DSC 5101 – Computer Programming in DSAI

HO 03- Using Class

Opim Salim Sitompul

Department of Data Science and Artificial Intelligence
Universitas Sumatera Utara

Co-funded by the Erasmus+ Programme of the European Union

Outline I

Building Class

- Now that we have already know how to structure our program into functions, we are ready to pack those functions into class.
- Class is an Object-Oriented Programming approach, by which functions and data are packed into one container.
- Functions defined in a class are called methods and should manipulate those data that are located in the same class.

Defining Class

- Class is functioned as *template* which will be used to create *instances* of the class [2].
- Therefore, we can say that an *instance* or object is an example of a class.
- All *instance*/objects of a class have the same data variables but each variable keep different values.
- Every *instance* of a class responses to the same set of requests and has the same behavior.
- Using class, programmers are able to specify the structure and behavior of an object in terms of its attributes or fields, etc., separately from the objects themselves.

Defining Class

- In general, we can define a class in Python, as in the following:

  class *nameOfClass*(*SuperClass*):

      __init__

      attributes

      methods
- __init__ is aspecial method in a class that functions to initialize instances of the class, which also called constructor of the class.
- Attributes are all variables used in the class that can be shared by all methods located in the same class.
- Methods are all function definitions used in the class.

Defining Class

- To add clarity when importing a class into another class or function, it is conventionally agreed that the name of the file containing the class is the same as the name of the class.

Defining Class

- Example 1:

```
''''''
Filename: Array.py
''''''
import random as rnd

class Array:

    def __init__(self, N):
        self.N = N
        self.myarr = []
        for i in range(self.N):
            self.myarr.append(rnd.randint(1,
               self.N))
```

Defining Class

```
    def bubble_sort(self):
        for i in range(self.N):
            for j in range(self.N):
                if self.myarr[i] <
                               self.myarr[j]:
                    temp = self.myarr[i]
                    self.myArr[i] =
                               self.myarr[j]
                    self.myarr[j] = temp

    def print_array(self):
        for i in range(self.N):
            print(self.myarr[i], end=’ ’)
        print()
```

Defining Class

```
def main():
    N = 10
    rnd.seed(a=None, version=2)
    myarr = Array(N)

    print("Before sorted: ")
    myarr.print_array()

    myarr.bubble_sort()

    print("After sorted: ")
    myarr.print_array()

if __name__ == "__main__":
    main()
```

Defining Class

- The filename is the same as the class name, this is good practice, even though not mandatory.
- Pseudo random is imported to facilitate integer random number generator (line 12) as well as the seed for random number (line 29).
- There are three methods in the Array class:
  1. Special function __init__()
  2. Function bubble_sort()
  3. Function printArray()

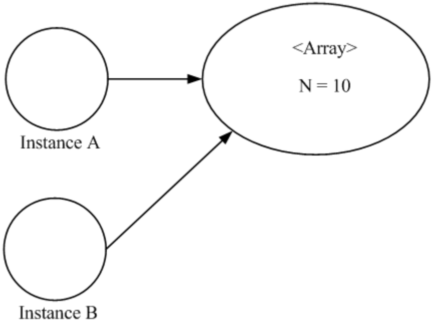
Defining Class

- Special function __init__(): is an initialiser (also known as a *constructor*) for the class.
  - Indicating what data should be supplied when an instance of the Array class is created.
  - How data are stored internally in an instance, which is represented by special variable *self*.
  - Instance variables (referred by *self*) will exist for as long as the object is available.
  - Arguments supplied to a function will only exist locally and disappear whenever exit from the function.
- An instance *myarr* of Array class is created within the main() function (line 30) using:

  myarr = Array(N)

Assigning Instance of a Class

- When an object is created from a class, it holds the address where that instance is located in memory.
- Whenever an object *A* is assigned to another object *B*, then object *B* will receive a complete copy of object *A* by referring to the same address location.



