

2020

## Lab 3: Object Oriented Programming in Python



Dr. Mohammed Al-Sarem

Taibah University, Information System

Department

2/8/2020

## Lab 3: Object Oriented Prograing in Python

### Lab Objectives:

Python is a class-based language. A class is a blueprint for an object that binds together specified variables and routines. Creating and using custom classes is often a good way to write clean, efficient, well-designed programs. In this lab we will first get familiar with basic structure of classes in Python and then get used to:

- Familiarizing with pass by value and reference
- Instantiating objects and calling methods.
- Creating simple classes and working with objects
- Using Constructors

### Methodology

#### In class task:

At the end of this lab, the student will be able to:

- Define and use Python classes.
- Write a complete body of Python class and how to pass information through class's constructor.
- Instantiate a class and trigger a method inside the class.
- Reuse code in other projects.

### Homework:

### References:

For more information, see:

1. <https://www.w3schools.com/python/>
2. <https://www.programiz.com/python-programming/class>
3. <https://www.youtube.com/watch?v=UumoPVDRtIM>
4. <https://www.youtube.com/watch?v=ZDa-Z5JzLYM>
5. <https://www.youtube.com/watch?v=RSI87lqOXDE>

## Lab 3: Object Oriented Prograing in Python

### 1. Passing Variables by value and reference

**Pass by value** means that the value is directly passed as the value to the argument of the function. In this case, the operation is done on the value and then the value is stored at the address. **Pass by reference** is the term used in some programming languages, where values to the argument of the function are passed by reference, that is, the address of the variable is passed and then the operation is done on the value stored at these addresses. In Python arguments, the values are passed by reference. During the function call, the called function uses the value stored at the address passed to it and any changes to it also affect the source variable. Consider the following code:

#### Exaple 1.1:



In the main function, Did the value of **fav** variable change after invoking the main function? \_\_\_\_\_

Can you explain the reason behind that?

---

---

---

Python uses a mechanism, which is known as "**Cal-by-Object** ", sometimes also called "**Calby Object Reference** " or "**Calby Sharing** "

If you pass immutable arguments like integers, strings or tuples to a function, the passing acts like **Cal-by-value**. It's different, if we pass mutable arguments.

All **parameters (arguments)** in the Python language are **passed by reference**. It means if you change what a parameter refers to within a function, the change also reflects back in the calling function.

#### Exaple 1.2:

```
student={'A':28,'B':25,'C':32,'D':25}
def test(student):
    new={'E':30,'F':28}
    student.update(new)
    print("Inside the function",student)
    return
test(student)
print("outside the function:",student)
```

Are the outputs same in both call? ~~Me~~ your observation below:

## Lab 3: Object Oriented Prograing in Python

Can you explain the difference between the outputs of Example 1 and Example 2??

## 2. Python Classes

Similar to any programming language that support Object-Oriented Concept, Python is built to be a class-based language. A class, in general, a code block that defines a custom object and determines its behavior. To define a class in Python, the first thing that you should use is **Class** keyword which defines and names a new class in Python. Other statements follow, indented below the class name, to determine the behavior of objects instantiated by the class. A class needs a method called a constructor that is called whenever the class instantiates a new object. The constructor specifies the initial state of the object. In Python, a class's constructor is always named `__init__()`. An attribute is a variable stored within an object.

- ❶ **Exercise 2.1:** Create a class **Student** whose two variable, **name** and **list of courses** studied during semester. Initialize the class and display the result on Jupyter.
- Launch the Jupyter as shown before in the previous labs.
  - Paste the code below on cell [].

```
In [ ]: class Student: ❶
        def __init__(self, name): ❷
            self.name=name
            self.course_list= [] ❸
        std=Student("Set_here_your_name") ❹
        print(std.name)
```

Note:

- ❶ The class name **Student** ends with **:**. Without the colons, Python's interpreter will fail to recognize the class body.
- ❷ Function `__init__()` is a constructor in which you can initial local variables of the class. The **"self"** keyword represents the instance of the class. By using **"self"** keyword, we can access the attributes and methods of the class in python. It binds the attributes with the given arguments.

## Lab 3: Object Oriented Programming in Python

- 1 Initialize some attributes
- 1 `std` is an object that instantiates class `Student`. Since we would to pass a value during the initialization, invoking class's constructor and passing the value through it is the right place to set the class parameters.

Hooray, you wrote your first Python class.

