ISTRED
INSTITUTO SUPERIOR TÉCNICO REY DAVID

PERIODO II
2023 - 2024

GUÍA DE APRENDIZAJE DEL ESTUDIANTE
LÓGICA DE SISTEMAS

LIC. VICTOR BARRERA REA

GESTIÓN DE BASE DE DATOS
1ER. SEMESTRE

Contenido

I. DATOS GENERALES DE LA ASIGNATURA 7

Información General 7

Requisitos 7

Descripción de la asignatura 7

Objetivos de la asignatura: 7

Introducción de la Asignatura 8

I. DATOS GENERALES DE LA UNIDAD 1 9

Información General 9

Resultado de Aprendizaje de Unidad 9

Metodología 9

DESARROLLO 10

Unidad 1: Programación Algorítmica 10

Tema 1: Conceptos básicos 10

Conceptos básicos de programación 10

Algoritmos 11

Variables y tipos de datos 11

Diagramas de Flujo y Pseudocódigo 12

ACTIVIDADES PRÁCTICAS 14

FORO DE DEBATE 15

PRÁCTICA INVESTIGATIVA 15

PRÁCTICA TEST 16

Tema 2: Funciones y operaciones 17

Funciones Entradas y Salidas de Datos 17
Operadores Relacionales y Lógicos 20
ACTIVIDADES PRÁCTICAS 22
FORO DE DEBATE 23
PRÁCTICA INVESTIGATIVA 23
PRÁCTICA TEST 24
II. DATOS GENERALES DE LA UNIDAD 2 25
Información General 25
Resultado de Aprendizaje de Unidad 25
Metodología 25
DESARROLLO 26
Unidad 2: Estructuras de control 26
Tema 1: Estructuras 26
Estructuras de selección 26
Estructura simple 26
Estructura doble 27
Estructura anidada y múltiple 28
Estructura Anidada 28
Estructura Múltiple 29
ACTIVIDADES PRÁCTICAS 32
FORO DE DEBATE 33
PRÁCTICA INVESTIGATIVA 33
PRÁCTICA TEST 34

Tema 2: Bucles 35
Estructuras de repetición 35
Bucles y sus partes 35
Bucle while (Mientras) 36
Bucle for (Para) 36
Diferencia entre while y for 37
ACTIVIDADES PRÁCTICAS 38
FORO DE DEBATE 39
PRÁCTICA INVESTIGATIVA 39
PRÁCTICA TEST 40
III. DATOS GENERALES DE LA UNIDAD 3 41
Información General 41
Resultado de Aprendizaje de Unidad 41
Metodología 41
DESARROLLO 42
Unidad 3: Arreglos de programación 42
Tema 1: Arreglos 42
Arreglos en programación 42
Declarar un array 43
En Python (listas dinámicas): 43
En Java (arreglo de tamaño fijo): 43
En C (arreglo estático): 43
Elemento del arreglo 44

Cantidad de información en arreglos 44
En Python (usando len()): 45
En Java (usando length): 45
En C (tamaño del arreglo dividiendo su memoria total por el tamaño de un elemento): 45
ACTIVIDADES PRÁCTICAS 47
FORO DE DEBATE 48
PRÁCTICA INVESTIGATIVA 48
PRÁCTICA TEST 49
Tema 2: Tipos de arreglos. 50
Tipos de arreglos 50
Arreglos Unidimensionales (Vectores): 50
Arreglos Bidimensionales (Matrices): 51
Arreglos Multidimensionales: 52
Arreglos Dinámicos: 52
Distintos lenguajes, misma idea 53
Funciones matemáticas y arreglos 53
Operaciones Matemáticas con Arreglos 53
1. Suma de elementos de un arreglo: 53
2. Cálculo de media, desviación estándar y suma con NumPy: 54
3. Multiplicación de Matrices: 54
4. Generación de un arreglo con valores aleatorios: 54
ACTIVIDADES PRÁCTICAS 56

FORO DE DEBATE 57
PRÁCTICA INVESTIGATIVA 57
PRÁCTICA TEST 58
IV. DATOS GENERALES DE LA UNIDAD 4 59
Información General 59
Resultado de Aprendizaje de Unidad 59
Metodología 59
DESARROLLO 60
Fundamentos de programación 60
Concepto y Sintaxis 61
Herramientas de control de flujo if for while do while 62
Condicionales (if) 62
Bucles (for, while, do while) 63
ACTIVIDADES PRÁCTICAS 67
FORO DE DEBATE 68
PRÁCTICA INVESTIGATIVA 68
PRÁCTICA TEST 69
Tema 2: Funciones 70
ACTIVIDADES PRÁCTICAS 73
FORO DE DEBATE 74
PRÁCTICA INVESTIGATIVA 74
PRÁCTICA TEST 75
V. Bibliografía 76

Bibliografía básica. 76

Bibliografía de consulta. 76

Anexo 1: Modelo de Foro 77

Anexo 2: Esquema de Ensayo 78

Anexo 3: Esquema de Taller/Investigación 79

Anexo 4: Rubrica de taller práctico 80

Anexo 5: Rúbrica de Foro 81

Anexo 6: Rubrica de Ensayo 82

Anexo 7: Rúbrica de Taller/Investigación 83

I. DATOS GENERALES DE LA ASIGNATURA

Información General

1.1.	Asignatura	Lógica de Sistemas					
1.2.	Carrera	Gestión de Base de Datos					
1.3.	Código de asignatura	GBD1233					
1.4.	Créditos	2,50					
1.5.	Nivel	Primero					
1.6.	Detalle de horas	ACD	60	AA	12	APE	48

Requisitos

Prerrequisitos		Correquisitos	
Asignatura	**Código**	**Asignatura**	**Código**
N/A	N/A	Metodología de Diseño de Software	GBD22312

Descripción de la asignatura

La asignatura Lógica de Sistemas está orientada al desarrollo del pensamiento lógico y estructurado para la resolución de problemas en el ámbito de la informática y la ingeniería de software. Proporciona los fundamentos teóricos y prácticos sobre lógica proposicional, lógica de predicados, algoritmos, estructuras de control y diagramación de procesos, permitiendo a los estudiantes comprender la base del diseño y desarrollo de sistemas computacionales. Además, se enfoca en la aplicación de técnicas de modelado lógico y razonamiento formal para optimizar la toma de decisiones en el análisis y diseño de sistemas.

Objetivos de la asignatura:

El objetivo principal de la asignatura es capacitar a los estudiantes en el uso de la lógica matemática y la representación simbólica para la resolución de problemas computacionales. Se busca que adquieran habilidades en la construcción de algoritmos eficientes, el diseño de diagramas de flujo y la implementación de estructuras lógicas en el desarrollo de software. Asimismo, se pretende fortalecer el

razonamiento analítico, promoviendo el uso de modelos lógicos para la optimización y estructuración de procesos en sistemas de información.

Introducción de la Asignatura

En el ámbito de la informática y la programación, la Lógica de Sistemas es un pilar fundamental para la estructuración del pensamiento computacional y la creación de soluciones eficientes. A lo largo de esta asignatura, los estudiantes explorarán principios de lógica matemática, estructuras de control y métodos para la representación de procesos mediante diagramas y algoritmos. Mediante el análisis y la resolución de problemas lógicos, los estudiantes desarrollarán habilidades clave para la programación, la inteligencia artificial y el diseño de sistemas, aplicando conocimientos esenciales para la automatización y la optimización de procesos computacionales.

I. DATOS GENERALES DE LA UNIDAD 1

Información General

1.1.	Nivel	Primero					
1.2.	Nombre Unidad	Programación Algorítmica					
1.3.	Tema 1 – Unidad	Conceptos Básicas					
1.4.	Tema 2 – Unidad	Funciones y operaciones					
1.5.	Unidad organizacional	Unidad Profesional					
1.6.	Unidad	1					
1.7.	Total, Horas Unidad	30					
1.8.	Detalle de horas Unidad	ACD	15	AA	3	APE	12

Resultado de Aprendizaje de Unidad

Al finalizar la unidad, el estudiante comprenderá los conceptos básicos de la programación, incluyendo la importancia de los algoritmos, el uso de variables y tipos de datos, y la representación gráfica de procesos mediante diagramas de flujo y pseudocódigo. Además, desarrollará habilidades en la implementación de funciones y el manejo de entradas y salidas de datos, aplicando operadores relacionales y lógicos para la resolución de problemas computacionales. A través de ejercicios prácticos, fortalecerá su capacidad de estructurar y diseñar soluciones algorítmicas eficientes.

Metodología

El aprendizaje se llevará a cabo mediante materiales interactivos, videos explicativos y guías de estudio sobre los fundamentos de la programación algorítmica. Se fomentará el uso de herramientas de simulación de pseudocódigo y diagramas de flujo, como PSeInt, Flowgorithm o Visual Paradigm, para facilitar la comprensión de estructuras lógicas. Se implementarán actividades prácticas progresivas, donde los estudiantes diseñarán algoritmos y resolverán ejercicios utilizando operadores relacionales y lógicos. También se promoverán foros de discusión y sesiones de resolución de problemas en entornos colaborativos. La evaluación se basará en el desarrollo de pequeños proyectos que permitan la aplicación de los conocimientos adquiridos en situaciones reales.

DESARROLLO

Unidad 1: Programación Algorítmica

Tema 1: Conceptos básicos

Objetivo: Con estos conocimientos, podrá aprender sobre los conceptos básicos de programación algorítmica, desarrollando habilidades para estructurar soluciones lógicas mediante algoritmos y fortalecer su pensamiento computacional, facilitando la resolución de problemas en entornos digitales dentro de su formación online.

La programación algorítmica es la base del desarrollo de software y consiste en la construcción de instrucciones lógicas y estructuradas que permiten la resolución de problemas computacionales. Para comprender su funcionamiento, es fundamental conocer los conceptos esenciales de la programación, el uso de algoritmos, las variables y los tipos de datos, así como la representación de los procesos mediante diagramas de flujo y pseudocódigo.

Conceptos básicos de programación

La programación es el proceso de diseñar y construir instrucciones que una computadora puede ejecutar para realizar tareas específicas. Se basa en la lógica y la estructuración de códigos mediante lenguajes de programación, los cuales permiten la comunicación entre el ser humano y la máquina.

Los principales elementos de la programación incluyen:

- **Sintaxis**: Conjunto de reglas que definen cómo se deben escribir las instrucciones en un lenguaje de programación.
- **Semántica**: Interpretación del significado de cada instrucción dentro de un programa.
- **Paradigmas de programación**: Métodos o enfoques utilizados para desarrollar software, entre los más comunes se encuentran la programación estructurada, orientada a objetos y funcional.
- **Lenguajes de programación**: Existen diversos lenguajes, como Python, Java, C++, JavaScript y PHP, cada uno con sus propias características y aplicaciones.

