

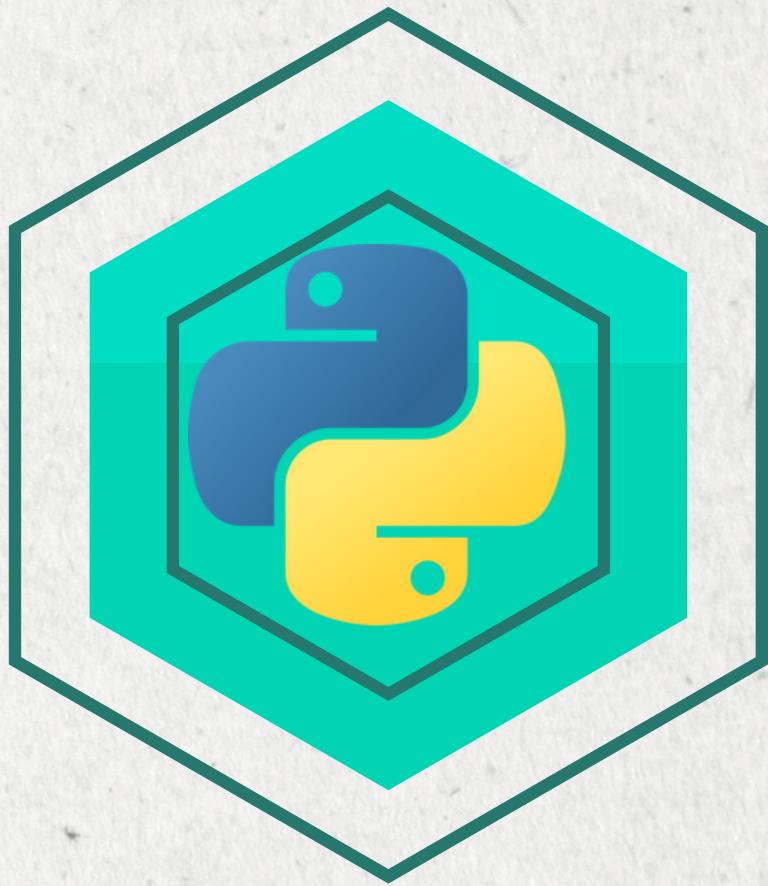
PYTHON

KEVIN STEVEN MARIN CARDENAS II52477

GIAN KARLO ABRIL FIERRO II52466

ANDRES SEBASTIAN CASADIEGO RODRIGUEZ II52501

JUAN DAVID LLANOS CASTAÑEDA II52462



CONTENIDOS

- 1 Historia
- 2 Ubicación en el Ranking
- 3 Utilidad del Lenguaje
- 4 Clases y Objetos
- 5 Contenedores
- 6 Herencia
- 7 Tecnología y Herramientas

HISTORIA

Python es un lenguaje de programación de alto nivel, concebido por Guido van Rossum a fines de 1989 mientras trabajaba en el Centrum Wiskunde & Informatica (CWI) en los Países Bajos. Van Rossum creó Python con la intención de diseñar un lenguaje que fuera fácil de leer y escribir, manteniendo al mismo tiempo una gran potencia y flexibilidad.

