Python 2023

Diego Saavedra

Sep 20, 2023

Table of contents

1 Bienvenida

¡Bienvenidos al Curso Completo de Python, analizaremos desde los 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!

Part I

Unidad 1: Introducción a Python

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.

> Tip
>
> **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.

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

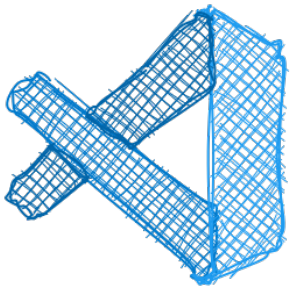
Python es un lenguaje interpretado, lo que significa que se ejecuta línea por línea en tiempo real.

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:

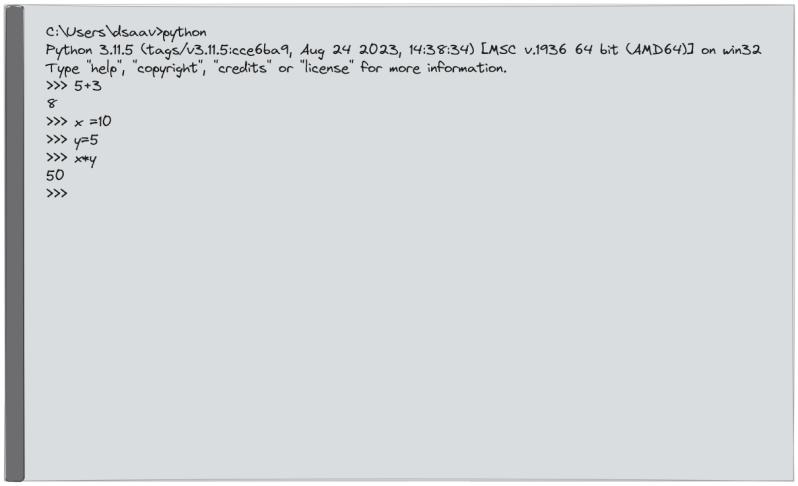
```
print("Python se ha instalado correctamente.")
```

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.

4 Uso del REPL, PEP 8 y el Zen de Python



4.0.1 El REPL (Read-Eval-Print Loop).

Definición y Propósito del REPL.

- El REPL (Read-Eval-Print Loop) es una herramienta interactiva que permite ejecutar código Python de forma inmediata y ver los resultados de las operaciones en tiempo real. Es una excelente manera de probar pequeños fragmentos de código, experimentar y depurar sin necesidad de escribir un programa completo.

Uso Básico del REPL

Para iniciar el REPL, simplemente abre una terminal o línea de comandos y escribe python o python3 (dependiendo de tu instalación) seguido de Enter. Esto te llevará al entorno interactivo de Python.

4.0.2 Ejemplos de Interacción con el REPL

```
# Ejemplo 1: Realizar cálculos simples
>>> 5 + 3
8

# Ejemplo 2: Definir variables y realizar operaciones
>>> x = 10
>>> y = 5
```

```
>>> x * y
50

# Ejemplo 3: Trabajar con cadenas de texto
>>> mensaje = "Hola, mundo!"
>>> mensaje.upper()
'HOLA, MUNDO!'

# Ejemplo 4: Importar módulos y usar funciones
>>> import math
>>> math.sqrt(16)
4.0
```

4.0.3 PEP 8: Guía de Estilo de Python.

¿Qué es PEP 8 y Por Qué es Importante?

PEP 8 (Python Enhancement Proposal 8) es una guía de estilo que establece convenciones para escribir código Python legible y consistente. La adopción de PEP 8 es importante porque facilita la colaboración en proyectos, mejora la legibilidad del código y ayuda a mantener una base de código ordenada y coherente.

Convenciones de Nombres

PEP 8 establece reglas para nombrar variables, funciones, clases y módulos en Python. Algunas convenciones clave incluyen:

- Las variables y funciones deben usar minúsculas y palabras separadas por guiones bajos (snake_case).
- Las clases deben usar CamelCase (con la primera letra en mayúscula). Los módulos deben tener nombres cortos y en minúsculas.

Reglas de Formato y Estilo

PEP 8 también define reglas de formato, como el uso de espacios en lugar de tabulaciones, la longitud máxima de línea y la organización de importaciones.

Herramientas para Verificar el Cumplimiento de PEP 8

Existen herramientas como flake8 y complementos para editores de código que pueden analizar el código en busca de posibles violaciones de PEP 8 y proporcionar sugerencias de corrección. 2.4.3. El Zen de Python

4.0.4 Introducción al Zen de Python (PEP 20).

El Zen de Python es un conjunto de principios y filosofía de diseño que guían el desarrollo de Python. Estos principios se pueden acceder desde el intérprete de Python utilizando el siguiente comando:

```
import this
```

Los principios del Zen de Python proporcionan orientación sobre cómo escribir código Python de manera clara y elegante.

Principios y Filosofía de Diseño de Python

Algunos de los principios más destacados del Zen de Python incluyen:

- **La legibilidad cuenta:** El código debe ser legible para los humanos, ya que se lee con más frecuencia de lo que se escribe.
- **Explícito es mejor que implícito:** El código debe ser claro y no dejar lugar a ambigüedades.
- **La simplicidad vence a la complejidad:** Debe preferirse la simplicidad en el diseño y la implementación.
- **Los errores nunca deben pasar en silencio:** Los errores deben manejarse adecuadamente y, si es posible, informar de manera explícita.

4.0.5 Ejercicios Prácticos

- **Ejercicio 1:** Uso del REPL para Realizar Cálculos Simples

① Abre el REPL de Python.

- **Ejercicio 2:** Verificación de Cumplimiento de PEP 8 en Código Python
- Escribe un pequeño programa en Python que incluya variables, funciones y comentarios.

- Utiliza la herramienta flake8 o un complemento de tu editor de código para verificar si tu código cumple con las reglas de PEP 8.
- Corrige cualquier violación de PEP 8 y vuelve a verificar el código.
- **Ejercicio 3:** Exploración y Reflexión sobre los Principios del Zen de Python
- Ejecuta el comando import this en el REPL para acceder a los principios del Zen de Python.
- Lee y reflexiona sobre cada uno de los principios.
- Escribe un breve párrafo sobre cómo un principio específico del Zen de Python puede aplicarse al desarrollo de software.

Tip

Actividad Práctica

- Desarrolla un pequeño programa Python que siga las pautas de PEP 8 y refleje los principios del Zen de Python en su diseño y estilo de codificación. Asegúrate de que el código sea legible y cumpla con las convenciones de nombres y formato de PEP 8.
- Esta subunidad proporciona a los estudiantes una comprensión más profunda de las herramientas y las convenciones de estilo que se utilizan en la programación en Python. Además, les ayuda a reflexionar sobre la filosofía de diseño de Python y cómo aplicarla en la práctica.

4.0.6 Explicación

Esta actividad te invita a desarrollar un pequeño programa en Python que siga las pautas de PEP 8 y refleje los principios del Zen de Python en su diseño y estilo de codificación. La importancia de esta tarea radica en aprender a escribir código que sea limpio, legible y siga las convenciones de la comunidad de Python.

- **Cumplir con PEP 8**: PEP 8 es el estándar de estilo de código para Python, y seguirlo es una práctica recomendada en la comunidad de programadores. Tu programa debe seguir las convenciones de formato, nombres de variables, estructura de código, entre otros aspectos que se describen en PEP 8.
- **Reflejar el Zen de Python**: El Zen de Python es una colección de principios filosóficos que guían el diseño del lenguaje Python. Algunos de estos principios incluyen la legibilidad del código, la simplicidad y la importancia de los casos especiales. Tu programa debe reflejar estos principios en su diseño y estilo de codificación.
- **Legibilidad y Comentarios**: Asegúrate de que tu código sea legible para otras personas. Usa nombres de variables descriptivos, agrega comentarios explicativos cuando sea necesario y sigue las mejores prácticas para hacer que tu código sea fácil de entender.

- **Aplicación Práctica**: Esta actividad te brinda la oportunidad de aplicar los conceptos de estilo de código y filosofía de diseño de Python en un proyecto real. Esto es importante ya que en la programación colaborativa, otros desarrolladores deben poder entender y trabajar con tu código de manera eficiente.

Al completar esta actividad, habrás mejorado tus habilidades en la escritura de código Python de alta calidad y te habrás familiarizado con las convenciones y filosofía de diseño de Python. Recuerda que escribir código limpio es una habilidad esencial para cualquier programador.

5 Entornos de Desarrollo

Part II

Unidad 2: Introducción a la Programación con Python

6 Identación y Comentarios

6.1 Identación

6.2 Conceptos Clave

6.2.1 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.

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")
```

Actividad Práctica

1. Escribe un programa que solicite al usuario su edad y muestre un mensaje según si es mayor de 18 años o no.

Posible solución

Resumen:

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.

```
# Programa que solicita la edad y muestra un mensaje
edad = int(input("Ingrese su edad: "))

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

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

¿Qué hicimos?

① Se solicita la edad al usuario y se almacena en la variable edad.

6.3 Comentarios

6.4 Conceptos Clave

6.4.1 Comentarios

- Son notas en el código que no se ejecutan y se utilizan para explicar el propósito y funcionamiento de partes del programa.
- Comentarios de una línea: Se crean con el símbolo "#" y abarcan una sola línea.
- Comentarios de múltiples líneas: Se crean entre triple comillas (" " " o ' ' ') y pueden abarcar múltiples líneas.

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
```

> 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.

Posible solución

```
# Este programa calcula el área de un triángulo
# solicitando la base y la altura al usuario.

# Solicitar la base y almacenarla en la variable 'base'
base = float(input("Ingrese la base del triángulo: "))

# Solicitar la altura y almacenarla en la variable 'altura'
```

```
altura = float(input("Ingrese la altura del triángulo: "))

# Calcular el área del triángulo
area = 0.5 * base * altura

# Mostrar el resultado
print(f"El área del triángulo es: {area}")
```

6.5 ¿Qué aprendimos?

En este tema, aprendimos la importancia de la identación en Python para estructurar nuestro código correctamente. La identación nos permite definir bloques de código, como en los condicionales, de manera clara y legible.

También aprendimos cómo agregar comentarios en Python para documentar nuestro código. Los comentarios son esenciales para explicar el propósito y el funcionamiento de las partes del programa y facilitan la colaboración entre programadores.

7 Variables y Variables Múltiples

7.1 Variables

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

7.1.1 Conceptos Clave

Variables

- Nombres que representan ubicaciones de memoria donde se almacenan datos.

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).

Ejemplo:

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

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

1. Crea variables para almacenar información personal, como tu ciudad, tu edad y tu ocupación.
2. Declara variables para almacenar cantidades numéricas, como el precio de un producto y la cantidad de unidades disponibles.

7.2 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. 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.

7.2.1 Conceptos Clave

Asignación Múltiple

- Permite asignar valores a varias variables en una línea.

Desempaquetado de Valores

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

Intercambio de Valores

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

Ejemplo:

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

7.2.2 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.

Actividad Práctica

1. Crea una lista con los nombres de tus tres colores favoritos.
2. Utiliza la asignación múltiple para asignar los valores de la lista a tres variables individuales.

7.2.3 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.

8 Concatenación

8.1 Conceptos Clave

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

Operador +: Se utiliza para concatenar cadenas de texto.

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

8.2 Ejemplo

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

8.2.1 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.

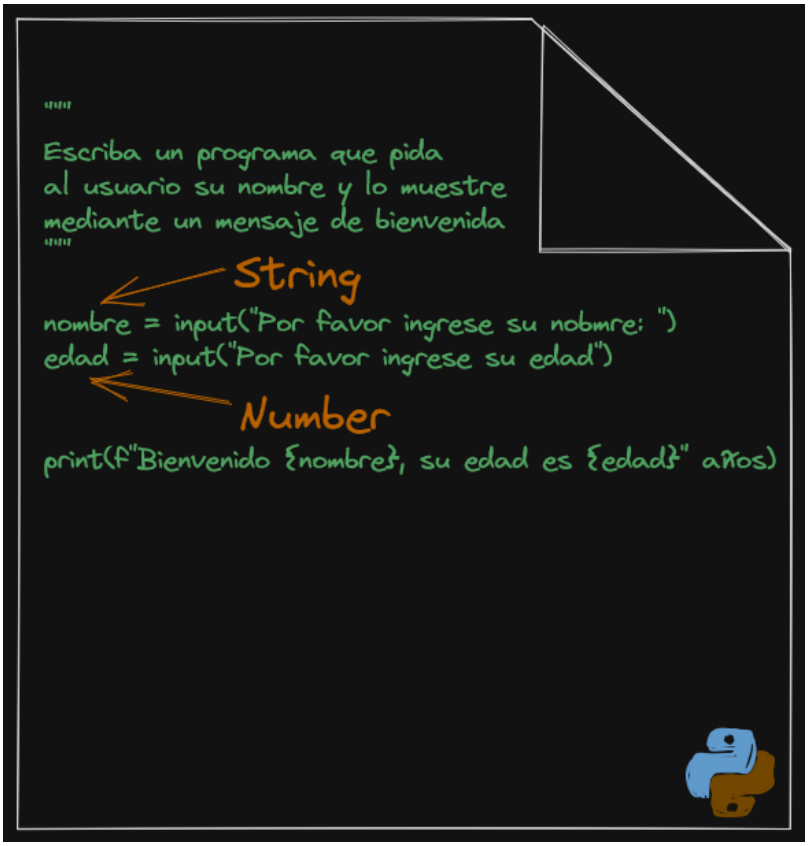
8.2.2 Explicación de la Actividad:

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.

Part III

Unidad 3: Tipos de Datos

9 String y Números



9.1 Conceptos Clave

String

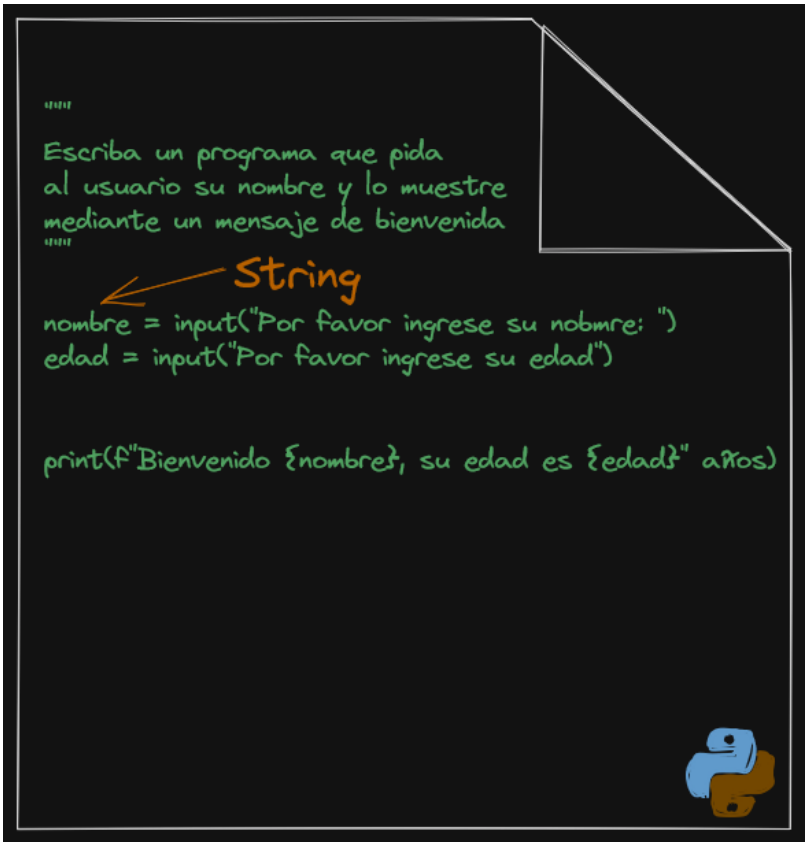
Un string es una secuencia de caracteres alfanuméricos. Se pueden definir utilizando comillas simples o dobles.

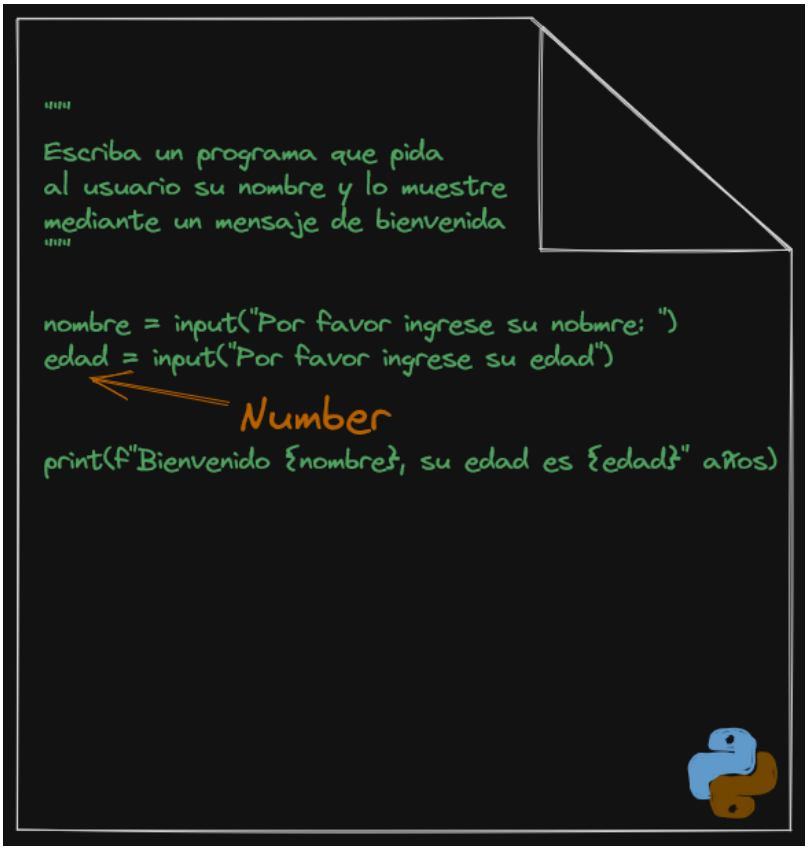
Números Enteros (int)

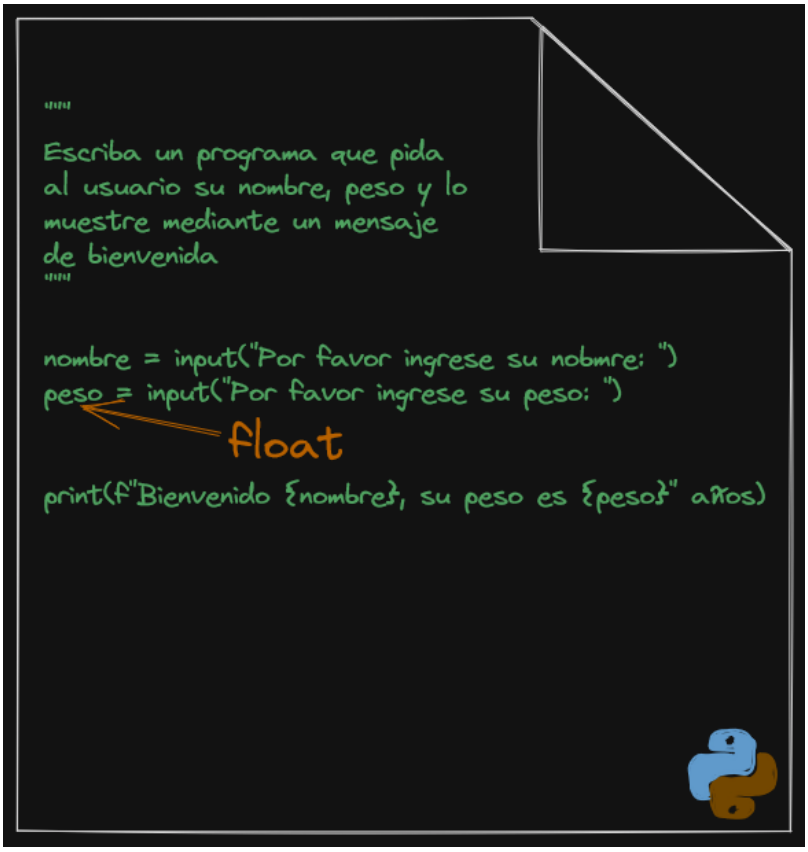
Los números enteros representan valores numéricos enteros, ya sean positivos o negativos.

Números de Punto Flotante (float)

Los números de punto flotante representan valores numéricos con decimales.


"""
Escriba un programa que pida
al usuario su nombre y lo muestre
mediante un mensaje de bienvenida
"""
String
nombre = input("Por favor ingrese su nobmre: ")
edad = input("Por favor ingrese su edad")
print(f"Bienvenido {nombre}, su edad es {edad}" años)


"""
Escriba un programa que pida
al usuario su nombre y lo muestre
mediante un mensaje de bienvenida
"""
nombre = input("Por favor ingrese su nobmre: ")
edad = input("Por favor ingrese su edad")
Number
print(f"Bienvenido {nombre}, su edad es {edad}" años)



9.2 Ejemplo

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

# Números
edad = 25
saldo = 1500.75
```

9.2.1 Explicación:

En este ejemplo, se crean variables que almacenan strings y números. Los strings se definen utilizando comillas simples o dobles, y los números enteros y de punto flotante se asignan directamente a variables.

