

Python 2023

Diego Saavedra

Sep 5, 2023

Table of contents

1 Bienvenida	13
1.1 ¿Qué es este Curso?	13
1.2 ¿A quién está dirigido?	13
1.3 ¿Cómo contribuir?	14
I Unidad 1: Introducción a la Programación	15
2 Introducción general a la Programación	16
2.1 Conceptos Clave	16
2.1.1 Instrucciones	16
2.1.2 Lenguajes de Programación.	16
2.1.3 Algoritmos	17
2.1.4 Depuración	17
2.2 Ejemplo:	18
2.3 Explicación	18
2.4 Explicación de la Actividad	18
II Unidad 2: Instalación de Python y más herramientas	19
3 Instalación de Python	20
3.1 Conceptos Clave	20
3.1.1 Python	20
3.1.2 Interprete	20
3.1.3 IDE	20
3.2 Ejemplo	20
3.3 Explicación	21
3.4 Explicación de la Actividad	21
4 Uso del REPL, PEP 8 y el Zen de Python	23
4.0.1 El REPL (Read-Eval-Print Loop).	23
4.0.2 Ejemplos de Interacción con el REPL	23
4.0.3 PEP 8: Guía de Estilo de Python.	24
4.0.4 Introducción al Zen de Python (PEP 20).	25
4.0.5 Ejercicios Prácticos	25
4.0.6 Explicación	26

III Unidad 3: Introducción a Python	28
5 Distintas formas de trabajar con Python.	29
5.1 Conceptos Clave:	29
5.1.1 Intérprete Interactivo:	29
5.1.2 Scripts de Python:	29
5.1.3 Ambientes Virtuales	29
5.1.4 Ejemplo	31
5.2 Explicación	31
5.3 Explicación de la Actividad	31
6 Las bases de Python.	32
6.1 Conceptos Clave	33
6.1.1 Variables	33
6.1.2 Tipos de Datos	33
6.1.3 Operadores	36
6.1.4 Ejemplo	36
6.2 Explicación:	36
6.3 Explicación de la Actividad	37
7 Identación.	38
7.1 Conceptos Clave	39
7.1.1 Identación	39
7.1.2 Ejemplo	39
7.2 Explicación	39
7.3 Explicación de la Actividad.	39
8 Comentarios	40
8.1 Conceptos Clave:	40
8.1.1 Comentarios	40
8.1.2 Comentarios de una línea	40
8.1.3 Comentarios de múltiples líneas	40
8.2 Ejemplo	40
8.3 Explicación:	41
8.4 Explicación de la Actividad	41
9 Variables	42
9.1 Conceptos Clave:	42
9.1.1 Variables	42
9.1.2 Declaración de Variables.	42
9.2 Ejemplo	42
9.3 Explicación:	42
9.4 Explicación de la Actividad	43
10 Múltiples Variables	44
10.1 Conceptos Clave:	44
10.1.1 Asignación Múltiple	44
10.1.2 Desempaquetado de Valores	44
10.1.3 Intercambio de Valores	44

10.2 Ejemplo	44
10.3 Explicación:	44
10.4 Explicación de la Actividad	45
11 Concatenación	46
11.1 Conceptos Clave:	46
11.1.1 Concatenación:	46
11.1.2 Operador +	46
11.1.3 Conversión a Cadena	46
11.2 Ejemplo	46
11.3 Explicación:	46
11.4 Explicación de la Actividad	47
IV Unidad 4: Tipos de Datos	48
12 String y Números	49
12.1 Conceptos Clave:	49
12.1.1 String	49
12.1.2 Números Enteros (int)	49
12.1.3 Números de Punto Flotante (float)	51
12.2 Ejemplo	51
12.3 Explicación	51
12.4 Explicación de la Actividad:	52
13 Listas	53
13.1 Conceptos Clave:	53
13.1.1 Listas	53
13.1.2 Índices	53
13.1.3 Acceso a Elementos.	55
13.2 Ejemplo	55
13.3 Explicación	55
13.4 Explicación de la Actividad:	56
14 Tuplas.	57
14.1 Conceptos Clave:	57
14.1.1 Tuplas	57
14.1.2 Inmutabilidad	57
14.1.3 Acceso a Elementos	59
14.2 Ejemplo	59
14.3 Explicación	59
14.4 Explicación de la Actividad:	59
15 Range	60
15.1 Conceptos Clave:	60
15.1.1 range	60
15.1.2 Parámetros de range	60
15.1.3 Conversión a Listas	61
15.2 Ejemplo	61

15.3 Explicación	61
15.4 Explicación de la Actividad:	61
16 Diccionarios	62
16.1 Conceptos Clave:	63
16.1.1 Diccionarios	63
16.1.2 Claves	63
16.1.3 Valores	63
16.2 Ejemplo	63
16.3 Explicación	64
16.4 Explicación de la Actividad:	64
17 Booleanos.	65
17.1 Conceptos Clave:	65
17.1.1 Booleanos	65
17.2 Ejemplo	66
17.3 Explicación	66
17.4 Explicación de la Actividad:	66
V Unidad 5: Control de Flujo	67
18 Introducción a If	68
18.1 Conceptos Clave:	68
18.1.1 Control de Flujo	68
18.1.2 Estructura if	68
18.1.3 Bloque de Código	68
18.2 Ejemplo:	68
18.3 Explicación:	68
18.4 Explicación de la Actividad:	69
19 If y Condicionales	70
19.1 Conceptos Clave	70
19.1.1 Estructura elif	70
19.1.2 Estructura else	70
19.1.3 Anidación de Estructuras if	70
19.2 Ejemplo:	70
19.3 Explicación:	70
19.4 Explicación de la Actividad:	71
20 If, elif y else	72
20.1 Conceptos Clave:	72
20.1.1 Anidación de Estructuras	72
20.1.2 Jerarquía de Condiciones	72
20.2 Ejemplo	72
20.3 Explicación:	72
20.4 Explicación de la Actividad:	73

21 And y Or	74
21.1 Conceptos Clave:	74
21.1.1 Operador and	74
21.1.2 Operador or	74
21.1.3 Combinación de Condiciones	74
21.2 Ejemplo:	74
21.3 Explicación:	74
21.4 Explicación de la Actividad:	75
22 Introducción a While	76
22.1 Conceptos Clave:	76
22.1.1 Estructura while	76
22.1.2 Condición	76
22.2 Ejemplo:	76
22.3 Explicación:	76
22.4 Explicación de la Actividad:	77
23 While loop	78
23.1 Conceptos Clave:	78
23.1.1 Sentencia break:	78
23.1.2 Bucles Infinitos:	78
23.2 Ejemplo:	78
23.3 Explicación:	78
23.4 Explicación de la Actividad:	79
24 While, break y continue	80
24.1 Conceptos Clave:	80
24.1.1 Sentencia continue	80
24.1.2 Saltar Iteraciones	80
24.2 Ejemplo	80
24.3 Explicación:	80
24.4 Explicación de la Actividad	81
25 For Loop	82
25.1 Conceptos Clave:	82
25.1.1 Bucle for	82
25.1.2 Iteración	82
25.1.3 range() con Bucles for	82
25.2 Ejemplo:	82
25.3 Explicación:	82
25.4 Explicación de la Actividad:	83
VI Unidad 6: Funciones	84
26 Introducción a Funciones	85
26.0.1 Conceptos Clave	85
26.0.2 Definición de Funciones	85
26.0.3 Argumentos	85

26.0.4 Retorno de Valores	85
26.0.5 Explicación de la Actividad	86
27 Recursividad	87
27.0.1 Conceptos Clave	87
27.0.2 Explicación	88
27.0.3 Explicación de la Actividad	88
VII Unidad 7: Objetos, Clases y Herencia	89
28 Programación Orientada a Objetos (POO)	90
28.1 Conceptos Clave en POO	90
28.1.1 Objetos	90
28.1.2 Clases	91
28.1.3 Atributos	91
28.1.4 Métodos	92
28.1.5 Explicación	92
29 Objetos y Clases	93
29.0.1 Atributos de Instancia	93
29.0.2 Métodos de Instancia	94
29.0.3 Explicación	95
30 Métodos	96
30.0.1 Acceso a Atributos	96
30.0.2 Explicación	98
31 Self, Eliminar Propiedades y Objetos.	99
31.0.1 Eliminar Atributos	99
31.0.2 Eliminar Objetos	100
31.0.3 Explicación	102
32 Herencia	103
32.0.1 Herencia	103
33 Polimorfismo	106
33.1 Conceptos Clave:	106
33.1.1 Polimorfismo	106
33.1.2 Explicación:	108
34 Encapsulación	109
34.1 Conceptos Clave:	109
34.1.1 Encapsulación	109
34.1.2 Atributos Privados	109
34.1.3 Métodos Privados	109
34.1.4 Métodos de Acceso (Getters y Setters)	109
34.1.5 Explicación:	110
34.2 ¿Qué apendimos en esta actividad?	111

VIII Unidad 8: Módulos	112
35 Introducción	113
35.1 Conceptos Clave:	113
35.1.1 Módulos	113
35.1.2 Importar Módulos	113
35.1.3 Usar Funciones y Clases	113
35.2 Ejemplo:	113
35.3 Explicación:	114
35.4 Explicación de la Actividad:	114
36 Creando Nuestro Primer Módulo	115
36.1 Pasos para Crear un Módulo:	115
36.2 Ejemplo:	115
36.3 Explicación:	115
36.4 Explicación de la Actividad:	116
37 Renombrando Módulos	117
37.1 Renombrando Módulos al Importar:	117
37.2 Seleccionando Elementos Específicos para Importar:	117
37.3 Ejemplo - Renombrando Módulos:	117
37.4 Ejemplo - Seleccionando Elementos Específicos:	117
37.5 Explicación de la Actividad:	118
38 Seleccionando lo Importado y Pip	119
38.1 Seleccionando Elementos Específicos para Importar:	119
38.1.1 Usando Pip:	119
38.2 Ejemplo - Instalando un Paquete con Pip:	119
38.3 Explicación de la Actividad:	119
IX Unidad 9: Introducción a Bases de Datos	120
39 Introducción a Bases de Datos	121
39.1 Conceptos Clave:	121
39.1.1 Base de Datos	121
39.1.2 Sistemas de Gestión de Bases de Datos (DBMS)	121
39.1.3 Beneficios de las Bases de Datos	121
39.2 Ejemplo:	121
39.3 Explicación:	121
39.4 Explicación de la Actividad:	122
40 Introducción a PostgreSQL	123
40.0.1 Instalación de PostgreSQL:	123
40.0.2 Operaciones Básicas en PostgreSQL:	123
40.0.3 Crear una base de datos:	123
40.0.4 Conectar a una base de datos:	123
40.0.5 Crear una tabla:	123
40.0.6 Insertar registros:	123
40.0.7 Consultar registros:	123