El propósito principal de Python era brindar una alternativa más intuitiva a los lenguajes de programación existentes, permitiendo a los desarrolladores centrarse en la resolución de problemas en lugar de preocuparse por la sintaxis del lenguaje. Python prioriza la legibilidad del código y la simplicidad, factores que han contribuido a su adopción masiva.

```
response = requests.get(url)

if response.status_code != 200:
    print(f"Status: {response.status_code}")

print(f"Status: {response.status_code}\n\nUsing BeautifulSoup to parse the response")
soup = BeautifulSoup(response.content, "html.parser")


s = soup.find_all("img", attrs={"alt": "Post images"})

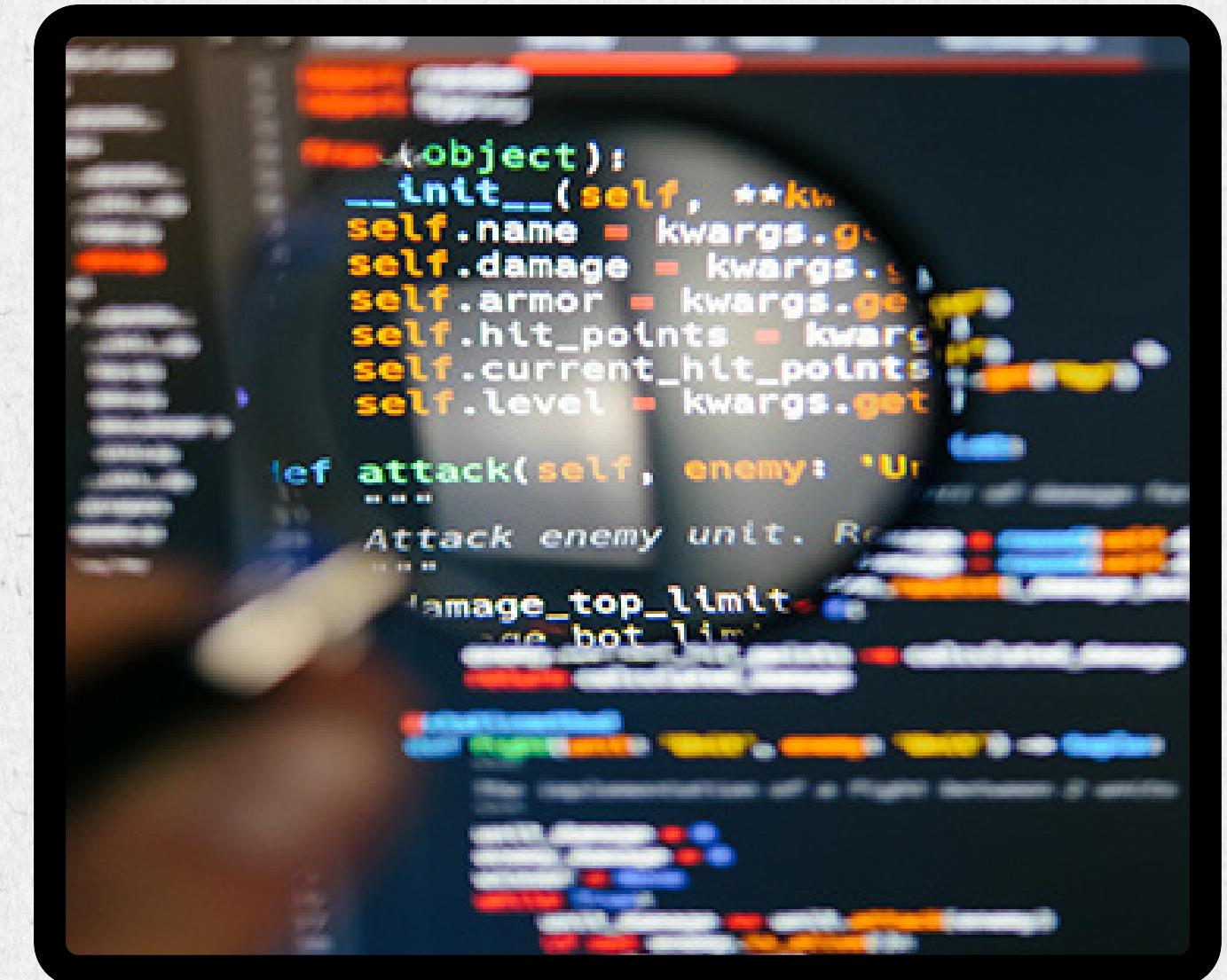
unloading images
images = 0
for image in s:
    if image['src'] == None:
        continue
    else:
        images += 1
        print(f"Unloading image {images} from {url}
```

DESAFIOS

Cuando Guido van Rossum creó Python, uno de sus objetivos principales era reducir la complejidad de los lenguajes de programación existentes.

Python se diseñó con una sintaxis clara y legible que facilita la comprensión del código. La estructura coherente y la sangría obligatoria no solo mejoran la legibilidad, sino que también ayudan a evitar errores comunes.

Al ser un lenguaje de alto nivel, Python gestiona muchos detalles técnicos automáticamente, permitiendo a los desarrolladores centrarse en la lógica del problema.



UBICACIÓN EN EL RANKING

Python sigue liderando los rankings de popularidad en programación según varias fuentes importantes como los índices TIOBE y PYPL. Este lenguaje se mantiene en el puesto número 1 debido a su versatilidad, facilidad de aprendizaje y uso en áreas clave como inteligencia artificial, ciencia de datos y desarrollo web.



UTILIDADES DEL LENGUAJE

DESARROLLO WEB

python es un lenguaje que se usa ampliamente en el desarrollo web ,principalmente para el desarrollo backend ,que se usa para crear aplicaciones web potentes y escalables ,lo que tambien significa que se usa para construir la logica detras del procedimiento

MACHINE LEARNING

python es un lenguaje de programación muy utilizado en machine learning por sus ventajas como tener una sintaxis simple ,cuenta con varias bibliotecas de código abierto como SciPy y TensorFlow ,paquetes de software que permiten construir histogramas,gráficos y diagramas para entender mejor los datos,ademas de la facilidad de aprender el lenguaje y su eficiencia.

INTELIGENCIA ARTIFICIAL

python esta entre los mejores lenguajes de alto nivel ,ya que , posee una amplia gama de bibliotecas de datos y fuentes con algoritmos establecidos ,y su código de programación es mucho mas amigable facilitando el proceso de desarrollo de algoritmos complejos de ia ,permitiendo a los programadores concentrarse en la logica y el diseño.

AUTOMATIZACION DE TAREAS

Python es una herramienta poderosa y versátil para la automatización de tareas. Su simplicidad y vasta colección de bibliotecas facilitan la automatización de procesos repetitivos y tediosos, mejorando la eficiencia y reduciendo el error humano.

ejemplos:

- Automatización de Archivos y Carpetas
- Automatización de Tareas en la Web
- Automatización de Emails
- Tareas de Sistema y Administración
- Automatización en Excel y Documentos
- Automatización de Redes y Servidores

CIENCIA DE DATOS

Python se ha convertido en un elemento básico en la ciencia de datos, permitiendo a los analistas de datos y a otros profesionales utilizar el lenguaje para realizar cálculos estadísticos complejos, crear visualizaciones de datos, construir algoritmos de aprendizaje automático, manipular y analizar datos, y completar otras tareas relacionadas con los datos.

