UCSC

UNIVERSIDAD CATÓLICA DE LA SANTÍSIMA CONCEPCIÓN

Taller de programación II - IN1071C

Introducción

Diego Maldonado

Facultad de Ingeniería
Departamento de Ingeniería Informática

31 de agosto de 2022

La Programación Orientada a Objetos (POO), cumple un rol muy immportante en el mundo de la informática: el de facilitar el vínculo entre los problemas del mundo real y la potencia de cálculo de los computadores para resolverlos. En este curos se le enseñarán los principios de este paradigma de programación, con el objetivo de que pueda "pensar" en objetos con la ayuda del ULM (Lenguaje Unificado de Modelado,por sus siglas en inglés, Unified Modeling Language), ponerlos en práctica a través del código `Python` y, lo más importante, le enseñara la forma correcta de reflexionar respecto al desarrollo de software orientado a objetos.

De acuerdo a Wikipedia "Python es un lenguaje de alto nivel de programación interpretado cuya filosofía hace hincapié en la legibilidad de su código, se utiliza para desarrollar aplicaciones de todo tipo, ejemplos: Instagram, Netflix, Spotify, Panda 3D, entre otros. Se trata de un lenguaje de programación multiparadigma, ya que soporta la orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, dinámico y multiplataforma."

Python fue creado por Guido van Rossum en el Centro para las Matemáticas y la Informática de Países Bajos a finales de la década de los 80s y lleva su nombre por la afición de Guido a los humoristas ingleses Monty Python.

Actualmente Python está bajo la administración de la *Python Software Foundation* bajo una licencia de código abierto propia llamada *Python Software Foundation License* y cuenta con una gran comunidad que está constantemente en desarrollando bibliotecas y actualizando su código. La última versión estable al momento de escribir este documento es la versión 3.10.5.

Virtudes de Python:

- Lenguaje interpretado
- Multi-paradigma
- Alto nivel
- Privilegia el orden. Tabulación obligatoria.
- No requiere definir los tipos de datos de las variables.
- ETC.

Ejemplo (Hola mundo)

```
1  print('Hola Mundo')
```

Ejemplo (Tipos de datos básicos en Python)

```
1  n=10          % Numero entero
2  x=10.0        % Numero flotante
3  s='hola'      % String
4  L=[n,x,s]     % Lista
```

Ejercicio

```
1  print(L)
```

Ejemplo (Estructuras de control)

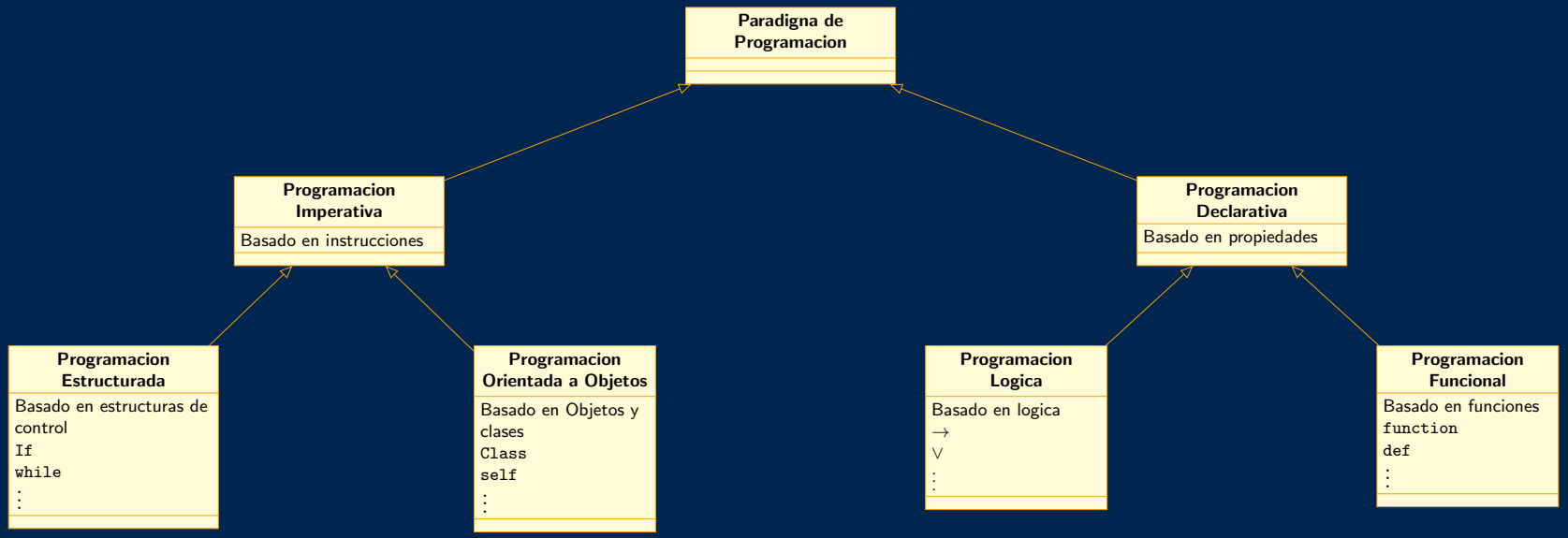
```
1  for x in range(10):
2      pass
3  n=1
4  while n<10:
5      n=n+1
6      pass
7  z=1
8  if z==1:
9      pass
10 elif z==2:
11     pass
12 else:
13     pass
```

