Clases y Objetos en Python

Hugo Araya Carrasco

Las Bases Conceptuales

- Antes de ver clases, primero se debe decir algo acerca de las reglas de ámbito de Python.
- Las definiciones de clases hacen trucos con los espacios de nombres, y se necesita saber como funcionan los alcances y espacios de nombres
- De paso, los conocimientos en este tema son útiles para cualquier programador Python avanzado.

- Un *espacio de nombres* es una relación de nombres a objetos.
- Muchos espacios de nombres están implementados en este momento como diccionarios de Python, pero eso no se nota (excepto por el desempeño).
- Como ejemplos de espacios de nombres se tienen: los nombres globales en un módulo; y los nombres locales en la invocación a una función.
- No hay relación entre los nombres de espacios de nombres distintos; por ejemplo, dos módulos diferentes pueden tener definidos los dos una función maximizar sin confusión; los usuarios de los módulos deben usar el nombre del módulo como prefijo.

- Generalmente se usa la palabra ***atributo*** para cualquier cosa después de un punto; por ejemplo, en la expresión **z.real**, real es un atributo del objeto z.
- Estrictamente hablando, las referencias a nombres en módulos son **referencias** a atributos: en la expresión ***modulo.funcion***, módulo es un objeto módulo y función es un atributo de éste.
- En este caso hay una relación directa entre los atributos del módulo y los nombres globales definidos en el módulo: ¡están compartiendo el mismo espacio de nombres!.

Introducción

- La programación orientada a objetos se basa en la programación de clases; a diferencia de la programación estructurada, que está centrada en las funciones.
- Una clase es una plantilla del que luego se pueden crear múltiples objetos, con similares características.
- Una clase define atributos (lo que conocemos como variables) y métodos (lo que conocemos como funciones).
- La clase define los atributos y métodos comunes a los objetos de ese tipo, pero luego, cada objeto tendrá sus propios valores y compartirán las mismas funciones.

Introducción

- Se debe crear una clase antes de poder crear objetos (**instancias**) de esa clase. Al crear un objeto de una clase, se dice que se crea una instancia de la clase o un objeto propiamente dicho.
- Se confeccionará una clase para conocer la sintaxis en el lenguaje Python, luego se definirán dos objetos de dicha clase.
- Como ejemplo se implementará una clase llamada Persona que tendrá como atributo (variable) su nombre y dos métodos (funciones), uno de dichos métodos se llama el constructor de la clase y el siguiente método mostrará el contenido del mismo.

Ejemplo

```
class Persona:
    def __init__(self, nom):
        self.nombre = nom

    def imprimir(self):
        print ('Nombre:', self.nombre)

persona1 = Persona(”Hugo")
persona1.imprimir()

persona2 = Persona(”Araya")
persona2.imprimir()
```

- Conviene buscar un nombre de clase lo más próximo a lo que representa. La palabra clave para declarar la clase es **class**, seguida por el nombre de la clase y luego dos puntos.
- Los métodos de una clase se definen utilizando la misma sintaxis que para la definición de funciones.
- Todo método tiene como primer parámetro el identificador **self** que tiene la referencia (dirección de memoria) del objeto que llamó al método.
- Luego dentro del método diferenciamos los atributos del objeto antecediendo el identificador **self**:

self.nombre=nom

- La operación de instanciación crea un objeto vacío. Muchas clases necesitan crear objetos con instancias en un estado inicial particular. Por lo tanto una clase puede definir un método especial llamado **__init__()**, de esta forma:

```
def __init__(self):
  self.datos = []
```

- Cuando una clase define un método **__init__()**, la instanciación de la clase automáticamente invoca a **__init__()** para la instancia recién creada.

```
class Bolsa:
    def __init__(self):
        self.datos = []
    def agregar(self, x):
        self.datos.append(x)
    def dobleagregar(self, x):
        self.agregar(x)
        self.agregar(x)
```

Métodos Especiales

- Las clases en Python cuentan con múltiples métodos especiales, los cuales se encuentran entre dobles guiones bajos `__<metodo>__()`.

Los métodos especiales más utilizados son

__init__(),

__str__() **y**

__del__().

- El método ***__str__()*** es un método especial, el cual se ejecuta al momento en el cual un objeto se manda a mostrar, es decir es una cadena representativa de la clase, la cual puede incluir formatos personalizados de presentación del mismo.

```
def __str__(self):

    return str(self.cedula) + self.nombre,
self.apellido + self.getGenero(self.sexo)
```

- El método **`__del__()`** es un método especial, el cual se ejecuta al momento en el cual un objeto es descartado por el intérprete.
- El comportamiento de **`__del__()`** es muy similar a los “destructores” en otros lenguajes

```
def __add__(self, otro):
        """ Devuelve la suma de ambos puntos. """
        return Punto(self.x + otro.x, self.y + otro.y)
def __sub__(self, otro):
        """ Devuelve la resta de ambos puntos. """
        return Punto(self.x - otro.x, self.y - otro.y)
```

```
__add__( self, other)
__sub__( self, other)
__mul__( self, other)
__rmul__( self, other)
__floordiv__( self, other)
__mod__( self, other)
__divmod__( self, other)
__pow__( self, other[, modulo])
__and__( self, other)
__xor__( self, other)
__or__( self, other)
```

Las variables "***privadas***" de instancia, que solo se pueden accederse desde dentro de un objeto, no existen en Python.

Sin embargo, hay una convención que se sigue en la mayoría del código Python:

"un nombre prefijado con un guion bajo (por ejemplo, _spam) debería tratarse como una parte no pública del objeto, mas allá de que sea una función, un método o un dato)."