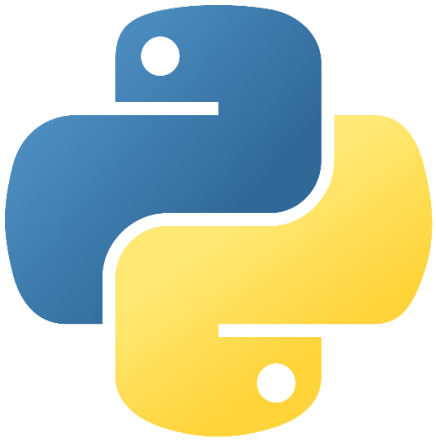

EJERCICIOS DE PYTHON IV

Clases y Objetos

- Ptogramación orientada a objetos (POO)
- Clases y Objetos
- Atributos instancia
- Métodos
- Método __init__()
- Método __str__()
- Método __de__()
- Colaboración de clases
- Atributos de clase
- Composición

Codo a Codo

2024

1) **Ejercicio 1:** Implementar una clase llamada Estudiante que tendrá como atributos (variables) su nombre, su apellido, dni y dos métodos (funciones), uno de dichos métodos inicializará los atributos y el siguiente método los mostrará en pantalla. Definir dos objetos de la clase Estudiante e incorporar una variable de clase (piernas).

```
class Estudiante: # Creamos la clase (sustantivo 1° letra mayúscula)

    piernas = 2 # Atributo / variable de clase

    def inicializar(self, nombre, apellido, dni): # Método constructor
        self.nombre = nombre # Atributo de instancia
        self.apellido = apellido # Atributo de instancia
        self.dni = dni # Atributo de instancia

    def imprimir(self): # Método para mostrar datos
        print(f'Apellido y nombre: {self.apellido.upper()},
{self.nombre}\nDNI: {self.dni}')

# Programa principal
estud1 = Estudiante() # Creamos un objeto basado en la clase
Estudiante
estud1.inicializar("Juan Pablo", "Nardone", 12345678) # Llamamos al
constructor con un nombre
estud1.imprimir() # Mostramos los datos

estud2 = Estudiante()
estud2.inicializar("Luciana", "Pérez", 31234567)
estud2.imprimir()

print(f'Los estudiantes tienen {Estudiante.piernas} piernas')
print(f'{estud1.nombre} tiene {estud1.piernas} piernas')
print(f'{estud2.nombre} tiene {estud2.piernas} piernas')

print()
estud1.piernas = 4
print(f'{estud1.nombre} tiene {estud1.piernas} piernas') # 4 piernas
estud2.nombre = "Fernanda"
print(f'{estud2.nombre} tiene {estud2.piernas} piernas') # 2 piernas

estud1.edad = 37 # Podemos agregar un atributo de instancia
print(estud1.edad)
print(estud2.edad) # Error, no tiene ese atributo
```

2) **Ejercicio 2:** Implementar una nueva clase llamada Estudiante. Esta clase tendrá como atributos su nombre y su nota. Definir los métodos para inicializar sus atributos, imprimirlos y mostrar un mensaje que indique: “Promocionó” (nota >= 7), “Rinde final” (nota >= 4) o “Desaprobó”.
Definir tres objetos de la clase Alumno, cada uno con una condición de aprobación distinta.

```
class Estudiante:

    def inicializar(self, nombreCompleto, nota): # Método constructor
        self.nombreCompleto = nombreCompleto # Atributo de instancia
        self.nota = nota # Atributo de instancia

    def imprimir(self): # Método para mostrar datos
        print(f'Nombre completo: {self.nombreCompleto.title()} - Nota:
{self.nota}')
        if self.nota >= 7:
            print("Promocionó")
        elif self.nota >= 4:
            print("Rinde final")
        else:
            print("Desaprobó")
        print()

# Programa principal

alumno1 = Estudiante() # Creamos el objeto
alumno1.inicializar("Juan Serrano", 8)
alumno1.imprimir()

alumno2 = Estudiante() # Creamos el objeto
alumno2.inicializar("Luisa López", 4)
alumno2.imprimir()

alumno3 = Estudiante() # Creamos el objeto
alumno3.inicializar("Diego Fernández", 2)
alumno3.imprimir()
```

