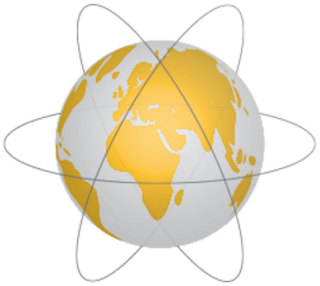
UNIVERSITY of SALZBURG

Curso de Python 2023

Lcdo. Diego Saavedra Mgtr.



UNIGIS América Latina

Z_GIS

DEPARTMENT OF GEOINFORMATICS - Z_GIS

Tabla de contenidos

1. Bienvenida

¡Bienvenidos al Curso Completo de Python: Desde Fundamentos hasta Aplicaciones Prácticas!

1.1. ¿Qué es este Curso?

Este curso exhaustivo te llevará desde los fundamentos básicos de la programación hasta la creación de aplicaciones prácticas utilizando el lenguaje de programación Python. A través de una combinación de teoría y ejercicios prácticos, te sumergirás en los conceptos esenciales de la programación y avanzarás hacia la construcción de proyectos reales. Desde la instalación de herramientas hasta la creación de una API con Django Rest Framework, este curso te proporcionará una comprensión sólida y práctica de Python y su aplicación en el mundo real.

1.2. ¿A quién está dirigido?

Este curso está diseñado para principiantes y aquellos con poca o ninguna experiencia en programación. No importa si eres un estudiante curioso, un profesional que busca cambiar de carrera o simplemente alguien que desea aprender a programar: este curso es para ti. Desde adolescentes hasta adultos, todos son bienvenidos a participar y explorar el emocionante mundo de la programación a través de Python.

1.3. ¿Cómo contribuir?

Valoramos tu participación en este curso. Si encuentras errores, deseas sugerir mejoras o agregar contenido adicional, ¡nos encantaría escucharte! Puedes contribuir a través de nuestra plataforma en línea, donde puedes compartir tus comentarios y sugerencias. Juntos, podemos mejorar continuamente este recurso educativo para beneficiar a la comunidad de estudiantes y entusiastas de la programación.

Este libro ha sido creado con el objetivo de brindar acceso gratuito y universal al conocimiento. Estará disponible en línea para que cualquiera, sin importar su ubicación o circunstancias, pueda acceder y aprender a su propio ritmo.

¡Esperamos que disfrutes este emocionante viaje de aprendizaje y descubrimiento en el mundo de la programación con Python!

Lección I.

Unidad 1: Introducción a la Programación

2. Introducción general a la Programación

La programación es el proceso de crear secuencias de instrucciones que le indican a una computadora cómo realizar una tarea específica.

Estas instrucciones se escriben en lenguajes de programación, que son conjuntos de reglas y símbolos utilizados para comunicarse con la máquina. La programación es una habilidad esencial en la era digital, ya que se aplica en una amplia variedad de campos, desde desarrollo de software y análisis de datos hasta diseño de juegos y automatización.

2.1. Conceptos Clave

2.1.1. Instrucciones

Son comandos específicos que le indican a la computadora qué hacer. Pueden ser simples, como imprimir un mensaje en pantalla, o complejas, como realizar cálculos matemáticos.

2.1.2. Lenguajes de Programación

Son sistemas de comunicación entre humanos y máquinas. Cada lenguaje tiene reglas sintácticas y semánticas que determinan cómo se escriben y ejecutan las instrucciones.

2.1.3. Algoritmos

Son conjuntos ordenados de instrucciones diseñados para resolver un problema específico. Los algoritmos son la base de la programación y se utilizan para desarrollar software eficiente.

2.1.4. Depuración

Es el proceso de identificar y corregir errores en el código. Los programadores pasan tiempo depurando para asegurarse de que sus programas funcionen correctamente.

2.2. Ejemplo:

```
print("Hola, bienvenido al mundo de la programación.")  ①
```

① Este es un ejemplo sencillo de un programa en Python que imprime un mensaje en pantalla.

2.3. Explicación

En Python, los comentarios comienzan con el símbolo `#`. No afectan la ejecución del programa, pero son útiles para documentar el código.

La línea `print("Hola, bienvenido al mundo de la programación.")` es una instrucción de impresión. La función `print()` muestra el texto entre paréntesis en la consola.

> **!** Actividad Práctica
>
> Escribe un programa que solicite al usuario su nombre y luego imprima un mensaje de bienvenida personalizado.

2.4. Explicación de la Actividad

El programa utilizará la función `input()` para recibir la entrada del usuario. Luego, utilizará la entrada proporcionada para imprimir un mensaje de bienvenida personalizado.

Lección II.

Unidad 2: Instalación de Python y más herramientas

3. Instalación de Python

La instalación de Python es el primer paso para comenzar a programar en este lenguaje. Python es un lenguaje de programación versátil y ampliamente utilizado, conocido por su sintaxis clara y legible. Aquí aprenderemos cómo instalar Python en diferentes sistemas operativos.

3.1. Conceptos Clave

3.1.1. Python

Lenguaje de programación de alto nivel que se utiliza para desarrollar aplicaciones web, científicas, de automatización y más.

3.1.2. Interprete

Python es un lenguaje interpretado, lo que significa que se ejecuta línea por línea en tiempo real.

3.1.3. IDE

Los entornos de desarrollo integrados (IDE) como Visual Studio Code (VS Code) o PyCharm brindan herramientas para escribir, depurar y ejecutar código de manera más eficiente.

3.2. Ejemplo

No se necesita código para esta lección, ya que se trata de instrucciones para la instalación de Python en diferentes sistemas operativos.

3.3. Explicación

Para instalar Python en sistemas Windows, macOS y Linux, se pueden seguir las instrucciones detalladas proporcionadas en el sitio web oficial de Python www.python.org/downloads/.

La instalación de Python generalmente incluye el intérprete de Python y una serie de herramientas y bibliotecas estándar que hacen que sea fácil comenzar a programar.

! Actividad Práctica

Instala Python en tu sistema operativo siguiendo las instrucciones del sitio web oficial de Python. Luego, verifica que Python esté correctamente instalado ejecutando el intérprete y escribiendo el siguiente código:

```
print("Python se ha instalado correctamente.")
```

3.4. Explicación de la Actividad

Esta actividad permite a los participantes aplicar lo aprendido instalando Python en su propio sistema y ejecutando un programa sencillo para confirmar que la instalación fue exitosa.

Lección III.

Unidad 3: Introducción a Python

4. Distintas formas de trabajar con Python

Python es un lenguaje de programación versátil que ofrece diferentes formas de interactuar con él. Aprenderemos las dos formas principales de trabajar con Python: **el intérprete interactivo** y los **scripts de Python**.

4.1. Conceptos Clave:

4.1.1. Intérprete Interactivo:

Permite ejecutar instrucciones de Python en tiempo real y ver los resultados inmediatamente en la consola.

4.1.2. Scripts de Python:

Son archivos que contienen una serie de instrucciones de Python que se pueden ejecutar en conjunto.

4.1.3. Ambientes Virtuales

Son entornos aislados que permiten tener instalaciones y bibliotecas de Python separadas para diferentes proyectos.

4.1.4. Ejemplo

```
>>> 2 + 3                                                    ①
5


                                                             ②
numero1 = 5
numero2 = 7
resultado = numero1 + numero2
print("El resultado de la suma es:", resultado)
                                                             ③
```

① Uso del intérprete interactivo
② Ejecución de un script de Python
③ Guarda este código en un archivo llamado “suma.py”

4.2. Explicación

El intérprete interactivo permite ejecutar expresiones de Python directamente en la consola y ver los resultados en tiempo real.

Los scripts de Python son archivos que contienen un conjunto de instrucciones. En este ejemplo, se muestra cómo crear un script simple que calcula la suma de dos números y lo imprime en la consola.

> ! Actividad Práctica
>
> Abre el intérprete interactivo de Python y realiza algunas operaciones matemáticas simples.
> Crea un archivo llamado “operaciones.py” y escribe un programa que realice operaciones aritméticas básicas y las muestre en la consola.

4.3. Explicación de la Actividad

Esta actividad permite a los participantes experimentar con el intérprete interactivo de Python y crear su propio script para realizar operaciones matemáticas.

5. Las bases de Python

En esta lección, exploraremos las bases fundamentales de Python. Aprenderemos sobre las variables, tipos de datos y operadores básicos que forman la base de cualquier programa en Python.

5.1. Conceptos Clave

5.1.1. Variables

Son nombres que se utilizan para almacenar valores en la memoria de la computadora.

5.1.2. Tipos de Datos

Incluyen enteros, flotantes, cadenas, booleanos y más. Cada tipo de dato tiene sus propias características y operaciones.

5.1.3. Operadores

Incluyen operadores aritméticos, de comparación y lógicos que se utilizan para realizar diferentes tipos de cálculos y comparaciones.

5.1.4. Ejemplo

```
# Variables y tipos de datos
nombre = "Juan"
edad = 25
altura = 1.75
es_mayor_de_edad = True

# Operadores aritméticos
suma = 5 + 3
resta = 10 - 2
multiplicacion = 4 * 6
division = 15 / 3

# Operadores de comparación
igual = 5 == 5
```

```
mayor_que = 10 > 5
menor_que = 7 < 12

# Operadores lógicos
and_logico = True and False
or_logico = True or False
not_logico = not True
```

5.2. Explicación:

Las variables se utilizan para almacenar información, como el nombre, la edad, la altura y si una persona es mayor de edad.

Los operadores aritméticos se utilizan para realizar cálculos matemáticos básicos.

Los operadores de comparación se utilizan para comparar valores y devuelven un valor booleano (verdadero o falso).

Los operadores lógicos se utilizan para combinar expresiones booleanas y realizar operaciones lógicas.

> ! Actividad Práctica:
>
> Crea variables que almacenen información sobre ti, como tu nombre, edad y altura.
> Realiza operaciones aritméticas y utiliza operadores de comparación para comparar valores.
> Combina expresiones booleanas utilizando operadores lógicos y observa los resultados.

5.3. Explicación de la Actividad

Esta actividad permite a los participantes aplicar los conceptos de variables, tipos de datos y operadores en ejemplos prácticos. Les ayuda a comprender cómo trabajar con diferentes tipos de datos y cómo realizar operaciones básicas en Python.

6. Identación

En Python, la identación (sangría) juega un papel crucial en la estructura y organización del código. Aprenderemos cómo se utiliza la identación para delimitar bloques de código y mejorar la legibilidad.

6.1. Conceptos Clave

6.1.1. Identación

- Espacios o tabulaciones al comienzo de una línea que indican la estructura del código.
- Bloques de Código: Conjuntos de instrucciones que se agrupan juntas y se ejecutan en conjunto.
- PEP 8: Guía de estilo para la escritura de código en Python que recomienda el uso de cuatro espacios para la identación.

6.1.2. Ejemplo

```
# Uso de la identación en un condicional
numero = 10

if numero > 5:
    print("El número es mayor que 5")
else:
    print("El número no es mayor que 5")
```

6.2. Explicación

- En este ejemplo, la identación se utiliza para delimitar los bloques de código dentro de las instrucciones “if” y “else”.
- La identación es crucial para que Python sepa qué instrucciones están dentro de un bloque y cuáles están fuera.

! Actividad Práctica:

Escribe un programa que solicite al usuario su edad y muestre un mensaje según si es mayor de 18 años o no.
Intenta cambiar la identación incorrectamente y observa cómo afecta al

funcionamiento del programa.