La programación permite la automatización de tareas, la creación de aplicaciones y el desarrollo de soluciones tecnológicas en múltiples áreas, como la inteligencia artificial, el análisis de datos y la ingeniería de software.

Algoritmos

Un algoritmo es un conjunto ordenado y finito de pasos o instrucciones diseñadas para resolver un problema o ejecutar una tarea específica. Se considera la base de la programación, ya que permite estructurar de manera lógica el desarrollo de software antes de implementarlo en un lenguaje de programación.

Los algoritmos tienen características esenciales, entre ellas:

- Precisión: Cada paso debe estar definido claramente.
- Orden lógico: Debe seguir una secuencia estructurada de ejecución.
- Finitud: Debe contar con un número determinado de pasos y llegar a un resultado.
- Entrada y salida: Recibe datos de entrada y produce un resultado de salida.

Ejemplo de un algoritmo simple para sumar dos números:

- Ingresar dos números.
- Sumar los números.
- Mostrar el resultado.

Los algoritmos pueden representarse en diferentes formas, como en lenguaje natural, diagramas de flujo y pseudocódigo, los cuales facilitan su comprensión antes de la implementación en código.

Variables y tipos de datos

Las variables son espacios en la memoria de la computadora que almacenan valores que pueden cambiar durante la ejecución de un programa. Cada variable tiene un nombre, un tipo de dato y un valor asignado.

Los tipos de datos definen la naturaleza de los valores que pueden almacenar las variables. Los más comunes son:

- Enteros (int): Números sin decimales (Ejemplo: 5, -12, 100).
- Decimales o flotantes (float, double): Números con decimales (Ejemplo: 3.14, -2.5, 0.001).

- Cadenas de texto (string): Secuencia de caracteres (Ejemplo: "Hola", "Programación").
- Booleanos (bool): Valores lógicos de verdadero o falso (Ejemplo: True, False).

Ejemplo de declaración de variables en Python:

- edad = 25 # Variable entera
- precio = 9.99 # Variable flotante
- nombre = "Carlos" # Variable de tipo cadena
- es_mayor = True # Variable booleana

Las variables permiten almacenar y manipular información dentro de un programa, facilitando la resolución de problemas mediante cálculos, comparaciones y estructuras de control.

Diagramas de Flujo y Pseudocódigo

Para representar algoritmos de manera visual o textual, se utilizan herramientas como los diagramas de flujo y el pseudocódigo.

Un diagrama de flujo es una representación gráfica de un algoritmo mediante símbolos estandarizados. Sus principales elementos incluyen:

- **Óvalo**: Indica el inicio o fin del diagrama.
- **Paralelogramo**: Representa entradas o salidas de datos.
- **Rectángulo**: Expresa una operación o proceso.
- **Rombo**: Representa una condición o decisión.
- **Flechas**: Indican el flujo de ejecución del algoritmo.

Ejemplo de diagrama de flujo para sumar dos números:

1. Inicio
2. Ingresar número 1
3. Ingresar número 2
4. Sumar los números

5. Mostrar resultado
6. Fin

El pseudocódigo, por otro lado, es una forma de escribir algoritmos utilizando instrucciones en lenguaje natural estructurado. Se asemeja a la lógica de la programación, pero sin la sintaxis de un lenguaje específico, lo que facilita la comprensión del proceso antes de codificarlo.

Ejemplo de pseudocódigo para sumar dos números:

- Inicio
    - Leer número1
    - Leer número2
    - suma ← número1 + número2
    - Escribir "El resultado es:", suma
- Fin

El uso de diagramas de flujo y pseudocódigo es esencial en la programación, ya que permite estructurar la lógica de los algoritmos de manera clara y organizada antes de implementarlos en un lenguaje de programación.

La programación algorítmica es el fundamento de la informática y el desarrollo de software. Comprender los conceptos básicos de la programación, el uso de algoritmos, las variables y los tipos de datos permite estructurar soluciones eficientes a problemas computacionales. La representación visual mediante diagramas de flujo y pseudocódigo facilita la planificación de los programas antes de su codificación. Dominar estos conceptos es esencial para cualquier persona que desee incursionar en el mundo de la programación y la automatización de procesos computacionales.

ACTIVIDADES PRÁCTICAS

Tabla

Detalle de la Práctica

Practicas	Instrucciones	Duración de la Practica	Mecanismo de Evaluación
Foro de debate	Revisar Actividades Prácticas	0:30 mm	Por rúbricas
Práctica investiga: Taller/Ensayo/Investigación	Revisar Actividades Prácticas	3:30 h	Por rúbricas
Tutorías Prácticas	Revisar Actividades Prácticas	2:00 mm	Por participación en clase
Test evaluativo	Revisar Actividades Prácticas	0:30 mm	Formativa

Nota. Elaboración propia

Consideraciones Generales

1. Lee y revisa la Guía de Estudio.
2. Observa el video de la clase magistral.
3. Participa activamente en el FORO DEBATE.
4. Revisa en el calendario el día y hora en que se realizará las actividades prácticas y la tutoría.
5. Realizar la práctica propuesta por el docente.
6. Revisa las Rúbricas acorde a la actividad prácticas.
7. Toma los principales apuntes que vayas rescatando del texto.
8. Reflexiona sobre los temas tratados en la clase.
9. Sube el documento en formato PDF en la plataforma Moodle.
10. Revisa la calificación de tus actividades prácticas.

FORO DE DEBATE

Tema: La programación algorítmica es la base del desarrollo de software, ya que permite estructurar soluciones a problemas mediante algoritmos, variables y tipos de datos.

- ✓ ¿Por qué crees que es importante aprender a diseñar algoritmos antes de escribir código en un lenguaje de programación?
- ✓ Explica cómo los diagramas de flujo y pseudocódigo ayudan a organizar el pensamiento lógico antes de programar.

Instrucciones para Participar:

- Exponga su punto de vista acerca del tema.
- Comenta las respuestas de otros participantes.
- Promueve un ambiente respetuoso y constructivo.

PRÁCTICA INVESTIGATIVA

Ensayo: Conceptos Básicos de Programación Algorítmica.

- ✓ Escribe un ensayo de 1 a 2 páginas en el que expliques:
- ✓ ¿Qué es la programación algorítmica y por qué es fundamental en la informática?
- ✓ ¿Qué son los algoritmos y cómo ayudan en la resolución de problemas?
- ✓ Importancia de las variables y los tipos de datos en la programación.
- ✓ Cómo los diagramas de flujo y el pseudocódigo facilitan la comprensión de un algoritmo antes de su implementación.

Consigna:

- El trabajo de investigación debe presentarse de manera estructurada y coherente para comunicar efectivamente los hallazgos y conclusiones.
- Asegúrese de que el estilo y formato del trabajo sigan las pautas de citación y presentación establecidas por la institución.
- Utilice un lenguaje claro y evite jergas o términos técnicos innecesarios.
- Revise y corrija la redacción y la gramática antes de la presentación final (ver anexo).

PRÁCTICA TEST

Antes de realizar el test, es importante considerar varios aspectos para aprovechar al máximo la experiencia.

Consideraciones para realizar el test

1. Lee y revisa el documento.
2. Observa el video de la clase magistral.
3. Revisa en el calendario el día y hora en que se debe de realizar el Test.
4. Realiza el Test de la unidad considerando lo Siguiente:

- No cambiarte entre paginas durante la realización del Test.
- Revisa que tu conexión a internet sea estable.
- Recuerda que todo el contenido está inmerso dentro del documento base.

5. Revisa tu puntuación, de inmediato al finalizar de la actividad.

Tema 2: Funciones y operaciones

Objetivo: Con estos conocimientos, podrá aprender sobre el uso de funciones y operaciones en programación algorítmica, comprendiendo su estructura, utilidad y aplicación para la creación de soluciones eficientes, optimizando el desarrollo de algoritmos en entornos digitales dentro de su formación online.

En programación, las funciones son bloques de código que ejecutan una tarea específica y permiten reutilizar código para mejorar la eficiencia y organización de los programas. Además, los operadores relacionales y lógicos permiten realizar comparaciones y tomar decisiones dentro del flujo de un programa. Comprender estos conceptos es fundamental para la construcción de algoritmos más avanzados y dinámicos.

Funciones Entradas y Salidas de Datos

Las funciones en programación son secciones de código que reciben datos de entrada, realizan operaciones con ellos y devuelven un resultado como salida. Su propósito es modularizar el código, facilitando la reutilización y el mantenimiento del programa.

Una función consta de tres partes principales:

- Definición: Se especifica el nombre de la función y los parámetros que recibe.
- Ejecución de instrucciones: Se realiza el procesamiento de los datos.
- Retorno de un resultado: La función devuelve un valor o simplemente ejecuta una acción.

Ejemplo de función en Python que suma dos números:

```
def sumar(a, b):
    resultado = a + b
    return resultado


numero1 = 5
numero2 = 10
print("La suma es:", sumar(numero1, numero2))
```

En este caso, la función sumar recibe dos entradas (a y b), realiza la operación de suma y devuelve el resultado.

Las funciones pueden clasificarse en:

- **Funciones con retorno**: Devuelven un valor con la instrucción return.
- **Funciones sin retorno**: Ejecutan una tarea pero no devuelven datos.
- **Funciones con parámetros**: Reciben valores de entrada para ser procesados.
- **Funciones sin parámetros**: No requieren datos de entrada y ejecutan instrucciones independientes.

Las entradas de datos en las funciones se realizan mediante parámetros o directamente con funciones como input() en Python, que permiten capturar información del usuario.

Ejemplo de función con entrada de datos desde el usuario:

```
def saludar(nombre):
    print("Hola,", nombre, "¡Bienvenido!")


usuario = input("Ingresa tu nombre: ")
saludar(usuario)
```

En este caso, la función saludar ( ) recibe el nombre del usuario y lo imprime en pantalla, demostrando cómo una función puede interactuar con datos de entrada y salida.

Los operadores permiten realizar comparaciones y evaluar condiciones en un programa, facilitando la toma de decisiones mediante estructuras de control como condicionales (if, else, elif).

Operadores Relacionales

Los operadores relacionales comparan valores y devuelven un resultado booleano (True o False):

- == → Igualdad (Ejemplo: 5 == 5 es True)

- != → Diferente (Ejemplo: 5 != 3 es True)
- > → Mayor que (Ejemplo: 10 > 5 es True)
- < → Menor que (Ejemplo: 3 < 8 es True)
- >= → Mayor o igual que (Ejemplo: 5 >= 5 es True)
- <= → Menor o igual que (Ejemplo: 2 <= 4 es True)

Ejemplo de uso en una condición:

```
edad = int(input("Ingresa tu edad: "))

if edad >= 18:

    print("Eres mayor de edad.")

else:

    print("Eres menor de edad.")
```

En este caso, el operador >= compara la edad ingresada por el usuario para determinar si es mayor de edad o no.

