Python Programming Language

Prepared by: Mohamed Ayman

Algorithm Engineer at Valeo | Machine Learning Researcher

spring 2019

sw.eng.MohamedAyman@gmail.com

facebook.com/cs.MohamedAyman

linkedin.com/in/cs-MohamedAyman

codeforces.com/profile/Mohamed_Ayman

Lecture 1

Object Oriented Overview

Course Roadmap

Lecture Agenda

We will discuss in this lecture the following topics

1- Introduction to OOP
2- Classes & Objects
3- Attributes
4- Methods
5- Constructors
6- Destructors

Let's
START UP

Lecture Agenda

Introduction to OOP

- Object-oriented Programming, or OOP for short, is a programming paradigm which provides a means of structuring programs so that properties and behaviors are bundled into individual objects.
- For instance, an object could represent a person with a name property, age, address, etc., with behaviors like walking, talking, breathing, and running. Or an email with properties like recipient list, subject, body, etc., and behaviors like adding attachments and sending.
- Put another way, object-oriented programming is an approach for modeling concrete, real-world things like cars as well as relations between things like companies and employees, students and teachers, etc. OOP models real-world entities as software objects, which have some data associated with them and can perform certain functions.
- Another common programming paradigm is procedural programming which structures a program like a recipe in that it provides a set of steps, in the form of functions and code blocks, which flow sequentially in order to complete a task.

Introduction to OOP

- Python is an object oriented programming language. Unlike procedure oriented programming, where the main emphasis is on functions, object oriented programming stress on objects.
- Object is simply a collection of data (variables) and methods (functions) that act on those data. And, class is a blueprint for the object.
- We can think of class as a sketch (prototype) of a house. It contains all the details about the floors, doors, windows etc. Based on these descriptions we build the house. House is the object.
- As, many houses can be made from a description, we can create many objects from a class. An object is also called an instance of a class and the process of creating this object is called instantiation.
- 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)

Introduction to OOP

- Classes: Just like every other Object Oriented Programming language Python supports classes. Let's look at some points on Python classes. Classes are created by keyword class. Attributes are the variables that belong to class. Attributes are always public and can be accessed using dot (.) operator.

- Object: A unique instance of a data structure that is defined by its class. An object comprises both data members (class variables and instance variables) and methods.

- Instance: An individual object of a certain class. An object obj that belongs to a class Circle, for example, is an instance of the class Circle.

- Methods: Method is a bunch of code that is intended to perform a particular task in your Python's code. Function that belongs to a class is called an Method. All methods require 'self' parameter. If you have coded in other OOP language you can think of 'self' as the 'this' keyword which is used for the the current object. It unhides the current instance variable. 'self' mostly work like 'this'. 'def' keyword is used to create a new method.

Lecture Agenda

Classes

- The primitive data structures available in Python, like numbers, strings, and lists are designed to represent simple things like the cost of something, the name of a poem, and your favorite colors, respectively, but what if you wanted to represent something much more complicated?
- For example, let's say you wanted to track a number of different animals. If you used a list, the first element could be the animal's name while the second element could represent its age.
- How would you know which element is supposed to be which? What if you had 100 different animals? Are you certain each animal has both a name and an age, and so forth? What if you wanted to add other properties to these animals? This lacks organization, and it's the exact need for classes.
- Classes are used to create new user-defined data structures that contain arbitrary information about something. In the case of an animal, we could create an Animal() class to track properties about the Animal like the name and age. It's important to note that a class just provides structure - it's a blueprint for how something should be defined, but it doesn't actually provide any real content itself. it may help to think of a class as an idea for how something should be defined.

Classes

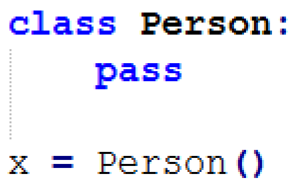
- A class is a blueprint for the object. We can think of class as an sketch of a person with labels. It contains all the details about the name, colors, size etc. Based on these descriptions, we can study about the person. Here, person is an object.

```python
class Person:
    pass
```

- Here, we use class keyword to define an empty class Person. From class, we construct instances. An instance is a specific object created from a particular class.
- Like function definitions begin with the keyword def, in Python, we define a class using the keyword class.
- As soon as we define a class, a new class object is created with the same name. This class object allows us to access the different attributes as well as to instantiate new objects of that class.