6.3. Explicación de la Actividad.

Esta actividad permite a los participantes comprender la importancia de la identación en Python al trabajar con bloques de código como los condicionales. Les ayuda a desarrollar el hábito de utilizar la identación adecuada para mantener el código organizado y legible.

7. Comentarios

Los comentarios son una herramienta importante en la programación para añadir explicaciones y notas en el código. Aprenderemos cómo agregar comentarios en Python y cómo pueden mejorar la comprensión del código.

7.1. Conceptos Clave:

7.1.1. Comentarios

Son notas en el código que no se ejecutan y se utilizan para explicar el propósito y funcionamiento de partes del programa.

7.1.2. Comentarios de una línea

Se crean con el símbolo "#" y abarcan una sola línea.

7.1.3. Comentarios de múltiples líneas

Se crean entre triple comillas (" " " o ' ' ') y pueden abarcar múltiples líneas.

7.2. Ejemplo

```
# Este es un comentario de una línea

"""
Este es un comentario
de múltiples líneas.
Puede abarcar varias líneas.
"""

numero = 42  # Este comentario está después de una instrucción
```

7.3. Explicación:

Los comentarios son ignorados por el intérprete y no afectan la ejecución del código.

Los comentarios son útiles para documentar el código, explicar partes difíciles de entender o dejar notas para otros programadores.

> **!** Importante
>
> Actividad Práctica:
> Escribe un programa que realice una tarea sencilla y agrega comentarios para explicar lo que hace cada parte.
> Escribe un comentario de múltiples líneas que explique el propósito general de tu programa.

7.4. Explicación de la Actividad

Esta actividad permite a los participantes practicar la adición de comentarios en su código para mejorar la legibilidad y la comprensión. Les ayuda a desarrollar la habilidad de documentar su código de manera efectiva para ellos mismos y para otros programadores.

8. Variables

Las variables son fundamentales en la programación ya que permiten almacenar y manipular datos. Aprenderemos cómo declarar y utilizar variables en Python.

8.1. Conceptos Clave:

8.1.1. Variables

Nombres que representan ubicaciones de memoria donde se almacenan datos.

8.1.2. Declaración de Variables.

- Asignación de un valor a un nombre utilizando el operador "=".
- Convenciones de Nombres: Siguen reglas para ser descriptivos y seguir una estructura (por ejemplo, letras minúsculas y guiones bajos para espacios).

8.2. Ejemplo

```
nombre = "Ana"
edad = 30
saldo_bancario = 1500.75
es_mayor_de_edad = True
```

8.3. Explicación:

En este ejemplo, se declaran variables para almacenar el nombre de una persona, su edad, su saldo bancario y un valor booleano que indica si es mayor de edad.

Los nombres de variables son descriptivos y siguen la convención de nombres recomendada (letras minúsculas y guiones bajos para espacios).

> ! Actividad Práctica
>
> Crea variables para almacenar información personal, como tu ciudad, tu edad y tu ocupación.
> Declara variables para almacenar cantidades numéricas, como el precio de un

producto y la cantidad de unidades disponibles.

8.4. Explicación de la Actividad

Esta actividad permite a los participantes practicar la declaración de variables en Python y aplicar el concepto de convenciones de nombres.

Les ayuda a comprender cómo almacenar y acceder a datos utilizando variables descriptivas y significativas.

9. Múltiples Variables

En Python, es posible asignar valores a múltiples variables en una sola línea. Aprenderemos cómo declarar y utilizar múltiples variables de manera eficiente.

9.1. Conceptos Clave:

9.1.1. Asignación Múltiple

Permite asignar valores a varias variables en una línea.

9.1.2. Desempaquetado de Valores

Se pueden asignar valores de una lista o tupla a múltiples variables en una sola operación.

9.1.3. Intercambio de Valores

Se pueden intercambiar los valores de dos variables utilizando asignación múltiple.

9.2. Ejemplo

```
nombre, edad, altura = "María", 28, 1.65
productos = ("Manzanas", "Peras", "Uvas")
producto1, producto2, producto3 = productos
```

9.3. Explicación:

En el primer ejemplo, se utilizó la asignación múltiple para declarar tres variables en una sola línea.

En el segundo ejemplo, se desempaquetaron los valores de una tupla en variables individuales.

::: {.callout-important} ### Actividad Práctica:

Crea una lista con los nombres de tus tres colores favoritos.

Utiliza la asignación múltiple para asignar los valores de la lista a tres variables individuales.

9.4. Explicación de la Actividad

Esta actividad permite a los participantes practicar la asignación múltiple y el desempaquetado de valores.

Les ayuda a comprender cómo trabajar eficientemente con múltiples variables y cómo aprovechar estas técnicas para simplificar el código.

10. Concatenación

La concatenación es la unión de cadenas de texto. Aprenderemos cómo combinar cadenas de texto en Python para crear mensajes más complejos.

10.1. Conceptos Clave:

10.1.1. Concatenación:

Operación que combina dos o más cadenas de texto para formar una cadena más larga.

10.1.2. Operador +

Se utiliza para concatenar cadenas de texto.

10.1.3. Conversión a Cadena

Es necesario convertir valores no string a cadenas antes de concatenarlos.

10.2. Ejemplo

```
nombre = "Luisa"
mensaje = "Hola, " + nombre + ". ¿Cómo estás?"
edad = 25
mensaje_edad = "Tienes " + str(edad) + " años."
```

10.3. Explicación:

En este ejemplo, se utilizó el operador "+" para concatenar cadenas de texto.

La variable "edad" se convirtió a una cadena utilizando la función "str()" antes de concatenarla.

! Actividad Práctica:

Crea una variable con tu comida favorita. Utiliza la concatenación para crear un mensaje que incluya tu comida favorita.

10.4. Explicación de la Actividad

Esta actividad permite a los participantes practicar la concatenación de cadenas de texto y comprender cómo construir mensajes más complejos utilizando variables y texto. Les ayuda a mejorar su capacidad para crear mensajes personalizados en sus programas.

Lección IV.

Unidad 4: Tipos de Dato

11. String y Números

Los strings y los números son dos tipos de datos fundamentales en Python. Aprenderemos cómo trabajar con strings (cadenas de texto) y los diferentes tipos de números en Python.

11.1. Conceptos Clave:

11.1.1. String

Secuencia de caracteres alfanuméricos. Se pueden definir utilizando comillas simples o dobles.

11.1.2. Números Enteros (int)

Representan números enteros positivos o negativos.

11.1.3. Números de Punto Flotante (float)

Representan números con decimales.

11.2. Ejemplo

```
# Strings
mensaje = "Hola, bienvenido al curso de Python."
nombre = 'María'

# Números
edad = 25
saldo = 1500.75
```

11.3. Explicación

En este ejemplo, se crean variables que almacenan strings y números.

Los strings se definen utilizando comillas simples o dobles.

Los números enteros y de punto flotante se asignan directamente a variables.

! Actividad Práctica:

Crea una variable con tu canción favorita.
Asigna tu edad a una variable y tu altura a otra variable.
Combina las variables para crear un mensaje personalizado.

11.4. Explicación de la Actividad:

Esta actividad permite a los participantes practicar la creación de strings y trabajar con números enteros y de punto flotante.

Les ayuda a comprender cómo almacenar y manipular diferentes tipos de datos en Python.

12. Listas

Las listas son estructuras de datos que permiten almacenar varios elementos en una sola variable. Aprenderemos cómo crear y manipular listas en Python.

12.1. Conceptos Clave:

12.1.1. Listas

Secuencias ordenadas de elementos que pueden ser de diferentes tipos.

12.1.2. Índices

Números que indican la posición de un elemento en la lista.

12.1.3. Acceso a Elementos

Se utiliza el índice para acceder a un elemento específico de la lista.

12.2. Ejemplo

```
frutas = ["manzana", "banana", "naranja", "uva"]
primer_fruta = frutas[0]
segunda_fruta = frutas[1]
```

12.3. Explicación

En este ejemplo, se crea una lista de frutas y se accede a elementos individuales utilizando índices.

Los índices comienzan desde 0, por lo que la primera fruta tiene el índice 0.

> ! Actividad Práctica:
>
> Crea una lista con los nombres de tus tres películas favoritas.
> Accede al segundo elemento de la lista e imprímelo en la consola.

12.4. Explicación de la Actividad:

Esta actividad permite a los participantes practicar la creación de listas y el acceso a elementos utilizando índices. Les ayuda a comprender cómo organizar y acceder a múltiples elementos en una sola variable.

13. Tuplas

Las tuplas son estructuras de datos similares a las listas, pero son inmutables, lo que significa que no se pueden modificar después de ser creadas. Aprenderemos cómo trabajar con tuplas en Python.

13.1. Conceptos Clave:

13.1.1. Tuplas

Secuencias ordenadas de elementos que, a diferencia de las listas, no se pueden modificar.

13.1.2. Inmutabilidad

Una vez creada una tupla, no se pueden agregar, modificar o eliminar elementos.

13.1.3. Acceso a Elementos

Se utiliza el índice para acceder a un elemento específico de la tupla.

13.2. Ejemplo

```
coordenadas = (3, 5)
x = coordenadas[0]
y = coordenadas[1]
```

13.3. Explicación

En este ejemplo, se crea una tupla que almacena coordenadas (x, y) y se accede a los valores individuales utilizando índices.

! Actividad Práctica:

Crea una tupla con las estaciones del año.
Intenta modificar un elemento de la tupla y observa el error que se produce.

13.4. Explicación de la Actividad:

Esta actividad permite a los participantes practicar la creación de tuplas y comprender la diferencia entre listas y tuplas en términos de inmutabilidad. Les ayuda a comprender cómo utilizar tuplas cuando necesitan almacenar datos que no deben cambiar.

14. Range

El tipo de dato range se utiliza para generar secuencias de números. Aprenderemos cómo utilizar range en Python para crear secuencias de números en rangos específicos.

14.1. Conceptos Clave:

14.1.1. range

Tipo de dato utilizado para generar secuencias de números en un rango.

14.1.2. Parámetros de range

Se pueden especificar el valor inicial, valor final y paso de la secuencia.

14.1.3. Conversión a Listas

Es posible convertir un objeto range en una lista utilizando la función list().

14.2. Ejemplo

```
# Generación de secuencias de números
secuencia1 = range(5)          # 0, 1, 2, 3, 4
secuencia2 = range(2, 10)      # 2, 3, 4, 5, 6, 7, 8, 9
secuencia3 = range(1, 11, 2)   # 1, 3, 5, 7, 9

# Conversión a lista
lista_secuencia1 = list(secuencia1)
```

14.3. Explicación

En este ejemplo, se utilizan diferentes valores para crear secuencias de números utilizando el tipo de dato range.

La función list() se utiliza para convertir una secuencia de range en una lista.

! Actividad Práctica:

Crea una secuencia de números del 10 al 20 con un paso de 2.
Convierte la secuencia de números en una lista y muestra los elementos en la consola.

14.4. Explicación de la Actividad:

Esta actividad permite a los participantes practicar la creación de secuencias de números utilizando range y cómo convertirlas en listas para trabajar con los elementos individualmente. Les ayuda a comprender cómo generar secuencias de números en diferentes rangos.

15. Diccionarios

Los diccionarios son estructuras de datos que permiten almacenar pares clave-valor. Aprenderemos cómo crear y trabajar con diccionarios en Python.

15.1. Conceptos Clave:

15.1.1. Diccionarios

Estructuras de datos que almacenan pares clave-valor.

15.1.2. Claves

Son los nombres o etiquetas utilizados para acceder a los valores en el diccionario.

15.1.3. Valores

Son los datos asociados a cada clave en el diccionario.