Operadores Relacionales y Lógicos

Los operadores lógicos permiten combinar condiciones en una sola expresión:

- **and (y lógico)**: Retorna True si ambas condiciones son verdaderas.
- **or (o lógico)**: Retorna True si al menos una condición es verdadera.
- **not (negación lógica)**: Invierte el valor de la condición.

Ejemplo con operadores lógicos:

```
usuario = "admin"

clave = "1234"

if usuario == "admin" and clave == "1234":

    print("Acceso permitido")

else:

    print("Acceso denegado")
```

En este caso, se utiliza el operador and para verificar que el usuario y la clave sean correctos antes de conceder acceso.

Otro ejemplo con or:

```
dia = "sábado"

if dia == "sábado" or dia == "domingo":

    print("Es fin de semana.")

else:

    print("Es un día laboral.")
```

Aquí, el programa evalúa si el día ingresado es un fin de semana utilizando el operador or.

El operador not puede utilizarse para negar condiciones:

```
llueve = False

if not llueve:
    print("Puedes salir sin paraguas.")
```

En este caso, la condición not llueve evalúa si no está lloviendo para mostrar el mensaje correspondiente.

Las funciones permiten estructurar el código de manera eficiente, facilitando la reutilización y la organización del programa. A través de las entradas y salidas de datos, se puede interactuar con el usuario y realizar operaciones dinámicas. Los operadores relacionales y lógicos son esenciales en la toma de decisiones dentro de los programas, permitiendo la comparación de valores y la evaluación de condiciones complejas. Con la práctica de estos conceptos, se sientan las bases para desarrollar programas más avanzados y funcionales en cualquier lenguaje de programación.

ACTIVIDADES PRÁCTICAS

Tabla

Detalle de la Práctica

Practicas	Instrucciones	Duración de la Practica	Mecanismo de Evaluación
Foro de debate	Revisar Actividades Prácticas	0:30 mm	Por rúbricas
Práctica investiga: Taller/Ensayo/Investigación	Revisar Actividades Prácticas	3:30 h	Por rúbricas
Tutorías Prácticas	Revisar Actividades Prácticas	2:00 mm	Por participación en clase
Test evaluativo	Revisar Actividades Prácticas	0:30 mm	Formativa

Nota. Elaboración propia

Consideraciones Generales

1. Lee y revisa la Guía de Estudio.
2. Observa el video de la clase magistral.
3. Participa activamente en el FORO DEBATE.
4. Revisa en el calendario el día y hora en que se realizará las actividades prácticas y la tutoría.
5. Realizar la práctica propuesta por el docente.
6. Revisa las Rúbricas acorde a la actividad prácticas.
7. Toma los principales apuntes que vayas rescatando del texto.
8. Reflexiona sobre los temas tratados en la clase.
9. Sube el documento en formato PDF en la plataforma Moodle.
10. Revisa la calificación de tus actividades prácticas.

FORO DE DEBATE

Tema: En la programación, las funciones permiten organizar el código de manera modular, facilitando su reutilización y mantenimiento. Además, el uso de operadores relacionales y lógicos es esencial para la toma de decisiones en los programas.

- ✓ ¿Por qué crees que las funciones y operadores son fundamentales en la resolución de problemas mediante la programación?
- ✓ Proporciona un ejemplo de una función que utilice operadores lógicos y relacionales y explica su utilidad

Instrucciones para Participar:

- Exponga su punto de vista acerca del tema.
- Comenta las respuestas de otros participantes.
- Promueve un ambiente respetuoso y constructivo.

PRÁCTICA INVESTIGATIVA

Taller: Desarrollo de Ejercicios Prácticos sobre Funciones y Operadores.

Implementa tres ejercicios en pseudocódigo o en un lenguaje de programación que incluyan:

- ✓ Una función con entrada y salida de datos.
- ✓ Uso de operadores relacionales y lógicos en una toma de decisión.
- ✓ Explicación de cómo funciona cada parte del código.

Consigna:

- El trabajo de investigación debe presentarse de manera estructurada y coherente para comunicar efectivamente los hallazgos y conclusiones.
- Asegúrese de que el estilo y formato del trabajo sigan las pautas de citación y presentación establecidas por la institución.
- Utilice un lenguaje claro y evite jergas o términos técnicos innecesarios.
- Revise y corrija la redacción y la gramática antes de la presentación final (ver anexo).

PRÁCTICA TEST

Antes de realizar el test, es importante considerar varios aspectos para aprovechar al máximo la experiencia.

Consideraciones para realizar el test

1. Lee y revisa el documento.
2. Observa el video de la clase magistral.
3. Revisa en el calendario el día y hora en que se debe de realizar el Test.
4. Realiza el Test de la unidad considerando lo Siguiente:

- No cambiarte entre paginas durante la realización del Test.
- Revisa que tu conexión a internet sea estable.
- Recuerda que todo el contenido está inmerso dentro del documento base.

5. Revisa tu puntuación, de inmediato al finalizar de la actividad.

II. DATOS GENERALES DE LA UNIDAD 2

Información General

1.1.	Nivel	Primero					
1.2.	Nombre Unidad	Estructuras de control					
1.3.	Tema 1 – Unidad	Estructuras					
1.4.	Tema 2 – Unidad	Bucles					
1.5.	Unidad organizacional	Unidad Profesional					
1.6.	Unidad	2					
1.7.	Total, Horas Unidad	30					
1.8.	Detalle de horas Unidad	ACD	15	AA	3	APE	12

Resultado de Aprendizaje de Unidad

Al finalizar la unidad, el estudiante comprenderá el funcionamiento de las estructuras de control en la programación, diferenciando entre estructuras de selección y estructuras de repetición. Aprenderá a aplicar estructuras condicionales (simples, dobles, anidadas y múltiples) para la toma de decisiones en un programa. Asimismo, identificará el uso de bucles y sus partes, implementando estructuras de repetición para optimizar el procesamiento de datos y la automatización de tareas. Mediante la resolución de ejercicios prácticos, fortalecerá sus habilidades en la construcción de programas eficientes y bien estructurados.

Metodología

El aprendizaje se desarrollará a través de lecturas interactivas, videos explicativos y diagramas ilustrativos sobre estructuras de control. Se fomentará la implementación de código en plataformas como PSeInt, Python o C, permitiendo al estudiante aplicar de manera práctica los conceptos aprendidos. Se realizarán ejercicios guiados sobre la creación de programas que utilicen estructuras condicionales y bucles para la resolución de problemas computacionales. Además, se promoverá la interacción en foros de discusión y la realización de desafíos de programación en línea para reforzar la comprensión y la aplicación de los temas. La evaluación se basará en proyectos pequeños donde los estudiantes diseñarán y ejecutarán algoritmos que incorporen estructuras de control adecuadas.

DESARROLLO

Unidad 2: Estructuras de control

Tema 1: Estructuras

Objetivo: Con estos conocimientos, podrá aprender sobre las estructuras de control de selección, incluyendo estructuras simples, dobles, anidadas y múltiples, comprendiendo su lógica y aplicación en la programación algorítmica para la toma de decisiones dentro de los algoritmos, optimizando la resolución de problemas en entornos digitales dentro de su formación online.

Las estructuras de control en programación permiten definir el flujo de ejecución de un programa, determinando qué instrucciones se ejecutan en función de ciertas condiciones. Estas estructuras facilitan la toma de decisiones y el control del flujo lógico de los algoritmos, lo que permite crear programas dinámicos y eficientes. Dentro de ellas, las estructuras de selección son fundamentales para evaluar condiciones y ejecutar diferentes acciones según los valores de entrada.

Estructuras de selección

Las estructuras de selección permiten que un programa tome decisiones basadas en comparaciones lógicas. Dependiendo de si una condición se cumple o no, el flujo del programa se bifurca para ejecutar diferentes instrucciones. Estas estructuras incluyen la estructura simple, doble, anidada y múltiple, cada una con un uso específico en la resolución de problemas computacionales.

Estructura simple

La estructura de selección simple permite ejecutar una acción si se cumple una condición determinada; de lo contrario, el programa continúa su ejecución sin realizar ninguna acción alternativa. Se representa con la palabra clave if.

Sintaxis en Python:

python

CopiarEditar

if condición:

```
    instrucción
```

Si la condición es verdadera (True), se ejecuta la instrucción dentro del bloque if. Si es falsa (False), el programa sigue su curso sin ejecutar dicho bloque.

Ejemplo: Determinar si un número ingresado por el usuario es positivo.

python

CopiarEditar

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

if numero > 0:

    print("El número es positivo.")
```

En este caso, si el número es mayor que cero, se imprime el mensaje. Si no, el programa simplemente continúa sin hacer nada.

Estructura doble

La estructura de selección doble permite ejecutar una acción si la condición es verdadera y otra acción alternativa si la condición es falsa. Se utiliza la instrucción if-else.

Sintaxis en Python:

python

CopiarEditar

```
if condición:

    instrucción_1

else:

    instrucción_2
```

Si la condición se cumple, se ejecuta la instrucción_1. Si no se cumple, se ejecuta la instrucción_2.

Ejemplo: Determinar si un número ingresado es par o impar.

python

CopiarEditar

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

if numero % 2 == 0:

    print("El número es par.")

else:

    print("El número es impar.")
```

Aquí, si el número es divisible por 2, se imprime que es par; de lo contrario, se indica que es impar.

Estructura anidada y múltiple

Estructura Anidada

La estructura de selección anidada consiste en colocar una instrucción if-else dentro de otra, permitiendo evaluar múltiples condiciones en diferentes niveles. Se utiliza cuando una decisión depende de otra decisión previa.

Sintaxis en Python:

python

CopiarEditar

```
if condición_1:

    if condición_2:

        instrucción_1

    else:

        instrucción_2

else:
```

```
    instrucción_3
```

Esto significa que si la primera condición es verdadera, se evalúa una segunda condición dentro de ella. Si la primera condición es falsa, se ejecuta la alternativa del else.

