



Python II

UNIDAD I



Material de lectura

Contenido

INTRODUCCIÓN	¡ERROR! MARCADOR NO DEFINIDO.
INTRODUCCIÓN A LA PROGRAMACIÓN ORIENTADA A OBJETOS (POO)	¡ERROR! MARCADOR NO DEFINIDO.
CARACTERÍSTICAS PRINCIPALES DE LA POO Clases y Objetos:	¡ERROR! MARCADOR NO DEFINIDO. ¡Error! Marcador no definido.
VENTAJAS DE LA POO	¡ERROR! MARCADOR NO DEFINIDO.
IMPLEMENTACIÓN DE LA POO EN PYTHON	¡ERROR! MARCADOR NO DEFINIDO.
EJEMPLO DE MODULARIDAD: GESTIÓN DE HOSPITAL	¡ERROR! MARCADOR NO DEFINIDO.
REFERENCIAS BIBLIOGRÁFICAS	7

Implementación en Python usando Visual Studio Code

Cómo ejecutar en Visual Studio Code

1. Abre **Visual Studio Code**.
2. Crea un archivo con extensión .py (por ejemplo, clases_y_objetos.py).
3. Copia y pega el código en el archivo.
4. Guarda el archivo.
5. Abre la terminal en Visual Studio Code (puedes usar el atajo Ctrl + ').
6. Ejecuta el archivo escribiendo:
7. `python clases_y_objetos.py`
8. Observa la salida en la terminal.

Ejemplo 1: Clase simple para representar un vehículo

Este ejemplo muestra cómo definir una clase, crear un objeto y acceder a sus atributos y métodos.

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

    def describir(self):
        return f"{self.color} {self.marca} {self.modelo}"

# Crear un objeto de la clase Vehiculo
mi_auto = Vehiculo("Toyota", "Corolla", "Rojo")

# Acceder a los atributos y métodos
print("Detalles del vehículo:")
print(mi_auto.describir())
```

Explicación:

1. Se define la clase Vehiculo con atributos marca, modelo y color.
2. Se utiliza el método especial __init__ para inicializar los atributos.
3. Se define un método describir que devuelve una descripción del vehículo.
4. Se crea un objeto mi_auto con los valores "Toyota", "Corolla" y "Rojo".

5. Se imprime la descripción del vehículo llamando al método describir.

Ejemplo 2: Clase para gestionar cuentas bancarias

Este ejemplo implementa una clase que simula operaciones básicas de una cuenta bancaria.

✕ Cuenta Bancaria

```
1  class CuentaBancaria:
2      def __init__(self, titular, saldo=0):
3          self.titular = titular
4          self.saldo = saldo
5
6      def depositar(self, cantidad):
7          self.saldo += cantidad
8          print(f"Depósito de {cantidad} realizado. Nuevo saldo: {self.saldo}")
9
10     def retirar(self, cantidad):
11         if cantidad <= self.saldo:
12             self.saldo -= cantidad
13             print(f"Retiro de {cantidad} realizado. Nuevo saldo: {self.saldo}")
14         else:
15             print("Saldo insuficiente.")
16
17     # Crear una cuenta bancaria
18     mi_cuenta = CuentaBancaria("Juan", 500)
19
20     # Realizar operaciones
21     print(f"Titular: {mi_cuenta.titular}")
22     mi_cuenta.depositar(200)
23     mi_cuenta.retirar(300)
24     mi_cuenta.retirar(500)
25
```

Explicación:

1. La clase CuentaBancaria tiene atributos titular y saldo, con un saldo inicial opcional.
2. Los métodos depositar y retirar permiten realizar operaciones con validación de saldo.
3. Se crea un objeto mi_cuenta con el titular "Juan" y un saldo inicial de 500.
4. Se realizan operaciones de depósito y retiro, mostrando mensajes según el caso.

Ejemplo 3: Herencia en POO

Este ejemplo muestra cómo una clase hija hereda atributos y métodos de una clase padre.

✕ Persona Y Estudiante

```
1  class Persona:
2      def __init__(self, nombre, edad):
3          self.nombre = nombre
4          self.edad = edad
5
6      def presentarse(self):
7          print(f"Hola, soy {self.nombre} y tengo {self.edad} años.")
8
9  class Estudiante(Persona):
10     def __init__(self, nombre, edad, carrera):
11         super().__init__(nombre, edad)
12         self.carrera = carrera
13
14     def presentarse(self):
15         super().presentarse()
16         print(f"Estoy estudiando {self.carrera}.")
17
18     # Crear un objeto de la clase Estudiante
19     estudiante1 = Estudiante("Ana", 20, "Ingeniería Informática")
20     estudiante1.presentarse()
21
```

Explicación:

1. La clase Persona define atributos comunes y un método para presentarse.
2. La clase Estudiante hereda de Persona y añade un atributo carrera.
3. El método presentarse en Estudiante amplía el comportamiento del método en la clase padre usando super().
4. Se crea un objeto estudiante1 y se llama a su método presentarse.

Referencias Bibliográficas

Aquí tienes las referencias bibliográficas en formato APA para la información presentada:

- Ceder, N. (2018). *The Quick Python Book* (3rd ed.). Manning Publications.
Una guía práctica para aprender Python, con un enfoque claro en la programación orientada a objetos.
- Downey, A. B. (2015). *Think Python: How to Think Like a Computer Scientist* (2nd ed.). O'Reilly Media.
Proporciona una introducción detallada a Python, incluyendo los conceptos fundamentales de la programación orientada a objetos.
- Griffiths, D., & Griffiths, P. (2020). *Head First Python* (2nd ed.). O'Reilly Media.
Una guía visual para aprender Python, que incluye ejemplos prácticos y explicaciones sobre el paradigma de programación orientado a objetos.
- Guttag, J. V. (2016). *Introduction to Computation and Programming Using Python* (2nd ed.). MIT Press.
Cubre los fundamentos de la programación con Python y una sólida introducción a la programación orientada a objetos.
- Lutz, M. (2013). *Learning Python* (5th ed.). O'Reilly Media.
Un recurso integral para aprender Python, desde los fundamentos básicos hasta temas avanzados como la programación orientada a objetos.
- Python Software Foundation. (2023). *The Python Tutorial*. Recuperado de <https://docs.python.org/3/tutorial/>
Documentación oficial de Python, que incluye una sección sobre clases y objetos con ejemplos prácticos.
- Eckel, B. (2006). *Thinking in Java* (4th ed.). Prentice Hall.
Aunque centrado en Java, este libro es ideal para entender los principios de la programación orientada a objetos, aplicables a otros lenguajes como Python.