19CSE201 :Advanced Programming

Lecture 22
Classes & Objects in Python

By
Ritwik M
Assistant Professor(SrGr)
Dept. Of Computer Science & Engg

Based on material from Matthes, E. (2019). Python crash course: A hands-on, project-based introduction to programming. No Starch Press.

A Quick Recap

- Names
- Name Binding
- Namespaces
- Variable Scope
- Examples

Classes & Objects

- Almost everything in Python is an object, with its properties and methods.
- A Class is like an object constructor, or a "blueprint" for creating objects.
- To create a class, use the keyword `class`:
  - ```
    class MyClass:
        x = 5
    ```
- To create an object named p1 of myClass, and print the value of x:
  - ```
    p1 = MyClass()
    print(p1.x)
    ```

The __init__() Function

- To understand the meaning of classes we must understand the built-in `__init__()` function.
- All classes have a function called `__init__()`, which is always/automatically executed when the class is being initiated.
- Use the `__init__()` function to assign values to object properties, or other operations that are necessary to do when the object is being created:
  - ```
    class Person:
      def __init__(self, name, age):
        self.name = name
        self.age = age

    p1 = Person("John", 36)

    print(p1.name)
    print(p1.age)
    ```
    ```

Object Methods

- Insert a function that prints a greeting, and execute it on the p1 object
  - ```
    class Person:
    def __init__(self, name, age):
      self.name = name
      self.age = age

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

    p1 = Person("John", 36)
    p1.myfunc()
    ```
    ```

The self parameter

- 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 Person:
  def __init__(mysillyobject, name, age):
    mysillyobject.name = name
    mysillyobject.age = age
  def myfunc(abc):
    print("Hello my name is " + abc.name)
    print("Hello my age is " , abc.age)
p1 = Person("John", 36)
p1.myfunc()
```

Modify & Delete Object properties

- You can modify properties on objects like this:

```
p1 = Person("John", 36)
p1.age=40
p1.myfunc()
```

- You can delete properties on objects by using the `del` keyword:

```
del p1.age
```

- You can delete objects by using the `del` keyword:

```
del p1
```

Additional Functions

- Apart from using the normal statements to access attributes, we can use the following functions
  - `getattr(obj, name[, default])`
    - to access the attribute of object.
  - `hasattr(obj,name)`
    - to check if an attribute exists or not.
  - `setattr(obj,name,value)`
    - to set an attribute. If attribute does not exist, then it would be created.
  - `delattr(obj, name)`
    - to delete an attribute.

Additional Functions - Example

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

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

p1 = Person("John", 36)
p1.myfunc()
```

```
setattr(p1,'name','Felix')

print(hasattr(p1, 'name'))

print(getattr(p1,'age'))

p1.myfunc()

delattr(p1,'age‘)

print(hasattr(p1,''))
```

Built-In Class Attributes

- Every class keeps the following built-in attributes, and they can be accessed using dot operator like any other attribute
  - `__dict__`
    - Dictionary containing the class's namespace.
  - `__doc__`
    - Class documentation string or none, if undefined.
  - `__name__`
    - Class name.
  - `__module__`
    - Module name in which the class is defined. This attribute is "__main__" in interactive mode.
  - `__bases__`
    - A possibly empty tuple containing the base classes, in the order of their occurrence in the base class list.

Built-In Class Attributes - Example

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

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

print ("Person.__doc__:", Person.__doc__)
print ("Person.__name__:", Person.__name__)
print ("Person.__module__:", Person.__module__)
print ("Person.__bases__:", Person.__bases__)
print ("Person.__dict__:", Person.__dict__ )
```

Access modifiers

- Public
  - The members declared as Public are accessible from outside the Class through an object of the class.
- Protected
  - The members declared as Protected are accessible from outside the class but only in a class derived from it that is in the child or subclass.
- Private
  - These members are only accessible from within the class. No outside Access is allowed.

Public access modifier

- By default, all the variables and member functions of a class are public in a python program.

