FUNDAMENTOS DE COMPUTACIÓN
Grado en ingeniería en tecnologías Industriales
Universidad de Cantabria

Práctica 11 (laboratorio)
12 de diciembre del 2018

Programación Orientada a Objetos (POO)

- Crear una carpeta en el escritorio denominada *practicas_python* y guardar todos los ejercicios y programas realizados durante la práctica en esa carpeta.
- Al finalizar la practica, subir todos los archivos generados durante la práctica a vuestro directorio en moodle y eliminar la carpeta *practicas_python* del escritorio.

1. 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.

1.1. Módulos

- 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**)

```
$python3 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()
'/usuariouc/Escritorio/PracticasPython'
>>> from math import *
>>> cos(45)
0.5253219888177297
>>>  from random import randint
randint(1, 2018)
1519
```

2. Objetos (objects)

Conocemos una variedad de datos:

```
2018        3.14159         "Hola"        [2,3 5, 7, 11, 13]
```

- Cada uno de ellos es un objeto y cada objeto tiene:
  1. un tipo (**type**)
  2. una representación interna (escalar o no escalar)
  3. un conjunto de procedimientos para interactuar entre ellos.
- Un objeto es una instancia de un tipo:
  - 2018 es una instancia de tipo **int**
  - "*Hola*" es una instancia de tipo **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
```

3. Métodos

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 entrara formar parte de los argumentos.

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

4. Atributos

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)
```

5. Clases y objetos

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 de **complex**
- Usar la clase involucra:
  1. crear instancias de objetos y realizar operaciones con ellos.
  2. Por ejemplo, a=complex(2,1), a.conjugate()

6. Declaración de clases

Definimos una clase que permita manejar puntos en el plano:

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

- Usamos la palabra reservada **class** para definir este nuevo tipo.
- El nombre(tipo) 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
  - PUNTO es una subclase de **object**
  - **Object** es una superclase de PUNTO.

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

7. Creación de instancias y atributos de la clase

Obviamente, la declaración del código anterior resulta de poca utilidad. Podemos crear objetos «miembros» de los 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 par cada objeto Punto.

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>
```

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'
```

8. 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
```

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
```

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
```

9. 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)
```

Si lo ejecutamos:

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

10. 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()
```

Ejecutando:

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

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

https://docs.python.org/3/reference/datamodel.htmlbasic-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
(.....)
```

11. 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 subclase, pero no a la clase.
- **Encapsulamiento:** Los datos pertenecen al objeto y podemos ocultarles.

11.1. Herencia

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
```

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

```
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
```

12. Ejercicios

1.
   - Descarga el fichero `punto.py` y ejecuta la clase PUNTO y la subclase CIRCULO explicadas en las secciones anteriores, e invoca a todos los métodos.
   - Observar como se **herendan** los atributos y métodos de la subclase Çirculo de la clase PUNTO.
   - Implementar el método especial `__eq__`, para que devuelva el booleano TRUE si los dos objetos son iguales o FALSE en caso contrario en las dos clases (PUNTO y CIRCULO):

     ```
     def __eq__ (self, otro):
     (...)
     ```

     Al ejecutarlo:

     ```
     >>> P = Punto(3,4)
     >>> Q = Punto(1,2); R=(3,4)
     >>> P == Q
     False
     >>> P == R
     True
     >>> C1 = Circulo(1,2,3)
     >>> C2 = Circulo(0,1,3)
     >>> C1 == C2
     False
     ```

2. Crear una clase de nombre **EmpresaContabilidad** y que tenga como único atributo un número de tipo FLOAT denominado `presupuesto` con una cantidad inicial de 1024.1 euros. Y dos métodos

   - `ingresos`: con un parámetro de tipo FLOAT para añadir al `presupuesto`.
   - `gastos`: con un parámetro para restar al `presupuesto`.

   Utiliza esta clase para implementar un programa con `ingresos` de 2017.1, 150.42 y 35 euros y, luego `gastos` de 252.25 euros. Mostrar por pantalla cada operación y el `presupuesto`.