Objects

- 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 person class can be:



```
class Person:
    pass

x = Person()
```

- Here, x is object of class Person. It can also be used to create new object instances (instantiation) of that class. The procedure to create an object is similar to a function call.
- This will create a new instance object named x. We can access attributes of objects using the object name prefix.

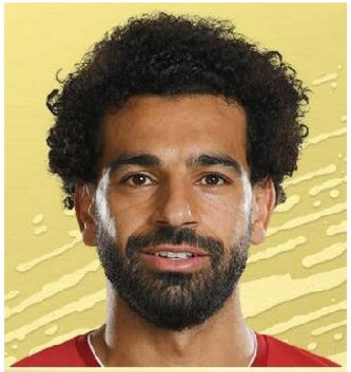
Classes & Objects

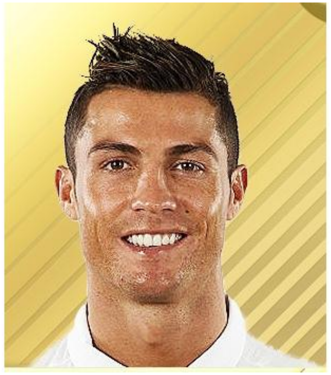
Class: Player

Object: Leo Messi

Object: Mo Salah

Object: C. Ronaldo

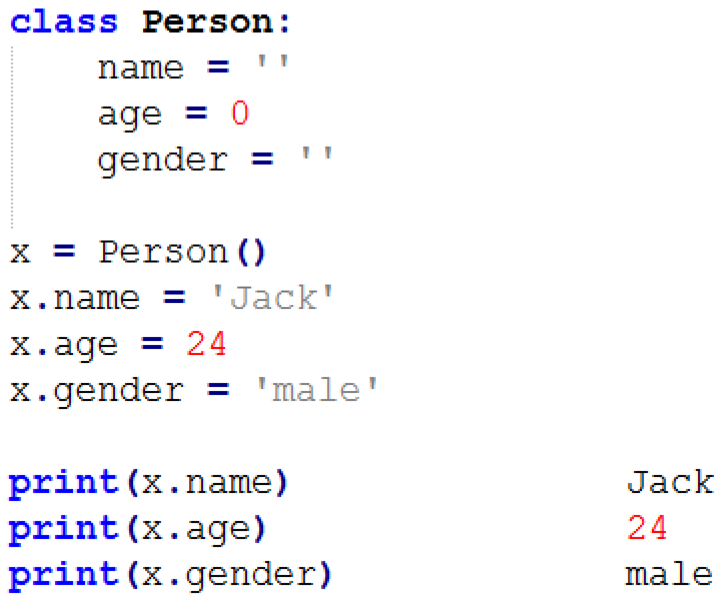




Lecture Agenda

Attributes

- All classes create objects, and all objects contain characteristics called attributes (referred to as properties in the opening paragraph).
- Suppose we have details of parrot. Now, we are going to show how to build the class and objects of parrot.
- In this program, we create a class with name Person. Then, we define attributes. The attributes are a characteristic of an object. Then, we create instances of the Person class. Here, x is reference (value) to our new object.
- Then, we access the class attribute using class.attribute. Class attributes are same for all instances of a class. Similarly, we access the instance attributes using x.name and x.age. However, instance attributes are different for every instance of a class.



```
class Person:
    name = ''
    age = 0
    gender = ''

x = Person()
x.name = 'Jack'
x.age = 24
x.gender = 'male'

print(x.name)
print(x.age)
print(x.gender)
```

```
Jack
24
male
```

Attributes

- Attributes of Class Player:

- Name
- Date of Birth
- Nation
- Club
- League
- Position
- Height
- Weight
- Foot
- Attacking Work Rate
- Defensive Work Rate
- Weak Foot
- Skill Moves
- Pace
- Shooting
- Passing
- Dribbling
- Defending
- Physical
- Overall Rate

