritmos, Estructura de Datos y Objetos* (cuarta edición). McGRAW-HILL. España. ISBN: 978-84-481-6111-8. url: https://combomix.net/wp-content/uploads/2017/03/Fundamentos-de-programaci%C3%B3n-4ta-Edici%C3%B3n-Luis-Joyanes-Aguilar-2.pdf.		X

DESCRIPTIVA DE LAS COMPETENCIAS DE LA GUÍA DE SISTEMAS DE ORGANIZACIÓN EMPRESARIAL

La guía de Fundamentos de Programación está diseñada para desarrollar competencias específicas en los estudiantes, enfocándose en los conceptos clave de la programación estructurada, la misma que se basa en las estructuras de control, a continuación, se describen las competencias de cada unidad:

COMPETENCIAS ESPECÍFICAS:

Unidad 1: METODOLOGÍA LÓGICA PARA RESOLVER PROBLEMAS COMPUTACIONALES.

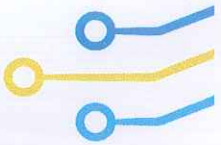
- Aplica metodologías y técnicas de investigación en la búsqueda, fundamentación y elaboración de soluciones informáticas.
- Maneja métodos y técnicas de comunicación e interacción social aplicados a la profesión.
- Comunica de manera asertiva, coopera activamente en la solución de conflictos con sus pares profesionsales.

Unidad 2: ESTRUCTURAS BÁSICAS Y TÉCNICAS PARA REPRESENTAR ALGORITMOS

- Aplica metodologías y técnicas de investigación en la búsqueda, fundamentación y elaboración de soluciones informáticas.
- Identifica oportunidades para mejorar el desempeño de las organizaciones a través del uso eficiente y eficaz de soluciones informáticas.
- Elabora documentación técnica

Unidad 3: LENGUAJE ALGORÍTMICO Y ESTRUCTURAS DE CONTROL

- Aplica metodologías y técnicas de investigación en la búsqueda, fundamentación y elaboración de soluciones informáticas.
- Desarrolla aplicaciones utilizando plataformas actuales de programación



Unidad 4: SUBPROGRAMAS Y ARREGLOS.

- Aplica metodologías y técnicas de investigación en la búsqueda, fundamentación y elaboración de soluciones informáticas.
- Realiza el desarrollo de aplicaciones utilizando plataformas actuales de programación.

Unidad 5: PROGRAMACIÓN ESTRUCTURADA

- Ejecuta proyectos de investigación aplicados a la profesión.
- Realiza el desarrollo de aplicaciones utilizando plataformas actuales de programación.
- Realiza pruebas que garanticen la calidad de software.
- Aplicar estándares de calidad en el desarrollo y evaluación de soluciones informáticas.

UNIDAD 1. METODOLOGÍA LÓGICA PARA RESOLVER PROBLEMAS COMPUTACIONALES.

1.1 Introducción a la programación.

1.2 Programación por computadora.

1.3 Datos

1.3.1 Datos de tipo String

1.3.2 Datos de tipo Integer

1.3.3 Datos de tipo Float

1.3.4 Datos de tipo Double

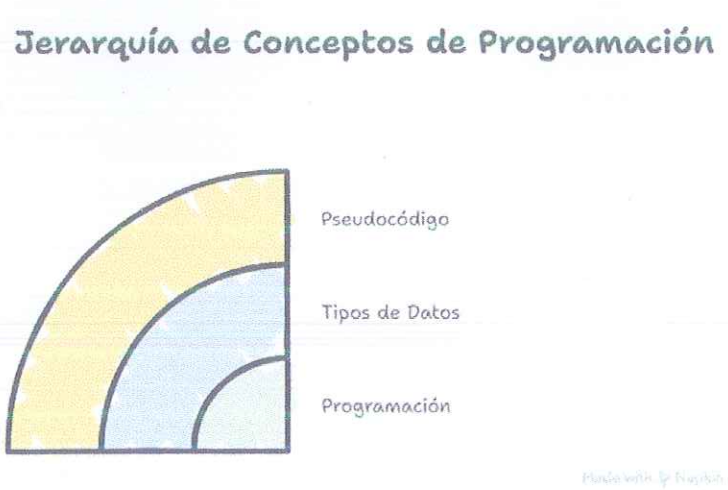
1.3.5 Datos de tipo Booleano

1.4 Pseudocódigo

Resultado de Aprendizaje

Brinda asistencia técnica en el desarrollo de aplicaciones de software, desde el análisis del problema y la planificación del proyecto, hasta la implementación, el mantenimiento, la prueba y la documentación.

DIAGRAMA DE APRENDIZAJE

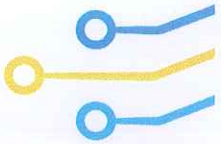


SÍNTESIS

La programación consiste en crear instrucciones para que un ordenador realice tareas específicas, agrupadas en programas para automatizar procesos y resolver problemas de manera eficiente. La programación estructurada utiliza estructuras lógicas como secuencias, decisiones y repeticiones, promoviendo claridad y modularidad.

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.

1.1. Introducción a la programación.

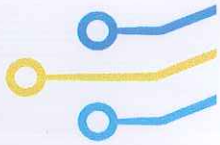
La programación es el proceso de creación de instrucciones para que un ordenador realice tareas específicas. Estas instrucciones se agrupan en programas, que permiten automatizar procesos y resolver problemas de manera eficiente. La programación estructurada es considerada como la metodología clásica, la misma que se basa en el uso de estructuras lógicas: secuencias, decisiones y repeticiones, promoviendo claridad y modularidad (Vallalobos, 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.

1.2. Programación por computadora.

La programación por computadora implica la traducción de algoritmos y soluciones diseñadas en lenguaje humano a un lenguaje comprensible para las máquinas, como C, Java, Python, entre otros. Estos lenguajes funcionan como intermediarios entre los programadores y el hardware, utilizando compiladores o intérpretes para transformar código fuente en instrucciones ejecutables (Sebesta, Los lenguajes de programación se dividen en tres tipos de lenguaje de programación.

- **Lenguaje máquina:** Programación binaria, difícil de programar y dependiente de la máquina
- **Lenguaje de bajo nivel (Ensamblador**: Usa símbolos nemotécnicos, necesita ser traducido



- **Lenguajes de alto nivel**: Cercano al lenguaje natural, tiempo de programación relativamente corto, es independiente de la máquina.

1.3. 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.

1.3.1. Datos de tipo String

Estos datos Según Fernández y Gómez (2021) son muy utilizados para almacenar información textual, como nombres, direcciones o descripciones, son una clase que representa cadenas de caracteres, es decir, texto escrito que puede contener letras, números y símbolos, son uno de los tipos de datos más utilizados en los lenguajes de programación como Java, Python, etc.

Ejemplo en Java

```
String nombre = "Juan";
System.out.println(nombre);
```

Ejemplo en Python

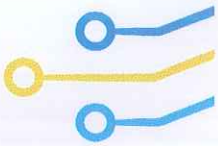
```
mensaje = "Hola, Mundo"
print(mensaje)
```

1.3.2. Datos de tipo Integer

Como señalan Martínez y López (2020), los enteros tienen un rango limitado según el lenguaje y la arquitectura del sistema, debido al tamaño de los bits asignados. En Java y Python son un tipo de dato que se utiliza para representar números enteros, es decir, números que no son decimales y pueden ser positivos o negativos.

Ejemplo en Java

```
int num = 5;
System.out.println(num);
```



Ejemplo en Python

```
num = 5
print(num)
```

1.3.3. Datos de tipo Float

Según Rodríguez y Pérez (2022), los valores *Float* son datos de tipo primitivo que almacena números con punto flotante, en los lenguajes de programación son datos que representa una coma flotante de precisión simple. Se utiliza para ahorrar memoria en grandes matrices de números en coma flotante y es menos preciso que el tipo de datos double.