3) **Ejercicio 3:** Crear una clase que represente una Materia de la universidad. Definir como atributos su nombre, carrera, duración en meses y un atributo de clase booleano para definir que todas las materias no son promocionables. Desarrollar un método __init__ para incializarlos. Crear un método para imprimir los datos del objeto, luego sustituirlo por el método __str__(). Crear dos objetos. Eliminar uno de ellos a través del método __del__().

```
class Materia:
    promo = False # Atributo de clase

    # Nuevo método constructor
    def __init__(self, nombre, carrera, duracion): # Permite
instanciar y agregar valores a los atributos
        self.nombre = nombre
        self.carrera = carrera
        self.duracion = duracion

    def imprimir(self):
```

```
        print(f'Materia: {self.nombre.title()}\nCarrera:
{self.carrera.upper()}\nDuración: {self.duracion}
meses\nPromocionable: {self.promo}')

    def __str__(self): # Reemplaza al método imprimir
        return f'Materia: {self.nombre.title()}\nCarrera:
{self.carrera.upper()}\nDuración: {self.duracion}
meses\nPromocionable: {self.promo}'

    def __del__(self): # Elimina el objeto
        print(f'{self.nombre} ha sido eliminada')

# Programa principal
materia1 = Materia("introd. a python", "ing. en informática", 4)
# materia1.imprimir()
print(materia1)
print()
materia2 = Materia("inteligencia artificial", "ing. en informática",
8)
print(materia2)
print()
# del materia2
```

4) **Ejercicio Integrador:** Implementar una clase llamada Empleado, que posee un atributo de clase (nro_empleados) que lleva la cuenta de los objetos instanciados.
Cada objeto de la clase Empleado posee un legajo, nombre completo y un sueldo.
Definir métodos para inicializar sus atributos, definir su categoría (variable de clase), procesar su eliminación de la memoria y para mostrar un texto con la categoría asignada.
La categoría es "Full Time" para los legajos comenzados en "F", "Part time" para los legajos comenzados en "P" y "Contratado" para los comenzados en "C", para el resto la categoría es vacía.
En el programa principal instanciar distintos objetos de la clase Empleado y mostrar sus características encolumnadas. Al salir del programa eliminarlos de la memoria.

```
class Empleado:  # Creamos la clase
    nro_empleados = 0 # Cantidad de empleados
    categoria = ""

    # Constructor
    def __init__(self, legajo, nombreCompleto, sueldo):
        self.legajo = legajo
        self.nombreCompleto = nombreCompleto
        self.sueldo = sueldo
        Empleado.nro_empleados += 1 # Agregamos un empleado

    # Mostrar datos del objeto
    def __str__(self):
        inicial_legajo = self.legajo[0] # Primera inicial del legajo
```

```
        if inicial_legajo == "F":
            self.categoria = "Full Time"
        elif inicial_legajo == "P":
            self.categoria = "Part time"
        elif inicial_legajo == "C":
            self.categoria = "Contratado"

        cadena =
f'{self.legajo}\t{self.nombreCompleto}\t{self.sueldo}\t{self.categoria
}'
        return cadena

    def titulos():
        print(f'Legajo\tNombre Completo\tSueldo\tCategoria')

    # Damos de baja al producto
    def __del__(self):
        Empleado.nro_empleados -= 1  # Restamos un empleado
        print(f'{self.nombreCompleto} ha sido dado de baja - Restan:
{Empleado.nro_empleados}')

# Programa principal
emp1 = Empleado("F1000", "Juan Pérez", 15000)
emp2 = Empleado("P1001", "Ana Gómez", 24000)
emp3 = Empleado("C1002", "Luciano García", 13000)
emp4 = Empleado("H1003", "Marcela Priore", 22000)

Empleado.titulos()
print(emp1)
print(emp2)
print(emp3)
print(emp4)
print()
input("Pulse Enter para salir: ")
print()
```

5) **Ejercicio 5 (colaboración de clases):** El comedor de la universidad tiene un sistema de beneficios a través del cual los alumnos pueden sumar puntos o canjearlos por menús. El comedor necesita al final del día un reporte de la cantidad de puntos que sus alumnos poseen. Crear 3 alumnos cuyos atributos son su nombre y la cantidad de puntos que poseen, inicializado en 0. Crear un método que simule las operaciones de ingresar puntos y de canjear puntos.

```
# ------------------------ Clase Alumno -------------------------
class Alumno:

    def __init__(self,nombre):
        self.nombre = nombre
        self.puntos = 0
```

```
    def ingresar_puntos(self,puntos):
        self.puntos = self.puntos + puntos

    def canjear_puntos(self,puntos):
        self.puntos = self.puntos - puntos

    def retornar_puntos(self):
        return self.puntos

    def __str__(self):
        return f'{self.nombre}\t\t{self.puntos} pts.'

# ------------------------ Clase Comedor ------------------------
class Comedor:

    def __init__(self, sede):
        self.sede = sede
        self.alumno1=Alumno("Juan")
        self.alumno2=Alumno("Luciana")
        self.alumno3=Alumno("Sofía")

    def __str__(self):
        return f'{"-"*30}\n{self.sede}\n{"-"*30}\n'

    def operar(self): # simula movimientos
        self.alumno1.ingresar_puntos(100)
        self.alumno2.ingresar_puntos(150)
        self.alumno3.ingresar_puntos(200)
        self.alumno3.canjear_puntos(150)
        self.alumno1.ingresar_puntos(300)
        self.alumno1.canjear_puntos(75)

    def puntos_totales(self):
        print("Detalle de puntos del día:")
        print(self.alumno1)
        print(self.alumno2)
        print(self.alumno3)
        print("-"*30)
        total = self.alumno1.retornar_puntos() +
self.alumno2.retornar_puntos() + self.alumno3.retornar_puntos()
        print(f'Total\t\t{total} pts.')

# ------------------------ Programa principal ----------------------
--
comedor1 = Comedor("El comedor de la Universidad")
print(comedor1)
comedor1.operar()
comedor1.puntos_totales()
```

