Classes, Objects and OOP

- What is a OOP
- Creating Instance Objects
- Constructor and Destructor
- Class/Static and Instance variable
- Destroying Objects
- Creating a Class: Syntax
- Adding Attributes
- Adding Methods
- Built-In Class Attributes

What is a OOP

- OOP is Object Oriented Programming , representing objects as in real world.
- Everything in OOP is represented using **Classes** and **Objects**
- A **class** is a concept whereas, an **Object** is an actual thing or an actual instance of a class that exists in memory.
- Class defines a template on the basis of which an object gets created.

trainer.cpp@gmail.com

What is a Class

- A **class** is a way of binding **data** and **methods/Operations** on the data together.
- Class represent OOP (Object Oriented Programming).
- Syntax of a class in python :

```
class <Name of Class>:
    ... # method or attribute definition
    ...
```

- Class Names are identifiers

trainer.cpp@gmail.com

Creating Instance Objects

- Syntax of creating an Object:

```
<object_name> = <class_name>(zero or more arguments)
```

- Example:

```
l = list()              # creates empty list object
l = list([1,2])         # takes one argument-another list object
class MyClass:
    pass
ob = MyClass()          # create an object of the class MyClass
```

trainer.cpp@gmail.com

Adding Attributes

- Attributes refer to the data available or attached to an instance/object of a class.
- We can create a class **Person**, with attributes : **name** and **age**.
- The attributes of an object are accessed using the **dot (.)** notation in python

```
p = Person()
p.name        # access the attribute name in p
```


Constructor and Destructor

- Constructor and Destructor are special methods in terms of object oriented programming, used for managing objects
- Constructor defines some block of code that should get executed when any new instance of a class is created or whenever an object is instantiated.
- Destructor is the opposite of constructor and executes when object is destroyed.

Constructor and __init__ method

- In python work of constructor is done by special **__init__** method.
- It takes a **self** argument, apart from other arguments, which acts as a reference to the object that is being created.

```
class <class_name>:
    def __init__(self, <other arguments if needed>):
        # code for construction
```

* **Update** the person class with the __init__ method.

****Self** is just a notational convention, you can use any other name, but better to stick to self

Destructor and __del__()

- The code for **Destructor** in python goes into the **__del__(self)** method.
- So the **__del__** method is invoked only when the object is garbage collected.
- Since python uses reference counting mechanism to keep track of objects, your object may never be destroyed, till the program terminates.

* Add a destructor to the **Person** class

Adding Methods

- Methods are added to a class, just by defining the method inside the class.
- The methods usually take **self** as the first argument.
- Methods are liken normal functions, except that they are bound to the class in which they are defined, and hence also bound to the objects of the class.

* add a **print()** method to the **Person** class, to print details.

** also add a method **celebrateBirthday()** to increase the age of the person

The Note Class

- Create a Note class with following attributes
  - *id*
  - *title*
  - *content*
  - *timestamp*
- Of the above attributes, initialize the first three from the *constructor* and the ***timestamp*** using the ***datetime*** modules' ***now()*** method

** Make sure you use the same case as used here.

Note Continued

- Add a method **search()**

  This method should take a string pattern *as argument and return* True *if the* pattern *is found in either the* title *or* content

- search(pattern) -> True/False

Adding Doc string to the Note class

- Add doc string on class and method level
- Use the triple Quote string.

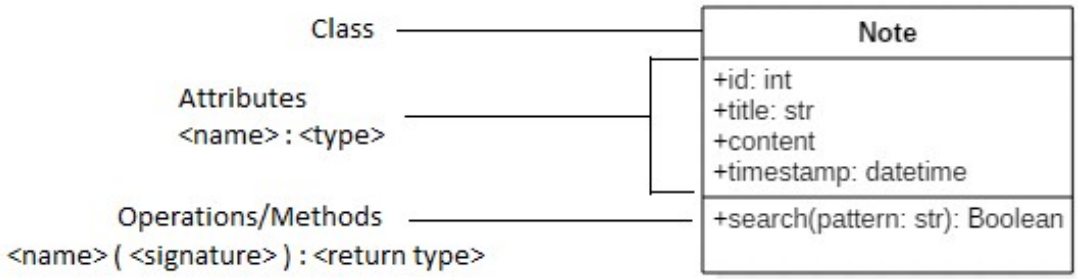
- *class Note:*

  """

  Note class keeps track of your notes

  """

Reading a Class Diagram



- **First section** : **name** of **class**
- **Second section : Attributes/Data members**
- **Third section : Methods/Operations**

Class/Static and Instance variable

- Attributes can be bound to both **class** and an **instance.**
- **Class** and its **instance** are both separate namespaces
- **Class attributes** can be created directly inside the class like method, or can be assigned later.
- **Instance attributes**, like class attributes, can be attached to the object in methods, using the self argument, or directly to the instances.

* implement a class **Circle**

Built-In Class Attributes

- __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 set to "__main__" in interactive mode.
- __bases__ :A possibly empty tuple containing the base classes, in the order of their occurrence in the base class list.

Destroying Objects

- Objects or rather labels can be deleted from a scope using the **del**

  del <object_name>

- Using this syntax removes the reference from current scope, but it does not guarantee the actual deletion of the underlying object.
- Using del just decreases the reference count for that object.