Ejemplo en Java

```
float f = 35.5f;
System.out.println(f);
```

Ejemplo en Python

```
f = 19.99
print(f)
```

1.3.4. Datos de tipo Double

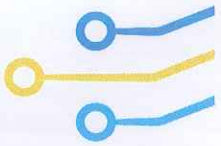
Son un tipo de datos numéricos que se utilizan para representar números de punto flotante de doble precisión, son útiles en situaciones que requieren un alto grado de precisión o un amplio rango de valores, se utilizan en cálculos científicos, aplicaciones financieras y cálculos matemáticos complejos, Torres, F. y Morales, S. (2023).

Ejemplo en Java

```
double n1 = 8.9587979797797;
System.out.println(n1);
```

Ejemplo en Python

```
n1 = 8.9587979797797
print(n1)
```



1.3.5. 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, Gutiérrez y Ramírez (2021).

Ejemplo en Java

```
boolean mayor_edad=true;
    if (mayor_edad =!false){
       System.out.println("es niño");
    }
```

Ejemplo en Python

```
es_valido=True
   if es_valido:
   print("El valor es válido")
```

1.4. 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 (2020), 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
```

PREGUNTAS:

¿Qué entiende por programación por computadores?

¿Qué es un dato en programación?

¿La palabra "Hola", a que tipo de dato corresponde?

¿Defina al dato de tipo integer?

¿Cuál es la diferencia entre un dato float y double?

¿Defina al dato de tipo booleano?

UNIDAD 2: ESTRUCTURAS BÁSICAS Y TÉCNICAS PARA REPRESENTAR ALGORITMOS

2.1 Algoritmos

2.1.1 Introducción, características y clasificación

2.2 Diagramas de flujo

2.2.1 Introducción a los diagramas de flujo

2.2.2 Características

2.2.3 Simbología

2.3 Estructuras Algorítmicas secuenciales

2.3.1 Introducción a las estructuras algorítmicas secuenciales

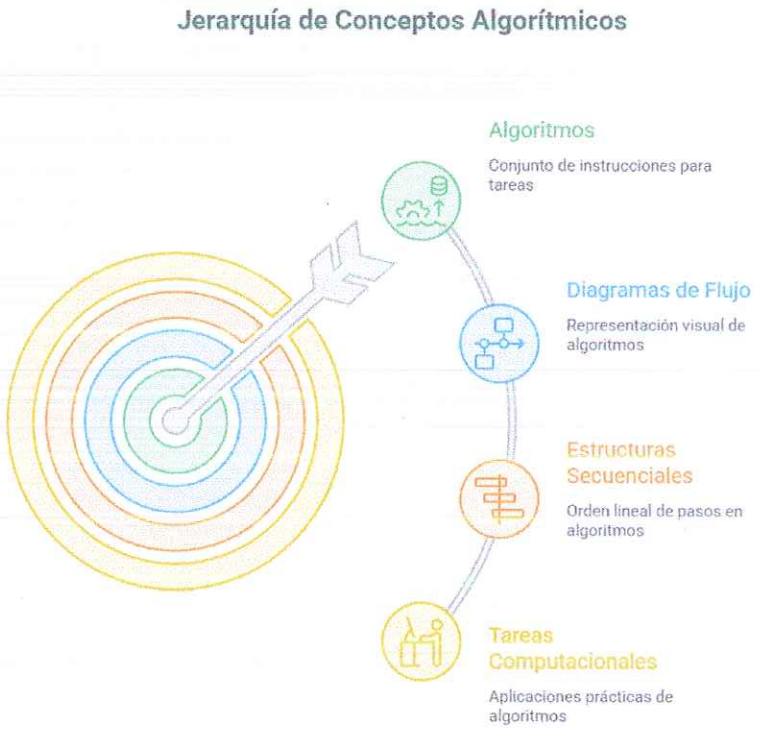
2.3.2 Características

2.3.3 ejemplos

Resultado de Aprendizaje

Brinda asistencia técnica en el desarrollo de aplicaciones de software, desde el análisis del problema y la planificación del proyecto, hasta la implementación, el mantenimiento, la prueba y la documentación.

DIAGRAMA DE APRENDIZAJE



SÍNTESIS

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.

La integración de algoritmos es crucial para mejorar el rendimiento de los sistemas informáticos. Un algoritmo bien estructurado puede reducir significativamente el tiempo de ejecución de un programa y el uso de recursos, lo cual es especialmente importante en aplicaciones de gran escala y sistemas de tiempo real.

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.

2.1. 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, Leiserson, Rivest, & Stein, 2009).

La integración de algoritmos es crucial para mejorar el rendimiento de los sistemas informáticos. Un algoritmo bien estructurado puede reducir significativamente el tiempo de ejecución de un programa y el uso de recursos, lo cual es especialmente importante en aplicaciones de gran escala y sistemas de tiempo real. La optimización de algoritmos también juega un papel importante en la ciencia de datos y el aprendizaje automático, donde grandes volúmenes de datos deben ser procesados rápidamente (Mitchell, 1997).

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).

2.1.1. 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, Leiserson, Rivest & Stein, 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 (Hopcroft, Motwani & Ullman, 2006).

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, Hopcroft & Ullman, 1983).

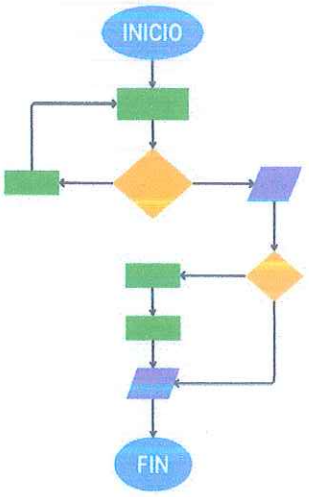
Efectividad. - Las operaciones de un algoritmo deben ser básicas y realizables en un tiempo finito. Esto implica que cada operación puede ser llevada a cabo, al menos en principio, por una persona usando lápiz y papel (Knuth, 1997).

2.1.2. 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.

2.2. Diagramas de flujo

Ilustración 1 Diagrama de Flujo



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

2.2.1. Introducción a los diagramas de flujo

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, Ramírez, A., & Torres, C. (2022).

2.2.2. 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

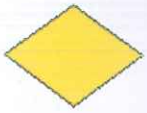
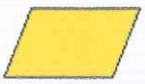
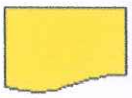
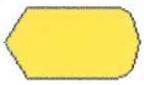
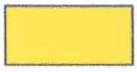
Son versátiles y pueden aplicarse a diversos campos, desde la ingeniería de software hasta la gestión de procesos empresariales, son útiles tanto para documentar sistemas existentes como para diseñar nuevos sistemas.

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, L. (2023).

2.2.3. 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

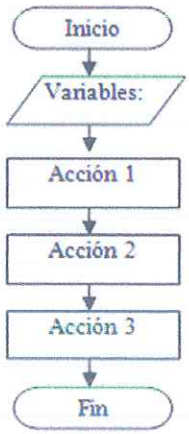
	**Inicio/Final** Se utiliza para indicar el inicio y el final de un diagrama; de Inicio sólo puede salir una línea de flujo y al final sólo debe llegar una línea		**Decisión** Indica la comparación de dos datos y dependiendo del resultado lógico (falso o verdadero) se toma la decisión de seguir un camino 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; Tomado de CITATION Rod24 \l 3082 (Rodríguez, 2024)

2.3. Estructuras Algorítmicas secuenciales

2.3.1. Introducción a las estructuras algorítmicas secuenciales

Ilustración 3 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, Fernández, J., & Torres, C. (2022).

2.3.2. 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.

Ejemplos

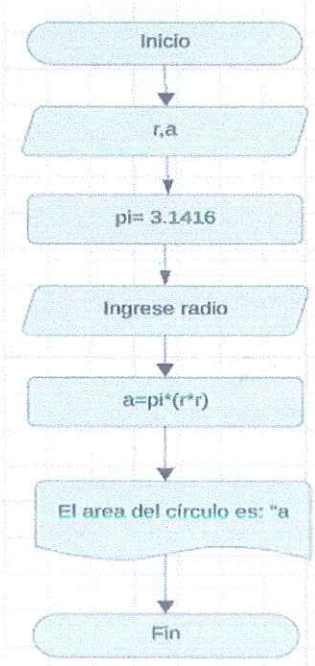
Ejemplo 1: Desarrollar un algoritmo que calcule el área de un círculo.

Ejemplo en Java

Algoritmo

