# Objects and Classes

Vivek Singh

Information Systems Decision Sciences (ISDS)
MUMA College of Business
University of South Florida
Tampa, Florida

2018

# What are Objects?

- Everything in Python is an object. That includes numbers, strings, sequences, functions, modules and packages.
- An object contains both data and the methods that define the behaviour it could be part of, in the program.
- For example, an integer object contains numerical data and could become part of arithmetic operations such as addition,exponentiation etc.
- Objects are instances of larger, concrete templates called as Classes.

# Classes

- Classes are used to define data types and behaviours in Object-oriented programming paradigm.
- Once a class is defined, instances of the classes could be spawned easily whenever required.
- A simple class definition looks like this:

```
class Student():
        def __init__(self, name):
                self.name = name

student = Student('Rick')
student.name
'Rick'
```

# Explanation

- The code has different components to it. The keyword 'class' is used to create a new class definition.
- The class has a name and in this case, it is "Student". We are going to create a custom Student data type.
- No arguments are passed to it.
- The __init__ is the initializer function in Python. Note that it is written with two trailing underscores and two leading underscores.
- The first argument passed to the init function is always a reference that points to the current object instance. In Python, we use "self" to represent this reference.
- *Self* is not a keyword as such. But it is good practice to use *Self* for this purpose.
- The instance has one property called the name and it is initialized as seen in the code: self.name = name

# Explanation

Here is what the following code does:
student = Student('Rick')

- Looks up the definition of the Student class in the current namespace.
- Instantiates/Creates a new object in the memory.
- Calls the init method (which is the initializer), passing the newly created object reference as "self" along with the other argument 'Rick' as the name of the student.
- Assigns the value of the name to the newly created object.
- And it finally returns the new object.
- Assigns the object itself to the variable student. Now student is a reference that points to the Student object residing in the memory with the name "Rick".
- This new object can be used for any purpose in the program, as far as its data type and behavioral definition supports them.

# The Constructor and Initializer Function

- The init method is used to initialize the object we are creating.
- In this example, we are initializing the instance's *name* property with a value that is passed to the name argument(Rick).
- The self is simply a reference to that particular instance.
- Since, self already is a reference to the instance, it is understood that the instance has been already been created.
- This is done implicitly by the __**new**__ function whose implementation is hidden from us.
- The new function is the Constructor function in Python. It builds/instantiates(creates) the object instance in the memory.
- It can be overridden in special cases.

# Inheritance

Inheritance is the way to extend an existing Class's definition to add or change some behaviors of that class without disturbing the original definition. It is a very powerful feature and offers various advantages.

- When a class is inherited from, all the properties of it are inherited without the need to copy any code.
- If anything is to be changed, we simply override that particular behavior.
- The original class is commonly called as base, parent or superclass and the inheriting class is called a derived, child or subclass.

# Overriding a Method

A child class inherits everything from its parent class. But the methods of the Parent class can be overridden. All the methods including the init method can be overriden.

```
class Faculty():
        def __init__(self, name):
                self.name = name
class Doctor(Faculty):
        def __init__(self, name):
                self.name = 'Doctor ' + name
class Professor(Faculty):
        def __init__(self, name):
                self.name = 'Professor ' + name

Felicia = Doctor('Felicia')
Warren = Professor('Warren')
```

# Adding a Method

A child class could also add a method that is not there as part of the Parent's class definition.

```
class Faculty():
    def __init__(self, name):
        self.name = name
class Doctor(Faculty):
    def __init__(self, name):
        self.name = 'Doctor ' + name
    def teach(self):
        print(self.name,'teaches Graduates.')
class Professor(Faculty):
    def __init__(self, name):
        self.name = 'Professor ' + name
    def teach(self):
        print(self.name,'teaches Undergrads.')
```

# Using Super()

- Sometimes, the child might want to do things the exact same way the Parent does.
- In these cases, it is advisable that we directly invoke the property of choice of the Parent using super().
- The super() holds the definition of the Parent class.

```
class Faculty():
    def __init__(self, name):
        self.name = name

class Professor(Faculty):
    def __init__(self, name, email):
        super().__init__(name)
        self.email = email
```

- The advantage of this is that whenever the Parent class definition changes, it is automatically reflected in the child as well.

# Types of Methods in Classes

- There are three types of methods in classes.
  - Class methods,
  - Instance methods and
  - Static methods.
- Some attributes and methods are part of the class itself and some are only part of the object instances of the class.
- The way to differentiate between them is by using the "self" argument. When we encounter a self argument in the methods within the class, we can conclude that they are instance methods.
- The first argument to an instance method is always self, and whenever these instance methods are called (with the objects using dot notation method of calling), the object itself is passed to the method.
- Instance method only affect the object instance calling them. Whereas, a class method affects the entire class. Any change made to the class affects all the objects.

# Types of Methods in Classes

- A @classmethod decorator indicates a class method.Here, the first parameter to the method is the class itself.
- Like how "self" is used for object instances, we use "cls" as a reference to the class object.

```
class Car():
        count = 0
        def __init__(self):
                Car.count += 1
        @classmethod
        def car_counter(cls):
                print("Car has ",cls.count,
                " cars")
```

- Instead of cls.count, we could very well use "Car.count" as well.
- The car_counter() method can be accessed by each object of the class as well.

# Types of Methods in Classes - Static Methods

- A third type of method is the static method. It neither affects the class nor the objects.
- Its defined with a @staticmethod decorator with no initial self or cls parameter.

```
class Car():
        @staticmethod
        def promo():
                print('All these cars are for
                sale!!')
Car.promo()
All these cars are for sale!!
```

- Notice that we did not have to create objects of the class to access this method.
- Static methods do not have access to instance attributes or class attributes.

# Summary

- Object oriented programming gives the programmer a great deal of advantage in creating custom data types and to contain logic and data together.
- We learned how to create classes and object instances. We learned what constructors and initialzers are.
- We learned how to define methods that need to work on the data type inside the class definition.
- We learned method overriding and class inheritance, and
- We learned the different types of methods.