Ejercicio

- print(range(10))
- Imprima los números del 1 al 10.
- Imprima los números pares del 1 al 10.
- Dado un número x, determine si es o no divisible por 3 (use el comando % para obtener el resto). Entregue como salida "Es divisible por 3" o "No es divisible por 3".

El lenguaje unificado de modelado (UML, por sus siglas en inglés, Unified Modeling Language) es un lenguaje de modelado de sistemas de software. Está respaldado por el Object Management Group (OMG), que se dedica a establecer estándares de tecnologías orientadas a objetos.

EL UML es un lenguaje de modelado para especificar o para describir métodos o procesos. Se utiliza para definir un sistema, para detallar los artefactos en el sistema, para documentar y construir. Se describe mediante diagramas UML que detallan las relaciones y propiedades de los artefactos que componen el sistema.



Factorial: Programacion Estructurada

```
factorial=1
for i = 1...n
  factorial=factorial· i
```

Factorial: Programacion Orientada a Objetos

```
class entero:
  def __init__(self, valor):
    self.valor = valor
    def factorial(self):
      factorial=1
      for i = 1...self.valor
        factorial=factorial· i
      Return entero(factorial)
```

Factorial: Programacion Logica

$factorial(1) = 1 \wedge factorial(n) = n \cdot factorial(n-1)$

Factorial: Programacion Funcional

```
def factorial(n):
  if n==1:
    Return 1
  else:
    Return n*factorial(n-1)
```

Objetos en la vida cotidiana:

1. percibir
2. sentir
3. manipular
4. hacen cosas

Objetos en desarrollo de software (intangibles):

1. tiene información
2. hacen cosas

Ejemplo

Objeto real

Nombre $\rightarrow$ "cubo 1"
Caras $\rightarrow$ 6
PonerSobre(Objeto)

Objeto Software

Definición

Un objeto es una colección de datos y comportamientos asociados.

En la vida, podemos clasificar los objetos de acuerdo a características comunes. Por ejemplo, los objetos con: ruedas y capacidad de llevar pasajeros, los podemos clasificar en la clase vehículo. Los barcos, aviones, autos, motocicletas, bicicletas, etc., puede catalogarse como vehículos, siendo cada uno de ellos un objeto diferente.

Definición

Una clase es el conjunto de datos y acciones que deben tener todos los objetos de una misma clase. A los datos se les llama atributos y a las acciones que pueden realizar se les llama métodos.

Ejemplo

Objeto real

Clase :: Cubo
Nombre :: str
Caras :: array
EstaSobre :: str
PonerSobre(Objeto)

Clase Cubo

Clase = Cubo
Nombre = 'cubo1'
Caras = [z,y,x,w,v,u]
Cara[1] = "z"
Cara[2] = "y"
EstaSobre = 'cubo2'
PonerSobre(Objeto)

Un Cubo

En UML la clase se definen:

Nombre de clase
nombreAtributo : tipo de dato nombreAtributo : tipo de dato = valor por defecto
nombreMetodo (lista,de,parametros) : tipo de salida

Ejemplo

Clase vehículo

Vehiculo
Ruedas : int Capacidad : int = 1 Pasajeros : int = 0
SubirPasajero (int) BajarPasajero (int)

Objeto particular

Auto: Vehiculo
Ruedas : 4 Capacidad : 5 Pasajeros : 0
SubirPasajero (int) BajarPasajero (int)