```
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);
    }
}
```

Diagrama de Flujo



Ejemplo en Python

```
r=float(input("Ingrese el radio del círculo \n"))
pi=3.1416
a=pi*(r*r)
print("El área del círculo es: ",a)
```

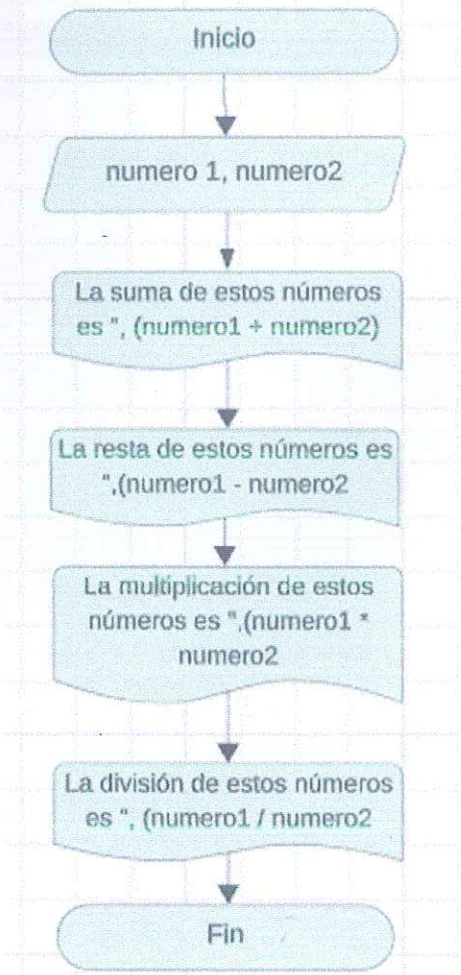
Ejemplo 2: Desarrollar un algoritmo que dados dos números, muestre la suma, resta, división y multiplicación de ambos.

Ejemplo en Java

Algoritmo

```
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));
   System.out.println("La división de estos números es " + (numero1 / numero2));
   }
}
```

Diagrama de Flujo



Ejemplo en Python

```
#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))
print("La división de estos números es ", (numero1 / numero2))
```

Actividad: Desarrollar los siguientes algoritmos (Java, Python)

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 cuanto 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.

UNIDAD 3: LENGUAJE ALGORÍTMICO Y ESTRUCTURAS DE CONTROL

3.1 Estructuras Algorítmicas selectivas

3.1.1 Introducción a las estructuras Algorítmicas selectivas.

3.1.2 Características

3.1.3 Clasificación

3.1.3.1 Estructuras Algorítmicas selectivas simples

3.1.3.2 Estructuras Algorítmicas selectivas dobles

3.1.3.3 Estructuras Algorítmicas selectivas múltiples

3.2 Estructuras Algorítmicas cíclicas

3.2.1 Introducción a las estructuras Algorítmicas cíclicas

3.2.2 Características

3.2.3 Clasificación

3.2.3.1 Estructuras Algorítmicas cíclicas While

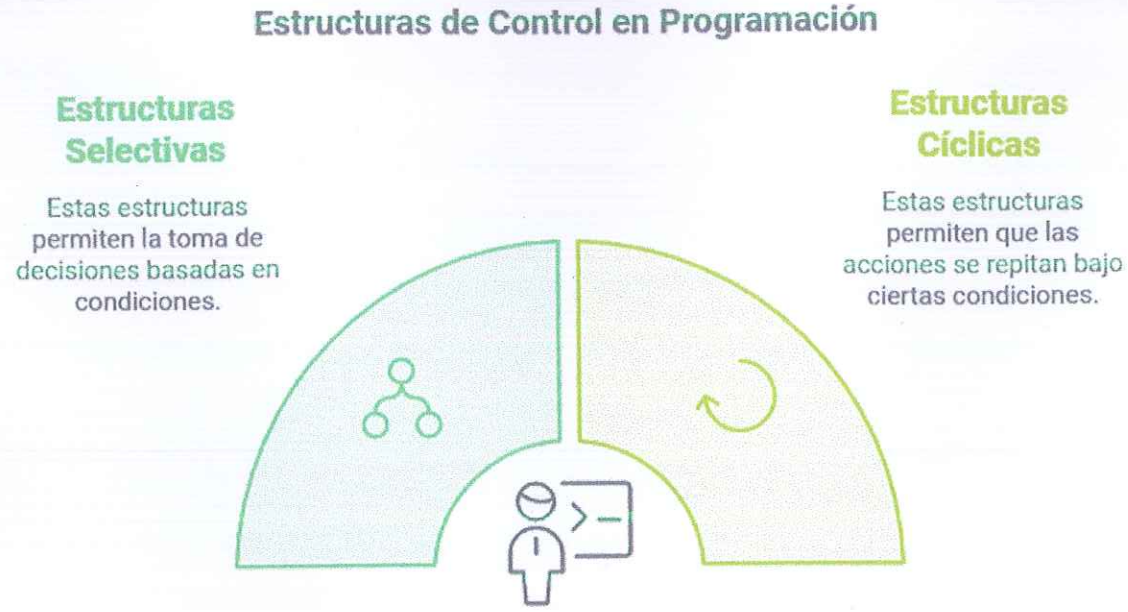
3.2.3.2 Estructuras Algorítmicas cíclicas Do-While

3.2.3.3 Estructuras Algorítmicas cíclicas For

Resultado de Aprendizaje

Utiliza herramientas y tecnologías de programación para llevar a cabo tareas específicas en el campo de desarrollo de software.

DIAGRAMA DE APRENDIZAJE



SÍNTESIS.

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.

3.1. Estructuras Algorítmicas Selectivas

3.1.1. 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 (Fernández & Torres, 2022).

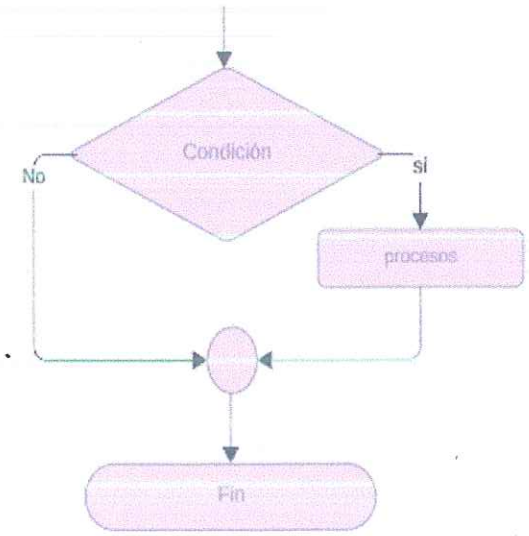
3.1.2. Características

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

3.1.3. Clasificación

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

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.

3.1.3.1. Estructuras Selectivas Simples: Las estructuras selectivas simples utilizan una sola condición para determinar la ejecución de un bloque de código. La instrucción if es el ejemplo más común (López & Hernández, 2019).

Características

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

Ejemplos

Ejemplo 1: Desarrollar un algoritmo que me permita ingresar un número y determinar si el número ingresado es positivo.

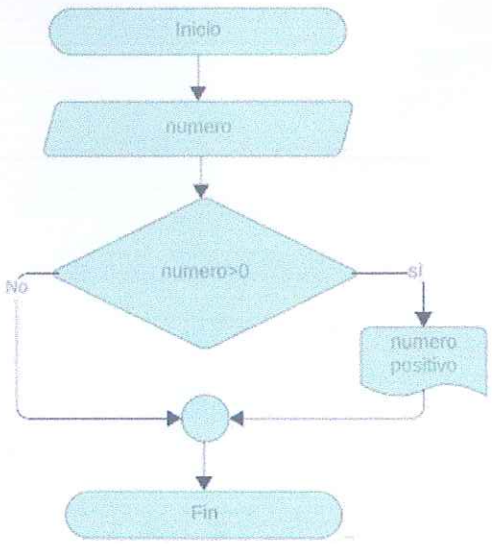
Ejemplo en Java:

Algoritmo