40.0.8 Actualizar registros:	124
40.0.9 Eliminar registros:	124
40.1 Ejemplo - Creación de una Tabla en PostgreSQL:	124
40.1.1 Actividad Práctica:	124
40.2 Explicación de la Actividad:	124
41 Introducción a MongoDB	125
41.1 Instalación de MongoDB:	125
41.2 Operaciones Básicas en MongoDB:	125
41.2.1 Crear una base de datos:	125
41.2.2 Crear una colección (tabla):	125
41.2.3 Insertar documentos (registros):	125
41.2.4 Consultar documentos:	125
41.2.5 Actualizar documentos:	125
41.2.6 Eliminar documentos:	126
41.3 Ejemplo - Creación de una Colección en MongoDB:	126
41.4 Explicación de la Actividad:	126
X Unidad 10: Operaciones Básicas en Bases de Datos	127
42 Introducción e Instalación	128
42.1 Instalación de MySQL:	128
42.2 Instalación de PostgreSQL:	128
42.3 Instalación de MongoDB:	128
42.4 Conexión a la Base de Datos:	128
42.4.1 MySQL y PostgreSQL:	128
42.4.2 MongoDB:	128
42.5 Ejemplo - Conexión a MySQL:	129
42.6 Ejemplo - Conexión a MongoDB:	129
42.7 Explicación de la Actividad:	129
43 Bases de Datos en MySQL	130
43.1 Operaciones en MySQL:	130
43.1.1 Crear una tabla:	130
43.1.2 Insertar registros:	130
43.1.3 Consultar registros:	130
43.1.4 Actualizar registros:	130
43.1.5 Eliminar registros:	130
43.1.6 Eliminar tabla:	130
43.2 Ejemplo - Creación de una Tabla en MySQL:	131
43.3 Explicación de la Actividad:	131
44 Crear y Eliminar Tablas en PostgreSQL	132
44.1 Operaciones en PostgreSQL:	132
44.1.1 Crear una tabla:	132
44.1.2 Insertar registros:	132
44.1.3 Consultar registros:	132
44.1.4 Actualizar registros:	132

44.1.5 Eliminar registros:	132
44.1.6 Eliminar tabla:	132
44.2 Ejemplo - Creación de una Tabla en PostgreSQL:	133
44.3 Conéctate a la base de datos PostgreSQL.	133
44.4 Explicación de la Actividad:	133
45 Operaciones Básicas en MongoDB	134
45.1 Operaciones en MongoDB:	134
45.1.1 Insertar documentos:	134
45.1.2 Consultar documentos:	134
45.1.3 Actualizar documentos:	134
45.1.4 Eliminar documentos:	134
45.2 Ejemplo - Inserción de un Documento en MongoDB:	134
45.3 Explicación de la Actividad:	135
XI Unidad 11: ¿Cómo me amplío con Python?	136
46 Introducción a Data Science	137
46.1 Conceptos Clave:	137
46.1.1 Ciencia de Datos:	137
46.1.2 Uso de Python en Data Science:	137
46.1.3 Ejemplos de Aplicación:	137
46.2 Ejemplo - Uso de Pandas para Análisis de Datos:	137
46.3 Explicación de la Actividad:	138
47 Introducción a Django Framework	139
47.1 Qué es Django:	139
47.2 Instalación de Django:	139
47.2.1 Instalar Django utilizando pip:	139
47.2.2 Verificar la instalación:	139
47.3 Creación de una Aplicación Web Básica:	139
47.3.1 Crear un nuevo proyecto:	139
47.3.2 Crear una nueva aplicación dentro del proyecto:	139
47.4 Ejemplo - Creación de una Página Web con Django:	140
47.5 Explicación de la Actividad:	140
48 Introducción a Django Framework y Django Rest Framework	141
48.0.1 ¿Qué es Django?	141
48.0.2 Arquitectura de Django	141
48.0.3 Modelos en Django	142
48.0.4 Vistas en Django	142
48.0.5 Creación de una Aplicación Django	143
48.0.6 Definición de Rutas y Vistas	143
48.1 Administración y Base de Datos en Django	144
48.1.1 Interfaz de Administración de Django	144
48.1.2 ORM de Django	144
48.2 Ejercicios Prácticos	145
48.2.1 Ejercicio 1: Creación de un Modelo en Django	145

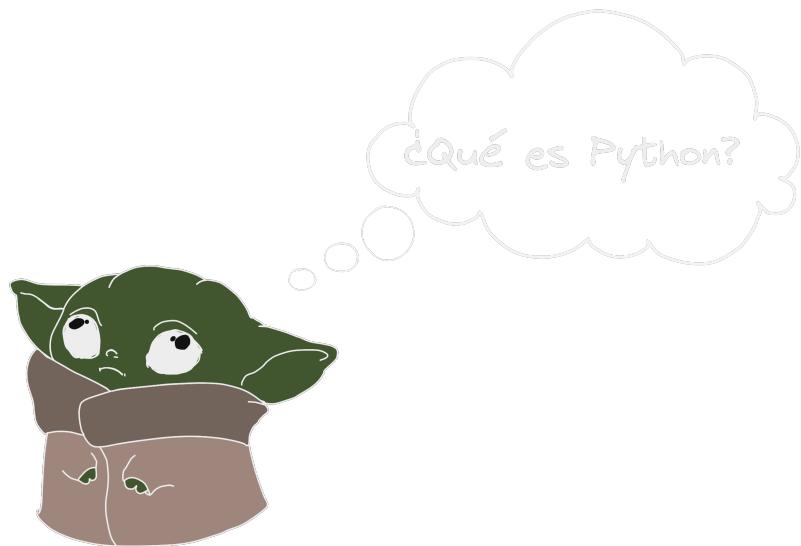
48.2.2 Ejercicio 2: Creación de una Vista y Plantilla en Django	145
48.2.3 Ejercicio 3: Uso de la Interfaz de Administración de Django	146
48.2.4 Ejercicio 4: Uso del ORM de Django	146
48.3 API de libros utilizando Django Rest Framework (DRF).	147
48.3.1 Paso 1: Configuración Inicial	147
48.4 Paso 2: Modelado de Datos	148
48.4.1 Paso 3: Serialización	148
48.4.2 Paso 4: Vistas y Rutas	148
48.4.3 Paso 5: Configuración de URLs Principales	149
48.4.4 Paso 6: Ejecutar el Servidor	149
48.4.5 Paso 7: Prueba de la API	149
48.5 Conclusiones	150
48.6 Recomendaciones	150
49 Introducción a Flask Framework	151
49.1 ¿Qué es Flask?	151
49.1.1 Ventajas de Usar Flask	151
49.2 Ecosistema de Extensiones de Flask	151
49.3 Instalación de Flask	151
49.3.1 Cómo Instalar Flask Usando pip	151
49.3.2 Creación de un Entorno Virtual para Proyectos Flask.	152
49.4 Tu Primera Aplicación en Flask	152
49.4.1 Creación de una Aplicación Web Simple	152
49.5 Definición de Rutas y Vistas en Flask	153
49.6 Plantillas HTML en Flask	153
49.7 Manejo de Formularios	153
49.7.1 Creación y Procesamiento de Formularios en Flask	153
49.7.2 Validación de Datos del Formulario.	154
49.8 Ejercicios Prácticos	154
49.9 Conclusiones	156
49.10 Recomendaciones para Trabajar con Flask	156
XII Unidad 12: Proyecto: API de Tareas con Django Rest Framework	157
50 Explicación del Proyecto	158
50.1 Qué se necesita conocer:	158
50.2 Estructura del Proyecto:	158
50.3 Código:	159
50.4 Explicación de la Actividad:	160
XIII Ejercicios	161
51 Ejercicio 1:	162
52 Ejercicio 2:	163
53 Ejercicio 3:	164

54 Ejercicio 4:	165
55 Ejercicio 5:	166
56 Ejercicio 6:	167
57 Ejercicio 7:	168
58 Ejercicio 8:	169
59 Ejercicio 9:	170
60 Ejercicio 10:	171
61 Ejercicio 11:	172
62 Ejercicio 12:	173
63 Ejercicio 13:	174
64 Ejercicio 14:	175
65 Ejercicio 15:	176
66 Ejercicio 16:	177
67 Ejercicio 17:	178
68 Ejercicio 18:	179
69 Ejercicio 19:	180
70 Ejercicio 20:	181
71 UNIDAD I: Introducción a la programación	182
71.1 UNIDAD II: Instalación de Python y más herramientas	183
71.2 UNIDAD III: Introducción a Python	183
71.3 UNIDAD IV: Tipos de Datos	184
71.4 UNIDAD V: Control de Flujo	185
71.5 UNIDAD VI: Funciones	186
71.6 UNIDAD VII: Objetos, clases y herencia	187
71.7 UNIDAD VIII: Módulos	188
71.8 UNIDAD IX: Introducción a Bases de Datos	189
71.9 UNIDAD XI: ¿Cómo me amplío con Python?	190