En Python la clase se define:

```
1  class Nombre:
2    def __init__(self, atr1,...):
3      self.atr1 = atr1
4    def metodo1(self,arg1,...):
5      Acciones
```

Ejemplo

Clase vehículo

```
1  class Vehiculo:
2    def __init__(self,ruedas,cap=1,pas=0):
3      self.ruedas = ruedas
4      self.cap = cap
5      self.pas = pas
6    % COMENTARIO PARA SEPARAR METODOS
7    def SubirPasajero(self,n):
8      pass
9    def BajarPasajero(self,n):
10     pass
```

Objeto particular

```
1  auto = Vehiculo(4,5,0)
```

Ejercicio

1. Implemente la clase `Vehículo` en Python.
2. Cree el vehículo `auto` del ejemplo anterior.
3. Añada el atributo "Color" de tipo `str`.
4. Implemente los métodos `SubirPasajero` y `BajarPasajero`, que reciben como argumento un entero y aumentan y disminuyen la cantidad de pasajeros en el vehículo respectivamente.
5. Implemente una validación a los métodos anteriores para no agregar más pasajeros que la capacidad del vehículo y no quitar más pasajeros de los que hay. Imprima un mensaje de error con el comando `print`.
6. Cree los vehículos:
   - Avión de 3 ruedas, color rojo, capacidad de 2 y 0 pasajeros abordo.
   - Bus de 8 ruedas, color azul, capacidad de 40 y 0 pasajeros abordo.
   - Bicicleta de 2 ruedas, color verde , capacidad de 1 y 0 pasajeros abordo.
   - Bote de 0 ruedas, color gris, capacidad de 4 y 2 pasajeros abordo.

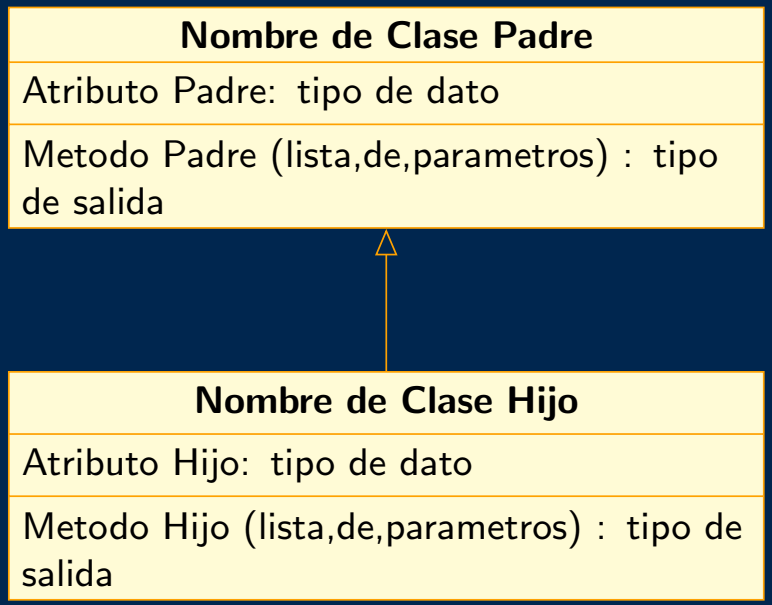
   Verifique que `Avión` tiene 3 ruedas, capacidad de 2 y 0 pasajeros abordo con los comandos `Avion.ruedas`, `Avion.cap` y `Avion.pas` respectivamente.
7. Añada un pasajero en cada vehículo con el comando `objeto.SubirPasajero(1)`.

En el mundo real, la clasificación de los objetos depende del nivel de abstracción en el que estemos interesados, pero siempre podemos ser menos abstractos detallando más las clases que tenemos. En este proceso, esperamos que las nuevas clases, conserven las características de sus predecesoras. Llamamos a este proceso herencia.

Ejemplo

Ya conocimos la clase `Vehículo`, pero existen distintos tipos de `Vehículo`, por ejemplo podemos separar los `Vehículos` en terrestres, aéreos y acuáticos, y cada uno de ellos tiene características diferentes a los otros, pero todos conservan las características generales de a clase `Vehículo`, por ejemplo, llevar pasajeros.

