TECHelium

Lets Face it, now

Project Oriented Python

Object oriented Programming

Oops

- Python is an object-oriented language.
- Every piece of data and even functions and types are objects.
- We are using the new object syntax:
  - object.method( )
- meaning that the method associated with the object's type is applied to the object. This is just a special syntax for a function call with an object.

Basic Oops terminology

- **Class:**
  - A group of variables and function that act on these variables.
  - These are called attributes
  - The attributes are data members (class variables and instance variables) and methods.
  - Attributes are accessed via dot notation.
  - Example: strings and list we use in python are class
    - S = "something"
    - S.upper(), S.lower() are all methods
  - Example: Animal and bird could be a class where in animal can have attributes like legs, hands, tail, teeth, etc and bird can have attributes like feather, beak , etc

Basic Oops terminology ….continue

- **Object:**
  - Object is an instance of class.
  - Example student = “Name1”
    - Student is object of class string.
  - All fields/variable and function defined in class becomes the attribute of object
    - student.upper() is a Methods of object student
- **Class variable:**
  - A variable that is shared by all instance of a class. Usually defined outside of methods
- **Instance variable**:
  - A variable that belong to only individual instance of class. Usually defined inside a method

Creating class

- The ***class*** statement creates a new class definition.

```
class  <class name> :
      'Class document describing class'
      class attributes
```

- Example:

```
class  Device :
     'This class define the network devices'
     device_count = 0  # common to all instance
     def __init__(self, type, ip_address) :
            self.type = type
            self.ip  = ip_address
            Device.device_count += 1
```

device_count is a class variable.

__init__: is a special function called constructor

type and ip are instance variable

Key points

- Constructor are called by python when object is created.
- All functions/methods by default has self as first parameters.
- Self is not used while calling the function.
- Python add the self variable to the list of argument with object name when calling methods of class
- All instance variable are called with object dot variable.
- All class variable are called with ClassName dot variable

Using class

```
Dev = Device(‘cisco_router’,’192.168.1.1’)
 print Dev.ip, Dev.type
 Dev.ip = ‘192.168.101.10’
 Dev.type = ‘cisco_switch’


 Dev.password = ‘cisco10’   # you can add new attribute on fly


 del Dev.password            # you can delete attribute on fly
 print Device.device_count   # will print 1
```

Destructor

```
def __del__(self) :
    Device.device_count -= 1
```

- Python will call destructor when the object is deleted
- Router = Device(‘newrouter’, ‘192.100.12.111’)
- print Device.device_count   # will print 1
- print Router.device_count # will print 1
- del Router
- print Device.device_count   # will print 0
- Usually used for clean up

Functions

- hasattr(router, ‘prompt') # Returns true if ‘prompt' attribute exists
- getattr(router, ‘ip') # Returns value of ‘ip' attribute
- setattr(router, ‘password', ‘cisco1’)
  # Set attribute ‘password‘ if present or add new attribute
- delattr(router, ‘prompt') # Delete attribute ‘prompt‘

Build in attribute

- __dict__ :   Dictionary containing the class's namespace.
  - Example: print router.__dict__   ##{‘ip’ : ‘1.1.1.1’, ‘type’:’cisco’}

- __doc__ :   Class documentation string or None if undefined.
  - Example: print router.__doc__   ## will print ‘Class document describing class’

- __class__. __name__: To find class name.
  - Example: print router.__class__.__name   # will print NewClass

- __module__: To find module name in which the class is defined.
  - Example: print router.__module__   #will print __main__

Class Inheritance

- Attribute of existing class can be used, instead of writing attributes for new class
  - Class Router(Device):

```
    r_count = 0
    def __init__(self, name,IP,dev_type):
        Router.r_count += 1
         self.name = name
         Device.__init__(self,dev_type, IP)
```

  - Now we can use

```
Dut = Router(‘cisco100’, ‘cisco’, ’192.168.1.1’)
Print Dut.ip, Dut.type, Dut.name, Dut.r_count, Dut.device_count
```

- We inherits class when we need most of the attributes of existing class, but with new attributes.

Overriding

- Overriding parent method.
  - A new method in child class will override parent method.
  - Example: if parent class has function add_two and child class define new function add_two, then it will override parent method.
- In actual, by defining the constructor, __init__ function in child class we are overriding parent method

Overloading operator

- To understand overloading operator, we need to understand how operators work in python
- There is a special method for every operator sign.
- The method for the "+" sign is the __add__ method.
- If we have an expression "x + y" and x is an instance of class K, then
  - Python will check the class definition of K.
  - If K has a method __add__ it will be called with x.__add__(y),
  - otherwise Python will show an error message.

Binary operator method

Operator	Method
+	object.__add__(self, other)
-	object.__sub__(self, other)
*	object.__mul__(self, other)
/	object.__div__(self, other)
%	object.__mod__(self, other)
+=	object.__iadd__(self, other)
-=	object.__isub__(self, other)
*=	object.__imul__(self, other)
/=	object.__idiv__(self, other)
%=	object.__imod__(self, other)
<	object.__lt__(self, other)
<=	object.__le__(self, other)
==	object.__eq__(self, other)
!=	object.__ne__(self, other)
>=	object.__ge__(self, other)
>	object.__gt__(self, other)

Exercise

1.1 Write a class for your project and overload a operator.

Thanks