15.2. Ejemplo

```
# Creación de un diccionario
persona = {
    "nombre": "Juan",
    "edad": 30,
    "ciudad": "México"
}

# Acceso a valores utilizando claves
nombre = persona["nombre"]
edad = persona["edad"]
```

15.3. Explicación

En este ejemplo, se crea un diccionario que almacena información de una persona, como nombre, edad y ciudad.

Se accede a los valores del diccionario utilizando las claves correspondientes.

> ! Actividad Práctica
>
> Crea un diccionario que almacene información de tus libros favoritos, incluyendo título y autor.
> Accede a los valores del diccionario utilizando las claves y muestra la información en la consola.

15.4. Explicación de la Actividad:

Esta actividad permite a los participantes practicar la creación de diccionarios y acceder a los valores utilizando las claves. Les ayuda a comprender cómo organizar datos en pares clave-valor y cómo acceder a la información de manera eficiente.

16. Booleanos

Los booleanos son un tipo de dato que puede tener dos valores: True (verdadero) o False (falso). Aprenderemos cómo trabajar con booleanos en Python y cómo utilizarlos en expresiones lógicas.

16.1. Conceptos Clave:

16.1.1. Booleanos

Tipo de dato que representa valores de verdad (True o False).

Expresiones Lógicas: Combinaciones de valores booleanos utilizando operadores lógicos como and, or y not.

16.2. Ejemplo

```
# Variables booleanas
es_mayor_de_edad = True
tiene_tarjeta = False

# Expresiones lógicas
puede_ingresar = es_mayor_de_edad and tiene_tarjeta
```

16.3. Explicación

En este ejemplo, se utilizan variables booleanas para representar si alguien es mayor de edad y si tiene una tarjeta.

Se utiliza una expresión lógica para evaluar si alguien puede ingresar basado en ambas condiciones.

! Actividad Práctica:

Crea variables booleanas que representen si tienes una mascota y si te gusta el deporte.
Utiliza expresiones lógicas para determinar si puedes llevar a tu mascota a un lugar que requiere tu atención durante un partido de tu deporte favorito.

16.4. Explicación de la Actividad:

Esta actividad permite a los participantes practicar el uso de variables booleanas y expresiones lógicas para tomar decisiones basadas en condiciones booleanas. Les ayuda a comprender cómo trabajar con valores de verdad y cómo utilizarlos para evaluar situaciones en el código.

Lección V.

Unidad 5: Control de Flujo

17. Introducción a If

El control de flujo es fundamental en la programación para tomar decisiones basadas en condiciones. Aprenderemos cómo utilizar la estructura if para ejecutar diferentes bloques de código según una condición.

17.1. Conceptos Clave:

17.1.1. Control de Flujo

Manejo de la ejecución del código basado en condiciones.

17.1.2. Estructura if

Permite ejecutar un bloque de código si una condición es verdadera.

17.1.3. Bloque de Código

Conjunto de instrucciones que se ejecutan si la condición es verdadera.

17.2. Ejemplo:

```
edad = 18

if edad >= 18:
    print("Eres mayor de edad.")
```

17.3. Explicación:

En este ejemplo, se utiliza la estructura if para verificar si la variable “edad” es mayor o igual a 18.

Si la condición es verdadera, se ejecuta el bloque de código que muestra un mensaje.

! Actividad Práctica:

Crea una variable que represente tu puntuación en un juego.
Utiliza una estructura if para mostrar un mensaje diferente según si tu puntuación es mayor o igual a 100.

17.4. Explicación de la Actividad:

Esta actividad permite a los participantes practicar la utilización de la estructura if para tomar decisiones basadas en condiciones. Les ayuda a comprender cómo ejecutar diferentes bloques de código según la situación y cómo utilizar el control de flujo en sus programas.

18. If y Condicionales

En esta lección, aprenderemos cómo trabajar con múltiples condiciones utilizando la estructura if, elif y else. Esto permite ejecutar diferentes bloques de código según diferentes condiciones.

18.1. Conceptos Clave

18.1.1. Estructura elif

Permite verificar una condición adicional si la condición anterior es falsa.

18.1.2. Estructura else

Define un bloque de código que se ejecuta si todas las condiciones anteriores son falsas.

18.1.3. Anidación de Estructuras if

Es posible anidar múltiples estructuras if para manejar situaciones más complejas.

18.2. Ejemplo:

```
puntaje = 85

if puntaje >= 90:
    print("¡Excelente trabajo!")
elif puntaje >= 70:
    print("Buen trabajo.")
else:
    print("Necesitas mejorar.")
```

18.3. Explicación:

En este ejemplo, se utiliza la estructura if, elif y else para evaluar diferentes rangos de puntajes y mostrar mensajes correspondientes.

! Actividad Práctica:

Crea una variable que represente tu calificación en un examen.
Utiliza una estructura if, elif y else para mostrar mensajes diferentes según la calificación obtenida.

18.4. Explicación de la Actividad:

Esta actividad permite a los participantes practicar el uso de la estructura if, elif y else para manejar múltiples condiciones y decisiones en sus programas. Les ayuda a comprender cómo ejecutar diferentes bloques de código en función de los resultados de las pruebas.

19. If, elif y else

En esta lección, continuaremos trabajando con la estructura if, elif y else, pero esta vez exploraremos cómo anidar estas estructuras para manejar situaciones más complejas.

19.1. Conceptos Clave:

19.1.1. Anidación de Estructuras

Posibilidad de colocar una estructura if, elif y else dentro de otra.

19.1.2. Jerarquía de Condiciones

Se evalúan las condiciones de manera secuencial y se ejecuta el primer bloque de código correspondiente a una condición verdadera.

19.2. Ejemplo

```
edad = 16
permiso_padres = True

if edad >= 18:
    print("Eres mayor de edad.")
else:
    if permiso_padres:
        print("Eres menor de edad pero tienes permiso de tus padres.")
    else:
        print("Eres menor de edad y no tienes permiso de tus padres.")
```

19.3. Explicación:

En este ejemplo, se anidan estructuras if para manejar diferentes casos basados en la edad y el permiso de los padres.

! Actividad Práctica:

Crea una variable que represente si un usuario está registrado en un sitio web.
Si el usuario está registrado, muestra un mensaje de bienvenida. Si no está registrado, muestra un mensaje que lo invite a registrarse.

19.4. Explicación de la Actividad:

Esta actividad permite a los participantes practicar la anidación de estructuras if, elif y else para manejar situaciones con múltiples condiciones. Les ayuda a comprender cómo trabajar con jerarquías de condiciones y cómo manejar casos más complejos en sus programas.

20. And y Or

En esta lección, exploraremos los operadores lógicos and y or, que permiten combinar condiciones para crear expresiones más complejas en las estructuras if, elif y else.

20.1. Conceptos Clave:

20.1.1. Operador and

Retorna True si ambas condiciones son verdaderas.

20.1.2. Operador or

Retorna True si al menos una de las condiciones es verdadera.

20.1.3. Combinación de Condiciones

Los operadores and y or permiten combinar múltiples condiciones en una sola expresión.

20.2. Ejemplo:

```
edad = 20
tiene_permiso = True

if edad >= 18 and tiene_permiso:
    print("Puedes ingresar.")
else:
    print("No puedes ingresar.")
```

20.3. Explicación:

En este ejemplo, se utiliza el operador and para evaluar si la edad es mayor o igual a 18 y si el usuario tiene permiso.

Si ambas condiciones son verdaderas, se permite el ingreso.

! Actividad Práctica:

Crea dos variables que representen si un usuario tiene una cuenta premium y si su suscripción está activa.
Utiliza una estructura if y el operador and para determinar si el usuario tiene acceso premium.

20.4. Explicación de la Actividad:

Esta actividad permite a los participantes practicar la combinación de condiciones utilizando los operadores and y or. Les ayuda a comprender cómo crear expresiones más complejas para tomar decisiones basadas en múltiples condiciones en sus programas.

21. Introducción a While

En esta lección, introduciremos la estructura while, que permite ejecutar un bloque de código repetidamente mientras se cumpla una condición.

21.1. Conceptos Clave:

21.1.1. Estructura while

Permite ejecutar un bloque de código mientras una condición sea verdadera.

21.1.2. Condición

La condición se verifica antes de cada iteración. Si es verdadera, se ejecuta el bloque de código.

21.2. Ejemplo:

```
contador = 0

while contador < 5:
    print("Contador:", contador)
    contador += 1
```

21.3. Explicación:

En este ejemplo, se utiliza la estructura while para imprimir el valor del contador mientras sea menor que 5.

Después de cada iteración, se incrementa el contador en 1.

::: {.callout-important} ### Actividad Práctica:

Crea una variable que represente la cantidad de intentos de un usuario para ingresar una contraseña correcta.

Utiliza una estructura while para pedir al usuario que ingrese su contraseña hasta que lo haga correctamente.

21.4. Explicación de la Actividad:

Esta actividad permite a los participantes practicar el uso de la estructura while para crear bucles que se ejecutan repetidamente mientras se cumple una condición. Les ayuda a comprender cómo implementar lógica de repetición en sus programas.

22. While loop

En esta lección, profundizaremos en el uso de la estructura while para crear bucles que se ejecutan repetidamente mientras se cumpla una condición, y aprenderemos a utilizar la sentencia break para salir de un bucle.

22.1. Conceptos Clave:

22.1.1. Sentencia break:

Se utiliza para salir de un bucle antes de que la condición sea falsa.

22.1.2. Bucles Infinitos:

Si no se maneja adecuadamente, un bucle while puede ejecutarse infinitamente.

22.2. Ejemplo:

```
contador = 0

while True:
    print("Contador:", contador)
    contador += 1
    if contador >= 5:
        break
```

22.3. Explicación:

En este ejemplo, se utiliza un bucle while que se ejecuta infinitamente.

Se utiliza la sentencia break para salir del bucle cuando el contador llega a 5.

> ! Actividad Práctica:
>
> Crea un bucle while que pida al usuario ingresar un número positivo menor que 10. Utiliza la sentencia break para salir del bucle una vez que el usuario ingrese un número válido.

22.4. Explicación de la Actividad:

Esta actividad permite a los participantes practicar el uso de la sentencia break para controlar la ejecución de un bucle while y evitar bucles infinitos. Les ayuda a comprender cómo manejar situaciones en las que es necesario salir de un bucle antes de que la condición sea falsa.

23. While, break y continue

En esta lección, continuaremos explorando cómo trabajar con la estructura while y aprenderemos a utilizar la sentencia continue para saltar a la siguiente iteración del bucle.

23.1. Conceptos Clave:

23.1.1. Sentencia continue

Se utiliza para saltar a la siguiente iteración del bucle sin ejecutar el resto del código en esa iteración.

23.1.2. Saltar Iteraciones

La sentencia continue permite omitir ciertas iteraciones basadas en una condición.

23.2. Ejemplo

```
contador = 0

while contador < 5:
    contador += 1
    if contador == 3:
        continue
    print("Contador:", contador)
```

23.3. Explicación:

En este ejemplo, se utiliza un bucle while para imprimir el valor del contador.

Se utiliza la sentencia continue para omitir la iteración cuando el contador es igual a 3.

> ! Actividad Práctica:
>
> Crea un bucle while que imprima los números del 1 al 10, pero omite la impresión del número 5.

Utiliza la sentencia continue para lograr esto.

23.4. Explicación de la Actividad

Esta actividad permite a los participantes practicar el uso de la sentencia continue para omitir iteraciones específicas en un bucle while. Les ayuda a comprender cómo controlar la ejecución de un bucle y realizar acciones selectivas en cada iteración.

