Programación Orientada a Objeto en Python

Objetos

- Python provee distintos tipos de datos:
- enteros: 1, de punto flotante: .3.14, texto: ”prueba”, listas: [1,2,3], tuplas: (”a”,1), diccionarios: {1:”a”,2:”b”}…
- Todos estos son **objetos**
- En python todo es **objeto**

Objetos

- Todo **objeto** tiene al menos un **tipo**
- En python se pueden crear **objetos** de algún **tipo**
- Estos **objetos** se pueden **manipular**
- Estos **objetos** se pueden **destruir**
  - Explícitamente (usando el comando **del**)
  - Implícitamente (con el mecanismo de **recolección de basura**)

Objetos

- Los **objetos** son abstracciones de datos
  - Poseen una **representación interna** (a través **atributos de datos**)
  - Poseen una interfaz para interactuar con ellos (a través de atributos de procedimiento [**métodos**])
- Encapsulan tanto la **representación interna** como la implementación de los **métodos**

Porqué programar Orientado a Objetos

- Agrupar en paquetes funcionalidades y abstracciones de datos con interfaces bien definidas
- Desarrollo modular que reduce la complejidad y permite implementar y verificar de manera separada
- Se puede reusar el código. Cada paquete define clases, cada clase tiene un espacio diferente, la **herencia** permite **redefinir** o **extender** nuevas funcionalidades o comportamientos

Clases e Instancias

- Una **clase** define un tipo de objeto, una **instancia** de una **clase** es un **objeto**
- Crear una clase involucra:
  - Definir el nombre de la clase
  - Definir los atributos de datos de la clase
  - Definir los métodos de la clase
- Usar una clase involucra:
  - Crear una instancia de la clase, un objeto
  - Realizar operaciones con el objeto, manipular su estado interno

Definir una clase

```
class Product(object):
    #implementación
```

- **class** es la palabra clave para definir una clase (como **def** lo es para definir una función)
- Product es el nombre de la nueva clase
- **object** es la clase de la que **hereda** Product. **object** es la clase base de toda la jerarquía de python
  - Product es una subclase de **object**
  - **object** es la superclase de Product

Atributos

- **atributos de datos**
  - Los datos que conforman la representación interna de la clase (pueden ser otros **objetos**)
- **métodos**
  - Las funciones, propias de esta clase de objetos, que determinan como interactuar con este tipo de objetos

De la Clase a la Instancia

- Primero se debe definir cómo crear una instancia. Para ello se utiliza el método __**init**__

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

De la Clase a la Instancia

```
C = Coordinate(5,7)

origin =  Coordinate(0,0)

print(c.x)

print(origin.y)
```

- Note que el parámetro **self** no hay que pasarlo, python lo hace automáticamente

Qué es un **método**

- Se pueden tomar como funciones que solo funcionan con la clase en la que se definen
- Python pasa al mismo objeto de la instancia, **self**, como el nombre del primer argumento de todos los métodos
- El operador “**.**” es usado para acceder a cualquier atributo
  - Atributos de datos (también llamados **variables de instancia**)
  - Métodos

Métodos

```
class Coordinate(object):
    def __init__(self,x,y):
        self.x = x
        self.y = y
    def distance(self,other):
        x_diff_2 = (self.x – other.x)**2
        y_diff_2 = (self.y – other.y)**2
        return (x_diff_2 + y_diff_2)**0.5
```

Métodos

```
c1 = Coordinate(0,0)
c2 = Coordinate(3,4)
c1.distance(c2)
c2.distance(c1)
#Otra manera
Coordinate(c1,c2)
```

Métodos heredados

- Los métodos heredados se pueden sobreescribir
- Ejemplo, la representación en texto de un objeto __**str**__

  def __str__(**self**):

  return ”<”+**self**.x+”,”+**self**.y+”>”
- Otros métodos: __**add**__, __**sub**__,__**eq**__,etc