Ejemplo: Determinar si un número es positivo, negativo o cero.

python

CopiarEditar

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

if numero > 0:

    print("El número es positivo.")

else:

    if numero < 0:

        print("El número es negativo.")

    else:

        print("El número es cero.")
```

Aquí, primero se verifica si el número es mayor que cero. Si no lo es, se evalúa si es menor que cero o igual a cero.

Estructura Múltiple

La estructura de selección múltiple permite evaluar varias condiciones y ejecutar diferentes instrucciones según la que se cumpla. En Python, se implementa con if-elif-else.

Sintaxis en Python:

python

CopiarEditar

```
if condición_1:
```

```
    instrucción_1
elif condición_2:
    instrucción_2
elif condición_3:
    instrucción_3
else:
    instrucción_n
```

Se evalúan las condiciones en orden, ejecutando la primera que se cumpla. Si ninguna se cumple, se ejecuta el bloque else.

Ejemplo: Asignar una calificación según la nota de un estudiante.

python

CopiarEditar

```
nota = float(input("Ingrese la nota del estudiante: "))

if nota >= 90:
    print("Calificación: A")
elif nota >= 80:
    print("Calificación: B")
elif nota >= 70:
    print("Calificación: C")
elif nota >= 60:
    print("Calificación: D")
else:
```

```
    print("Calificación: F")
```

En este caso, el programa evalúa cada rango de notas y asigna una calificación según corresponda.

Las estructuras de selección son esenciales en la programación para la toma de decisiones en los algoritmos. La estructura simple evalúa una condición y ejecuta una acción si se cumple. La estructura doble permite definir una alternativa en caso de que la condición no se cumpla. La estructura anidada permite evaluar condiciones dentro de otras condiciones, y la estructura múltiple permite evaluar varias posibilidades en secuencia. Estas estructuras proporcionan la flexibilidad necesaria para desarrollar programas dinámicos y adaptables a diferentes escenarios.

ACTIVIDADES PRÁCTICAS

Tabla

Detalle de la Práctica

Practicas	Instrucciones	Duración de la Practica	Mecanismo de Evaluación
Foro de debate	Revisar Actividades Prácticas	0:30 mm	Por rúbricas
Práctica investiga: Taller/Ensayo/Investigación	Revisar Actividades Prácticas	3:30 h	Por rúbricas
Tutorías Prácticas	Revisar Actividades Prácticas	2:00 mm	Por participación en clase
Test evaluativo	Revisar Actividades Prácticas	0:30 mm	Formativa

Nota. Elaboración propia

Consideraciones Generales

1. Lee y revisa la Guía de Estudio.
2. Observa el video de la clase magistral.
3. Participa activamente en el FORO DEBATE.
4. Revisa en el calendario el día y hora en que se realizará las actividades prácticas y la tutoría.
5. Realizar la práctica propuesta por el docente.
6. Revisa las Rúbricas acorde a la actividad prácticas.
7. Toma los principales apuntes que vayas rescatando del texto.
8. Reflexiona sobre los temas tratados en la clase.
9. Sube el documento en formato PDF en la plataforma Moodle.
10. Revisa la calificación de tus actividades prácticas.

FORO DE DEBATE

Tema: Las estructuras de selección permiten que un programa tome decisiones basadas en condiciones establecidas.

- ✓ ¿Por qué crees que las estructuras de selección son fundamentales en la programación? Explica la diferencia entre una estructura simple, doble, anidada y múltiple, y proporciona un ejemplo de su uso en un problema cotidiano.

Instrucciones para Participar:

- Exponga su punto de vista acerca del tema.
- Comenta las respuestas de otros participantes.
- Promueve un ambiente respetuoso y constructivo.

PRÁCTICA INVESTIGATIVA

Taller: Comparación de Estructuras de Selección en Diferentes Lenguajes de Programación.

- ✓ Investiga cómo se representan las estructuras de selección en al menos tres lenguajes de programación (Python, Java, C++, JavaScript, etc.).
- ✓ Explica en qué se diferencian en cuanto a sintaxis y funcionalidad.
- ✓ Presenta una tabla comparativa con ejemplos en cada lenguaje.
- ✓ Entrega en Word o PDF, incluyendo fragmentos de código y gráficos explicativos.

Consigna:

- El trabajo de investigación debe presentarse de manera estructurada y coherente para comunicar efectivamente los hallazgos y conclusiones.
- Asegúrese de que el estilo y formato del trabajo sigan las pautas de citación y presentación establecidas por la institución.
- Utilice un lenguaje claro y evite jergas o términos técnicos innecesarios.
- Revise y corrija la redacción y la gramática antes de la presentación final (ver anexo).

PRÁCTICA TEST

Antes de realizar el test, es importante considerar varios aspectos para aprovechar al máximo la experiencia.

Consideraciones para realizar el test

1. Lee y revisa el documento.
2. Observa el video de la clase magistral.
3. Revisa en el calendario el día y hora en que se debe de realizar el Test.
4. Realiza el Test de la unidad considerando lo Siguiente:

- No cambiarte entre paginas durante la realización del Test.
- Revisa que tu conexión a internet sea estable.
- Recuerda que todo el contenido está inmerso dentro del documento base.

5. Revisa tu puntuación, de inmediato al finalizar de la actividad.

Tema 2: Bucles

Objetivo: Con estos conocimientos, podrá aprender sobre las estructuras de repetición y el funcionamiento de los bucles, comprendiendo sus partes y aplicándolos en la programación algorítmica para la ejecución eficiente de tareas repetitivas. A través de ejercicios prácticos, desarrollará habilidades para optimizar procesos y mejorar la lógica de programación en su formación online.

Las estructuras de repetición, conocidas como bucles o ciclos, permiten ejecutar un conjunto de instrucciones varias veces de manera controlada. Estas estructuras son fundamentales en la programación, ya que optimizan el código al evitar la repetición manual de instrucciones y permiten realizar tareas automatizadas, como cálculos iterativos, procesamiento de datos y generación de secuencias.

Estructuras de repetición

Las estructuras de repetición se utilizan cuando es necesario ejecutar un bloque de código varias veces. Su funcionamiento se basa en una **condición de control**, la cual determina cuándo debe continuar el ciclo y cuándo debe finalizar.

Existen dos tipos principales de estructuras de repetición:

1. **Bucles con condición**: Se ejecutan mientras una condición sea verdadera. (Ejemplo: while)
2. **Bucles controlados por un contador**: Se ejecutan un número determinado de veces. (Ejemplo: for)

Bucles y sus partes

Cada bucle tiene tres elementos esenciales:

- Inicialización: Se establece el valor inicial de la variable de control.
- Condición: Se evalúa antes de cada iteración; si es verdadera, el bucle continúa, si es falsa, el bucle finaliza.
- Incremento o decremento: Se actualiza la variable de control en cada iteración para evitar bucles infinitos.

Bucle while (Mientras)

El bucle while ejecuta un bloque de código mientras la condición sea verdadera. Se usa cuando no se conoce de antemano cuántas veces se repetirá el ciclo.

Sintaxis:

```
while condición:
    # Código a ejecutar en cada iteración
```

Ejemplo: Contar del 1 al 5 utilizando while.

```
contador = 1

while contador <= 5:
    print("Número:", contador)
    contador += 1  # Incremento para evitar bucle infinito
```

En este caso, el ciclo se ejecuta mientras contador sea menor o igual a 5. En cada iteración, se imprime el número y se incrementa el contador.

Bucle for (Para)

El bucle for se usa cuando se conoce el número exacto de repeticiones. En Python, se utiliza junto con la función range() para generar secuencias de números.

Sintaxis:

Ejemplo: Imprimir los números del 1 al 5 utilizando for.

python

CopiarEditar

```
for i in range(1, 6):
    print("Número:", i)
```

En este caso, la variable i toma los valores del 1 al 5, ya que range(1, 6) genera números desde 1 hasta 5 (sin incluir 6).

Diferencia entre while y for

Característica	while	for
Uso principal	Cuando no se conoce el número exacto de iteraciones	Cuando se conoce el número de repeticiones
Condición	Se evalúa antes de cada iteración	Se establece un rango de valores
Control	Se maneja manualmente con una variable de control	Se usa un iterador automático

Los bucles son estructuras esenciales en programación, ya que permiten repetir acciones sin necesidad de escribir múltiples líneas de código. El bucle while se usa cuando no se conoce el número exacto de iteraciones y depende de una condición, mientras que el bucle for se emplea cuando se conoce la cantidad de repeticiones y se usa con rangos o secuencias. Comprender cómo funcionan estos bucles permite desarrollar programas más eficientes y dinámicos, optimizando procesos y automatizando tareas repetitivas.

ACTIVIDADES PRÁCTICAS

Tabla

Detalle de la Práctica

Practicas	Instrucciones	Duración de la Practica	Mecanismo de Evaluación
Foro de debate	Revisar Actividades Prácticas	0:30 mm	Por rúbricas
Práctica investiga: Taller/Ensayo/Investigación	Revisar Actividades Prácticas	3:30 h	Por rúbricas
Tutorías Prácticas	Revisar Actividades Prácticas	2:00 mm	Por participación en clase
Test evaluativo	Revisar Actividades Prácticas	0:30 mm	Formativa

Nota. Elaboración propia

Consideraciones Generales

1. Lee y revisa la Guía de Estudio.
2. Observa el video de la clase magistral.
3. Participa activamente en el FORO DEBATE.
4. Revisa en el calendario el día y hora en que se realizará las actividades prácticas y la tutoría.
5. Realizar la práctica propuesta por el docente.
6. Revisa las Rúbricas acorde a la actividad prácticas.
7. Toma los principales apuntes que vayas rescatando del texto.
8. Reflexiona sobre los temas tratados en la clase.
9. Sube el documento en formato PDF en la plataforma Moodle.
10. Revisa la calificación de tus actividades prácticas.

FORO DE DEBATE

Tema: Las estructuras de repetición (bucles) permiten ejecutar un conjunto de instrucciones varias veces hasta que se cumpla una condición.

- ✓ ¿Por qué crees que los bucles son esenciales en la programación?
- ✓ Explica la diferencia entre los bucles "for", "while" y "do-while" y menciona un ejemplo práctico de cómo se pueden utilizar para automatizar tareas.

Instrucciones para Participar:

- Exponga su punto de vista acerca del tema.
- Comenta las respuestas de otros participantes.
- Promueve un ambiente respetuoso y constructivo.

PRÁCTICA INVESTIGATIVA

Taller: Desarrollo de Ejercicios Prácticos sobre Bucles.

Implementa tres ejercicios en pseudocódigo o en un lenguaje de programación donde se apliquen:

- ✓ Bucle "for" (ejemplo: imprimir los números del 1 al 10).
- ✓ Bucle "while" (ejemplo: pedir al usuario una contraseña hasta que ingrese la correcta).
- ✓ Bucle "do-while" (ejemplo: validar la entrada de un número dentro de un rango permitido).
- ✓ Explica cómo funciona cada parte del código y justifica su uso.

Consigna:

- El trabajo de investigación debe presentarse de manera estructurada y coherente para comunicar efectivamente los hallazgos y conclusiones.
- Asegúrese de que el estilo y formato del trabajo sigan las pautas de citación y presentación establecidas por la institución.
- Utilice un lenguaje claro y evite jergas o términos técnicos innecesarios.
- Revise y corrija la redacción y la gramática antes de la presentación final (ver anexo).

PRÁCTICA TEST

Antes de realizar el test, es importante considerar varios aspectos para aprovechar al máximo la experiencia.

Consideraciones para realizar el test

1. Lee y revisa el documento.
2. Observa el video de la clase magistral.
3. Revisa en el calendario el día y hora en que se debe de realizar el Test.
4. Realiza el Test de la unidad considerando lo Siguiente:

- No cambiarte entre paginas durante la realización del Test.
- Revisa que tu conexión a internet sea estable.
- Recuerda que todo el contenido está inmerso dentro del documento base.

5. Revisa tu puntuación, de inmediato al finalizar de la actividad.

III. DATOS GENERALES DE LA UNIDAD 3

Información General

1.1.	Nivel	Primero					
1.2.	Nombre Unidad	Arreglos de programación					
1.3.	Tema 1 – Unidad	Arreglos					
1.4.	Tema 2 – Unidad	Tipos de arreglos					
1.5.	Unidad organizacional	Unidad Profesional					
1.6.	Unidad	3					
1.7.	Total, Horas Unidad	30					
1.8.	Detalle de horas Unidad	ACD	15	AA	3	APE	12

Resultado de Aprendizaje de Unidad

Al finalizar la unidad, el estudiante comprenderá el concepto y uso de los arreglos en programación, identificando su importancia para almacenar y manipular grandes volúmenes de datos de manera estructurada. Aprenderá a declarar y utilizar arreglos, accediendo a sus elementos y gestionando la cantidad de información dentro de ellos. Además, analizará los tipos de arreglos en distintos lenguajes de programación, reconociendo que, aunque la sintaxis puede variar, la lógica detrás de su funcionamiento es similar. Finalmente, aplicará funciones matemáticas en el manejo de arreglos para la resolución de problemas computacionales.

Metodología

El aprendizaje se desarrollará a través de recursos interactivos, videos explicativos y ejemplos prácticos en lenguajes como Python, Java y C++, permitiendo la comparación de la sintaxis de los arreglos en distintos entornos. Se realizarán ejercicios prácticos de declaración, inicialización y manipulación de arreglos, junto con el uso de funciones matemáticas para procesar datos dentro de ellos. Se fomentará la participación en foros de discusión para compartir experiencias y resolver dudas en la implementación de arreglos en diferentes lenguajes. Además, se utilizarán entornos de programación en línea, como Replit o Jupyter Notebook, para realizar pruebas y evaluaciones formativas mediante pequeños proyectos de aplicación.

DESARROLLO

Unidad 3: Arreglos de programación

Tema 1: Arreglos

Objetivo: Con estos conocimientos, podrá aprender sobre el uso de arreglos en programación, comprendiendo su estructura, la forma de declarar un array, la manipulación de sus elementos y la gestión de la cantidad de información almacenada. Esto le permitirá optimizar la organización y procesamiento de datos en sus programas dentro de su formación online.

Los arreglos son estructuras de datos fundamentales en programación que permiten almacenar y organizar múltiples valores bajo un mismo nombre de variable. Son especialmente útiles cuando se necesita manejar grandes volúmenes de datos de manera eficiente, facilitando el acceso y manipulación de la información.

Arreglos en programación

Un **arreglo (array)** es una estructura de datos que almacena múltiples elementos del mismo tipo en una secuencia ordenada. A diferencia de las variables individuales, que solo pueden contener un único valor, un arreglo permite gestionar un conjunto de valores dentro de una misma variable.

Las características principales de los arreglos son:

- **Índice**: Cada elemento dentro del arreglo tiene una posición única, comenzando desde el índice 0.
- **Tamaño fijo o dinámico**: Dependiendo del lenguaje de programación, los arreglos pueden ser de tamaño fijo (como en C) o dinámico (como en Python).
- **Acceso rápido**: Se puede acceder a cualquier elemento utilizando su índice.
- **Almacenamiento homogéneo**: En la mayoría de los lenguajes, todos los elementos de un arreglo deben ser del mismo tipo de dato.

Ejemplo de un **arreglo de números enteros** en Python:

python

CopiarEditar

