Python Cheat Sheet: Object Orientation Terms

"*A puzzle a day to learn, code, and play*" → Visit finxter.com

	Description	Example
Class	A blueprint to create objects. It defines the data (attributes) and functionality (methods) of the objects. You can access both attributes and methods via the dot notation.	
Object (=instance)	A piece of encapsulated data with functionality in your Python program that is built according to a class definition. Often, an object corresponds to a thing in the real world. An example is the object "Obama" that is created according to the class definition "Person". An object consists of an arbitrary number of attributes and methods, encapsulated within a single unit.	
Instantiation	The process of creating an object of a class. This is done with the constructor method __init__(self, ...).	
Method	A subset of the overall functionality of an object. The method is defined similarly to a function (using the keyword "def") in the class definition. An object can have an arbitrary number of methods.	
Self	The first argument when defining any method is always the `self` argument. This argument specifies the instance on which you call the method. `self` gives the Python interpreter the information about the concrete instance. To *define* a method, you use `self` to modify the instance attributes. But to *call* an instance method, you do not need to specify `self`.	
Encapsulation	Binding together data and functionality that manipulates the data.	
Attribute	A variable defined for a class (class attribute) or for an object (instance attribute). You use attributes to package data into enclosed units (class or instance).	
Class attribute	(=class variable, static variable, static attribute) A variable that is created statically in the class definition and that is shared by all class objects.	
Instance attribute (=instance variable)	A variable that holds data that belongs only to a single instance. Other instances do not share this variable (in contrast to class attributes). In most cases, you create an instance attribute x in the constructor when creating the instance itself using the self keywords (e.g. self.x = <val>).	
Dynamic attribute	An instance attribute that is defined dynamically during the execution of the program and that is not defined within any method. For example, you can simply add a new attribute `neew` to any object o by calling `o.neew = <val>`.	
Method overloading	You may want to define a method in a way so that there are multiple options to call it. For example for class X, you define a method f(...) that can be called in three ways: f(a), f(a,b), or f(a,b,c). To this end, you can define the method with default parameters (e.g. f(a, b=None, c=None).	
Inheritance	Class A can inherit certain characteristics (like attributes or methods) from class B. For example, the class "Dog" may inherit the attribute "number_of_legs" from the class "Animal". In this case, you would define the inherited class "Dog" as follows: "class Dog(Animal): ..."	

```
class Dog:

    # class attribute
    is_hairy = True

    # constructor
    def __init__(self, name):
        # instance attribute
        self.name = name

    # method
    def bark(self):
        print("Wuff")


bello = Dog("bello")
paris = Dog("paris")

print(bello.name)
"bello"

print(paris.name)
"paris"


class Cat:

    # method overloading
    def miau(self, times=1):
        print("miau " * times)

fifi = Cat()

fifi.miau()
"miau "

fifi.miau(5)
"miau miau miau miau miau "

# Dynamic attribute
fifi.likes = "mice"
print(fifi.likes)
"mice"

# Inheritance
class Persian_Cat(Cat):
    classification = "Persian"

mimi = Persian_Cat()
print(mimi.miau(3))
"miau miau miau "


print(mimi.classification)
```

finxter