```
 Contabilidad de comienzo: presupuesto =  1024.1
Ventas consumidores: presupuesto =  3041.2
Ventas consumidores: presupuesto =  3191.62
Ventas consumidores: presupuesto =  3226.62
Gastos compra: presupuesto =  2974.37
```

3. Crear una clase llamada **Consumidor** y que tenga como atributos el nombre, DNI, dirección, telefóno y e-mail(este último opcional) de una persona. Y el método `mostrar` que imprimirá por pantalla los atributos de cada consumidor. Utilizar esa clase para crear una lista de consumidores y mostrar cada uno de ellos. Los datos de cada consumidor están en el fichero *consumidores.txt*. El formato de este fichero es una linea con los datos de cada persona y separados por ';'. Contiene información por este orden:

$$nombre; DNI; direccion; telefono; e-mail; presupuesto$$

consumidores.txt

```
Claude E. Shannon; 12345Z; Unican01; 94234561; clau@gmail.campoo; 123.5
George Boole; 12545A; Unican20; 94534561; boole@gmail.campoo; 623.1
Bertrand Russell; 672345Z;Unican31; 84234561;russell@gmail.campoo; 108
```

   Al ejecutarlo podría parecerse a esto:

```
Nombre =  Claude E. Shannon
DNI =   12345Z
Direccion =   Unican01
Telefono =  94234561
E-mail =   clau@gmail.campoo
--------------------
Nombre =  George Boole
DNI =   12545A
Direccion =   Unican20
Telefono =   94534561
E-mail =   boole@gmail.campoo
--------------------
Nombre =  Bertrand Russell
DNI =   672345Z
Direccion =  Unican31
Telefono =   84234561
E-mail =  russell@gmail.campoo
--------------------
```

4. Modificar la clase del Ejercicio 2, haciendo que el atributo `presupuesto` este oculto y para que cuando alguno quiera leer este atributo deba escribir un número secreto. Este valor se introducirá como parámetro de la función `leer_presupuesto` y que se añadirá como método de la clase. El número secreto para realizar la operación es 2222. Utiliza esta nueva clase para implementar un programa con `ingresos` de 2017.1, 150.42 y 35 euros y, luego `gastos` de 252.25 euros. Mostrar por pantalla cada operación y el `presupuesto`. y el método nuevo `leer_presupuesto.`

```
Dame el numero secreto:  2222
Contabilidad de comienzo: presupuesto =  1024.1
Ventas consumidores: presupuesto =  3041.2
Ventas consumidores: presupuesto =  3191.62
```

```
Ventas consumidores: presupuesto =  3226.62
Gastos compra: presupuesto =  2974.37
```

---

5. El objetivo de este ejercicio es manipular los números racionales (de forma exacta) , es decir, cocientes de enteros `numerador` y `denominador`. Como sabemos Python convierte los números racionales a "decimales", es decir de tipo FLOAT . Y ocurren, situaciones como la siguiente

---

```
>>> 1/10 + 1/10 + 1/10
0.30000000000000004
```

Pero las matemáticas, dicen que

$$1/10 + 1/10 + 1/10 = 3/10$$

- Crear una clase llamada FRACCIONES con dos atributos `numerador` y `denominador` y con varios métodos: sumar, multiplicar, dividir, simplificar (utilizando el máximo común divisor y el mínimo común múltiplo de los denominadores), restar,...etc de dos fracciones. Adaptando cuando sea posible estas operaciones con los métodos especiales:

  ```
  __eq__(self,otro)
  __sub__(self,otro)
  __mul__(self,otro)
  __eq__(self,otro)
  __add__(self,otro)
  etc...
  ```

- Construir para cada primo p una familia de subclases denominada FRACIONES_P-ADICAS de la clase FRACCIONES y que tengan como atributo un numerador, el primo p, y el denominador una potencia entera de p. Por ejemplo, si $p = 2$, los objetos de esta clase serán de la forma:

$$1/2, -45/256, 67/1024, ..., -5/2^{30}, ...$$

```
----------------------------------------
 Bienvenidos a mi clase de Fracciones
----------------------------------------
>>> a = Fraccion(1,10)
>>> b = a + a + a
>>> print(b)
3/10
>>>
```