```
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.");
      }
      // Fin del programa
      System.out.println("Programa finalizado.");
   }
}
```

Diagrama de Flujo



Ejemplo en Python:

```
numero=int(input("Ingrese un número"))
#Estructura selectiva simple para verificar si el número es positivo
if (numero > 0):
 print("El número es positivo.")
```

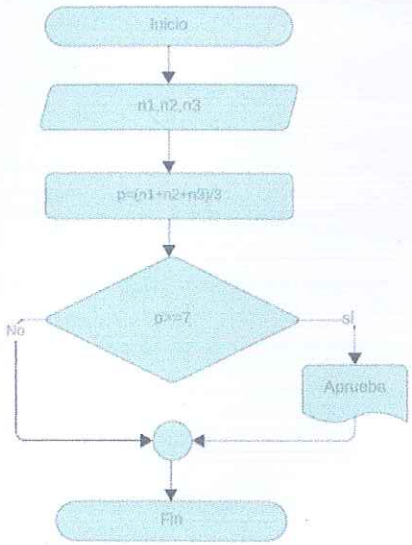
Ejemplo 2: Desarrollar un algoritmo que, en base a tres notas ingresadas por teclado, me permita obtener el promedio de un estudiante, si el promedio es mayor o igual a siete, imprimir un mensaje que diga, el estudiante aprueba.

Ejemplo en Java:

Algoritmo

```
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);
      }
   }
}
```

Diagrama de Flujo



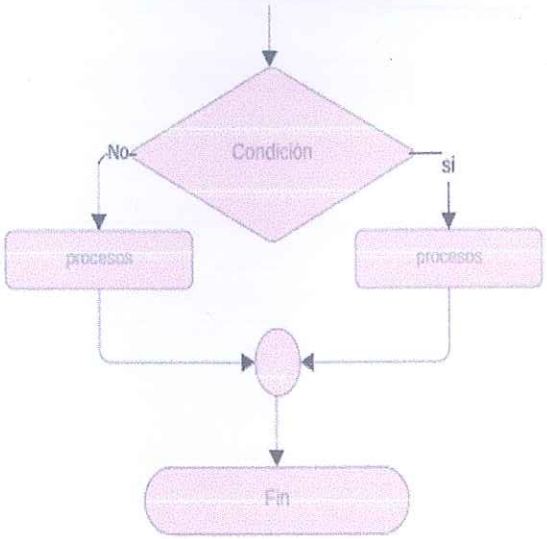
Ejemplo en Python:

Algoritmo

```
# 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)
```

3.1.3.2. Estructuras Algorítmicas Selectivas Dobles

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.

Las estructuras selectivas dobles permiten evaluar dos caminos posibles basados en una condición: si la condición es verdadera, se ejecuta un bloque de código; si es falsa, se ejecuta otro (Rodríguez & García, 2021).

Características

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

Ejemplos

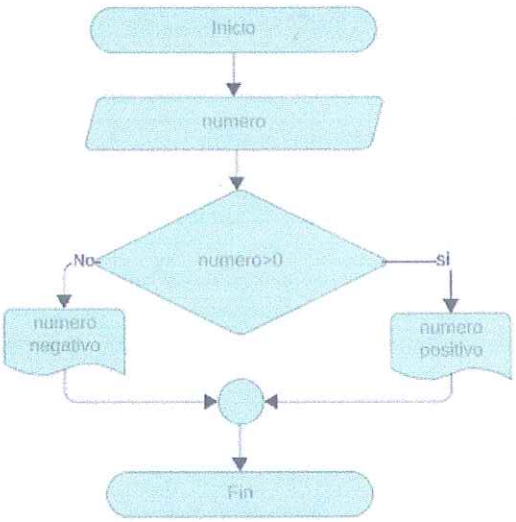
Ejemplo 1. Desarrollar un algoritmo que me permita ingresar un número y determinar si el número ingresado es positivo o negativo.

Ejemplo en Java:

Algoritmo

```
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{
      System.out.println("El número es negativo.");
    }
   }
}
```

Diagrama de Flujo



Ejemplo en Python:

Algoritmos

```
numero=int(input("Ingrese un número"))
#Estructura selectiva simple para verificar si el número es positivo
if (numero > 0):
 print("El número es positivo.")
else:
 print("El número es negativo.")
```

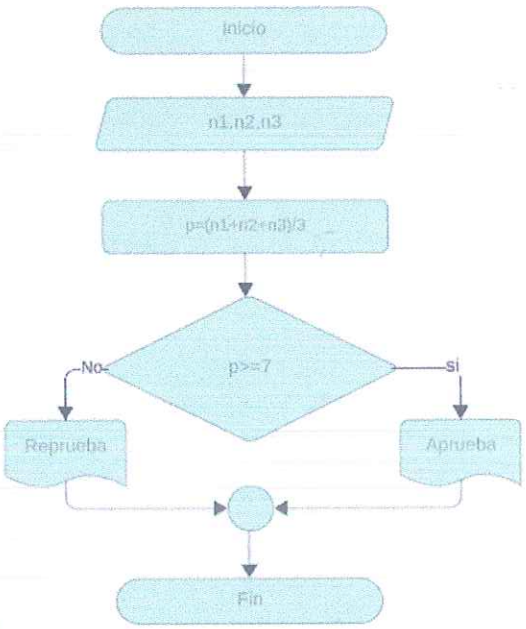
Ejemplo 2: Desarrollar un algoritmo que, en base a tres notas ingresadas por teclado, me permita obtener el promedio de un estudiante, si el promedio es mayor o igual a siete, imprimir un mensaje que diga, el estudiante aprueba, caso contrario el estudiante reprueba.

Ejemplo en Java:

Algoritmo

```
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 apruet
      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);
     }
   }
 }
```

Diagrama de Flujo



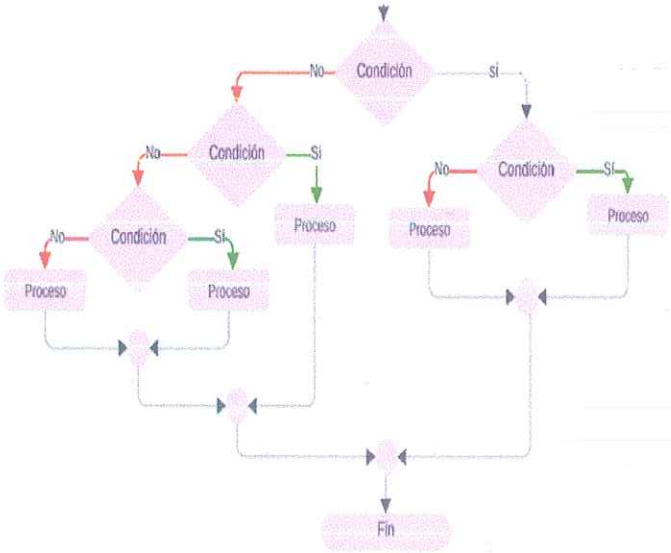
Ejemplo en Python:

Algoritmo

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

3.1.3.3. Estructuras Algorítmicas Selectivas Múltiples

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.

Las estructuras selectivas múltiples permiten evaluar más de dos condiciones y ejecutar diferentes bloques de código según el resultado de la evaluación. El switch en Java es un ejemplo típico (Pérez & Martínez, 2020).

Características

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

Ejemplos

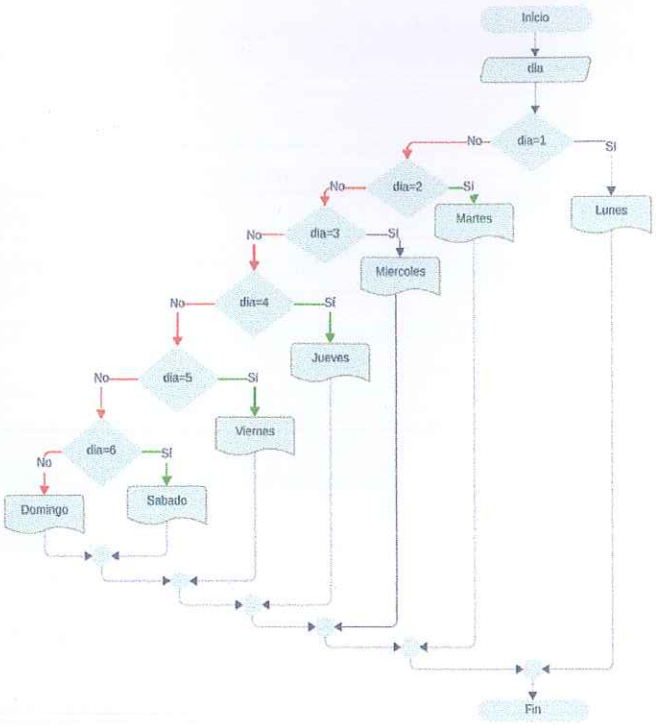
Ejemplo 1: Desarrollar un algoritmo que me permita ingresar un número del 1 al 7 y según el número ingresado imprimir el día de la semana correspondiente.

Ejemplo en Java:

Algoritmo