```
numeros = [10, 20, 30, 40, 50]

print(numeros)  # Salida: [10, 20, 30, 40, 50]
```

Declarar un array

La forma de declarar un arreglo varía según el lenguaje de programación. En algunos lenguajes, es necesario especificar el tamaño y tipo de datos, mientras que en otros, se permite la creación de listas dinámicas.

En Python (listas dinámicas):

python

CopiarEditar

```
numeros = [1, 2, 3, 4, 5]  # Lista de enteros

nombres = ["Ana", "Carlos", "Elena"]  # Lista de cadenas
```

En Java (arreglo de tamaño fijo):

java

CopiarEditar

```
int[] numeros = new int[5];  // Arreglo de enteros con 5 espacios
```

En C (arreglo estático):

c

CopiarEditar

```
int numeros[5] = {1, 2, 3, 4, 5};  // Arreglo de tamaño fijo con valores predefinidos
```

Elemento del arreglo

Cada elemento dentro de un arreglo tiene una posición única, identificada por su índice. En la mayoría de los lenguajes, los índices comienzan desde 0.

Para acceder a un elemento específico, se usa su índice:

python

CopiarEditar

```
numeros = [10, 20, 30, 40, 50]

print(numeros[0])  # Salida: 10

print(numeros[3])  # Salida: 40
```

En este ejemplo, numeros[0] accede al primer elemento (10) y numeros[3] al cuarto elemento (40).

También es posible modificar un elemento de un arreglo:

python

CopiarEditar

```
numeros[2] = 100  # Cambia el tercer elemento a 100

print(numeros)  # Salida: [10, 20, 100, 40, 50]
```

Si se intenta acceder a un índice fuera del rango del arreglo, se generará un error:

python

CopiarEditar

```
print(numeros[10])  # Error: Índice fuera de rango
```

Cantidad de información en arreglos

Para conocer la cantidad de elementos almacenados en un arreglo, se pueden utilizar funciones o métodos específicos según el lenguaje:

En Python (usando len()):

python

CopiarEditar

```
numeros = [10, 20, 30, 40, 50]

print(len(numeros))  # Salida: 5
```

En Java (usando length):

java

CopiarEditar

```
int[] numeros = {1, 2, 3, 4, 5};

System.out.println(numeros.length);  // Salida: 5
```

En C (tamaño del arreglo dividiendo su memoria total por el tamaño de un elemento):

c

CopiarEditar

```
int numeros[] = {1, 2, 3, 4, 5};

int size = sizeof(numeros) / sizeof(numeros[0]);

printf("%d", size);  // Salida: 5
```

Conocer el tamaño de un arreglo es útil cuando se desea recorrerlo en un bucle, ya que evita errores por acceso a índices inexistentes.

Ejemplo de recorrido de un arreglo con un bucle for en Python:

python

CopiarEditar

```
numeros = [10, 20, 30, 40, 50]

for i in range(len(numeros)):
```

```
print(f"Índice {i}: {numeros[i]}")
```

Salida:

yaml

CopiarEditar

Índice 0: 10

Índice 1: 20

Índice 2: 30

Índice 3: 40

Índice 4: 50

Los arreglos son una estructura fundamental en programación que permite almacenar y manipular múltiples datos de manera eficiente. Su uso facilita la organización y procesamiento de información, permitiendo acceder, modificar y recorrer sus elementos de manera sencilla. Comprender cómo declarar arreglos, acceder a sus elementos y determinar su tamaño es esencial para el desarrollo de programas más complejos y estructurados. Su aplicación se extiende a diversos campos, desde el análisis de datos hasta la inteligencia artificial y la optimización de algoritmos.

ACTIVIDADES PRÁCTICAS

Tabla

Detalle de la Práctica

Practicas	Instrucciones	Duración de la Practica	Mecanismo de Evaluación
Foro de debate	Revisar Actividades Prácticas	0:30 mm	Por rúbricas
Práctica investiga: Taller/Ensayo/Investigación	Revisar Actividades Prácticas	3:30 h	Por rúbricas
Tutorías Prácticas	Revisar Actividades Prácticas	2:00 mm	Por participación en clase
Test evaluativo	Revisar Actividades Prácticas	0:30 mm	Formativa

Nota. Elaboración propia

Consideraciones Generales

1. Lee y revisa la Guía de Estudio.
2. Observa el video de la clase magistral.
3. Participa activamente en el FORO DEBATE.
4. Revisa en el calendario el día y hora en que se realizará las actividades prácticas y la tutoría.
5. Realizar la práctica propuesta por el docente.
6. Revisa las Rúbricas acorde a la actividad prácticas.
7. Toma los principales apuntes que vayas rescatando del texto.
8. Reflexiona sobre los temas tratados en la clase.
9. Sube el documento en formato PDF en la plataforma Moodle.
10. Revisa la calificación de tus actividades prácticas.

FORO DE DEBATE

Tema: Los arreglos (arrays) son estructuras de datos fundamentales en la programación, ya que permiten almacenar y gestionar múltiples valores de manera eficiente.

- ✓ ¿Por qué crees que los arreglos son importantes en la resolución de problemas computacionales?
- ✓ Explica cómo se declara un array, cómo se accede a sus elementos y menciona un ejemplo en el que un arreglo sea útil en la vida real.

Instrucciones para Participar:

- Exponga su punto de vista acerca del tema.
- Comenta las respuestas de otros participantes.
- Promueve un ambiente respetuoso y constructivo.

PRÁCTICA INVESTIGATIVA

Taller: Comparación de Arreglos en Diferentes Lenguajes de Programación.

- ✓ Investiga cómo se manejan los arreglos en al menos tres lenguajes de programación (Python, Java, C++, JavaScript, etc.).
- ✓ Explica las diferencias en su sintaxis, declaración y acceso a los elementos.
- ✓ Presenta una tabla comparativa con ejemplos en cada lenguaje.
- ✓ Entrega en Word o PDF, incluyendo fragmentos de código y gráficos explicativos.

Consigna:

- El trabajo de investigación debe presentarse de manera estructurada y coherente para comunicar efectivamente los hallazgos y conclusiones.
- Asegúrese de que el estilo y formato del trabajo sigan las pautas de citación y presentación establecidas por la institución.
- Utilice un lenguaje claro y evite jergas o términos técnicos innecesarios.
- Revise y corrija la redacción y la gramática antes de la presentación final (ver anexo).

PRÁCTICA TEST

Antes de realizar el test, es importante considerar varios aspectos para aprovechar al máximo la experiencia.

Consideraciones para realizar el test

1. Lee y revisa el documento.
2. Observa el video de la clase magistral.
3. Revisa en el calendario el día y hora en que se debe de realizar el Test.
4. Realiza el Test de la unidad considerando lo Siguiente:

- No cambiarte entre paginas durante la realización del Test.
- Revisa que tu conexión a internet sea estable.
- Recuerda que todo el contenido está inmerso dentro del documento base.

5. Revisa tu puntuación, de inmediato al finalizar de la actividad.

Tema 2: Tipos de arreglos.

Objetivo: Con estos conocimientos, podrá aprender sobre los tipos de arreglos en programación, comprendiendo su aplicación en distintos lenguajes bajo una misma lógica. Además, explorará la relación entre funciones matemáticas y arreglos, permitiéndole manipular y procesar datos de manera eficiente en su formación online.

Los arreglos son una de las estructuras de datos más utilizadas en programación, permitiendo almacenar y manipular conjuntos de valores de manera eficiente. Dependiendo de su organización y características, los arreglos pueden clasificarse en distintos tipos, y aunque su implementación varía según el lenguaje de programación, la idea central sigue siendo la misma. Además, los arreglos pueden utilizarse en operaciones matemáticas para realizar cálculos avanzados y optimizar el procesamiento de datos.

Tipos de arreglos

Los arreglos pueden diferenciarse según su estructura y la manera en que se almacenan los datos. A continuación, se presentan los principales tipos:

Arreglos Unidimensionales (Vectores):
Son listas ordenadas de datos en una sola dimensión, donde cada elemento es accesible mediante un índice numérico.

Ejemplo en Python:

python

CopiarEditar