Actividad Práctica

1. Crea una variable con el título de tu canción favorita.
2. Asigna tu edad a una variable y tu altura a otra variable.
3. Combina las variables para crear un mensaje personalizado.

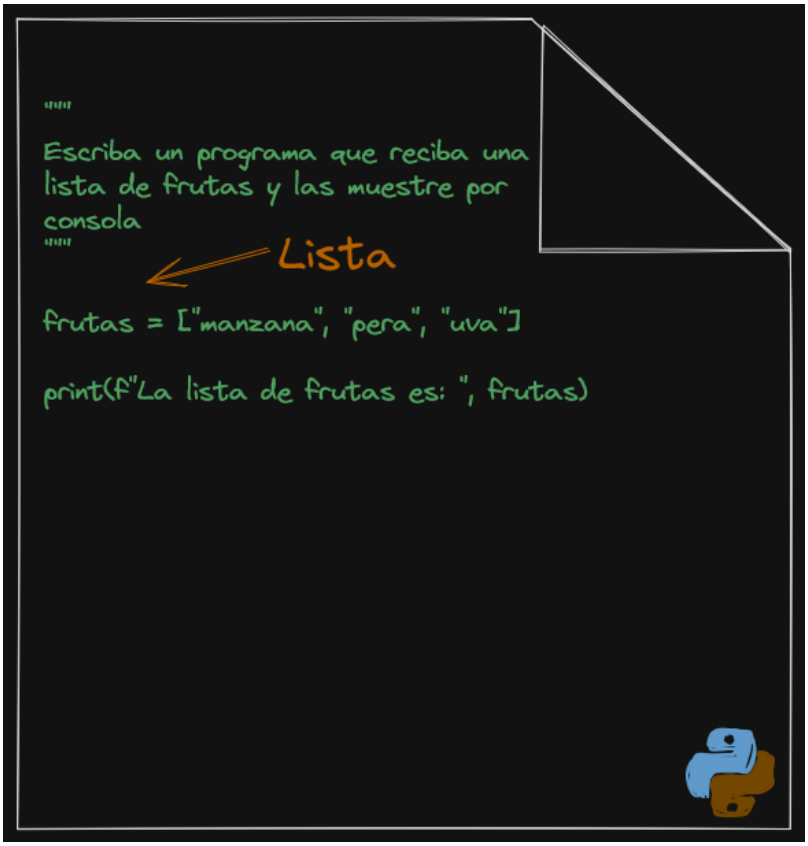
9.2.2 Explicación:

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.

9.3 ¿Qué Aprendimos?

En esta lección, aprendimos sobre los dos tipos de datos fundamentales en Python: strings y números. Aprendimos cómo crear y trabajar con strings utilizando comillas simples o dobles. También exploramos dos tipos numéricos clave: números enteros (int) y números de punto flotante (float). Estos conceptos son esenciales para manejar información en Python y forman la base de muchos programas y aplicaciones.

10 Listas y Tuplas



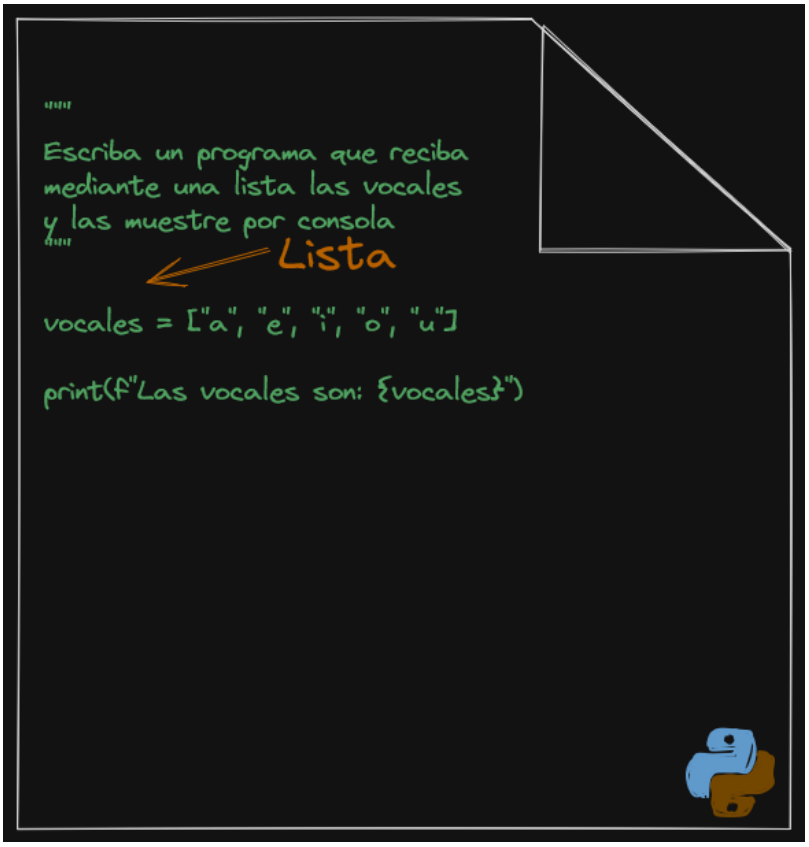
10.1 Conceptos Clave

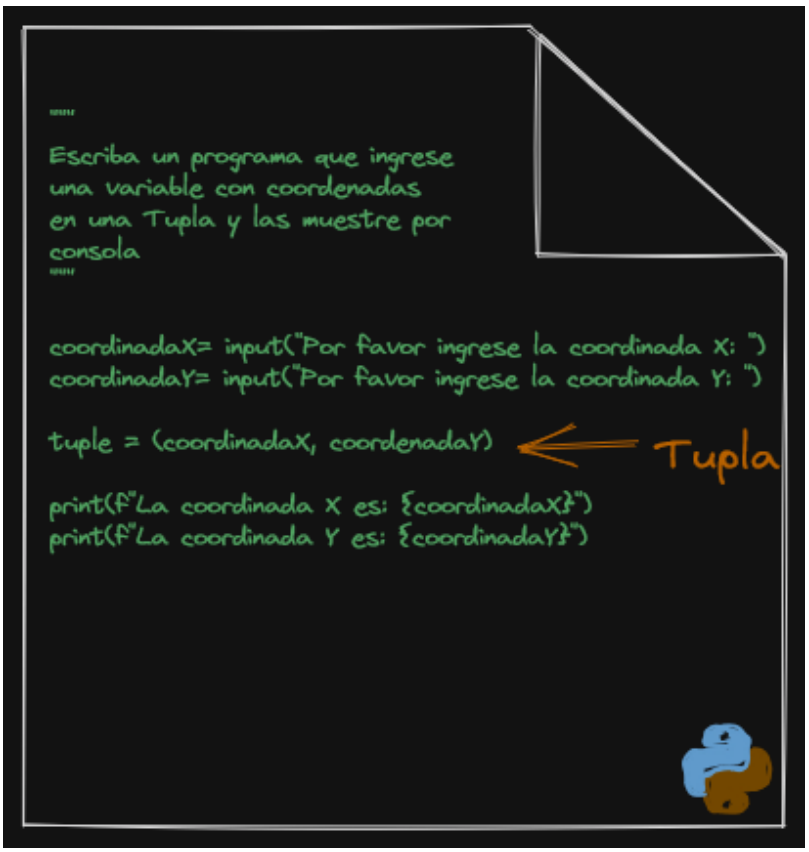
10.1.1 Listas

Las listas son secuencias ordenadas de elementos que pueden ser de diferentes tipos. Permiten almacenar varios elementos en una sola variable.

10.1.2 Tuplas

Las tuplas son similares a las listas, pero son inmutables, lo que significa que no se pueden modificar después de ser creadas.


"""
Escriba un programa que reciba
mediante una lista las vocales
y las muestre por consola
"""
Lista
vocales = ["a", "e", "i", "o", "u"]
print(f"Las vocales son: {vocales}")


"""
Escriba un programa que ingrese
una variable con coordenadas
en una Tupla y las muestre por
consola
"""
coordinadaX= input("Por favor ingrese la coordinada X: ")
coordinadaY= input("Por favor ingrese la coordinada Y: ")
tuple = (coordinadaX, coordenadaY)
Tupla
print(f"La coordinada X es: {coordinadaX}")
print(f"La coordinada Y es: {coordinadaY}")

10.2 Ejemplo

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

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

10.2.1 Explicación:

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

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

Actividad Práctica (Listas):

1. Crea una lista con los nombres de tus tres películas favoritas.
2. Accede al segundo elemento de la lista e imprímelo en la consola.

10.2.2 Explicación:

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.

Actividad Práctica (Tuplas):

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

10.2.3 Explicación:

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.

10.3 ¿Qué Aprendimos?

En esta lección, aprendimos sobre dos tipos de estructuras de datos en Python: listas y tuplas.

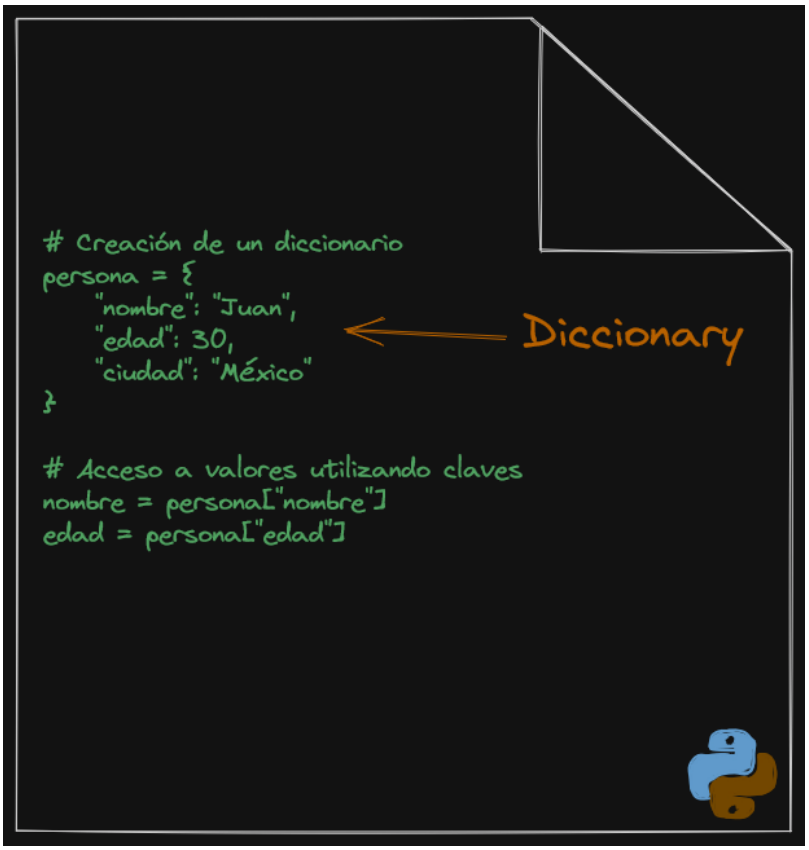
Listas: Son secuencias ordenadas de elementos que pueden modificarse. Se accede a los elementos utilizando índices.

Tuplas: Son similares a las listas, pero son inmutables, lo que significa que no pueden modificarse después de su creación. También se accede a los elementos utilizando índices.

Estas estructuras nos permiten almacenar y organizar datos de manera eficiente en Python, y la elección entre listas y tuplas depende de si necesitamos datos modificables o inmutables en nuestros programas.

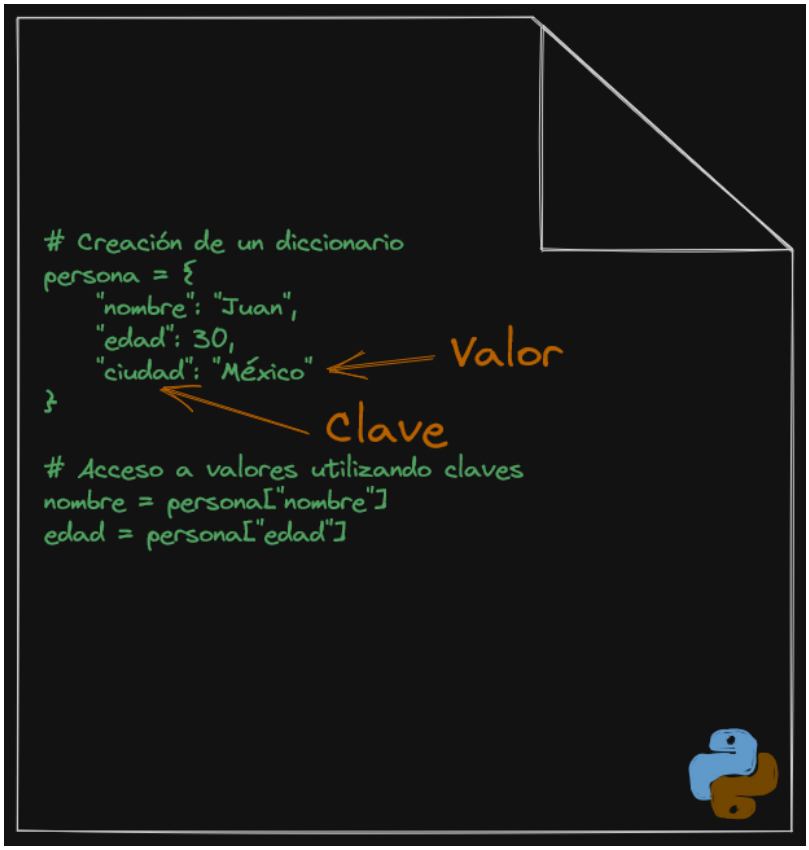
11 Diccionarios y Booleanos

11.1 Diccionarios



11.1.1 Conceptos Clave

11.1.1.1 Diccionarios



Los diccionarios son estructuras de datos que almacenan pares clave-valor.

11.1.1.2 Claves

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

11.1.1.3 Valores

Son los datos asociados a cada clave en el diccionario.

11.1.2 Ejemplo

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

```
}

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

11.1.3 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
>
> 1. Crea un diccionario que almacene información de tus libros favoritos, incluyendo título y autor.
> 2. Accede a los valores del diccionario utilizando las claves y muestra la información en la consola.

11.1.4 Explicación:

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.

11.2 Booleanos

11.3 Conceptos Clave

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. Ejemplo

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

# Expresiones lógicas
puede_ingresar = es_mayor_de_edad and tiene_tarjeta
```

11.3.1 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.

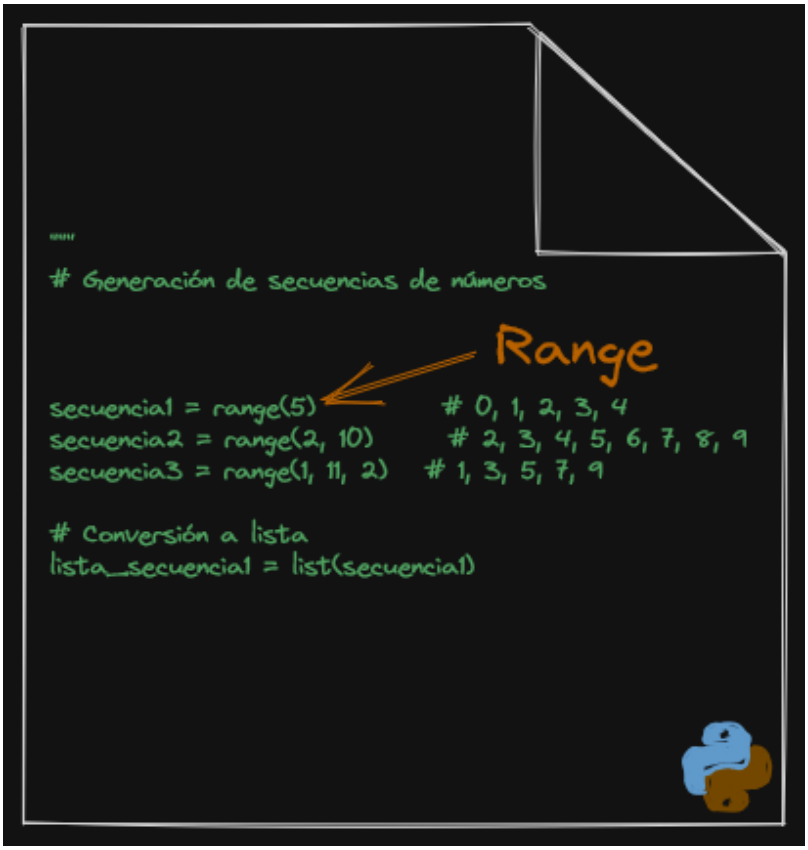
Actividad Práctica

1. Crea variables booleanas que representen si tienes una mascota y si te gusta el deporte.
2. 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.

11.3.2 Explicación:

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.

12 Range



12.1 Conceptos Clave

12.1.1 range

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

12.1.2 Parámetros de range

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

12.1.3 Conversión a Listas

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

12.2 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)
```

12.2.1 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

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

12.2.2 Explicación:

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.

Part IV

Unidad 4: Control de Flujo

13 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.

13.1 Conceptos Clave

13.1.1 Control de Flujo

Manejo de la ejecución del código basado en condiciones.

13.1.2 Estructura if

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

13.1.3 Bloque de Código

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

13.2 Ejemplo

```
edad = 18

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

13.2.1 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

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

13.2.2 Explicación:

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.

13.3 ¿Qué aprendimos?

En este tema, aprendimos los conceptos fundamentales del control de flujo en programación y cómo utilizar la estructura if para ejecutar código condicionalmente. Ahora somos capaces de tomar decisiones en nuestros programas basadas en condiciones específicas.

14 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.

14.1 Conceptos Clave

14.1.1 Estructura elif

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

14.1.2 Estructura else

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

14.1.3 Anidación de Estructuras if

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

14.2 Ejemplo

```
puntaje = 85

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

14.2.1 Explicación:

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

Actividad Práctica

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

14.2.2 Explicación:

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.

14.3 ¿Qué aprendimos?

En este tema, aprendimos a utilizar la estructura if, elif y else para manejar múltiples condiciones y ejecutar código basado en resultados específicos. También comprendimos cómo anidar estructuras if para manejar situaciones más complejas en la programación.

15 If, elif y else

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

15.1 Conceptos Clave

15.1.1 Estructura elif

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

15.1.2 Estructura else

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

15.1.3 Anidación de Estructuras if

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

15.2 Ejemplo

```
puntaje = 85

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

15.2.1 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.

15.2.2 Explicación:

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.

15.3 ¿Qué aprendimos?

En esta lección, aprendimos cómo utilizar la estructura if, elif y else para tomar decisiones basadas en múltiples condiciones. Esto nos permite ejecutar diferentes bloques de código según diferentes situaciones. También exploramos la anidación de estructuras if, lo que nos permite manejar situaciones aún más complejas en la programación.

16 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.

16.1 Conceptos Clave:

16.1.1 Operador and

Retorna True si ambas condiciones son verdaderas.

16.1.2 Operador or

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

16.1.3 Combinación de Condiciones

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

16.2 Ejemplo:

```
edad = 20
tiene_permiso = True

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

16.2.1 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.

16.2.2 Explicación:

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.

16.3 ¿Qué aprendimos?

En esta lección, aprendimos cómo utilizar los operadores lógicos and y or para combinar condiciones y crear expresiones más complejas en nuestras estructuras de control de flujo. Estos operadores son útiles cuando necesitamos tomar decisiones basadas en múltiples condiciones en nuestros programas.

17 Introducción a While.

17.1 Introducción a While

En esta lección, comenzaremos a explorar la estructura de control de flujo while, que nos permite crear bucles que se ejecutan repetidamente mientras se cumple una condición.

17.2 Conceptos Clave

17.2.1 Bucle While

Un bucle que se ejecuta mientras una condición sea verdadera.

17.2.2 Condición

La expresión que se evalúa para determinar si el bucle debe continuar ejecutándose.

17.2.3 Bloque de Código

El conjunto de instrucciones que se ejecutan dentro del bucle while.

17.3 Ejemplo

```
contador = 0

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

17.3.1 Explicación:

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

17.3.2 Explicación:

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.

17.4 ¿Qué aprendimos?