DESARROLLO DE SOFTWARE

Python es ampliamente utilizado en el desarrollo de software por sus múltiples ventajas y utilidades.

ejemplos:

- Desarrollo de Aplicaciones Web
- Desarrollo de Aplicaciones de Escritorio
- Desarrollo de Juegos
- Desarrollo de APIs y Servicios Web

CLASES

Una clase es una plantilla o molde para crear **objetos**.

Define las **propiedades** (atributos) y **métodos** (comportamientos) que tendrán los objetos creados a partir de ella.

```
class Punto:  
  
    def __init__(self, x, y): # <-- Método Constructor  
        self.x = x # <-- Propiedad  
        self.y = y
```

PROPIEDADES

Las variables en Python son dinámicas y el tipo de dato se asigna en tiempo de ejecución basado en el valor asignado.

```
# Entero  
edad = 25  
print(f'Edad: {edad}')  
  
# Flotante  
precio = 19.99  
print(f'Precio: {precio}')  
  
# Cadena de texto  
nombre = "Juan"  
print(f'Nombre: {nombre}')  
  
# Booleano  
es_mayor_de_edad = True  
print(f'Es mayor de edad: {es_mayor_de_edad}')
```

MÉTODOS

MÉTODOS DE INSTANCIA:

Los métodos de instancia son los métodos más comunes. Se definen dentro de una clase y operan sobre instancias específicas de esa clase. El primer parámetro de un método de instancia siempre es **self**, que se refiere a la instancia actual de la clase

```
def distancia(self, otro_punto):
    """Calcula la distancia entre este punto y otro punto"""
    dx = self.x - otro_punto.x
    dy = self.y - otro_punto.y
    return math.sqrt(dx ** 2 + dy ** 2)
```

MÉTODOS

MÉTODOS DE CLASE:

Los métodos de clase son métodos que operan sobre la clase en sí, en lugar de sobre instancias de la clase.

El **primer parámetro es cls**, que se refiere a la clase misma.

```
@classmethod  
def contar_figuras(cls):  
    return cls.contador_figuras
```

MÉTODOS

MÉTODOS ESTÁTICOS:

Los métodos estáticos son métodos que no dependen de la instancia ni de la clase. **No** tienen ningún parámetro implícito como **self** o **cls**. Se utilizan para realizar tareas que tienen una relación lógica con la clase, pero no necesitan acceder a sus atributos o métodos.

```
@staticmethod  
def es_cuadrado(lados):  
    if len(lados) != 4:  
        return False  
    longitudes = [lado.calcular_longitud() for lado in lados]  
    return all(l == longitudes[0] for l in longitudes)
```

MÉTODOS ESPECIALES (DUNDER METHODS)

Los métodos especiales, también conocidos como métodos "dunder" (doble guion bajo), tienen un propósito específico y son reconocidos por Python para realizar ciertas operaciones.

Algunos ejemplos comunes son `__init__`, `__str__`, `__repr__`, `__len__`, entre otros.

MÉTODOS

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

def __str__(self):
    return f"({self.x}, {self.y})"
```

ENCAPSULAMIENTO

Es el principio de ocultar los detalles internos de una clase y exponer solo lo necesario. Esto se logra utilizando métodos de acceso (**getters** y **setters**) y convenciones de nombres (**lados**).

```
class Figura:  
    def __init__(self, lados):  
        self._lados = lados # Propiedad encapsulada  
  
    def get_lados(self):  
        return self._lados # Método de acceso
```

OBJETOS

```
# Crear puntos  
punto1 = Punto(0, 0)  
punto2 = Punto(0, 2)  
punto3 = Punto(2, 2)  
punto4 = Punto(2, 0)
```

Un objeto es una instancia de una clase. Es una entidad concreta que sigue la estructura y comportamientos definidos por su clase.

CONTENEDORES

LISTAS (LIST)

Secuencias ordenadas de elementos que pueden ser de diferentes tipos.

```
mi_lista = [1, 2, 3]
mi_lista.append(4)
print(mi_lista) # Salida: [1, 2, 3, 4]
```

CONTENEDORES

TUPLAS (TUPLE)

Secuencias ordenadas de elementos, pero inmutables
(no pueden modificarse después de su creación).

```
mi_tupla = (1, 2, 3)
print(mi_tupla.count(2)) # Salida: 1
```

CONTENEDORES

CONJUNTOS

Colección desordenada de elementos únicos (sin duplicados).

```
conjunto = {1, 2, 3, 4, 5}
```

CONTENEDORES

DICCIONARIOS

Son colecciones desordenadas de pares clave-valor, lo que significa que cada elemento en un diccionario tiene una clave única asociada a un valor.