```
class Employee:
    # constructor
    def __init__(self, name, sal):
        self.name = name;
        self.sal = sal;

emp = Employee("Ironman", 999000);
print(emp.sal)
```

Protected access modifier

- Adding a prefix _ (single underscore) to a variable name makes it protected

```
class Employee:
def __init__(self, name, sal):
        self._name = name;
        self._sal = sal;

class HR(Employee):
def task(self):
        print ("We manage Employees")

HRemp = HR("Ironman", 999000);
print(HRemp._sal)
HRemp.task()
```

Private access modifier

- Adding a prefix __ (double underscore) to a variable name makes it private

```
class Employee:
    # constructor
    def __init__(self, name, sal):
        self._name = name;
        self.__sal = sal;

emp = Employee("Ironman", 999000);

print(emp._sal)
```

Access modifiers - summary

Access modifiers	Same Class	Same package	Sub Class	Other packages
Public	Y	Y	Y	Y
Protected	Y	Y	Y	N
Private	Y	N	N	N

Destroying objects – Garbage Collection

- Python deletes unneeded objects (built-in types or class instances) automatically to free the memory space.
- The process by which Python periodically reclaims blocks of memory that no longer are in use is termed as Garbage Collection.
- Python's garbage collector runs during program execution and is triggered when an object's reference count reaches zero.
- An object's reference count changes as the number of aliases that point to it changes

Garbage Collection Cont.

- An object's reference count increases when it is assigned a new name or placed in a container (list, tuple, or dictionary).
- The object's reference count decreases when it is deleted with del, its reference is reassigned, or its reference goes out of scope.
- When an object's reference count reaches zero, Python collects it automatically.

Garbage Collection Cont.

- The class can implement the special method __del__(),called a destructor, that is invoked when the instance is about to be destroyed.

- This method might be used to clean up any non-memory resources used by an instance

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age
  def myfunc(self):
    print("Hello my name is " + self.name)
   def __del__(self):
    class_name = self.__class__.__name__
print (class_name, "destroyed")
p1 = Person("John", 36)
p1.myfunc()
 print(id(p1))
```

Method Overloading

- Python does not support method overloading by default. But there are different ways to achieve method overloading in Python.

```
def product(a, b):
    p = a * b
    print(p)
 def product(a, b, c):
    p = a * b*c
    print(p)
 product(4, 5)
product(4, 5, 5)
```

Method Overloading Cont.

```
def product(a, b):
    p = a * b
    print(p)

def product(a, b, c):
    p = a * b*c
    print(p)

# product(4, 5)

product(4, 5, 5)
```

```
class Compute:
 def area(self, x = None, y = None):
    if x != None and y != None:
        return x * y
    elif x != None:
        return x * x
    else:
        return 0
obj = Compute()
print("Area Value:", obj.area())
print("Area Value:", obj.area(4))
print("Area Value:", obj.area(3, 5))
```

Operator Overloading

```
#Example1
class Point:
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y


p1 = Point(1, 2)
p2 = Point(2, 3)
print(p1+p2)
```

```
#Example2
class Point:
    def __init__(self,a ,b):
        self.a = a
        self.b = b
     def __str__(self):
        return self.a , self.b
    def __add__(self, other):
        return self.a + other.a,
  self.b + other.b
Ob1 = Point(1, 2)
Ob2 = Point(2, 3)
Ob3 = Ob1 + Ob2
print(Ob3)
```

Operator Overloading - Summary

- We can overload other operators as well. The special function that we need to implement is tabulated .

Operator	Method
**	__pow__(self, other)
>>	__rshift__(self, other)
<<	__lshift__(self, other)
&	__and__(self, other)
|	__or__(self, other)
^	__xor__(self, other)
<	__LT__(SELF, OTHER)
>	__GT__(SELF, OTHER)
<=	__LE__(SELF, OTHER)
>=	__GE__(SELF, OTHER)

Operator	Method
+	__add__(self, other)
–	__sub__(self, other)
*	__mul__(self, other)
/	__truediv__(self, other)
//	__floordiv__(self, other)
%	__mod__(self, other)

Exercises

- Write a Python class to convert an integer to a roman numeral
- Write a Python class to convert a roman numeral to an integer.
- Write a Python class to find validity of a string of parentheses, '(', ')', '{', '}', '[' and ']'.
- These brackets must be close in the correct order, for example "()" and "() [] {}" are valid but "[)", "({[)]" and "{{{" are invalid
- Write a Python class to get all possible unique subsets from a set of distinct integers.
  - Input: [4, 5, 6]
  - Output: [[], [6], [5], [5, 6], [4], [4, 6], [4, 5], [4, 5, 6]]

Exercises

- Write a Python class to find a pair of elements (indices of the two numbers) from a given array whose sum equals a specific target number.
  - Input: numbers= [10,20,10,40,50,60,70], target=50
    Output: 3, 4

- Write a Python class to find the three elements that sum to zero from a set of n real numbers
  - Input array : [-25, -10, -7, -3, 2, 4, 8, 10]
    Output : [[-10, 2, 8], [-7, -3, 10]]

Quick Summary

- Classes
- Objects
- Access Specifiers
- Garbage Collection
- Method Overloading
- Operator Overloading
- Examples
- Exercises

Up Next

Inheritance in python