6) **Ejercicio 6 (colaboración de clases):** Se debe implementar un programa que permita jugar a un juego de dados. El juego sigue las siguientes reglas:
   - Se tiran tres dados al mismo tiempo.
   - Si los tres dados muestran el mismo valor, el jugador gana.
   - Si no se cumplen las condiciones del punto 2, el jugador pierde.

   El programa debe constar de las siguientes partes:
   a) Una clase llamada **Dado** que debe tener los siguientes métodos:
      - **tirar():** Este método debe generar un valor aleatorio entre 1 y 6 y asignarlo como el valor del dado.
      - **imprimir():** Este método debe imprimir el valor actual del dado.
      - **retornar_valor():** Este método debe devolver el valor actual del dado.
   b) Una clase llamada **JuegoDeDados** que debe tener los siguientes métodos:
      - **__init__():** Este método debe inicializar tres instancias de la clase Dado como atributos (dado1, dado2 y dado3).
      - **jugar():** Este método debe simular una jugada del juego. Para ello, debe tirar los tres dados, imprimir los resultados y determinar si el jugador ganó o perdió según las reglas mencionadas anteriormente.
      - **simular_jugadas(cantidad):** Este método debe simular una cantidad especificada de jugadas del juego. Debe llamar al método **jugar()** la cantidad de veces especificada y mostrar los resultados.

   Resultado esperado: El programa debe simular 20 jugadas del juego de dados y mostrar los resultados de cada jugada.

```python
import random

# ----------------- Clase Dado -----------------
class Dado:

    def tirar(self):
        self.valor = random.randint(1,6)

    def imprimir(self):
        print(f'{self.valor}', end=' ')

    def retornar_valor(self):
        return self.valor

# ----------------- Clase Juego De Dados -----------------
class JuegoDeDados:

    def __init__(self):
        self.dado1=Dado()
        self.dado2=Dado()
        self.dado3=Dado()

    def jugar(self):
        self.dado1.tirar()
        self.dado1.imprimir()
```

```
        self.dado2.tirar()
        self.dado2.imprimir()
        self.dado3.tirar()
        self.dado3.imprimir()

        valor_dado1 = self.dado1.retornar_valor()
        valor_dado2 = self.dado2.retornar_valor()
        valor_dado3 = self.dado3.retornar_valor()
        if valor_dado1 == valor_dado2 and valor_dado1 == valor_dado3:
            print("Ganó")
        else:
            print("Perdió")

    def simular_jugadas(self, cantidad):
        self.cantidad = cantidad
        i = 1
        while i <= cantidad:
            self.jugar()
            i += 1

# ----------------- Bloque principal -----------------
juego_dados=JuegoDeDados()
#juego_dados.jugar()
juego_dados.simular_jugadas(20)
```

7) **Ejercicio 7 (variables de clase):** Se debe implementar una clase **Alumno** que almacene la información de un estudiante, incluyendo un código de alumno, un nombre y su nota final en una materia. Además, la clase **Alumno** debe mantener dos listas de variables de clase, una de ellas llamada aprobados, que almacenará los códigos de los alumnos aprobados (nota mayor o igual a 4) y otra llamada reprobados que almacenará los códigos de los alumnos reprobados (nota menor a 4).
La clase Alumno debe tener los siguientes métodos:

- **__init__(self, codigo, nombre, nota):** Este método inicializa un objeto de tipo Alumno con el código, nombre y nota proporcionados. Además, invoca el método definir_estado() para determinar si el alumno está aprobado o reprobado.
- **__str__(self):** Este método devuelve una representación en forma de cadena del objeto Alumno, que incluye el código, nombre y nota.
- **definir_estado(self):** Este método determina si el alumno está aprobado o reprobado, y actualiza las listas aprobados y reprobados en consecuencia.