```
# Crear un diccionario
diccionario = {"nombre": "Juan", "edad": 30, "ciudad": "Bogotá"}

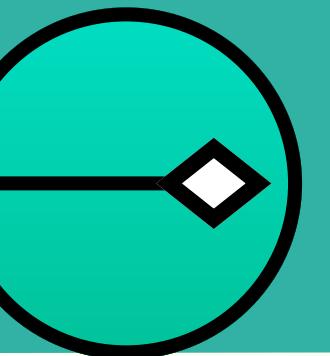
# Acceder a un valor
print(diccionario["nombre"]) # Salida: Juan
```

ASOCIACIÓN →

La asociación es una relación entre dos clases donde una clase puede referenciar a una instancia de otra clase. Esta relación puede ser de muchos a muchos, uno a muchos, o uno a uno. La asociación no implica propiedad.

```
class Lado:  
    def __init__(self, punto_inicial, punto_final):  
        self.__punto_inicial = punto_inicial  
        self.__punto_final = punto_final  
  
    def __str__(self):  
        return f'Lado({self.__punto_inicial}, {self.__punto_final}, Longitud: {self.calcular_longitud()})'  
  
    def calcular_longitud(self):  
        return self.__punto_inicial.calculardistancia(self.__punto_final)
```

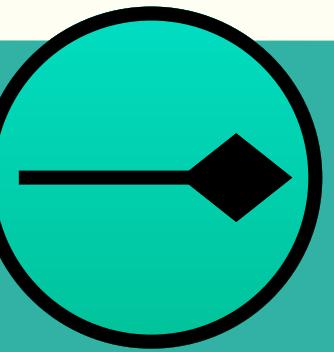
AGREGACIÓN



La agregación es un tipo específico de asociación donde una clase contiene una colección de otras clases. En la agregación, las clases componentes pueden existir independientemente de la clase contenedora. Se considera una relación "tiene un" (has-a).

```
class Figura:  
    def __init__(self, lados):  
        self.__lados = lados  
  
    def agregar_lado(self, lado):  
        self.__lados.append(lado)
```

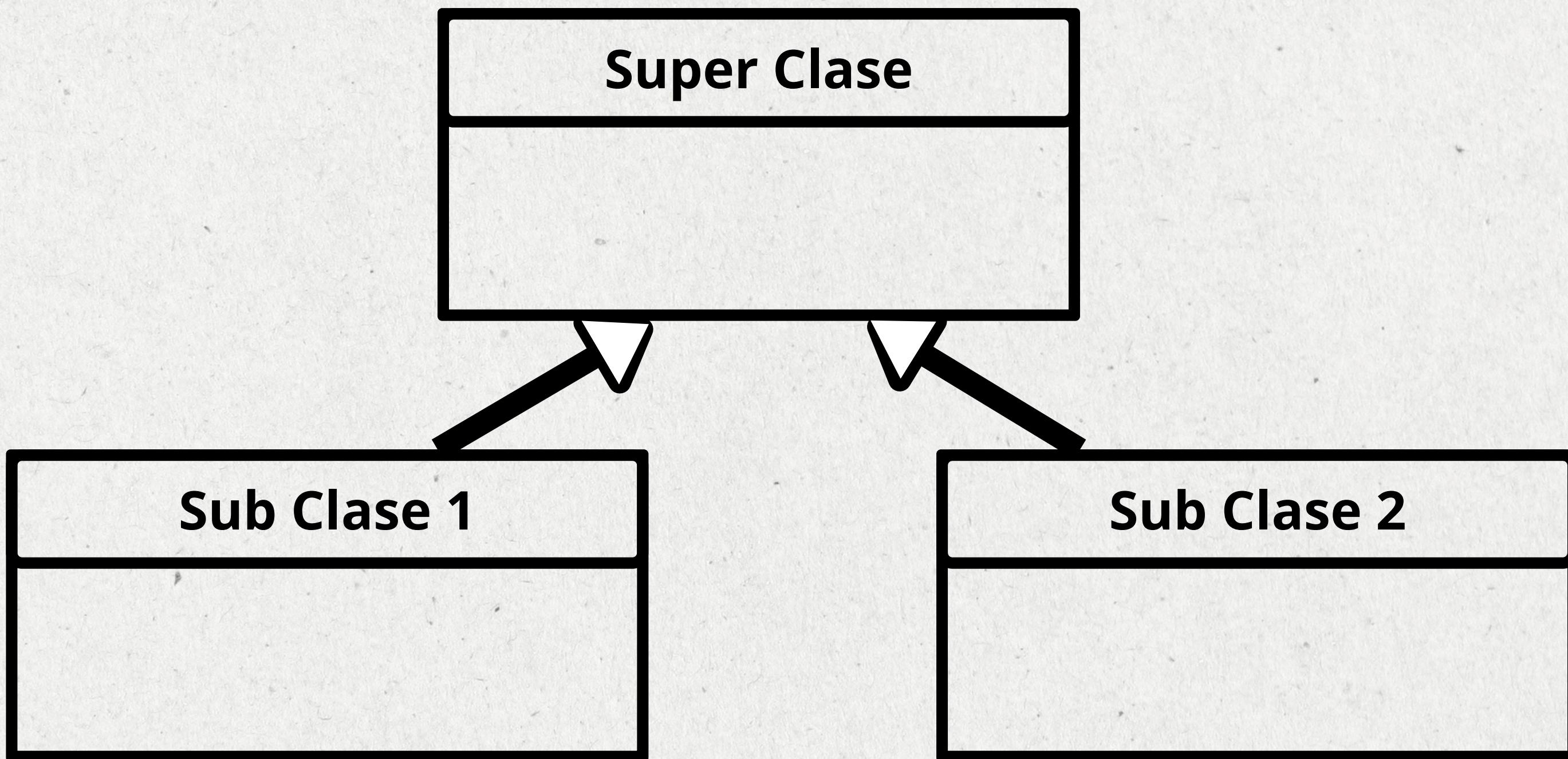
COMPOSICIÓN



La composición es una forma más fuerte de agregación. En la composición, la vida de los objetos componentes está fuertemente ligada al objeto contenedor, y no pueden existir independientemente de él.

