04.1. Ejercicios POO

May 5, 2025

1 Programación Orientada a Objetos

1.1 Python

1.1.1 Ejemplo.

Cree una clase llamada Persona, use la función *___init___()* para asignar valores para nombre y edad:

[ ]:
```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)

print(p1.name)
print(p1.age)
```

```
John
36
```

[ ]:
```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)

print(p1)
```

```
<__main__.Person object at 0x7d8759ac9240>
```

[ ]:
```
class Person:
  def __init__(mysillyobject, name, age):
    mysillyobject.name = name
    mysillyobject.age = age

  def myfunc(abc):
    print("Hello my name is " + abc.name)
```

```
p1 = Person("John", 36)
p1.myfunc()
```

```
Hello my name is John
```

[ ]:
```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def __str__(self):
    return f"{self.name} is {self.age} years old."

p1 = Person("John", 36)

print(p1)
```

```
John is 36 years old.
```

1.1.2 Ejemplo.

Crear una clase que reciba la parte real y la parte imaginaria de un número complejo. Debe devolver la representación del número.

$$2 \pm 3i$$

$$-4 \pm 5i$$

[1]:
```
class complex_number:
    def __init__(self, real, imag):
        self.real = real
        self.imag = imag

    def __str__(self):
        return f'{self.real} ± {self.imag}i'

num1 = complex_number(2,3)
print(num1)
num2 = complex_number(-4,5)
print(num2)
```

```
2 ± 3i
-4 ± 5i
```

1.1.3 Ejemplo.

Inserte una función que imprima un saludo y ejecútelo en el objeto p1:

[2]:
```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name + " and I'm " + str(self.age) + "␣
 ↪years old.")

p1 = Person("John", 36)
p1.myfunc()
```

```
Hello my name is John and I'm 36 years old.
```

1.1.4 Ejemplo.

Registro de perros

[3]:
```
class Perro:
    def __init__(self, name, raza, talla, peso):
        self.name = name
        self.raza = raza
        self.talla = talla
        self.peso = peso

    def obtener_peso(self):
        return self.peso

miPerro = Perro("Firulais", "mestizo", "Mediano", 15)
print(f'Mi perro pesa {miPerro.obtener_peso()} kilos')
```

```
Mi perro pesa 15 kilos
```

1.1.5 Ejemplo.

Registro de perros con promedio de peso

[11]:
```
class Perro:
    # Peso promedio
    peso = 20

    def __init__(self, name, raza, talla, peso):
        self.name = name
        self.raza = raza
        self.talla = talla
        self.peso = peso

    @classmethod # Método de clase: Acceso a los valores de clase en lugar de␣
 ↪instancia
```

```
    def obtener_peso(self):
        return self.peso

miPerro = Perro("Firulais", "mestizo", "Mediano", 12)
print(f'Mi perro pesa {miPerro.peso} kilos')
print(f'El peso promedio de los perros es {miPerro.obtener_peso()} kilos')
```

```
Mi perro pesa 12 kilos
El peso promedio de los perros es 20 kilos
```

`@classmethod` convierte un método para que trabaje con la clase en sí (y sus atributos de clase) y no con instancias específicas.

1.1.6 Ejemplo.

Calculadora

[19]:
```
class calcu:

    @staticmethod
    def sumar(num1, num2):
        return num1 + num2

    @staticmethod
    def resta(num1, num2):
        return num1 - num2

    @staticmethod
    def multiplicacion(num1, num2):
        return num1 * num2

    @staticmethod
    def division(num1, num2):
        return num1 / num2

print(calcu.sumar(4,5))
print(calcu.resta(8,4))
print(calcu.multiplicacion(2,9))
print(calcu.division(6,3))
```

```
9
4
18
2.0
```

[20]:
```
x = calcu.sumar(4,5)
print(x)
```

```
9
```

1.1.7 Ejercicio.

Realizar una clase que reciba un par de argumentos numéricos y realice las operaciones aritméticas básicas

[21]:
```
class calculadora:
    def __init__(self, num1, num2):
        self.num1 = num1
        self.num2 = num2

    def suma(self):
        return self.num1 + self.num2

    def resta(self):
        return self.num1 - self.num2

    def multiplicacion(self):
        return self.num1 * self.num2

    def division(self):
        return self.num1 / self.num2

operacion = calculadora(5,9)
print(operacion.num1)
print(operacion.num2)
print(operacion.suma())
print(operacion.resta())
print(operacion.multiplicacion())
print(operacion.division())
```

```
5
9
14
-4
45
0.5555555555555556
```

1.1.8 Ejercicio.

Realice un clase *Circulo* que reciba las coordenadas (x, y) donde se ubica el centro y el radio r. La clase debe tener métodos que determinen el perímetro, área, cuadrante donde se ubica el cirulo en el plano cartesiano.