```
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) {
      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 en Python:

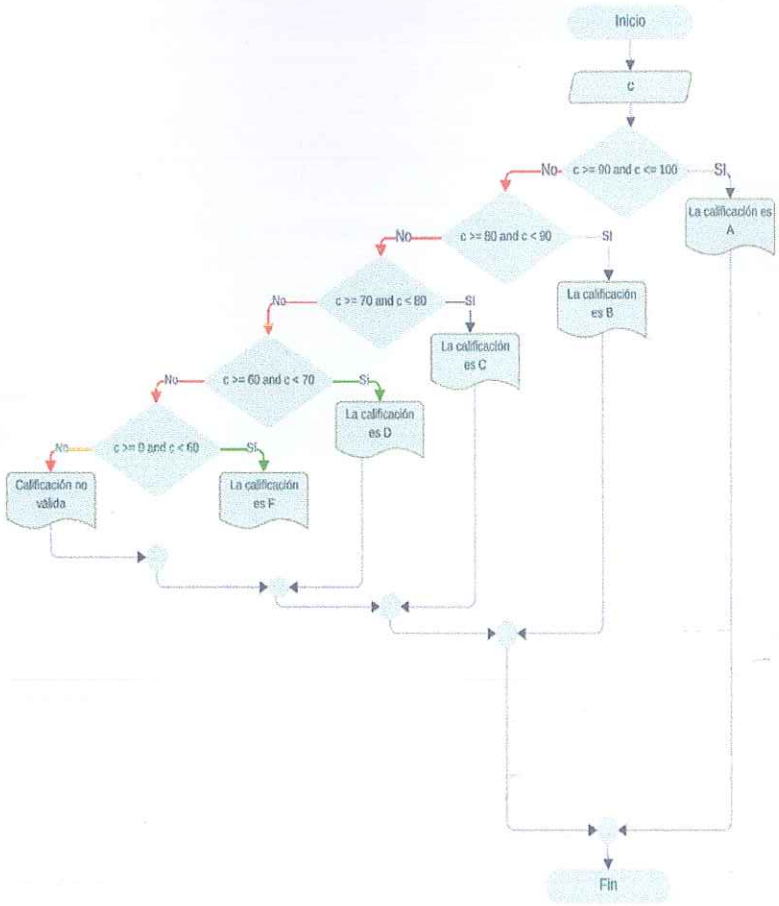
Algoritmo

```
# 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")
```

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:

Diagrama de Flujo



Algoritmo

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

Ejemplo en Python:

Algoritmo

```
#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.")
```

3.2. Estructuras Algorítmicas Cíclicas

3.2.1. Introducción a las Estructuras Algorítmicas Cíclicas

Las estructuras algorítmicas cíclicas permiten ejecutar un bloque de código repetidamente mientras se cumpla una condición, estas estructuras son esenciales para tareas un grupo de procesos o un bloque de código debe repetirse varias veces (Gómez & Ramírez, 2023).

3.2.2. Características

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

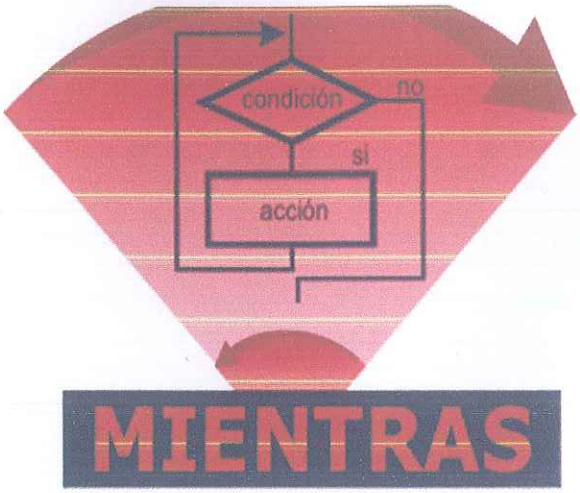
3.2.3 Clasificación

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

3.2.3.1. Estructuras Algorítmicas Cíclicas While

Las estructuras algorítmicas while evalúan una condición antes de cada iteración y se ejecuta mientras esta condición sea verdadera (Rodríguez & García, 2021).

Ilustración 7 Estructuras Algorítmicas Cíclicas While



Nota. La estructura While evalúa la condición al inicio del algoritmo. Tomado de CITATION For \l 3082 *(Formación en Ambientes Virtuales de Aprendizaje, s.f)*

Características

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

Ejemplos

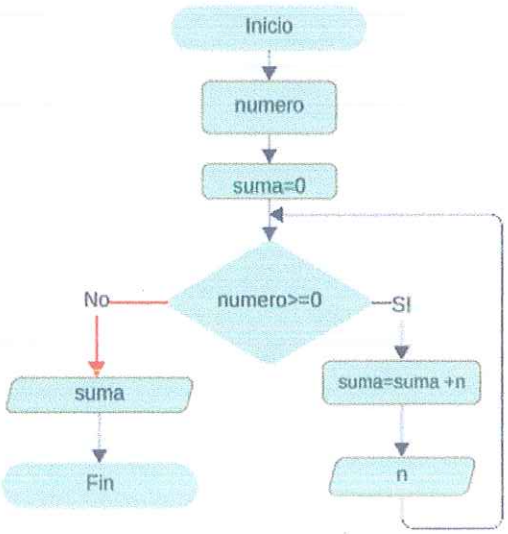
Ejemplo 1: Desarrollar un algoritmo que permita ingresar números positivos y calcula la suma de estos números. El bucle continúa hasta que el usuario ingresa un número negativo.

Ejemplo en Java:

Algoritmo

```
package algoritmos_ciclicos;
import java.util.Scanner;
public class Numeros_positivos {
   public static void main(String[] args) {
   Scanner sc = new Scanner(System.in);
   int numero,suma = 0;
   // Solicitar al usuario que ingrese números positivos
   System.out.println("Ingrese un número positivo (o un número negativo para terminar): ");
   numero = sc.nextInt();
   // Bucle while para sumar los números positivos
   while (numero >= 0) {
      suma += numero;// Sumar el número ingresado a la suma total
      // Solicitar al usuario que ingrese otro número
      System.out.println("Ingrese otro número positivo (o un número negativo para terminar): ");
```

Diagrama de Flujo



```
        numero = sc.nextInt();
        }

        // Mostrar la suma total de los números positivos ingresados
        System.out.println("La suma de los números positivos ingresados es: " + suma);
    }
}
```

Ejemplo en Python:

Algoritmo

```
#Solicitar al usuario que ingrese números positivos
numero=int(input("Ingrese un número positivo (o un número negativo para terminar:\n "))
#Bucle while para sumar los números positivos
suma=0
while (numero >= 0):

    suma = suma+numero;#Sumar el número ingresado a la suma total
    #Solicitar al usuario que ingrese otro número
    numero=int(input(f"Ingrese otro número positivo (o un número negativo para terminar:\n"))
  else:
    #Mostrar la suma total de los números positivos ingresados
    print(f"La suma de los números positivos ingresados es: {suma}")
```