```
class Lado:  
    def __init__(self, x1, x2, y1, y2):  
        self.__punto_inicial = Punto(x1, y1)  
        self.__punto_final = Punto(x2, y2)
```

HERENCIA



Figura

```
class Figura:  
    def __init__(self, lados):  
        self.__lados = lados  
  
    1 usage  
    def agregar_lado(self, lado):  
        self.__lados.append(lado)  
  
    def calcular_perimetro(self):  
        perimetro = 0  
        for lado in self.__lados:  
            perimetro += lado.calcular_longitud()  
        return perimetro  
  
    5 usages  
    def get_lados(self):  
        return self.__lados
```

SUPER CLASE

MÉTODO ABSTRACTO

Figura

```
@abstractmethod  
def calcular_area(self):  
    # Método para sobrescribir  
    pass
```

SUBCLASE

Triangulo

```
class Triangulo(Figura):  
    def __init__(self, lados):  
        if len(lados) != 3:  
            raise ValueError("No tiene 3 lados")  
        super().__init__(lados)  
  
    def agregar_lado(self, lado):  
        self.get_lados().pop(0)  
        super().agregar_lado(lado)
```

MÉTODO ABSTRACTO

Triangulo

```
def calcular_area(self):  
    if len(self.get_lados()) != 3:  
        raise ValueError("No tiene 3 lados")  
  
    a = self.get_lados()[0].calcular_longitud()  
    b = self.get_lados()[1].calcular_longitud()  
    c = self.get_lados()[2].calcular_longitud()  
  
    s = (a + b + c) / 2  
    area = math.sqrt(s * (s - a) * (s - b) * (s - c))  
    return round(area, 2)
```

CONSTRUCCIÓN DE LA FIGURA

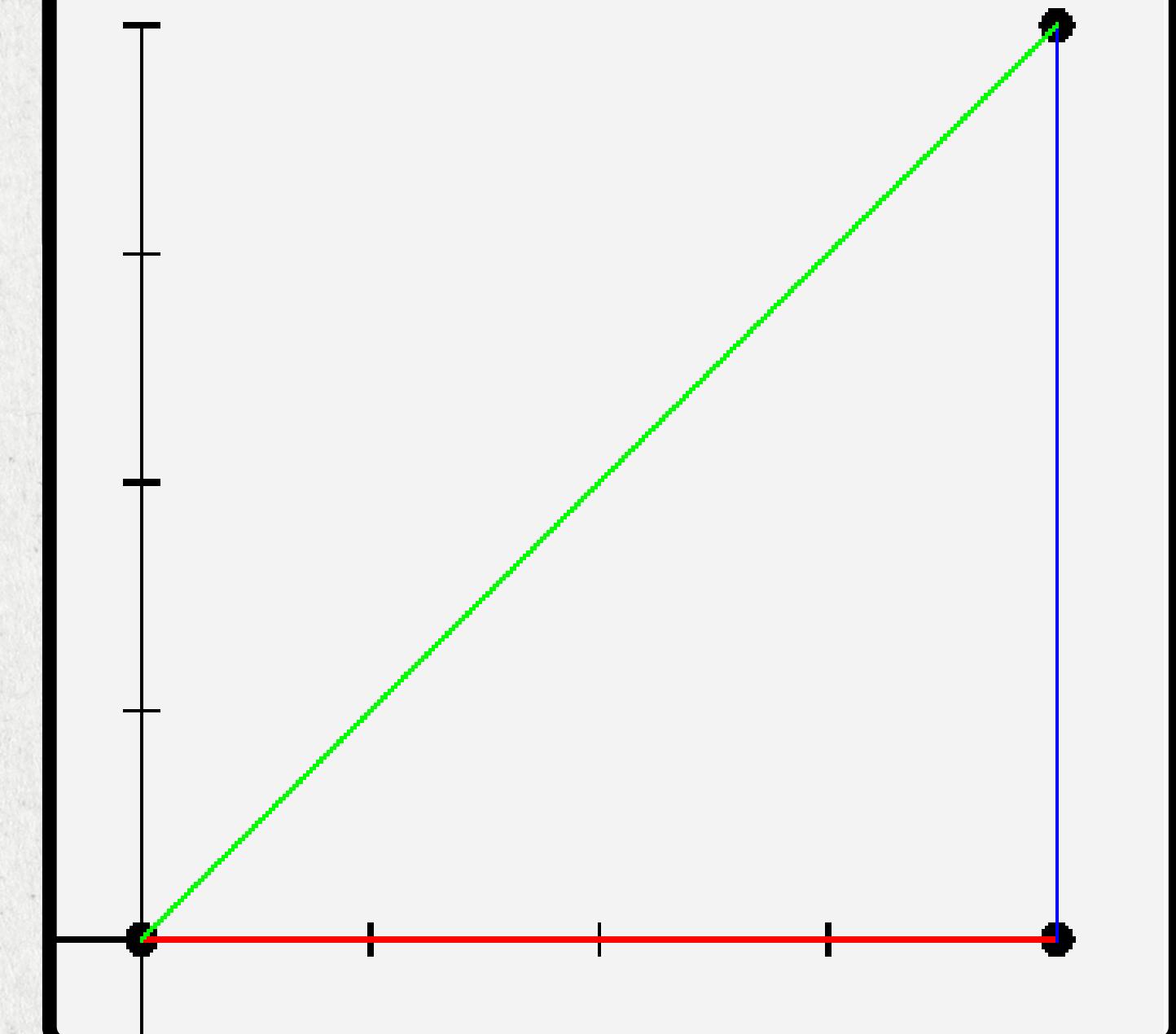
Main