### 2.1. Methods

- In addition to storing variables as attributes, classes can have functions attached to them. A function that belongs to a specific class is called a method. Now, backing to what did you learn at the 2<sup>nd</sup> lab, initialize the `__init__` found in `Student` class. To do that:
- Click on the cell [1] where you wrote your previous code. Then, below the `__init__()` function, write the following code:

```
In [3]: class Student:
        def __init__(self, name):
            self.name=name
            self.course_list= []
        def add(self, new_course):
            self.course_list.append(new_course)

std=Student("Set_here_your_name")
std.add("Python")
print(std.course_list)

['Python']
```

**Exercise 2.2** In the function `add()`, the variable `course_list` is initialized by using the `self` keyword as we saw in the previous code. To add a new variable to the list we use the built-in function `append()`. Can you explain why we did that?

\_\_\_\_\_ We use append to insert more value in the last index

\_\_\_\_\_

\_\_\_\_\_.

**Exercise 2.3** Add more courses to your list. Hint! Use loop to ask user to add his preferred course to the list. Then remove a item from the list!

```
In [ ]: class Student:
        def __init__(self, name):
            self.name=name
            self.course_list= []
        def add(self, new_course):
            self.course_list.append(new_course)

std=Student("Set_here_your_name")
txt = input("Type something to test this out: ")
std.add(txt)
print(std.course_list)
```

### 2.2. Inheritance

To create a new class that is similar to one that already exists, it is often better to inherit the methods and attributes from an existing class rather than create a new class from scratch. This creates a class hierarchy: a class that inherits from another class is called a subclass, and the class that a subclass inherits from is called a superclass. To define a subclass, add the name of the superclass as an argument at the end of the class declaration.

```
In [ ]: class Person:
        def __init__(self, fname, lname):
            self.firstname = fname
            self.lastname = lname

        def printname(self):
            print(self.firstname, self.lastname)

class Professor(Person):
    pass
mhd = Professor("Mohammed", "AlSarem")
mhd.printname()
```

Mohammed AlSarem

**Exercise 2.4** In the code above, the class Professor contains a `pass` keyword. What did this mean?


use all the values inside of above class to add to this class used the key word "Pass".

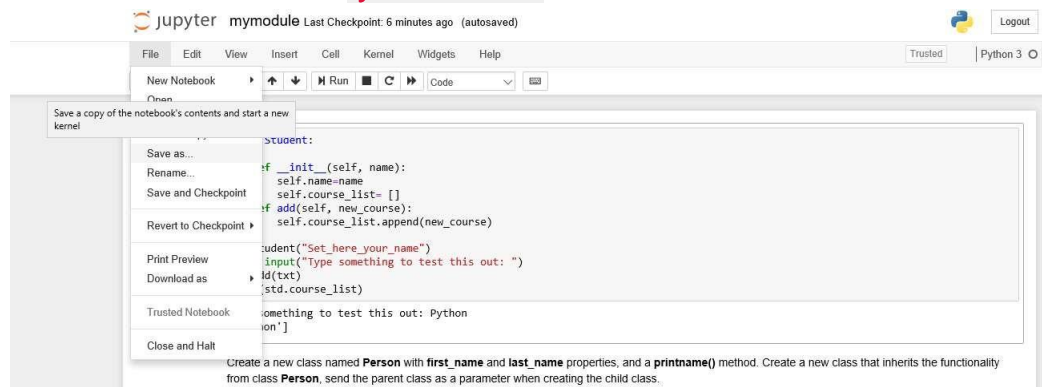
**Exercise 2.5** Replace the `pass` keyword in the child class with `__init__()` function. Note that The child's `__init__()` function overrides the inheritance of the parent's `__init__()` function.

## Lab 3: Object Oriented Programming in Python

**Exercise 2.3** Add a new variable to child class and initialize its value. Print the result. Hint: to keep the inheritance of the parent's `__init__()` function, add `super().__init__()` to the parent's `__init__()` function.

A module is a file containing a set of functions you want to include in your application. To create a module just save the code you want in a file with the file extension `*.py`. To do that:

- Let before saving our module to modify what we did during this lab. Add a new `__init__` function. Separate the class body from the invoking process. Then, go to main menu of your Jupyter editor and click on File  Save as. Then, save this code in a file named `myModule.py`



- The file will be saved in the active directory you set before. By default the file can be found by <http://localhost:8888/tree>
- To use module, we just created, use `import myModule` statement as follows:

`myModule`