Assigning Instance of a Class

- From the previous example, we can assign another object to the existing object.

```
def main():
    N = 10
    rnd.seed(a=None, version=2)
    myArr1 = Array(N)
    myArr2 = myArr1
    print(myArr1)
    print(myArr2)
    print("Before sorted: ")
    myArr1.print_array()
    myArr1.bubble_sort()
    print("After sorted: ")
    myArr2.print_array()
```

Assigning Instance of a Class

- In line 5, *myArr1* is assigned to *myArr2*
- Printing this two objects will result similar ids

```
<__main__.Array object at 0x0000016A594A55B0>
<__main__.Array object at 0x0000016A594A55B0>
```

- Printing the array using either one of these two objects will give the same result:

```
Before sorted:
5 8 7 3 8 7 1 1 4 1
After sorted:
1 1 1 3 4 5 7 7 8 8
```

Accessing Object Attributes

- Attributes of an object, which are in the form of variables and methods, could be accessed from outside of the class using a dot ('.') notation.
- In the previous example, we accessed two methods print_array() and bubble_sort():

```
    myArr1.print_array()
    myArr1.bubble_sort()
```

Accessing Object Attributes

- We can as well access the data attributes of an object directly using dot (‘.’) notation.
- As in the following function main():

```
def main():
    N = 10
    rnd.seed(a=None, version=2)
    myArr1 = Array(N)

    print("Number of elements: ", myArr1.N)
    print("Before sorted: ")
    print(myArr1.myArr)

    myArr1.bubble_sort()

    print("After sorted: ")
    print(myArr1.myArr)
```

Accessing Object Attributes

- Namely,
  myArr1.N (line 6)
  myArr1.myArr (line 8 and line 13)
- Please note that *N* and *myArr* are two data attributes of the *Array* class.
- Giving the same result:

```
Number of elements: 10
Before sorted:
[10, 1, 5, 5, 9, 1, 8, 6, 3, 10]
After sorted:
[1, 1, 3, 5, 5, 6, 8, 9, 10, 10]
```

Accessing Object Attributes

- Actually, accessing class's attributes directly is not considered as a good object-oriented programming practice.
- Instead, we should use the method mechanism in which all data attributes should only be access by the methods located in the same class.
- In this case, we need to add another simple method that returns *N*.

```
    def return_data(self):
            return self.N
```

Default String Representation

- Detail internal structure of a class could be access using a special method, called __str()__.
- In Python, function name surrounded by a double underscore (__) is considered special function that should be used only to access internal Python's environment.
- Let's add an __str()__ function to the *Array* class.

```
    def __str()__():
            return str(self.myArr) + " has " + str(self.N) +
                " elements."
```

Default String Representation

- The addition of __str()__ function is in fact makes definition of *main()* becomes simpler.

```
def main():
    rnd.seed(a=None, version=2)
    myArr1 = Array()

    print(myArr1)
    myArr1.bubble_sort()

    print("After sorted: ")
    myArr1.print_array()
```

Default String Representation

- In this example, we further modify the *Array* class to generate number of elements randomly, between 1 and any given number.

```
def __init__(self):
  self.N = rnd.randint(1,10)
  self.myArr = []
  for i in range(self.N):
    self.myArr.append(rnd.randint(1, self.N))
```

Adding Documentation to a Class

- In order to provide information on what a class performs, it is a common practice to provide a documentation.
- In Python this kind of documentation could be provided using **docstrings** located just after the class header.
- The **docstrings** can be written in multiple lines using a pair of triple quote string, ” ” ” ... ” ” ”.

```
class Array:
    """
    The Class array generate array elements
     randomly, and then sort the elements
     using bubble_sort()
    """
```

Adding Documentation to a Class

- In this example, the **docstrings** could be displayed from function *main* using __doc__ attribute.

  ```
  print(Array.__doc__)
  ```

- Giving an output such as,

  The Class array generate array elements randomly, and then sort the elements using bubble_sort()

Intrinsic Attributes

- Python created some intrinsic attributes for every class and every object during *runtime*.

Intrinsic Function	
Class	Obj
__name__ name of the class	__c
__module__ name of source library	__d
key-value pairs of object attribute __bases__ colection of base class	
__dict__ key-value pairs of class attribute	
__doc__ documentation string	

References I

[1] VanderPlas J., Python Data Science Handbook: Essential Tools for Working with Data, O'Reilly, USA, 2016.

[2] Hunt, J., A Beginners Guide to Python 3 Programming, Springer Nature Switzerland AG, Switzerland, 2019.

[3] Hunt, J., Advance Guide to Python 3 Programming, Springer Nature Switzerland AG, Switzerland, 2019.