```
punto1 = Punto( x: 0, y: 0)
punto2 = Punto( x: 4, y: 0)
punto3 = Punto( x: 4, y: 4)

lado1 = Lado(punto1, punto2)
lado2 = Lado(punto2, punto3)
lado3 = Lado(punto3, punto1)

lados = [lado1, lado2, lado3]
```

Grafico Triangulo



MÉTODOS HEREDADOS

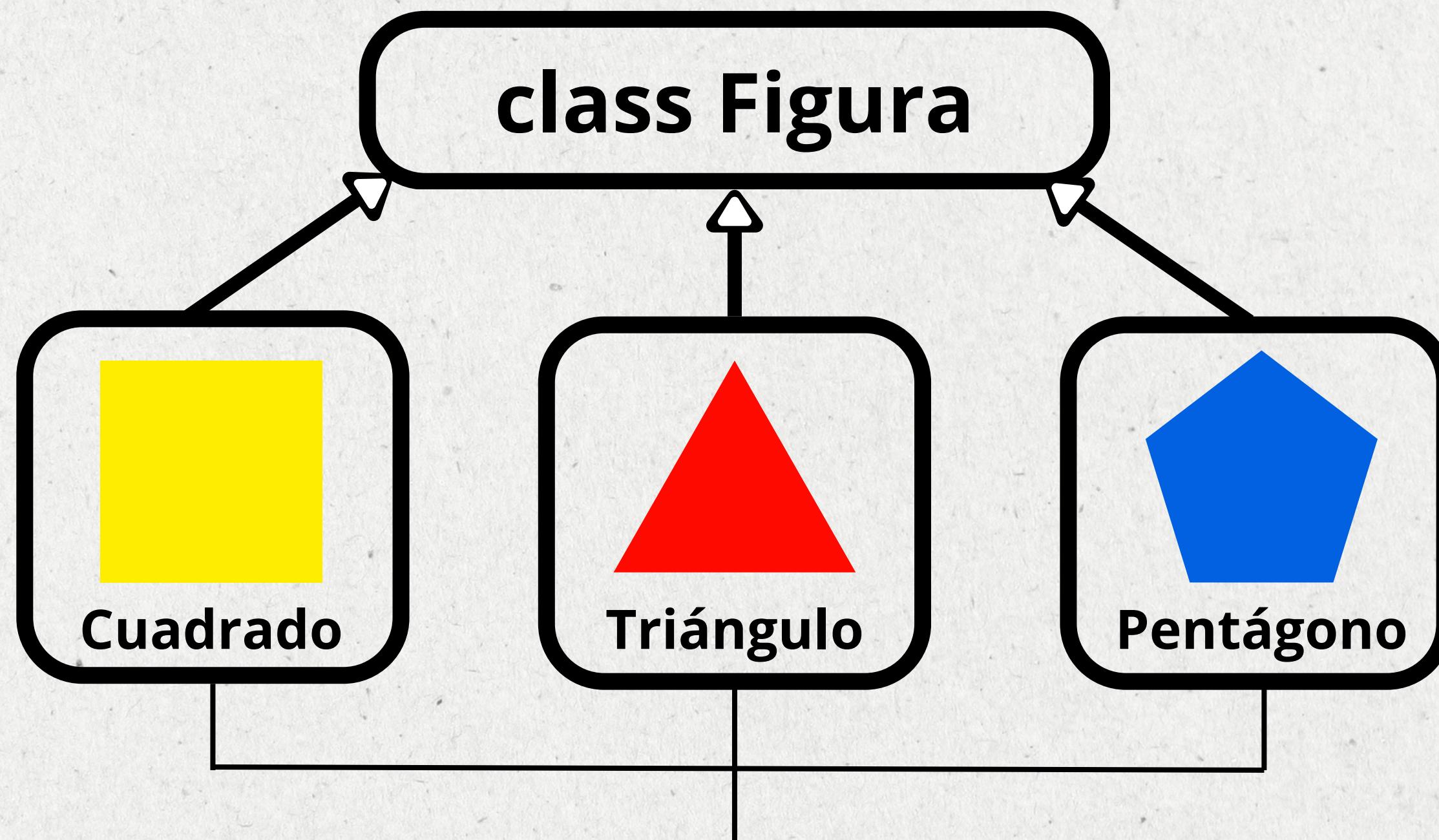
Main

