Notes on Chapter 8 - Classes and Object Oriented Programming

Swarup Tripathy *

March 2022

A curated list of important points for my reference.

1. Objects are the core things that Python programs manipulate. Every object has a type that defines the kinds of things that programs can do with that object.
2. An **Abstract Data type** is a set of objects and the operations on those objects.
3. The two powerful mechanisms for managing the complexity of programming are
   - Decomposition → Creates the structure of the program
   - Abstraction → Suppresses the detail
4. One implements data abstractions using **classes**.

: is a slice syntax for every element in the array.

5. When a function definition occurs within a class definition, the defined function is called as **method** and is associated with the class. These methods are sometimes referred to as **method attributes** of the class.
6. **OBJECTS**
   - They have individuality and multiple names can be bound to the same object.
   - Known as Aliasing in other languages.
   - A *Namespace* is a mapping from names to objects.
   - In the expression, *z.real*, *real* is an attribute of the object z.
7. Class supports 2 kinds of operations
   - **Instantiation** is used to create instances of the class.
     For ex., the statement s = IntSet() creates a new object of type IntSet. This object is called an Instance of IntSet.

*John V Guttag

- **Attribute References** use dot notations to access attributes associated with the class. For ex., s.member refers to the method member associated with the instance s of type IntSet.

8. Whenever a class is instantiated, a call is made to the __init__method defined in that class.

9. The __init __method lets the class initialize the object's attributes and serves no other purpose.

```
s=IntSet()
s.insert(3)
print(s.member(3))
```

   creates a new instance of IntSet, inserts the integer 3 into that IntSet, and then prints true.

10. When data attributes are associated with a class we call them **Class variables**. When they are associated with an instance we call them **instance variables**.

11. All instances of user-defined classes are hashable, and therefore can be used as dictionary keys.

12. What actually **Hashable** means in python? Ref: Geeks for Geeks

    - hashable is a feature of Python objects that tells if the object has a hash value or not.
    - If the object has a hash value then it can be used as a key for a dictionary or as an element in a set.
    - An object is hashable if it has a hash value that does not change during its entire lifetime.
    - Python has a built-in hash method (__hash__()) that can be compared to other objects.
    - if the hashable objects are equal then they have the same hash value.
    - All immutable built-in objects in Python are hashable like tuples while the mutable containers like lists and dictionaries are not hashable. An example below

```
t1 = (1, 5, 6)
t2 = (1, 5, 6)
# show the id of object
print(id(t1))
print(id(t2))
######## output ########
```

```
1954294958784
1954294958784
```

13. Abstract Data Types

    - An abstract data type is a type with associated operations, but whose representation is hidden.
    - They lead to a different way of thinking about organising large programs.
    - Data Objects → A data object is a region of storage that contains a value or group of values.
    - writing expressions that directly access instance variables is considered poor form and should be avoided.

14. **Inheritance**

    - It provides a convenient mechanism for building groups of related abstractions.

15. You can use *args and **kwargs as arguments of a function when you are unsure about the number of arguments to pass in the functions.

16. ****kwargs(Keyword Arguments)**

    - allows us to pass a variable number of keyword arguments to a python function. In the function, we use the double asterisk(**) before the parameter name to denote this type of argument.

17. **rindex**: String method finds the last occurrence of the specified value. Example given below

```
name = "Swarup Tripathy"
index = name.rindex(' ')    # index = 6
index2 = name.rindex('a')   # index2 = 11
```

18. **Encapsulation** → Bundling together of data attributes and the methods for operating on them.

19. **Information Hiding** → one of the keys to modularity. Programmers can make the attributes of a class private, so that clients of the class can access the data only through the object's methods. When the name of an attribute starts with __but does not end with __, that attribute is not visibile outside the class.

```
class infoHiding():
def __init__(self):
    self.visible = 'Look at me'
    self.__alsoVisisble__ = 'Look at me too'
    self.__invisible = 'Don\'t look at me directly'

def printVisible(self):
    print(self.visible)
def printInvisible(self):
    print(self.__invisible)
def __printInvisible(self):
    print(self.__invisible)
def __printInvisible__(self):
    print(self.__invisible)

s = infoHiding()
s.printInvisible()      # Don't look at me directly
s.printVisible()        # Look at me
s.__printInvisible()
# AttributeError: 'infoHiding' object has no attribute '__printInvisible'
s.__printInvisible__()  # Don't look at me directly
```

20. **Generators** → the presence of *yield* tells the python system that the function is a generator. Generators are typically used in conjunction with *for* statements.