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 la Programación

2 Introducción general a la Programación

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

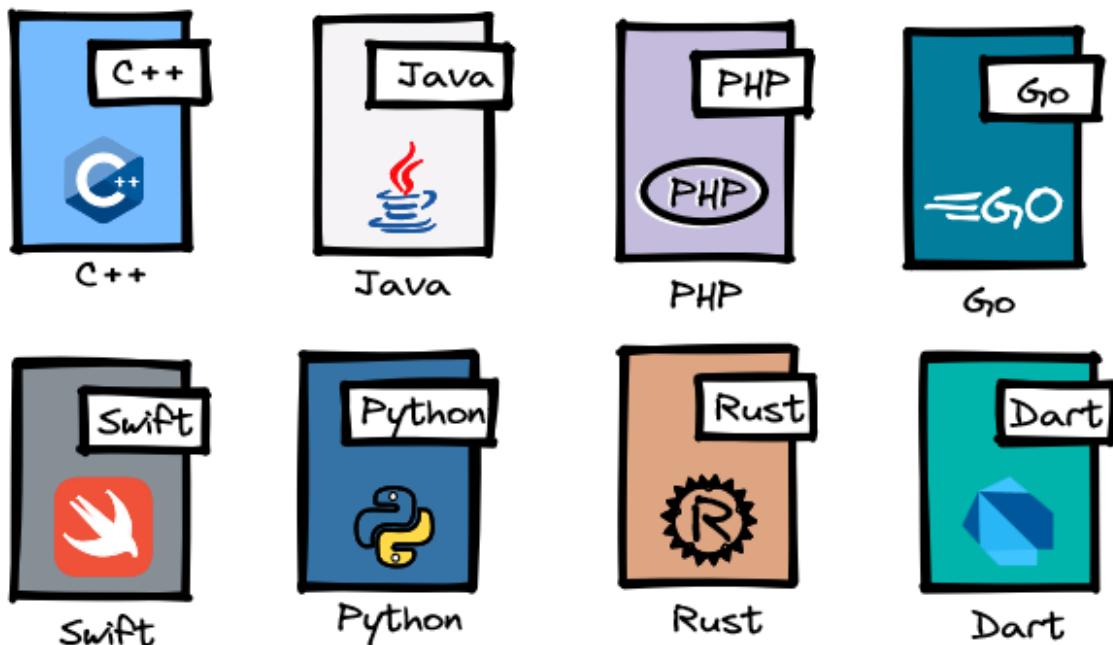
Estas instrucciones se escriben en lenguajes de programación, que son conjuntos de reglas y símbolos utilizados para comunicarse con la máquina. La programación es una habilidad esencial en la era digital, ya que se aplica en una amplia variedad de campos, desde desarrollo de software y análisis de datos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

2.1 Conceptos Clave

2.1.1 Instrucciones

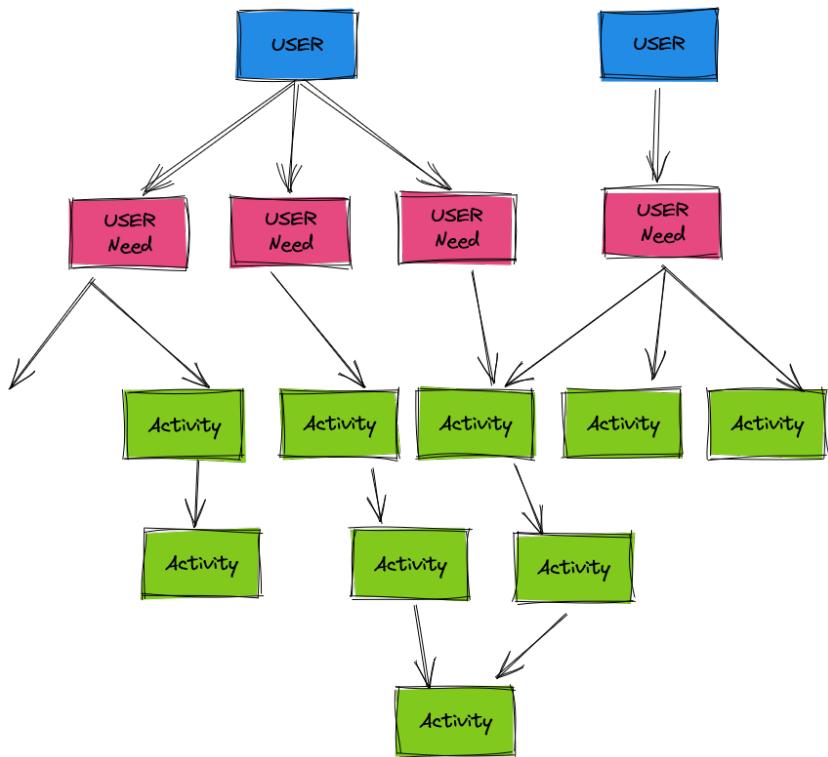
Son comandos específicos que le indican a la computadora qué hacer. Pueden ser simples, como imprimir un mensaje en pantalla, o complejas, como realizar cálculos matemáticos.

2.1.2 Lenguajes de Programación.



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

2.1.3 Algoritmos



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

2.1.4 Depuración



Debug

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.

Part II

Unidad 2: Instalación de Python y más herramientas

3 Instalación de Python

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

3.1 Conceptos Clave

3.1.1 Python



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

3.1.2 Interprete

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

3.1.3 IDE

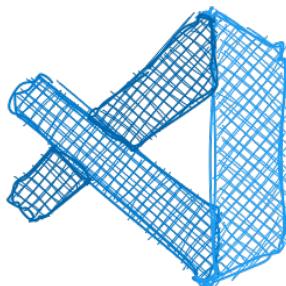
Los entornos de desarrollo integrados (IDE) como Visual Studio Code (VS Code) o PyCharm brindan herramientas para escribir, depurar y ejecutar código de manera más eficiente.

3.2 Ejemplo

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



C:\Users\Alvaro\python
Python 3.8.5 (v3.8.5:186bb86ff9, Aug 27 2020, 14:54:37) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>



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:

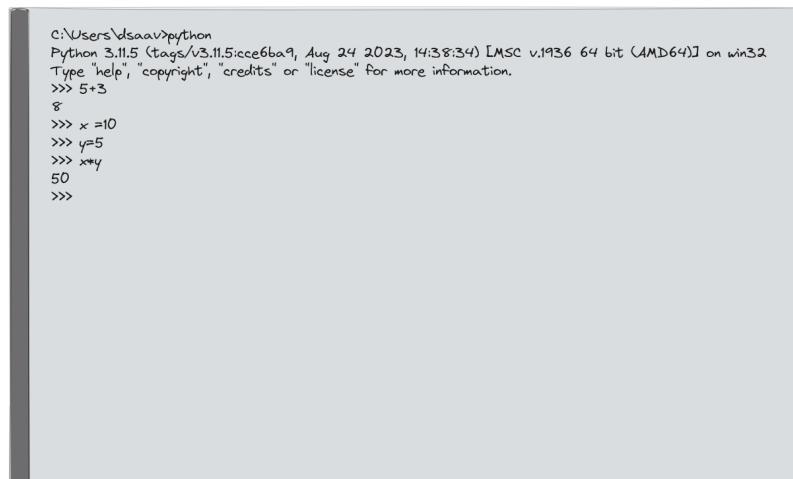
```
print("Python se ha instalado correctamente.")
```

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



```
C:\Users\dsaa\python
Python 3.11.5 (tags/v3.11.5:ccce6ba9, Aug 24 2023, 14:38:34) [MSC v.1936 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 5+3
8
>>> x =10
>>> y=5
>>> x*y
50
>>>
```

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).

```
>>> import this
```



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.

Part III

Unidad 3: Introducción a Python

5 Distintas formas de trabajar con Python.



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

5.1 Conceptos Clave:

5.1.1 Intérprete Interactivo:

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

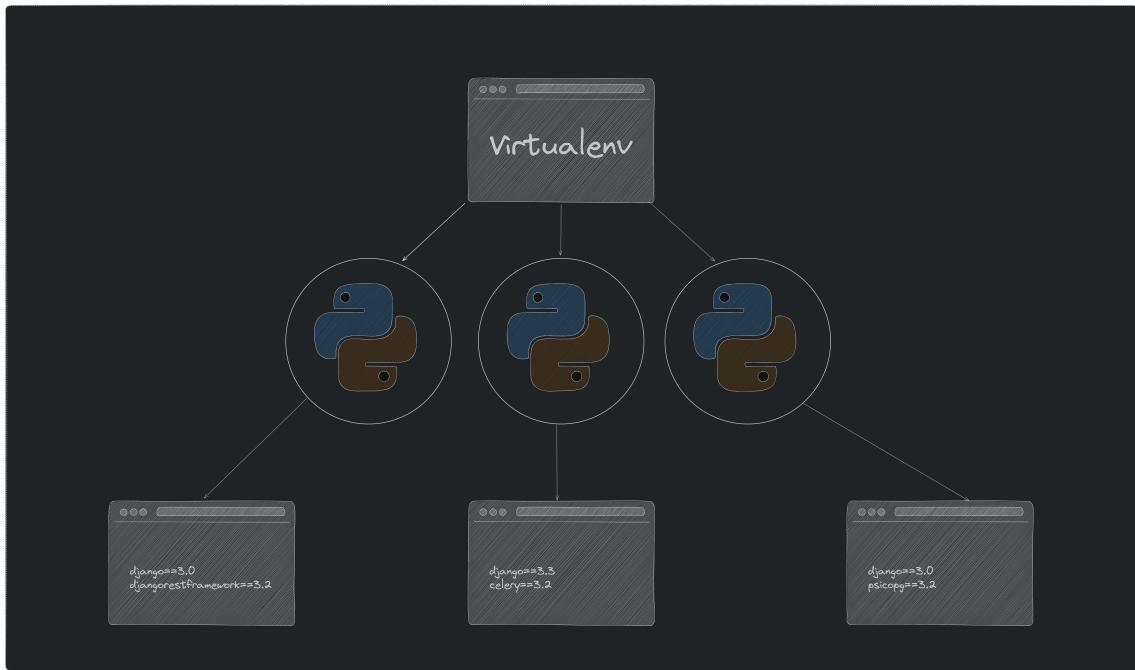
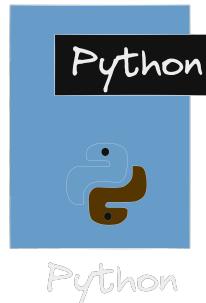
5.1.2 Scripts de Python:

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

5.1.3 Ambientes Virtuales

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

```
>>> print("Hello World")
>>> Hello World
```



5.1.4 Ejemplo

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

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

5.2 Explicación

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

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

! Actividad Práctica

Abre el intérprete interactivo de Python y realiza algunas operaciones matemáticas simples.

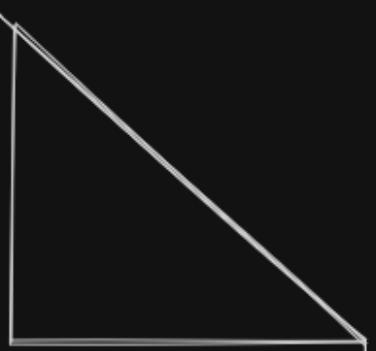
Crea un archivo llamado “operaciones.py” y escribe un programa que realice operaciones aritméticas básicas y las muestre en la consola.