[22]:
```
import math

class Circulo:
    def __init__(self, x, y, r):
        self.x = x
        self.y = y
```

```
        self.r = r

    def perimetro(self):
        return 2 * math.pi * self.r

    def area(self):
        return math.pi * self.r**2

    def cuadrante(self):
        if self.x > 0 and self.y > 0:
            return "I"
        elif self.x < 0 and self.y > 0:
            return "II"
        elif self.x > 0 and self.y < 0:
            return "IV"
        else:
            return "III"

c1 = Circulo(-6,-7,2)
print(c1.perimetro())
print(c1.area())
print(c1.cuadrante())
```

```
12.566370614359172
12.566370614359172
III
```

1.1.9 Ejercicio.

Realice una clase que reciba la parte real y la parte imaginaria de un número complejo. Debe tener métodos que determine el módulo del número complejo, la suma y resta de dos números complejos.

$$|2 + -3i| = \sqrt{2^2 + 3^2} = \sqrt{13}$$
$$(2 + -3i) + (4 + -5i) = 6 + -8i$$
$$(2 + -3i) - (4 + -5i) = -2 + -2i$$

[23]:
```
import math

class ComplexNumber:
    def __init__(self, r, i):
        self.r = r
        self.i = i

    def modulo(self):
        return math.sqrt(self.r**2 + self.i**2)

    def __str__(self):
```

```
        return f'{self.r} ± {self.i}i'

    # Métodos Mágicos (Dunder -> double under (score))
    def __add__(self, other):
        return ComplexNumber(self.r + other.r, self.i + other.i)

    def __sub__(self, other):
        return ComplexNumber(self.r - other.r, self.i - other.i)

# Ejemplo
c1 = ComplexNumber(2,3)
print(c1)
print(c1.modulo())

c2 = ComplexNumber(4,5)
print(c2)
print(c2.modulo())

c3r = c1.r + c2.r
c3i = c1.i + c2.i
c3 = ComplexNumber(c3r, c3i)
print(c3)

c4 = c1 + c2
print(c4)

c5 = c1 - c2
print(c5)
```

```
2 ± 3i
3.605551275463989
4 ± 5i
6.4031242374328485
6 ± 8i
6 ± 8i
-2 ± -2i
```

En Python, los métodos ___*add*___ y ___*sub*___ son métodos especiales que permiten definir el comportamiento de los operadores de suma (+) y resta (-) para los objetos de una clase personalizada. Estos métodos se denominan "**métodos mágicos**" o "**métodos dunder**" (double underscore, por sus siglas en inglés).

1.2 Lista de ejercicios

1.2.1 Ejercicio 1: Clase *Persona*

Crea una clase *Persona* con atributos nombre, edad y métodos para mostrar esta información y calcular si la persona es mayor de edad.

[2]:
```python
class Persona:
    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad

    def mostrar_info(self):
        return(f'Nombre: {self.nombre}, Edad: {self.edad}')

    def mayor_de_edad(self):
        if self.edad >= 18:
            return True
        else:
            return False

# Ejemplo
persona1 = Persona("Juan", 23)
print(persona1.mostrar_info())
print(persona1.mayor_de_edad())
```

```
Nombre: Juan, Edad: 23
True
```

1.2.2 Ejercicio 2: Clase *Rectángulo*

Crea una clase *Rectángulo* con atributos largo y ancho. Incluye métodos para calcular el área y el perímetro del rectángulo.

[ ]:
```python
class Rectangulo:
    def __init__(self, largo, ancho):
        self.largo = largo
        self.ancho = ancho

    def perimetro(self):
        return 2 * (self.largo + self.ancho)

    def area(self):
        return self.largo * self.ancho


# Ejemplo
r1 = Rectangulo(9, 5)
print(r1.perimetro())
print(r1.area())
```

```
28
45
```

1.2.3 Ejercicio 3: Clase *Círculo*

Crea una clase *Círculo* con un atributo radio. Añade métodos para calcular el área y la circunferencia del círculo.