24. For Loop

En esta lección, aprenderemos sobre el bucle for, que se utiliza para iterar sobre secuencias como listas, tuplas o cadenas de texto.

24.1. Conceptos Clave:

24.1.1. Bucle for

Se utiliza para recorrer elementos en una secuencia uno por uno.

24.1.2. Iteración

En cada iteración del bucle, se ejecuta un bloque de código con un valor diferente de la secuencia.

24.1.3. range() con Bucles for

Se puede utilizar la función range() para generar una secuencia de números y recorrerla con un bucle for.

24.2. Ejemplo:

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

for fruta in frutas:
    print("Me gusta", fruta)
```

24.3. Explicación:

En este ejemplo, se utiliza un bucle for para recorrer una lista de frutas.

En cada iteración, se asigna el valor actual de la lista a la variable fruta.

! Actividad Práctica:

Crea una lista de colores.
Utiliza un bucle for para imprimir cada color en la lista precedido por la palabra "Color:".

24.4. Explicación de la Actividad:

Esta actividad permite a los participantes practicar el uso del bucle for para iterar sobre elementos en una secuencia. Les ayuda a comprender cómo trabajar con bucles para recorrer listas y otros tipos de secuencias en sus programas.

Lección VI.

Unidad 6: Funciones

25. Introducción a Funciones

En esta lección, exploraremos el concepto de funciones en Python. Aprenderemos cómo definir funciones, pasar argumentos y cómo retornar valores.

25.1. Conceptos Clave:

25.1.1. Funciones.

Bloques de código reutilizables que realizan una tarea específica.

25.1.2. Definición de Funciones

Se utiliza la palabra clave def para definir una función.

25.1.3. Argumentos

Valores que se pasan a una función para que trabaje con ellos.

25.2. Retorno de Valores

Una función puede retornar un valor utilizando la palabra clave return.

25.3. Ejemplo:

```
def saludar(nombre):
    return "Hola, " + nombre

mensaje = saludar("Juan")
print(mensaje)
```

25.4. Explicación:

En este ejemplo, se define una función llamada saludar que toma un argumento nombre.

La función retorna un saludo personalizado utilizando el argumento.

El resultado se asigna a la variable mensaje y se muestra en la consola.

> ! Actividad Práctica:
>
> Crea una función llamada calcular_cuadrado que reciba un número como argumento y retorne el cuadrado de ese número.
> Utiliza la función para calcular el cuadrado de un número y mostrarlo en la consola.

25.5. Explicación de la Actividad:

Esta actividad permite a los participantes practicar la creación y uso de funciones en Python. Les ayuda a comprender cómo definir funciones, pasar argumentos y cómo trabajar con valores retornados por las funciones.

26. Recursividad

En esta lección, exploraremos el concepto de recursividad, que es cuando una función se llama a sí misma para resolver un problema. Aprenderemos cómo implementar funciones recursivas y cuándo es apropiado usarlas.

26.1. Conceptos Clave:

26.1.1. Recursividad

Técnica en la que una función se llama a sí misma para resolver un problema.

26.1.2. Caso Base

Condición que indica cuándo la recursión debe detenerse.

26.1.3. Caso Recursivo

Cómo se divide el problema en partes más pequeñas en cada llamada recursiva.

26.2. Ejemplo:

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)

resultado = factorial(5)
print(resultado)
```

26.3. Explicación:

En este ejemplo, se define una función recursiva llamada factorial para calcular el factorial de un número.

La función utiliza un caso base (cuando n es 0) y un caso recursivo (llamando a la función con n - 1).

> ! Actividad Práctica:
>
> Crea una función recursiva llamada potencia que calcule la potencia de un número base elevado a un exponente.
> Utiliza la función para calcular 2^3 y muestra el resultado en la consola.

26.4. Explicación de la Actividad:

Esta actividad permite a los participantes practicar la implementación de funciones recursivas y comprender cómo se dividen los problemas en partes más pequeñas para resolverlos. Les ayuda a comprender cómo aplicar la recursividad de manera efectiva en la solución de problemas.

Lección VII.

Unidad 7: Objetos, Clases y Herencia

27. Introducción

En esta lección, exploraremos el concepto de programación orientada a objetos (POO). Aprenderemos sobre objetos, clases y cómo la POO nos permite organizar y estructurar nuestro código de manera más eficiente.

27.1. Conceptos Clave:

27.1.1. Programación Orientada a Objetos (POO)

Paradigma de programación que se basa en el uso de objetos y clases.

27.1.2. Objetos

Instancias de clases que representan entidades en el mundo real.

27.1.3. Clases

Plantillas o moldes que definen la estructura y el comportamiento de los objetos.

27.1.4. Atributos

Características o propiedades de un objeto.

27.1.5. Métodos

Funciones que definen el comportamiento de un objeto.

27.2. Ejemplo:

```
class Persona:
    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad

    def saludar(self):
```

```
        print(f"Hola, mi nombre es {self.nombre} y tengo {self.edad} años.")

persona1 = Persona("Juan", 25)
persona1.saludar()
```

27.3. Explicación:

En este ejemplo, se define una clase llamada Persona con un constructor (**init**) que inicializa atributos.

La clase tiene un método llamado saludar que muestra un mensaje con el nombre y edad del objeto.

> ! Actividad Práctica:
>
> Crea una clase llamada Libro con atributos titulo y autor, y un método mostrar_info que imprima los atributos.
> Crea una instancia de la clase Libro y llama al método mostrar_info.

27.4. Explicación de la Actividad:

Esta actividad permite a los participantes practicar la definición de clases y la creación de objetos. Les ayuda a comprender cómo la POO nos permite modelar entidades y organizar el código de manera más estructurada y eficiente.

28. Objetos y Clases

En esta lección, continuaremos explorando los conceptos de objetos y clases en la programación orientada a objetos. Aprenderemos cómo crear múltiples objetos a partir de una misma clase y cómo trabajar con sus atributos y métodos.

28.1. Conceptos Clave:

28.1.1. Instancias de Clase

Cuando se crea un objeto a partir de una clase, se crea una instancia de esa clase.

28.1.2. Atributos de Instancia:

Características específicas de un objeto que se almacenan como variables en la instancia.

28.1.3. Métodos de Instancia

Funciones definidas en la clase que operan en los atributos de la instancia.

28.2. Ejemplo:

```
class Perro:
    def __init__(self, nombre, raza):
        self.nombre = nombre
        self.raza = raza

    def ladrar(self):
        print(f"{self.nombre} está ladrando.")

perro1 = Perro("Max", "Labrador")
perro2 = Perro("Buddy", "Chihuahua")

perro1.ladrar()
perro2.ladrar()
```

28.3. Explicación:

En este ejemplo, se define una clase Perro con un constructor y un método ladrar.

Se crean dos objetos (perro1 y perro2) a partir de la misma clase y se les asignan diferentes valores para sus atributos.

Los métodos de instancia son llamados en cada objeto para realizar acciones específicas.

! Actividad Práctica:

Crea una clase Rectangulo con atributos ancho y alto, y un método calcular_area que calcule y retorne el área del rectángulo.
Crea dos instancias de la clase Rectangulo con diferentes valores de ancho y alto, y llama al método calcular_area en cada una.

28.4. Explicación de la Actividad:

Esta actividad permite a los participantes practicar la creación de instancias de clase y trabajar con sus atributos y métodos. Les ayuda a comprender cómo cada objeto puede tener valores diferentes para sus atributos y cómo ejecutar acciones específicas en cada objeto.

29. Métodos

En esta lección, profundizaremos en el concepto de métodos en la programación orientada a objetos. Aprenderemos cómo definir y utilizar métodos en una clase, y cómo acceder a los atributos de instancia dentro de los métodos.

29.1. Conceptos Clave:

29.1.1. Métodos de Clase

Funciones definidas dentro de una clase que operan en los atributos de instancia.

29.1.2. Acceso a Atributos

Dentro de un método, se puede acceder a los atributos de instancia utilizando self.atributo.

29.2. Ejemplo:

```
class Circulo:
    def __init__(self, radio):
        self.radio = radio

    def calcular_area(self):
        area = 3.14 * self.radio ** 2
        return area

circulo1 = Circulo(5)
area_circulo = circulo1.calcular_area()
print("Área del círculo:", area_circulo)
```

29.3. Explicación:

En este ejemplo, se define una clase Circulo con un constructor y un método calcular_area.

Dentro del método, se accede al atributo de instancia radio utilizando self.radio para calcular el área.

> ! Actividad Práctica:
>
> Crea una clase Triangulo con atributos base y altura, y un método calcular_area que calcule y retorne el área del triángulo.
> Crea una instancia de la clase Triangulo y llama al método calcular_area para calcular el área.

29.4. Explicación de la Actividad:

Esta actividad permite a los participantes practicar la definición y uso de métodos en una clase. Les ayuda a comprender cómo trabajar con atributos de instancia dentro de los métodos y cómo implementar lógica específica para cada objeto.

30. Self, Eliminar Propiedades y Objetos

En esta lección, aprenderemos más sobre el uso de self en los métodos de clase. También exploraremos cómo eliminar atributos de instancia y objetos en Python.

30.1. Conceptos Clave:

30.1.1. self

Referencia al objeto actual en un método de clase.

30.1.2. Eliminar Atributos

Se puede usar la palabra clave del para eliminar un atributo de instancia.

30.1.3. Eliminar Objetos

Se utiliza la función del para eliminar un objeto y liberar memoria.

30.2. Ejemplo:

```
class Coche:
    def __init__(self, marca, modelo):
        self.marca = marca
        self.modelo = modelo

    def mostrar_info(self):
        print(f"Coche {self.marca} {self.modelo}")

coche1 = Coche("Toyota", "Corolla")
coche1.mostrar_info()

# Eliminar el atributo 'modelo'
del coche1.modelo

# Intentar acceder al atributo eliminado generará un error
# print(coche1.modelo)
```

30.3. Explicación:

En este ejemplo, se define una clase Coche con un constructor y un método mostrar_info.

Se crea una instancia coche1 y se muestra su información. Luego, se elimina el atributo modelo utilizando del.

> ! Actividad Práctica:
>
> Crea una clase Estudiante con atributos nombre y edad, y un método mostrar_info para mostrar la información del estudiante.
> Crea una instancia de la clase Estudiante y llama al método mostrar_info.
> Utiliza del para eliminar el atributo nombre de la instancia y verifica el resultado.

30.4. Explicación de la Actividad:

Esta actividad permite a los participantes practicar el uso de self en los métodos de clase y cómo eliminar atributos de instancia. Les ayuda a comprender cómo trabajar con objetos y atributos, y cómo gestionar la memoria en Python.

31. Herencia

En esta lección, exploraremos el concepto de herencia en la programación orientada a objetos. Aprenderemos cómo crear clases que heredan atributos y métodos de una clase base.

31.1. Conceptos Clave:

31.1.1. Herencia

Mecanismo que permite que una clase herede atributos y métodos de otra clase base.

31.1.2. Clase Padre (o Base)

La clase de la que se heredan atributos y métodos.

31.1.3. Clase Hija (o Derivada)

La clase que hereda de la clase base.

31.2. Ejemplo:

```
class Animal:
    def __init__(self, nombre):
        self.nombre = nombre

    def saludar(self):
        print(f"{self.nombre} saluda")

class Perro(Animal):
    def ladrar(self):
        print(f"{self.nombre} está ladrando")

perro1 = Perro("Buddy")
perro1.saludar()
perro1.ladrar()
```

31.2.1. Explicación:

En este ejemplo, se define una clase base Animal con un constructor y un método saludar.

