POO en Python

Introducción a Python

José Miguel Gimeno

Características POO

- Clase
- Atributos
- Objeto
- Herencia
- Métodos

- Abstracción
- Encapsulamiento
- Polimorfismo
- Sobrecarga
- …

Clases en Python

La sintaxis general es:

class nombreClase:

Atributos en Python

A continuación y después del *docstring* se crean las variables de clase (atributos)

Métodos en Python

Son funciones que representan las acciones propias que se pueden realizar con el objeto.

Siempre tienen al menos el parámetro self en primer lugar, que hace referencia al propio objeto.

Método __init__()

Es un método especial que sirve para inicializar lo atributos del objeto.

- es el primer método en ejecutarse al crear un objeto
- es llamado siempre que se crea un objeto

Objetos

Para acceder a sus atributos:

objeto.atributo

Para acceder a sus métodos:

objeto.método()

Ejercicio

Desarrolla un programa que cargue los datos de un triángulo (los tres lados: a,b y c).

1.- Implementar una clase con los métodos para inicializar los atributos

2.- Un método para imprimir el valor de cada lado

3.- Un método para imprimir el tipo (equilátero, isósceles o escaleno).

4.- Un método que calcule su área

Herencia

La sintaxis general para definir una subclase:

class nombreSubClase (claseSuperior):

Para acceder a los métodos de la clase superior utilizamos **super()**

Herencia

En realidad, todas las clases en Python heredan jerárquicamente de una clase raíz denominada object. De forma que cuando ponemos:

```
class nombreClase():
```

Es lo mismo que:

```
class nombreClase(object):
```

Herencia múltiple

Añado las clases de las que hereda:

class nombreSubClase (clase1, clase2, ...):

Encapsulamiento

- De forma nominal añadiendo un comentario
- De forma activa añadiendo dos guiones bajos al principio del nombre del atributo o método

* En realidad no existen los atributos/métodos privados, nos lo podemos saltar indirectamente

@property, get, set y delete

- @property para definir el acceso al atributo (getter)
- @atributo.setter para poder actualizar el atributo
- @atributo.deleter para poder eliminar el atributo

Métodos especiales

__str__
__new__
@classmethod
@staticmethod

Métodos mágicos

P.ej. comparaciones y operadores matemáticos

Operator	Method	Expression
`+` Addition	`__add__(self, other)`	`a1 + a2`
`-` Subtraction	`__sub__(self, other)`	`a1 - a2`
`*` Multiplication	`__mul__(self, other)`	`a1 * a2`
`@` Matrix Multiplication	`__matmul__(self, other)`	`a1 @ a2` (*Python 3.5*)
`/` Division	`__div__(self, other)`	`a1 / a2` (*Python 2 only*)
`/` Division	`__truediv__(self, other)`	`a1 / a2` (*Python 3*)
`//` Floor Division	`__floordiv__(self, other)`	`a1 // a2`
`%` Modulo/Remainder	`__mod__(self, other)`	`a1 % a2`
`**` Power	`__pow__(self, other[, modulo])`	`a1 ** a2`
`<<` Bitwise Left Shift	`__lshift__(self, other)`	`a1 << a2`
`>>` Bitwise Right Shift	`__rshift__(self, other)`	`a1 >> a2`
`&` Bitwise AND	`__and__(self, other)`	`a1 & a2`
`^` Bitwise XOR	`__xor__(self, other)`	`a1 ^ a2`
`|` (Bitwise OR)	`__or__(self, other)`	`a1 | a2`
`-` Negation (Arithmetic)	`__neg__(self)`	`-a1`
`+` Positive	`__pos__(self)`	`+a1`
`~` Bitwise NOT	`__invert__(self)`	`~a1`
`<` Less than	`__lt__(self, other)`	`a1 < a2`
`<=` Less than or Equal to	`__le__(self, other)`	`a1 <= a2`
`==` Equal to	`__eq__(self, other)`	`a1 == a2`
`!=` Not Equal to	`__ne__(self, other)`	`a1 != a2`
`>` Greater than	`__gt__(self, other)`	`a1 > a2`
`>=` Greater than or Equal to	`__ge__(self, other)`	`a1 >= a2`
`[index]` Index operator	`__getitem__(self, index)`	`a1[index]`
`in` In operator	`__contains__(self, other)`	`a2 in a1`
`(*args, ...)` Calling	`__call__(self, *args, **kwargs)`	`a1(*args, **kwargs)`