A continuación, crear cuatro objetos de tipo Alumno con notas distintas y almacenarlos en una lista. Posteriormente, mostrar los datos de los 4 alumnos y finalmente mostrar la lista de los códigos de los alumnos aprobados y además los códigos de los alumnos reprobados, uno debajo del otro.

```
class Alumno:
    aprobados = []
    reprobados = []

    def __init__(self, codigo, nombre, nota):
```

```
        self.codigo = codigo
        self.nombre = nombre
        self.nota = nota
        Alumno.definir_estado(self)

    def __str__(self):
        return f'{self.codigo}\t{self.nombre}\t{self.nota}'

    def definir_estado(self):
        if self.nota >= 4:
            Alumno.aprobados.append(self.codigo)
        else:
            Alumno.reprobados.append(self.codigo)

# Programa principal
alumnos = [ # Lista de objetos
    Alumno(1, "Juan", 5),
    Alumno(2, "Ana", 2),
    Alumno(3, "Diego", 9),
    Alumno(4, "Pedro", 3)
]

print("Legajo\tNombre\tNota")
for alumno in alumnos:
    print(alumno)

#Dos formas de mostrar la información
print(f'Aprobados: {Alumno.aprobados}')

print("Reprobados:")
for i in range(len(Alumno.reprobados)):
    print(Alumno.reprobados[i])
```

8) **Ejercicio 8 (composición):** Se debe implementar una clase **Materia** que gestione los datos de las materias de una universidad. Cada instancia de Materia debe tener los siguientes atributos: código, nombre y duración en meses.
Además, la clase **Materia** debe implementar el método **__str__** para mostrar sus datos. La duración en meses debe ser representada de la siguiente manera: "Cuatrimestral" (4 meses), "Anual" (8 meses) o "A confirmar" (en el resto de los casos).
Luego, se debe definir la clase **Carrera**, que administra una lista de materias. Cada materia en la lista es un objeto de la clase Materia.
La clase **Carrera** debe tener los siguientes métodos:

- **__init__(self, nombre, materias=[]):** Este método inicializa una carrera con un nombre y una lista opcional de materias. Por defecto, la lista de materias está vacía.
- **agregar(self, mat):** Este método permite agregar una materia (representada por un objeto de la clase Materia) a la lista de materias de la carrera.
- **__str__(self):** Este método devuelve una representación en forma de cadena de la carrera, incluyendo su nombre y el listado de materias.

A continuación, crear una Materia e instanciar una carrera con esa materia. Luego agregar se agregan materias adicionales e imprimir el listado de materias de la carrera.

```
class Materia:

    # Constructor de clase
    def __init__(self, codigo, nombre, duracion):
        self.codigo = codigo
        self.nombre = nombre
        self.duracion = duracion
        print(f'Se ha creado la materia: {self.codigo}-{self.nombre}')

    def __str__(self):
        if self.duracion == 4:
            durac = "Cuatrimestral"
        elif self.duracion == 8:
            durac = "Anual"
        else:
            durac = "A confirmar"

        return f'{self.codigo}: {self.nombre} ({durac})'

class Carrera:

    materias = []  # Esta lista contendrá objetos de la clase Materia

    def __init__(self, nombre, materias=[]):
        self.nombre = nombre
        self.materias = materias

    def agregar(self, mat):  # mat será un objeto Materia
        self.materias.append(mat)

    def __str__(self):
        linea = "="*60
        cad = f'\n{linea}\n{self.nombre.upper()} - Listado de
materias:\n'
        for i in range(len(self.materias)):
            cad = cad + str(self.materias[i]) + "\n"
        cad = cad + linea + "\n"
        return cad

#Programa principal

materia1 = Materia("A1000", "Algoritmos I", 4) # Creamos nuestra
primera materia
print(materia1)
```

```
carrera1 = Carrera("Licenciatura en Python", [materia1])  #
Instanciamos la carrera con una materia
print(carrera1) # Imprimimos la carrera con una materia

# Agregamos 3 materias
carrera1.agregar(Materia("A1001", "Python I", 8))  # Añadimos otra
materia
carrera1.agregar(Materia("A1002", "Objetos I", 0))  # Añadimos otra
materia
carrera1.agregar(Materia("A1003", "Inteligencia Artificial", 4))  #
Añadimos otra materia

print(carrera1) # Imprimimos la carrera con sus materias
```