Tema 4: Objetos

Héctor Xavier Limón Riaño

April 3, 2024

Contents

1 Conceptos generales

- Los objetos son el tipo de dato principal del paradigma de programación orientado a objetos (POO)
- En este curso no se verán muchos aspectos de este paradigma (para eso está el curso de lenguajes y paradigmas de programación)
- Sólo se abordaran objetos como estructura de datos
- Se cubrirán los siguientes conceptos:
  - Clases
  - Objetos
  - Propiedades
  - Métodos

2 Clases

- Son plantillas para crear objetos
- Varios objetos pueden ser de la misma clase
- Una clase define un tipo de dato, es la forma que tienen los programadores de crear sus propios tipos
- Python implementa sus tipos de datos mediante clases (muchas cosas en Python son objetos)
- Son elementos estáticos, esto es, no son `instancias` (no están siendo administradas por el proceso)
- Una práctica común es nombrar las clases con una mayúscula al inicio, así se distinguen de otro tipo de identificadores (como variables, constantes o nombres de funciones)

```
  # forma
class Nombre_Clase():
    # se necesita un método especial __init__
    def __init__(self):
        pass
```

- `self` es una referencia especial hacia el propio objeto (se verá más adelante junto con el método `__init__`)

3 Objetos

- Son `instancias` de una clase, administradas por el proceso
- Esencialmente son contenedores de valores (atributos/propiedades) y pueden tener un comportamiento (métodos)
- Pueden cambiar de `estado`, esto es, sus propiedades pueden cambiar de valor

```
class Ejemplo:
    # se necesita un método especial __init__
    def __init__(self):
        pass
```

```
objeto1 = Ejemplo() # instanciación de clase
objeto2 = Ejemplo() # otro objeto de la misma clase
print(type(objeto1))
```

```
<class '__main__.Ejemplo'>
```

4 Atributos

- También se les puede llamar propiedades
- Son similares a las variables, pero sólo existen dentro del objeto
- Se pueden crear desde el método especial `__init__`
- Este método es el que se invoca implícitamente al crear el objeto (instanciarlo)
- Para recuperar el valor de una propiedad a partir de un objeto, se utiliza el operador .
- Una práctica común es que los parámetros de `__init__` se

. correspondan en nombre con el de las propiedades

- Dentro de la clase, para hacer referencia a una propiedad se utiliza la referencia especial `self`
- Al conjunto de valores de las propiedades de un objeto en un momento dado se le llama `estado` del objeto
- Notar que los objetos pueden ser estáticos/dinámicos o mutables/no mutables dependiendo del tipo de sus propiedades
- Dada su flexibilidad (y métodos especiales de python) los objetos pueden comportarse como lineales, no lineales o jerárquicos (dependiendo de lo que quiera el programador)

```
class Persona():
    def __init__(self, nombre, edad):
        self.nombre = nombre # son variables diferentes
        self.edad = edad
```

```
pepe = Persona('Pepe', 12) # se pasan los parámetros en el orden de __init__
pepe.edad = 22
brenda = Persona('Brenda', 20)
print(pepe.nombre)
print(brenda.edad)
print(pepe.edad)
print(type(pepe))
print(pepe)
```

```
Pepe
20
22
<class '__main__.Persona'>
<__main__.Persona object at 0x76fbdcd1c8d0>
```

5 Métodos

- Son similares a las funciones, pero en el contexto de un objeto
- Esto es, tienen acceso directo a las propiedades (a través de `self`)
- En un método, el primer atributo debe ser la referencia especial `self`, incluso si el método no recibe parámetros se debe poner
- Al invocar el método desde un objeto, no se pone el `self` (va implícito, es el propio objeto que invoca)
- Existen métodos especiales que se pueden definir, por ejemplo para establecer qué hacer con operadores com +, qué cadena regresar cuando se pone el objeto en un `print`, o cómo determinar si dos objetos son iguales o diferentes cuando se usan operadores de igualdad, entre otros (no se entrará mucho en detalle pero se pueden hacer varias cosas)

```
class Fraccion():
    def __init__(self, numerador, denominador):
        self.numerador = numerador
        self.denominador = denominador

    def sumar1(self):
        self.numerador += 1
```

```
    # función que se invoca al hacer print del objeto
    def __repr__(self):
        return '%s/%s' % (self.numerador, self.denominador)

    # suma la fracción actual y otra y regresa la suma
    def sumar_fracciones(self, otra_fraccion):
        comun_denominador = self.denominador * otra_fraccion.denominador
        nuevo_numerador = (self.numerador * otra_fraccion.denominador) + (otra_fraccion
        return Fraccion(nuevo_numerador, comun_denominador)

f1 = Fraccion(1, 2)
print(f1)
f1.sumar1()
print(f1)
f2 = Fraccion(1, 4)
print(f2)
nueva = f1.sumar_fracciones(f2)
print(nueva)
```

```
1/2
2/2
1/4
10/8
```

6 Igualdad en objetos

- Ten cuidado al comparar objetos de clases que creaste, puedes tener resultados inesperados

```
class Ejemplo():
    def __init__(self, valor):
        self.valor = valor

ob1 = Ejemplo(1)
ob2 = ob1
ob3 = Ejemplo(1)

print(ob1 == ob2)
print(ob1 == ob3)
```

```
True
False
```

- Al comparar objetos entre si, y no haber implementado métodos especiales como `__eq__`, sólo se compara memoria, esto es, solo que sea el mismo objeto en memoria el resultado es verdadero
- Normalmente los tipos de datos oficiales de Python, y de terceros toman esto en cuenta, por lo que no hay que preocuparse

```
l1 = [1, 2, 3]
l2 = [1, 2, 3]

print(l1 == l2)

s1 = {5, 7, 7, 8, 5} # tipo set
s2 = {8, 7, 5}
print(s1 == s2)
```

```
True
True
```

```
class Ejemplo():
    def __init__(self, valor):
        self.valor = valor

    def __eq__(self, otro_objeto):
        if not isinstance(otro_objeto, Ejemplo):
            False
        return self.valor == otro_objeto.valor

ob1 = Ejemplo(1)
ob2 = ob1
ob3 = Ejemplo(1)

print(ob1 == ob2)
print(ob1 == ob3)
```

```
True
True
```

- La función `isinstance` recibe un objeto y una clase, determina si el objeto pertenece a la clase (es una instancia)