Objects in python

Carlos Tavares

November 20, 2023

Objects in python i

Python is a multi-paradigm language: imperative, structured and, to some extent, functional

It also allows object-oriented programming which makes it also an object-oriented language.

Objects in python ii

What is an object?

To define an object one has to define a **class**: a collection of elements that shares similar properties. For instance, the class of all persons, the class of all cars, the class of all houses, the class of all triangles

An object is an instance of a class. For instance, Ferrari FXX is an instance of a car, an isosceles triangle is an instance of a triangle.

Objects in python iii

In python as well as in most programming languages, an object is something that allows to aggregate data and behaviour.

A sort of tuple (data) with functions associated.

There are plenty of examples in python: **strings**, or **files**.

Whenever we have to do **entity.function ()**, entity is an object.

Classes and objects i

How to define a class in python:

class class_name:
(...)

After the class is defined one can instantiate objects:

object = Class ()

Classes and objects ii

Example:

The **empty** class:

```
class empty:
    pass

obj = empty()
```

Classes and objects iii

Another example: a tuple equivalent

```
class tuple_immitation:
    name = ""
    age = 0

obj = tuple_immitation ()
```

We can see the values in obj

```
» obj.name
""
» obj.age
0
```

Classes and objects iv

We can also **modify** the vaues in obj

» obj.name = "tarzan"
» obj.age = "150"

We can also instantiate other objects of the same class:

» obj2 = tuple_immitation ()
» obj2.name = "Mogli"
» obj2.age = 13

Adding behaviour to objects: constructor i

Objects consist of data and behaviour

We can add methods to objects. A particularly important method is the constructor method:

class **some_class**:

```
    def __init__(self, arguments):
        (...)
```

The init method is invoked when the object is instantiated. All methods that belong to a class must receive self as argument.

Adding behaviour to objects: constructor ii

Example: tuple with constructor

```
class tuple_with_constructor:
    def __init (self, name = "", age = 0):
        self.name = name
        self.age = age
```

This creates an object with the names and age given as arguments to the constructor. This constructor accepts no arguments as both arguments have default values.

Adding behaviour to objects: other methods i

Still about the addition of methods (we can add as much methods as we like)

class **some_class**:

```
    def __init__(self, arguments):
        (...)

    def another_method (self, arguments):
        (...)
```

The init method is invoked when the object is instantiated. All methods that belong to a class must receive self as argument.

Adding behaviour to objects: other methods ii

Example: tuple with constructor and method

```
class tuple_that_does_something
    def __init (self, name = "", age = 0):
        self.name = name
        self.age = age

    def set_to_tarzan (self):
        self.name = "Tarzan"
        self.age = 120
```

Inheritance i

Functions already allow code reusability...

... so why objects?

Object allow the reusability of a whole body of data and behaviour through inheritance, which offers a structural advantage.

The main mechanism of reusability is inhertiance: it is possible to derive classes from other classes.

Inheritance ii

Example: let's say we have a class Person

```
Class Person:
    def __init__(self):
        self.dead = False
        self.energy = 0
        self.rested = 0

    def eat (self):
        self.energy = 100

    def sleep (self):
        self.rested = 100
```

Inheritance iii

```
def live (self):
    while not (self.dead):
        self.eat ()
        self.sleep ()

def die (self):
    self.dead = True
```

Inheritance iv

```
# We could create a subclass child from this
class Child (Person):

    def __init__ (self):
        super().__init__ ()
        self.happy = True

    def play ():
        self.happy = True

    def live ():
        while !dead:
            play ()
            eat ()
```

Inheritance v

sleep ()

Class child inherits all methods and properties of class person. It is also possible to replace methods of the parent class (see the live method) and invoking methods of the parent class via super.method, see the constructor method.

Python admits imperative, structured, functional (higher-order functions) and object-oriented programming.

Questions?