Sharif University of Technology
Department of Computer Engineering

Fundamentals of Programming

Python Language

Arman Malekzadeh
PhD Candidate in Artificial Intelligence

Table of contents

Object-Oriented Programming

Object-Oriented Programming

- Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects", which can contain data, in the form of fields (often known as attributes or properties), and code, in the form of procedures (often known as methods).

- A feature of objects is an object's procedures that can access and often modify the data fields of the object with which they are associated (objects have a notion of "this" or "self").

- In OOP, computer programs are designed by making them out of objects that interact with one another.

A Class in Python

- A class is a code template for creating objects. Objects have member variables and have behaviour associated with them. In python a class is created by the keyword class.

- An object is created using the constructor of the class. This object will then be called the instance of the class. In Python we create instances in the following manner

```
        instance = class_name(arguments)
```

Terminology

- Class: A user-defined prototype for an object that defines a set of attributes that characterize any object of the class.
- 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.
- Method: A special kind of function that is defined in a class definition.

Class Definition in Python

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

    def myfunc(self):
        print("Hello my name is " + self.name)
```

- The **__init__()** function is called automatically every time the class is being used to create a new object.
- The **self** parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.
- It does not have to be named self , you can call it whatever you like, but it has to be the first parameter of any function in the class.

Class Constructors in Python

- A constructor is a special kind of method that Python calls when it instantiates an object using the definitions found in your class. Python relies on the constructor to perform tasks such as initializing (assigning values to) any instance variables that the object will need when it starts.

- Constructors can also verify that there are enough resources for the object and perform any other initialization tasks you can think of.

- Constructors are typically used to create and initialize the instance variables that represent the object's state. Like any other method, a constructor can accept parameters and use them to initialize the instance variables.

Class Variables in Python: Example

```python
class Person:
    age = 25

    def greet(self):
        print('Hello')
```

We can access the class attribute using **dot** operator like this:

```python
print(Person.age)
```

We can also create a new object of class Person and access age attribute in the following manner:

```python
person = Person()
print(person.age)
```

The concept of Encapsulation

- Encapsulation is one of the fundamental concepts in object-oriented programming (OOP). It describes the idea of bundling data and methods that work on that data within one unit, e.g., a class in Python.

- This concept is also often used to hide the internal representation, or state, of an object from the outside.

- This is called information hiding. The general idea of this mechanism is simple. If you have an attribute that should not be visible from outside the class definition, you can prefix the name of the attribute with a double underscore __.

The concept of Encapsulation: Example

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.__age = age # private attribute

    def greet(self):
        print('Hello, I am ' + str(self.__age) + ' years old')

person = Person('Adam', 19)
person.greet() # Hello, I am 19 years old
print(person.name) # Adam
print(person.__age) #AttributeError: 'Person' object has no attribute '__age'
```

Protected Attributes

- Protected attributes are those attributes that begin with a single underscore _. These attributes are only accessible from within the class and it’s subclasses.
- But, Python does not have any mechanism that effectively restricts access to any instance variable or method. Python prescribes a convention of prefixing the name of the variable/method with a single or double underscore to emulate the behaviour of protected and private access specifiers.
- All the members which are to be considered public should have no underscores or should have a single underscore at the beginning.

Protected Attributes: Example

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

    def greet(self):
        print('Hello')
```

We can create a new object of class Person and access age attribute in the following manner:

```python
person = Person()
print(person._age) # 19
```

References

References I

[1] B Downey, A. (2015). Think Python: How to Think Like a Computer Scientist-2nd Edition.

[2] Deitel, H. M., & Deitel, P. J. (2004). C: How to program. Pearson Educacion.

Sharif University of Technology
Department of Computer Engineering

Arman Malekzadeh

Fundamentals of Programming
Python Language