[31]:
```
import math

class Circulo:
    def __init__(self, r):
        self.r = r

    def area(self):
        return math.pi * self.r**2

    def circunferencia(self):
        return 2*math.pi*self.r

# Ejemplo
c1 = Circulo(4)
print(f'Area = {c1.area()} u²')
print(f'Perimetro = {c1.circunferencia()} u')
```

```
Area = 50.26548245743669 u²
Perimetro = 25.132741228718345 u
```

1.2.4 Ejercicio 4: Clase *CuentaBancaria*

Crea una clase *CuentaBancaria* con atributos titular y saldo. Incluye métodos para depositar, retirar y mostrar el saldo.

[ ]:
```
class CuentaBancaria:
    def __init__(self, titular, saldo=0):
        self.titular = titular
        self.saldo = saldo

    def depositar(self, monto):
        self.saldo += monto

    def retirar(self, monto):
        if monto < self.saldo:
            self.saldo = self.saldo - monto # self.saldo -= monto
        else:
            print("Saldo insuficiente")

    def mostrarSaldo(self):
        print(f'Saldo de {self.titular}: $ {self.saldo}')

# Ejemplo
cliente1 = CuentaBancaria("Juan Perez", 4500)
```

```
cliente1.depositar(430)
cliente1.mostrarSaldo()
cliente1.retirar(10000)
```

```
Saldo de Juan Perez: $ 4930
Saldo insuficiente
```

1.2.5 Ejercicio 5: Clase *Estudiante*

Crea una clase Estudiante que herede de Persona. Añade un atributo promedio y un método para determinar si el estudiante está aprobado (promedio >= 6).

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

    def mostrarInfo(self):
        print(f'Nombre: {self.nombre}, Edad: {self.edad}')

class Estudiante(Persona):
    def __init__(self, nombre, edad, promedio):
        super().__init__(nombre, edad)
        self.promedio = promedio

    def aprobado(self):
        if self.promedio >= 6:
            return True
        else:
            return False

# Ejemplo
estudiante1 = Estudiante("Maria", 20, 7.5)
print(estudiante1.aprobado())
estudiante1.mostrarInfo()
```

```
True
Nombre: Maria, Edad: 20
```

1.2.6 Ejercicio 6: Clase *Libro*

Crea una clase Libro con atributos título, autor y año. Incluye un método para mostrar la información del libro y otro para determinar si es un libro antiguo (año < 2000).

[40]:
```
class Libro:
    def __init__(self, titulo, autor, anio):
        self.titulo = titulo
        self.autor = autor
        self.anio = anio
```

```
    def info(self):
        return f'{self.autor}; {self.titulo}; {self.anio}'

    def esAntiguo(self):
        return True if self.anio < 2000 else False

libro1 = Libro("Algebra", "Baldor", 1980)
print(libro1.info())
print(libro1.esAntiguo())
```

```
Baldor; Algebra; 1980
True
```

1.2.7 Ejercicio 7: Clase *Vehículo*

Crea una clase Vehículo con atributos marca, modelo y año. Añade un método para mostrar la información del vehículo y otro para determinar si es un vehículo clásico (año < 1980).

[ ]:

1.2.8 Ejercicio 8: Clase *Empleado*

Crea una clase Empleado con atributos nombre, salario y años_de_experiencia. Añade un método para calcular un aumento salarial basado en los años de experiencia (5%).

[43]:
```
class Empleado:
    def __init__(self, nombre, salario, exp):
        self.nombre = nombre
        self.salario = salario
        self.exp = exp

    def __str__(self):
        return f'Nombre: {self.nombre}, Salario: {self.salario}, Años de␣
 ↪Experiencia: {self.exp}'

    def mostrarInfo(self):
        print(f'Nombre: {self.nombre}, Salario: {self.salario}, Años de␣
 ↪Experiencia: {self.exp}')

    def aumento(self):
        aumento = 0.05*self.exp
        self.salario += self.salario*aumento

# Ejemplo
empleado1 = Empleado("Carlos", 15000, 10)
print(empleado1)
empleado1.mostrarInfo()
```

```
empleado1.aumento()
print(f'Nuevo salario: {empleado1.salario}')
```

```
Nombre: Carlos, Salario: 15000, Años de Experiencia: 10
Nombre: Carlos, Salario: 15000, Años de Experiencia: 10
Nuevo salario: 22500.0
```

1.2.9 Ejercicio 9: Clase *CuentaDeAhorro*

Crea una clase CuentaDeAhorro que herede de CuentaBancaria. Añade un atributo interés y un método para aplicar el interés al saldo.

[ ]:

1.2.10 Ejercicio 10: Clase *Empresa*

Crea una clase Empresa con atributos nombre y empleados (una lista de objetos de la clase Empleado). Incluye métodos para agregar empleados, eliminar empleados y calcular el salario total pagado por la empresa.

[ ]: