UNIVERSITY OF ENGINEERING AND TECHNOLOGY PESHAWAR, JALOZAI CAMPUS

Lab 4: Classes, objects, and instances

Lab Title: EE-271 "OOP & Data Structures Lab"

Time: 10 min/ Task

Lab report task:

1. Consider the point class below.
   a. Make an object and print its x and y coordinates.

```
import math
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def distance(self, p2):
        return math.sqrt((self.x-p2.x)**2 + (self.y-p2.y)**2)
```

   a. Define point1 and pass two number.
   b. Make another instance of the point class, say its name is p2.
   c. Print p1 and p2, for this use print command and pass the point as input.
   d. Print the coordinate by using object and dot operator.
   e. Add a new method to the point class that can print the points in effective way. Print both the point using that function.
   f. Calculate the distance between these two pints.

Lab work tasks:

1. Motivation

   Interest is to make a list that can store employee data.

   `[name, age, position, the year they started working]`

   For example

```
kirk = ["James Kirk", 34, "Captain", 2265]
spock = ["Spock", 35, "Science Officer", 2254]
mccoy = ["Leonard McCoy", "Chief Medical Officer", 2266]
```

   a. `Print(kirk[0])`

   Hints: First, when you reference kirk[0] several lines away from where the kirk list is declared, will you remember that the 0th element of the list is the employee's name?

   b. `Print(mccoy[1])`

   Hints: What if not every employee has the same number of elements in the list? In the mccoy list above, the age is missing, so mccoy[1] will return "Chief Medical Officer" instead of Dr. McCoy's age.

   A great way to make this type of code more manageable and more maintainable is to use **classes**.

   In the following section a brief introduction to classes, how to make instances/ objects of those classes, and how to use them.

2. Dog class

```
# dog.py
```

```
class Dog:
    pass
```

Creating a new object from a class is called **instantiating** a class. You can create a new object by typing the name of the class, followed by opening and closing parentheses:

a. `Dog()`

This can be assigned to a variable.

b. `Inst=Dog()`

c. Create two new Dog objects and assign them to the variables a and b.

```
a = Dog()
b = Dog()
```

d. When you compare a and b using the == operator, the result is False. Even though a and b are both instances of the Dog class, they represent two distinct objects in memory.

```
a == b
```

e. Use the print command and print both instances/ objects.

3. Following is a class named MyFirstClass.

```
class MyFirstClass:
    pass
```

a. Make two instances named FC1 and FC2.
b. Check whether FC1 and FC2 represent the same object or not.
c. Use the print command and print both instances.

Note:

- The pass keyword on the second line indicates that no further action needs to be taken.
- When printed, the two objects tell us which class they are and what memory address they live at. Memory addresses aren't used much in Python code, but here, they demonstrate that there are two distinct objects involved.

4. Consider the following class definition.

```
class Point:
    pass
```

a. Make two instances named p1 and p2.
b. Print both objects p1 and p2.
c. Try different names for objects, instead of p1 and p2. Also, use different attribute names.

Note:

- The value assigned to attribute can be anything: a Python primitive, a built-in data type, or another object. It can even be a function or another class!

5. Consider the following class.

```
class MyClass:
    """A simple example class"""
    i = 12345

    def f(self):
        return 'hello world'
```

a. `Print: MyClass.i`
b. `Print: MyClass.f`
c. `Run: x = MyClass()`

Where x is called the instance or object.
Note: Class *instantiation* uses function notation. Just pretend that the class object is a parameterless function that returns a new instance of the class.

6. Update the Dog class with an __init__() method that creates .name and .age attributes:

```
# dog.py


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

   a. Run: `miles = Dog("Miles", 4)`
   b. Print `miles`

7. Create a new Dog class with a class attribute called. species and two instance attributes called .name and .age:

   Class attribute: You define class attributes directly beneath the first line of the class name and indent them by four spaces.