```
numeros = [10, 20, 30, 40, 50]

print(numeros[2])  # Salida: 30
```

Ejemplo en Java:

java

CopiarEditar

```
int[] numeros = {10, 20, 30, 40, 50};
```

```
System.out.println(numeros[2]);  // Salida: 30
```

Arreglos Bidimensionales (Matrices):
Son arreglos organizados en filas y columnas, útiles para representar estructuras como tablas o imágenes.

Ejemplo en Python:

```
python

CopiarEditar

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

print(matriz[1][2])  # Salida: 6
```

Ejemplo en Java:

```
java

CopiarEditar

int[][] matriz = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
};

System.out.println(matriz[1][2]);  // Salida: 6
```

Arreglos Multidimensionales:

Son extensiones de los arreglos bidimensionales, organizados en múltiples niveles, utilizados en inteligencia artificial y gráficos en 3D.

Ejemplo en Python con NumPy:

python

CopiarEditar

```
import numpy as np

cubo = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])

print(cubo[1][0][1])  # Salida: 6
```

Arreglos Dinámicos:

En lenguajes como Python o JavaScript, los arreglos pueden crecer y reducirse dinámicamente, a diferencia de los arreglos estáticos en C o Java.

Ejemplo en Python:

python

CopiarEditar

```
lista = []

lista.append(10)

lista.append(20)

print(lista)  # Salida: [10, 20]
```

Distintos lenguajes, misma idea

A pesar de las diferencias en sintaxis y funciones específicas, la idea de los arreglos sigue siendo la misma en todos los lenguajes: almacenar múltiples valores en una estructura indexada.

Lenguaje	Declaración de un Arreglo
Python	numeros = [1, 2, 3, 4, 5]
Java	int[] numeros = {1, 2, 3, 4, 5};
C	int numeros[] = {1, 2, 3, 4, 5};
JavaScript	let numeros = [1, 2, 3, 4, 5];

Aunque los métodos de manipulación varían, todos los lenguajes permiten acceder a los elementos por medio de índices y recorrer los arreglos con estructuras de control como for o while.

Funciones matemáticas y arreglos

Los arreglos se pueden utilizar en **operaciones matemáticas** para optimizar cálculos y análisis de datos. En Python, la biblioteca **NumPy** proporciona herramientas avanzadas para trabajar con matrices y cálculos numéricos.

Operaciones Matemáticas con Arreglos

1. **Suma de elementos de un arreglo:**

python

CopiarEditar

```
numeros = [10, 20, 30, 40, 50]

suma = sum(numeros)

print(suma)  # Salida: 150
```

2. **Cálculo de media, desviación estándar y suma con NumPy:**

python

CopiarEditar

```
import numpy as np

datos = np.array([1, 2, 3, 4, 5])

print(np.mean(datos))  # Media aritmética

print(np.std(datos))   # Desviación estándar

print(np.sum(datos))   # Suma de elementos
```

3. **Multiplicación de Matrices:**

python

CopiarEditar

```
import numpy as np

A = np.array([[1, 2], [3, 4]])

B = np.array([[5, 6], [7, 8]])

resultado = np.dot(A, B)  # Producto de matrices

print(resultado)
```

4. **Generación de un arreglo con valores aleatorios:**

python

CopiarEditar

```
import numpy as np

aleatorios = np.random.randint(1, 100, size=10)

print(aleatorios)
```

Las funciones matemáticas aplicadas a arreglos son esenciales en campos como estadística, aprendizaje automático, procesamiento de imágenes y simulaciones científicas.

Los arreglos son una estructura fundamental en programación que permite almacenar y manipular múltiples datos de manera eficiente. Existen diferentes tipos de arreglos, desde unidimensionales hasta multidimensionales, y su implementación varía según el lenguaje de programación. Sin embargo, la idea principal sigue siendo la misma en todos los lenguajes: proporcionar una forma ordenada y eficiente de manejar datos. Además, el uso de arreglos en operaciones matemáticas optimiza cálculos y análisis de datos, siendo una herramienta fundamental en el desarrollo de software, inteligencia artificial y aplicaciones científicas.

ACTIVIDADES PRÁCTICAS

Tabla

Detalle de la Práctica

Practicas	Instrucciones	Duración de la Practica	Mecanismo de Evaluación
Foro de debate	Revisar Actividades Prácticas	0:30 mm	Por rúbricas
Práctica investiga: Taller/Ensayo/Investigación	Revisar Actividades Prácticas	3:30 h	Por rúbricas
Tutorías Prácticas	Revisar Actividades Prácticas	2:00 mm	Por participación en clase
Test evaluativo	Revisar Actividades Prácticas	0:30 mm	Formativa

Nota. Elaboración propia

Consideraciones Generales

1. Lee y revisa la Guía de Estudio.
2. Observa el video de la clase magistral.
3. Participa activamente en el FORO DEBATE.
4. Revisa en el calendario el día y hora en que se realizará las actividades prácticas y la tutoría.
5. Realizar la práctica propuesta por el docente.
6. Revisa las Rúbricas acorde a la actividad prácticas.
7. Toma los principales apuntes que vayas rescatando del texto.
8. Reflexiona sobre los temas tratados en la clase.
9. Sube el documento en formato PDF en la plataforma Moodle.
10. Revisa la calificación de tus actividades prácticas.

FORO DE DEBATE

Tema: Los arreglos pueden clasificarse en distintos tipos, como unidimensionales, bidimensionales y multidimensionales. Aunque la sintaxis varía según el lenguaje de programación, el concepto sigue siendo el mismo.

- ✓ ¿Por qué crees que los arreglos son esenciales en la manipulación de datos? Explica cómo se usan los arreglos en diferentes lenguajes de programación y menciona un ejemplo de una función matemática que pueda aplicarse sobre un arreglo.

Instrucciones para Participar:

- Exponga su punto de vista acerca del tema.
- Comenta las respuestas de otros participantes.
- Promueve un ambiente respetuoso y constructivo.

PRÁCTICA INVESTIGATIVA

Taller: Infografía sobre Tipos de Arreglos y su Uso en Distintos Lenguajes.

Diseña una infografía que explique:

- Las diferencias entre arreglos unidimensionales, bidimensionales y multidimensionales.
- Ejemplo de cómo se declaran los arreglos en diferentes lenguajes de programación (Python, Java, C++, etc.).
- Cómo se pueden utilizar en funciones matemáticas básicas y avanzadas.
- Usa herramientas digitales como Canva, Piktochart o PowerPoint y sube el archivo en PDF o imagen (JPG/PNG) a la plataforma.

Consigna:

- El trabajo de investigación debe presentarse de manera estructurada y coherente para comunicar efectivamente los hallazgos y conclusiones.
- Asegúrese de que el estilo y formato del trabajo sigan las pautas de citación y presentación establecidas por la institución.
- Utilice un lenguaje claro y evite jergas o términos técnicos innecesarios.

- Revise y corrija la redacción y la gramática antes de la presentación final (ver anexo).

PRÁCTICA TEST

Antes de realizar el test, es importante considerar varios aspectos para aprovechar al máximo la experiencia.

Consideraciones para realizar el test

1. Lee y revisa el documento.
2. Observa el video de la clase magistral.
3. Revisa en el calendario el día y hora en que se debe de realizar el Test.
4. Realiza el Test de la unidad considerando lo Siguiente:

- No cambiarte entre paginas durante la realización del Test.
- Revisa que tu conexión a internet sea estable.
- Recuerda que todo el contenido está inmerso dentro del documento base.

5. Revisa tu puntuación, de inmediato al finalizar de la actividad.

IV. DATOS GENERALES DE LA UNIDAD 4

Información General

1.1.	Nivel	Primero					
1.2.	Nombre Unidad	Lenguaje de programación					
1.3.	Tema 1 – Unidad	Introducción a Lenguaje de Programación					
1.4.	Tema 2 – Unidad	Funciones					
1.5.	Unidad organizacional	Unidad Profesional					
1.6.	Unidad	4					
1.7.	Total, Horas Unidad	30					
1.8.	Detalle de horas Unidad	ACD	15	AA	3	APE	12

Resultado de Aprendizaje de Unidad

Al finalizar la unidad, el estudiante comprenderá los fundamentos de los lenguajes de programación, incluyendo su estructura, sintaxis y aplicación en el desarrollo de software. Identificará las principales herramientas de control de flujo (if, for, while, do while), utilizándolas para estructurar la lógica de los programas. Además, aprenderá el uso de funciones en programación, entendiendo su importancia para modularizar el código y mejorar su eficiencia. Como resultado, será capaz de desarrollar un programa básico aplicando los conocimientos adquiridos en un lenguaje de programación específico.

Metodología

El aprendizaje se llevará a cabo mediante materiales didácticos interactivos, videos explicativos y análisis de ejemplos de código en lenguajes como Python, Java o C++. Se realizarán ejercicios prácticos sobre la implementación de estructuras de control de flujo, permitiendo al estudiante comprender cómo mejorar la lógica de un programa. Para el desarrollo de funciones, se fomentará la experimentación con programas modulares, creando funciones propias y reutilizables. Se promoverá el uso de entornos de programación en línea como Replit, Jupyter Notebook o Visual Studio Code para la práctica y desarrollo de código. La evaluación se basará en proyectos donde los estudiantes diseñarán y ejecutarán un programa funcional que integre estructuras de control y funciones.

DESARROLLO

Unidad 4: Lenguaje de programación

Tema 1: Introducción a Lenguaje de Programación

Objetivo: Con estos conocimientos, podrá aprender los fundamentos de la programación, comprendiendo el concepto y la sintaxis de los lenguajes de programación, así como el uso de herramientas de control de flujo como if, for, while y do while. Esto le permitirá desarrollar soluciones lógicas y estructuradas para la resolución de problemas en su formación online.

Los lenguajes de programación son el medio a través del cual los desarrolladores comunican instrucciones a las computadoras para ejecutar tareas específicas. Estos lenguajes se basan en reglas de sintaxis y semántica que permiten escribir código comprensible para las máquinas. Para comprender cómo funciona un lenguaje de programación, es fundamental conocer sus fundamentos, su estructura sintáctica y las herramientas de control de flujo que permiten definir la lógica de ejecución de los programas.

Fundamentos de programación

La programación es el proceso de creación de software mediante la escritura de instrucciones en un lenguaje comprensible para las computadoras. Los fundamentos de la programación incluyen:

1. **Variables y Tipos de Datos:**
   Las variables almacenan información dentro de un programa y pueden ser de diferentes tipos, como enteros, flotantes, cadenas de texto y booleanos.

2. **Operadores:**
   Son símbolos que permiten realizar operaciones matemáticas (+, -, *, /), comparaciones (==, !=, >, <) y operaciones lógicas (and, or, not).

3. **Estructuras de Control:**
   Permiten decidir el flujo de ejecución del programa, incluyendo condicionales y bucles.

4. **Funciones y Procedimientos:**
   Son bloques de código reutilizables que encapsulan una serie de instrucciones para ser ejecutadas cuando sea necesario.

5. **Manejo de Entradas y Salidas:**
   Los programas pueden recibir datos del usuario (entrada) y mostrar resultados (salida), utilizando funciones como input() en Python o scanf() en C.

Concepto y Sintaxis

Cada lenguaje de programación tiene su propia sintaxis, es decir, un conjunto de reglas que definen cómo deben escribirse las instrucciones. Sin embargo, los conceptos básicos son comunes entre la mayoría de los lenguajes.

Ejemplo de un programa en Python que imprime "Hola, mundo":

python

CopiarEditar