```
triangulo1 = Triangulo(lados)  
  
print("Perímetro: "+str(triangulo1.calcular_perimetro()))  
print("Área: "+str(triangulo1.calcular_area()))
```

consola

```
Perímetro: 13.65  
Área: 8.0
```

POLIMORFISMO



SIGUEN SIENDO FIGURAS

Cuadrado

```
class Cuadrado(Figura):
    def __init__(self, lados):
        if len(lados) != 4:
            raise ValueError("No tiene 4 lados")
        for i in range(len(lados)):
            if i > 0:
                if (lados[i].calcular_longitud() !=
                    lados[i - 1].calcular_longitud()):
                    raise ValueError("Los lados no miden lo mismo")
        super().__init__(lados)

    def agregar_lado(self, lado):
        pass

    def calcular_area(self):
        lado = self.get_lados()[0].calcular_longitud()
        return lado * lado
```

CUADRADO

CONSTRUCCION DE LAS FIGURAS

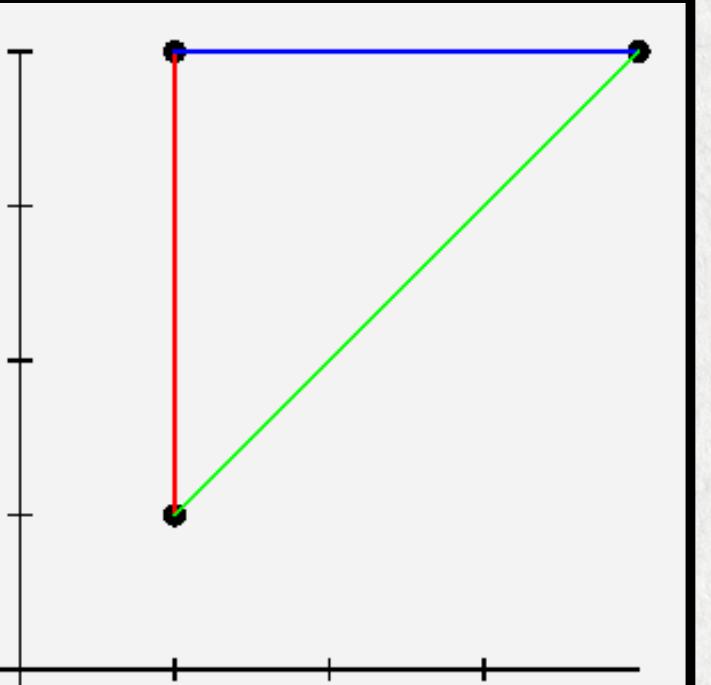
Main

```
punto1 = Punto( x: 1, y: 1)
punto2 = Punto( x: 1, y: 4)
punto3 = Punto( x: 4, y: 4)
punto4 = Punto( x: 4, y: 1)

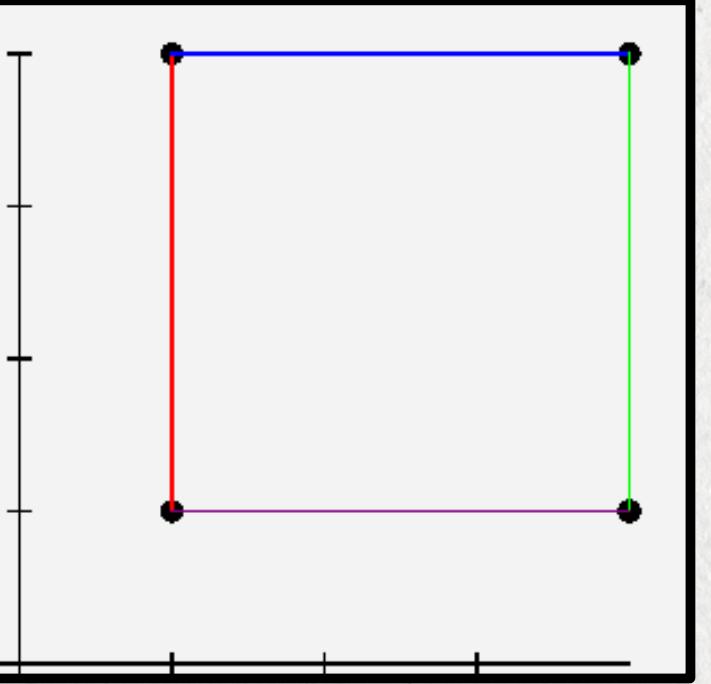
lado1 = Lado(punto1, punto2)
lado2 = Lado(punto2, punto3)
lado3 = Lado(punto3, punto4)
lado4 = Lado(punto4, punto1)
lado5 = Lado(punto3, punto1)

lados_cuadrado = [lado1, lado2, lado3, lado4]
lados_triangulo = [lado1, lado2, lado5]
```

Triangulo



Cuadrado



INSTANCIA Y POLIMORFISMO

Main