En UML la herencia se definen:

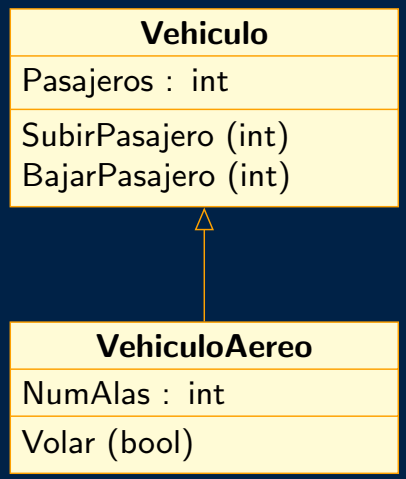


En Python la herencia se define:

```
1  class NombrePadre:
2    def __init__(self, atr1,...):
3      self.atr1 = atr1
4    def metodo1(self,arg1,...):
5      Acciones
6  class NombreHijo(NombrePadre):
7    def __init__(self, atr1,...):
8      self.atr1 = atr1
9    def metodo2(self,arg1,...):
10     Acciones
```

Ejemplo

Clase vehículo



Ejemplo

Clase vehículo

```
1  class vehiculo():
2      def __init__(self,nombre,ruedas,cap):
3          self.nombre = nombre
4          self.ruedas = ruedas
5          self.cap = cap
6      def datos(self):
7          print(self.nombre,self.ruedas,self
                 .cap)
8  class vehiculoTerrestre(vehiculo):
9      def __init__(self,nombre,ruedas,cap,
            motor):
10         vehiculo.__init__(self,nombre,
                ruedas,cap)
11         self.motor=motor
```

Los atributos del padres también se pueden asignar con `super().__init__(atr1,atr2,...)`:

```
class NombrePadre:
  def __init__(self, atr1,...):
    self.atr1 = atr1
  def metodo1(self,arg1,...):
    Acciones
class NombreHijo(NombrePadre):
  def __init__(self, atr1,...):
    super().__init__(atr1,atr2,...)
  def metodo2(self,arg1,...):
    Acciones
```

Ejemplo (Clase vehículo)

```
class vehiculo():
    def __init__(self,nombre,ruedas,cap):
        self.nombre = nombre
        self.ruedas = ruedas
        self.cap = cap
    def datos(self):
        print(self.nombre,self.ruedas,self.cap)
class vehiculoTerrestre(vehiculo):
    def __init__(self,nombre,ruedas,cap,motor):
        super().__init__(nombre,ruedas,cap)
        self.motor=motor
```

Nota

Con `super().__init__(atr1,atr2,...)` no es necesario especificar el padre ni `self`.

Ejercicio

1. Implemente la clase Persona, con los atributos "fecha de nacimiento","RUT". Cree algunas Personas
2. Implemente la subclase "Ingeniero Informático" con atributos propios"Cargo:str", "Proyectos realizados: int" y el método "HacerProyecto()" que aumenta en 1 la cantidad de proyectos realizados.
3. Implemente la subsubclase "Alumno en Práctica de Informática" con atributos propios "fecha de ingreso:str", "Proyectos arruinados: int" y el método "HacerProyecto()" que aumenta en 1 la cantidad de proyectos realizados con probabilidad 0.3. Si no aumenta proyecto realizado entonces aumenta proyectos arruinados.
   Para obtener números aleatorios use

```
1 import random # Biblioteca de numeros aleatorios
2 print(random.random())
```

4. Dibuje el diagrama de clases UML de su modelo.

Definición (agregación)

La agregación es un tipo de asociación que indica que una clase es parte de otra clase (composición débil). Los componentes pueden ser compartidos por varios compuestos. La destrucción del compuesto no conlleva la destrucción de los componentes. El objeto el cual se agregan los otros se llama contenedor.
La agregación se representa en UML mediante un diamante de transparente colocado en el extremo en el que está la clase que representa el "todo".

Ejemplo

En Python, se debe agregar un atributo con una lista para los objetos del contenedor.

Ejercicio

Cree en Python las clases Alumno y Asignatura, de tal forma que el alumno tenga un nombre y la asignatura tenga un nombre y una lista de alumnos inscritos.

Definición (Composición)

Una composición no es más que una agregación donde el contenido no sobrevive a su contenedor. El objeto el cual se agregan los otros se llama contenedor.
La composición se representa en UML mediante un diamante de color colocado en el extremo en el que está la clase contenedora.