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

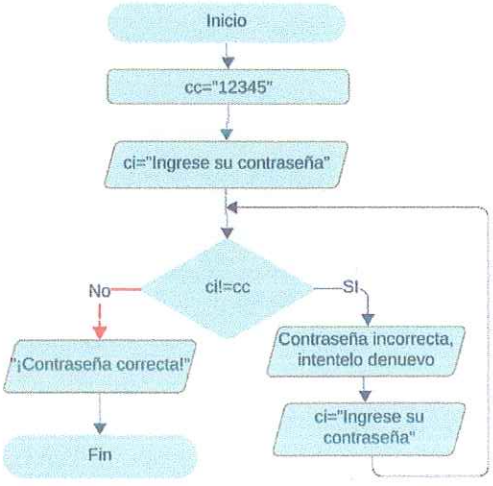
Ejemplo en Java:

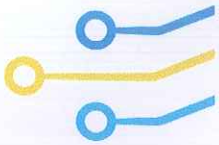
Algoritmo

```
package algoritmos_ciclicos_while;
import java.util.Scanner;
public class contrasena {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String cc = "12345";
        String ci;
// Solicitar al usuario que ingrese la contraseña
        System.out.println("Ingrese la contraseña: ");
```

Diagrama de Flujo





```
        ci = scanner.nextLine();
// Bucle while para verificar la contraseña
        while (!ci.equals(cc)) {
            System.out.println("Contraseña incorrecta, inténtelo de nuevo.");
            System.out.println("Ingrese la contraseña: ");
            ci = scanner.nextLine();
        } // Mostrar mensaje de éxito
        System.out.println("¡Contraseña correcta!");
    }
}
```

Ejemplo en Python:

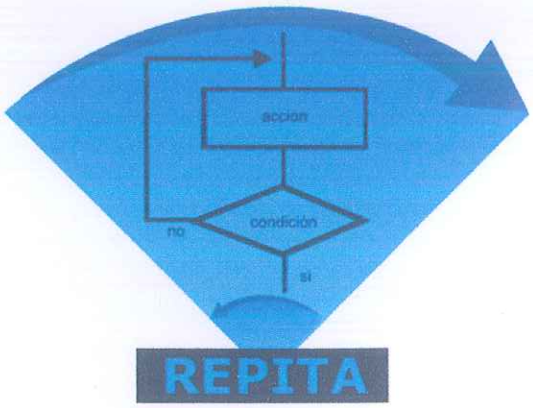
Algoritmo

```
# Definir la contraseña correcta
cc= "12345"
# Solicitar al usuario que ingrese la contraseña
ci = input("Ingrese la contraseña: \n ")
# Bucle while para verificar la contraseña
while ci != cc:
    print("Contraseña incorrecta, inténtelo de nuevo.")
    ci = input("Ingrese la contraseña: \n ")
    # Mostrar mensaje de éxito
else:
    print("¡Contraseña correcta!")
```

3.2.3.2. Estructuras Algorítmicas Cíclicas Do-While

Ilustración 8 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 (Fernández & Torres, 2022).



Nota. La estructura D-While ejecuta los procesos por lo menos una vez. Tomado de CITATION For \l 3082 *(Formación en Ambientes Virtuales*

Características

- **Evaluación Posterior**: La condición se evalúa después de la ejecución del bloque.
- **Garantía de Ejecución**: El bloque se ejecuta al menos una vez.

Ejemplos

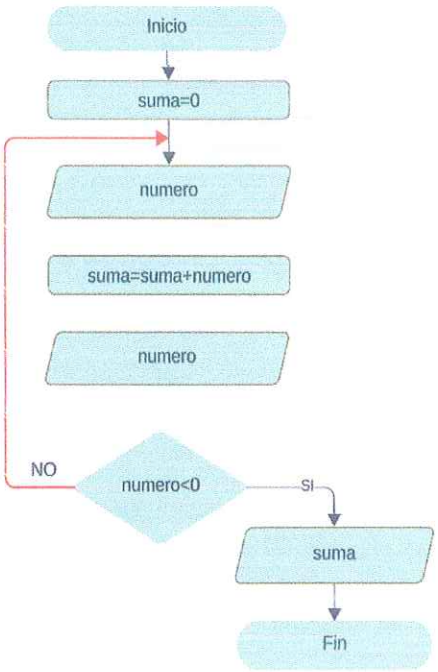
Ejemplo 1: Desarrollar un algoritmo que permita ingresar números positivos y calcula la suma de estos números. El bucle continúa hasta que el usuario ingresa un número negativo.

Ejemplo en Java:

Algoritmo

```
package algorítmos_do_while;
import java.util.Scanner;
public class numeros_positivos {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      int numero;
      int suma = 0;
      // Estructura do-while para sumar los números positivos
      do {
         System.out.println("Ingrese un número positivo (o un número negativo para terminar): ");
         numero = scanner.nextInt();
         if (numero >= 0) {
            suma += numero;
         }
         // Sumar el número ingresado a la suma total
      } while (numero >= 0);
      // Mostrar la suma total de los números positivos ingresados
```

Diagrama de Flujo



```
        System.out.println("La suma de los números positivos ingresados es: " + suma);
    }
}
```

Ejemplo en Python:

Algoritmo

```
suma = 0
# Solicitar al usuario que ingrese números positivos
numero=int(input("Ingrese un número positivo (o un número negativo para terminar): \n"))
# Bucle while para sumar los números positivos
while True:
    suma+= numero;# Sumar el número ingresado a la suma total
    # Solicitar al usuario que ingrese otro número
    numero=int(input("Ingrese otro número positivo (o un número negativo para terminar): \n"))
    if(numero <0):
    #Mostrar la suma total de los números positivos ingresados
     print("La suma de los números positivos ingresados es: ",suma)
```

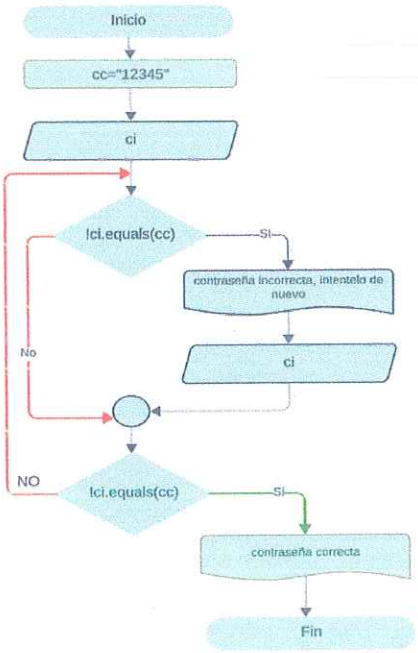
Ejemplo 2: Desarrollar un algoritmo que le permita al usuario ingresar una contraseña y continúa solicitándola hasta que ingrese la correcta.

Ejemplo en Java:

Algoritmo

```
package algoritmos_do_while;
import java.util.Scanner;
public class Contraseña {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String cc = "12345";
        String ci;
// Solicitar al usuario que ingrese la contraseña
        System.out.println("Ingrese la contraseña: ");
        ci = scanner.nextLine();
// Bucle do while para verificar la contraseña
        do {
            if (!ci.equals(cc)) {
                System.out.println("Contraseña incorrecta, inténtelo de nuevo.");
                System.out.println("Ingrese la contraseña: ");
                ci = scanner.nextLine();
            }
```

Diagrama de Flujo



```
    } while (!ci.equals(cc));
    // Mostrar mensaje de éxito
    System.out.println("¡Contraseña correcta!");
  }
}
```

Ejemplo en Python:

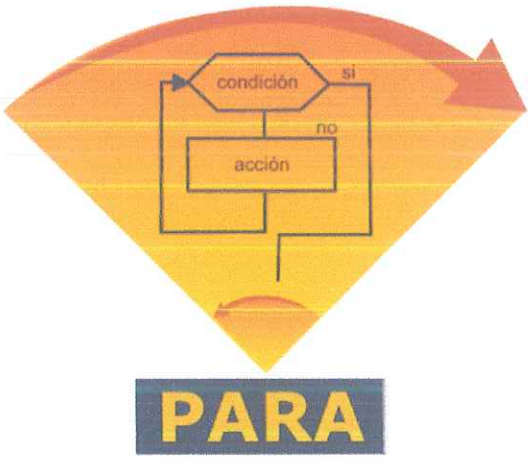
Algoritmo

```
# Definir la contraseña correcta
cc = "12345"
# Inicializar la variable para la contraseña ingresada
# Bucle que emula un do-while
while True:
  ci = input("Ingrese la contraseña:\n ")
  if ci == cc:
    break
  print("Contraseña incorrecta, inténtelo de nuevo.")
# Mostrar mensaje de éxito
print("¡Contraseña correcta!")
```

3.2.3.3. Estructuras Algorítmicas Cíclicas For

Ilustración 9 Estructuras Algorítmicas Cíclicas For

La estructura for se utiliza cuando se conoce de antemano el número de iteraciones que se deben realizar, es ideal para recorrer colecciones y contar iteraciones (Gómez & Ramírez, 2023).



