oop

January 28, 2020

1 Programación Orientada a Objetos

La programacion orientada a objetos (OOP, por sus siglas en ingles) es un metodo utilizado para estructurar un programa a fin de reunir propiedades y comportamientos relaciones en **objetos** individuales.

Conceptualmente, los objetos son como componentes de un sistema. Consideremos que un programa es como una linea de montaje de fabrica y que cada componente del sistema procesa algun material en cada paso del proceso, poco a poco, concluyendo con el producto final.

Un objeto contiene data, asi como los materiales pre-procesados o materia prima en cada etapa de la linea de montaje, y comportamiento, como la accion de cada componente en cada estapa del proceso.

En esta seccion, aprenderemos a: * Crear una clase con `class`, que es como un plano de prototipo * Utilizar clases para crear nuevos objetos * Modelar sistemas con herencia de clase

1.1 Definicion de una Clase

Las estructuras primitivas de datos, como los numeros, strings y listas, estan diseñados para representar cosas simples, como el costo de una cosa, el nombre de un poema, colores favoritos, entre otras cosas. No obstante, qué tal si queremos representar algo un poco mas complicado?

Por ejemplo, digamos que queremos darle seguimiento a todos los empleados en una organizacion. Necesitamos guardar informacion basica de cada empleado como nombre, edad, posicion y fecha de inicio de labores.

[2]:
```
# una forma de hacerlo
juan = ['Juan Perez', 31, 'Asistente Administrativo', 1001]
```

[3]:
```
# cuando no estamos cerca del codigo anterior, es dificil saber a que␣
↪referenciamos con:
juan[1]  # edad
```

[3]:
```
31
```

[5]:
```
# otro problema es que si creamos otro registro
juana = ['Juana Perez', 'Asistente Ejecutiva', 1002]  # sin la edad
juana[1]  # ya no es edad, sino posicion
```

[5]:
```
'Asistente Ejecutiva'
```

1.2 Clases vs Instancias

Las clases son utilizadas para crear estructuras de datos definidas por el usuario. Las clases tambien tienen funciones especiales, llamadas **metodos**, que definen comportamiento y acciones que un objeto de esa clasa puede realizar con su data.

Importante notar que una clase solo proporciona estructura. Una clase es como un plano que contiene el diseño o especificaciones de como algo debe ser definido. Por ejemplo, una clase `Canino` puede especificar que el nombre y edad son necesarios para definir un perro, pero no especificará como el nombre debe llamarse o que edad debe tener.

Una clase es un plano, y una instancia de un objeto es un prototipo construido a base de ese plano y que contiene data real. Por ejemplo, una instancia de un `Canino` ya no es un plano, sino una representacion de un canino, con un nombre de `Rocky` y que tienes `4` años.

Es decir, una clase es como un formulario o un cuestionario, toda vez que define la información requerida. Después de llenar el formulario, esa copia específica del formulario llenado, es una instancia de una clase, toda vez que contiene informacion actual relevante.

En efecto, podemos llenar multiples copias del formulario para crear multiples instancias, pero sin el formulario como guia, no sabremos cual información es requerida. Por tanto, antes de crear instancias individuales de un objeto, primero necesitamos especificar que se necesita para definir una clase.

1.3 Como definir una clase

Toda definición de clase empieza con la palabra `class`, la cual es seguida por el nombre de la clase y un color. Esto guarda similutd con la firma de una función, excepto no necesitamos agregar parametros en parentesis. Cualquier codigo que esta indentado abajo de la definicion de la clase es considerada parte del cuerpo de la clase.

[8]:
```
class Canino:
    pass  # esta palabra hace el efecto de ignorar lo seguido, sirve como␣
↪plantilla, permite correr sin error
```

[9]:
```
class Canino:  # sin pass
```

```
  File "<ipython-input-9-21297a748627>", line 1
    class Canino:  # sin pass
                              ^
SyntaxError: unexpected EOF while parsing
```

La clase `Canino` debería contener un conjunto de propiedades particulares a un canino como nombre, edad, color, y raza, sin embargo para mantener las cosas simples vamos a utilizar nombre y edad por ahora.