5.3 Explicación de la Actividad

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

6 Las bases de Python.

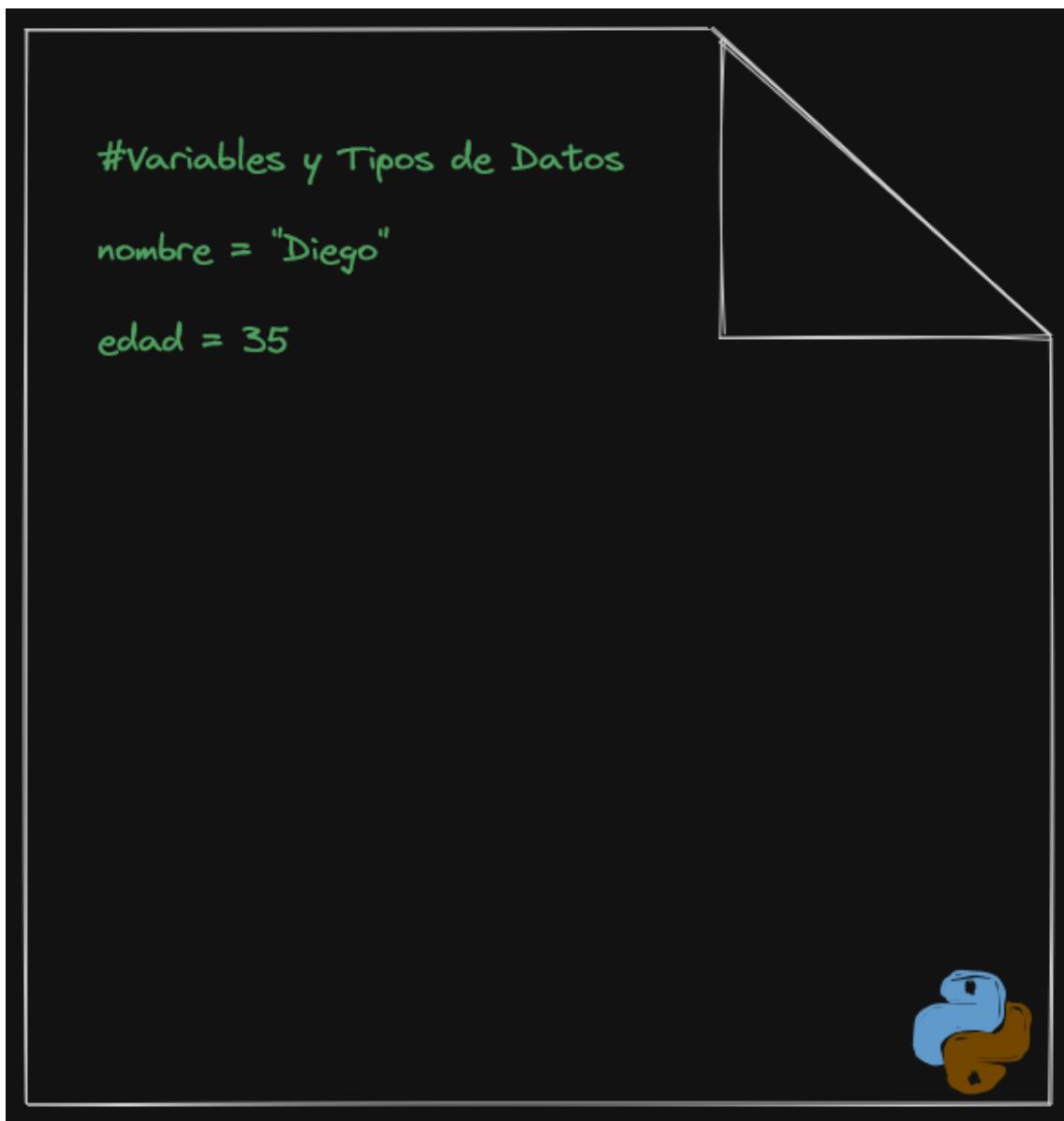
```
#Variables  
nombre = "Diego"  
edad = 35  
  
# Operadores Aritméticos  
suma = 5 + 3  
resta = 10 - 2  
  
# Operadores de Comparación  
igual = 5 == 5  
mayor_que = 10 > 5  
  
# Operadores Lógicos  
and_logico = True and False
```



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

6.1 Conceptos Clave

6.1.1 Variables



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

6.1.2 Tipos de Datos

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

#Tipos de Datos

numero = 9 Entero

nombre = "Juan" String

estatura = 1.65 Float

mayorDeEdad = True Booleano

hobbies = ["movies", "music", "read"]

List



```
#numeroMayor  
mayor = 9 >= 7  
  
#numeroMenor  
menor = 7 < 9 ← Operador de Compración  
  
#mayorEdad  
if edad >= 18:  
    print("Es mayor de edad")  
else:  
    print("Es menor de edad")
```



6.1.3 Operadores

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

6.1.4 Ejemplo

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

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

# Operadores de comparación
igual = 5 == 5
mayor_que = 10 > 5
menor_que = 7 < 12

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

6.2 Explicación:

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

Los operadores aritméticos se utilizan para realizar cálculos matemáticos básicos.

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

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

! Actividad Práctica:

Crea variables que almacenen información sobre ti, como tu nombre, edad y altura. Realiza operaciones aritméticas y utiliza operadores de comparación para comparar

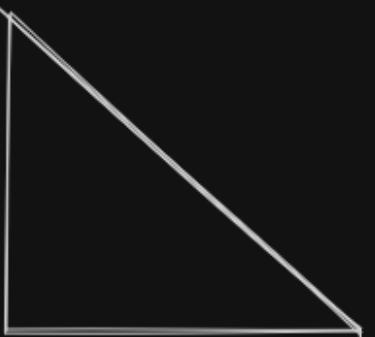
valores.

Combina expresiones booleanas utilizando operadores lógicos y observa los resultados.

6.3 Explicación de la Actividad

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

7 Identación.



.....

Escriba un programa para de
terminar si una persona es
mayor o menor de edad

.....

```
edad = int(input("ingrese su edad"))

if edad >= 18:
    print("Usted es mayor de edad")
else:
    print("Usted es menor de edad")
```

Identación



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

7.1 Conceptos Clave

7.1.1 Identación

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

7.1.2 Ejemplo

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

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

7.2 Explicación

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

! Actividad Práctica:

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

Intenta cambiar la identación incorrectamente y observa cómo afecta al funcionamiento del programa.

7.3 Explicación de la Actividad.

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

8 Comentarios

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

8.1 Conceptos Clave:

8.1.1 Comentarios

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

8.1.2 Comentarios de una línea

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

8.1.3 Comentarios de múltiples líneas

Se crean entre triple comillas (““” o ’’’) y pueden abarcar múltiples líneas.

8.2 Ejemplo

```
# Este es un comentario de una línea
"""
Este es un comentario
de múltiples líneas.
Puede abarcar varias líneas.
"""

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

8.3 Explicación:

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

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

! Important

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.

8.4 Explicación de la Actividad

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

9 Variables

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

9.1 Conceptos Clave:

9.1.1 Variables

Nombres que representan ubicaciones de memoria donde se almacenan datos.

9.1.2 Declaración de Variables.

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

9.2 Ejemplo

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

9.3 Explicación:

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

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

! Actividad Práctica

Crea variables para almacenar información personal, como tu ciudad, tu edad y tu ocupación.

Declara variables para almacenar cantidades numéricas, como el precio de un pro-

ducto y la cantidad de unidades disponibles.

9.4 Explicación de la Actividad

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

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

10 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.

10.1 Conceptos Clave:

10.1.1 Asignación Múltiple

Permite asignar valores a varias variables en una línea.

10.1.2 Desempaquetado de Valores

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

10.1.3 Intercambio de Valores

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

10.2 Ejemplo

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

10.3 Explicación:

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

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

! Actividad Práctica:

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

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

10.4 Explicación de la Actividad

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

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

11 Concatenación

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

11.1 Conceptos Clave:

11.1.1 Concatenación:

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

11.1.2 Operador +

Se utiliza para concatenar cadenas de texto.

11.1.3 Conversión a Cadena

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

11.2 Ejemplo

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

11.3 Explicación:

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

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

! Actividad Práctica:

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

11.4 Explicación de la Actividad

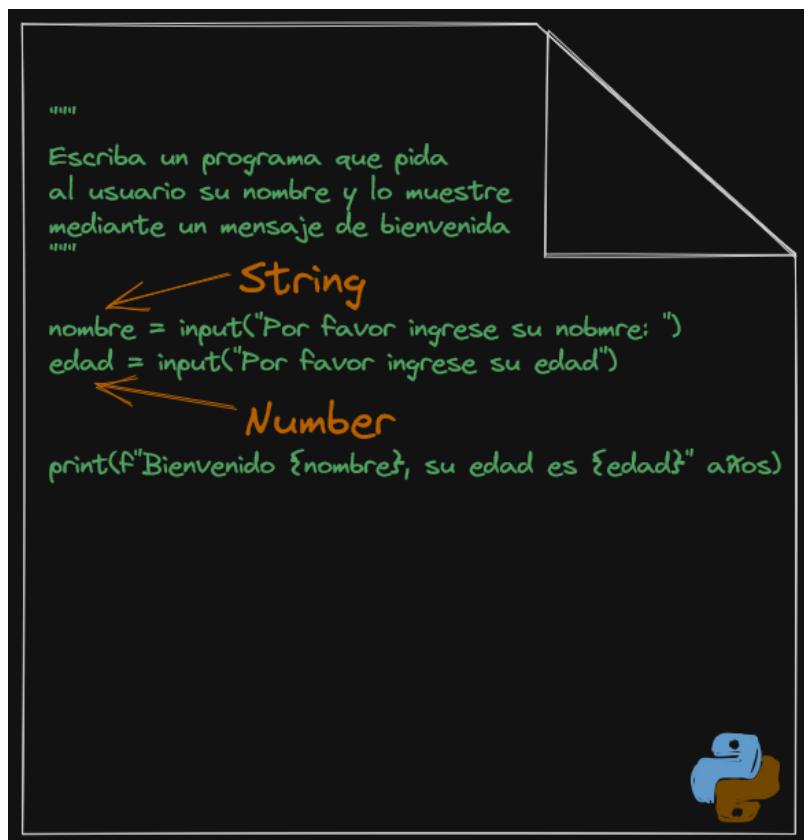
Esta actividad permite a los participantes practicar la concatenación de cadenas de texto y comprender cómo construir mensajes más complejos utilizando variables y texto. Les ayuda a mejorar su capacidad para crear mensajes personalizados en sus programas.

Part IV

Unidad 4: Tipos de Datos

12 String y Números

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



12.1 Conceptos Clave:

12.1.1 String

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

12.1.2 Números Enteros (int)

Representan números enteros positivos o negativos.

Escriba un programa que pida al usuario su nombre y lo muestre mediante un mensaje de bienvenida

String