```
print("Hola, mundo")
```

Ejemplo en C:

c

CopiarEditar

```
#include <stdio.h>

int main() {
    printf("Hola, mundo\n");
    return 0;
}
```

Ejemplo en Java:

java

CopiarEditar

```
public class HolaMundo {
    public static void main(String[] args) {
```

```
        System.out.println("Hola, mundo");
    }
}
```

Aunque cada lenguaje tiene su propia estructura, todos cumplen la misma función: enviar instrucciones a la computadora para que ejecute una acción.

Herramientas de control de flujo if for while do while

Las estructuras de control de flujo determinan el orden en el que se ejecutan las instrucciones dentro de un programa. Existen dos tipos principales:

Condicionales (if)

Los condicionales permiten tomar decisiones dentro de un programa.

Ejemplo en Python:

python

CopiarEditar

```
edad = int(input("Ingrese su edad: "))
if edad >= 18:
    print("Eres mayor de edad")
else:
    print("Eres menor de edad")
```

Ejemplo en **C**:

c

CopiarEditar

```
#include <stdio.h>
int main() {
```

```
    int edad;
    printf("Ingrese su edad: ");
    scanf("%d", &edad);
    if (edad >= 18) {
        printf("Eres mayor de edad\n");
    } else {
        printf("Eres menor de edad\n");
    }
    return 0;
}
```

Bucles (for, while, do while)

Los bucles permiten ejecutar repetitivamente un bloque de código hasta que se cumpla una condición específica.

Bucle for

Se usa cuando se conoce la cantidad exacta de iteraciones.
Ejemplo en Python:

python

CopiarEditar

```
for i in range(1, 6):
    print("Número:", i)
```

Ejemplo en **C**:

c

CopiarEditar

```
#include <stdio.h>

int main() {

    for (int i = 1; i <= 5; i++) {

        printf("Número: %d\n", i);

    }

    return 0;

}
```

Bucle while

Se ejecuta mientras la condición sea verdadera.
Ejemplo en Python:

python

CopiarEditar

```
numero = 1

while numero <= 5:

    print("Número:", numero)

    numero += 1
```

Ejemplo en **C**:

c

CopiarEditar

```
#include <stdio.h>

int main() {

    int numero = 1;

    while (numero <= 5) {
```

```
        printf("Número: %d\n", numero);
        numero++;
    }
    return 0;
}
```

Bucle do while

Similar al while, pero se ejecuta al menos una vez, ya que la condición se evalúa al final. Ejemplo en **C**:

c

CopiarEditar

```
#include <stdio.h>
int main() {
    int numero = 1;
    do {
        printf("Número: %d\n", numero);
        numero++;
    } while (numero <= 5);
    return 0;
}
```

En Python, no existe do while, pero se puede lograr un comportamiento similar usando while con una condición inicial True y un break para salir del bucle.

El lenguaje de programación es la base del desarrollo de software y permite que los desarrolladores creen aplicaciones mediante la escritura de código con reglas sintácticas específicas. Para programar eficazmente, es esencial comprender los fundamentos de

programación, la sintaxis del lenguaje y las estructuras de control de flujo, como condicionales (if) y bucles (for, while, do while). Dominar estos conceptos permite escribir programas estructurados, eficientes y escalables en diversos lenguajes de programación.

ACTIVIDADES PRÁCTICAS

Tabla

Detalle de la Práctica

Practicas	Instrucciones	Duración de la Practica	Mecanismo de Evaluación
Foro de debate	Revisar Actividades Prácticas	0:30 mm	Por rúbricas
Práctica investiga: Taller/Ensayo/Investigación	Revisar Actividades Prácticas	3:30 h	Por rúbricas
Tutorías Prácticas	Revisar Actividades Prácticas	2:00 mm	Por participación en clase
Test evaluativo	Revisar Actividades Prácticas	0:30 mm	Formativa

Nota. Elaboración propia

Consideraciones Generales

1. Lee y revisa la Guía de Estudio.
2. Observa el video de la clase magistral.
3. Participa activamente en el FORO DEBATE.
4. Revisa en el calendario el día y hora en que se realizará las actividades prácticas y la tutoría.
5. Realizar la práctica propuesta por el docente.
6. Revisa las Rúbricas acorde a la actividad prácticas.
7. Toma los principales apuntes que vayas rescatando del texto.
8. Reflexiona sobre los temas tratados en la clase.
9. Sube el documento en formato PDF en la plataforma Moodle.
10. Revisa la calificación de tus actividades prácticas.

FORO DE DEBATE

Tema: Los lenguajes de programación son herramientas esenciales en el desarrollo de software, ya que permiten instruir a una computadora para realizar tareas específicas.

- ✓ ¿Por qué crees que es importante entender los fundamentos de programación, la sintaxis y las estructuras de control de flujo antes de aprender un lenguaje de programación en particular?
- ✓ Explica cómo las estructuras if, for, while y do-while permiten controlar el flujo de ejecución de un programa.

Instrucciones para Participar:

- Exponga su punto de vista acerca del tema.
- Comenta las respuestas de otros participantes.
- Promueve un ambiente respetuoso y constructivo.

PRÁCTICA INVESTIGATIVA

Ensayo: Introducción a los Lenguajes de Programación.

Consigna:

- El trabajo de investigación debe presentarse de manera estructurada y coherente para comunicar efectivamente los hallazgos y conclusiones.
- Asegúrese de que el estilo y formato del trabajo sigan las pautas de citación y presentación establecidas por la institución.
- Utilice un lenguaje claro y evite jergas o términos técnicos innecesarios.
- Revise y corrija la redacción y la gramática antes de la presentación final (ver anexo).

PRÁCTICA TEST

Antes de realizar el test, es importante considerar varios aspectos para aprovechar al máximo la experiencia.

Consideraciones para realizar el test

1. Lee y revisa el documento.
2. Observa el video de la clase magistral.
3. Revisa en el calendario el día y hora en que se debe de realizar el Test.
4. Realiza el Test de la unidad considerando lo Siguiente:

- No cambiarte entre paginas durante la realización del Test.
- Revisa que tu conexión a internet sea estable.
- Recuerda que todo el contenido está inmerso dentro del documento base.

5. Revisa tu puntuación, de inmediato al finalizar de la actividad.

Tema 2: Funciones

Objetivo: Con estos conocimientos, podrá aprender sobre el uso de funciones en un lenguaje de programación, comprendiendo su estructura y aplicación para modularizar el código y mejorar su eficiencia. Además, adquirirá la capacidad de crear programas funcionales, aplicando buenas prácticas de programación en su formación online.

Las funciones son bloques de código reutilizables dentro de un programa que permiten realizar una tarea específica. Su uso facilita la organización del código, mejora la legibilidad y reduce la duplicación de instrucciones. En todos los lenguajes de programación, las funciones son fundamentales para estructurar programas de manera eficiente.

Funciones en lenguaje de programación

Una función en programación se define por tres elementos clave:

- Definición: Se establece el nombre de la función y los parámetros que recibe.
- Llamado: Se ejecuta la función desde otro lugar del programa.
- Retorno: Opcionalmente, la función devuelve un valor como resultado.

La sintaxis para definir funciones varía según el lenguaje de programación, pero su propósito es el mismo: encapsular un conjunto de instrucciones para ser reutilizadas.

Ejemplo de Función en Python

python

CopiarEditar

```
def suma(a, b):
    return a + b

resultado = suma(10, 5)

print("La suma es:", resultado)
```

Aquí, la función suma recibe dos valores (a y b), los suma y devuelve el resultado, que luego es impreso.

Ejemplo de Función en C

c

CopiarEditar

```
#include <stdio.h>


int suma(int a, int b) {
    return a + b;
}
int main() {
    int resultado = suma(10, 5);
    printf("La suma es: %d\n", resultado);
    return 0;
}
```

En este caso, la función suma en C devuelve un valor entero (int), y se llama desde la función main().

Ejemplo de Función en Java

java

CopiarEditar

```
public class Funciones {
    public static int suma(int a, int b) {
        return a + b;
    }
```

