Taller de Python: Desde Cero hasta Experto

Módulo 1: Fundamentos Básicos

Objetivo: Entender la sintaxis y conceptos esenciales.

- Temas:
  - Instalación de Python y uso de un editor (como VSCode o IDLE).
  - Variables y tipos de datos (`int`, `float`, `str`, `bool`).
  - Operadores básicos (`+`, `-`, `*`, `/`, `%`, `==`, `<`, `>`).
  - Entrada y salida (`input()`, `print()`).
  -
- Ejemplo:

```
python
nombre = input("¿Cuál es tu nombre? ")
edad = int(input("¿Cuántos años tienes? "))
print(f"Hola, {nombre}. Tienes {edad} años.")
```

- Ejercicio: Crea un programa que pida dos números, los sume y muestre el resultado.

Módulo 2: Estructuras de Control

Objetivo: Tomar decisiones y repetir acciones.

- Temas:
  - Condicionales (`if`, `elif`, `else`).
  - Bucles (`for`, `while`, `break`, `continue`).
  -
- Ejemplo:

```
python
numero = int(input("Dame un número: "))
if numero > 0:
    print("Positivo")
elif numero == 0:
    print("Cero")
else:
    print("Negativo")
python
for i in range(5):
    print(i)  # 0, 1, 2, 3, 4
```

- Ejercicio: Escribe un programa que imprima los números del 1 al 10 y diga si cada uno es par o impar.

Módulo 3: Funciones Básicas

Objetivo: Reutilizar código y organizar programas.

- Temas:
  - Definir funciones (`def`).
  - Parámetros y retorno (`return`).
  - Parámetros por defecto.
- Ejemplo:

```
python
def calcular_area(base, altura=10):
    return base * altura / 2
print(calcular_area(5))      # 25.0
print(calcular_area(5, 8))   # 20.0
```

- Ejercicio: Crea una función que reciba tres números y devuelva el mayor.

Módulo 4: Estructuras de Datos

Objetivo: Almacenar y manipular colecciones de datos.

- Temas:
  - Listas (crear, añadir, eliminar elementos).
  - Tuplas (inmutables).
  - Diccionarios (clave-valor).
  - Conjuntos (elementos únicos).
- Ejemplo:

```
python
frutas = ["manzana", "banana"]
frutas.append("naranja")
print(frutas)  # ['manzana', 'banana', 'naranja']

persona = {"nombre": "Ana", "edad": 25}
print(persona["nombre"])  # Ana
```

- Ejercicio: Crea una lista de 5 nombres y un diccionario con sus edades. Imprime cada nombre con su edad.

Módulo 5: Manejo de Excepciones

Objetivo: Controlar errores sin que el programa falle.

- Temas:
  - `try`, `except`, `finally`.
  - Lanzar excepciones (`raise`).
- Ejemplo:

```
python
try:
    numero = int(input("Dame un número: "))
    print(10 / numero)
except ZeroDivisionError:
    print("No puedes dividir por cero.")
except ValueError:
    print("Debes ingresar un número válido.")
```

- Ejercicio: Haz una función que divida dos números y maneje el error si el divisor es cero.

Módulo 6: Programación Orientada a Objetos (POO)

Objetivo: Crear tus propios tipos de datos.

- Temas:
  - Clases y objetos.
  - Métodos y atributos.
  - Herencia.
- Ejemplo:

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

    def ladrar(self):
        print(f"{self.nombre} dice: ¡Guau!")

mi_perro = Perro("Rex")
mi_perro.ladrar()  # Rex dice: ¡Guau!
```

- Ejercicio: Crea una clase `Coche` con atributos `marca` y `velocidad`, y un método que imprima su velocidad.

Módulo 7: Funciones Avanzadas

Objetivo: Explorar herramientas más potentes.

- Temas:
  - Funciones lambda.
  - `*args` y `**kwargs`.
  - Decoradores.
- Ejemplo:

```
python
duplicar = lambda x: x * 2
print(duplicar(5))  # 10

def suma_todo(*args):
    return sum(args)
print(suma_todo(1, 2, 3, 4))  # 10
```

- Ejercicio: Crea una función con `*args` que multiplique todos los números recibidos.

Módulo 8: Módulos y Bibliotecas

Objetivo: Usar código externo.

- Temas:
  - Importar módulos (`math`, `random`).
  - Instalar bibliotecas con `pip`.
- Ejemplo:

```
python
import random
print(random.randint(1, 10))  # Número aleatorio entre 1 y 10
```

- Ejercicio: Usa `math` para calcular la raíz cuadrada de un número ingresado por el usuario.

Módulo 9: Trabajo con Archivos

Objetivo: Leer y escribir datos permanentes.

- Temas:
  - Leer y escribir archivos de texto.
  - Usar `with` para manejo seguro.
- Ejemplo:

```
python
with open("notas.txt", "w") as archivo:
    archivo.write("Hola desde Python")
```

- Ejercicio: Escribe un programa que guarde una lista de tareas en un archivo y luego las lea.

Módulo 10: Proyecto Final

Objetivo: Integrar todo lo aprendido.

- Proyecto: Gestor de Tareas
  - Usa clases para representar tareas (con atributos como nombre, fecha, estado).
  - Almacena tareas en un archivo.
  - Permite añadir, listar y completar tareas con un menú interactivo.
- Ejemplo parcial:

```
python
class Tarea:
    def __init__(self, nombre):
        self.nombre = nombre
        self.completada = False

tareas = []
while True:
    opcion = input("1. Añadir tarea\n2. Listar\n3. Salir\n> ")
    if opcion == "1":
        nombre = input("Nombre de la tarea: ")
        tareas.append(Tarea(nombre))
    elif opcion == "2":
        for t in tareas:
            print(f"{t.nombre} - {'Hecha' if t.completada else
'Pendiente'}")
    elif opcion == "3":
        break
```