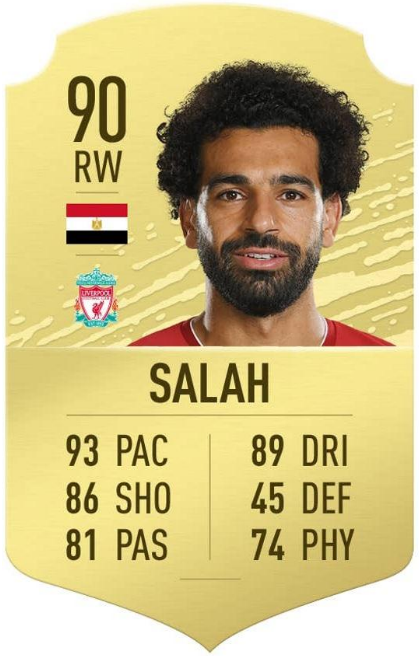
Class: Player

Attributes

- Attributes of Object Mo Salah:

- Name: Mohamed Salah
- Date of Birth: 15 Jun 1992
- Nation: Egypt
- Club: Liverpool
- League: England Premier League
- Position: RW
- Height: 175 cm
- Weight: 71 kg
- Foot: Left
- Attacking Work Rate: 5/5
- Defensive Work Rate: 4/5
- Weak Foot: 3/5
- Skill Moves: 5/5
- Pace: 93%
- Shooting: 86%
- Passing: 81%
- Dribbling: 89%
- Defending: 45%
- Physical: 74%
- Overall Rate: 90%

Object: Mo Salah



Attributes

- Attributes of Object Leo Messi:

- Name: Lionel Messi	- Weak Foot: 4/5
- Date of Birth: 24 Jun 1987	- Skill Moves: 4/5
- Nation: Argentina	- Pace: 87%
- Club: FC Barcelona	- Shooting: 92%
- League: Spain Primera Division	- Passing: 92%
- Position: RW	- Dribbling: 96%
- Height: 170 cm	- Defending: 39%
- Weight: 72 kg	- Physical: 66%
- Foot: Left	- Overall Rate: 94%
- Attacking Work Rate: 5/5	
- Defensive Work Rate: 3/5	

Object: Leo Messi

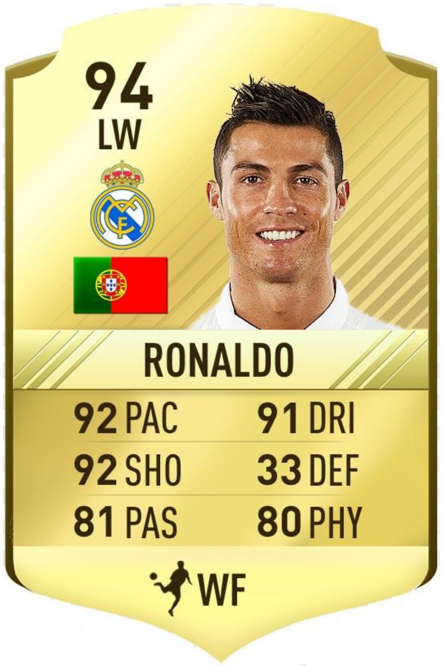
94
CF
MESSI
88 PAC
91 SHO
88 PAS
96 DRI
32 DEF
61 PHY

Attributes

- Attributes of Object C. Ronaldo:

- Name: Cristiano Ronaldo
- Date of Birth: 5 Feb 1985
- Nation: Portugal
- Club: Real Madrid CF
- League: Spain Primera Division
- Position: LW
- Height: 185 cm
- Weight: 84 kg
- Foot: Right
- Attacking Work Rate: 5/5
- Defensive Work Rate: 3/5
- Weak Foot: 4/5
- Skill Moves: 5/5
- Pace: 90%
- Shooting: 93%
- Passing: 82%
- Dribbling: 89%
- Defending: 35%
- Physical: 78%
- Overall Rate: 94%

Object: C. Ronaldo



Lecture Agenda

Methods

- Methods are functions defined inside the body of a class. They are used to define the behaviors of an object. In this program, we define two methods i.e walking() and running(). These are called instance method because they are called on an instance object i.e x.

