Gaurav Gupta

Classes

- Class Syntax
- Writing simple class
- Creating objects
- Class vs Object

tuteur.py@gmail.com

Gaurav Gupta

Class Syntax

```
class <YourClassName>:
        # methods of class after one level of indentation
```

- Class names are **identifiers**.
- Class is a way of binding data and operations.
- This however looks different in python.

tuteur.py@gmail.com

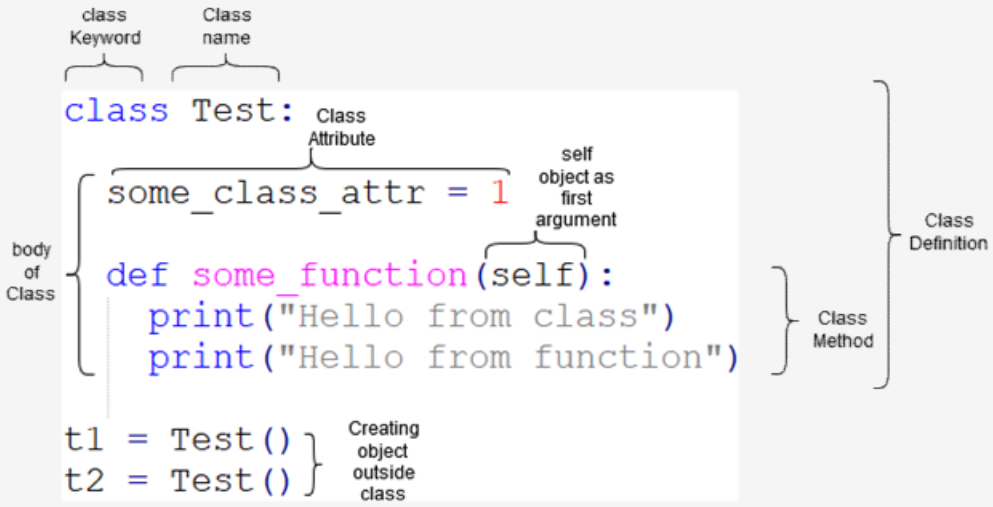
Gaurav Gupta

What to do once you have a class

- Classes are generally meant for object/ instance creation.
- Instances are created in python, using class name and function call operator.
- While creating objects, there may be some arguments, just like functions.
- Syntax of creating an Object:
  <object_name> = <class_name>(zero or more arguments)
- **Example:**

```
s = str()          # creates without argument
s = str(100)       # give one argument: another list object
```


Gaurav Gupta



Gaurav Gupta

Adding Attributes

- Attributes refer to the data available or attached to an instance/object of a class.
- The attributes of an object are accessed using the **dot (.)** notation in python
- *Create a class **Contact**, with attributes : **name, phone and email**.*

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


Gaurav Gupta

Constructor and Destructor

- Constructor and Destructor are special methods in OOP, used for managing objects creation and destruction.
- 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.

Gaurav Gupta

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 is a reference to the newly created object.

```
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

Gaurav Gupta

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

Gaurav Gupta

Operator Overloading in Python

- Operators are defined for types like integers, floats, lists …
- Ex: 1>2 ; 1+2 ;

```
l = [1,2,3,4]
print(l)
```

- But for custom classes, these operations have to be defined.

tuteur.py@gmail.com

Gaurav Gupta

Operator Overloading with ComplexClass

- Implement a class **ComplexNumber** that contains following attributes and methods:

  re : attribute for real part

  im : attribute for imaginary part

- Define a method **show(),** that displays the attributes of the class object
- Also define a method **add()**, that takes another **Complex** Object and returns a **new Complex Object** containing the **sum** of two objects.

tuteur.py@gmail.com

Gaurav Gupta

Operator	Expression	Internally
Addition	p1 + p2	p1.__add__(p2)
Subtraction	p1 - p2	p1.__sub__(p2)
Multiplication	p1 * p2	p1.__mul__(p2)
Power	p1 ** p2	p1.__pow__(p2)
Division	p1 / p2	p1.__truediv__(p2)
Floor Division	p1 // p2	p1.__floordiv__(p2)
Remainder (modulo)	p1 % p2	p1.__mod__(p2)
Bitwise Left Shift	p1 << p2	p1.__lshift__(p2)
Bitwise Right Shift	p1 >> p2	p1.__rshift__(p2)
Bitwise AND	p1 & p2	p1.__and__(p2)
Bitwise OR	p1 | p2	p1.__or__(p2)
Bitwise XOR	p1 ^ p2	p1.__xor__(p2)
Bitwise NOT	~p1	p1.__invert__()

Operator	Expression	Internally
Less than	p1 < p2	p1.__lt__(p2)
Less than or equal to	p1 <= p2	p1.__le__(p2)
Equal to	p1 == p2	p1.__eq__(p2)
Not equal to	p1 != p2	p1.__ne__(p2)
Greater than	p1 > p2	p1.__gt__(p2)
Greater than or equal to	p1 >= p2	p1.__ge__(p2)

tuteur.py@gmail.com

Gaurav Gupta

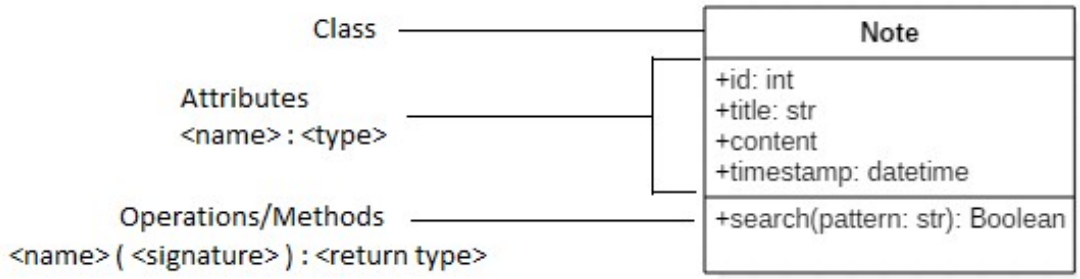
Class vs Instance Attributes

- Attributes can be bound to either **class** or its **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**, are attached to the object.
- When looking a variable using object first it is searched in the objects namespace, if not found, then in the class namespace.

tuteur.py@gmail.com

Gaurav Gupta

UML Notation



Gaurav Gupta

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.