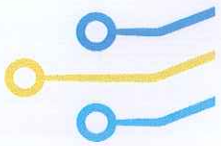
Nota. Esta estructura es ideal cuando se conoce el número de interacciones de debe cumplir el algoritmo, Tomado de CITATION For \l 3082 *(Formación en Ambientes Virtuales de Aprendizaje,*

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:

Algoritmo

```
package pkgfor;
import java.util.Scanner;
public class Area_circulo {
   public static void main(String[] args) {
      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();
   }
}
```

Ejemplo en Python:

Algoritmo

```
# 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: "))
# 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:

Algoritmo

```
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);
        }
    }
}
```

Ejemplo en Python:

Algoritmo

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

Actividad: 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.

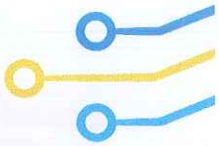
Clasificación	IMC (Kg/m²)	Riesgo
Normal	18.5 - 24.9	Promedio
Sobrepeso	25 - 29.9	Aumentado
Obesidad grado I	30 - 34.9	Moderado
Obesidad grado II	35 - 39.9	Severo
Obesidad grado III	Más de 40	Muy Severo

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 cuales no.

UNIDAD 4: SUBPROGRAMAS Y ARREGLOS

4.1 Listas

4.1.1 Introducción

4.1.2 Características

4.2 Vectores

4.2.1 Introducción

4.2.2 Características

4.3 Matrices

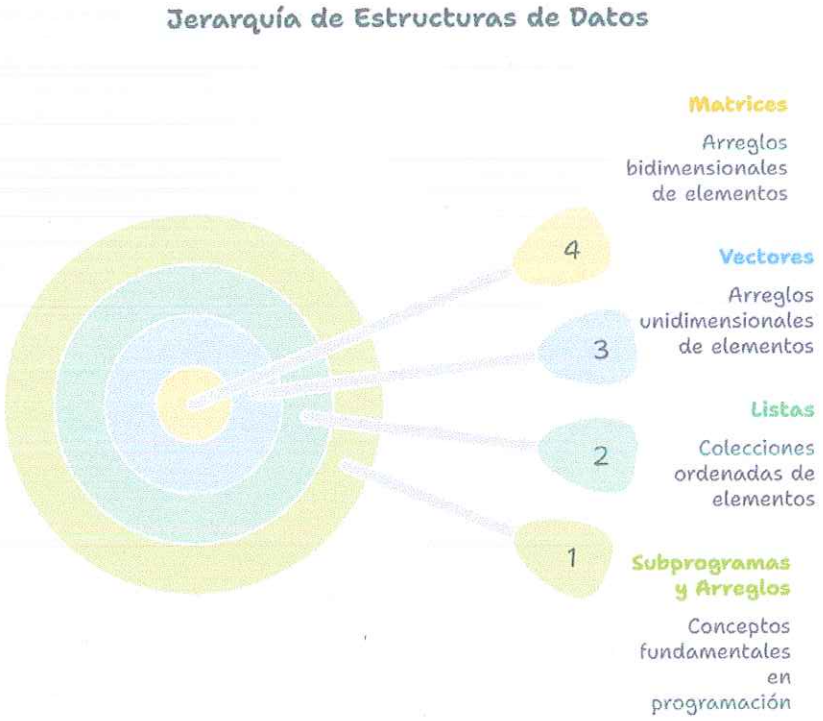
4.3.1 Introducción

4.3.2 Características

Resultado de Aprendizaje

Utiliza herramientas y tecnologías de programación para llevar a cabo tareas específicas en el campo de desarrollo de software.

DIAGRAMA DE APRENDIZAJE



SINTESIS.

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.

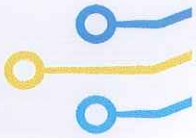
- Las **listas** se caracterizan por su flexibilidad y capacidad dinámica, lo que las convierte en estructuras ideales para gestionar colecciones de datos cuyo tamaño varía durante la ejecución del programa. Su implementación puede variar según el lenguaje, incluyendo listas enlazadas, dobles o circulares, 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**, como arreglos bidimensionales, permiten representar datos en forma de filas y columnas. Son ampliamente utilizadas en el procesamiento de imágenes, álgebra lineal, simulaciones científicas y redes neuronales, su correcta implementación y manipulación son clave para el desarrollo de algoritmos que requieren estructuras complejas y gran volumen de información.

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.

4.1. Listas

4.1.1. 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).

4.1.2. 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 (Ramírez & Torres, 2021).

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

Ejemplo en Java:

Algoritmo

```
import java.util.ArrayList;
public class ListaEjemplo {
    public static void main(String[] args) {
        ArrayList<String> frutas = new ArrayList<>();
        frutas.add("manzana");
        frutas.add("banana");
        frutas.add("naranja");
        frutas.remove("banana");
        System.out.println(frutas); // [manzana, naranja]
    }
}
```

Ejemplo en Python:

Algoritmo

```
# Lista de frutas
frutas = ["manzana", "banana", "naranja"]
frutas.append("mango")  # Agregar elemento
frutas.remove("banana")  # Eliminar elemento
print(frutas)  # ['manzana', 'naranja', 'mango']
```

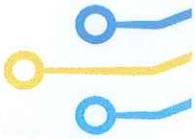
4.2. Vectores

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:

Algoritmo

```
public class VectorEjemplo {
   public static void main(String[] args) {
      int[] edades = {23, 45, 18, 30};
      for (int edad : edades) {
         System.out.println(edad);
      }
   }
}
```

Ejemplo en Python:

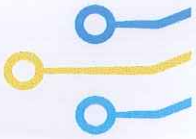
Algoritmo

```
# 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:

Algoritmo

```
public class MatrizEjemplo {
    public static void main(String[] args) {
        int[][] matriz = {
            {1, 2},
            {3, 4}
        };
        for (int i = 0; i < matriz.length; i++) {
            for (int j = 0; j < matriz[i].length; j++) {
                System.out.print(matriz[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

Ejemplo en Python:

Algoritmo