En esta lección, aprendimos los conceptos clave de la estructura de control de flujo while. Descubrimos cómo crear bucles que se ejecutan mientras se cumple una condición y cómo utilizarlos en situaciones prácticas en la programación.

18 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.

18.1 Conceptos Clave:

18.1.1 Sentencia break:

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

18.1.2 Bucles Infinitos:

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

18.2 Ejemplo:

```
contador = 0

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

18.2.1 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:

1. Crea un bucle while que pida al usuario ingresar un número positivo menor que 10.
2. Utiliza la sentencia break para salir del bucle una vez que el usuario ingrese un número válido.

18.2.2 Explicación:

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.

18.3 ¿Qué aprendimos?

En esta lección, aprendimos cómo utilizar la estructura while para crear bucles en Python que se ejecutan mientras se cumpla una condición. También aprendimos a utilizar la sentencia break para salir de un bucle antes de que la condición sea falsa. Los bucles while son útiles cuando necesitamos ejecutar un bloque de código repetidamente hasta que se cumpla una condición específica.

19 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.

19.1 Conceptos Clave:

19.1.1 Sentencia continue

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

19.1.2 Saltar Iteraciones

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

19.2 Ejemplo

```
contador = 0

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

19.2.1 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.

19.2.2 Explicación:

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.

19.3 ¿Qué aprendimos?

En esta lección, aprendimos cómo utilizar la sentencia continue en un bucle while para saltar a la siguiente iteración sin ejecutar el resto del código en esa iteración. Esto es útil cuando queremos omitir ciertas iteraciones basadas en una condición específica. El control preciso de las iteraciones en un bucle puede ser esencial para realizar tareas específicas en un programa.

20 For Loop

En esta lección, aprenderemos sobre la estructura de control de flujo for loop en Python. El bucle for nos permite recorrer elementos de una secuencia, como una lista o una cadena de texto.

20.1 Conceptos Clave

20.1.1 Bucle For

Un bucle que itera a través de una secuencia de elementos y ejecuta un bloque de código para cada elemento.

20.1.2 Iteración

Cada ejecución del bloque de código en un bucle for se llama iteración.

20.1.3 Elemento de la Secuencia

Los elementos individuales en la secuencia que se está recorriendo.

20.2 Ejemplo

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

for fruta in frutas:
    print(fruta)
```

20.2.1 Explicación:

En este ejemplo, utilizamos un bucle for para iterar a través de la lista frutas e imprimimos cada fruta en la consola.

> Actividad Práctica
>
> Crea una lista de números del 1 al 5 y utiliza un bucle for para imprimir el cuadrado de cada número.

20.2.2 Explicación:

Esta actividad te permite practicar cómo usar un bucle for para recorrer una secuencia de números y realizar operaciones en cada elemento. Los bucles for son muy útiles para procesar datos en una variedad de situaciones de programación.

20.3 ¿Qué aprendimos?

En esta lección, aprendimos cómo usar un bucle for para iterar a través de una secuencia de elementos en Python. Comprendimos los conceptos clave relacionados con los bucles for y cómo aplicarlos en la escritura de código.

Part V

Unidad 5: Funciones y Recursividad

21 Introducción a Funciones

En esta lección, exploraremos el concepto de funciones en Python. Las funciones son bloques de código reutilizables que realizan tareas específicas. Aprenderemos cómo definir y utilizar funciones en nuestros programas.

21.1 Conceptos Clave

21.1.1 Función

Un bloque de código reutilizable que realiza una tarea específica cuando se llama.

21.1.2 Definición de Función

Crear una función especificando su nombre, parámetros y cuerpo.

21.1.3 Llamada de Función

Ejecutar una función para que realice su tarea específica.

21.2 Ejemplo

```
# Definición de una función
def saludar(nombre):
    print("¡Hola, " + nombre + "!")

# Llamada de función
saludar("Juan")
```

21.2.1 Explicación:

En este ejemplo, definimos una función llamada saludar que toma un parámetro nombre e imprime un saludo personalizado. Luego, llamamos a esta función con el nombre “Juan”.

Tip

Actividad Práctica
Crea una función llamada calcular_area_rectangulo que tome dos parámetros: largo y ancho. La función debe calcular y devolver el área de un rectángulo. Luego, llama a la función con valores diferentes para largo y ancho y muestra el resultado.

21.2.2 Explicación:

Esta actividad te permite practicar cómo definir funciones con parámetros y cómo utilizarlas para realizar cálculos específicos. Las funciones son una parte fundamental de la programación, ya que permiten organizar y reutilizar el código de manera efectiva.

21.3 ¿Qué aprendimos?

En esta lección, aprendimos qué son las funciones en Python y cómo definirlas y usarlas en nuestros programas. Comprendimos los conceptos clave relacionados con las funciones y cómo aplicarlos en la escritura de código.

22 Recursividad

En esta lección, exploraremos el concepto de recursividad en la programación. La recursividad es una técnica en la que una función se llama a sí misma para resolver un problema. Aprenderemos cómo funciona y cuándo es apropiado utilizarla.

22.1 Conceptos Clave

22.1.1 Recursividad

Una técnica en la que una función se llama a sí misma para resolver un problema más grande.

22.1.2 Caso Base

Un caso en el que la función recursiva se detiene y no se llama a sí misma nuevamente.

22.1.3 Llamada Recursiva

La acción de una función llamándose a sí misma.

22.2 Ejemplo

```
# Función recursiva para calcular el factorial
def factorial(n):
    # Caso base
    if n == 1:
        return 1
    # Llamada recursiva
    else:
        return n * factorial(n - 1)

resultado = factorial(5)
print("El factorial de 5 es:", resultado)
```

22.2.1 Explicación:

En este ejemplo, definimos una función llamada factorial que calcula el factorial de un número n. Utilizamos la recursividad para dividir el problema en partes más pequeñas hasta llegar al caso base (cuando n es igual a 1). Luego, multiplicamos los resultados de las llamadas recursivas para obtener el factorial.

> Actividad Práctica
>
> Crea una función recursiva llamada sumatoria que calcule la suma de los números del 1 al n. Utiliza la recursividad para resolver este problema y luego llama a la función con diferentes valores de n para calcular sumatorias diferentes.

22.2.2 Explicación:

Esta actividad te permite practicar el concepto de recursividad al resolver un problema diferente. La recursividad es especialmente útil cuando un problema se puede descomponer en subproblemas similares.

22.3 ¿Qué aprendimos?

En esta lección, aprendimos qué es la recursividad y cómo funciona. Comprendimos los conceptos clave de la recursividad, incluyendo el caso base y las llamadas recursivas. La recursividad es una técnica poderosa que se utiliza en muchos algoritmos y problemas de programación.

Part VI

Unidad 6: Programación Orientada a Objetos

23 Programación Orientada a Objetos (POO)

En esta lección, exploraremos la Programación Orientada a Objetos (POO), un paradigma de programación que se basa en el uso de objetos y clases.

23.1 Conceptos Clave en POO

23.1.1 Objetos

Los objetos son instancias de clases y representan entidades del mundo real. Pueden tener atributos que describen sus características y métodos que definen su comportamiento.

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()
```

En este ejemplo, hemos creado una clase llamada “Persona”. La clase tiene un constructor (init) que inicializa los atributos “nombre” y “edad” de la persona. También tiene un método llamado “saludar” que muestra un mensaje con el nombre y la edad de la persona.

23.1.2 Clases

Las clases son plantillas o moldes que definen la estructura y el comportamiento de los objetos. En una clase, puedes especificar qué atributos y métodos tendrán sus objetos.

Ejemplo:

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

    def mostrar_info(self):
        print(f"Marca: {self.marca}, Modelo: {self.modelo}")

coche1 = Coche("Toyota", "Camry")
coche1.mostrar_info()
```

En este ejemplo, hemos creado una clase llamada “Coche” que tiene atributos “marca” y “modelo”. También tiene un método “mostrar_info” que imprime la información del coche.

23.1.3 Atributos

Los atributos son características o propiedades de un objeto que almacenan datos.

Ejemplo:

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

producto1 = Producto("Teléfono", 500)
print(f"Producto: {producto1.nombre}, Precio: {producto1.precio}")
```

En este ejemplo, hemos creado una clase “Producto” con atributos “nombre” y “precio”.

23.1.4 Métodos

Los métodos son funciones definidas en una clase que representan el comportamiento de los objetos de esa clase.

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")
perro1.ladrar()
```

En este ejemplo, hemos creado una clase “Perro” con el método “ladrar” que muestra un mensaje cuando un perro ladra.

Actividad Práctica

Crea una clase “Libro” que represente libros con atributos como “título” y “autor”. Luego, implementa un método llamado “mostrar_info” que imprima los atributos del libro. A continuación, crea una instancia de la clase “Libro” y llama al método “mostrar_info” para mostrar la información del libro.

23.1.5 Explicación:

Esta actividad te permitirá practicar la creación de clases y objetos, así como comprender cómo la Programación Orientada a Objetos nos ayuda a modelar y organizar nuestros programas de manera más efectiva.

23.2 ¿Qué aprendimos?

- Aprendimos los conceptos fundamentales de la Programación Orientada a Objetos (POO).
- Comprendimos cómo definir clases y objetos en Python.
- Practicamos la creación de atributos y métodos en una clase.
- Realizamos una actividad práctica para aplicar los conocimientos adquiridos en la creación de una clase y su uso.

24 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.

24.1 Instancias de Clase

Cuando se crea un objeto a partir de una clase, se crea una instancia de esa clase. Cada instancia es independiente y puede tener sus propios valores de atributos.

Ejemplo:

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

coche1 = Coche("Toyota", "Camry")
coche2 = Coche("Honda", "Civic")
```

En este ejemplo, hemos creado dos instancias de la clase “Coche” (coche1 y coche2) con diferentes valores de atributos “marca” y “modelo”. Atributos de Instancia

Los atributos de instancia son características específicas de un objeto que se almacenan como variables en la instancia. Cada objeto puede tener sus propios valores de atributos.

Ejemplo:

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

estudiante1 = Estudiante("Ana", 20)
estudiante2 = Estudiante("Juan", 22)
```

En este ejemplo, hemos creado dos instancias de la clase “Estudiante” con atributos “nombre” y “edad” que son específicos para cada estudiante.

24.1.1 Métodos de Instancia

Los métodos de instancia son funciones definidas en la clase que operan en los atributos de la instancia. Cada objeto puede llamar a los métodos de instancia para realizar acciones específicas.

Ejemplo:

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

    def maullar(self):
        print(f"{self.nombre} está maullando.")

gato1 = Gato("Mittens")
gato2 = Gato("Whiskers")

gato1.maullar()
gato2.maullar()
```

En este ejemplo, hemos creado dos instancias de la clase “Gato” y llamado al método “maullar” en cada gato para que realicen la acción específica.

> Actividad Práctica
>
> Crea una clase Rectángulo con atributos “ancho” y “alto”. Luego, implementa un método llamado “calcular_area” que calcule y retorne el área del rectángulo (ancho * alto).

Ejemplo de Clase Rectángulo

Resumen:

En este ejemplo, crearemos una clase llamada “Rectángulo” con atributos “ancho” y “alto”. Implementaremos un método llamado “calcular_area” que calculará y retornará el área del rectángulo (ancho * alto). Luego, crearemos dos instancias de la clase “Rectángulo” con diferentes dimensiones y mostraremos el área de cada rectángulo.

Resolución:

```
class Rectangulo:
    def __init__(self, ancho, alto):
        self.ancho = ancho
        self.alto = alto

    def calcular_area(self):
        return self.ancho * self.alto

# Crear dos instancias de Rectangulo
```

```
rectangulo1 = Rectangulo(5, 10)
rectangulo2 = Rectangulo(3, 7)

# Calcular el área de cada rectángulo
area1 = rectangulo1.calcular_area()
area2 = rectangulo2.calcular_area()

# Mostrar el área de cada rectángulo
print(f"Área del Rectángulo 1: {area1}")
print(f"Área del Rectángulo 2: {area2}")
```

Explicación:

① Definimos la clase "Rectangulo" con un constructor init que toma dos argumentos: "ancho" y "alto". Estos argumentos se utilizan para inicializar los atributos de instancia "ancho" y "alto".

24.2 ¿Qué aprendimos?

- Aprendimos los conceptos fundamentales de la Programación Orientada a Objetos (POO).
- Comprendimos cómo definir clases y objetos en Python.
- Practicamos la creación de atributos y métodos de instancia en una clase.
- Realizamos una actividad práctica para aplicar los conocimientos adquiridos en la creación de una clase y su uso.

25 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.

25.1 Conceptos Clave

25.1.1 Métodos de Clase

Los métodos de clase son funciones definidas dentro de una clase que operan en los atributos de instancia. Cada instancia de la clase puede llamar a estos métodos para realizar acciones específicas.

25.1.2 Acceso a Atributos

Dentro de un método, se puede acceder a los atributos de instancia utilizando "self.atributo". Esto permite manipular y utilizar los valores de los atributos dentro de los métodos.

Ejemplo

```
# Ejemplo de código en Python
# Puede incluir múltiples bloques de código si es necesario.

class Cuadrado:
    def __init__(self, lado):
        self.lado = lado

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

# Crear una instancia de Cuadrado
cuadrado1 = Cuadrado(4)

# Calcular y mostrar el área del cuadrado
area_cuadrado = cuadrado1.calcular_area()
print(f"Área del Cuadrado: {area_cuadrado}")
```

25.1.3 Explicación

En este ejemplo, hemos definido una clase “Cuadrado” con un método “calcular_area”. El método accede al atributo de instancia “lado” utilizando “self.lado” y calcula el área del cuadrado.

> Actividad Práctica
>
> Crea una clase Triángulo con atributos “base” y “altura”, y un método “calcular_area” que calcule y retorne el área del triángulo (base * altura / 2).

Ejemplo de Clase Triángulo

Resumen:

En esta actividad, crearemos una clase llamada “Triángulo” con atributos “base” y “altura”. Implementaremos un método llamado “calcular_area” que calculará y retornará el área del triángulo (base * altura / 2). Luego, crearemos una instancia de la clase “Triángulo” con dimensiones específicas y mostraremos el área del triángulo.

Resolución:

```
class Triangulo:
    def __init__(self, base, altura):
        self.base = base
        self.altura = altura

    def calcular_area(self):
        return (self.base * self.altura) / 2

# Crear una instancia de Triángulo
triangulo1 = Triangulo(6, 8)

# Calcular el área del triángulo
area_triangulo = triangulo1.calcular_area()

# Mostrar el área del triángulo
print(f"Área del Triángulo: {area_triangulo}")
```

25.1.4 Explicación:

① Definimos la clase “Triángulo” con un constructor init que toma dos argumentos: “base” y “altura”. Estos argumentos se utilizan para inicializar los atributos de instancia “base” y “altura”.

25.2 ¿Qué aprendimos?

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

Ahora tenemos una comprensión más sólida de cómo las clases pueden tener no solo atributos, sino también comportamientos definidos por métodos.

26 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.

26.1 Conceptos Clave

26.1.1 Self

La palabra clave "self" se refiere al objeto actual en un método de clase. Permite acceder y manipular los atributos de instancia dentro de ese método.

26.1.2 Eliminar Atributos

Se puede eliminar un atributo de instancia utilizando la palabra clave "del".

26.1.3 Eliminar Objetos

Para eliminar un objeto y liberar memoria, se utiliza la función "del" seguida del nombre del objeto.

Ejemplo

```
# Ejemplo de código en Python
# Puede incluir múltiples bloques de código si es necesario.

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

    def presentarse(self):
        print(f"Me llamo {self.nombre} y tengo {self.edad} años.")

# Crear una instancia de Persona
persona1 = Persona("Carlos", 28)

# Llamar al método para presentarse
persona1.presentarse()
```

26.1.4 Explicación

En este ejemplo, “self” se utiliza para acceder a los atributos “nombre” y “edad” dentro del método “presentarse”.

> Actividad Práctica
>
> Crea una clase Estudiante con atributos “nombre” y “edad”, y un método “mostrar_info” para mostrar la información del estudiante.

Ejemplo de Clase Estudiante

Resumen:

Este ejemplo demuestra cómo crear una clase Estudiante con atributos, un método para mostrar información y cómo eliminar atributos de instancia.

```
# Clase Estudiante
class Estudiante:
    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad

    def mostrar_info(self):
        print(f"Estudiante: {self.nombre}, Edad: {self.edad}")

# Crear una instancia de Estudiante
estudiante1 = Estudiante("María", 22)

# Llamar al método para mostrar la información
estudiante1.mostrar_info()

# Eliminar el atributo 'nombre' de la instancia
del estudiante1.nombre

# Intentar acceder al atributo eliminado generará un error
# estudiante1.mostrar_info()
```

Explicación:

① Hemos definido una clase llamada Estudiante con un constructor (init) que toma dos atributos: nombre y edad. 2. Estos atributos representan el nombre y la edad del estudiante.

26.2 ¿Qué aprendimos?:

En esta lección, hemos profundizado en el uso de “self” en los métodos de clase, cómo eliminar atributos de instancia y objetos en Python, y cómo gestionar la memoria.

27 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.

27.1 Conceptos Clave

27.1.1 Herencia

La herencia es un mecanismo que permite que una clase herede atributos y métodos de otra clase base. Esto facilita la creación de jerarquías de clases y la reutilización de código.

27.2 Ejemplo

```
# Ejemplo de código en Python
# Puede incluir múltiples bloques de código si es necesario.

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")

# Crear una instancia de la clase Perro
perro1 = Perro("Buddy")

# Llamar a métodos de la clase base y derivada
perro1.saludar()
perro1.ladrar()
```

27.2.1 Explicación

En este ejemplo, tenemos una clase base “Animal” con un constructor y un método “saludar”. Luego, creamos una clase derivada “Perro” que hereda de “Animal” y agrega su propio método “ladrar”. Las instancias de “Perro” heredan los atributos y métodos de “Animal”.

> Actividad Práctica
>
> Crea una clase Figura con un atributo “color” y un método “mostrar_color” para mostrar el color de la figura.
> Crea una clase derivada Circulo que herede de “Figura” y agregue un atributo “radio” y un método “calcular_area” para calcular el área del círculo.

Posible solución

Resumen:

En este código, se crea una clase base llamada “Figura” que tiene un atributo “color” y un método “mostrar_color”. Luego, se define una clase derivada “Circulo” que hereda de “Figura” y agrega un atributo “radio” y un método “calcular_area”. Se crea una instancia de “Circulo”, se muestra su color y se calcula su área.

```
# Definición de la clase base "Figura" con un constructor que toma el atributo "color".
class Figura:
    def __init__(self, color):
        self.color = color

    # Método en la clase base para mostrar el color de la figura.
    def mostrar_color(self):
        print(f"Color: {self.color}")

# Definición de la clase derivada "Circulo" que hereda de "Figura" y agrega un atributo "
class Circulo(Figura):
    def __init__(self, color, radio):
        # Llamamos al constructor de la clase base "Figura" utilizando "super()".
        super().__init__(color)
        self.radio = radio

    # Método en la clase derivada para calcular el área del círculo.
    def calcular_area(self):
        area = 3.14 * self.radio ** 2
        return area

# Creación de una instancia de la clase "Circulo" llamada "circulo1" con color "Rojo" y r
circulo1 = Circulo("Rojo", 5)

# Llamada al método "mostrar_color" de la clase base para mostrar el color del círculo.
circulo1.mostrar_color()
```

```
# Llamada al método "calcular_area" de la clase derivada para calcular el área del círcul
area = circulo1.calcular_area()

# Imprimir el resultado del cálculo del área.
print(f"Área del círculo: {area}")
```

27.2.2 Explicación:

En esta actividad, creamos una clase base “Figura” con un atributo “color” y un método “mostrar_color”. Luego, creamos una clase derivada “Circulo” que hereda de “Figura” y agrega un atributo “radio” y un método “calcular_area”. Las instancias de “Circulo” heredan los atributos y métodos de “Figura” y extienden su funcionalidad.

27.3 ¿Qué aprendimos?:

En esta lección, hemos explorado el concepto de herencia en la programación orientada a objetos y cómo crear clases derivadas que heredan atributos y métodos de una clase base. Esto nos permite reutilizar código y establecer relaciones jerárquicas entre clases.

28 Polimorfismo

En esta lección, exploraremos el concepto de polimorfismo en la programación orientada a objetos. Aprenderemos cómo el polimorfismo nos permite tratar objetos de diferentes clases de manera uniforme y cómo se implementa a través de métodos y herencia.

28.1 Conceptos Clave

28.1.1 Polimorfismo

El polimorfismo es un principio de la programación orientada a objetos que permite que objetos de diferentes clases sean tratados como objetos de una clase base común.