```
# dog.p


class Dog:
    species = "Canis familiaris"


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

   a. To instantiate this Dog class, you need to provide values for name and age. If you don't, then Python raises a TypeError:

- `Run: Dog()`

   b. To pass arguments to the name and age parameters, put values into the parentheses after the class name:

```
miles = Dog("Miles", 4)
buddy = Dog("Buddy", 9)
```

   c. After you create the Dog instances, you can access their instance attributes using **dot notation**:

```
miles.name
        miles.age


buddy.name
buddy.age
```

   d. Although the attributes are guaranteed to exist, their values *can* change dynamically:

```
buddy.age = 10
buddy.age


miles.species = "Felis silvestris"
miles.species
```

8. Create a `Car` class with two instance attributes:
   - `.color`, which stores the name of the car's color as a string
   - `.mileage`, which stores the number of miles on the car as an integer

   Then create two `Car` objects—a blue car with twenty thousand miles and a red car with thirty thousand miles—and print out their colors and mileage. Your output should look like this:

```
The blue car has 20,000 miles
The red car has 30,000 miles
```

   Solution: Run the code.

```
class Car:
    def __init__(self, color, mileage):
        self.color = color
        self.mileage = mileage
    def __str__(self):
        return f"The {self.color} car has {self.mileage:,} miles"

blue_car = Car(color="blue", mileage=20_000)
red_car = Car(color="red", mileage=30_000)

for car in (blue_car, red_car):
    print(car)
```

   a. Modify the Car class with an instance method called .drive() that takes a number as an argument and adds that number to the .mileage attribute. Test that your solution works by instantiating a car with 0 miles, then call .drive(100) and print the .mileage attribute to check that it is set to 100.

9. Consider the point class below.

   b. Make an object and print its x and y coordinates.

```
import math
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def distance(self, p2):
        return math.sqrt((self.x-p2.x)**2 + (self.y-p2.y)**2)
```

g. Define point1 and pass two number.
h. Make another instance of the point class, say its name is p2.
i. Print p1 and p2, for this use print command and pass the point as input.
j. Print the coordinate by using object and dot operator.
k. Add a new method to the point class that can print the points in effective way. Print both the point using that function.
l. Calculate the distance between these two pints.

10. Consider the following class definition.

```
class Point:
    pass
```

a. Create two objects and print them.
b. Modify the x and y coordinate using the syntax like:
Obj1.x= 2
Obj1.y =5

Note: Here the attributes are defined after object definition.

c. Add the x and y coordinate objects as the addition of components is needed for vector addition.
d. Add the following function to the class.

You can pass an instance as a parameter in the usual way. For example:

```
def printPoint(p):
  print '(' + str(p.x) + ', ' + str(p.y) + ')'
```

`printPoint` takes a point as an argument and displays it in the standard format. If you call `printPoint(blank)`, the output is `(3.0, 4.0)`.

11. Run and play with the following code.

```
class Rectangle:
  pass
```

And instantiate it:

```
box = Rectangle()
box.width = 100.0
box.height = 200.0
```

This code creates a new `Rectangle` object with two floating-point attributes. To specify the upper-left corner, we can embed an object within an object!

```
box.corner = Point()
box.corner.x = 0.0;
box.corner.y = 0.0;
```

The dot operator composes. The expression `box.corner.x` means, "Go to the object `box` refers to and select the attribute named `corner`; then go to that object and select the attribute named `x`."

a. A function can return an object or instance. Following is a function to determine the center of a rectangle. Find the box to the function and store the value in a variable name center and print it.

```
def findCenter(box):
  p = Point()
  p.x = box.corner.x + box.width/2.0
  p.y = box.corner.y + box.height/2.0
  return p
```

b. The tuple and string are not mutable.

We can change the state of an object by making an assignment to one of its attributes. For example, to change the size of a rectangle without changing its position, we could modify the values of `width` and `height`:

```
box.width = box.width + 50
box.height = box.height + 100
```

Note:

i. Functions that belong to a class are called **instance methods** because they belong to the instance of a class. For example, list.append() and string.find() are instance methods.

ii.

Encouraging

a. Modify the Dog class to include a third instance attribute called coat_color that stores the color of the dog's coat as a string. Store your new class in a script and test it out by adding the following code at the bottom of the script: philo = Dog("Philo", 5, "brown") print(f"{philo.name}'s coat is {philo.coat_color}.") The output of your script should be: Philo's coat is brown.

Recommended Reading

1. https://realpython.com/python3-object-oriented-programming/
2. Book