```
cuadrado1 = Cuadrado(Lados_cuadrado)
triangulo1 = Triangulo(lados_triangulo)
figuras = [cuadrado1, triangulo1]

for i in range(len(figuras)):
    print("Figura "+str(i+1)+":")
    print("Perímetro: "+str(figuras[i].calcular_perimetro()))
    print("Área: "+str(figuras[i].calcular_area())+"\n")
```

consola

```
Figura 1:
Perímetro: 12.0
Área: 9.0
```

consola

```
Figura 2:
Perímetro: 10.24
Área: 4.5
```

GUI: GRAFICADOR

Graficador

Graficador

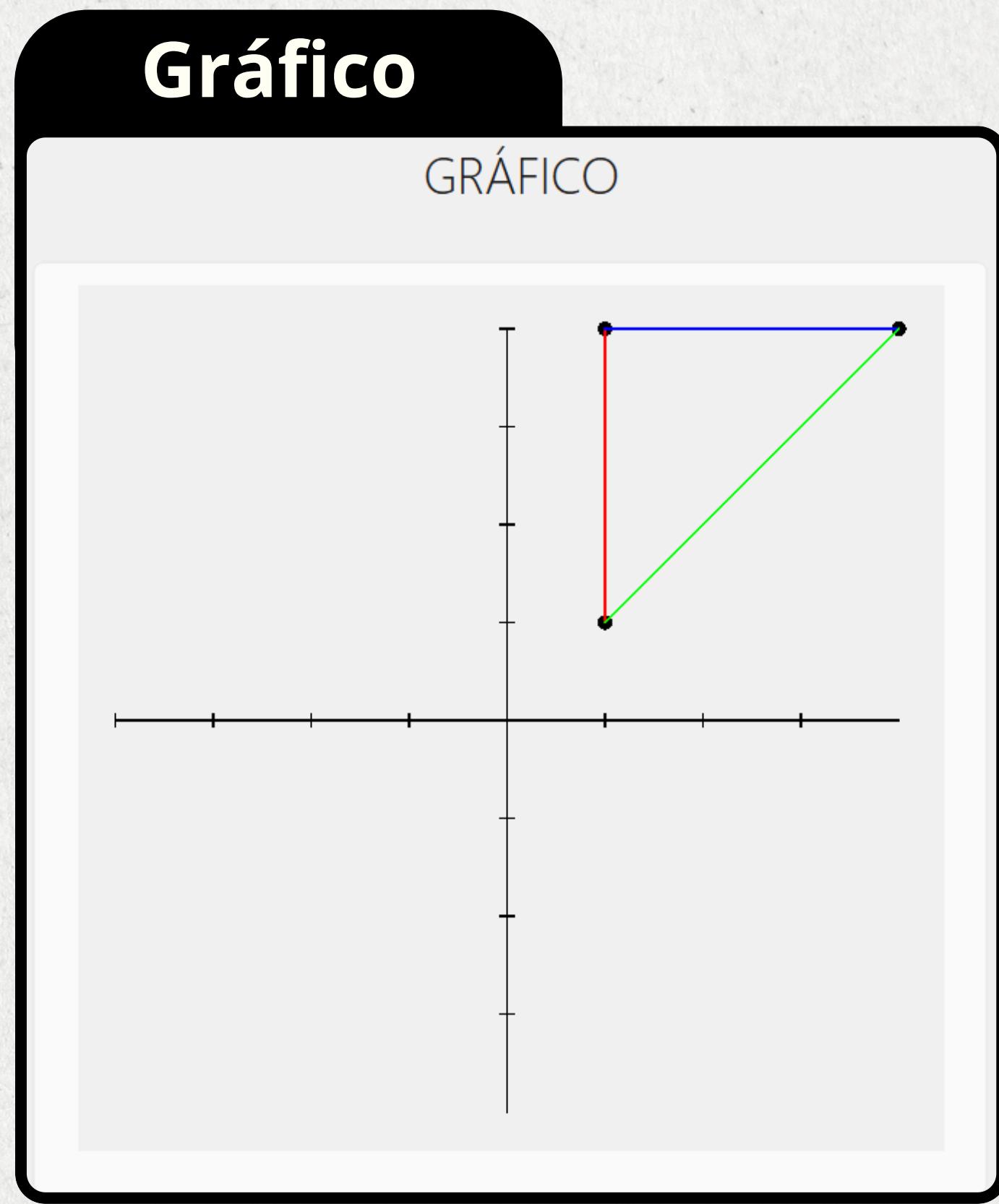
GRAFICADOR

NUMERO DE COORDENADAS:

Punto 1 x= y=

Punto 2 x= y=

Punto 3 x= y=



iGRACIAS!