28.1.2 Métodos Polimórficos

Son métodos que pueden ser implementados de manera diferente en las clases derivadas, pero tienen el mismo nombre y firma en la clase base.

28.1.3 Sobreescritura de Métodos

La sobreescritura de métodos es la capacidad de una clase derivada para proporcionar una implementación específica de un método heredado de la clase base.

28.2 Ejemplo

```
class Animal:
    def sonido(self):
        pass

class Perro(Animal):
    def sonido(self):
        return "Guau"

class Gato(Animal):
    def sonido(self):
        return "Miau"
```

```
def hacer_sonar(animal):
    print(animal.sonido())

# Crear instancias de las clases
perro = Perro()
gato = Gato()

# Llamar a la función con objetos de diferentes clases
hacer_sonar(perro)  # Salida: Guau
hacer_sonar(gato)   # Salida: Miau
```

En este ejemplo, tenemos una clase base "Animal" con un método "sonido". Luego, creamos dos clases derivadas "Perro" y "Gato" que heredan de "Animal" y sobrescriben el método "sonido" para proporcionar su propio sonido característico.

La función "hacer_sonar" toma un objeto de la clase "Animal" como argumento y llama a su método "sonido". A pesar de que se pasan objetos de diferentes clases ("Perro" y "Gato"), el polimorfismo permite que el método "sonido" adecuado se ejecute para cada objeto.

> Actividad Práctica
>
> Crea una clase "Vehiculo" con un método "arrancar" que imprima "El vehículo arranca". Luego, crea clases derivadas "Coche" y "Motocicleta" que sobrescriban el método "arrancar" para proporcionar un mensaje específico para cada tipo de vehículo.

Solución

Resumen:

Definición de la Clase Base - "Vehiculo": Creamos una clase base llamada "Vehiculo", que representa vehículos en general. Esta clase base tiene un método llamado "arrancar" que imprimirá "El vehículo arranca" cuando se llame.

Definición de la Clase Derivada - "Coche": Creamos una clase derivada llamada "Coche" que hereda de la clase base "Vehiculo". La clase "Coche" sobrescribe el método "arrancar" para proporcionar una implementación específica: imprimir "El coche arranca" cuando se llame. **Definición de la Clase Derivada - "Motocicleta"**: Creamos otra clase derivada llamada "Motocicleta" que también hereda de la clase base "Vehiculo".

Al igual que la clase "Coche", la clase "Motocicleta" sobrescribe el método "arrancar" para proporcionar una implementación específica: imprimir "La motocicleta arranca" cuando se llame.

```
class Vehiculo:
    def arrancar(self):
        print("El vehículo arranca")

class Coche(Vehiculo):
```

```
    def arrancar(self):
        print("El coche arranca")

class Motocicleta(Vehiculo):
    def arrancar(self):
        print("La motocicleta arranca")
```

Explicación:

① Definimos la clase base "Animal" que tiene un método "sonido" vacío utilizando la declaración pass. Esta clase base servirá como base para las clases derivadas "Perro" y "Gato".

28.3 ¿Qué aprendimos?:

En esta lección, hemos explorado el concepto de polimorfismo en la programación orientada a objetos y cómo permite tratar objetos de diferentes clases de manera uniforme. Además, hemos visto cómo se implementa el polimorfismo a través de métodos polimórficos y la sobreescritura de métodos en clases derivadas.

29 Encapsulación

En esta lección, exploraremos el concepto de encapsulación en la programación orientada a objetos. Aprenderemos cómo se utiliza para ocultar los detalles internos de una clase y cómo se implementa en Python utilizando convenciones de nombres. Conceptos Clave:

29.0.1 Encapsulació:

La encapsulación es uno de los principios de la POO que consiste en ocultar los detalles internos de una clase y proporcionar una interfaz pública para interactuar con ella.

29.1 Atributos Privados:

En Python, se utiliza una convención de nombres para marcar atributos como privados agregando un guion bajo al principio del nombre (por ejemplo, _nombre).

29.2 Métodos Privados:

De manera similar, los métodos privados se marcan agregando un guion bajo al principio del nombre del método (por ejemplo, _calcular()).

29.3 Métodos de Acceso (Getters y Setters):

Los métodos de acceso permiten controlar el acceso a los atributos privados de una clase. Los métodos "get" obtienen el valor de un atributo y los métodos "set" lo modifican.

Ejemplo:

```
class CuentaBancaria:
    def __init__(self, saldo):
        # Atributo privado con un guion bajo al principio
        self._saldo = saldo

    # Método de acceso (Getter)
    def obtener_saldo(self):
        return self._saldo

    # Método de acceso (Setter)
```

```
    def depositar(self, cantidad):
        if cantidad > 0:
            self._saldo += cantidad

    # Método de acceso (Setter)
    def retirar(self, cantidad):
        if cantidad > 0 and cantidad <= self._saldo:
            self._saldo -= cantidad

# Crear una instancia de la clase CuentaBancaria
cuenta = CuentaBancaria(1000)

# Acceder al saldo utilizando el método de acceso
print("Saldo inicial:", cuenta.obtener_saldo())

# Realizar un depósito
cuenta.depositar(500)
print("Saldo después del depósito:", cuenta.obtener_saldo())

# Realizar un retiro
cuenta.retirar(200)
print("Saldo después del retiro:", cuenta.obtener_saldo())
```

29.3.1 Explicación:

En este ejemplo, la clase CuentaBancaria utiliza la convención de nombres con un guion bajo para marcar el atributo _saldo como privado. Los métodos obtener_saldo, depositar, y retirar proporcionan una interfaz pública para interactuar con la cuenta bancaria mientras ocultan los detalles internos.

> Actividad Práctica:
>
> Crea una clase Estudiante con un atributo privado _nombre. Implementa métodos de acceso get_nombre y set_nombre para obtener y establecer el nombre del estudiante.

Solución

Resumen:

En este código, se define una clase Estudiante con un atributo privado de _nombre y métodos de acceso (get_nombre y set_nombre) para obtener y cambiar el nombre del estudiante.

```
class Estudiante:
    def __init__(self, nombre):
        # Atributo privado con un guion bajo al principio
        self._nombre = nombre
```

```
    # Método de acceso (Getter)
    def get_nombre(self):
        return self._nombre

    # Método de acceso (Setter)
    def set_nombre(self, nuevo_nombre):
        if len(nuevo_nombre) > 0:
            self._nombre = nuevo_nombre

# Crea una instancia de la clase Estudiante con nombre "Juan"
estudiante = Estudiante("Juan")

# Acceder al nombre utilizando el método de acceso get_nombre
print("Nombre del estudiante:", estudiante.get_nombre())

# Cambiar el nombre utilizando el método de acceso set_nombre
estudiante.set_nombre("María")
# Imprimir el nombre después del cambio
print("Nombre del estudiante después del cambio:", estudiante.get_nombre())
```

Explicación:

① En este código, se crea una clase Estudiante con un atributo privado _nombre y dos métodos de acceso (get_nombre y set_nombre).

29.4 ¿Qué Aprendimos en esta Actividad?

Esta actividad te ayudará a practicar la encapsulación en Python utilizando métodos de acceso y atributos privados.

Part VII

Unidad 7: Módulos y Bases de Datos

30 Introducción a Módulos en Python

Los módulos son archivos que contienen código Python y se utilizan para organizar y reutilizar funciones, clases y variables en programas más grandes. Los módulos permiten una mejor estructuración del código y la creación de bibliotecas de funciones reutilizables.

30.0.1 Importar Módulos

Para utilizar un módulo en Python, se utiliza la palabra clave import seguida del nombre del módulo. Esto carga el módulo en el programa y permite acceder a sus funciones y clases.

Ejemplo:

```
# Ejemplo de código en Python

# Módulo calculadora.py
def suma(a, b):
    return a + b

# En otro archivo
import calculadora

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

30.0.2 Explicación

En este ejemplo, se crea un módulo llamado calculadora.py que contiene una función suma. Luego, en otro archivo, se importa el módulo calculadora y se utiliza la función suma del módulo para sumar dos números. Esto demuestra cómo organizar el código en módulos reutilizables y cómo importarlos en otros programas.

Actividad Práctica

1. Crea un módulo llamado matematicas con una función multiplicacion que multiplique dos números.

2. Explicación de la Actividad

3. En esta actividad, crearás un módulo llamado matematicas que contiene una función multiplicacion. Luego, importarás este módulo en otro archivo y uti-

lizarás la función multiplicacion para calcular el producto de dos números. Esta práctica te ayudará a comprender cómo trabajar con módulos en Python.

Solución

Resumen:

La solución a la actividad práctica consiste en crear el módulo matematicas con la función multiplicacion, importarlo en otro archivo y utilizar la función para multiplicar dos números.

```
# En el módulo matematicas.py
def multiplicacion(a, b):
    return a * b

# En otro archivo
import matematicas

resultado = matematicas.multiplicacion(4, 7)
print("Resultado de la multiplicación:", resultado)
```

Explicación

① Se crea un módulo llamado matematicas.py que contiene la función multiplicacion.

30.1 ¿Qué Aprendimos?

En esta lección, aprendimos a trabajar con módulos en Python, cómo importarlos en nuestros programas y cómo crear funciones reutilizables en módulos separados. También practicamos la importación y uso de módulos mediante una actividad práctica. La encapsulación es un concepto clave en la programación orientada a objetos (POO) que se refiere a la ocultación de los detalles internos de una clase y a la provisión de una interfaz pública para interactuar con ella. La encapsulación se implementa en Python utilizando convenciones de nombres para marcar atributos y métodos como privados, y mediante el uso de métodos de acceso (getters y setters) para controlar el acceso a los atributos privados.

31 Creando Nuestro Primer Módulo

En esta lección, aprenderemos a crear nuestro propio módulo en Python, lo que nos permitirá organizar y reutilizar código de manera efectiva. Crearemos un módulo que contendrá funciones y clases para realizar operaciones matemáticas básicas.

31.1 Pasos para Crear un Módulo:

- Crea un archivo de Python con la extensión .py.
- Define funciones y/o clases en el archivo.
- Guarda el archivo en una ubicación accesible.

Ejemplo:

```
# 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
## 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.

31.2 Pasos para Crear un Módulo:

① Crea un archivo de Python con la extensión .py.

31.3 Ejemplo:

```python
# 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
```

31.4 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:

1. Crea un módulo llamado geometria con funciones para calcular el área de un círculo y el perímetro de un cuadrado.
2. En otro archivo, importa el módulo geometria y utiliza las funciones para realizar cálculos geométricos.

31.5 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.

En este ejemplo, creamos un módulo llamado operaciones.py. Dentro de este módulo, definimos dos funciones, suma y resta, que realizan operaciones matemáticas básicas. También creamos una clase Calculadora con un método multiplicacion para llevar a cabo multiplicaciones.

Actividad Práctica:

Crea un módulo llamado geometria con funciones para calcular el área de un círculo y el perímetro de un cuadrado.

31.5.1 Explicación:

Esta actividad te permitirá practicar la creación de módulos con funciones. Debes crear un módulo llamado geometria.py que contenga dos funciones: una para calcular el área de

un círculo y otra para calcular el perímetro de un cuadrado. Luego, importa este módulo en otro archivo y utiliza las funciones para realizar cálculos geométricos.

Solución

La solución a la actividad práctica implica crear el módulo geometria.py con las funciones calcular_area_circulo y calcular_perimetro_cuadrado. Luego, importa este módulo en otro archivo y utiliza las funciones para realizar cálculos geométricos.

```
# geometria.py

import math

def calcular_area_circulo(radio):
    return math.pi * radio**2

def calcular_perimetro_cuadrado(lado):
    return 4 * lado
```

```
# archivo_principal.py

import geometria

radio_circulo = 5
lado_cuadrado = 4

area = geometria.calcular_area_circulo(radio_circulo)
perimetro = geometria.calcular_perimetro_cuadrado(lado_cuadrado)

print(f"Área del círculo: {area}")
print(f"Perímetro del cuadrado: {perimetro}")
```

Explicación paso a paso de la Solución:

① Creamos el módulo geometria.py que contiene dos funciones, calcular_area_circulo y calcular_perimetro_cuadrado, para calcular el área de un círculo y el perímetro de un cuadrado, respectivamente.

31.6 ¿Qué Aprendimos?

En esta lección, aprendimos a crear nuestro propio módulo en Python y cómo organizar funciones y clases en él. También practicamos la importación de módulos en otros archivos y cómo utilizar las funcionalidades proporcionadas por esos módulos. La creación de módulos es una técnica fundamental para mantener nuestro código organizado y promover la reutilización de código.

32 Renombrando Módulos y Seleccionando Elementos

En esta lección, exploraremos cómo renombrar módulos al importarlos y cómo seleccionar elementos específicos para importar.

Estas técnicas nos brindan un mayor control sobre los nombres y las funcionalidades que utilizamos en nuestro código.

32.0.1 Renombrando Módulos al Importar

A veces, los nombres de los módulos pueden ser largos o difíciles de recordar.

Podemos solucionar esto renombrando el módulo cuando lo importamos. Veamos cómo:

```
import modulo_largo as ml
```

32.0.2 Seleccionando Elementos Específicos para Importar

En lugar de importar todo un módulo, a veces solo necesitamos algunas funciones o clases específicas de ese módulo. Podemos hacer esto de la siguiente manera:

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

Ejemplo:

Supongamos que tenemos un módulo llamado calculadora y queremos abreviar su nombre al importarlo:

```
import calculadora as calc

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

Ejemplo

Si solo necesitamos unas pocas funciones o clases de un módulo grande, podemos importarlas directamente:

```
from operaciones import resta, Calculadora

resultado = resta(10, 5)
```

Actividad Práctica:

1. Renombra el módulo geometria como geo al importarlo en otro archivo.
2. Luego, importa solo la función para calcular el área de un círculo y calcula el área de un círculo con radio 5.

Solución

Resumen:

Para resolver esta actividad, renombraremos el módulo geometria como geo al importarlo en otro archivo. Luego, importaremos solo la función para calcular el área de un círculo y calcularemos el área de un círculo con radio 5.

Código:

```
# Renombrar el módulo geometria como geo al importarlo
import geometria as geo

# Importar solo la función para calcular el área de un círculo
from geometria import calcular_area_circulo

# Calcular el área de un círculo con radio 5
radio = 5
area = calcular_area_circulo(radio)
```

Explicación:

① En primer lugar, renombramos el módulo geometria como geo al importarlo. Esto significa que podemos usar geo como un alias para el módulo geometria en nuestro código.

32.0.3 Explicación:

Esta actividad te permitirá practicar cómo renombrar módulos al importarlos y cómo seleccionar funciones específicas para importar. Aprenderás a personalizar los nombres de los módulos y a importar solo las funcionalidades necesarias en tu código.

32.1 ¿Qué Aprendimos?

En esta lección, aprendimos dos técnicas útiles al trabajar con módulos en Python:

Renombrar Módulos al Importar: Pudimos asignar un nombre más corto o legible a un módulo al importarlo. Esto hace que nuestro código sea más claro y fácil de entender.

Seleccionar Elementos Específicos para Importar: Aprendimos cómo importar solo las funciones o clases necesarias de un módulo en lugar de importar todo el módulo. Esto puede ayudar a reducir la cantidad de código innecesario en nuestro programa.

Estas técnicas nos proporcionan un mayor control sobre cómo interactuamos con los módulos, lo que puede hacer que nuestro código sea más eficiente y legible.

33 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.

33.1 Conceptos Clave

33.1.1 Seleccionar Elementos para Importar

Cuando importamos módulos en Python, podemos seleccionar elementos específicos, como funciones o clases, en lugar de importar todo el módulo.

33.1.2 pip (Python Package Installer)

pip es una herramienta de línea de comandos que se utiliza para instalar, actualizar y desinstalar paquetes de Python. Los paquetes son conjuntos de módulos y recursos que se utilizan para ampliar la funcionalidad de Python.

33.2 Ejemplo - Instalando un Paquete con Pip

```
pip install requests
```

En este ejemplo, utilizamos pip para instalar el paquete requests, que es comúnmente utilizado para hacer solicitudes HTTP en Python.

> 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.

Ejemplo de Instalación de Paquete con Pip

Resumen:

En este ejemplo, instalaremos el paquete requests utilizando la herramienta pip.

```
pip install requests
```

Explicación:

① Utilizamos la línea de comando y el comando pip install seguido del nombre del paquete (requests) para instalarlo.

33.3 ¿Qué aprendimos?

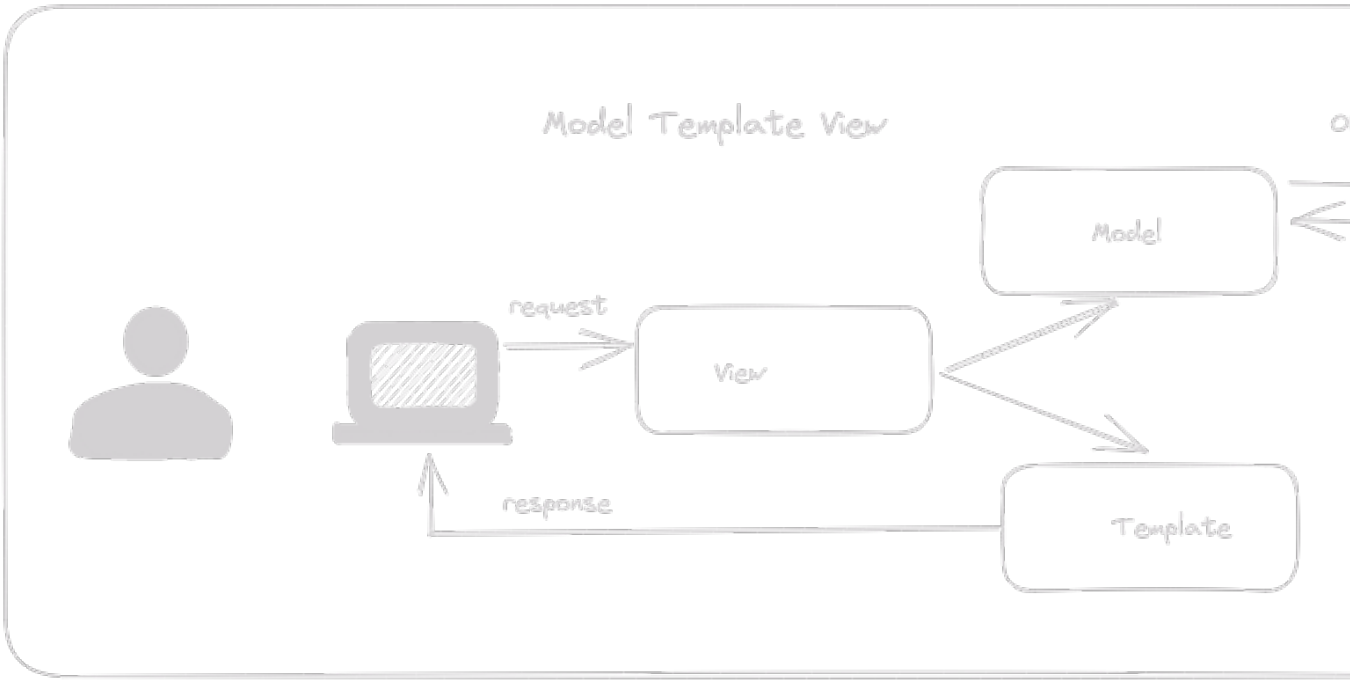
En esta lección, aprendimos a seleccionar elementos específicos para importar de un módulo en Python, lo que nos permite utilizar solo las funcionalidades que necesitamos en nuestro código. También aprendimos sobre pip, una herramienta esencial para instalar paquetes externos que amplían las capacidades de Python.

Part VIII

Unidad 8: Frameworks

34 Introducción a Django

Antes de iniciar con Django, es necesario conocer el patron de arquitectura que utiliza este Framework.



A diferencia de otro patrón muy conocido llamado Model View Controller.

Django es un marco de desarrollo web de alto nivel y de código abierto que facilita la creación de aplicaciones web rápidamente. En esta lección, introduciremos los conceptos básicos de Django y su arquitectura.

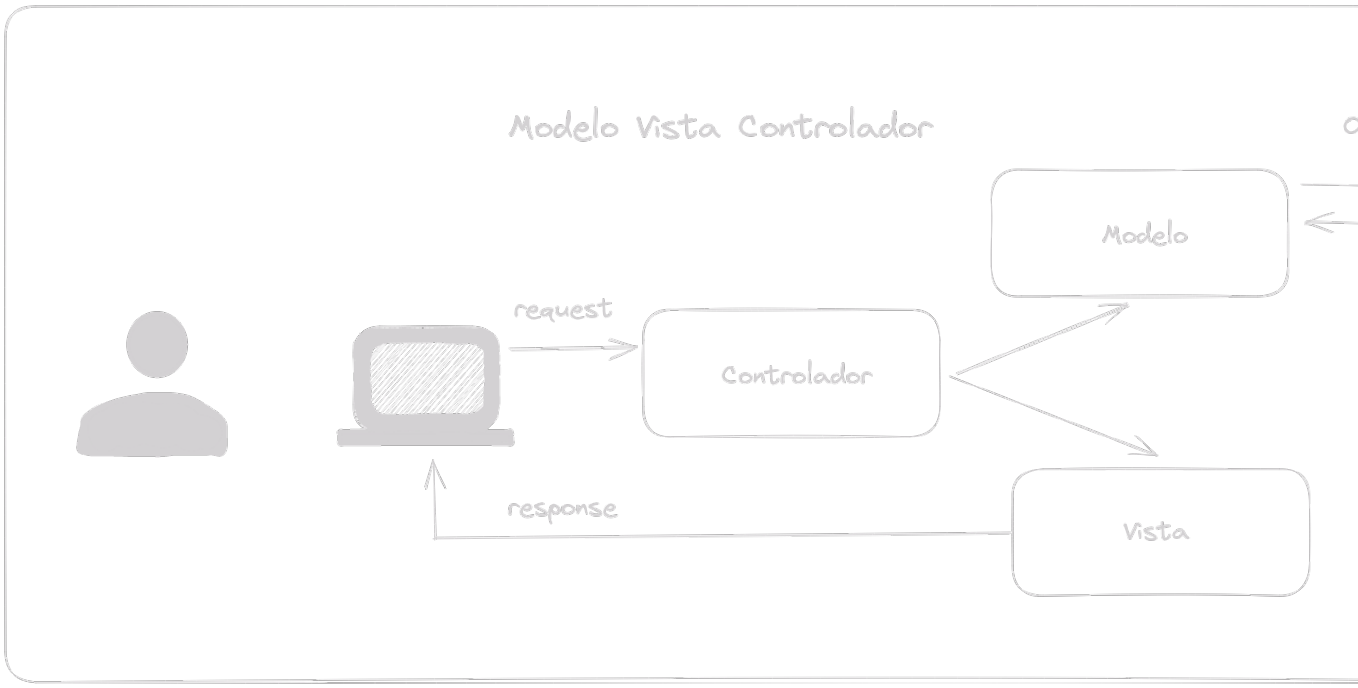
34.1 Conceptos Clave

Django es un marco de desarrollo web basado en Python que sigue el patrón de diseño Model Template View (MTV).

La arquitectura de Django se basa en proyectos y aplicaciones.

Un proyecto Django puede contener varias aplicaciones que se pueden reutilizar en diferentes proyectos.

Ejemplo