Para definir propiedades, conocidas como **atributos de instancia**, que todos los objetos `Canino` deberan tener, tenemos que definir un metodo especial llamado `.__init__()`, que hace la inicializacion del objeto. Este metodo es especial porque le dice a Python el **estado** inicial del objeto, es decir, los valores iniciales de las propiedades del objeto.

El primero parametro / argumento posicional de `.__init__()` siempre es una variable que hace referencia a la instancia de clase misma. Esta variable es universalmente denominada `self`. Despues del argumento `self`, podemos especificar los argumentos requeridos para crear una instancia de una clase.

[12]:
```
class Canino:
    def __init__(self, nombre, edad):
        self.nombre = nombre  # atributo de instancia nombre se le asigna la␣
↪variable nombre
        self.edad = edad  # atributo de instancia edad se le asigna la variable␣
↪edad
```

[15]:
```
# self.atributo es particular a una instancia, pero un atributo de clase es de␣
↪todas las instancias
class Perro:
    especie = 'Canino'

    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad
```

Los atributos de clase se definen directamente abajo de la definicion de la clase y antes de cualquier metodo. Deberiamos utilizar atributos de clase cuando una propiedad deberia tener el mismo valor inicial para todas las instancias de una clase, y atributos de instancia para propiedades que deben ser especificadas antes de que una instancia sea creada.

Ahora que tenemos una clase `Perro`, vamos a crear algunos perros.

2 Instanciando un Objeto

[24]:
```
class Gato:
    pass
```

[25]:
```
# instanciemos un objeto Gato
Gato()  # el dato de salida indica que ahora tenemos un objeto Gato ubicado en␣
↪la direccion de memoria 0x...
```

[25]:
```
<__main__.Gato at 0x7fc604359640>
```

[26]:
```
# instanciemos otro objeto Gato
Gato()  # observemos que la direccion de la ubicacion en memoria es distinta 0x.
↪..
```

[26]:
```
<__main__.Gato at 0x7fc6043599d0>
```

[27]:
```
# las direcciones de ubicacion en memoria son distintas porque son dos␣
↪instancias distintas
a = Gato()
b = Gato()
a == b  # no son los mismos objetos en memoria (la direccion no es la misma)
```

[27]:
```
False
```

[28]:
```
type(a)  # es un objeto Gato
```

[28]:
```
__main__.Gato
```

[29]:
```
type(a) == type(b)   # si es el mismo tipo de objeto Gato
```

[29]:
```
True
```

2.1 Atributos de Clase y de Instancia

[2]:
```
class Perro:
    especie = 'Canino'

    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad
```

[3]:
```
# el metodo __init__ especifica la necesidad de 3 argumentos / parametros pero␣
↪solo ingresamos 2
rocky = Perro('Rocky', 7)
lucky = Perro('Lucky', 3)
```

[4]:
```
# solo 2 argumentos porque cuando instanciamos el objeto, Python le pase el␣
↪objeto a self para representarlo
rocky.nombre
```

[4]:
```
'Rocky'
```

[39]:
```
rocky.edad
```

[39]:
```
7
```

[40]:
```
lucky.nombre
```

[40]:
```
'Lucky'
```

[41]:
```
lucky.edad
```

[41]:
```
3
```

[42]:
```
rocky.especie
```

[42]:
```
'Canino'
```

[43]:
```
lucky.especie
```

[43]:
```
'Canino'
```

[44]:
```
lucky.especie == rocky.especie
```

[44]:
```
True
```

[46]:
```
lucky.especie = 'Felino'
lucky.especie
```

[46]:
```
'Felino'
```

[47]:
```
rocky.especie
```

[47]:
```
'Canino'
```

[49]:
```
rocky.edad = 10
rocky.edad
```

[49]:
```
10
```

2.2 Metodos de instancias

Metodos de instancia son funciones definidas dentro de una clase. Por tanto, unicamente existen dentro del contexto del objeto y no pueden ser utilizados sin referenciar el objeto. Al igual que `__init__()` el primer argumento de un metodo de instancia siempre es `self`.