```
    public static void main(String[] args) {
        int resultado = suma(10, 5);
        System.out.println("La suma es: " + resultado);
    }
}
```

Aquí, la función suma está dentro de una clase, siguiendo la estructura de Java.

Las funciones son esenciales en cualquier lenguaje de programación, ya que permiten la reutilización de código y la organización eficiente de los programas. A través de ellas, se pueden realizar cálculos, gestionar flujos de ejecución y modularizar el código para mejorar su mantenibilidad. Dominar las funciones es clave para desarrollar programas más estructurados, eficientes y escalables en cualquier lenguaje de programación.

ACTIVIDADES PRÁCTICAS

Tabla

Detalle de la Práctica

Practicas	Instrucciones	Duración de la Practica	Mecanismo de Evaluación
Foro de debate	Revisar Actividades Prácticas	0:30 mm	Por rúbricas
Práctica investiga: Taller/Ensayo/Investigación	Revisar Actividades Prácticas	3:30 h	Por rúbricas
Tutorías Prácticas	Revisar Actividades Prácticas	2:00 mm	Por participación en clase
Test evaluativo	Revisar Actividades Prácticas	0:30 mm	Formativa

Nota. Elaboración propia

Consideraciones Generales

1. Lee y revisa la Guía de Estudio.
2. Observa el video de la clase magistral.
3. Participa activamente en el FORO DEBATE.
4. Revisa en el calendario el día y hora en que se realizará las actividades prácticas y la tutoría.
5. Realizar la práctica propuesta por el docente.
6. Revisa las Rúbricas acorde a la actividad prácticas.
7. Toma los principales apuntes que vayas rescatando del texto.
8. Reflexiona sobre los temas tratados en la clase.
9. Sube el documento en formato PDF en la plataforma Moodle.
10. Revisa la calificación de tus actividades prácticas.

FORO DE DEBATE

Tema: Las funciones en los lenguajes de programación permiten estructurar el código de manera más eficiente, facilitando su reutilización y mantenimiento.

- ✓ ¿Por qué crees que el uso de funciones es esencial en la programación?
- ✓ Explica cómo se definen y utilizan las funciones en un lenguaje de programación y proporciona un ejemplo práctico donde sean útiles.

Instrucciones para Participar:

- Exponga su punto de vista acerca del tema.
- Comenta las respuestas de otros participantes.
- Promueve un ambiente respetuoso y constructivo.

PRÁCTICA INVESTIGATIVA

Taller: Creación de un Programa con Funciones en un Lenguaje de Programación.

- ✓ Escribe un programa en Python, Java, C++ o cualquier lenguaje que incluya:
- ✓ Una función que reciba parámetros y devuelva un resultado (ejemplo: calcular el área de un triángulo o convertir temperaturas).
- ✓ Una función sin retorno que realice una tarea específica (ejemplo: mostrar un mensaje de bienvenida).
- ✓ Explicación detallada del código y su ejecución.

Consigna:

- El trabajo de investigación debe presentarse de manera estructurada y coherente para comunicar efectivamente los hallazgos y conclusiones.
- Asegúrese de que el estilo y formato del trabajo sigan las pautas de citación y presentación establecidas por la institución.
- Utilice un lenguaje claro y evite jergas o términos técnicos innecesarios.
- Revise y corrija la redacción y la gramática antes de la presentación final (ver anexo).

PRÁCTICA TEST

Antes de realizar el test, es importante considerar varios aspectos para aprovechar al máximo la experiencia.

Consideraciones para realizar el test

1. Lee y revisa el documento.
2. Observa el video de la clase magistral.
3. Revisa en el calendario el día y hora en que se debe de realizar el Test.
4. Realiza el Test de la unidad considerando lo Siguiente:

- No cambiarte entre paginas durante la realización del Test.
- Revisa que tu conexión a internet sea estable.
- Recuerda que todo el contenido está inmerso dentro del documento base.

5. Revisa tu puntuación, de inmediato al finalizar de la actividad.

V. Bibliografía

Bibliografía básica.

Trejos Buritica, O. I. (2021). Lógica de Programación: Solucionario en pseudocódigo/ Ejercicios resueltos. EDICIONES DE LA U

Trejos Buritica, O. I. (2023). Lógica de Programación. 2 edición. EDICIONES DE LA U

Bibliografía de consulta.

Durán Pico, U. (2014). Las matemáticas en el desarrollo de algoritmos computacionales en estudiantes de análisis de sistemas. Instituto Tecnológico Superior Portoviejo ITSUP. Artículo obtenido de: https://dialnet.unirioja.es/servlet/articulo?codigo=8280820

Alcaraz, J., Hernández, L., Paredes, S. (2011). Sistema de generación personalizada de ejercicios de lógica computacional. Universidad de Murcia. Artículo obtenido de: https://dialnet.unirioja.es/servlet/articulo?codigo=8644495

Prieto Taborda, M. A., Bermón Angarita, L., & Ramírez Castañeda, L. A. (2019). Diseño, desarrollo y evaluación de un recurso educativo digital para la introducción a la Administración de Sistemas Informáticos. Revista Virtual Universidad Católica del Norte. Obtenido de https://www.redalyc.org/journal/1942/194259585004/194259585004.pdf.

Anexo 1: Modelo de Foro

Título del Foro:

[Nombre del Tema o Pregunta]

Desarrollo:

[Breve descripción del tema o pregunta]

Reglas del Foro:

Regla 1: Antes de participar en el debate, realiza una investigación exhaustiva sobre el tema en cuestión. Comprender los hechos, datos y argumentos relacionados con el tema fortalecerá tu posición y mejorarán la calidad de la discusión.

Regla 2: Muestra respeto hacia tus compañeros de debate, incluso si no estás de acuerdo con sus opiniones. Escucha activamente sus argumentos y evita interrupciones. Expresa tus puntos de vista de manera respetuosa y constructiva.

Regla 3: Contribuye de manera equitativa al debate, evitando la monopolización de la conversación. Respeta el tiempo asignado y asegúrate de que otros también tengan la oportunidad de expresar sus opiniones.

Anexo 2: Esquema de Ensayo

Caratula: (Escribe aquí los datos referentes a la materia, nombres completos, tema, periodo, Institución Educativa y logos)

Título:

(Escribe aquí el título de tu ensayo)

Introducción:

Desarrollo:

Conclusión:

Bibliografía

Anexo 3: Esquema de Taller/Investigación

Caratula:

(Escribe aquí los datos referentes a la materia, tema, periodo, Institución Educativa y logos)

Título:

(Escribe aquí el título principal de tu investigación)

Temas a investigar:

- **Tema 1:**
- **Tema 2:**
- **Tema 3:**

Conclusión:

Bibliografía

Anexos

Anexo 4: Rubrica de taller práctico

INDICADORES	MS 10 PUNTOS	S 6 PUNTOS	PS 2 PUNTOS	PUNTAJE
Número de ejercicios resueltos	Realiza en su totalidad los ejercicios que se propones.	Realiza entre la mitad y más de los ejercicios propuestos.	Realiza la menor parte o no realiza ningún ejercicio propuesto.	**40%**
Procedimiento y resultados de los ejercicios resueltos	Desarrolla el procedimiento, lo detalla y organiza los datos obtenidos.	Desarrolla el procedimiento, lo detalla poco y organiza algunos, pero otros no.	Detalla poco o no detalla nada del procedimiento.	**40%**
Presentación	Maneja una presentación prolija de la tarea.	La tarea se presenta de una forma aceptable.	La tarea no cuenta con una buena presentación	**20%**
			TOTAL=	100 %

Puntaje: 10 puntos

Anexo 5: Rúbrica de Foro

INDICADORES	MS 5 PUNTOS	S 3 PUNTOS	PS 2 PUNTOS	PUNTAJE
Comprensión del tema	La respuesta argumentada demuestra conocimiento concreto del tema tratado.	La respuesta argumentada demuestra conocimiento general de la mayoría del tema.	La respuesta argumentada demuestra un conocimiento limitado del tema.	20 %
Cuestionamientos y criterios	Presentó las dudas e inquietudes que se le presentaron durante el desarrollo del tema, o al menos sacó diferentes conclusiones, que las expuso en el debate.	Presenta al menos 2 preguntas o argumentos, aunque demuestra que no ha comprende ni revisado en profundidad el material.	No presenta dudas ni argumentos, aunque conozca del tema, no aporta al debate.	20 %
Redacción y ortografía	No presenta errores ortográficos ni de redacción.	Presenta algunos errores ortográficos y/o de redacción.	Presenta muchos errores de redacción y de ortografía.	20 %
Interacción: relación/ confrontación con intervenciones de otros/as	Comparte activamente con de sus compañeros los temas planteados y resuelve dudas de sus compañeros con argumentos valederos en más de 2 intervenciones.	Comparte con de sus compañeros los temas planteados, argumentando en al menos de 2 intervenciones.	Plantea dudas a sus compañeros con el tema, sin embargo, no participa ni responde con argumentos lógicos. Solo interviene una vez.	20 %
Participación	Participa con más de 2 intervenciones, además de su contestación.	Participa al menos con 5 Intervenciones, además de su contestación.	No interviene en otra participación a más de la suya.	20 %
			TOTAL=	100 %

Puntaje: 5 puntos

Anexo 6: Rubrica de Ensayo

INDICACIONES	MS 10 PUNTOS	S 6 PUNTOS	PS 2 PUNTOS	PUNTAJE
Enfoque o Idea principal	La idea principal nombra el tema del ensayo y esquematiza los puntos principales a discutir.	La idea principal nombra el tema del ensayo.	La idea principal esquematiza algunos o todos los puntos a discutir, pero no menciona el tema.	20 %
Transiciones o Conectores	Una variedad bien pensada de transiciones fue usada. Estas muestran claramente como están las ideas conectadas.	Las transiciones muestran cÃ³mo están las ideas conectadas, pero hay muy poca variedad.	Algunas transiciones funcionan bien, pero la conexiÃ³n entre las ideas no es clara.	20 %
Conclusión	La conclusión es fuerte y dejó al lector con una idea absolutamente clara de la posición del autor. Un parafraseo efectivo de la idea principal empieza la conclusión.	La conclusión es evidente. La posición del autor es parafraseada en las primeras dos oraciones de la conclusión.	La posición del autor es parafraseada en la conclusión, pero no al principio de la misma.	20 %
Gramática y ortografía	El autor no comete errores de gramática ni de ortografía que distraen al lector del contenido del ensayo.	El autor comete 1 a 2 errores gramaticales u ortográficos que distraen al lector del contenido del ensayo.	El autor comete 3 a 4 errores gramaticales u ortográficos que distraen al lector del contenido del ensayo.	20 %
Fuentes	Todas las fuentes usadas para las citas, las estadísticas y los hechos son creíbles y están citadas correctamente.	Todas las fuentes usadas para las citas, las estadísticas y los hechos son creíbles y la mayoría está citada correctamente.	La mayoría de las fuentes usadas para las citas, las estadísticas y los hechos es creíble y está citada correctamente.	20 %
			TOTAL=	100 %

Puntaje: 10 puntos

Anexo 7: Rúbrica de Taller/Investigación

INDICADORES	MS 10 PUNTOS	S 6 PUNTOS	PS 2 PUNTOS	PUNTAJE
Organización de las ideas	El texto es coherente y tiene sentido toda la lectura.	Presenta ideas aisladas, no enlazadas entre sí.	El texto no tiene coherencia y las ideas no están claras.	30 %
Ortografía	Tiene 4 o menos faltas ortográficas	Tiene entre 5 y 9 faltas ortográficas.	Tiene más de 10 faltas ortográficas.	30 %
Calidad de la información presentada	La información está claramente relacionada con el tema y proporciona varios niveles de relación con sus diferentes entornos.	La información da respuesta a la consigna principal del artículo, mas no aporta niveles de relaciones de la comunicación con el entorno próximo.	La información proporcionada guarda relación con las categorías, aunque su calidad argumentativa es deficiente.	40 %
			TOTAL=	100 %

Puntaje: 10 puntos