```
# Creación de un proyecto Django llamado "mi_proyecto"
django-admin startproject mi_proyecto

# Creación de una aplicación dentro del proyecto
cd mi_proyecto
python manage.py startapp mi_aplicacion
```

34.1.1 Explicación

En este ejemplo, hemos creado un proyecto Django llamado **mi_proyecto** utilizando el comando **django-admin startproject**.

Luego, dentro del proyecto, creamos una aplicación llamada **mi_aplicacion** utilizando **python manage.py startapp**.

Django organiza el código en proyectos y aplicaciones para mantenerlo modular y reutilizable.

Actividad Práctica

1. Crea un proyecto Django llamado “blog”.
2. Dentro del proyecto, crea una aplicación llamada “articulos”.
3. Verifica la estructura de carpetas generada por Django y explora los archivos creados.

Solución

Resumen:

En esta actividad práctica, se creará un proyecto Django llamado **blog** y se generará una aplicación llamada **articulos** dentro de ese proyecto. Luego, se explorará la estructura de carpetas y los archivos generados por Django.

Código:

Para crear el proyecto Django y la aplicación, sigue estos pasos:

① Abre una terminal o línea de comandos.

```
# Crear un proyecto Django llamado "blog"
django-admin startproject blog

# Cambia al directorio del proyecto
cd blog

# Crear una aplicación llamada "articulos"
python manage.py startapp articulos
```

Explicación:

① Utilizamos el comando django-admin startproject para crear un nuevo proyecto Django llamado **blog**. Esto generará la estructura de carpetas y los archivos iniciales del proyecto.

Una vez completados estos pasos, tendrás un proyecto **blog** con una aplicación **articulos** lista para desarrollar.

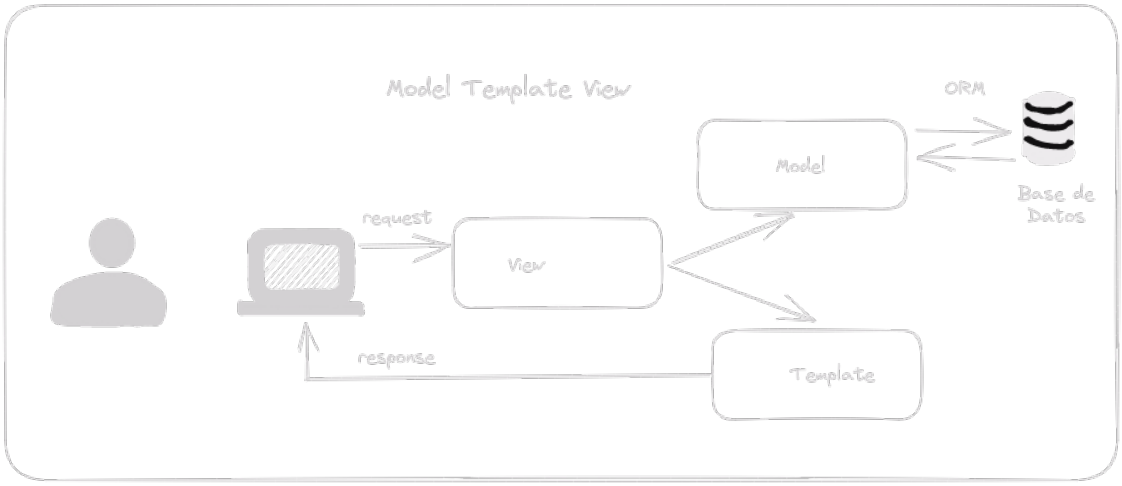
Puedes explorar la estructura de carpetas y archivos generados en el proyecto **blog** y la aplicación **articulos** para comprender mejor cómo se organiza un proyecto Django y cómo se crean las aplicaciones dentro de él. Esto proporciona la base para desarrollar una aplicación web utilizando Django.

34.2 ¿Qué aprendimos?

En esta lección, aprendimos los conceptos básicos de Django y cómo crear proyectos y aplicaciones en Django. También exploramos la estructura de carpetas generada por Django y su enfoque en la modularidad.

35 Creación de una Vista Hola Mundo en el Framework Django.

En esta lección, crearemos una vista en Django que mostrará un mensaje “Hola Mundo” en la página web.



35.1 Conceptos Clave

35.1.1 Vista

En Django, una vista es una función que toma una solicitud web y devuelve una respuesta. Se utiliza para definir qué contenido se muestra en una página web.

35.1.2 Plantillas

Las plantillas en Django son archivos HTML que permiten la presentación de datos dinámicos. Se utilizan para generar la interfaz de usuario de una aplicación web.

35.2 Ejemplo

```
# En el archivo views.py de la aplicación Django
from django.http import HttpResponse

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

35.2.1 Explicación

En este ejemplo, hemos creado una vista llamada hola_mundo en Django. Esta vista toma una solicitud web y devuelve una respuesta que consiste en el mensaje "¡Hola Mundo!".

35.2.2 Urls.py

Ahora podemos configurar una URL para que apunte a esta vista, de modo que cuando un usuario acceda a esa **URL**, verá el mensaje "**¡Hola Mundo!**" en la página.

```
# En el archivo urls.py de la aplicación Django
from django.urls import path
from . import views

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

Configuración del archivo settings.py

```
INSTALLED_APPS = [
    # ...
    'hola_mundo_app',
    # ...
]
```

Archivo urls.py del proyecto principal


```
from django.contrib import admin
from django.urls import include, path

urlpatterns = [
    path('admin/', admin.site.urls),
    path('hola-mundo/', include('hola_mundo_app.urls')),
]
```

Tip

Actividad Práctica

1. Crea una aplicación Django llamada “hola_mundo_app” en tu proyecto.
2. Crea una vista similar a la vista hola_mundo en esta lección en el archivo views.py de la aplicación “hola_mundo_app”.

3. Configura una URL para que apunte a esta vista en el archivo urls.py de la aplicación “hola_mundo_app”.
4. Ejecuta tu servidor Django y accede a la URL correspondiente para ver el mensaje “¡Hola Mundo!” en tu navegador.

Solución

Resumen:

Hemos creado una vista en Django que muestra el mensaje “¡Hola Mundo!” en la página web.

Resolución:

① Creamos una vista llamada hola_mundo en views.py.

```
# En el archivo views.py de la aplicación "hola_mundo_app"
from django.http import HttpResponse

def hola_mundo_personalizado(request):
    return HttpResponse("¡Hola Mundo Personalizado!")
```

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

urlpatterns = [
    path('', views.hola_mundo_personalizado, name='hola_mundo_personalizado'),
]
```

```
python manage.py runserver
```

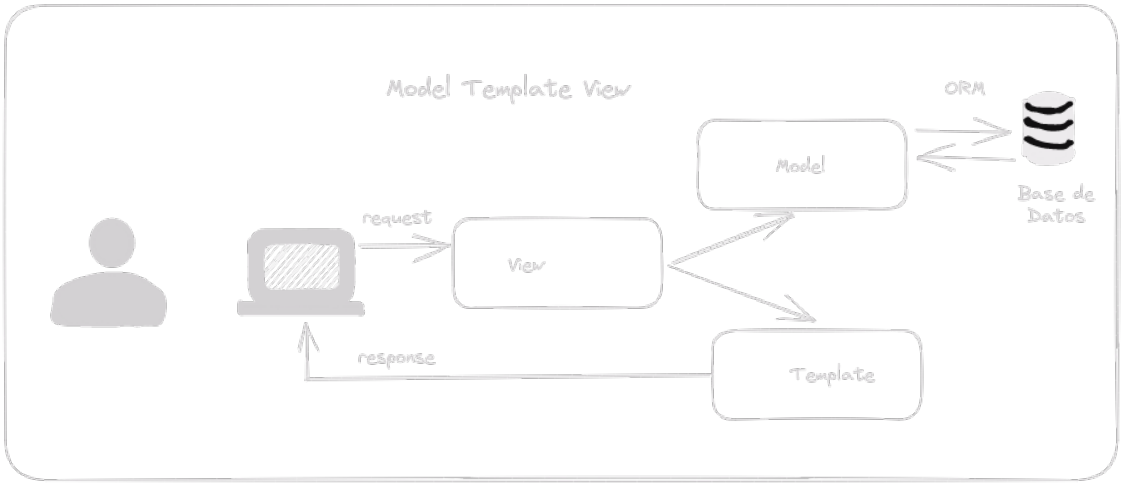
Explicación:

La vista hola_mundo es una función que toma una solicitud web y devuelve una respuesta que contiene el mensaje “¡Hola Mundo!”. Configuramos una URL para que los usuarios puedan acceder a esta vista y ver el mensaje en la página.

35.3 ¿Qué aprendimos?

En esta lección, aprendimos cómo crear una vista en Django y cóm/unidades/unidad8/unidad8.2.htmlo configurar una URL para mostrar contenido en una página web. También vimos cómo se utiliza la plantilla de la vista para generar la respuesta que se muestra al usuario.

36 Model Template View en Django.



En esta lección, profundizaremos en el patrón de arquitectura que utiliza el Framework Django Modelo-Template-View (MTV), adicional a ello vamos a aprender cómo se aplica al crear un **CRUD de tareas**.

36.1 Conceptos Clave

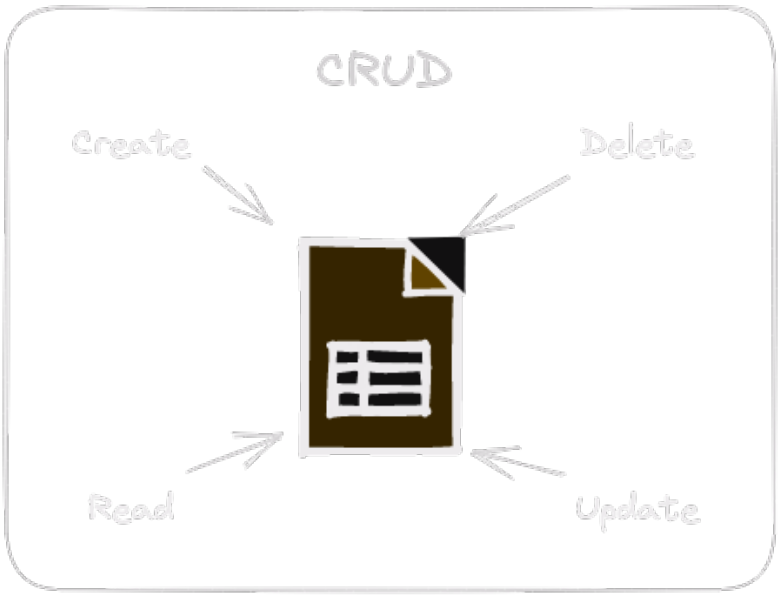
36.1.1 CRUD.

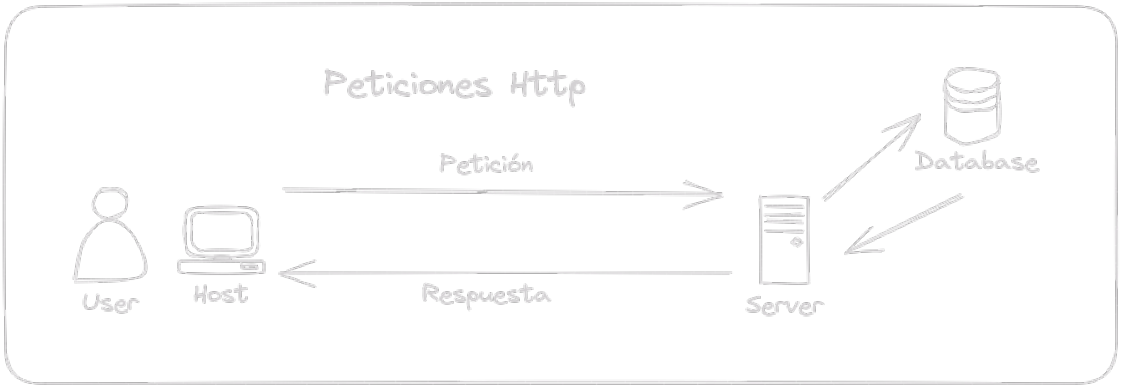
Crud es un acrónimo conformado por las iniciales de estas 4 palabras en ingles:

Create	Crear
Read	Leer
Update	Actualizar
Delete	Eliminar

Esto hace referencia al desarrollo de una aplicación que permita Crear registros, Leer registros, Actualizar registros y Eliminar registros, en esta clase trabajaremos un **CRUD** de **Tareas** para aprender del proceso.

Un punto importante es conocer acerca de las peticiones http, ya que es la forma en como nos comunicamos con un servidor creado por un framework web en la actualidad.


CRUD
Create
Delete
Read
Update


Peticiones Http
Petición
Database
User
Host
Respuesta
Server

36.1.2 Peticiones Http.

Gracias a este artículo Mozilla (2023) de Mozilla Developers podemos conocer los métodos básicos crear peticiones http.

Método	Concepto
GET	El método **GET** solicita una representación de un recurso específico. Las peticiones que usan el método **GET** sólo deben recuperar datos.
POST	El método **POST** se utiliza para enviar una entidad a un recurso en específico, causando a menudo un cambio en el estado o efectos secundarios en el servidor.
PUT	El modo **PUT** reemplaza todas las representaciones actuales del recurso de destino con la carga útil de la petición.
DELETE	El método **DELETE** borra un recurso en específico.
PATCH	El método **PATCH** es utilizado para aplicar modificaciones parciales a un recurso.

Esta información es de suma importancia cuando creemos un **API Rest** con **Django Rest Framework**

36.1.3 Modelo (Model).

El Modelo en Django define la estructura de la base de datos y cómo se almacenan los datos. Cada modelo corresponde a una tabla en la base de datos.

36.1.4 Plantilla (Template).

Las Plantillas en Django son archivos **HTML** que definen la estructura visual de las páginas web. Permiten la presentación de datos a los usuarios.

36.1.5 Vista (View).

Las Vistas en el Framework Django controlan **qué datos** se muestran en una **página web** y **cómo se presentan**. Se encargan de la **lógica de negocio** y trabajan con los **modelos** y **plantillas**.

Ejemplo de Modelo

```
# Definición de un modelo para crear Tareas en Django
from django.db import models

class Tarea(models.Model):
    titulo = models.CharField(max_length=200)
    descripcion = models.TextField()
    fecha_creacion = models.DateTimeField('fecha de creación')

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

Ejemplo de Plantilla

```
# Plantilla HTML para mostrar la lista de tareas
<!DOCTYPE html>
<html>
<head>
    <title>Lista de Tareas</title>
</head>
<body>
    <h1>Tareas Pendientes</h1>
    <ul>
        {% for tarea in lista_tareas %}
            <li>{{ tarea.titulo }}</li>
        {% endfor %}
    </ul>
</body>
</html>
```

Ejemplo de Vista

```
# Vista en Django para mostrar la lista de tareas
from django.shortcuts import render
from .models import Tarea

def lista_tareas(request):
    tareas = Tarea.objects.all()
    return render(request, 'lista_tareas.html', {'lista_tareas': tareas})
```

Ejemplo del archivo url de la aplicación

```
# En el archivo urls.py de la aplicación Django
from django.urls import path
from . import views

urlpatterns = [
    path('tareas/', views.lista_tareas, name='lista_tareas'),
]
```

Ejemplo del archivo url del proyecto

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('tarea.urls')),
]
```

Ejemplo de la modificación del archivo settings.py

```
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'tarea',
]
```

Ejemplo de la modificación del archivo admin.py de la aplicación

```
from django.contrib import admin
from .models import Tarea

admin.site.register(Tarea)
```

36.1.6 Explicación

En este ejemplo, hemos definido un modelo **Tarea** que representa una tarea en nuestra base de datos. Luego, creamos una plantilla HTML **lista_tareas.html** que muestra una lista de tareas. Finalmente, creamos una vista **lista_tareas** que recupera todas las tareas y las muestra utilizando la plantilla.

> Actividad Práctica
>
> 1. Crea un **modelo** Django llamado **Nota** que tenga un campo para el **título** y otro para el **contenido** de la nota.
> 2. Crea una **plantilla** HTML llamada **lista_notas.html** que muestre una **lista de notas.**
> 3. Crea una **vista** en Django llamada **lista_notas** que recupere todas las **notas** y las muestre utilizando la plantilla.

Solucion

Resumen:

En esta actividad práctica, se creará un modelo Django llamado "Nota" con campos para el título y el contenido de la nota. Luego, se creará una plantilla HTML llamada "lista_notas.html" para mostrar una lista de notas. Finalmente, se creará una vista Django llamada "lista_notas" que recuperará todas las notas y las mostrará utilizando la plantilla.

Código:

① Crear el Modelo Django **Nota**:

En el archivo **models.py** de la aplicación correspondiente (generalmente llamada **articulos**), define el modelo **Nota** con campos para el **título** y el **contenido** de la **nota**:

```
# Importar el módulo 'models' de Django
from django.db import models

class Nota(models.Model):
    # Definir los campos del modelo Nota
    titulo = models.CharField(max_length=200)
    contenido = models.TextField()

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

Dentro de la carpeta de la aplicación (**articulos** en este ejemplo), crea una carpeta llamada **templates** si aún no existe. Luego, crea un archivo HTML llamado **lista_notas.html** en la carpeta "templates" con el siguiente contenido:

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Lista de Notas</title>
</head>
<body>
    <h1>Lista de Notas</h1>
    <ul>
        {% for nota in notas %}
            <li>{{ nota.titulo }}</li>
            <p>{{ nota.contenido }}</p>
        {% endfor %}
    </ul>
</body>
</html>
```

En el archivo **views.py** de la aplicación (**articulos** en este ejemplo), crea una vista llamada **lista_notas** que recupere todas las notas y las pase a la plantilla **lista_notas.html**:

```
# Importar el módulo 'render' de Django y el modelo 'Nota'
from django.shortcuts import render
from .models import Nota

def lista_notas(request):
    # Recuperar todas las notas de la base de datos
    notas = Nota.objects.all()
    # Renderizar la plantilla 'lista_notas.html' con las notas como contexto
    return render(request, 'lista_notas.html', {'notas': notas})
```

Explicación:

① Hemos creado un modelo Django llamado **Nota** que contiene dos campos: **titulo** y **contenido**." El campo **titulo** es un CharField con una longitud máxima de 200 caracteres, y el campo **contenido** es un TextField para almacenar el contenido más extenso de la nota.