```
nombre = input("Por favor ingrese su nombre: ")  
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

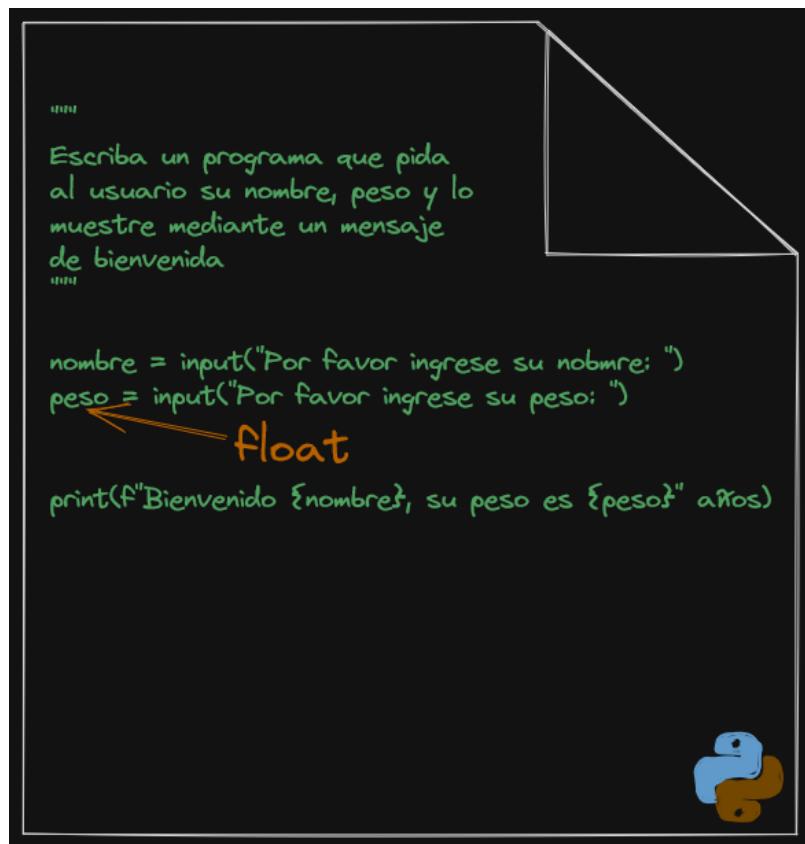
```
nombre = input("Por favor ingrese su nombre: ")  
edad = input("Por favor ingrese su edad")
```

Number

```
print(f"Bienvenido {nombre}, su edad es {edad} años")
```



12.1.3 Números de Punto Flotante (float)



Representan números con decimales.

12.2 Ejemplo

```
# Strings  
message = "Hola, bienvenido al curso de Python."  
name = 'María'  
  
# Números  
age = 25  
balance = 1500.75
```

12.3 Explicación

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

Los strings se definen utilizando comillas simples o dobles.

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

! Actividad Práctica:

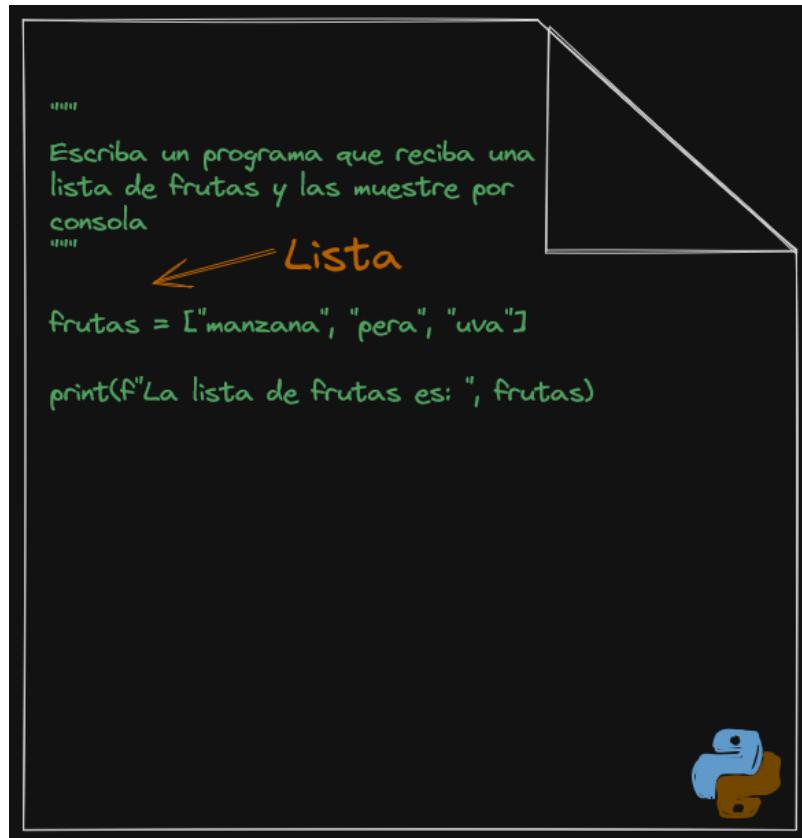
1. Crea una variable con tu canción favorita.
2. Asigna tu edad a una variable y tu altura a otra variable.
3. Combina las variables para crear un mensaje personalizado.

12.4 Explicación de la Actividad:

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

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

13 Listas



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

13.1 Conceptos Clave:

13.1.1 Listas

Secuencias ordenadas de elementos que pueden ser de diferentes tipos.

13.1.2 Índices

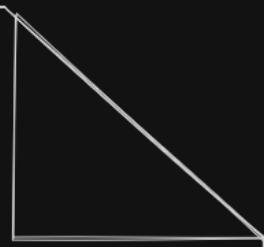
Números que indican la posición de un elemento en la lista.

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}")
```



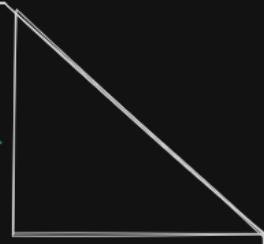
Escriba un programa que reciba una lista de frutas y las muestre por consola

Lista

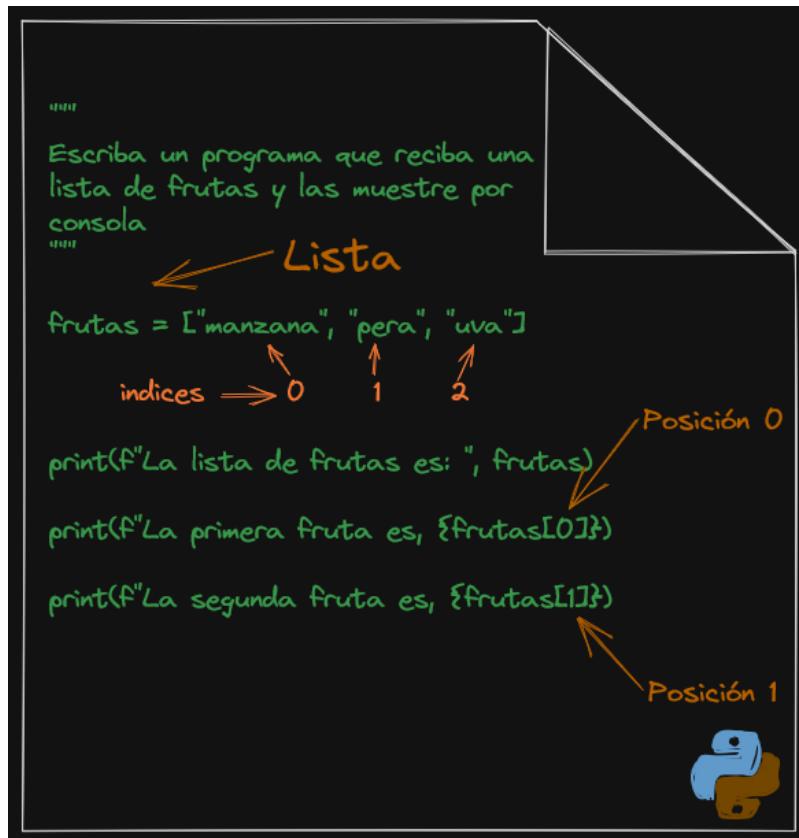
```
frutas = ["manzana", "pera", "uva"]
```

indices \Rightarrow 0 1 2