- Attributes may be data or method. Method of an object are corresponding functions of that class. Any function object that is a class attribute defines a method for objects of that class. This means to say, since Class.func is a function object (attribute of class), x.func will be a method object.

```python
class Person:
    def walking(self):
        print('I am walking now')
    def running(self):
        print('I am running now')

x = Person()
x.walking()
x.running()
```

```
I am walking now
I am running now
```

Self Keyword

- Class methods must have an extra first parameter in method definition. We do not give a value for this parameter when we call the method, Python provides it. If we have a method which takes no arguments, then we still have to have one argument - the self.
- When we call a method of this object as myObject.methodName(arg1, arg2), this is automatically converted by Python into MyClass.method(myobject, arg1, arg2) - this is all the special self is about.
- self represents the instance of the class. By using the "self" keyword we can access the attributes and methods of the class in python. It binds the attributes with the given arguments.
- The reason you need to use self. is because Python does not use the @ syntax to refer to instance attributes. Python decided to do methods in a way that makes the instance to which the method belongs be passed automatically, but not received automatically: the first parameter of methods is the instance the method is called on

Methods

- Methods of Class Player:

- Running
- Walking
- Short Shooting
- Long Shooting
- Short Passing
- Long Passing
- Crossing
- Dribbling
- Jumping
- Finishing
- Slide Tackling
- Stand Tackling
- Heading

Class: Player

Methods

Tackling

Passing

Jumping

Long Shooting

Methods

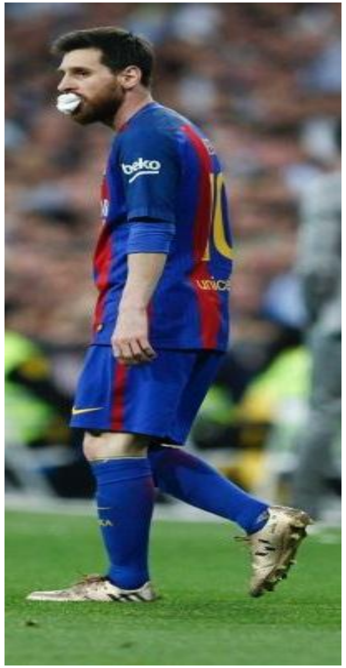
Finishing

Crossing

Dribbling

Short Shooting

Methods


Walking

Dribbling

Heading

Running

Lecture Agenda

Constructors

- Constructors are generally used for instantiating an object. The task of constructors is to initialize(assign values) to the data members of the class when an object of class is created. In Python the __init__() method is called the constructor and is always called when an object is created.
- Syntax of constructor declaration :

```
class className:
    def __init__(self):
        # body of the constructor
        pass
```

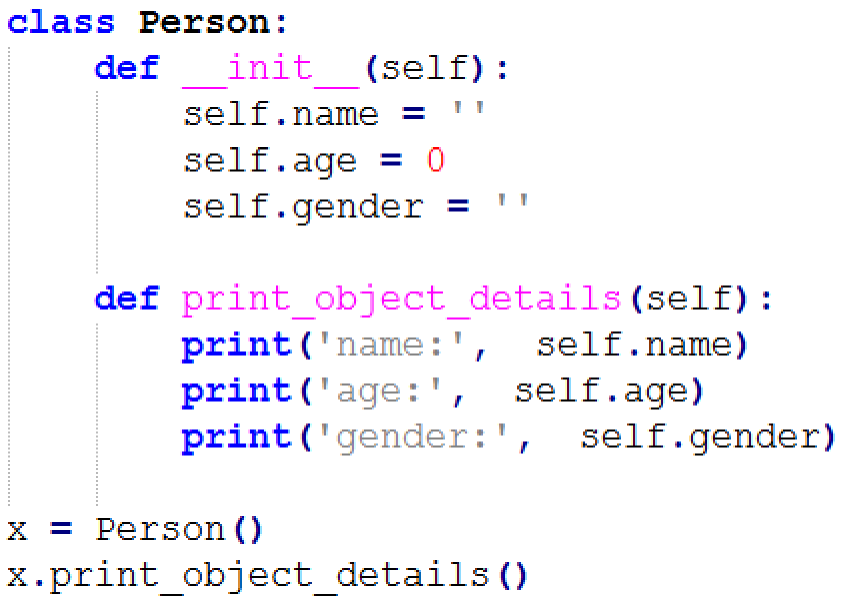
- Types of constructors :

