POO con Python - Tema 1 (extendido): Introduccion a la POO

1. Proposito y contexto

Este tema presenta los fundamentos conceptuales de la Programacion Orientada a Objetos (POO) y su aplicacion en Python. Se abordan los principios de abstraccion, encapsulacion, herencia y polimorfismo, con ejemplos tecnicos y UML.

2. Conceptos basicos

Clase: plantilla que define estructura y comportamiento. Objeto: instancia concreta de una clase. Atributos: datos asociados a una instancia o a la clase. Metodos: funciones asociadas a una clase u objeto.

3. Principios fundamentales

3.1 Abstraccion

Proceso de modelar entidades relevantes y sus comportamientos esenciales, ignorando detalles accidentales.

3.2 Encapsulacion

Organiza datos y comportamiento dentro de una clase, exponiendo una interfaz publica y ocultando detalles internos.

3.3 Herencia

Permite derivar nuevas clases (subclases) a partir de otras (superclases), reutilizando y especializando comportamiento.

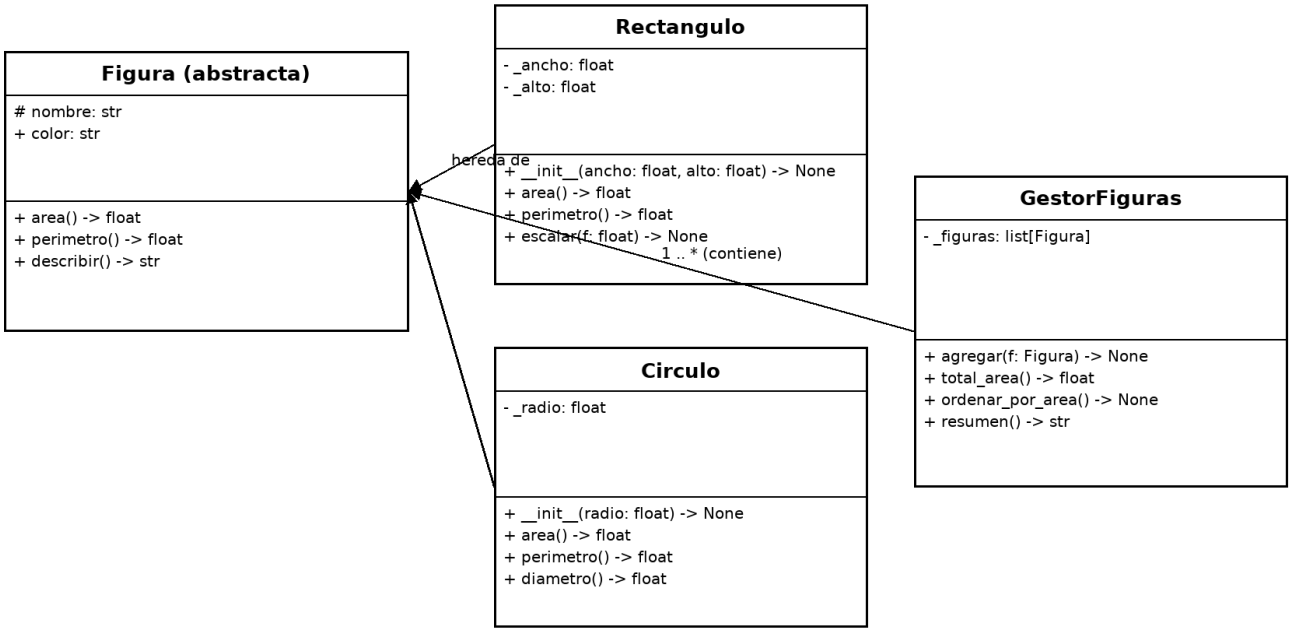
3.4 Polimorfismo

Capacidad de invocar operaciones a traves de una interfaz comun y obtener comportamientos especificos segun el tipo concreto.

4. UML tecnico de referencia

Diagrama que muestra una jerarquia de Figura con implementaciones concretas y un gestor que compone una coleccion de figuras.

POO con Python - Tema 1 (extendido): Introduccion a la POO



POO con Python - Tema 1 (extendido): Introduccion a la POO

5. Ejemplos en Python (comentados)

5.1 Clase abstracta y subclases

```
from abc import ABC, abstractmethod

class Figura(ABC):
    def __init__(self, nombre: str, color: str = 'negro') -> None:
        self._nombre = nombre      # protegido (convencion)
        self.color = color         # publico

    @abstractmethod
    def area(self) -> float: ...

    @abstractmethod
    def perimetro(self) -> float: ...

    def describir(self) -> str:
        return f'Figura {self._nombre} de color {self.color}'

class Rectangulo(Figura):
    def __init__(self, ancho: float, alto: float, color: str = 'negro') -> None:
        super().__init__('rectangulo', color)
        self._ancho = float(ancho)
        self._alto = float(alto)
    def area(self) -> float:
        return self._ancho * self._alto
    def perimetro(self) -> float:
        return 2 * (self._ancho + self._alto)

class Circulo(Figura):
    def __init__(self, radio: float, color: str = 'negro') -> None:
        super().__init__('circulo', color)
        self._radio = float(radio)
    def area(self) -> float:
        from math import pi
        return pi * (self._radio ** 2)
    def perimetro(self) -> float:
        from math import pi
        return 2 * pi * self._radio
```

5.2 Gestor de figuras (composicion y polimorfismo)

```
class GestorFiguras:
    def __init__(self) -> None:
        self._figuras: list[Figura] = []
    def agregar(self, f: Figura) -> None:
        self._figuras.append(f)
    def total_area(self) -> float:
        return sum(f.area() for f in self._figuras)
    def ordenar_por_area(self) -> None:
        self._figuras.sort(key=lambda f: f.area())
    def resumen(self) -> str:
        return ', '.join(type(f).__name__ for f in self._figuras)

g = GestorFiguras()
g.agregar(Rectangulo(3, 4))
```

```
g.agregar(Circulo(2))
print('Area total:', g.total_area())
```

POO con Python - Tema 1 (extendido): Introduccion a la POO

6. Buenas practicas y advertencias

- Elegir nombres significativos para clases, metodos y atributos.
- Mantener la cohesion y aplicar SRP (una responsabilidad por clase cuando sea razonable).
- Preferir propiedades (@property) para validar acceso en lugar de exponer atributos directamente.
- Evitar jerarquias profundas sin necesidad; considerar composicion.

7. Errores comunes

- Confundir atributos de clase con atributos de instancia.
- Implementar metodos que rompen invariantes sin validacion.
- No usar __repr__/__str__ y dificultar la depuracion.
- Hacer downcasting manual innecesario en lugar de polimorfismo.

8. Ejercicios propuestos

Basico

Implemente Triangulo(base: float, altura: float) como subclase de Figura. Defina area() y perimetro() (suponga equilatero para simplificar).

Intermedio

Extienda GestorFiguras con eliminar_por_tipo(tipo: type) y mayor_figura() que devuelva la figura con mayor area.

Avanzado (reto)

Disene una jerarquia Figura3D (abstracta) con Cubo y Esfera; implemente volumen() y superficie(). Adapte un Gestor3D.

9. Autoevaluacion (preguntas)

1) Defina polimorfismo y explique una ventaja practica.
2) Cual es la diferencia entre herencia y composicion; cite un caso para cada una.
3) Por que conviene usar clases abstractas con @abstractmethod en Python.
4) Que problema evita exponer directamente atributos publicos sin validacion.