```
print(f"La lista de frutas es: ", frutas)
```



13.1.3 Acceso a Elementos.



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

13.2 Ejemplo

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

13.3 Explicación

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

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

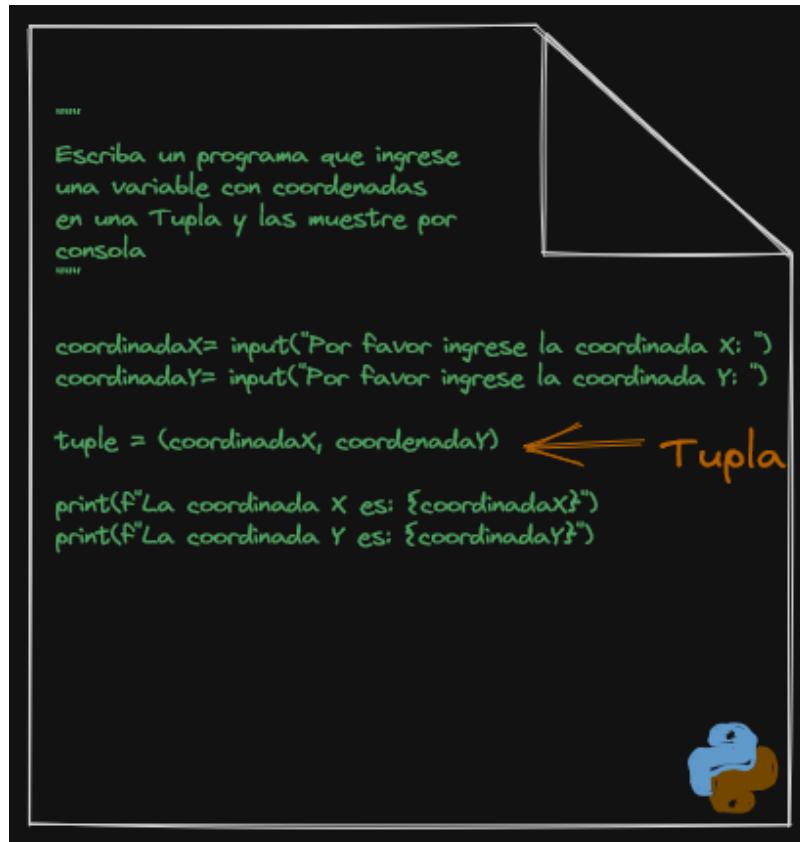
! Actividad Práctica:

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

13.4 Explicación de la Actividad:

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

14 Tuplas.



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

14.1 Conceptos Clave:

14.1.1 Tuplas

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

14.1.2 Inmutabilidad

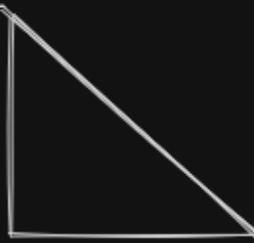
Una vez creada una tupla, no se pueden agregar, modificar o eliminar elementos.

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, coordinadaY) ← Tupla

print(f"La coordinada X es: {coordinadaX}")
print(f"La coordinada Y es: {coordinadaY}")
```

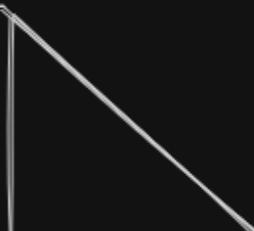


Escriba un programa que ingrese una variable con coordenadas en una Tupla y las muestre por consola

```
coordinadaX= input("Por favor ingrese la coordinada X: ")
coordinadaY= input("Por favor ingrese la coordinada Y: ")

tuple = (coordinadaX, coordinadaY) ← Inmutable

print(f"La coordinada X es: {coordinadaX}")
print(f"La coordinada Y es: {coordinadaY}")
```



14.1.3 Acceso a Elementos

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

14.2 Ejemplo

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

14.3 Explicación

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

! Actividad Práctica:

Crea una tupla con las estaciones del año.

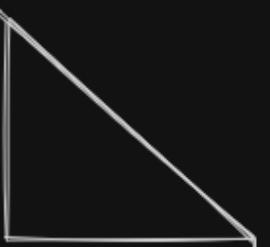
Intenta modificar un elemento de la tupla y observa el error que se produce.

14.4 Explicación de la Actividad:

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

15 Range

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



```
# Generación de secuencias de números
Range
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_secuencia = list(secuencia1)
```



15.1 Conceptos Clave:

15.1.1 range

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

15.1.2 Parámetros de range

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

15.1.3 Conversión a Listas

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

15.2 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)
```

15.3 Explicación

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

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

! Actividad Práctica:

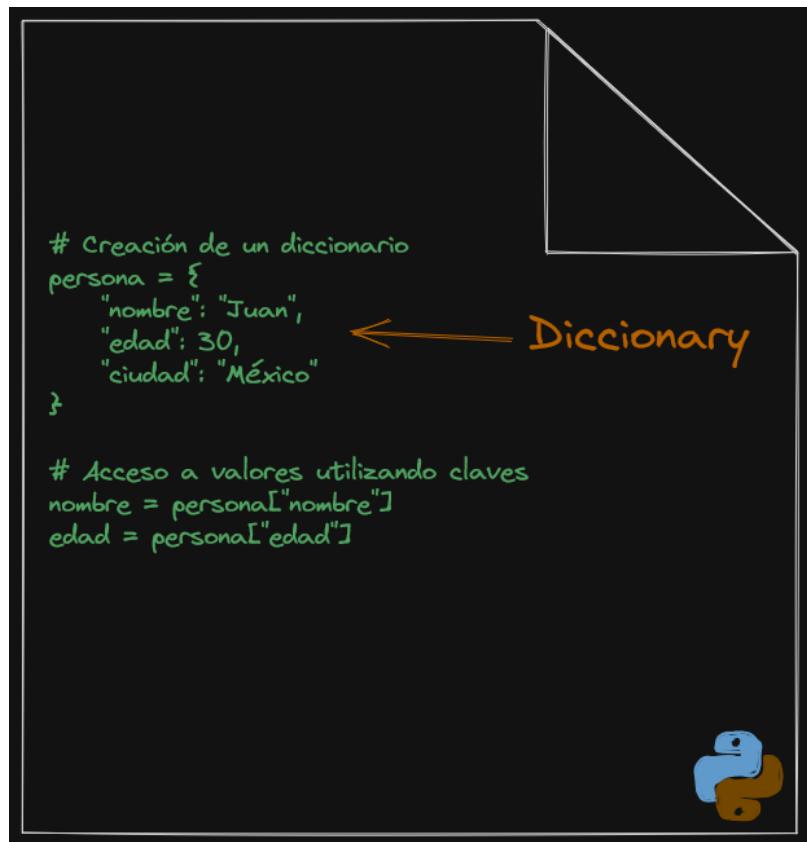
Crea una secuencia de números del 10 al 20 con un paso de 2.

Convierte la secuencia de números en una lista y muestra los elementos en la consola.

15.4 Explicación de la Actividad:

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

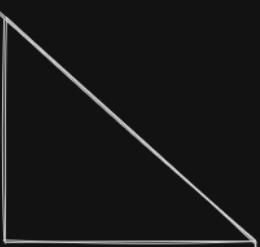
16 Diccionarios



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

16.1 Conceptos Clave:

16.1.1 Diccionarios



```
# Creación de un diccionario
persona = {
    "nombre": "Juan",
    "edad": 30,
    "ciudad": "México"
}
# Acceso a valores utilizando claves
nombre = persona["nombre"]
edad = persona["edad"]
```



Estructuras de datos que almacenan pares clave-valor.

16.1.2 Claves

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

16.1.3 Valores

Son los datos asociados a cada clave en el diccionario.

16.2 Ejemplo

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

```
    "ciudad": "México"  
}  
  
# Acceso a valores utilizando claves  
nombre = persona["nombre"]  
edad = persona["edad"]
```

16.3 Explicación

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

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

! Actividad Práctica

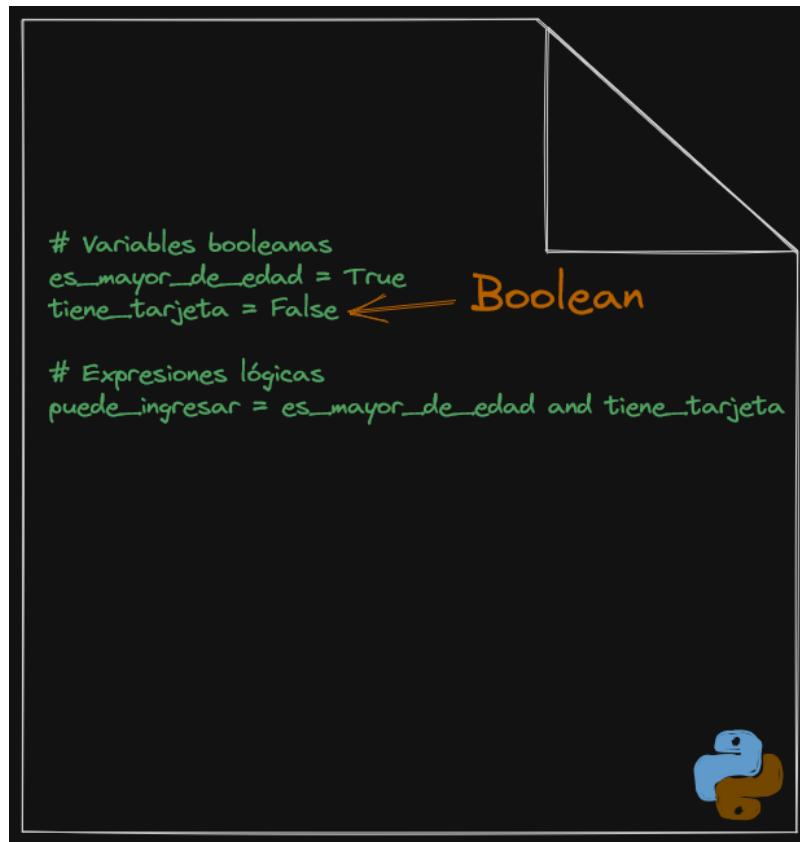
Crea un diccionario que almacene información de tus libros favoritos, incluyendo título y autor.

Accede a los valores del diccionario utilizando las claves y muestra la información en la consola.

16.4 Explicación de la Actividad:

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

17 Booleanos.



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

17.1 Conceptos Clave:

17.1.1 Booleanos

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

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

17.2 Ejemplo

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

# Expresiones lógicas
puede_ingresar = es_mayor_de_edad and tiene_tarjeta
```

17.3 Explicación

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

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

! Actividad Práctica:

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

Utiliza expresiones lógicas para determinar si puedes llevar a tu mascota a un lugar que requiere tu atención durante un partido de tu deporte favorito.

17.4 Explicación de la Actividad:

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

Part V

Unidad 5: Control de Flujo

18 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.

18.1 Conceptos Clave:

18.1.1 Control de Flujo

Manejo de la ejecución del código basado en condiciones.

18.1.2 Estructura if

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

18.1.3 Bloque de Código

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

18.2 Ejemplo:

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

18.3 Explicación:

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

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

! Actividad Práctica:

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

18.4 Explicación de la Actividad:

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

19 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.

19.1 Conceptos Clave

19.1.1 Estructura elif

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

19.1.2 Estructura else

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

19.1.3 Anidación de Estructuras if

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

19.2 Ejemplo:

```
puntaje = 85

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

19.3 Explicación:

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

! Actividad Práctica:

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

19.4 Explicación de la Actividad:

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

20 If, elif y else

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

20.1 Conceptos Clave:

20.1.1 Anidación de Estructuras

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

20.1.2 Jerarquía de Condiciones

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

20.2 Ejemplo

```
edad = 16
permiso_padres = True

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

20.3 Explicación:

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

! Actividad Práctica:

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

20.4 Explicación de la Actividad:

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

21 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.

21.1 Conceptos Clave:

21.1.1 Operador and

Retorna True si ambas condiciones son verdaderas.

21.1.2 Operador or

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

21.1.3 Combinación de Condiciones

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

21.2 Ejemplo:

```
edad = 20
tiene_permiso = True

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

21.3 Explicación:

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

Si ambas condiciones son verdaderas, se permite el ingreso.

! Actividad Práctica:

Crea dos variables que representen si un usuario tiene una cuenta premium y si su suscripción está activa.

Utiliza una estructura if y el operador and para determinar si el usuario tiene acceso premium.

21.4 Explicación de la Actividad:

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

22 Introducción a While

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

22.1 Conceptos Clave:

22.1.1 Estructura while

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

22.1.2 Condición

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

22.2 Ejemplo:

```
contador = 0

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

22.3 Explicación:

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

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

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

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

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

22.4 Explicación de la Actividad:

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

23 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.

23.1 Conceptos Clave:

23.1.1 Sentencia break:

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

23.1.2 Bucles Infinitos:

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

23.2 Ejemplo:

```
contador = 0

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

23.3 Explicación:

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

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

! Actividad Práctica:

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

23.4 Explicación de la Actividad:

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

24 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.

24.1 Conceptos Clave:

24.1.1 Sentencia continue

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

24.1.2 Saltar Iteraciones

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

24.2 Ejemplo

```
contador = 0

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

24.3 Explicación:

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

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

! Actividad Práctica:

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

Utiliza la sentencia continue para lograr esto.

24.4 Explicación de la Actividad

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

25 For Loop

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

25.1 Conceptos Clave:

25.1.1 Bucle for

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

25.1.2 Iteración

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

25.1.3 range() con Bucles for

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

25.2 Ejemplo:

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

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

25.3 Explicación:

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

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

! Actividad Práctica:

Crea una lista de colores.

Utiliza un bucle for para imprimir cada color en la lista precedido por la palabra “Color:”.

25.4 Explicación de la Actividad:

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

Part VI

Unidad 6: Funciones

26 Introducción a Funciones

En esta lección, nos adentraremos en el mundo de las funciones en Python, una de las características más poderosas del lenguaje. Aprenderemos cómo **definir funciones, pasar argumentos y retornar valores**.

26.0.1 Conceptos Clave

26.0.1.1 Funciones:

Son bloques de código reutilizables que realizan tareas específicas. Las funciones permiten dividir el código en partes más pequeñas y manejables, lo que facilita la lectura, la depuración y la reutilización del código.

26.0.2 Definición de Funciones

Para crear una función en Python, utilizamos la palabra clave def, seguida del nombre de la función y paréntesis. Veamos un ejemplo:

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

En este caso, hemos definido una función llamada saludar que toma un argumento llamado nombre.

26.0.3 Argumentos

Los argumentos son valores que se pasan a una función para que trabaje con ellos. En el ejemplo anterior, nombre es un argumento que se utiliza para personalizar el saludo.

26.0.4 Retorno de Valores

Una función puede devolver un valor utilizando la palabra clave return. En nuestro ejemplo, la función saludar retorna un saludo personalizado. Veamos cómo se utiliza:

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

En este fragmento de código, hemos llamado a la función saludar pasando el argumento “Juan”. La función devuelve “Hola, Juan”, que se almacena en la variable mensaje y se muestra en la consola.



Tip

Actividad Práctica

Ahora, pondremos en práctica lo que hemos aprendido. Tu tarea es crear una función llamada calcular_cuadrado que reciba un número como argumento y retorne el cuadrado de ese número.

```
def calcular_cuadrado(numero):
    return numero ** 2

resultado = calcular_cuadrado(5)
print("El cuadrado es:", resultado)
```

En este ejemplo, hemos definido la función calcular_cuadrado que toma un número como argumento, lo eleva al cuadrado y lo retorna. Luego, llamamos a esta función con el número 5 y mostramos el resultado en la consola.

26.0.5 Explicación de la Actividad

Esta actividad te permitirá practicar la creación y uso de funciones en Python. Aprenderás cómo definir una función que realice un cálculo específico y cómo utilizarla para obtener resultados. Las funciones son una parte fundamental de la programación en Python y te ayudarán a escribir código más organizado y eficiente.

27 Recursividad

En esta lección, nos adentraremos en el intrigante concepto de la recursividad.

La recursión es una técnica en la que una función se llama a sí misma para resolver un problema.

Aprenderemos cómo implementar **funciones recursivas** y cuándo es apropiado usarlas.

27.0.1 Conceptos Clave

Recursividad:

Es una técnica en la que una función se llama a sí misma para resolver un problema. Puede parecer un enfoque un tanto sorprendente al principio, pero es especialmente útil en situaciones en las que un problema se puede dividir en subproblemas más pequeños.

Caso Base:

La recursión debe tener una forma de detenerse. Un caso base es una condición que indica cuándo la recursión debe finalizar. Si no se establece un caso base adecuado, la recursión puede continuar indefinidamente.

Caso Recursivo:

En cada llamada recursiva, el problema original se divide en partes más pequeñas. Estas partes más pequeñas se resuelven mediante llamadas recursivas hasta que se alcanza el caso base. Ejemplo

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

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

27.0.2 Explicación

En este ejemplo, hemos definido una función recursiva llamada **factorial** para calcular el factorial de un número.

El factorial de un número entero no negativo n se define como el producto de todos los enteros positivos menores o iguales a n .

Por ejemplo, $5!$ (pronunciado “cinco factorial”) es igual a $5 \times 4 \times 3 \times 2 \times 1$.

La función **factorial** utiliza dos componentes clave de la recursión:

Caso Base: Cuando n es igual a 0, retornamos 1. Este es el caso base que detiene la recursión. Si no lo tuviéramos, la función se llamaría a sí misma infinitamente.

Caso Recursivo: En todas las demás situaciones (cuando n no es igual a 0), la función se llama a sí misma con $n - 1$ y multiplica el resultado por n . Esto es lo que hace que la función sea recursiva.

La llamada **factorial(5)** resultará en $5 * 4!$, que a su vez se descompone en $5 * 4 * 3!$, y así sucesivamente hasta que alcance el caso base.



Tip

Actividad Práctica

Tu tarea es crear una función recursiva llamada **potencia** que calcule la potencia de un número base elevado a un exponente. Utiliza la función para calcular 2^3 y muestra el resultado en la consola.

27.0.3 Explicación de la Actividad

Esta actividad te permitirá aplicar la recursividad de manera práctica. Aprenderás a definir una función recursiva que resuelve un problema matemático y a comprender cómo se dividen los problemas en subproblemas más pequeños en cada llamada recursiva. La recursión es una técnica poderosa que se utiliza en muchos campos de la informática, como algoritmos, estructuras de datos y más.

Part VII

Unidad 7: Objetos, Clases y Herencia

28 Programación Orientada a Objetos (POO)

La Programación Orientada a Objetos (POO) es un paradigma de programación que se basa en el uso de objetos y clases.

En POO, tratamos los elementos del mundo real como objetos, y estos objetos interactúan entre sí para realizar tareas específicas.

Este enfoque nos permite organizar nuestro código de manera más eficiente y estructurada.

28.1 Conceptos Clave en POO

28.1.1 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.

28.1.2 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.

28.1.3 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”.

28.1.4 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.

28.1.5 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.

29 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. 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 di-

29.0.1 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.

29.0.2 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:

- 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”.
- Implementamos un método de instancia llamado “`calcular_area`” que calcula el área del rectángulo multiplicando el ancho por el alto y lo retorna.
- Creamos dos instancias de la clase “Rectangulo” con diferentes dimensiones (5x10 y 3x7).
- Calculamos el área de cada rectángulo llamando al método “`calcular_area`” en cada instancia.
- Mostramos el área de cada rectángulo en la consola.

29.0.3 Explicación

Este ejemplo ilustra cómo trabajar con atributos y métodos de instancia en una clase, permitiendo que cada objeto tenga sus propios valores y pueda realizar acciones específicas.

Esta actividad te permite practicar la creación de instancias de clase, trabajar con atributos y métodos de instancia, y entender cómo cada objeto puede tener sus propios valores y realizar acciones específicas.

30 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. 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.

Ejemplo:

```
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}")
```

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.

30.0.1 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:

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

```

def saludar(self):
    print(f"Hola, soy {self.nombre} y tengo {self.edad} años.")

# Crear una instancia de Persona
personal1 = Persona("Ana", 30)

# Llamar al método saludar para mostrar un saludo personalizado
personal1.saludar()

```

En este ejemplo, el método “**saludar**” accede a los atributos de instancia “**nombre**” y “edad” utilizando “`self.nombre`” y “`self.edad`” para mostrar un saludo personalizado.

Actividad Práctica:

Crea una clase Triángulo con atributos “base” y “altura”, y un método “calcular_area” que calcule y retorne el área del triángulo ($\text{base} * \text{altura} / 2$).

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 ($\text{base} * \text{altura} / 2$). 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}")

```

Explicación:

- 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”.
- Implementamos un método de instancia llamado “calcular_area” que calcula el área del triángulo utilizando la fórmula $(\text{base} * \text{altura}) / 2$ y lo retorna.
- Creamos una instancia de la clase “Triángulo” con dimensiones específicas (base 6 y altura 8).
- Calculamos el área del triángulo llamando al método “calcular_area” en la instancia.
- Mostramos el área del triángulo en la consola.

30.0.2 Explicación

Esta actividad te permite practicar la definición y uso de métodos en una clase y cómo acceder a los atributos de instancia dentro de los métodos.

31 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. 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.

Ejemplo:

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

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

31.0.1 Eliminar Atributos

Se puede eliminar un atributo de instancia utilizando la palabra clave “**del**”.

Ejemplo:

```
class Coche:  
    def __init__(self, marca, modelo):  
        self.marca = marca  
        self.modelo = modelo  
  
    def mostrar_info():  
        print(f"Coche {self.marca} {self.modelo}")  
  
# Crear una instancia de Coche  
coche1 = Coche("Toyota", "Corolla")
```

```

# Eliminar el atributo 'modelo'
del coche1.modelo

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

```

Aquí, hemos creado una instancia de “Coche” y luego eliminado el atributo “modelo” utilizando “del”.

31.0.2 Eliminar Objetos

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

Ejemplo:

```

class Punto:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def mostrar(self):
        print(f"Punto ({self.x}, {self.y})")

# Crear una instancia de Punto
punto1 = Punto(2, 3)

# Llamar al método para mostrar el punto
punto1.mostrar()

# Eliminar el objeto punto1
del punto1

# Intentar acceder al objeto eliminado generará un error
# punto1.mostrar()

```

En este ejemplo, creamos una instancia “**punto1**” de la clase “**Punto**” y luego la eliminamos utilizando “del”.

💡 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 Triángulo

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():
        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()
```

Resolución.

- Hemos definido una clase llamada Estudiante con un constructor (**init**) que toma dos atributos: nombre y edad. Estos atributos representan el nombre y la edad del estudiante.
- La clase Estudiante también tiene un método llamado mostrar_info que imprime la información del estudiante en un formato específico.
- Luego, hemos creado una instancia estudiante1 de la clase Estudiante con el nombre “María” y la edad 22.
- Hemos llamado al método mostrar_info en la instancia estudiante1, lo que muestra la información del estudiante en la consola.
- Finalmente, hemos utilizado la palabra clave del para eliminar el atributo nombre de la instancia estudiante1.

Esto se hace para mostrar cómo eliminar un atributo de instancia.

Explicación:

Este código muestra cómo definir una clase en Python, crear una instancia de esa clase y llamar a sus métodos.

También ilustra cómo se pueden eliminar atributos de instancia utilizando del. La eliminación de atributos es útil en situaciones específicas donde se necesita gestionar dinámicamente los atributos de un objeto.

31.0.3 Explicación

Esta actividad te permite practicar el uso de “self” en los métodos de clase, cómo eliminar atributos de instancia y objetos en Python, y cómo gestionar la memoria.

32 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.

32.0.1 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.

Ejemplo:

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

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

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

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

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

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 Círculo que herede de “Figura” y agregue un atributo “radio” y un método “calcular_area” para calcular el área del círculo.

Possible 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 “Círculo” que hereda de “Figura” y agrega un atributo “radio” y un método “calcular_area”. Se crea una instancia de “Círculo”, 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 "Círculo" que hereda de "Figura" y agrega un atributo "radio".
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 "Círculo" llamada "circulo1" con color "Rojo" y radio 5.
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 círculo.
area = circulo1.calcular_area()

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

Explicación:

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

Esta práctica te ayudará a comprender cómo usar la herencia para crear clases derivadas y reutilizar código de clases base.

33 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.

33.1 Conceptos Clave:

33.1.1 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. 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. 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.

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 sobre escriben 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 sobre scriban el método “**arrancar**” para proporcionar un mensaje específico para cada tipo de vehículo.

Possible Solución

```
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")

# Crear instancias de las clases
coche = Coche()
moto = Motocicleta()

# Llamar al método "arrancar" para diferentes vehículos
coche.arrancar() # Salida: El coche arranca
moto.arrancar() # Salida: La motocicleta arranca
```

Explicación:

- 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**”.
- Definimos la clase derivada “**Perro**” que hereda de “**Animal**” mediante (`Animal`). Esta clase sobrescribe el método “**sonido**” para que devuelva “Guau” cuando se llama.

- Similar a la clase “Perro”, definimos la clase derivada “Gato” que hereda de “Animal” y sobrescribe el método “sonido” para que devuelva “Miau”.
- Creamos una función llamada “hacer_sonar” que toma un objeto de la clase “Animal” como argumento y llama a su método “sonido”. Esto permite tratar objetos de diferentes clases como si fueran de la misma clase base.
- Creamos instancias “perro” y “gato” de las clases “Perro” y “Gato”, respectivamente.
- Llamamos a la función “hacer_sonar” con objetos de diferentes clases (“perro” y “gato”). A pesar de que son objetos de clases diferentes, el polimorfismo permite que el método “sonido” adecuado se ejecute para cada objeto. En el caso de “perro”, imprime “Guau”, y en el caso de “gato”, imprime “Miau”.

El polimorfismo nos permite tratar objetos de clases derivadas de manera uniforme cuando comparten una clase base común, lo que facilita el diseño y la flexibilidad de nuestros programas.

33.1.2 Explicación:

Esta actividad te permitirá practicar cómo implementar el polimorfismo mediante la sobrescritura de métodos en clases derivadas y cómo tratar objetos de diferentes clases de manera uniforme.

34 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.

34.1 Conceptos Clave:

34.1.1 Encapsulación

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.

34.1.2 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`).

34.1.3 Métodos Privados

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

34.1.4 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())

cuenta = CuentaBancaria(1000)
print("Saldo inicial:", cuenta.obtener_saldo())
cuenta.depositar(500)
print("Saldo después del depósito:", cuenta.obtener_saldo())
cuenta.retirar(200)
print("Saldo después del retiro:", cuenta.obtener_saldo())

```

34.1.5 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`).
- ② El método `get_nombre` permite obtener el nombre del estudiante, y el método `set_nombre` permite cambiar el nombre si la nueva cadena de nombre tiene una longitud mayor que 0.
- ③ Se crea una instancia de la clase Estudiante con el nombre “Juan”, se imprime el nombre inicial utilizando `get_nombre`, se cambia el nombre a “María” utilizando `set_nombre`, y se imprime el nombre nuevamente después del cambio.

34.2 ¿Qué aprendimos en esta actividad?

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

Part VIII

Unidad 8: Módulos

35 Introducción

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

35.1 Conceptos Clave:

35.1.1 Módulos

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

35.1.2 Importar Módulos

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

35.1.3 Usar Funciones y Clases

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

35.2 Ejemplo:

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

# En otro archivo
import calculadora

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

35.3 Explicación:

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

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

! Actividad Práctica:

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

Importa el módulo en otro archivo y utiliza la función multiplicacion para calcular el producto de dos números.

35.4 Explicación de la Actividad:

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

36 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.

36.1 Pasos para Crear un Módulo:

Crea un archivo de Python con la extensión .py.

Define funciones y clases en el archivo.

Guarda el archivo en una ubicación accesible.

36.2 Ejemplo:

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

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

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

36.3 Explicación:

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

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

! Actividad Práctica:

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

En otro archivo, importa el módulo geometria y utiliza las funciones para realizar cálculos geométricos.

36.4 Explicación de la Actividad:

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

37 Renombrando Módulos

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

37.1 Renombrando Módulos al Importar:

```
import modulo_largo as ml
```

37.2 Seleccionando Elementos Específicos para Importar:

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

37.3 Ejemplo - Renombrando Módulos:

```
import calculadora as calc  
  
resultado = calc.suma(3, 4)
```

37.4 Ejemplo - Seleccionando Elementos Específicos:

```
from operaciones import resta, Calculadora  
  
resultado = resta(10, 5)
```

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

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

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

37.5 Explicación de la Actividad:

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

38 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.

38.1 Seleccionando Elementos Específicos para Importar:

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

38.1.1 Usando Pip:

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

38.2 Ejemplo - Instalando un Paquete con Pip:

```
pip install requests
```

! Actividad Práctica:

Utiliza pip para instalar el paquete matplotlib, que se utiliza para trazar gráficos en Python.

En tu archivo de código, importa la función plot de matplotlib.pyplot y crea un gráfico simple.

38.3 Explicación de la Actividad:

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

Part IX

Unidad 9: Introducción a Bases de Datos

39 Introducción a Bases de Datos

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

39.1 Conceptos Clave:

39.1.1 Base de Datos

Colección organizada de datos almacenados en formato estructurado.

39.1.2 Sistemas de Gestión de Bases de Datos (DBMS)

Software que administra y gestiona una base de datos.

39.1.3 Beneficios de las Bases de Datos

Almacenamiento eficiente, acceso rápido y seguridad de datos.

39.2 Ejemplo:

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

39.3 Explicación:

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

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

! Actividad Práctica:

Investiga y elige un Sistema de Gestión de Bases de Datos (DBMS) para utilizar en el curso.

Explica por qué es importante utilizar bases de datos en el desarrollo de aplicaciones.

39.4 Explicación de la Actividad:

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

40 Introducción a PostgreSQL

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

40.0.1 Instalación de PostgreSQL:

Descargar e instalar PostgreSQL desde el sitio oficial.

Configurar contraseña para el usuario ‘postgres’.

40.0.2 Operaciones Básicas en PostgreSQL:

40.0.3 Crear una base de datos:

```
CREATE DATABASE nombre;
```

40.0.4 Conectar a una base de datos:

```
\c nombre;
```

40.0.5 Crear una tabla:

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

40.0.6 Insertar registros:

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

40.0.7 Consultar registros:

```
SELECT * FROM tabla;
```

40.0.8 Actualizar registros:

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

40.0.9 Eliminar registros:

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

40.1 Ejemplo - Creación de una Tabla en PostgreSQL:

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

40.1.1 Actividad Práctica:

Instala PostgreSQL en tu entorno.

Crea una base de datos llamada ‘universidad’.

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

Inserta al menos dos registros en la tabla.

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

40.2 Explicación de la Actividad:

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

41 Introducción a MongoDB

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

41.1 Instalación de MongoDB:

Descargar e instalar MongoDB desde el sitio oficial.

Configurar directorio de datos y logs.

41.2 Operaciones Básicas en MongoDB:

41.2.1 Crear una base de datos:

```
use nombre;
```

41.2.2 Crear una colección (tabla):

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

41.2.3 Insertar documentos (registros):

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

41.2.4 Consultar documentos:

```
db.colección.find();
```

41.2.5 Actualizar documentos:

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

41.2.6 Eliminar documentos:

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

41.3 Ejemplo - Creación de una Colección en MongoDB:

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

! Actividad Práctica:

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

41.4 Explicación de la Actividad:

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

Part X

Unidad 10: Operaciones Básicas en 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 XI

Unidad 11: ¿Cómo me amplío con Python?

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 nuestros 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.).
- ② Definir al menos tres campos para el modelo.
- ③ Ejecutar las migraciones para aplicar el modelo a la base de datos.

```
# 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.
- ② Crear una plantilla HTML que muestre los datos en una página web.
- ③ Configurar una URL para acceder a la vista.

```
# 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 --&gt;

&lt;!DOCTYPE html&gt;
&lt;html&gt;</pre>
```

```

<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.
- ② Ejecuta el siguiente comando para crear un entorno virtual.

```
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.

```

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:

```

<!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:

```

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 XII

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

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 XIII

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.