Con estos pasos, hemos configurado un modelo, una plantilla y una vista para mostrar una lista de notas en una aplicación Django.

Ahora puedes acceder a la vista **lista_notas** en tu aplicación para ver la lista de notas en la plantilla correspondiente.

36.2 ¿Qué aprendimos?

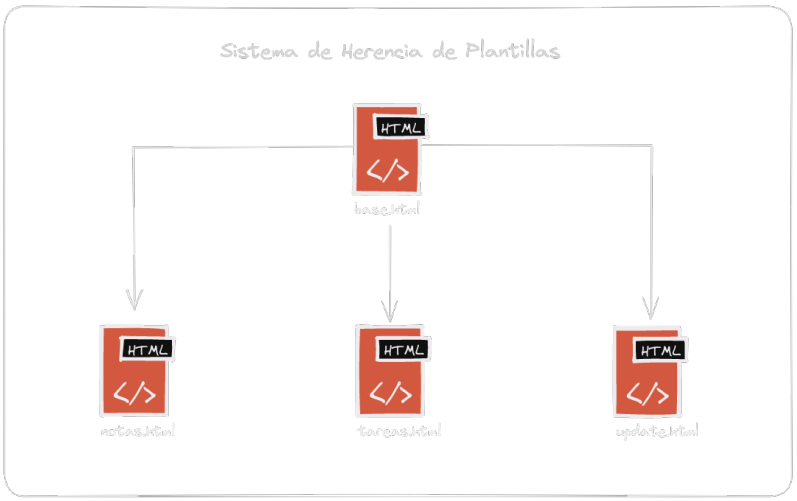
En esta lección, aprendimos cómo se aplican los conceptos **Modelo**, **Plantilla** y **Vista** en el Framework Django para crear una **lista de tareas**. Entendemos cómo este framework maneja la **lógica de negocio** y la **presentación de datos** de manera separada.

37 Herencia de Plantillas y Bootstrap en Django

En esta lección, aprenderemos cómo utilizar la **herencia de plantillas** en Django para crear un diseño consistente en nuestras páginas web y cómo integrar **Bootstrap** para mejorar la apariencia de la interfaz de usuario.

37.1 Conceptos Clave

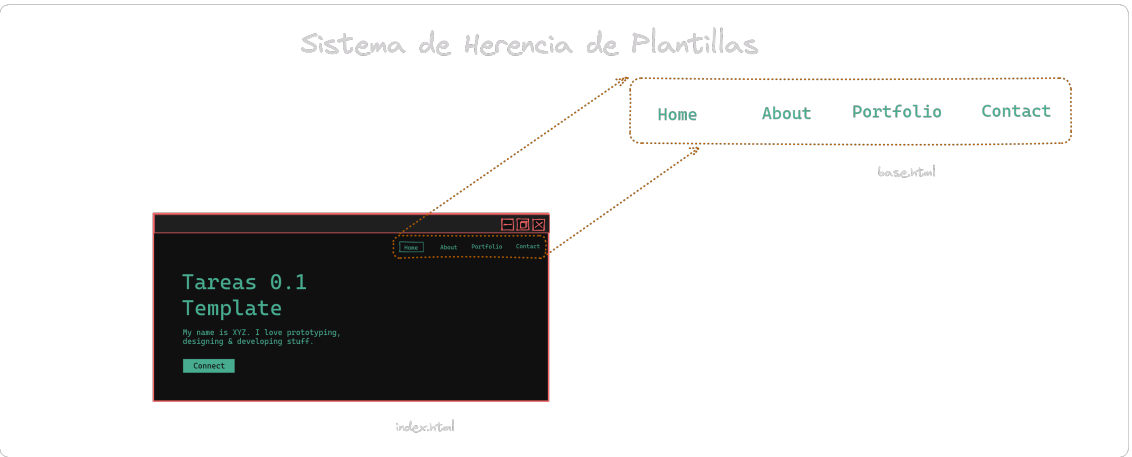
37.1.1 Herencia de Plantillas



En Django el sistema de **herencia** de plantillas es una técnica que permite a los desarrolladores crear una plantilla **base** que contiene elementos comunes a todas las páginas web de un sitio. Luego, se pueden crear plantillas **secundarias** que heredan de la plantilla **base** y agregan elementos específicos para cada página web.

Para usar la herencia de plantillas en Django, primero debes crear una plantilla base. Esta plantilla debe contener todo lo que es común a todas las páginas web de tu sitio, como el encabezado y el pie de página. Luego, puedes crear plantillas secundarias que heredan de la plantilla base y agregan elementos específicos para cada página web.

Para heredar una plantilla en Django, debes usar la directiva **{% extends %}**. Esta directiva le dice a Django que la plantilla actual hereda de otra plantilla. La plantilla actual puede invalidar bloques de contenido definidos en la plantilla base usando la directiva **{% block %}**.



En un sistema de tareas creado con Django, puedes heredar cualquier cosa que sea común a todas las páginas web del sitio. Por ejemplo, puedes heredar el **encabezado** y el **pie de página**, así como cualquier **estilo CSS** o **JavaScript** utilizado en todo el sitio.

También puedes agregar elementos específicos para cada página web, como el **título** y el **contenido** de la tarea.

37.1.2 Bootstrap.

Bootstrap es un marco de diseño de código abierto que proporciona estilos y componentes predefinidos para mejorar la apariencia y la usabilidad de un sitio web.

Vamos a continuar donde nos quedamos en el proyecto anterior,

① En este punto vamos a crear un archivo base que permita heredar a las demás plantillas parámetros generales, para este ejemplo.

Ejemplo de Plantilla Base.

```
<!DOCTYPE html>
<html>
<head>
    <title>{% block title %}Mi Sitio Web{% endblock %}</title>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bo
</head>
<body>
    <div class="container">
        <header>
            <h1>Mi Sitio Web</h1>
        </header>
        <nav>
            <ul class="nav">
                <li class="nav-item"><a class="nav-link" href="/">Inicio</a></li>
                <li class="nav-item"><a class="nav-link" href="/tareas/">Tareas</a></li>
            </ul>
        </nav>
        <main>
            {% block content %}
            {% endblock %}
        </main>
    </div>
</body>
</html>
```

Plantilla que hereda de **base.html** para la **página de inicio** (**inicio.html**).

Ahora creamos una plantilla my básica que va a ser nuestra plantilla de Inicio, es decir el lugar donde llegarán los usuarios con sus peticiones cuando ingresen a nuestro proyecto.

① Las plantillas que hereden de base deben adoptar esta estructura base, de esa forma cada elemento que se encuentre en el archivo base será heredado a las plantillas hijas sin la necesidad de copiar y pegar código como una mala práctica.

```
# Plantilla para la página de inicio que hereda de "base.html"
{% extends "base.html" %}

{% block title %}Inicio - Mi Sitio Web{% endblock %}

{% block content %}
    <h2>Bienvenido a la página de inicio</h2>
    <p>Esta es la página de inicio de mi sitio web.</p>
{% endblock %}
```

Es necesario modificar un poco los archivos views.py y urls.py de la aplicación y el archivo urls.py del proyecto para hacer uso de las nuevas plantillas.

```python
# En el archivo urls.py de la aplicación Django
from django.urls import path
from . import views

urlpatterns = [
    path('', views.lista_tareas2, name='inicio'),
    path('tareas/', views.lista_tareas, name='lista_tareas'),
    path('tareas_2/', views.lista_tareas2, name='lista_tareas'),
]
```

```python
# Vista en Django para mostrar la lista de tareas
from django.shortcuts import render
from .models import Tarea

def lista_tareas(request):
    tareas = Tarea.objects.all()
    return render(request, 'lista_tareas.html', {'lista_tareas': tareas})

def lista_tareas2(request):
    tareas = Tarea.objects.all()
    return render(request, 'inicio.html', {'lista_tareas': tareas})

def inicio(request):
    return render(request, 'inicio.html', 'inicio' )
```

```python
# urls.py del proyecto
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('tarea.urls')),
]
```

Con estos cambios podemos correr el servidor y observar los cambios realizados.

37.1.3 Explicación

En este ejemplo, hemos creado una plantilla base llamada **base.html** que define la estructura común de todas las páginas. Luego, creamos una plantilla específica para la página de inicio que hereda de **base.html**. Utilizamos **Bootstrap** para mejorar el aspecto de la página.

Actividad Práctica

1. Crea una **plantilla base** llamada **base.html** que incluya un **menú de navegación** y **un pie de página**.

2. Crea una **plantilla** llamada **lista_notas.html** que herede de **base.html** y muestre una lista de notas utilizando **Bootstrap**.

Solución

Creación de Plantillas Base y Específica en Django

Resumen:

En esta actividad práctica, se crearán plantillas en Django. Primero, se creará una plantilla base llamada **base.html** que contendrá un menú de navegación y un pie de página. Luego, se creará una plantilla específica llamada **lista_notas.html** que heredará de **base.html** y mostrará una lista de notas utilizando Bootstrap.

Código:

① **Crear la Plantilla Base** base.html:

Crea un archivo llamado lista_notas.html en la misma carpeta de plantillas de tu aplicación ("artículos" en este ejemplo). Esta plantilla heredará de base.html y mostrará una lista de notas utilizando Bootstrap:

```
{% extends "base.html" %}

{% block title %}Lista de Notas{% endblock %}

{% block content %}
    <h1>Lista de Notas</h1>
    <ul class="list-group">
        {% for nota in notas %}
            <li class="list-group-item">{{ nota.titulo }}</li>
            <p class="list-group-item">{{ nota.contenido }}</p>
        {% endfor %}
    </ul>
{% endblock %}
```

Explicación:

① Hemos creado una plantilla base llamada "base.html" que incluye un menú de navegación en la parte superior y un pie de página en la parte inferior. Esta plantilla utiliza Bootstrap para el estilo del menú y el pie de página.

37.2 ¿Qué aprendimos?

Aprendimos cómo utilizar la herencia de plantillas en Django para crear un diseño consistente en nuestras páginas web y cómo integrar Bootstrap para mejorar la apariencia de la interfaz de usuario. Esta práctica facilitará la creación de páginas similares en tu aplicación Django y mejorará la experiencia del usuario.

38 Creación de un CRUD de Tareas en Django

En esta lección, aprenderemos cómo crear un **CRUD** de Tareas en Django, utilizando **modelos**, **vistas** y **formularios**.

38.1 Conceptos Clave

38.1.1 CRUD

CRUD es un acrónimo de "Crear, Leer, Actualizar, Eliminar" y se refiere a las operaciones básicas de manipulación de datos en una aplicación.

38.1.2 Formularios en Django

Los formularios en Django nos permiten **crear** y **manejar** formularios HTML de manera sencilla y eficiente.

Ahora vamos a conocer el desarrollo de este proyecto paso a paso, empezamos a conocer cada uno de los archivos, su ubicación, el código necesario y una breve explicación de que se hace en cada archivo.

models.py: en la carpeta de la aplicación tarea en el proyecto Tareas.

Código:

```
from django.db import models

class Tarea(models.Model):
    titulo = models.CharField(max_length=200)
    descripcion = models.TextField()
    fecha_creacion = models.DateTimeField(auto_now_add=True)
    # Campo para registrar la fecha de actualización
    fecha_actualizacion = models.DateTimeField(auto_now=True)
```

Explicación:

① Este archivo, models.py, define el modelo de datos para la aplicación tarea.

forms.py en la carpeta de la aplicación tarea en el proyecto Tareas.

Código:

```
from django import forms
from .models import Tarea

class TareaForm(forms.ModelForm):
    class Meta:
        model = Tarea
        fields = ['titulo', 'descripcion']
```

Explicación:

① Este archivo, forms.py, define un formulario en Django para crear una tarea.

urls.py en la carpeta de la aplicación tarea en el proyecto Tareas.

Código:

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

urlpatterns = [
    path('', views.lista_tareas, name='lista_tareas'),
    path('crear_tarea/', views.crear_tarea, name='crear_tarea'),
    path('detalle_tarea/<int:pk>/', views.detalle_tarea, name='detalle_tarea'),
    path('editar_tarea/<int:pk>/', views.editar_tarea, name='editar_tarea'),
    path('eliminar_tarea/<int:pk>/', views.eliminar_tarea, name='eliminar_tarea'),
]
```

Explicación:

① Este archivo, urls.py, define las URL y las vistas asociadas para la aplicación tarea en el proyecto Tareas.

admin.py en la carpeta de la aplicación tarea en el proyecto Tareas.

Código:

```
from django.contrib import admin
from .models import Tarea

# Registra el modelo Tarea en el panel de administración
admin.site.register(Tarea)
```

Explicación:

① En este archivo, admin.py, importamos admin de django.contrib y el modelo Tarea desde .models.

views.py en la carpeta de la aplicación tarea en el proyecto Tareas.

Código:

```
from django.shortcuts import render, get_object_or_404, redirect
from .models import Tarea
from .forms import TareaForm

def lista_tareas(request):
    # Obtiene todas las tareas
    tareas = Tarea.objects.all()
    return render(request, 'tarea/lista_tareas.html', {'tareas': tareas})

def crear_tarea(request):
    if request.method == "POST":
        # Si se envía el formulario, procesa los datos
        form = TareaForm(request.POST)
        if form.is_valid():
            tarea = form.save()
            return redirect('detalle_tarea', pk=tarea.pk)
    else:
        # Si no se envía el formulario, muestra un formulario vacío
        form = TareaForm()
    return render(request, 'tarea/crear_tarea.html', {'form': form})

def detalle_tarea(request, pk):
    # Obtiene la tarea específica por su clave primaria (ID)
    tarea = get_object_or_404(Tarea, pk=pk)
    return render(request, 'tarea/detalle_tarea.html', {'tarea': tarea})

def editar_tarea(request, pk):
    # Obtiene la tarea específica por su clave primaria (ID)
    tarea = get_object_or_404(Tarea, pk=pk)
    if request.method == "POST":
        # Si se envía el formulario, procesa los datos
        form = TareaForm(request.POST, instance=tarea)
        if form.is_valid():
            tarea = form.save()
            return redirect('detalle_tarea', pk=tarea.pk)
    else:
        # Si no se envía el formulario, muestra el formulario con los datos de la tarea
        form = TareaForm(instance=tarea)
    return render(request, 'tarea/editar_tarea.html', {'form': form})

def eliminar_tarea(request, pk):
    # Obtiene la tarea específica por su clave primaria (ID)
    tarea = get_object_or_404(Tarea, pk=pk)
    if request.method == "POST":
        # Si se confirma la eliminación, elimina la tarea
```

```
        tarea.delete()
        return redirect('lista_tareas')
    return render(request, 'tarea/confirmar_eliminar.html', {'tarea': tarea})
```

Explicación:

① En este archivo, views.py, definimos las vistas que se encargan de manejar las acciones relacionadas con las tareas en nuestro proyecto.

Estas vistas forman parte de la funcionalidad CRUD (Crear, Leer, Actualizar, Eliminar) de nuestro sistema de tareas en Django.

base.html en la carpeta templates de la aplicación tarea en el proyecto Tareas.

Código:

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>{% block title %}Título por Defecto{% endblock %}</title>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bo
</head>
<body>
    <nav class="navbar navbar-expand-lg navbar-light bg-light">
        <a class="navbar-brand" href="#">Mi Aplicación de Tareas</a>
        <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="
            <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarNav">
            <ul class="navbar-nav">
                <li class="nav-item active">
                    <a class="nav-link" href="{% url 'lista_tareas' %}">Inicio <span clas
                </li>
                <li class="nav-item">
                    <a class="nav-link" href="{% url 'crear_tarea' %}">Crear Tarea</a>
                </li>
            </ul>
        </div>
    </nav>

    <div class="container">
        {% block content %}{% endblock %}
    </div>

    <footer class="footer mt-auto py-3">
        <div class="container">
            <span class="text-muted">© {% now "Y" %} Mi Aplicación de Tareas</span>
        </div>
```

```
    </footer>
    <!-- Agregar enlaces a Bootstrap y otros recursos aquí -->
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/js/bootstrap.min.js"><
</body>
</html>
```

Explicación:

① En este archivo HTML, hemos definido la estructura base de nuestras páginas web. Sirve como plantilla para todas las páginas y define la estructura común, incluyendo el encabezado de navegación, el contenido y el pie de página.

crear_tarea.html en la carpeta templates de la aplicación tarea en el proyecto Tareas.

Código:

```
{% extends "base.html" %}

{% block title %}Crear Tarea{% endblock %}

{% block content %}
    <h1>Crear Tarea</h1>
    <form method="POST">
        {% csrf_token %}
        {{ form.as_p }}
        <button type="submit" class="btn btn-success">Guardar</button>
    </form>
    <a class="btn btn-secondary" href="{% url 'lista_tareas' %}">Volver a la Lista de Tar
{% endblock %}
```

Explicación:

① Este archivo HTML hereda de la plantilla base "base.html" utilizando la directiva {% extends %}. Esto significa que incluirá todo el contenido de la plantilla base y permitirá personalizar bloques específicos.

En resumen, esta plantilla se utiliza para mostrar el formulario de creación de tareas y aprovecha la plantilla base para mantener una estructura y estilo consistentes en el sitio web.

detalle_tarea.html en la carpeta templates de la aplicación tarea en el proyecto Tareas.

Código:

```
{% extends "base.html" %}

{% block title %}Detalle de Tarea{% endblock %}

{% block content %}
```

```
    <h1>Detalle de Tarea</h1>
    <h3>{{ tarea.titulo }}</h3>
    <p>{{ tarea.descripcion }}</p>
    <a class="btn btn-primary" href="{% url 'editar_tarea' tarea.id %}">Editar Tarea</a>
    <a class="btn btn-danger" href="{% url 'eliminar_tarea' tarea.id %}">Eliminar Tarea</
    <a class="btn btn-secondary" href="{% url 'lista_tareas' %}">Volver a la Lista de Tar
{% endblock %}
```

Explicación:

① Este archivo HTML también hereda de la plantilla base "base.html" utilizando la directiva {% extends %}. Esto permite que comparta la estructura común y el estilo con otras páginas del sitio.

En resumen, esta plantilla se utiliza para mostrar los detalles de una tarea específica y proporciona enlaces para realizar acciones relacionadas con esa tarea.

editar_tarea.html en la carpeta templates de la aplicación tarea en el proyecto Tareas.

Código:

```
{% extends "base.html" %}

{% block title %}Editar Tarea{% endblock %}

{% block content %}
    <h1>Editar Tarea</h1>
    <form method="POST">
        {% csrf_token %}
        {{ form.as_p }}
        <button type="submit" class="btn btn-success">Guardar Cambios</button>
    </form>
    <a class="btn btn-secondary" href="{% url 'detalle_tarea' tarea.id %}">Volver al Deta
{% endblock %}
```

Explicación:

① Al igual que otras plantillas, esta también hereda de la plantilla base "base.html" utilizando {% extends %}.

Esta plantilla se utiliza para la edición de tareas y muestra un formulario que permite al usuario modificar los datos de una tarea existente.

templates/eliminar_tarea.html contiene la plantilla para la confirmación de eliminación de una tarea. Aquí está el código con comentarios:

```
{% extends "base.html" %}

{% block title %}Eliminar Tarea{% endblock %}

{% block content %}
    <h1>Eliminar Tarea</h1>
    <p>¿Estás seguro de que deseas eliminar esta tarea?</p>
    <form method="POST">
        {% csrf_token %}
        <button type="submit" class="btn btn-danger">Eliminar</button>
    </form>
    <a class="btn btn-secondary" href="{% url 'lista_tareas' %}">Cancelar y Volver a la L
{% endblock %}
```

Explicación:

① La plantilla hereda de "base.html" utilizando {% extends %} y establece el título de la página como "Eliminar Tarea" en el bloque {% block title %}.

Esta plantilla se utiliza para solicitar la confirmación de eliminación de una tarea antes de realizar la acción de eliminación.

templates/inicio.html contiene la plantilla para la página de inicio del proyecto. A continuación, se muestra el código con comentarios:

```
{% extends "base.html" %}

{% block title %}Inicio - Mi Sitio Web{% endblock %}

{% block content %}
    <h2>Bienvenido a la página de inicio</h2>
    <p>Esta es la página de inicio de mi sitio web.</p>
{% endblock %}
```

Explicación:

① La plantilla hereda de "base.html" utilizando {% extends %} y establece el título de la página como "Inicio - Mi Sitio Web" en el bloque {% block title %}.

