

Programación Orientada a Objetos

(POO)

Fundamentos de Computación

Grado en Ingeniería en Tecnologías Industriales

Noviembre 2021

Universidad de Cantabria

La estructura de la Programación

- 1 En el nivel más básico está el código máquina.
 - Nosotros no programamos en código máquina.
- 2 Tenemos líneas de código en un lenguaje de programación (alto nivel)
 - Pero muchas veces usamos el mismo código.
- 3 Entonces construimos funciones.
 - Podemos reutilizar el código; a menudo estas funciones tienen uso concreto.
 - También hay funciones integradas en el lenguaje de programación.
- 4 Así podemos dividir nuestro código en Módulos.
 - Esto significa que no necesitamos evaluar masivamente cantidades de código cada vez que necesitamos obtener algo.
 - Los módulos tienen sus propias funciones.
 - Ayuda con la organización de programas.

La estructura de la Programación

- Un módulo es un fichero que contiene código PYTHON y su extensión es `<.py>`. Almacena declaración de variables e implementación de funciones. Posibilidad de hacer referencia a otros módulos (mediante la instrucción **import**)

```
$python triangulo.py
```

- Pueden ser ejecutados desde la consola (terminal), llamados por otros módulos o desde la consola PYTHON. Para ser ejecutados o llamados tienen que estar en el directorio actual o indicar la RUTA.

```
>>> import os
>>> os.getcwd()
'/home/usuariouc/Escritorio/turing/'
>>> from math import *
>>> cos(45)
0.5253219888177297
>>> from random import randint
randint(1, 2027)
1057
```

Conocemos una variedad de datos:

```
2027          3.14159          "Hola"          [2,3 5, 7, 11, 13]
```

- Cada uno de ellos es un objeto y cada objeto tiene:
 - 1 un tipo, una clase (**class**)
 - 2 una representación interna (escalar o no escalar)
 - 3 un conjunto de procedimientos para interactuar entre ellos.
- Un objeto es una **instancia** de una clase:
 - 2027 es una instancia de la clase **int**
 - "Hola" es una instancia de la clase **str**
- Un objeto es una abstracción de los datos(un ordenador, un libro,.. son objetos, ¿no?). En PYTHON todo es un objeto:

```
>>> isinstance(345., object)
True
>>> isinstance(int, object)
True
```

Hemos encontrado alguna instrucciones que se comportaba como una función pero cuya invocación se hacia de manera algo diferente:

```
>>> texto="santander"  
>>> texto.replace("s",'S')  
'Santander'
```

Un **método** es una función que se asocia a un determinado tipo de objetos. Es necesario llamar a los métodos desde un objeto concreto, que entrará a formar parte de los argumentos.

```
>>> a = complex(3,-2)  
>>> a.conjugate()  
(3+2j)
```

Del mismo modo que los métodos son funciones «propias» de un objeto, algunos objetos se componen de varios datos (escalares), que reciben el nombre de «atributos». Podemos acceder a los atributos de un objeto de manera similar a como llamamos a sus métodos, eliminando los paréntesis. En general, no se pueden modificar directamente los atributos de un objeto:

```
>>> a = complex(3,2)
>>> a.real
3.0
>>> a.real = 5
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: readonly attribute
>>> a = complex(5,2);a
(5+2j)
```

La Programación Orientada a Objetos (POO) permite definir nuevos tipos de datos, mediante una clase (**class**).

Esto resulta de utilidad cuando se quiere diseñar un programa que hace uso habitual de ese tipo de datos (pensemos en polinomios, grafos, pedidos de un negocio ...) mediante operaciones específicas (suma de polinomios, dibujo de grafos, gestión de pedidos...)

- Distinguiremos entre crear un clase y utilizar una instancia de la clase.
- Crear la clase involucra:
 - 1 definir el nombre y los atributos
 - 2 Por ejemplo, la clase COMPLEJO
- Usar la clase involucra:
 - 1 crear instancias de objetos y realizar operaciones con ellos.
 - 2 Por ejemplo, `a = complex(2,1)`, `a.real()`

Declaración de clases

Definimos una clase que permita manejar puntos en el plano:

```
class Punto(object):  
    '''Puntos en el plano'''  
    # Aquí definimos los atributos y métodos.
```

- Usamos la palabra reservada **class** para definir este nuevo tipo.
- El nombre(class) es **PUNTO**, y la clase padre es **object**.
- Similar a **def**, sangrar el código para indicar que sentencias son parte de la definición de la clase.
- La palabra **object** significa que **PUNTO** es un objeto PYTHON y que [hereda](#) todos los atributos y métodos.
 - **PUNTO** es una subclase de **object**
 - **Object** es una superclase de **PUNTO**.

```
>>> isinstance(Punto,object)  
True
```


Creación de instancias y atributos de la clase

Obviamente, la declaración del código anterior resulta de poca utilidad. Podemos crear objetos «miembros» de la clase y dotarles de atributos con el **constructor**

```
__init__
```

y que se ejecuta al crear un objeto. Pongamos:

```
class Punto(object):
    ''' Puntos en el plano '''
    def __init__(self,x0,y0):
        self.x = x0
        self.y = y0
```

- SELF: parámetro que para referirse a una instancia de la clase.
- X0,Y0: datos iniciales del objeto PUNTO.
- SELF.X, SELF.Y: dos atributos para cada objeto PUNTO.