Se define una clase derivada Perro que hereda de Animal y agrega un método adicional ladrar.

Se crea una instancia perro1 de la clase Perro y se llama a sus métodos.

! Actividad Práctica:

Crea una clase Figura con un atributo color y un método mostrar_color para mostrar el color.
Crea una clase derivada Circulo que herede de Figura y agrega un atributo radio y un método calcular_area para calcular el área del círculo.
Crea una instancia de la clase Circulo, establece su color y calcula el área.

31.3. Explicación de la Actividad:

Esta actividad permite a los participantes practicar la creación de clases derivadas y la herencia de atributos y métodos. Les ayuda a comprender cómo utilizar la herencia para reutilizar código y extender funcionalidades en las clases derivadas.

Lección VIII.

Unidad 8: Módulos

32. Introducción

En esta lección, exploraremos cómo trabajar con módulos en Python. Aprenderemos cómo dividir nuestro código en módulos reutilizables y cómo importarlos en otros programas.

32.1. Conceptos Clave:

32.1.1. Módulos

Archivos que contienen código Python y se utilizan para organizar y reutilizar funciones, clases y variables.

32.1.2. Importar Módulos

Se utiliza la palabra clave import para cargar un módulo en un programa.

32.1.3. Usar Funciones y Clases

Después de importar un módulo, sus funciones y clases pueden ser utilizadas como si estuvieran definidas en el mismo archivo.

32.2. Ejemplo:

```
# En el archivo calculadora.py
def suma(a, b):
    return a + b

# En otro archivo
import calculadora

resultado = calculadora.suma(3, 5)
print("Resultado:", resultado)
```

32.3. Explicación:

En este ejemplo, se define una función suma en el módulo calculadora.py.

En otro archivo, se importa el módulo calculadora utilizando import y se utiliza la función suma del módulo.

> ! Actividad Práctica:
>
> Crea un módulo llamado matematicas con una función multiplicacion que multiplique dos números.
> Importa el módulo en otro archivo y utiliza la función multiplicacion para calcular el producto de dos números.

32.4. Explicación de la Actividad:

Esta actividad permite a los participantes practicar la creación y uso de módulos en Python. Les ayuda a comprender cómo organizar su código en módulos reutilizables y cómo importar funciones y clases desde otros archivos.

33. Creando Nuestro Primer Módulo

En esta lección, aprenderemos a crear nuestro propio módulo en Python. Crearemos un módulo que contenga funciones y clases para realizar operaciones matemáticas básicas.

33.1. Pasos para Crear un Módulo:

Crea un archivo de Python con la extensión .py.

Define funciones y clases en el archivo.

Guarda el archivo en una ubicación accesible.

33.2. Ejemplo:

```
# En el archivo operaciones.py
def suma(a, b):
    return a + b

def resta(a, b):
    return a - b

class Calculadora:
    def multiplicacion(self, a, b):
        return a * b
```

33.3. Explicación:

En este ejemplo, se crea un módulo llamado operaciones.py.

Se define una función suma y una función resta, junto con una clase Calculadora que tiene un método multiplicacion.

> ! Actividad Práctica:
>
> Crea un módulo llamado geometria con funciones para calcular el área de un círculo y el perímetro de un cuadrado.
> En otro archivo, importa el módulo geometria y utiliza las funciones para realizar cálculos geométricos.

33.4. Explicación de la Actividad:

Esta actividad permite a los participantes practicar la creación de módulos con funciones y clases. Les ayuda a comprender cómo organizar diferentes funcionalidades en módulos separados y cómo importar esas funcionalidades en otros archivos.

34. Renombrando Módulos

En esta lección, aprenderemos cómo renombrar módulos al importarlos y cómo seleccionar elementos específicos para importar. Esto nos permitirá tener un mayor control sobre los nombres y las funcionalidades que utilizamos en nuestro código.

34.1. Renombrando Módulos al Importar:

```
import modulo_largo as ml
```

34.2. Seleccionando Elementos Específicos para Importar:

```
from modulo import funcion1, funcion2
```

34.3. Ejemplo - Renombrando Módulos:

```
import calculadora as calc

resultado = calc.suma(3, 4)
```

34.4. Ejemplo - Seleccionando Elementos Específicos:

```
from operaciones import resta, Calculadora

resultado = resta(10, 5)
```

:::{.callout-important} ### Actividad Práctica:

Renombra el módulo geometria como geo al importarlo en otro archivo.

Importa solo la función para calcular el área de un círculo y calcula el área de un círculo con radio 5.

34.5. Explicación de la Actividad:

Esta actividad permite a los participantes practicar cómo renombrar módulos al importarlos y cómo seleccionar funciones específicas para importar. Les ayuda a comprender cómo personalizar los nombres de los módulos y cómo importar solo las funcionalidades que necesitan en su código.

35. Seleccionando lo Importado y Pip

En esta lección, continuaremos explorando cómo seleccionar elementos específicos para importar y aprenderemos sobre pip, la herramienta de gestión de paquetes de Python. pip nos permite instalar y gestionar paquetes externos que contienen funcionalidades adicionales para nuestros programas.

35.1. Seleccionando Elementos Específicos para Importar:

```
from modulo import funcion1, funcion2
```

35.1.1. Usando Pip:

```
pip install nombre_del_paquete: #Instalar un paquete.
pip uninstall nombre_del_paquete: #Desinstalar un paquete.
```

35.2. Ejemplo - Instalando un Paquete con Pip:

```
pip install requests
```

! Actividad Práctica:

Utiliza pip para instalar el paquete matplotlib, que se utiliza para trazar gráficos en Python.
En tu archivo de código, importa la función plot de matplotlib.pyplot y crea un gráfico simple.

35.3. Explicación de la Actividad:

Esta actividad permite a los participantes practicar cómo utilizar pip para instalar paquetes externos y cómo importar funcionalidades específicas de esos paquetes en su código. Les ayuda a comprender cómo expandir las capacidades de Python utilizando bibliotecas externas.

Lección IX.

Unidad 9: Introducción a Bases de Datos

36. Introducción a Bases de Datos

En esta lección, exploraremos el concepto de bases de datos y su importancia en el desarrollo de aplicaciones. Aprenderemos cómo las bases de datos nos permiten almacenar y recuperar información de manera eficiente.

36.1. Conceptos Clave:

36.1.1. Base de Datos

Colección organizada de datos almacenados en formato estructurado.

36.1.2. Sistemas de Gestión de Bases de Datos (DBMS)

Software que administra y gestiona una base de datos.

36.1.3. Beneficios de las Bases de Datos

Almacenamiento eficiente, acceso rápido y seguridad de datos.

36.2. Ejemplo:

```
# Ejemplo de una tabla 'usuarios' en una base de datos
| id | nombre   | edad | email           |
|----|----------|------|-----------------|
| 1  | Juan     | 25   | juan@email.com |
| 2  | María    | 30   | maria@email.com|
```

36.3. Explicación:

En este ejemplo, se muestra una tabla ficticia de una base de datos llamada 'usuarios'.

La tabla contiene filas que representan registros de usuarios con diferentes atributos.

! Actividad Práctica:

Investiga y elige un Sistema de Gestión de Bases de Datos (DBMS) para utilizar en el curso.
Explica por qué es importante utilizar bases de datos en el desarrollo de aplicaciones.

36.4. Explicación de la Actividad:

Esta actividad permite a los participantes comprender la importancia de las bases de datos en el desarrollo de aplicaciones y seleccionar una opción adecuada de DBMS para usar en el curso. Les ayuda a familiarizarse con el concepto de bases de datos y sus beneficios.

37. Introducción a PostgreSQL

En esta lección, nos centraremos en PostgreSQL, un Sistema de Gestión de Bases de Datos Relacionales (RDBMS) de código abierto. Aprenderemos cómo instalar PostgreSQL y cómo realizar operaciones básicas en una base de datos.

37.0.1. Instalación de PostgreSQL:

Descargar e instalar PostgreSQL desde el sitio oficial.

Configurar contraseña para el usuario 'postgres'.

37.0.2. Operaciones Básicas en PostgreSQL:

37.0.3. Crear una base de datos:

```
CREATE DATABASE nombre;
```

37.0.4. Conectar a una base de datos:

```
\c nombre;
```

37.0.5. Crear una tabla:

```
CREATE TABLE tabla (columna1 tipo, columna2 tipo);
```

37.0.6. Insertar registros:

```
INSERT INTO tabla (columna1, columna2) VALUES (valor1, valor2);
```

37.0.7. Consultar registros:

```
SELECT * FROM tabla;
```

37.0.8. Actualizar registros:

```
UPDATE tabla SET columna1 = valor WHERE condicion;**
```

37.0.9. Eliminar registros:

```
DELETE FROM tabla WHERE condicion;
```

37.1. Ejemplo - Creación de una Tabla en PostgreSQL:

```
CREATE TABLE estudiantes (
    id SERIAL PRIMARY KEY,
    nombre VARCHAR(100),
    edad INTEGER
);
```

37.1.1. Actividad Práctica:

Instala PostgreSQL en tu entorno.

Crea una base de datos llamada 'universidad'.

Crea una tabla 'alumnos' con las columnas 'id', 'nombre' y 'edad'.

Inserta al menos dos registros en la tabla.

Realiza una consulta para obtener todos los registros de la tabla 'alumnos'.

37.2. Explicación de la Actividad:

Esta actividad permite a los participantes familiarizarse con la instalación de PostgreSQL y realizar operaciones básicas de creación de base de datos, creación de tablas, inserción y consulta de registros. Les ayuda a adquirir experiencia práctica en la gestión de bases de datos utilizando PostgreSQL.

38. Introducción a MongoDB

En esta lección, nos centraremos en MongoDB, una base de datos NoSQL de código abierto. Aprenderemos cómo instalar MongoDB y cómo realizar operaciones básicas en una base de datos NoSQL.

38.1. Instalación de MongoDB:

Descargar e instalar MongoDB desde el sitio oficial.

Configurar directorio de datos y logs.

38.2. Operaciones Básicas en MongoDB:

38.2.1. Crear una base de datos:

```
use nombre;
```

38.2.2. Crear una colección (tabla):

```
db.createCollection("coleccion");
```

38.2.3. Insertar documentos (registros):

```
db.coleccion.insert({ campo1: valor1, campo2: valor2 });
```

38.2.4. Consultar documentos:

```
db.coleccion.find();
```

38.2.5. Actualizar documentos:

```
db.coleccion.update({ campo: valor }, { $set: { campo_actualizado: nuevo_valor } });
```

38.2.6. Eliminar documentos:

```
db.coleccion.remove({ campo: valor });
```

38.3. Ejemplo - Creación de una Colección en MongoDB:

```
use tienda;
db.createCollection("productos");
```

! Actividad Práctica:

Instala MongoDB en tu entorno.
Crea una base de datos llamada 'blog'.
Crea una colección 'articulos'.
Inserta al menos dos documentos (artículos) en la colección.
Realiza una consulta para obtener todos los documentos de la colección 'articulos'.

38.4. Explicación de la Actividad:

Esta actividad permite a los participantes familiarizarse con la instalación de MongoDB y realizar operaciones básicas en una base de datos NoSQL. Les ayuda a adquirir experiencia práctica en la gestión de datos en MongoDB y a comprender las diferencias entre bases de datos SQL y NoSQL.

Lección X.

Unidad 10: Operaciones Básicas en Bases de Datos

39. Introducción e Instalación

En esta lección, nos centraremos en realizar operaciones básicas en bases de datos utilizando diferentes sistemas de gestión: MySQL, PostgreSQL y MongoDB. Aprenderemos cómo realizar la instalación de estos sistemas y cómo conectarnos a las bases de datos.