[5]:
```
class Perro:
    especie = 'Canino'

    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad

    def descripcion(self):  # describe la instancia del objeto
        return f'{self.nombre} tiene {self.edad} años'  # con informacion util

    def habla(self, sonido):
```

```
        return f'{self.nombre} dice {sonido}'
```

[6]:
```
rocky = Perro('Rocky', 4)
```

[7]:
```
rocky.descripcion()
```

[7]:
```
'Rocky tiene 4 años'
```

[8]:
```
rocky.habla('Woof woof')
```

[8]:
```
'Rocky dice Woof woof'
```

El metodo `descripcion` nos comparte informacion util de la instancia del objeto Perro, e.g. nombre y edad. Sin embargo, Python tiene un metodo especial que uno puede definir para que cuando uno imprima el objeto, el mismo muestre informacion util al usuario.

[59]:
```
# cuando llamamos print(rocky)
print(rocky)
```

```
<__main__.Perro object at 0x7fc6042e4730>
```

[62]:
```
class Perro:
    especie = 'Canino'

    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad

    def habla(self, sonido):
        return f'{self.nombre} dice {sonido}'

    def __str__(self):
        return f'{self.nombre} tiene {self.edad} años'
```

[63]:
```
rocky = Perro('Rocky', 4)
print(rocky)
```

```
Rocky tiene 4 años
```

2.3 Ejercicios

1. Modifique la clase `Perro` para incluid una tercer instancia llamada `color` que guarda el color del perro como string. Guarde la nueva clase en un script y pruebe el codigo al agregar lo siguiente abajo del mismo:

```
mika = Perro('Mika', 5, 'chocolate')
print(f'El color de {mika.nombre} es {mika.color}')
```

Deberia imprimir: `El color de Mika es chocolate`

2. Escriba una clase `Carro` con dos atributos de instancia: color y kilometraje. Luego instancie 2 objetos `Carro`, asi:

- Uno es azul, con 20,000 kilometros
- El otro es rojo, con 30,000 kilometros

Deberia imprimir algo asi:

```
El carro azul tiene 20,000 kilometros
El carro azul tiene 30,000 kilometros
```

3. Modifique la clase `Carro` con un metodo de instancia llamado `.conduce()` que toma un numero como argumento y le agrega ese numero al atributo `.kilometraje`. Pruebe que su solucion funcione instanciando un carro con 0 kilometros, luego llame `.conduce(100)`, y luego imprima el atributo `.kilometraje` para revisar que es de `100`.

3 Hereda de otras clases

La herencia es un proceso donde una clase obtiene atributos y metodos de otra. Esto nos permite crear una clase general primero y luego crear clases más especializadas que re-utilizan el código de la clase general. Por tanto, creamos una clase general, y luego una subclase.

Las subclases pueden anular y extender los atributos y metodos de la clase general. Es decir, las subclases heredan todos los atributos y metodos pero tambien pueden especificar diferentes atributos y metodos que son exclusivos de la subclase o inclusive re-definir metodos de la clase general.

El concepto de la herencia de objetos se puede pensar como herencia genetica, aunque la analogia no es perfecta.

Por ejemplo, se puede heredar el color de cabello de una madre, y este es un atributo con el cual nacemos, sin embargo, podemos cambiarnos el color de nuestro cabella a rojo si asi lo deseamos. Esta ultima accion anula el atributo que heredamos.

Tambien se hereda, en cierto sentido, el idioma de los padres. Si los padres hablan español, entonces los hijos tambien hablarán español. No obstante, uno puede aprender un idioma nuevo, como el inglés. En este sentido, uno extiendo los atributos heredados, toda vez que hemos agregado un atributo nuevo.

3.1 La clase `object`

El tipo de dato fundamental / primordial es el `object`. Este es el objeto base de todas las clases. Cuando definimos una nueva clase, Python implicitamente utiliza `object` como la clase base.

Las proximas dos clases son equivalentes.

[67]:
```python
# una clase hereda explicitamente de "object"
class Persona(object):
    pass
```

[69]:
```python
# una clase hereda implicitamente de "object"
class Persona:
```

```
    pass
```

3.2 Ejemplo: Parque de Perros

Pretendamos que estamos en un parque de perros. Hay muchos perros de diferentes razas en el parque, todos comportandose como perros. Supongamos que tenemos que modelar un parque de parre con clases. La clase `Perro` puede distinguir perros por `nombre` y `edad` pero no por raza.

