

Working Time 😎

L7: Conceptos básicos de orientación a objeto en Python

Es hora de que pongas en práctica todo lo aprendido. 😊

Este apartado tiene el objetivo de ayudarte a seguir potenciando tus habilidades, por lo que a continuación encontrarás diferentes desafíos que podrás resolver de forma independiente y a tu ritmo.

Más adelante conseguirás las soluciones para que valides tus respuestas y puedas monitorear tu progreso. 😊

¡Manos a la obra!

1. Desafío 🎯

Crea una clase `CuentaBancaria` que represente una cuenta bancaria con los siguientes atributos y métodos:

- **Atributos:**
 - `titular` (nombre del titular)
 - `saldo` (monto en la cuenta)
- **Métodos:**
 - `depositar(cantidad)`: permite agregar dinero al saldo.
 - `retirar(cantidad)`: permite retirar dinero si hay suficiente saldo.
 - `mostrar_saldo()`: muestra el saldo actual.

2. Plus +

Agrega un método `transferir(cantidad, otra_cuenta)` que permita transferir dinero de una cuenta a otra, siempre que el saldo sea suficiente.

3. ¿Dónde se lleva a cabo? 🏠

En un **notebook Jupyter (.ipynb)** utilizando **Visual Studio Code**, **Google Colab**, o cualquier entorno que te permita **ejecutar celdas y visualizar las salidas de forma interactiva**.

A diferencia de un archivo `.py`, el formato `.ipynb` te permitirá observar el **código, su explicación y sus resultados** en un mismo lugar, ideal para el análisis de datos y la práctica de POO.

4. Tiempo de dedicación

1 Hora.

5. Recursos

- Documentación oficial de clases y objetos en Python
- Video introductorio a la Programación Orientada a Objetos (POO) en Python.

6. Condición

Esta práctica o ejercitación no requiere ser entregada y/o evaluada por el mentor.

No obstante, puedes compartir tus resultados con el resto de los bootcampers y construir conocimiento en conjunto.

7. Resolución del ejercicio

```
# Definición de la clase CuentaBancaria

class CuentaBancaria:

    def __init__(self, titular, saldo=0):
        self.titular = titular
        self.saldo = saldo

    def depositar(self, cantidad):
        """Agrega dinero al saldo actual."""
        self.saldo += cantidad
        print(f"Has depositado {cantidad}. Saldo actual: {self.saldo}")

    def retirar(self, cantidad):
        """Retira dinero si hay saldo suficiente."""
        if cantidad <= self.saldo:
            self.saldo -= cantidad
            print(f"Has retirado {cantidad}. Saldo actual: {self.saldo}")
        else:
            print("Saldo insuficiente para realizar el retiro.")

    def mostrar_saldo(self):
        """Muestra el saldo actual."""
        print(f"El saldo actual de la cuenta de {self.titular} es: {self.saldo}")

    def transferir(self, cantidad, otra_cuenta):
        pass
```

```
"""Transfiere dinero a otra cuenta si hay saldo suficiente."""
if cantidad <= self.saldo:
    self.retirar(cantidad)
    otra_cuenta.depositar(cantidad)
    print(f"Has transferido {cantidad} a la cuenta de
{otra_cuenta.titular}")
else:
    print("Saldo insuficiente para realizar la transferencia.")

# Creación de instancias de CuentaBancaria
cuenta1 = CuentaBancaria("Carlos", 1000)
cuenta2 = CuentaBancaria("Ana", 500)

# Operaciones en cuenta1
cuenta1.mostrar_saldo()
cuenta1.depositar(200)
cuenta1.retirar(150)
cuenta1.mostrar_saldo()

# Transferencia entre cuentas
cuenta1.transferir(300, cuenta2)

# Mostrar saldo de ambas cuentas
cuenta1.mostrar_saldo()
```

```
cuenta2.mostrar_saldo()
```