39.1. Instalación de MySQL:

Descargar e instalar MySQL desde el sitio oficial.

Configurar contraseña para el usuario 'root'.

39.2. Instalación de PostgreSQL:

Descargar e instalar PostgreSQL desde el sitio oficial.

Configurar contraseña para el usuario 'postgres'.

39.3. Instalación de MongoDB:

Descargar e instalar MongoDB desde el sitio oficial.

Configurar directorio de datos y logs.

39.4. Conexión a la Base de Datos:

39.4.1. MySQL y PostgreSQL:

Usar bibliotecas como mysql-connector-python o psycopg2 para conectarse y realizar operaciones.

39.4.2. MongoDB:

Usar la biblioteca pymongo para conectarse y realizar operaciones.

39.5. Ejemplo - Conexión a MySQL:

```
import mysql.connector

# Conexión a la base de datos
conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password="contraseña",
    database="basededatos"
)
```

39.6. Ejemplo - Conexión a MongoDB:

```
import pymongo

# Conexión al servidor MongoDB
client = pymongo.MongoClient("mongodb://localhost:27017/")
```

! Actividad Práctica:

Instala MySQL, PostgreSQL y MongoDB en tu entorno. Crea una base de datos en cada uno de los sistemas. Conéctate a cada una de las bases de datos utilizando las bibliotecas adecuadas. Realiza una consulta de prueba en cada sistema para verificar la conexión.

39.7. Explicación de la Actividad:

Esta actividad permite a los participantes adquirir experiencia práctica en la instalación de diferentes sistemas de bases de datos y en la conexión a estas bases de datos utilizando las bibliotecas correspondientes. Les ayuda a comprender cómo establecer una conexión exitosa y cómo preparar el entorno para las operaciones futuras en bases de datos.

40. Bases de Datos en MySQL

En esta lección, aprenderemos a realizar operaciones básicas en una base de datos MySQL, como crear y eliminar tablas, insertar registros y realizar consultas.

40.1. Operaciones en MySQL:

40.1.1. Crear una tabla:

```
CREATE TABLE nombre (columna1 tipo, columna2 tipo);
```

40.1.2. Insertar registros:

```
INSERT INTO nombre (columna1, columna2) VALUES (valor1, valor2);
```

40.1.3. Consultar registros:

```
SELECT * FROM nombre;
```

40.1.4. Actualizar registros:

```
UPDATE nombre SET columna = valor WHERE condicion;
```

40.1.5. Eliminar registros:

```
DELETE FROM nombre WHERE condicion;
```

40.1.6. Eliminar tabla:

```
DROP TABLE nombre;
```

40.2. Ejemplo - Creación de una Tabla en MySQL:

```
CREATE TABLE empleados (
    id INT PRIMARY KEY,
    nombre VARCHAR(100),
    salario DECIMAL(10, 2)
);
```

! Actividad Práctica:

Conéctate a la base de datos MySQL.
Crea una tabla 'productos' con las columnas 'id', 'nombre' y 'precio'.
Inserta al menos dos registros en la tabla 'productos'.
Realiza una consulta para obtener todos los registros de la tabla 'productos'.

40.3. Explicación de la Actividad:

Esta actividad permite a los participantes aplicar los conocimientos adquiridos en la creación de tablas, inserción de registros y consultas en una base de datos MySQL. Les ayuda a ganar experiencia práctica en la manipulación de datos utilizando SQL en MySQL.

41. Crear y Eliminar Tablas en PostgreSQL

En esta lección, aprenderemos a realizar operaciones básicas en una base de datos PostgreSQL, como crear y eliminar tablas, insertar registros y realizar consultas.

41.1. Operaciones en PostgreSQL:

41.1.1. Crear una tabla:

```
CREATE TABLE nombre (columna1 tipo, columna2 tipo);
```

41.1.2. Insertar registros:

```
INSERT INTO nombre (columna1, columna2) VALUES (valor1, valor2);
```

41.1.3. Consultar registros:

```
SELECT * FROM nombre;
```

41.1.4. Actualizar registros:

```
UPDATE nombre SET columna = valor WHERE condicion;
```

41.1.5. Eliminar registros:

```
DELETE FROM nombre WHERE condicion;
```

41.1.6. Eliminar tabla:

```
DROP TABLE nombre;
```

41.2. Ejemplo - Creación de una Tabla en PostgreSQL:

```
CREATE TABLE empleados (
    id SERIAL PRIMARY KEY,
    nombre VARCHAR(100),
    salario DECIMAL(10, 2)
);
```

! Actividad Práctica

41.3. Conéctate a la base de datos PostgreSQL.

Crea una tabla 'clientes' con las columnas 'id', 'nombre' y 'email'.
Inserta al menos dos registros en la tabla 'clientes'.
Realiza una consulta para obtener todos los registros de la tabla 'clientes'.

41.4. Explicación de la Actividad:

Esta actividad permite a los participantes aplicar los conocimientos adquiridos en la creación de tablas, inserción de registros y consultas en una base de datos PostgreSQL. Les ayuda a ganar experiencia práctica en la manipulación de datos utilizando SQL en PostgreSQL.

42. Operaciones Básicas en MongoDB

En esta lección, aprenderemos a realizar operaciones básicas en una base de datos MongoDB, como insertar documentos, consultar documentos y actualizar documentos.

42.1. Operaciones en MongoDB:

42.1.1. Insertar documentos:

```
db.coleccion.insert({ campo1: valor1, campo2: valor2 });
```

42.1.2. Consultar documentos:

```
db.coleccion.find();
```

42.1.3. Actualizar documentos:

```
db.coleccion.update({ campo: valor }, { $set: { campo_actualizado: nuevo_valor } });
```

42.1.4. Eliminar documentos:

```
db.coleccion.remove({ campo: valor });
```

42.2. Ejemplo - Inserción de un Documento en MongoDB:

```
// Insertar un documento en la colección 'productos'
db.productos.insert({ nombre: "Camiseta", precio: 20 });
```

! Actividad Práctica:

Conéctate a la base de datos MongoDB.
Inserta al menos dos documentos en la colección 'productos'.
Realiza una consulta para obtener todos los documentos de la colección 'productos'.

Actualiza el precio de uno de los documentos en la colección.
Elimina un documento de la colección.

42.3. Explicación de la Actividad:

Esta actividad permite a los participantes aplicar los conocimientos adquiridos en la inserción, consulta, actualización y eliminación de documentos en una base de datos MongoDB. Les ayuda a ganar experiencia práctica en la manipulación de datos en una base de datos NoSQL.

Lección XI.

Unidad 11: ¿Cómo me amplío con Python?

43. Introducción a Data Science

En esta lección, exploraremos el emocionante campo de la Ciencia de Datos y cómo Python se ha convertido en una herramienta esencial en este ámbito. Aprenderemos qué es la Ciencia de Datos, su importancia y cómo Python se utiliza para analizar y visualizar datos.

43.1. Conceptos Clave:

43.1.1. Ciencia de Datos:

Proceso de extracción de conocimiento y perspectivas a partir de datos.

43.1.2. Uso de Python en Data Science:

Bibliotecas como NumPy, Pandas y Matplotlib.

43.1.3. Ejemplos de Aplicación:

Análisis de datos,

Visualización,

Aprendizaje Automático,

etc.

43.2. Ejemplo - Uso de Pandas para Análisis de Datos:

```
import pandas as pd

data = {
    'nombre': ['Juan', 'María', 'Pedro'],
    'edad': [25, 30, 28]
}

df = pd.DataFrame(data)
print(df)
```

! Actividad Práctica:

Investiga y elige un conjunto de datos disponible en línea.
Utiliza la biblioteca Pandas para cargar y analizar los datos.
Realiza un análisis simple, como calcular estadísticas descriptivas, en el conjunto de datos.

43.3. Explicación de la Actividad:

Esta actividad permite a los participantes explorar la aplicación de Python en el campo de la Ciencia de Datos. Les ayuda a comprender cómo utilizar bibliotecas como Pandas para analizar datos y extraer información útil.

44. Introducción a Django Framework

En esta lección, nos adentraremos en el mundo de Django, un popular framework de desarrollo web en Python. Aprenderemos qué es Django, cómo instalarlo y cómo crear una aplicación web básica utilizando este framework.

44.1. Qué es Django:

Django es un framework de desarrollo web de alto nivel y de código abierto.

Proporciona una estructura organizada para crear aplicaciones web de manera eficiente.

44.2. Instalación de Django:

44.2.1. Instalar Django utilizando pip:

```
pip install django
```

44.2.2. Verificar la instalación:

```
django-admin --version
```

44.3. Creación de una Aplicación Web Básica:

44.3.1. Crear un nuevo proyecto:

```
django-admin startproject proyecto .
```

44.3.2. Crear una nueva aplicación dentro del proyecto:

```
python manage.py startapp app
```

44.4. Ejemplo - Creación de una Página Web con Django:

```
# views.py
from django.http import HttpResponse

def hola_mundo(request):
    return HttpResponse("¡Hola, mundo!")
```

```
# urls.py
from django.urls import path
from . import views

urlpatterns = [
    path('hola/', views.hola_mundo, name='hola_mundo'),
]
```

! Actividad Práctica:

Instala Django en tu entorno.
Crea un proyecto llamado 'blog' y una aplicación llamada 'articulos'.
Crea una vista que muestre un mensaje de bienvenida en la página principal.
Configura una URL para acceder a la vista creada.

44.5. Explicación de la Actividad:

Esta actividad permite a los participantes experimentar con la creación de proyectos y aplicaciones utilizando Django. Les ayuda a comprender cómo estructurar una aplicación web utilizando este framework y cómo definir rutas y vistas para mostrar contenido en el navegador.

45. Introducción a FastAPI y PyScript

En esta lección, exploraremos FastAPI, un moderno framework de desarrollo web en Python, y PyScript, una herramienta que permite crear scripts de Python en un entorno interactivo. Aprenderemos cómo utilizar FastAPI para construir APIs rápidas y cómo utilizar PyScript para escribir y ejecutar scripts de manera interactiva.

45.1. Qué es FastAPI:

FastAPI es un framework de desarrollo web rápido (high-performance) basado en Python.

Permite construir APIs rápidas y seguras de manera sencilla.

45.2. Instalación de FastAPI:

45.2.1. Instalar FastAPI utilizando pip:

```
pip install fastapi
```

45.2.2. Instalar el servidor ASGI (por ejemplo, Uvicorn):

```
pip install uvicorn
```

45.3. Creación de una API Básica con FastAPI:

```
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def leer_raiz():
    return {"mensaje": "¡Hola desde FastAPI!"}
```

45.4. Uso de PyScript:

45.4.1. Instalar PyScript utilizando pip:

```
pip install pyscript
```

45.4.2. Ejecutar PyScript en un entorno interactivo:

```
pyscript
```

45.5. Ejemplo - Ejecución de PyScript:

```
a = 5
b = 10
a + b
15
```

> ! Actividad Práctica:
>
> Instala FastAPI en tu entorno.
> Crea una API con FastAPI que tenga al menos un endpoint para obtener información.
> Instala PyScript en tu entorno y realiza algunos cálculos y experimentos interactivos.

45.6. Explicación de la Actividad:

Esta actividad permite a los participantes experimentar con FastAPI y PyScript para crear una API básica y ejecutar scripts interactivos. Les ayuda a comprender cómo utilizar FastAPI para construir APIs de manera rápida y cómo utilizar PyScript para escribir y ejecutar código Python de manera interactiva en la consola.

Lección XII.