[70]:
```
class Perro:
    species = 'Canino'

    def __init__(self, nombre, edad, raza):
        self.nombre = nombre
        self.edad = edad
        self.raza = raza  # podemos agregar un atributo raza para hacer la␣
↪distincion
```

[72]:
```
rocky = Perro('Rocky', 5, 'Doberman')
lucky = Perro('Lucky', 8, 'Beagle')
jack = Perro('Jack', 2, 'Labrador')
```

3.3 Clases Generales vs Subclases

Vamos a crear una clase para cada una de las razas mencionadas anteriormente: Doberman, Beagle, y Labrador.

[76]:
```
# esta es la clase general que utilizaremos
class Perro:
    species = 'Canino'

    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad

    def ladra(self, sonido):
        return f'{self.nombre} dice {sonido}'

    def __str__(self):
        return f'{self.nombre} tiene {self.edad} años de edad'
```

[77]:
```
# subclase 1
class Doberman(Perro):
    pass

# subclase 2
class Beagle(Perro):
    pass
```

```
# subclase 3
class Labrador(Perro):
    pass
```

[78]:
```
rocky = Doberman('Rocky', 5)
lucky = Beagle('Lucky', 8)
jack = Labrador('Jack', 2)
```

[79]:
```
# todas las subclases heredan los atributos y metodos de la clase general
rocky.species
```

[79]: 'Canino'

[80]:
```
lucky.nombre
```

[80]: 'Lucky'

[81]:
```
jack.ladra('Woof')
```

[81]: 'Jack dice Woof'

[84]:
```
# podemos determinar si la clase es una instancia de Perro con isinstance()
isinstance(rocky, Perro)
```

[84]: True

[83]:
```
# pero un Doberman no es una instancia de un Beagle
isinstance(rocky, Beagle)
```

[83]: False

3.4 Extendiendo la Funcionalidad de la Clase General

Hasta el momento tenemos cuatro clases: la clase general `Perro` y tres (3) subclases: `Doberman`, `Beagle` y `Labrador`. Toda las subclases heredan cada atributo y metodo de la clase general, incluyendo el metodo `ladra()`.

Ya que distintas razas de perro tienen un sonido particular cuando ladran, queremos proporcionar un valor particular para el sonido de cada raza.

[87]:
```
class Doberman(Perro):
    def ladra(self, sonido='Arf'):
        return f'{self.nombre} dice {sonido}'
```

[88]:
```
# el metodo ladra() ahora esta definido en la subclase Doberman con un sonido␣
↪particular
rocky = Doberman('Rocky', 5)
```

```
rocky.ladra()
```

[88]: 'Rocky dice Arf'

3.5 Ejercicios

1. Crea una subclase `GoldenRetriever` y agrega un metodo `ladra()` que utiliza un sonido particular. Puede utilizar la clase general compartida anteriormente.
2. Crea una clase `Rectangulo` que debe ser instanciada con 2 atributos: `largo` y `ancho`. Agrega un metodo llamado `area()` que retorna al area (`largo * ancho`) del rectangulo. Luego escribe una clase `Cuadrado` que hereda de `Rectangulo` y que se instancia con un solo atributo llamado `largo_de_lado`. Prueba la clase `Cuadrado` instanciandola con un `largo_de_lado` de `4`. Si ejecutamos el metodo `area()` debe retornar `16`.

4 Reto: Modela una Granja

Los requerimentos estan abiertos a interpretacion: 1. Debemos tener por lo menos cuatro (4) clases: la clase general `Animal` y por lo menos tres (3) subclases que heredan de la clase general. 2. Cada clase debe tener unos cuantos atributos y por lo menos un (1) metodo que modela algun comportamiento especifico del animal o todos los animales, e.g. caminar, correr, comer, dormir, entre otros. 3. Asegurese que los datos de salida detallan la informacion y comportamiento correctamente.

5 Resumen

Hemos aprendido un poco sobre la programacion orientada a objetos, lo cual es un paradigma de la programacion que no es exclusivo a Python.

Vimos como definir una clase, que es como un plano / prototipo de un objeto, y como instanciar un objeto de una clase. Tambien aprendimos sobre atributos, que son propiedades de un objeto, y metodos, que son comportamientos y acciones de un objeto.

Finalmente, aprendimos como funciona la herencia al crear subclases de clases generales, y como verificar si un objeto hereda de una clase utilizando `isinstance()`.