Esta plantilla se utiliza para representar la página de inicio del sitio web, que es una página simple con un mensaje de bienvenida.

templates/lista_tareas.html contiene la plantilla para la lista de tareas en el proyecto. A continuación, se muestra el código con comentarios:

```
{% extends "base.html" %}

{% block title %}Lista de Tareas{% endblock %}
```

```
{% block content %}
    <h1>Lista de Tareas</h1>
    <ul class="list-group">
        {% for tarea in tareas %}
            <li class="list-group-item">{{ tarea.titulo }}</li>
            <p class="list-group-item">{{ tarea.descripcion }}</p>
        {% endfor %}
    </ul>
    <a class="btn btn-primary" href="{% url 'crear_tarea' %}">Crear Tarea</a>
{% endblock %}
```

Explicación:

① La plantilla hereda de “base.html” utilizando {% extends %} y establece el título de la página como “Lista de Tareas” en el bloque {% block title %}.

Esta plantilla se utiliza para mostrar la lista de tareas en el sitio web, donde cada tarea se muestra en forma de lista junto con un botón para crear una nueva tarea.

Para agregar la aplicación “tarea” al archivo settings.py del proyecto Django, simplemente debes incluir ‘tarea’ en la lista INSTALLED_APPS. A continuación, te muestro la sección actualizada de INSTALLED_APPS en settings.py:

```
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'tarea',  # Agrega 'tarea' aquí para incluir la aplicación en tu proyecto
]
```

Con este cambio, la aplicación “tarea” se incorporará al proyecto y estará disponible para su uso en el mismo. Asegúrate de guardar el archivo después de hacer esta modificación en settings.py.

Para configurar las URL del proyecto Django y vincularlas con las URL de la aplicación “tarea”, debes modificar el archivo urls.py del proyecto. A continuación, te muestro cómo podría verse el archivo urls.py del proyecto:

```
from django.contrib import admin
from django.urls import path, include  # Importa include desde django.urls

urlpatterns = [
    path('admin/', admin.site.urls),
```

```
    path('', include('tarea.urls')),  # Incluye las URL de la aplicación 'tarea'
]
```

En este código, hemos importado include desde django.urls y luego hemos utilizado include('tarea.urls') para incluir las URL de la aplicación "tarea" en las URL del proyecto. Esto asegura que todas las URL definidas en el archivo urls.py de la aplicación "tarea" estén disponibles bajo la raíz del proyecto.

Asegúrate de guardar los cambios en el archivo urls.py del proyecto después de realizar esta configuración.

38.1.3 Explicación

En este ejemplo, hemos creado un modelo Tarea para representar las tareas en nuestra base de datos. Luego, creamos un formulario TareaForm utilizando formularios en Django. Además, creamos vistas para listar, crear y ver detalles de tareas. Las plantillas HTML correspondientes se utilizarán para renderizar las páginas web.

> Actividad Práctica
>
> 1. Crea una vista y un formulario para actualizar tareas existentes.
> 2. Agrega una función para eliminar tareas.
> 3. Utiliza las plantillas HTML para mostrar la lista de tareas, el formulario de creación y los detalles de la tarea.

Solucion

Resumen:

En esta actividad práctica, se creará una plantilla base llamada "base_notas.html" que contendrá un menú de navegación y un pie de página. Luego, se creará una plantilla específica llamada "lista_notas.html" que heredará de "base_notas.html" y mostrará una lista de notas utilizando Bootstrap.

Código:

① Crear la Plantilla Base "base_notas.html":

En la carpeta de plantillas de tu aplicación ("articulos" en este ejemplo), crea un archivo llamado "base_notas.html" con el siguiente contenido:

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>{% block title %}Título por Defecto{% endblock %}</title>
    <!-- Agregar enlaces a Bootstrap y otros recursos aquí -->
</head>
<body>
    <nav class="navbar navbar-expand-lg navbar-light bg-light">
```

```
        <a class="navbar-brand" href="#">Mi Aplicación de Notas</a>
        <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="
            <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarNav">
            <ul class="navbar-nav">
                <li class="nav-item active">
                    <a class="nav-link" href="#">Inicio <span class="sr-only">(current)</
                </li>
                <li class="nav-item">
                    <a class="nav-link" href="#">Notas</a>
                </li>
                <!-- Agregar más elementos de menú si es necesario -->
            </ul>
        </div>
    </nav>

    <div class="container">
        {% block content %}{% endblock %}
    </div>

    <footer class="footer mt-auto py-3">
        <div class="container">
            <span class="text-muted">© 2023 Mi Aplicación de Notas</span>
        </div>
    </footer>
    <!-- Agregar enlaces a Bootstrap y otros recursos aquí -->
</body>
</html>
```

Crea un archivo llamado “lista_notas.html” en la misma carpeta de plantillas de tu aplicación (“articulos” en este ejemplo). Esta plantilla heredará de “base_notas.html” y mostrará una lista de notas utilizando Bootstrap:

```
{% extends "base_notas.html" %}

{% block title %}Lista de Notas{% endblock %}

{% block content %}
    <h1>Lista de Notas</h1>
    <ul class="list-group">
        {% for nota in notas %}
            <li class="list-group-item">{{ nota.titulo }}</li>
            <p class="list-group-item">{{ nota.contenido }}</p>
        {% endfor %}
    </ul>
{% endblock %}
```

Explicación:

① Hemos creado una plantilla base llamada "base_notas.html" que incluye un menú de navegación en la parte superior y un pie de página en la parte inferior. Esta plantilla utiliza Bootstrap para el estilo del menú y el pie de página.

Con estos pasos, hemos creado una plantilla base reutilizable ("base_notas.html") y una plantilla específica ("lista_notas.html") que hereda de la base y muestra una lista de notas con estilo Bootstrap. Esto facilitará la creación de páginas similares en tu aplicación Django.

38.1.4 ¿Qué aprendimos?

En este proyecto, aprendimos a crear una aplicación web completa utilizando Django para gestionar una lista de tareas (to-do list) con operaciones CRUD (Crear, Leer, Actualizar y Eliminar). Aquí está un resumen de lo que aprendimos:

- **Modelos de Django:** Creamos un modelo llamado Tarea en Django para representar las tareas en nuestra aplicación. Utilizamos campos como CharField y DateTimeField para definir los atributos de las tareas.
- **Formularios en Django:** Utilizamos formularios en Django para crear y manejar formularios HTML de manera sencilla y eficiente. Creamos un formulario llamado TareaForm que se basa en el modelo Tarea.
- **Vistas en Django:** Creamos vistas en Django para manejar diferentes acciones, como listar tareas, crear tareas, ver detalles de tareas, editar tareas y eliminar tareas. Utilizamos funciones de vista basadas en clases y funciones de vista basadas en funciones para lograr estas funcionalidades.
- **Plantillas HTML:** Creamos plantillas HTML utilizando el sistema de herencia de plantillas de Django. Creamos una plantilla base que contiene elementos comunes, como el encabezado y el pie de página, y luego creamos plantillas específicas que heredan de la plantilla base y agregan contenido específico para cada página.
- **Enrutamiento de URL:** Configuramos las URL de la aplicación para que las vistas se llamen correctamente cuando los usuarios acceden a diferentes URL. Usamos el enrutador path para mapear las URL a las vistas correspondientes.
- **Integración de Bootstrap:** Mejoramos la apariencia de nuestra aplicación web utilizando Bootstrap, un marco de diseño de código abierto. Aplicamos estilos predefinidos y componentes de Bootstrap a nuestras plantillas HTML.
- **Administrador de Django:** Utilizamos el administrador de Django para gestionar fácilmente los datos de las tareas a través de una interfaz de administración generada automáticamente.
- **Configuración de Proyecto:** Aprendimos cómo configurar las aplicaciones en el archivo settings.py del proyecto y cómo conectar las URL de la aplicación con las URL del proyecto.

En general, este proyecto nos permitió comprender cómo construir una aplicación web completa con Django, desde la definición de modelos y formularios hasta la creación de vistas y plantillas HTML. También aprendimos a mejorar la interfaz de usuario utilizando Bootstrap y a gestionar los datos de la aplicación a través del administrador de Django. Estas habilidades son fundamentales para el desarrollo de aplicaciones web con Django.

39 Integración de Django Rest Framework en Proyecto de Tareas

En esta sección, aprenderemos cómo integrar Django Rest Framework (DRF) en nuestro proyecto de lista de tareas. DRF es una poderosa herramienta para construir APIs web en Django de una manera sencilla y eficiente.

Django Rest Framework es una biblioteca que se utiliza para construir APIs web en aplicaciones Django. Proporciona una amplia variedad de características y herramientas para facilitar la creación y el consumo de APIs RESTful. Configuración de Django Rest Framework

Para integrar Django Rest Framework en nuestro proyecto, sigamos estos pasos: Paso 1: Instalar Django Rest Framework

Primero, debemos instalar DRF utilizando pip:

```
pip install djangorestframework
```

Paso 2: Agregar DRF a la Configuración del Proyecto

En el archivo settings.py de nuestro proyecto, agreguemos 'rest_framework' a la lista de aplicaciones instaladas:

```
# settings.py

INSTALLED_APPS = [
    # ...
    'rest_framework',
    # ...
]
```

Paso 3: Configurar las Vistas de DRF

A continuación, crearemos vistas basadas en clases utilizando DRF para exponer nuestra lista de tareas a través de una API.

```
# tarea/views.py

from rest_framework import generics
from .models import Tarea
from .serializers import TareaSerializer
```

```
class ListaTareasAPI(generics.ListCreateAPIView):
    queryset = Tarea.objects.all()
    serializer_class = TareaSerializer

class DetalleTareaAPI(generics.RetrieveUpdateDestroyAPIView):
    queryset = Tarea.objects.all()
    serializer_class = TareaSerializer
```

Paso 4: Configurar las Rutas de la API

En el archivo urls.py de nuestra aplicación, definiremos las rutas para las vistas de DRF que hemos creado.

```
# tarea/urls.py

from django.urls import path
from . import views

urlpatterns = [
    # ...
    path('api/tareas/', views.ListaTareasAPI.as_view(), name='lista_tareas_api'),
    path('api/tareas/<int:pk>/', views.DetalleTareaAPI.as_view(), name='detalle_tarea_api
    # ...
]
```

Paso 5: Crear Serializadores

Para convertir nuestros objetos de modelo en datos JSON, crearemos un serializador en la aplicación.

```
# tarea/serializers.py

from rest_framework import serializers
from .models import Tarea

class TareaSerializer(serializers.ModelSerializer):
    class Meta:
        model = Tarea
        fields = ['id', 'titulo', 'descripcion', 'fecha_creacion']
```

Ejemplo

```
# Ejemplo de acceso a la API utilizando la biblioteca `requests` en Python

import requests

# Listar todas las tareas
response = requests.get('http://localhost:8000/api/tareas/')
```

```
tareas = response.json()
print(tareas)

# Crear una nueva tarea
nueva_tarea = {'titulo': 'Nueva Tarea', 'descripcion': 'Descripción de la nueva tarea'}
response = requests.post('http://localhost:8000/api/tareas/', data=nueva_tarea)
print(response.status_code)

# Obtener detalles de una tarea específica (reemplace 1 con el ID de la tarea)
response = requests.get('http://localhost:8000/api/tareas/1/')
tarea = response.json()
print(tarea)

# Actualizar una tarea (reemplace 1 con el ID de la tarea)
datos_actualizados = {'titulo': 'Tarea Actualizada', 'descripcion': 'Descripción actualiz
response = requests.put('http://localhost:8000/api/tareas/1/', data=datos_actualizados)
print(response.status_code)

# Eliminar una tarea (reemplace 1 con el ID de la tarea)
response = requests.delete('http://localhost:8000/api/tareas/1/')
print(response.status_code)
```

:::{.callout-tip} Actividad Práctica

① Siga los pasos de configuración para integrar Django Rest Framework en su proyecto de lista de tareas.

:::

40 ¿Qué aprendimos?

Aprendimos cómo integrar Django Rest Framework en nuestro proyecto Django para crear una API que expone las operaciones CRUD en nuestras tareas. Configuramos vistas, rutas y serializadores para habilitar la API y utilizamos una biblioteca como requests para interactuar con la API y realizar operaciones CRUD en nuestras tareas. Con esta integración, nuestra aplicación de lista de tareas ahora puede ser consumida por aplicaciones cliente, como aplicaciones móviles o SPA (aplicaciones de una sola página), que necesitan acceder a datos a través de una API RESTful.

41 Documentación de la API con DRF-YASG

Ahora aprenderemos a crear documentación para nuestra API utilizando la biblioteca DRF-YASG (Yet Another Swagger Generator).

41.1 Instalación de DRF-YASG

El método preferido de instalación de DRF-YASG es directamente desde PyPI. Asegúrate de que tu entorno virtual esté activo antes de ejecutar los comandos de instalación:

```
pip install -U drf-yasg
```

41.2 Configuración de DRF-YASG

Paso 1: Agregar DRF-YASG a la Configuración

En el archivo settings.py de tu proyecto, agrega 'drf_yasg' a la lista de aplicaciones instaladas:

```
# settings.py

INSTALLED_APPS = [
    # ...
    'drf_yasg',
    # ...
]
```

Paso 2: Configurar las URL de Documentación

En el archivo urls.py de tu proyecto, configura las URLs de documentación de DRF-YASG:

```
# urls.py

from django.urls import path, re_path
from rest_framework import permissions
from drf_yasg.views import get_schema_view
from drf_yasg import openapi
```

```
schema_view = get_schema_view(
    openapi.Info(
        title="Tareas API",
        default_version='v1',
        description="API para gestionar tareas",
        terms_of_service="https://www.google.com/policies/terms/",
        contact=openapi.Contact(email="contact@tareas.local"),
        license=openapi.License(name="BSD License"),
    ),
    public=True,
    permission_classes=(permissions.AllowAny,),
)

urlpatterns = [
    # ...
    path('swagger<format>/', schema_view.without_ui(cache_timeout=0), name='schema-json')
    path('swagger/', schema_view.with_ui('swagger', cache_timeout=0), name='schema-swagge
    path('redoc/', schema_view.with_ui('redoc', cache_timeout=0), name='schema-redoc'),
    # ...
]
```

41.3 Uso de DRF-YASG

Una vez que hayas configurado las URLs de documentación, podrás acceder a la documentación de tu API a través de las siguientes rutas:

- **/swagger/:** Interfaz Swagger UI para interactuar y probar la API.
- **/redoc/:** Interfaz ReDoc para una experiencia de usuario mejorada.

41.4 ¿Qué aprendimos?

Aprendimos a configurar y utilizar la biblioteca DRF-YASG para generar documentación para nuestra API Django Rest Framework. Esta documentación proporciona una forma fácil de explorar y probar las API de nuestro proyecto, lo que facilita su uso y comprensión tanto para nosotros como para otros desarrolladores que trabajan en el proyecto. Con DRF-YASG, nuestra API se vuelve más accesible y autodocumentada.

Part IX

Unidad 9: Introducción a Bases de Datos

42 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.

42.1 Instalación de MySQL:

Descargar e instalar MySQL desde el sitio oficial.

Configurar contraseña para el usuario 'root'.

42.2 Instalación de PostgreSQL:

Descargar e instalar PostgreSQL desde el sitio oficial.

Configurar contraseña para el usuario 'postgres'.

42.3 Instalación de MongoDB:

Descargar e instalar MongoDB desde el sitio oficial.

Configurar directorio de datos y logs.

42.4 Conexión a la Base de Datos:

42.4.1 MySQL y PostgreSQL:

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

42.4.2 MongoDB:

Usar la biblioteca pymongo para conectarse y realizar operaciones.

42.5 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"
)
```

42.6 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.

42.7 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.

43 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.

43.1 Operaciones en MySQL:

43.1.1 Crear una tabla:

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

43.1.2 Insertar registros:

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

43.1.3 Consultar registros:

```
SELECT * FROM nombre;
```

43.1.4 Actualizar registros:

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

43.1.5 Eliminar registros:

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

43.1.6 Eliminar tabla:

```
DROP TABLE nombre;
```

43.2 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'.

43.3 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.

44 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.

44.1 Operaciones en PostgreSQL:

44.1.1 Crear una tabla:

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

44.1.2 Insertar registros:

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

44.1.3 Consultar registros:

```
SELECT * FROM nombre;
```

44.1.4 Actualizar registros:

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

44.1.5 Eliminar registros:

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

44.1.6 Eliminar tabla:

```
DROP TABLE nombre;
```

44.2 Ejemplo - Creación de una Tabla en PostgreSQL:

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

! Actividad Práctica

44.3 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'.

44.4 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.

45 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.

45.1 Operaciones en MongoDB:

45.1.1 Insertar documentos:

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

45.1.2 Consultar documentos:

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

45.1.3 Actualizar documentos:

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

45.1.4 Eliminar documentos:

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

45.2 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.

45.3 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.

Part X

Unidad 10: Operaciones Básicas en Bases de Datos

46 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.

46.1 Conceptos Clave:

46.1.1 Ciencia de Datos:

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

46.1.2 Uso de Python en Data Science:

Bibliotecas como NumPy, Pandas y Matplotlib.

46.1.3 Ejemplos de Aplicación:

Análisis de datos,

Visualización,

Aprendizaje Automático,

etc.

46.2 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.

46.3 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.

47 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.

47.1 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.

47.2 Instalación de Django:

47.2.1 Instalar Django utilizando pip:

```
pip install django
```

47.2.2 Verificar la instalación:

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

47.3 Creación de una Aplicación Web Básica:

47.3.1 Crear un nuevo proyecto:

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

47.3.2 Crear una nueva aplicación dentro del proyecto:

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

47.4 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.

47.5 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.

48 Introducción a Django Framework y Django Rest Framework

48.0.1 ¿Qué es Django?

Django es un framework web de alto nivel desarrollado en Python que se utiliza para crear aplicaciones web robustas y escalables.

Algunas de las características que hacen que Django sea una opción popular para el desarrollo web son:

- **Productividad:** Django proporciona un conjunto de herramientas y características incorporadas que permiten a los desarrolladores crear aplicaciones web de manera más rápida y eficiente.
- **Seguridad:** Django incluye características de seguridad integradas, como la protección contra ataques de inyección SQL y protección contra ataques CSRF (Cross-Site Request Forgery).
- **ORM (Mapeo Objeto-Relacional):** Django incluye su propio ORM que permite interactuar con la base de datos utilizando objetos Python en lugar de escribir consultas SQL directamente.
- **Administración de Contenido:** Django proporciona una interfaz de administración fácil de usar que permite a los administradores del sitio gestionar contenido y datos sin necesidad de conocimientos técnicos.
- **Escalabilidad:** Django está diseñado para ser escalable y manejar aplicaciones web de cualquier tamaño.

48.0.2 Arquitectura de Django

Django sigue una arquitectura de diseño llamada MTV (Modelo, Plantilla, Vista), que es una variación del patrón MVC (Modelo-Vista-Controlador). Los componentes clave de MTV son:

- **Modelo:** Representa la estructura de la base de datos y define cómo se almacenan y recuperan los datos.
- **Plantilla:** Define la presentación de la aplicación web y cómo se muestra la información al usuario.
- **Vista:** Controla la lógica de negocio y maneja las solicitudes entrantes del usuario.

Django utiliza URLconf (Configuración de URL) para asignar las URL a las vistas correspondientes. Esto permite que las vistas se activen cuando un usuario accede a una URL específica en la aplicación. Componentes Clave de Django

- **Modelos:** Los modelos en Django definen la estructura de la base de datos. Cada modelo se traduce en una tabla en la base de datos y define los campos y relaciones entre los datos.
- **Vistas:** Las vistas en Django son responsables de procesar las solicitudes entrantes y devolver respuestas. Pueden acceder a los datos del modelo y renderizar plantillas para mostrar información al usuario.
- **Plantillas:** Las plantillas son archivos HTML que definen la presentación de las páginas web en Django. Pueden incluir etiquetas y filtros para mostrar dinámicamente datos desde las vistas.

48.0.3 Modelos en Django

Los modelos en Django definen la estructura de la base de datos y cómo se almacenan los datos. Cada modelo se define como una clase de Python que hereda de **models.Model**. Dentro de la clase, se definen los campos que representan las columnas de la tabla de la base de datos.

```
from django.db import models

class Producto(models.Model):
    nombre = models.CharField(max_length=100)
    precio = models.DecimalField(max_digits=5, decimal_places=2)
    descripcion = models.TextField()
```

En este ejemplo, hemos creado un modelo llamado **Producto** con tres campos: **nombre**, **precio** y **descripcion**.