1. default constructor :The default constructor is simple constructor which doesn't accept any arguments. It's definition has only one argument which is a reference to the instance being constructed.

2. parameterized constructor :constructor with parameters is known as parameterized constructor. The parameterized constructor take its first argument as a reference to the instance being constructed known as self and the rest of the arguments are provided by the programmer.

Constructors

- Class functions that begins with double underscore (__) are called special functions as they have special meaning. Of one particular interest is the __init__() function. This special function gets called whenever a new object of that class is instantiated.
- This type of function is also called constructors in Object Oriented Programming (OOP). We normally use it to initialize all the variables.
- In this example, we define a new class to represent complex numbers. It has two functions, __init__() to initialize the variables (defaults to zero) and print_object_details() to display the attributes of the object.



```
class Person:
    def __init__(self):
        self.name = ''
        self.age = 0
        self.gender = ''

    def print_object_details(self):
        print('name:',  self.name)
        print('age:',  self.age)
        print('gender:',  self.gender)

x = Person()
x.print_object_details()
```

Constructors

- The __init__() method is run as soon as an object of a class is instantiated. The method is useful to do any initialization you want to do with your object.
- In Python, instance variables are variables whose value is assigned inside a constructor or method with self. Class variables are variables whose value is assigned in class.

```
class Person:
    def __init__(self, name, age, gender):
        self.name = name
        self.age = age
        self.gender = gender

    def print_object_details(self):
        print('name:', self.name)
        print('age:', self.age)
        print('gender:', self.gender)

x = Person('Jack', 24, 'male')
x.print_object_details()
```

```
name: Jack
age: 24
gender: male
```

Lecture Agenda

Destructors

- Destructors are called when an object gets destroyed. In Python, destructors are not needed because Python has a garbage collector that handles memory management automatically.
- The __del__() method is a known as a destructor method in Python. It is called when all references to the object have been deleted i.e when an object is garbage collected.
- Syntax of destructor declaration :

```
class className:
    def __del__(self):
        # body of destructor
        pass
```

Destructors

- Here is the simple example of destructor. By using del keyword we deleted the all references of object 'obj', therefore destructor invoked automatically.

```
class Person:
    def __init__(self):
        print('Person created.')

    def __del__(self):
        print('Person deleted.')

x = Person()
del x
```

- The destructor was called after the program ended or when all the references to object are deleted i.e when the reference count becomes zero, not when object went out of scope.

Constructors & Destructors

The Iterations Tracing

1-
```
class Person:
    def __init__(self):
        print('Person created.')

    def __del__(self):
        print('Person deleted.')

x = Person()
del x
```

2-
```
class Person:
    def __init__(self):
        print('Person created.')

    def __del__(self):
        print('Person deleted.')

x = Person()
del x
```

3-
```
class Person:
    def __init__(self):
        print('Person created.')

    def __del__(self):
        print('Person deleted.')

x = Person()
del x
```

Constructors & Destructors

The Iterations Tracing

1-
```python
class Person:
    def __init__(self):
        print('Person created.')

    def __del__(self):
        print('Person deleted.')

x = Person()
del x
```

2-
```python
class Person:
    def __init__(self):
        print('Person created.')

    def __del__(self):
        print('Person deleted.')

x = Person()
del x
```

3-
```python
class Person:
    def __init__(self):
        print('Person created.')

    def __del__(self):
        print('Person deleted.')

x = Person()
del x
```

