- **Defining methods**

The method is a function that is associated with an object. In Python, a method is not unique to class instances. Any object type can have methods.

- **Differences between functions and methods**

Python Method

- Method is called by its name, but it is **associated to an object** (dependent).
- A method is **implicitly passed the object** on which it is invoked.
- It **may or may not return any data.**
- A method **can operate on the data (instance variables) that is contained by the corresponding class**

Basic Method Structure in Python :

```
# Basic Python method
class class_name
    def method_name () :
        ......
        # method body
        ......
```

Example:

```
class myclass:
   def first(self):
      print("i am in first method of myclass Class")


mc=myclass()
mc.first()
```

output:

i am in first method of myclass Class

Functions

- Function is block of code that is also **called by its name**. (independent)
- The function can have different parameters or may not have any at all. If **any data (parameters)** are passed, they are **passed explicitly**.
- It **may or may not return any data.**
- Function does not deal with Class and its instance concept.

Syntax:

```
def function_name ( arg1, arg2, ...) :
    ......
    # function body
    ......
```

Example:

```
def Subtract (a, b):
   return (a-b)
```

```
print( Subtract(10, 12) )
print( Subtract(15, 6) )
```

Difference between method and function

- Simply, function and method both look similar as they perform in almost similar way, but the key difference is the concept of '**Class and its Object**'.
- Functions can be called **only by its name**, as it is defined independently. But methods c**an't be called by its name** only, we need to invoke the class by a reference of that class in which it is defined, i.e. method is defined within a class and hence they are dependent on that class.

- **Instance method**

Instance methods are the most common type of methods in Python classes. These are so called because they can access unique data of their instance.

Example:

```
class student:
    clg= "JNTUK"
    def __init__(self, name, age):
        self.name= name
        self.age= age

    def info(self):
        print("name:",self.name ,"age:",self.age)
s1=student("naveen",23)
s1.info()
```

output:

name: naveen age: 23

- **Static method**

  Static methods are methods that are related to a class in some way, but don't need to access any class-specific data. You don't have to use **self**, and you don't even need to instantiate an instance, you can simply call your method:

static methods are created using the **@staticmethod** decorator.

- **Class method**
  - Class methods don't need **self** as an argument, but they do need a parameter called **cls**. This stands for **class**, and like self, gets automatically passed in by Python.
  - Class methods are created using the **@classmethod** decorator.

Example:

```
class student:
    clg= "JNTUK"
    def __init__(self, name, age):
        self.name= name
        self.age= age

    def info(self):
        print("name:",self.name ,"age:",self.age)

    @classmethod
    def college(cls):
        return cls.clg

    @staticmethod
    def display(age):
        return age

s1=student("naveen",23)
s1.info()
print(s1.college())
print(s1.display(25))
```

output:
name: naveen age: 23
JNTUK
25

- **Difference static and class methods**
- Class method takes cls as first parameter, while static method need no specific parameters.
- Static methods knows nothing about the class state, while class methods can access and modify class state.
- @classmethod decorators are used to create class method,@staticmethod decorators are used to create static method.