48.0.4 Vistas en Django

Las vistas en Django son funciones de Python que procesan las solicitudes entrantes y devuelven respuestas. Utilizan los modelos para acceder a los datos y pueden renderizar plantillas para mostrar información.

```
from django.shortcuts import render
from .models import Producto

def lista_productos(request):
    productos = Producto.objects.all()
    return render(request, 'lista_productos.html', {'productos': productos})
```

En esta vista, recuperamos todos los productos de la base de datos y los pasamos a una plantilla llamada lista_productos.html para su representación. Plantillas en Django

Las plantillas en Django son archivos HTML que definen cómo se presenta la información. Utilizan **etiquetas** y **filtros** para acceder a datos desde las vistas y mostrarlos en la página.

```
<!DOCTYPE html>
<html>
<head>
    <title>Lista de Productos</title>
</head>
<body>
    <h1>Lista de Productos</h1>
    <ul>
        {% for producto in productos %}
        <li>{{ producto.nombre }} - ${{ producto.precio }}</li>
        {% endfor %}
    </ul>
</body>
</html>
```

En esta plantilla, utilizamos la etiqueta **{% for producto in productos %}** para iterar sobre la lista de productos y mostrar sus nombres y precios. Creación de una Aplicación en Django

48.0.5 Creación de una Aplicación Django

Django permite dividir una aplicación web en múltiples aplicaciones más pequeñas, cada una con su propio conjunto de **modelos**, **vistas** y **plantillas**. Para crear una aplicación en Django, puedes utilizar el siguiente comando:

```
python manage.py startapp mi_aplicacion .
```

Esto creará una estructura de carpetas y archivos para tu nueva aplicación. Luego, puedes registrar la aplicación en la configuración del proyecto.

48.0.6 Definición de Rutas y Vistas

Las rutas en Django se definen en el archivo **urls.py** de la aplicación. Puedes asignar URL a funciones de vista específicas.

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

urlpatterns = [
    path('productos/', views.lista_productos, name='lista_productos'),
```

```
    # Otras rutas...
]
```

En este ejemplo, hemos asignado la URL **/productos/** a la vista **lista_productos** que creamos anteriormente.

48.1 Administración y Base de Datos en Django

48.1.1 Interfaz de Administración de Django

Django proporciona una potente interfaz de administración que facilita la gestión de datos y contenido. Esta interfaz se genera automáticamente a partir de los modelos definidos en la aplicación.

> **Tip**
>
> Para poder utilizar nustros modelos en la administración debemos registrarlos en el archivo **admin.py** de la aplicación.
>
> ```
> from django.contrib import admin
> from .models import Producto
>
> admin.site.register(Producto)
> ```

Los administradores del sitio pueden utilizar la interfaz para:

- Agregar, editar y eliminar registros de la base de datos.
- Gestionar usuarios y permisos.
- Realizar otras tareas administrativas.

48.1.2 ORM de Django

El ORM (Mapeo Objeto-Relacional) de Django permite interactuar con la base de datos utilizando objetos Python en lugar de escribir consultas SQL directamente. Esto simplifica la gestión de datos y hace que el código sea más legible.

Por ejemplo, para recuperar todos los productos de la base de datos, puedes hacer lo siguiente:

```
productos = Producto.objects.all()
```

Esto devuelve una lista de objetos Producto que representan los registros de la tabla de productos en la base de datos.

48.2 Ejercicios Prácticos

48.2.1 Ejercicio 1: Creación de un Modelo en Django

① Crear un modelo en Django para representar una entidad de su elección (por ejemplo, libros, películas, tareas, etc.).

```
# models.py

from django.db import models

class Libro(models.Model):
    titulo = models.CharField(max_length=100)
    autor = models.CharField(max_length=50)
    año_publicacion = models.IntegerField()

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

Después de definir el modelo, debes crear y aplicar las migraciones utilizando los siguientes comandos:

```
python manage.py makemigrations
python manage.py migrate
```

48.2.2 Ejercicio 2: Creación de una Vista y Plantilla en Django

① Crear una vista en Django que recupere datos de su modelo y los pase a una plantilla.

```
# views.py

from django.shortcuts import render
from .models import Libro

def lista_libros(request):
    libros = Libro.objects.all()
    return render(request, 'myapp/lista_libros.html', {'libros': libros})
```

```
<!-- lista_libros.html -->

<!DOCTYPE html>
<html>
<head>
    <title>Lista de Libros</title>
</head>
<body>
```

```
    <h1>Lista de Libros</h1>
    <ul>
        {% for libro in libros %}
        <li>{{ libro.titulo }} - {{ libro.autor }} ({{ libro.año_publicacion }})</li>
        {% empty %}
        <li>No hay libros disponibles.</li>
        {% endfor %}
    </ul>
</body>
</html>
```

```
# urls.py

from django.urls import path
from . import views

urlpatterns = [
    path('libros/', views.lista_libros, name='lista_libros'),
]
```

48.2.3 Ejercicio 3: Uso de la Interfaz de Administración de Django

- Registrar el modelo creado en el Ejercicio 1 en la interfaz de administración de Django.
- Utilizar la interfaz de administración para agregar al menos dos registros de ejemplo.

Primero, asegúrate de haber registrado el modelo Libro en el archivo admin.py de tu aplicación:

```
# admin.py

from django.contrib import admin
from .models import Libro

admin.site.register(Libro)
```

Luego, puedes acceder a la interfaz de administración de Django en http://tu-sitio/admin/ para agregar registros de ejemplo.

48.2.4 Ejercicio 4: Uso del ORM de Django

- Escribir código Python para recuperar datos de su modelo utilizando el ORM de Django.
- Mostrar los datos recuperados en la consola o en una página web.

Puedes utilizar el siguiente código para recuperar y mostrar datos de libros utilizando el ORM de Django en una vista:

```
# views.py

from django.shortcuts import render
from .models import Libro

def lista_libros(request):
    libros = Libro.objects.all()
    return render(request, 'lista_libros.html', {'libros': libros})
```

Este código recupera todos los registros de la base de datos y los pasa a la plantilla para su representación.

Estos ejercicios prácticos ayudarán a los estudiantes a aplicar los conceptos de modelos, vistas y plantillas en Django, así como a familiarizarse con la interfaz de administración y el ORM.

48.3 API de libros utilizando Django Rest Framework (DRF).

Para configurar y desarrollar este proyecto es necesario que realicemos los siguientes pasos:

48.3.1 Paso 1: Configuración Inicial

Asegúrate de tener Django Rest Framework instalado en tu entorno virtual. Puedes instalarlo usando pip:

```
pip install djangorestframework
```

Crea un nuevo proyecto de Django si aún no lo has hecho:

```
django-admin startproject proyecto_api
```

Luego, crea una nueva aplicación dentro del proyecto:

```
cd proyecto_api
python manage.py startapp api
```

Agrega **rest_framework** y **api** a la lista de aplicaciones en settings.py:

```
INSTALLED_APPS = [
        # ...
        'rest_framework',
        'api',
    ]
```

48.4 Paso 2: Modelado de Datos

Define el modelo de datos para los libros en el archivo api/models.py. Aquí tienes un ejemplo simple:

```
from django.db import models

class Libro(models.Model):
    titulo = models.CharField(max_length=100)
    autor = models.CharField(max_length=100)
    año_publicacion = models.PositiveIntegerField()

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

Luego, crea y aplica las migraciones para este modelo:

```
python manage.py makemigrations
python manage.py migrate
```

48.4.1 Paso 3: Serialización

Crea un serializador en el archivo api/serializers.py para convertir los objetos de modelo en datos JSON:

```
from rest_framework import serializers
from .models import Libro

class LibroSerializer(serializers.ModelSerializer):
    class Meta:
        model = Libro
        fields = ['id', 'titulo', 'autor', 'año_publicacion']
```

48.4.2 Paso 4: Vistas y Rutas

En api/views.py, define las vistas basadas en clases utilizando Django Rest Framework:

```
from rest_framework import generics
from .models import Libro
from .serializers import LibroSerializer

class ListaLibros(generics.ListCreateAPIView):
    queryset = Libro.objects.all()
    serializer_class = LibroSerializer

class DetalleLibro(generics.RetrieveUpdateDestroyAPIView):
```

```
    queryset = Libro.objects.all()
    serializer_class = LibroSerializer
```

Luego, configura las rutas en api/urls.py:

```
from django.urls import path
from .views import ListaLibros, DetalleLibro

urlpatterns = [
    path('libros/', ListaLibros.as_view(), name='lista_libros'),
    path('libros/<int:pk>/', DetalleLibro.as_view(), name='detalle_libro'),
]
```

48.4.3 Paso 5: Configuración de URLs Principales

En el archivo proyecto_api/urls.py, incluye las rutas de la aplicación de API:

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/', include('api.urls')),
]
```

48.4.4 Paso 6: Ejecutar el Servidor

Inicia el servidor de desarrollo:

```
python manage.py runserver
```

48.4.5 Paso 7: Prueba de la API

Ahora, puedes acceder a la API en las siguientes URL:

- Lista de libros: http://localhost:8000/api/libros/
- Detalle de libro: http://localhost:8000/api/libros/id/

> **Tip**
>
> En el **Detalle del Libro**, es necesario cambiar el **id** por el número de identificación del libro que se desea consultar.

Puedes utilizar herramientas como **Postman**, **Thunder Cliente** o **Rappid API Client** o simplemente un navegador web para probar las rutas y realizar operaciones CRUD en la API de libros.

Este proyecto proporciona una base sólida para desarrollar una API de libros con Django Rest Framework. Puedes personalizarlo según tus necesidades y agregar autenticación u otras características según sea necesario.

48.5 Conclusiones

En esta unidad, hemos explorado Django y Django Rest Framework (DRF), dos poderosas herramientas para el desarrollo web con Python. Aquí están algunas conclusiones clave:

- Django es un framework web de alto nivel que sigue el principio "baterías incluidas". Proporciona una estructura sólida y conveniente para desarrollar aplicaciones web, incluyendo la administración de bases de datos, la autenticación de usuarios y un sistema de rutas robusto.
- Django Rest Framework (DRF) es una extensión de Django que simplifica la creación de API REST. Proporciona clases y herramientas que permiten definir fácilmente puntos finales de API y serializar datos de manera eficiente.
- Algunas ventajas de utilizar Django incluyen su gran comunidad, documentación extensa y la capacidad de construir aplicaciones web rápidamente.
- DRF es ideal para crear API RESTful de manera rápida y eficiente. Proporciona una capa de serialización que facilita la conversión de objetos de modelo en datos JSON.

48.6 Recomendaciones

- **Aprender la Documentación:** Tanto Django como DRF tienen documentación detallada. Aprovecha estos recursos para comprender las funcionalidades y las mejores prácticas.
- **Practicar:** La práctica es clave para dominar estas herramientas. Crea proyectos pequeños para aplicar lo que has aprendido.
- **Comunidad:** La comunidad de Django es activa y solidaria. Únete a foros y grupos de discusión para obtener ayuda cuando la necesites.
- **Seguridad:** Django tiene características de seguridad incorporadas, pero debes estar atento a las mejores prácticas de seguridad web al desarrollar aplicaciones.

49 Introducción a Flask Framework

49.1 ¿Qué es Flask?

Flask es un microframework web de Python que permite crear aplicaciones web de manera rápida y sencilla. A diferencia de los frameworks más grandes, como **Django**, **Flask** se centra en proporcionar solo lo esencial para crear aplicaciones web, dejando a los desarrolladores la **libertad** de elegir las **herramientas** y **bibliotecas** adicionales que deseen.

49.1.1 Ventajas de Usar Flask

- **Simplicidad:** Flask se destaca por su simplicidad y facilidad de uso. Su estructura minimalista hace que sea fácil de aprender y comprender.
- **Flexibilidad:** Aunque es minimalista, Flask es altamente personalizable. Los desarrolladores pueden elegir las extensiones y bibliotecas que mejor se adapten a sus necesidades.
- **Comunidad Activa:** Flask cuenta con una comunidad activa de desarrolladores y una amplia documentación en línea.
- **Ideal para Proyectos Pequeños y Medianos:** Flask es perfecto para proyectos pequeños y medianos, prototipado rápido y aplicaciones que no requieren una gran cantidad de funcionalidades incorporadas.

49.2 Ecosistema de Extensiones de Flask

Flask tiene un ecosistema de extensiones que permiten agregar funcionalidades específicas a las aplicaciones web. Estas extensiones abarcan áreas como **autenticación de usuarios**, **bases de datos**, **manejo de formularios** y más. Algunas extensiones populares son **Flask-SQLAlchemy** para trabajar con bases de datos y **Flask-WTF** para manejar formularios.

49.3 Instalación de Flask

49.3.1 Cómo Instalar Flask Usando pip

Para comenzar a trabajar con Flask, primero debes instalarlo en tu **entorno de desarrollo** (virtualenv o docker). Puedes hacerlo utilizando la herramienta **pip**, que es el administrador de paquetes de Python.

Ejecuta el siguiente comando en tu terminal:

```
pip install flask
```

49.3.2 Creación de un Entorno Virtual para Proyectos Flask.

> **Tip**
>
> Se recomienda crear un entorno virtual para cada proyecto Flask.

> **Note**
>
> Un entorno virtual es un espacio aislado donde puedes instalar las dependencias específicas de tu proyecto sin interferir con otras aplicaciones.

Para crear un entorno virtual, sigue estos pasos:

① Abre una terminal y navega hasta la carpeta de tu proyecto.

```
python3 -m venv venv
```

Activa el entorno virtual:

En Windows:

```
venv\Scripts\activate
```

En Linux o macOS:

```
source venv/bin/activate
```

Ahora estás listo para trabajar en tu proyecto Flask en un entorno aislado.

49.4 Tu Primera Aplicación en Flask

49.4.1 Creación de una Aplicación Web Simple

En Flask, una aplicación web se crea mediante una instancia de la clase Flask. Aquí hay un ejemplo simple de cómo crear una aplicación web:

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello_world():
    return '¡Hola, mundo!'

if __name__ == '__main__':
    app.run()
```

- Importamos la clase Flask de la biblioteca Flask.
- Creamos una instancia de Flask y la asignamos a la variable app.
- Usamos el decorador **(app.route?)(‘/’)** para definir una ruta en nuestra aplicación. En este caso, la ruta raíz **‘/’** se maneja con la función **hello_world()**.
- Cuando se ejecuta la aplicación **(if name == ‘main’:)**, llamamos a **app.run()** para iniciar el servidor web.

49.5 Definición de Rutas y Vistas en Flask

En Flask, las rutas se definen utilizando decoradores como **(app.route?)(‘/’)**. Cada ruta está asociada a una **función** (vista) que se ejecuta cuando se accede a esa **ruta en el navegador**. **Las vistas** pueden devolver **contenido HTML** o cualquier otro tipo de respuesta web.

49.6 Plantillas HTML en Flask

Flask permite renderizar plantillas HTML para generar páginas web dinámicas. Para esto, generalmente se usa una biblioteca llamada **Jinja2**. Las plantillas pueden incluir variables y lógica de presentación.

49.7 Manejo de Formularios

49.7.1 Creación y Procesamiento de Formularios en Flask

Las aplicaciones web suelen requerir la entrada de datos de los usuarios a través de formularios. Flask facilita la creación y el procesamiento de formularios.

Para crear un formulario en Flask, generalmente se define una clase que hereda de **flask_wtf.FlaskForm**.

A través de esta clase, puedes definir los campos del formulario y las validaciones necesarias.

El siguiente es un ejemplo simple de cómo definir un formulario de inicio de sesión:

```
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField
from wtforms.validators import DataRequired

class LoginForm(FlaskForm):
    username = StringField('Nombre de Usuario', validators=[DataRequired()])
    password = PasswordField('Contraseña', validators=[DataRequired()])
```

En este ejemplo, hemos creado un formulario **LoginForm** con dos campos: **username** y **password**. También hemos especificado que ambos campos son obligatorios.

49.7.2 Validación de Datos del Formulario.

Flask-WTF, una extensión de Flask, facilita la validación de los datos del formulario. En el ejemplo anterior, usamos el validador **DataRequired()** para asegurarnos de que los campos no estén vacíos.

49.8 Ejercicios Prácticos

Ejercicio 1: Creación de una Aplicación Web Básica en Flask

Crea una aplicación web simple en Flask que muestre un mensaje de bienvenida en la ruta raíz ("/"). Puedes personalizar el mensaje de bienvenida.

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def welcome():
    return '¡Bienvenido a mi aplicación web en Flask!'

if __name__ == '__main__':
    app.run()
```

Este código crea una aplicación web en Flask que muestra un mensaje de bienvenida en la ruta raíz ("/").

Ejercicio 2: Implementación de un Formulario en Flask

Extiende la aplicación web anterior para incluir un formulario de contacto. Crea un formulario que solicite el nombre y el correo electrónico del usuario. Cuando el usuario envíe el formulario, muestra un mensaje de agradecimiento junto con los datos ingresados.

```python
from flask import Flask, render_template, request

app = Flask(__name__)

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/submit', methods=['POST'])
def submit():
    if request.method == 'POST':
        name = request.form['name']
        email = request.form['email']
        return f'Thank you, {name}! Your email ({email}) has been received.'

if __name__ == '__main__':
    app.run()
```

En este ejercicio, hemos creado un formulario de contacto que solicita el nombre y el correo electrónico del usuario. Cuando el usuario envía el formulario, se muestra un mensaje de agradecimiento junto con los datos ingresados.

Ejercicio 3: Uso de una Plantilla HTML en una Aplicación Flask

Crea una plantilla HTML que contenga un diseño básico para tu sitio web. Luego, utiliza Flask para renderizar esta plantilla y mostrarla en una ruta específica de tu aplicación.

Primero, crea una plantilla HTML llamada **template.html** en una carpeta llamada templates en el directorio de tu proyecto. El contenido de template.html podría ser el siguiente:

```html
<!DOCTYPE html>
<html>
<head>
    <title>Mi Sitio Web</title>
</head>
<body>
    <h1>Bienvenido a mi sitio web</h1>
    <p>Este es un sitio web de ejemplo creado con Flask.</p>
</body>
</html>
```

Luego, modifica tu aplicación Flask para renderizar esta plantilla:

```python
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/')
```

```
def index():
    return render_template('template.html')

if __name__ == '__main__':
    app.run()
```

Este código renderizará la plantilla HTML en la ruta raíz (“/”) de tu aplicación.

Puedes personalizar estas soluciones según tus necesidades y preferencias de diseño. Además, asegúrate de tener la estructura de carpetas adecuada con la carpeta templates para almacenar tus plantillas HTML.

> Tip
>
> **Actividad Práctica:** Construcción de un Blog Simple en Flask.
> En esta actividad práctica, construirás un blog básico utilizando Flask. El blog deberá permitir a los usuarios ver publicaciones, agregar nuevas publicaciones y editar publicaciones existentes. Aplicarás los conceptos aprendidos en esta unidad para desarrollar la aplicación.

49.9 Conclusiones

Hemos explorado Flask, un microframework web de Python que destaca por su simplicidad y flexibilidad. Flask proporciona una base sólida para desarrollar aplicaciones web desde cero y permite a los desarrolladores tomar decisiones sobre las herramientas y extensiones que desean utilizar. Algunas ventajas clave de Flask son su facilidad de aprendizaje, su comunidad activa y su capacidad para adaptarse a una variedad de proyectos.

49.10 Recomendaciones para Trabajar con Flask

Aprender Jinja2: Flask se combina frecuentemente con Jinja2, un motor de plantillas. Dominar Jinja2 te permitirá crear páginas web dinámicas y flexibles.

Explorar Extensiones: Flask tiene una amplia gama de extensiones disponibles. Investiga las extensiones que pueden simplificar tareas comunes, como el manejo de bases de datos, la autenticación de usuarios y la validación de formularios.

Estructura del Proyecto: A medida que tus proyectos con Flask crezcan, considera organizar tu código siguiendo una estructura de proyecto adecuada. Puedes separar las rutas, las vistas y las plantillas en directorios diferentes para mantener tu código limpio y organizado.

Documentación y Comunidad: Flask cuenta con una documentación detallada y una comunidad activa. Aprovecha estos recursos para aprender más y obtener ayuda cuando la necesites.

Part XI

Unidad 11: ¿Cómo me amplío con Python?

50 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.

50.1 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.

50.2 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
```

50.3 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.

50.4 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.

Part XII

Ejercicios

51 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"
```

52 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).

53 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
```

54 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.

55 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.

56 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)
```

57 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.

58 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
```

59 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.

60 Ejercicio 10:

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

Respuesta:

La sintaxis correcta es:

```
import math
```

61 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)
```

62 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.

63 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")
```

64 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)
```

65 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`.

66 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)
```

67 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
```

68 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.

69 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;
```

70 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})
```

71 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!")
```

71.1 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.

71.2 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.

71.3 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.")
```

71.4 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.

71.5 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.

71.6 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.

71.7 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.

71.8 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.

71.9 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.

Mozilla, Developers. 2023. “Métodos de petición HTTP - HTTP | MDN.” July 24, 2023. https://developer.mozilla.org/es/docs/Web/HTTP/Methods.