Ejemplo

En Python, se debe agregar el destructor __del__ para destruir todos los atributos del contenedor

```
1 def __del__(self):
2     pass
```

Ejercicio

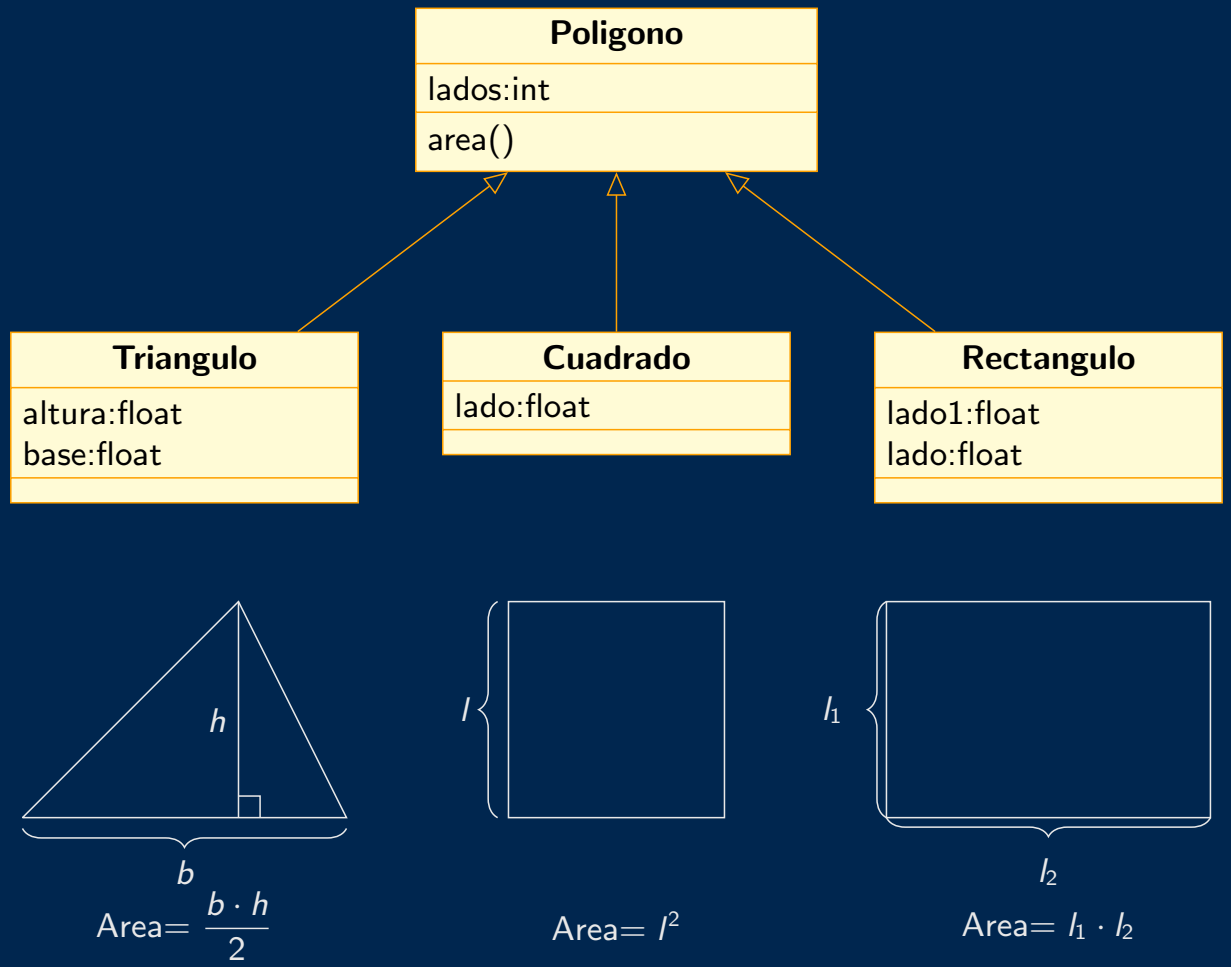
Cree en Python las clases Libro y Hoja, de tal forma que hoja tenga un string y libro tenga un título, un autor y al menos una hoja.