```
# Matriz 2x2
matriz = [[1, 2], [3, 4]]
for fila in matriz:
    for elemento in fila:
        print(elemento, end=" ")
    print()
```

ACTIVIDADES DE LA UNIDAD

PRÁCTICA: Desarrollar los siguientes algoritmos en Java, Python

Ejercicios de 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.

Ejercicios de 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.

Ejercicios de 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).

UNIDAD 5: PROGRAMACIÓN ESTRUCTURADA.

5.1 Subrutinas

5.2 Bloques

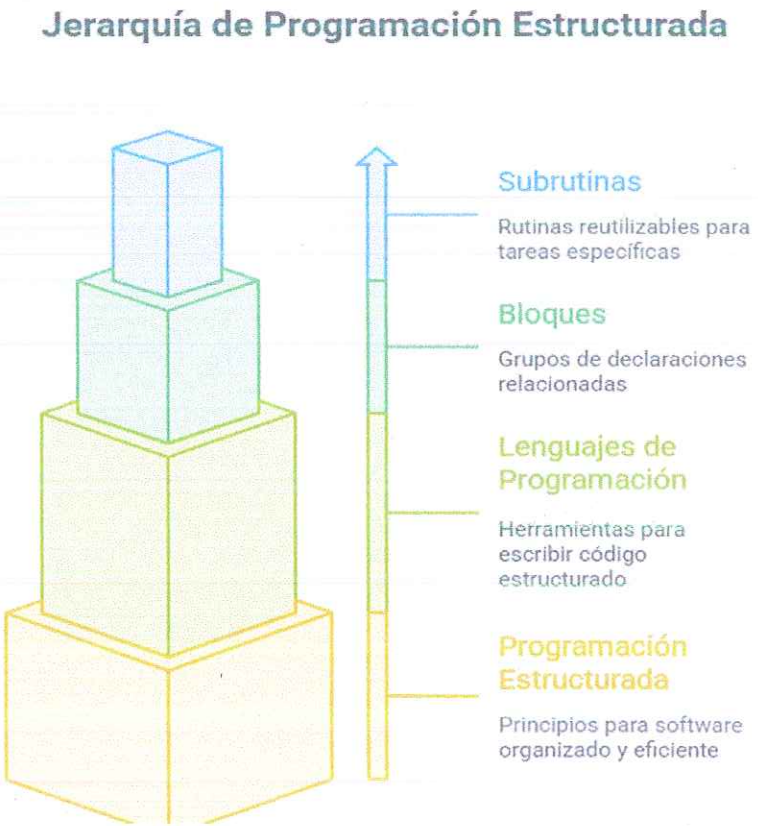
5.3 Lenguajes de programación

5.4 Programación estructurada

Resultado de Aprendizaje

Utiliza herramientas y tecnologías de programación para llevar a cabo tareas específicas en el campo de desarrollo de software.

DIAGRAMA DE APRENDIZAJE



SINTESIS.

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.

5.1 Subrutinas

¿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, 2021)

¿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:

Algoritmo

```
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:

Algoritmo

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

5.2 Bloques

¿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, 2021).

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:

Algoritmo

```
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:

Algoritmo

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

5.3 Lenguajes de programación

¿Qué son los lenguajes estructurados?

Los lenguajes de programación estructurada son aquellos que permiten escribir programas utilizando estructuras de control bien definidas como secuencia, selección e iteración. Estos lenguajes promueven la claridad, modularidad y la facilidad de mantenimiento del código, evitando el uso de instrucciones como goto, que dificultan el seguimiento del flujo lógico del programa (Gaddis, 2021).

Características clave

Los lenguajes estructurados comparten ciertas características fundamentales:

Uso de subrutinas o funciones para modularizar el código.

Control de flujo mediante estructuras como if, while, for.

Tipado explícito o implícito de variables.

Claridad sintáctica que facilita la lectura y depuración del código.

Estas características permiten que los programas sean más comprensibles, predecibles y fáciles de escalar, lo cual es esencial en proyectos colaborativos o de larga duración (Downey, 2020).

Ejemplos de lenguajes estructurados

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

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

5.4 Programación estructurada

La programación estructurada surgió como una respuesta a la complejidad creciente de los programas informáticos en las décadas de 1960 y 1970. Este paradigma se basa en tres estructuras fundamentales: *secuencia*, *selección* y *repetición*, que permiten organizar el flujo del programa de forma lógica y predecible. Según la Universidad San Marcos, "la programación estructurada es conocida como el método de desarrollo de programas más viable porque utiliza tres sistemas de control [...] que se pueden mezclar para crear programas que apliquen cualquier requerimiento de procesamiento de datos". Esta metodología facilita la comprensión del código, reduce errores y promueve buenas prácticas de desarrollo.

Un ejemplo clásico de programación estructurada es el cálculo del área de un triángulo utilizando funciones y estructuras secuenciales. En Python, se puede expresar como:

Ejemplo en Python:

Algoritmo

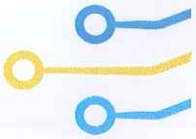
```
def area_triangulo(base, altura):
    return (base * altura) / 2

print(area_triangulo(10, 5))  # Resultado: 25.0
```

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

Bibliografía

- ✓ Fernández, R., & Gómez, M. (2021). *Estructuras de datos en programación moderna* . Editar
- ✓ Martínez, L., & López, J. (2020). *Introducción a los tipos de datos en lenguajes de programación* . Tecnología
- ✓ Rodríguez, C., & Pérez, A. (2022). *Optimización de datos numéricos en programación* . Editorial Tec
- ✓ Torres, F. y Morales, S. (2023). *Matemáticas computacionales aplicadas a la programación* . Edi
- ✓ Gutiérrez, H., & Ramírez, P. (2021). *Fundamentos de lógica en programación* .
- ✓ Instituto de Ingeniería de Software. (2021). *Guía práctica para el desarrollo de software*. Editorial Técnica.
- ✓ Larman, C. (2020). *Diseño orientado a objetos y UML*. Pearson Educación.
- ✓ Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press.
- ✓ Mitchell, T. M. (1997). Machine Learning. McGraw-Hill.
- ✓ Sedgewick, R., & Wayne, K. (2011). Algorithms (4th ed.). Addison-Wesley Professional.
- ✓ Aho, A. V., Hopcroft, J. E., & Ullman, J. D. (1983). *Estructuras de datos y algoritmos*. Addison-Wesley.
- ✓ Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introducción a los algoritmos* (3.a ed.). MIT Press.
- ✓ Knuth, D. E. (1997). El arte de programar computadoras, Volumen 1: Algoritmos fundamentales (3.a ed.). Addison-Wesley.



- ✓ Ramírez, A., & Torres, C. (2022). Modelado de Procesos y Análisis de Sistemas. Publicaciones Académicas.
- ✓ Rodríguez, L. (2023). Diagramas de Flujo en la Gestión de Proyectos. Editorial Innovación.
- ✓ Rodríguez, M. (17 de enero de 2024). *Símbolos del diagrama de flujo*. Tu Gimnasia Cerebral: https://tugimnasiacerebral.com/herramientas-de-estudio/diagrama-de-flujo-que-es-caracteristicas-y-como-hacerlo
- ✓ Fernández, J., & Torres, C. (2022). *Modelado de Procesos y Análisis de Sistemas*. Publicaciones Académicas.
- ✓ Gómez, L., & Ramírez, A. (2023). *Diagramas de Flujo en la Gestión de Proyectos*. Editorial Innovación.
- ✓ Ayala Molina, M. D. (02 de febrero de 2013). *Algoritmo selectivo simple*. Programación de Computadores: https://programaciondecomputadores.wordpress.com/category/2-estructuras-selectivas/
- ✓ Formación en Ambientes Virtuales de Aprendizaje. (s.f). Estructuras Cíclicas.
- ✓ González, M. (2020). Fundamentos de programación con Python. Alfaomega.
- ✓ Ramírez, J., & Torres, L. (2021). Estructuras de datos y algoritmos. McGraw-Hill.
- ✓ Universidad Nacional de Rosario. (2020). Estructura de datos: Vectores. Recuperado de https://www.fceia.unr.edu.ar/estruc/2006/vector.htm
- ✓ UNLaM. (2020). Vectores. Recuperado de https://miel.unlam.edu.ar/data/contenido/3635/Vectores.pdf
- ✓ Universidad Nacional del Sur. (2020). Tipos subindicados en Pascal. Recuperado de http://www.lip.uns.edu.ar/pci/Tipos_Subinidicados_en_Pascal.PDF

ELABORACIÓN, REVISIÓN Y APROBACIÓN DE PARES

Profesor(a)

Ing. Italo Marcelo Lara Pilco., MsC

Fecha de elaboración: 31/10/2025

Comisión de revisión de pares de guías de estudio del Instituto Superior Tecnológico Tena

Lcda. María Angélica Campoverde Encalada	Mg. Alvaro Santiago Toalombo Díaz
Mg. Henry Fabian Chango Chango	Mg. Duarte Mora Martha Janina

Abg. Danilo Alexander Zamora Núñez., Mg.

Fecha de revisión: 28/11/2025

Coordinador de Investigación, Desarrollo Tecnológico e Innovación

Abg. Danilo Alexander Zamora Núñez., Mg.

INSTITUTO SUPERIOR TECNOLÓGICO TENA
COORDINACIÓN DE INVESTIGACIÓN

Fecha de aprobación: 09/12/2025