Garbage Collection

- Python's memory allocation and deallocation method is automatic. The user does not have to reallocate or deallocate memory similar to using dynamic memory allocation in some languages.
- Python uses two strategies for memory allocation: (Reference counting - Garbage collection)
- Python deletes unneeded objects (built-in types or class instances) automatically to free the memory space. The process by which Python periodically reclaims blocks of memory that no longer are in use is termed as Garbage Collection.
- Python's garbage collector runs during program execution and is triggered when an object's reference count reaches zero. An object's reference count changes as the number of aliases that point to it changes.
- An object's reference count increase when it is assigned a new name or placed in a container (list, tuple or dictionary). The object's reference count decrease when it is deleted with del, it's reference is reassigned, or its reference goes out of scope. When an object's reference count reaches zero

Garbage Collection

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def __del__(self):
        print('Person deleted.')

x = Person('Jack', 24)
y = Person('Jack', 24)
z = x

print('id(x):', id(x))
print('id(y):', id(y))
print('id(z):', id(z))

print('we will delete x')
del x
print('we will delete y')
del y
print('we will delete z')
del z
```

```
id(x): 139905073400720
id(y): 139905073400784
id(z): 139905073400720

we will delete x

we will delete y
Person deleted.
we will delete z
Person deleted.
```

Lecture Agenda

- ✓ Section 1: Introduction to OOP
- ✓ Section 2: Classes & Objects
- ✓ Section 3: Attributes
- ✓ Section 4: Methods
- ✓ Section 5: Constructors
- ✓ Section 6: Destructors

```
<!DOCTYPE html>
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8"
  />
  <meta name="viewport" content="width=device-width,
  initial-scale=1, maximum-scale=1">
   <title>Christopher Gower | Portfolio</title>
  <link rel="stylesheet" type="text/css" href="style.css" />
</head>
<body>
  <div id="wrapper">
   <div id="header">
 <ul id="nav">
  <li><a href="#">About Us</a></li>
  <li><a href="#">Our Products</a></li>
  <li><a href="#">FAQs</a></li>
  <li><a href="#">Contact</a></li>
  <li><a href="#">Login</a></li>
</ul>
    </div><!-- #header -->
     <div id="content">
  <div class="banner" style="background-image: url(images/headerimage1.jpg); background-size: 100%; background-size: cover;">
 <h1 class="name">Hello, my name is Chris!</h1>
<!-- this is the blue banner at top of site -->
  </div>
  <div class="container">
   <div class="row">
     <div class="col-4">
     <h2 class="work">Work</h2>
     <p>Aliquam accumsan purus vel elit ultricies elementum.
     Vestibulum mattis vehicula molestie. Nunc porttitor
     magna in ligula tincidunt, id aliquam lacus tempus.
     Donec nunc tortor, ultrices ac est vitae, ultrices
     tempor magna. Pellentesque ut rhoncus dui, sed lobortis
     diam. Ut ornare ac nulla nec cursus. Proin laoreet
     ornare orci quis tincidunt.</p>
   <!-- This content will take up 3/12 (or 1/4) of the
   container -->
    </div>
  <div class="col-4">
  <h2 class="skills">Skills</h2>
  <img class="skillsimage" src="images/skillsimage.jpg" alt="
  skills">
  </div>
```

```
/**
 * CSSReset.com - How To Keep Footer At Bottom of Page with CSS
 *
 * Original Tutorial: http://www.cssreset.com/2010/css-tutorials/how-
 * License: Free - do whatever you like with it! Credit and linkbacks
 *
 * NB: Make sure the value for 'padding-bottom' on #content is equal
 */
html,
body {
  margin:0;
  padding:0;
  height:100%;
}

 #wrapper {
  min-height:100%;
  position:relative;
 }
 #header {
  background:#ededed;
  padding:0px 0px 0px;
 }
 #content {
  padding-bottom:300px; /* Height of the footer element */
 }
 #footer {
  background:#1A1D20;
  height:300px;
  position:absolute;
  bottom:0;
  left:0;
  width: 100%;
}

 .footercopyright {
  text-align: left;
  font-size: 15px;
  padding: 30px 0px 0px;
}

.contactfooter {
 text-align: left;
 font-size: 20px;
 padding:3px;
 color: #b2b2b2;
```

Practice

Assignment

DO
MORE.