Unidad 12: Proyecto: API de Tareas con Django Rest Framework

46. Explicación del Proyecto

En este proyecto, construiremos una API utilizando Django Rest Framework para gestionar tareas. La API permitirá a los usuarios crear, actualizar, listar y eliminar tareas. Utilizaremos Django Rest Framework para definir los modelos, las vistas y las URL necesarias para interactuar con la API.

46.1. Qué se necesita conocer:

- Conocimientos básicos de Python.
- Familiaridad con Django y Django Rest Framework.
- Entorno de desarrollo configurado con Django y Django Rest Framework.

46.2. Estructura del Proyecto:

```
proyecto_api_tareas/
   api_tareas/
       migrations/
       templates/
       __init__.py
       admin.py
       apps.py
       models.py
       serializers.py
       tests.py
       views.py
   proyecto_api_tareas/
       __init__.py
       asgi.py
       settings.py
       urls.py
       wsgi.py
   db.sqlite3
   manage.py
```

46.3. Código:

```
#models.py:
from django.db import models

class Tarea(models.Model):
    titulo = models.CharField(max_length=100)
    descripcion = models.TextField()
    fecha_creacion = models.DateTimeField(auto_now_add=True)
    completada = models.BooleanField(default=False)

    def __str__(self):
        return self.titulo
```

```
#serializers.py:

from rest_framework import serializers
from .models import Tarea

class TareaSerializer(serializers.ModelSerializer):
    class Meta:
        model = Tarea
        fields = '__all__'
```

```
#views.py:
from rest_framework import viewsets
from .models import Tarea
from .serializers import TareaSerializer

class TareaViewSet(viewsets.ModelViewSet):
    queryset = Tarea.objects.all()
    serializer_class = TareaSerializer

    urls.py (api_tareas):
```

```
from rest_framework.routers import DefaultRouter
from .views import TareaViewSet

router = DefaultRouter()
router.register(r'tareas', TareaViewSet)

urlpatterns = router.urls
```

A continuación en el archivo **settings.py** agregar **'rest_framework'** y **'api_tareas'** en **INSTALLED_APPS**.

! Actividad Práctica:

Configura un proyecto Django y una aplicación llamada 'api_tareas'.
Define el modelo Tarea en models.py con los campos necesarios.
Crea un serializador en serializers.py para el modelo Tarea.
Implementa las vistas en views.py utilizando Django Rest Framework.
Configura las URLs en urls.py para las vistas de la API.
Migrar y ejecutar el servidor para probar la API utilizando el navegador o herramientas como Postman.

46.4. Explicación de la Actividad:

Este proyecto permite a los participantes aplicar los conocimientos adquiridos en Django y Django Rest Framework para crear una API de gestión de tareas. Aprenden cómo definir modelos, serializadores, vistas y URLs en Django Rest Framework para construir una API completa. Les ayuda a comprender cómo desarrollar aplicaciones web con APIs utilizando tecnologías modernas.

Lección XIII.

Ejercicios

47. Ejercicio 1:

¿Cómo se define una variable en Python?

Respuesta:

Se define una variable en Python asignándole un nombre y un valor. Por ejemplo:

```
nombre = "Juan"
```

48. Ejercicio 2:

¿Cuál es el resultado de la siguiente expresión?

```
x = 10
y = 5
resultado = x + y
print(resultado)
```

Respuesta:

El resultado de la expresión es 15, ya que se suman los valores de las variables `x` (10) y `y` (5).

49. Ejercicio 3:

¿Qué hace el siguiente fragmento de código?

```
frutas = ["manzana", "banana", "naranja"]
for fruta in frutas:
    print(fruta)
```

Respuesta:

El código recorre la lista `frutas` e imprime cada elemento en una línea separada:

```
manzana
banana
naranja
```

50. Ejercicio 4:

¿Cuál es el valor de la variable resultado después de ejecutar el siguiente código?

```
numero = 7
resultado = numero * 2
resultado = resultado + 3
```

Respuesta:

El valor de la variable `resultado` es 17, ya que se multiplica `numero` por 2 (14) y luego se le suma 3.

51. Ejercicio 5:

¿Qué tipo de dato es el resultado de la siguiente expresión?

```
resultado = 10 / 2
```

Respuesta:

El resultado es de tipo `float` (número de punto flotante), ya que la división produce un valor decimal.

52. Ejercicio 6:

¿Cómo se define una función en Python?

Respuesta:

Una función en Python se define utilizando la palabra clave `def`, seguida del nombre de la función y los parámetros entre paréntesis. Por ejemplo:

```
def saludar(nombre):
    print("Hola,", nombre)
```

53. Ejercicio 7:

¿Cuál es la salida de este código?

```
numero = 5
if numero > 0:
    print("El número es positivo")
else:
    print("El número no es positivo")
```

Respuesta:

La salida es:

```
El número es positivo
```

ya que el valor de `numero` (5) es mayor que 0.

54. Ejercicio 8:

¿Qué hace el siguiente código?

```
for i in range(3):
    print(i)
```

Respuesta:

El código imprime los números del 0 al 2 en líneas separadas:

```
0
1
2
```

55. Ejercicio 9:

¿Cuál es el valor de la variable longitud después de ejecutar este código?

```
frase = "Hola, mundo"
longitud = len(frase)
```

Respuesta:

El valor de la variable `longitud` será 11, ya que la función `len()` retorna la cantidad de caracteres en la cadena.

56. Ejercicio 10:

¿Cuál es la sintaxis correcta para importar la biblioteca math en Python?

Respuesta:

La sintaxis correcta es:

```
import math
```

57. Ejercicio 11:

¿Qué método se utiliza para agregar un elemento al final de una lista?

Respuesta:

El método utilizado para agregar un elemento al final de una lista es `append()`. Por ejemplo:

```
mi_lista = [1, 2, 3]
mi_lista.append(4)
```

58. Ejercicio 12:

¿Cuál es el resultado de la siguiente expresión?

```
resultado = 2 ** 3
```

Respuesta:

El resultado de la expresión es 8, ya que `2 ** 3` representa la potencia de 2 elevado a la 3, que es 8.

59. Ejercicio 13:

¿Qué función se utiliza para convertir un valor a tipo int en Python?

Respuesta:

La función utilizada para convertir un valor a tipo `int` es `int()`. Por ejemplo:

```
numero = int("10")
```

60. Ejercicio 14:

¿Qué método se utiliza para unir elementos de una lista en una cadena?

Respuesta:

El método utilizado para unir elementos de una lista en una cadena es `join()`. Por ejemplo:

```
elementos = ["a", "b", "c"]
cadena = "-".join(elementos)
```

61. Ejercicio 15:

¿Cuál es la salida de este código?

```
for i in range(1, 6):
    if i == 3:
        continue
    print(i)
```

Respuesta:

La salida es:

```
1
2
4
5
```

ya que el valor `3` es omitido debido al uso de `continue`.

62. Ejercicio 16:

¿Qué método se utiliza para eliminar un elemento específico de una lista?

Respuesta:

El método utilizado para eliminar un elemento específico de una lista es `remove()`. Por ejemplo:

```
mi_lista = [1, 2, 3]
mi_lista.remove(2)
```

63. Ejercicio 17:

¿Cómo se define una clase en Python?

Respuesta:

Una clase en Python se define utilizando la palabra clave `class`, seguida del nombre de la clase y los métodos y atributos definidos dentro de la clase. Por ejemplo:

```
class Persona:
    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad
```

64. Ejercicio 18:

¿Cuál es el resultado de la siguiente expresión?

```
x = "Hola"
y = "Mundo"
resultado = x + " " + y
```

Respuesta:

El resultado de la expresión es la cadena “Hola Mundo”, ya que se concatenan las cadenas `x` y `y` junto con un espacio.

65. Ejercicio 19:

¿Cómo se crea una nueva base de datos en PostgreSQL utilizando SQL?

Respuesta:

Para crear una nueva base de datos en PostgreSQL utilizando SQL, se utiliza la siguiente consulta:

```
CREATE DATABASE nombre_basededatos;
```

66. Ejercicio 20:

¿Cuál es la forma correcta de realizar una consulta a una colección en MongoDB?

Respuesta:

La forma correcta de realizar una consulta a una colección en MongoDB es utilizando el método `find()`. Por ejemplo:

```
resultados = db.coleccion.find({"campo": valor})
```

67. UNIDAD I: Introducción a la programación

Ejercicio 1: ¿Cuál es el objetivo principal de la programación?

Respuesta:

El objetivo principal de la programación es resolver problemas y automatizar tareas utilizando un lenguaje de programación.

Ejercicio 2: ¿Qué es un algoritmo?

Respuesta:

Un algoritmo es un conjunto de instrucciones ordenadas y precisas que describen cómo realizar una tarea o resolver un problema.

Ejercicio 3: ¿Cuál es la importancia de la indentación en Python?

Respuesta:

La indentación en Python es importante porque define el bloque de código perteneciente a una estructura, como un bucle o una función. Python utiliza la indentación en lugar de llaves u otros caracteres para delimitar bloques de código.

Ejercicio 4: ¿Qué es un comentario en programación?

Respuesta:

Un comentario en programación es un texto explicativo que se agrega en el código para hacerlo más comprensible. Los comentarios son ignorados por el intérprete y son útiles para documentar el código.

Ejercicio 5: Escribe un programa en Python que imprima "¡Hola, mundo!".

Respuesta:

```
print("¡Hola, mundo!")
```

67.1. UNIDAD II: Instalación de Python y más herramientas

Ejercicio 6: ¿Cuál es la forma de verificar la versión de Python instalada en tu sistema?

Respuesta:

Ejecutando el comando `python --version` en la línea de comandos.

Ejercicio 7: ¿Cuál es el propósito de Git en el desarrollo de software?

Respuesta:

Git es un sistema de control de versiones que permite rastrear cambios en el código, colaborar con otros desarrolladores y mantener un historial completo de modificaciones en un proyecto.

Ejercicio 8: ¿Cómo se instala una extensión (extensión) en Visual Studio Code?

Respuesta:

En Visual Studio Code, puedes instalar extensiones desde la barra lateral izquierda, haciendo clic en el ícono de extensiones (cuatro cuadros) y buscando la extensión que deseas instalar.

Ejercicio 9: ¿Cuál es el resultado del siguiente código?

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

Respuesta:

El resultado es la cadena “Hola, mundo” al concatenar las dos cadenas.

Ejercicio 10: ¿Cuál es el propósito de un entorno virtual en Python?

Respuesta:

Un entorno virtual en Python permite aislar y gestionar las dependencias y paquetes utilizados en un proyecto específico, evitando conflictos con otros proyectos y asegurando un entorno limpio y controlado.

67.2. UNIDAD III: Introducción a Python

Ejercicio 11: ¿Cuál es la diferencia entre una variable y una constante en programación?

Respuesta:

Una variable puede cambiar su valor a lo largo del programa, mientras que una constante mantiene su valor constante durante la ejecución.

Ejercicio 12: Escribe un programa que solicite al usuario su nombre y luego imprima un mensaje de bienvenida con el nombre ingresado.

Respuesta:

```
nombre = input("Ingresa tu nombre: ")
print("¡Bienvenido,", nombre, "!")
```

Ejercicio 13: ¿Cuál es el valor de la variable resultado después de ejecutar el siguiente código?

```
x = 5
y = 2
resultado = x // y
```

Respuesta:

El valor de la variable `resultado` será 2, ya que `//` realiza la división entera de 5 entre 2.

Ejercicio 14: Escribe un programa en Python que determine si un número ingresado por el usuario es par o impar.