Ejercicio (En grupos)

- En UML:
  1. Implemente la clase Jugador, el cual tiene un nombre y una mano de 13 cartas y 2 tríos, dos espacios donde el jugador puede guardar cartas
  2. Implemente la clase Baraja, la cual tiene 54 cartas. La baraja se debe poder revolver.
  3. Implemente la clase Carta, con número (1-13), pinta y parte posterior.
  4. Implemente la clase Descarte, la cual puede contener cartas.
  5. En un juego de carioca, los jugadores roban una todos los turnos una carta y descartan una carta de su mano.
     Un jugador se puede bajar, es decir puede poner dos tríos en su sección de tríos si tiene dos grupos de 3 cartas del mismo número.
     Un jugador que ya se bajó puede aportar cartas a su trío o a los tríos de los otros jugadores.
     Gana el jugador que se queda sin cartas en su mano.
- En Python cree la clase Partida, que contenga las reglas del juego y lo coordine.
  Para barajar use `random.randint(0,N)` el cual entrega un entero al azar entre 0 y N.

```
1  import random # Biblioteca de numeros aleatorios
2  print(random.randint(0,N))
```

Polimorfismo significa, etimológicamente, *muchas formas* y en el caso de la orientación a objetos hace referencia a que diferentes sub-clases de una misma clase padre tiene distintos comportamientos para el mismo método.



En Python el polimorfismo se implementa mediante la sobrecarga de métodos, es decir, redefiniendo la función que define la forma de ejecutar el método sobrecargado en la sub-clase.

```
1  class NombrePadre:
2    def __init__(self, atr1,...):
3      self.atr1 = atr1
4    def metodoDelPadre(self,arg1,...):
5      Acciones1
6  class NombreHijo(NombrePadre):
7    def __init__(self, atr1,...):
8      super().__init__(atr1,atr2,...)
9    def metodoDelPadre(self,arg1,...):
10     Otras Acciones
```

Nota

Al sobrecargar un método se pierde su programación original.

Ejemplo (Clase Polígono)

```
1  class Poligono():
2      def __init__(self,lados):
3          self.lados = lados
4      def datos(self):
5          print('numero de lados:',self.lados)
6      def area(self):
7          pass
8
9  class Cuadrado(Poligono):
10     def __init__(self,lado):
11         super().__init__(2)
12         self.lado=lado
13         self.areaValor=self.area()
14     def area(self):
15         return self.lado*self.lado
16
17 cuadrado1=Cuadrado(1)
18 cuadrado1.datos()
19 print(cuadrado1.areaValor)
20
21 cuadrado2=Cuadrado(2)
22 cuadrado2.datos()
23 print(cuadrado2.areaValor)
```

Ejercicio

Implemente la clase triángulo y rectángulo.

También es posible sobrecargar operadores de python como $+, *,$

+	Adicion	__add__(self, other)	a1 + a2
-	Resta	__sub__(self, other)	a1 - a2
*	Multiplicacion	__mul__(self, other)	a1 * a2
@	Matrix Multiplication	__matmul__(self, other)	a1 @ a2
/	Division	__truediv__(self, other)	a1 / a2
%	Modulo	__mod__(self, other)	a1 % a2
**	Potencia	__pow__(self, other[, modulo])	a1 ** a2
&	Bitwise Y	__and__(self, other)	a1 & a2
^	Bitwise XOR	__xor__(self, other)	a1 ^ a2
|	(Bitwise OR)	__or__(self, other)	a1 | a2
-	Negacion (Aritmetica)	__neg__(self)	-a1
+	Positivo	__pos__(self)	+a1
~	Bitwise NO	__invert__(self)	~a1
<	Menos que	__lt__(self, other)	a1 < a2
<=	Menor que o igual a	__le__(self, other)	a1 <= a2
==	Igual a	__eq__(self, other)	a1 == a2
=!	No es igual a	__ne__(self, other)	a1= a2!
>	Mayor que	__gt__(self, other)	a1 > a2
>=	Mayor que o igual a	__ge__(self, other)	a1 >= a2
[index]	operador de indice	__getitem__(self, index)	a1[index]
in	En operador	__contains__(self, other)	a2 in a1

Ejercicio

1. Sobrecargue $+$ para que la suma de dos polígonos de como resultado la suma de sus áreas.
2. Sobrecargue $<=$ en la clase polígono para que retorne `True` si el área del primer polígono es menor o igual al área del segundo y retorne `False` en otro caso.
3. Sobrecargue `__str__` en la clase polígono para que retorne el string "el área del cuadrado es " AREA.
   Pruébelo con `print(cuadrado1)`, donde `cuadrado1` es un objeto de la clase Cuadrado.