Creación de instancias y atributos de la clase

Con este método, podemos crear objetos del modo siguiente:

```
>> P = Punto(1,-1)
>>> P.y
-1
>>> P.x += 99
>>> P.x
100
>>> type(P)
<class '__main__.Punto'>
>> Q = Punto("Hola",5)
>>> Q.x
'Hola'
>>> Q
<__main__.Punto object at 0x104dfbe48>
```

Creación de instancias y atributos de la clase

Podemos iniciar el objeto, por ejemplo en el origen (0,0) y «ocultar» atributos en los parámetros del constructor:

```
class Punto(object):
    '''Puntos en el plano '''
    def __init__(self, x0 = 0, y0 = 0):
        self.x = x0
        self.y = y0
        self.z = "autor:Nombre"
```

Si ejecutamos el programa:

```
>>> P = Punto()
>>> P.x
0
>>> P.z
'author=Nombre'
>>> P.z = "author = Marta"
>>> P.z
'author = Marta'
```

Declaración de métodos

Ahora definimos algunos métodos para hacer operaciones con los objetos de la clase:

```
import math
class Punto(object):
    '''Puntos en el plano '''

    def __init__(self,x0 = 0,y0 = 0):
        self.x = x0
        self.y = y0
        self.z = "author=Nombre"

    def norma(self):
        return math.sqrt(self.x**2 + self.y**2)
```

El método `norma` tiene un único argumento: el objeto desde el que se invoca:

```
>>> P = Punto(1,-1)
>>> P.norma()
1.4142135623730951
```

Declaración de métodos

Pero podemos definir métodos con más argumentos, siempre «dentro» de la declaración de la clase correspondiente:

```
class Punto(object):
    (...)

    def norma(self):
        return math.sqrt(self.x**2 + self.y**2)

    def distancia(self, otro):
        P = Punto(self.x - otro.x, self.y - otro.y)
        return P.norma()
```

Al invocar un método, el primer argumento es el objeto desde el que se llama y el resto se colocan entre paréntesis:

```
>>> P = Punto(1,-1)
>>> P = Punto(1,-1); Q = Punto(1,2)
>>> P.distancia(Q)
3.0
```

Declaración de métodos

O también, con la función `print` o sin la sentencia `return`.

```
class Punto(object):
    (...)
    def trasladar(self,t):
        self.x += t
        self.y += t

    def mostrar(self):
        print('{0},{1},{2}'.format(self.x,self.y,self.z))
===== RESTART: /Users/...
>>> P = Punto(1,-1)
>>> P.trasladar(5)
>> P.x; P.y
6
4
>>> print(P)
<__main__.Punto object at 0x1055fbda0>
>>> P.mostrar()
6,4,author = Nombre
```

Imprimir la representación de un objeto

Veamos ahora cómo controlar el comportamiento de nuestros objetos como argumentos de la instrucción `print`:

```
>>> P=Punto(1,2)
>>> print(P)
__main__.Punto object at 0x1055fbda0>
```

- La impresión por defecto no proporciona ninguna información.
- Definiremos el método `__str__` para la clase.
- Elegimos lo que queramos mostrar para el objeto `PUNTO`

```
def __str__(self):
    return "{0},{1}".format(self.x,self.y)
```

```
>>>print(Punto(5.1,-6))
>>>(5.1,-6)
```

Metodos/Operadores especiales

También podemos determinar a nuestro gusto el comportamiento de los operadores +, <:

```
def __add__(self,otro):  
    return Punto(self.x+otro.x,self.y+otro.y)  
  
def __lt__(self,otro):  
    return self.norma()<otro.norma()
```

```
H = Punto(4,5) + Punto(1,2)  
>>> H.x  
5  
>>> H.y  
7  
>>> P = Punto(1,2)  
>>> Q = Punto(6,7)  
>>> P < Q  
True
```


Metodos/Operadores especiales

Y muchos otros más operadores: -, *, ==, len(),....

[https:](https://docs.python.org/3/reference/datamodel.html#basic-customization)

[//docs.python.org/3/reference/datamodel.html#basic-customization](https://docs.python.org/3/reference/datamodel.html#basic-customization)

```
__sub__(self,otro)          ----> self - otro
__mul__(self,otro)         ----> self * otro
__eq__(self,otro)          ----> self == otro
__len__(self)              ----> len(self)
__iter__(self)             ----> iterar en los objetos
```

.
.
.
.

Paradigmas de programación: POO

- **Herencia:** Podemos crear nuevos datos(subclases) a partir de los ya existentes (clases) y se heredan sus atributos y métodos.
- **Polimorfismo:** Se dice de aquellas funciones que admiten distintos objetos, pero que se ejecuta la correspondiente a la suclase, pero no a la clase.
- **Encapsulamiento:** Los datos pertenecen al objeto y podemos ocultarles.

En el siguiente código construimos la clase `CIRCULO` que será una subclase de la clase `PUNTO`. La clase `CIRCULO` tendrá como parámetros el centro (un objeto de la clase `PUNTO`) y el radio un número de la clase `FLOAT`:

```
from Punto import*
class Circulo(Punto):
    def __init__(self, x0,y0,radio):
        self.centro = Punto(x0,y0)
        self.radio = radio
```

Herencia y Polimorfismo

Definimos la función `print` y el operador `+` en esta clase. Y, también el método `area` para calcular el área del círculo

```
def __str__(self):
    return "<{0},{1}>".format(self.centro,self.radio)

def __add__(self,otro):
    P=self.centro+otro.centro
    return Circulo(P.x,P.y, self.radio + otro.radio)

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

Si lo ejecutamos: `RESTART:/home/usuariouc/ ...`

```
>>> C1 = Circulo(1,2,3)
>>> C2 = Circulo(0,1,2)
>>> print(C1 + C2)
<(1,3),5>
>>> (C1+C2).area()
78.53981633974483
```