Respuesta:

```
numero = int(input("Ingresa un número: "))
if numero % 2 == 0:
    print("El número es par.")
else:
    print("El número es impar.")
```

Ejercicio 15: ¿Cuál es la función del operador not en Python?

Respuesta:

El operador `not` se utiliza para negar una expresión booleana. Si la expresión es verdadera, `not` la convierte en falsa, y viceversa.

67.3. UNIDAD IV: Tipos de Datos

Ejercicio 16: ¿Cuál es la diferencia entre una lista y una tupla en Python?

Respuesta:

La principal diferencia es que las listas son mutables (pueden cambiar) y las tuplas son inmutables (no pueden cambiar). En otras palabras, puedes agregar, eliminar y modificar elementos en una lista, pero no en una tupla.

Ejercicio 17: Escribe un programa que ordene una lista de números en orden ascendente.

Respuesta:

```
numeros = [4, 1, 6, 3, 2]
numeros.sort()
print(numeros)
```

Ejercicio 18: ¿Cómo se accede al tercer elemento de una lista en Python?

Respuesta:

Utilizando el índice 2. Por ejemplo, si la lista se llama `mi_lista`, puedes acceder al tercer elemento con `mi_lista[2]`.

Ejercicio 19: ¿Qué método se utiliza para agregar un elemento al final de una lista?

Respuesta:

El método utilizado es `append()`. Por ejemplo, `mi_lista.append(7)` agrega el número 7 al final de la lista.

Ejercicio 20: Escribe un programa que cuente cuántas veces aparece un elemento específico en una lista.

Respuesta:

```
mi_lista = [2, 4, 6, 4, 8, 4, 10]
elemento = 4
contador = mi_lista.count(elemento)
print("El elemento", elemento, "aparece", contador, "veces.")
```

67.4. UNIDAD V: Control de Flujo

Ejercicio 21: Escribe un programa que determine si un número ingresado por el usuario es positivo, negativo o cero.

Respuesta:

```
numero = int(input("Ingresa un número: "))
if numero > 0:
    print("El número es positivo.")
elif numero < 0:
    print("El número es negativo.")
else:
    print("El número es cero.")
```

Ejercicio 22: ¿Qué hace el siguiente código?

```
contador = 0
while contador < 5:
    print(contador)
    contador += 1
```

Respuesta:

El código imprime los números del 0 al 4 en líneas separadas utilizando un bucle `while`.

Ejercicio 23: ¿Cuál es el resultado de la siguiente expresión?

```
resultado = 0
for i in range(1, 6):
    resultado += i
print(resultado)
```

Respuesta:

El resultado es 15, ya que se suma los números del 1 al 5 en el bucle `for`.

Ejercicio 24: Escribe un programa que calcule la suma de todos los números pares entre 1 y 100.

Respuesta:

```
suma = 0
for i in range(2, 101, 2):
    suma += i
print("La suma de los números pares entre 1 y 100 es:", suma)
```

Ejercicio 25: ¿Cuál es el propósito de la instrucción break en un bucle?

Respuesta:

La instrucción `break` se utiliza para salir inmediatamente de un bucle, interrumpiendo su ejecución antes de que termine naturalmente.

67.5. UNIDAD VI: Funciones

Ejercicio 26: ¿Qué es una función en programación?

Respuesta:

Una función es un bloque de código reutilizable que realiza una tarea específica. Puede recibir argumentos, ejecutar instrucciones y devolver un valor.

Ejercicio 27: Escribe una función en Python que calcule el área de un círculo.

Respuesta:

```
import math

def area_circulo(radio):
return math.pi * radio ** 2
```

Ejercicio 28: ¿Qué es la recursividad en programación?

Respuesta:

La recursividad es una técnica donde una función se llama a sí misma para resolver un problema. Es útil para resolver problemas que se pueden descomponer en subproblemas similares.

Ejercicio 29: Escribe una función recursiva en Python para calcular el factorial de un número.

Respuesta:

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)
```

Ejercicio 30: ¿Por qué es importante utilizar funciones en la programación?

Respuesta:

Las funciones permiten dividir el código en bloques más pequeños y manejables, lo que facilita la reutilización, la depuración y la comprensión del código. Además, promueven la modularidad y el diseño limpio.

67.6. UNIDAD VII: Objetos, clases y herencia

Ejercicio 31: ¿Qué es una clase en programación orientada a objetos?

Respuesta:

Una clase es un plano o plantilla para crear objetos en programación orientada a objetos. Define las propiedades (atributos) y comportamientos (métodos) que tendrán los objetos creados a partir de ella.

Ejercicio 32: Escribe una clase en Python llamada Persona con los atributos nombre y edad, y un método saludar() que imprima un saludo con el nombre de la persona.

Respuesta:

```
class Persona:
    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad
```

```
def saludar(self):
    print("¡Hola, soy", self.nombre, "y tengo", self.edad, "años!")
```

Ejercicio 33: ¿Qué es la herencia en programación orientada a objetos?

Respuesta:

La herencia es un concepto en el que una clase (subclase) puede heredar atributos y métodos de otra clase (superclase). Permite reutilizar y extender el código de una clase existente para crear una nueva clase.

Ejercicio 34: Escribe una clase en Python llamada `Estudiante` que herede de la clase `Persona` y tenga un atributo adicional `curso`.

Respuesta:

```
class Estudiante(Persona):
    def __init__(self, nombre, edad, curso):
        super().__init__(nombre, edad)
        self.curso = curso
```

Ejercicio 35: ¿Por qué es beneficioso utilizar la herencia en programación?

Respuesta:

La herencia permite reutilizar código, promover la coherencia y facilitar la actualización y mantenimiento. También permite crear jerarquías de clases para modelar relaciones entre objetos del mundo real.

67.7. UNIDAD VIII: Módulos

Ejercicio 36: ¿Qué es un módulo en Python?

Respuesta:

Un módulo en Python es un archivo que contiene definiciones y declaraciones de variables, funciones y clases. Permite organizar y reutilizar el código en diferentes programas.

Ejercicio 37: Escribe un módulo en Python llamado operaciones que contenga una función suma para sumar dos números.

Respuesta:

Archivo `operaciones.py`:

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

Ejercicio 38: ¿Cómo se importa un módulo en Python?

Respuesta:

Se importa utilizando la palabra clave `import`, seguida del nombre del módulo. Por ejemplo, `import operaciones` importaría el módulo `operaciones`.

Ejercicio 39: Escribe un programa que utilice la función suma del módulo operaciones para sumar dos números ingresados por el usuario.

Respuesta:

```
import operaciones

num1 = float(input("Ingresa el primer número: "))
num2 = float(input("Ingresa el segundo número: "))
resultado = operaciones.suma(num1, num2)
print("La suma es:", resultado)
```

Ejercicio 40: ¿Cuál es la ventaja de utilizar módulos en Python?

Respuesta:

Los módulos permiten la modularidad, la reutilización de código y la organización efectiva del código en componentes separados. También facilitan la colaboración y la mantenibilidad.

67.8. UNIDAD IX: Introducción a Bases de Datos

Ejercicio 41: ¿Qué es una base de datos en el contexto de la programación?

Respuesta:

Una base de datos es un sistema organizado para almacenar, administrar y recuperar información de manera eficiente. Se utiliza para almacenar datos estructurados de manera persistente.

Ejercicio 42: ¿Qué es PostgreSQL?

Respuesta:

PostgreSQL es un sistema de gestión de bases de datos relacional de código abierto y potente. Es conocido por su capacidad de manejar cargas de trabajo complejas y por sus características avanzadas.

Ejercicio 43: ¿Qué es MongoDB?

Respuesta:

MongoDB es una base de datos NoSQL orientada a documentos. Almacena los datos en documentos JSON flexibles en lugar de en tablas tradicionales, lo que permite una gran flexibilidad y escalabilidad.

Ejercicio 44: ¿Cuál es la ventaja de utilizar bases de datos en programas?

Respuesta:

Las bases de datos permiten almacenar y administrar grandes cantidades de datos de manera estructurada y eficiente. Esto facilita el acceso y la manipulación de datos en aplicaciones.

Ejercicio 45: ¿Cuál es el propósito de una clave primaria en una base de datos?

Respuesta:

Una clave primaria es un campo único en una tabla que se utiliza para identificar de manera única cada registro en la tabla. Se utiliza como referencia para relacionar tablas y mantener la integridad de los datos.

UNIDAD X: MySQL, PostgreSQL y MongoDB: Operaciones básicas en bases de datos

Ejercicio 46: ¿Cómo se realiza una consulta básica a una tabla en SQL?

Respuesta:

Utilizando la sentencia `SELECT`. Por ejemplo, `SELECT * FROM tabla` recuperará todos los registros de la tabla.

Ejercicio 47: ¿Qué comando se utiliza para insertar un nuevo registro en una tabla en SQL?

Respuesta:

El comando utilizado es `INSERT INTO`. Por ejemplo, `INSERT INTO tabla (columna1, columna2) VALUES (valor1, valor2)` insertará un nuevo registro en la tabla.

Ejercicio 48: ¿Cómo se actualiza un registro en una tabla en SQL?

Respuesta:

Utilizando el comando `UPDATE`. Por ejemplo, `UPDATE tabla SET columna = valor WHERE condicion` actualizará los registros que cumplan con la condición.

Ejercicio 49: ¿Cuál es el propósito de la sentencia DELETE en SQL?

Respuesta:

La sentencia `DELETE` se utiliza para eliminar uno o varios registros de una tabla. Por ejemplo, `DELETE FROM tabla WHERE condicion` eliminará los registros que cumplan con la condición.

Ejercicio 50: ¿Cuál es la ventaja de utilizar bases de datos NoSQL como MongoDB?

Respuesta:

Las bases de datos NoSQL, como MongoDB, son flexibles y escalables, lo que las hace ideales para manejar grandes cantidades de datos no estructurados o semiestructurados. Son adecuadas para aplicaciones web y móviles modernas.

67.9. UNIDAD XI: ¿Cómo me amplío con Python?

Ejercicio 51: ¿Qué es la ciencia de datos y cómo se relaciona con Python?

Respuesta:

La ciencia de datos es el proceso de extracción, transformación y análisis de datos para obtener conocimientos y tomar decisiones informadas. Python es ampliamente utilizado en la ciencia de datos debido a su amplio ecosistema de bibliotecas y herramientas.

Ejercicio 52: ¿Qué es Django Framework y para qué se utiliza?

Respuesta:

Django es un framework web de alto nivel en Python que facilita la creación de aplicaciones web robustas y escalables. Se utiliza para construir sitios web y aplicaciones con características como autenticación, seguridad y manejo de bases de datos.

Ejercicio 53: ¿Qué es FastAPI y cómo se diferencia de otros frameworks?

Respuesta:

FastAPI es un framework web moderno y de alto rendimiento para construir APIs en Python. Se destaca por su velocidad, facilidad de uso y generación automática de documentación interactiva. Utiliza anotaciones de tipo para validar datos y reducir errores.

Ejercicio 54: ¿Cuál es el propósito de las APIs en el desarrollo web?

Respuesta:

Las APIs (Interfaces de Programación de Aplicaciones) se utilizan para permitir la comunicación y la integración entre diferentes aplicaciones y sistemas. Facilitan el intercambio de datos y funcionalidades entre aplicaciones.

Ejercicio 55: ¿Por qué es importante ampliarse en Python más allá de los conceptos básicos?

Respuesta:

Ampliarse en Python permite abordar proyectos más complejos y desafiantes, como desarrollo web, análisis de datos, automatización, inteligencia artificial y más. Además, mejora las habilidades y la versatilidad como programador.