9. Classes
==========

Creating the Dog Class
Each instance created from the Dog class will store a name and an age, and we'll give each dog the ability to sit() and roll_over():

```
class Dog():
    """A simple attempt to model a dog."""

    def __init__(self, name, age):
      """Initialize name and age attributes."""
       self.name = name
       self.age = age

    def sit(self):
      """Simulate a dog sitting in response to a command."""
      print(self.name.title() + " is now sitting.")
    def roll_over(self):
      """Simulate rolling over in response to a command."""
      print(self.name.title() + " rolled over!"
```

The __init__() method at w is a special method
Python runs automatically whenever we create a new instance based on the Dog class.

Creating Classes in Python 2.7
When you create a class in Python 2.7, you need to make one minor change. You include the term object in parentheses when you create a class:

```
class ClassName(object):
   --snip--
```

Making an Instance from a Class

```
my_dog = Dog('willie', 6)
print("My dog's name is " + my_dog.name.title() + ".")
print("My dog is " + str(my_dog.age) + " years old.")

class Car():
   """A simple attempt to represent a car."""
    def __init__(self, make, model, year):
      """Initialize attributes to describe a car."""
      self.make = make
      self.model = model
      self.year = year

   def get_descriptive_name(self):
      """Return a neatly formatted descriptive name."""
      long_name = str(self.year) + ' ' + self.make + ' ' + self.model
```

```
        return long_name.title()

my_new_car = Car('audi', 'a4', 2016)
print(my_new_car.get_descriptive_name())
```

Inheritance

You don’t always have to start from scratch when writing a class. If the class you’re writing is a specialized version of another class you wrote, you can use inheritance. When one class inherits from another, it automatically takes on all the attributes and methods of the first class.

```
 class ElectricCar(Car):
   """Represent aspects of a car, specific to electric vehicles."""

    def __init__(self, make, model, year):
      """Initialize attributes of the parent class."""
      super().__init__(make, model, year)

my_tesla = ElectricCar('tesla', 'model s', 2016)
print(my_tesla.get_descriptive_name())

class ElectricCar(Car):
   def __init__(self, make, model, year):
      super(ElectricCar, self).__init__(make, model, year)
```

Python Object & Class

Python OOP

Introduction to OOPs in Python

Python is a multi-paradigm programming language. Meaning, it supports different programming approach.

One of the popular approach to solve a programming problem is by creating objects. This is known as Object-Oriented Programming (OOP).

An object has two characteristics:

- attributes
- behavior

Let's take an example:

Parrot is an object,

- name, age, color are attributes
- singing, dancing are behavior

Inheritance
A process of using details from a new class without modifying existing class.
Encapsulation Hiding the private details of a class from other objects.
Polymorphism A concept of using common operation in different ways for different data input.

Class

A class is a blueprint for the object.

We can think of class as an sketch of a parrot with labels. It contains all the details about the name, colors, size etc. Based on these descriptions, we can study about the parrot. Here, parrot is an object.

The example for class of parrot can be :

```
class Parrot:
    pass
```

Here, we use class keyword to define an empty class Parrot. From class,
we construct instances. An instance is a specific object created from a particular class.
Object

An object (instance) is an instantiation of a class. When class is defined,
only the description for the object is defined. Therefore, no memory or storage is allocated.

The example for object of parrot class can be:

```
obj = Parrot()
```

Here, obj is object of class Parrot.

```
class Parrot:

    # class attribute
    species = "bird"

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

# instantiate the Parrot class
blu = Parrot("Blu", 10)
woo = Parrot("Woo", 15)

# access the class attributes
print("Blu is a {}".format(blu.__class__.species))
print("Woo is also a {}".format(woo.__class__.species))
```

```
# access the instance attributes
print("{} is {} years old".format( blu.name, blu.age))
print("{} is {} years old".format( woo.name, woo.age))

class Parrot:

        # instance attributes
        def __init__(self, name, age):
                self.name = name
                self.age = age

        # instance method
        def sing(self, song):
                return "{} sings {}".format(self.name, song)

        def dance(self):
                return "{} is now dancing".format(self.name)

# instantiate the object
blu = Parrot("Blu", 10)

# call our instance methods
print(blu.sing("'Happy'"))
print(blu.dance())
```

Python Inheritance

Inheritance

Inheritance is a way of creating new class for using details of existing class without modifying it. The newly formed class is a derived class (or child class). Similarly, the existing class is a base class (or parent class).

```
# parent class
class Bird:

        def __init__(self):
                print("Bird is ready")

        def whoisThis(self):
                print("Bird")

        def swim(self):
                print("Swim faster")

# child class
class Penguin(Bird):
```

```
        def __init__(self):
                # call super() function
                super().__init__()
                print("Penguin is ready")

        def whoisThis(self):
                print("Penguin")

        def run(self):
                print("Run faster")

peggy = Penguin()
peggy.whoisThis()
peggy.swim()
peggy.run()
```

Encapsulation

Using OOP in Python, we can restrict access to methods and variables.
This prevent data from direct modification which is called encapsulation.
In Python, we denote private attribute using underscore as prefix
i.e single " _ " or double " __".

```
class Computer:

        def __init__(self):
                self.__maxprice = 900

        def sell(self):
                print("Selling Price: {}".format(self.__maxprice))

        def setMaxPrice(self, price):
                self.__maxprice = price

c = Computer()
c.sell()

# change the price
c.__maxprice = 1000
c.sell()

# using setter function
c.setMaxPrice(1000)
c.sell()
```

Polymorphism

Polymorphism is an ability (in OOP) to use common interface for multiple form (data types).
Suppose, we need to color a shape, there are multiple shape option
(rectangle, square, circle). However we could use same method to color any shape.
This concept is called Polymorphism.

```
class Parrot:

	def fly(self):
		print("Parrot can fly")

	def swim(self):
		print("Parrot can't swim")

class Penguin:

	def fly(self):
		print("Penguin can't fly")

	def swim(self):
		print("Penguin can swim")

# common interface
def flying_test(bird):
	bird.fly()

#instantiate objects
blu = Parrot()
peggy = Penguin()

# passing the object
flying_test(blu)
flying_test(peggy)

class MyClass:
	"This is my second class"
	a = 10
	def func(self):
		print('Hello')

# Output: 10
print(MyClass.a)

# Output: <function MyClass.func at 0x0000000003079BF8>
print(MyClass.func)

# Output: 'This is my second class'
print(MyClass.__doc__)
```

What is Inheritance?

Inheritance is a powerful feature in object oriented programming.

It refers to defining a new class with little or no modification to an existing class. The new class is called derived (or child) class and the one from which it inherits is called the base (or parent) class.

Python Inheritance Syntax

```
class BaseClass:
  Body of base class
class DerivedClass(BaseClass):
  Body of derived class
```

Multiple Inheritance in Python

Like C++, a class can be derived from more than one base classes in Python. This is called multiple inheritance.

In multiple inheritance, the features of all the base classes are inherited into the derived class. The syntax for multiple inheritance is similar to single inheritance.

Example

```
class Base1:
        pass

class Base2:
        pass

class MultiDerived(Base1, Base2):
        pass
```