Programación Orientada a Objetos en Python

La programación orientada a objetos nos propone un paradigma de programación.

Paradigma: teoría cuyo núcleo central [...] suministra la base y modelo para resolver problemas [...] (Definición de la Real Academia Española, vigésimo tercera edición)

Los elementos de la POO, pueden entenderse como los materiales que necesitamos para diseñar y programar un sistema, mientras que las características, podrían asumirse como las herramientas de las cuáles disponemos para construir el sistema con esos materiales.

Clases

Las clases surgen de la generalización de los tipos de datos y permiten una representación más directa para la modelización de un problema permitiendo definir nuevos tipos al usuario.

Las clases permiten agrupar en un nuevo tipo los datos y las funcionalidades asociadas a dichos datos, favoreciendo la separación entre los detalles de la implementación de las propiedades esenciales para su uso. A esta cualidad, de no mostrar más que la información relevante, ocultando el estado y los métodos internos de la clase, es conocida como “encapsulación”, y es un principio heredado de la programación modular.

Las clases son los modelos sobre los cuáles se construirán nuestros objetos. Podemos tomar como ejemplo de clases, el gráfico que hicimos en la página 8 de este documento.

En Python, una clase se define con la instrucción class seguida de un nombre genérico para el objeto.

En python, el esquema para la definición de una clase es el siguiente:

```
class NombreClase:
  <instrucción_1>
  ...
  <instrucción_n>
```

En donde **class** es una palabra reservada que indica la declaración de una clase, **NombreClase** una etiqueta que da nombre a la clase y, los dos puntos, que señalan el inicio del bloque de instrucciones de la clase.

```
class Objeto:
    pass


class Antena:
    pass


class Pelo:
    pass


class Ojo:
    pass
```

Propiedades

Las propiedades, como hemos visto antes, son las características intrínsecas del objeto. Éstas, se representan a modo de variables, solo que técnicamente, pasan a denominarse propiedades:

```
class Antena():
    color = ""
    longitud = ""


class Pelo():
    color = ""
    textura = ""


class Ojo():
    forma = ""
    color = ""
    tamanio = ""
```

```
class Objeto():

    color = ""

    tamanio = ""

    aspecto = ""

    antenas = Antena() # propiedad compuesta por el objeto objeto Antena

    ojos = Ojo()       # propiedad compuesta por el objeto objeto Ojo

    pelos = Pelo()     # propiedad compuesta por el objeto objeto Pelo
```

Métodos

Los métodos son *funciones* (como las que vimos en el capítulo anterior), solo que técnicamente se denominan métodos, y representan acciones propias que puede realizar el objeto (y no otro):

```
class Objeto():

    color = "verde"

    tamanio = "grande"

    aspecto = "feo"

    antenas = Antena()

    ojos = Ojo()

    pelos = Pelo()


    def flotar(self):

        pass
```

Objeto

Las clases por sí mismas, no son más que modelos que nos servirán para crear objetos en concreto. Podemos decir que una clase, es el razonamiento abstracto de un objeto, mientras que el objeto, es su materialización. A la acción de crear objetos, se la denomina instanciar una clase y dicha instancia, consiste en asignar la clase, como valor a una variable:

```
class Objeto():

    color = "verde"

    tamanio = "grande"

    aspecto = "feo"

    antenas = Antena()

    ojos = Ojo()

    pelos = Pelo()


    def flotar(self):

        print 12


et = Objeto()

print et.color

print et.tamanio

print et.aspecto

et.color = "rosa"

print et.color
```

Herencia: característica principal de la POO

Como comentamos en el título anterior, algunos objetos comparten las mismas propiedades y métodos que otro objeto, y además agregan nuevas propiedades y métodos. A esto se lo denomina herencia: una clase que hereda de otra. Vale aclarar, que en Python, cuando una clase no hereda de ninguna otra, debe hacerse heredar de object, que es la clase principal de Python, que define un objeto.

```
class Antena(object):
    color = ""
    longitud = ""

class Pelo(object):
    color = ""
    textura = ""

class Ojo(object):
    forma = ""
    color = ""
    tamanio = ""

class Objeto(object):
    color = ""
    tamanio = ""
    aspecto = ""
    antenas = Antena()
    ojos = Ojo()
    pelos = Pelo()

    def flotar(self):
        pass

class Dedo(object):
    longitud = ""
```

```
    forma = ""
    color = ""


class Pie(object):
    forma = ""
    color = ""
    dedos = Dedo()


# NuevoObjeto sí hereda de otra clase: Objeto
class NuevoObjeto(Objeto):
    pie = Pie()


    def saltar(self):
        pass
```

Accediendo a los métodos y propiedades de un objeto

Una vez creado un objeto, es decir, una vez hecha la instancia de clase, es posible acceder a sus métodos y propiedades. Para ello, Python utiliza una sintaxis muy simple: el nombre del objeto, seguido de punto y la propiedad o método al cuál se desea acceder:

```
objeto = MiClase()
print objeto.propiedad
objeto.otra_propiedad = "Nuevo valor"
variable = objeto.metodo()
print variable
print objeto.otro_metodo()
```

Función __init__

La función __init__ que posee python.

Esta función siempre se ejecuta cuando se crea un objeto. Ahora mostraremos como es la estructura de una clase y como funciona lo que hemos explicado

```
1
2     class A:
3         def __init__(self):
4             print 'me ejecuto primero'
5
6         def funcion(self):
7             print 'soy una funcion.!!'
8
```

Ya tenemos la clase **A** donde hemos declarado dos funciones las cuales son opcionales. Ahora veremos que sucede cuando creamos una instancia de esta clase. Cabe resaltar que que una instancia es un objeto.

```
1     instancia = A()
2     # me ejecuto primero
```

Al crear la instancia de la clase **A** visualizamos que se ejecuta automáticamente el __init__ y por ende se visualiza en la consola el texto “me ejecuto primero”. Si imprimimos la instancia por si sola. Se mostrará la clase a la cual pertenece dicha instancia y donde se encuentra almacenado en memoria.

```
1     print instancia
2     # <__main__.A instance at 0x7f5942210320>
```

Ahora que tenemos la instancia somos capaces de llamar a las funciones que pertenecen a esta clase.

```
instancia.funcion()
soy una funcion.!!
```

Ahora procedamos a crear la clase Gato. Esta función gato tendrá el atributo energía y hambre para hacerlo más interesante.

```
class Gato:
    def __init__(self, energia, hambre):
        self.energia = energia
        self.hambre = hambre
        print 'Se creo un gato'

    def tomar_leche(self, leche_en_litros):
        self.hambre += leche_en_litros
        print 'el gato toma su leche'

    def acariciar(self):
        print 'prrrrr...'

    def jugar(self):
        if self.energia <= 0 or self.hambre <=1:
            print 'el gato no quiero jugar'
        else:
            self.energia -=1
            self.hambre -= 2
            print 'al gato le encanta jugar'

    def dormir(self, horas):
        self.energia += horas
        print 'el gato tomo una siesta'
```

Gracias al atributo “self” es posible utilizar las variables declaradas en **__init__** dentro de toda la clase.

Referencias:

http://frontendlabs.io/1305--tutorial-basico-de-python-parte-iv-programacion-orientada-a-objetos

http://blog.rvburke.com/2006/11/22/programacion-orientada-a-objetos-en-python/comment-page-1/

https://librosweb.es/libro/python/capitulo_5/programacion_orientada_a_objetos.html

https://librosweb.es/libro/python/capitulo_6/metodos_de_formato.html