OOPs in Python

Methods : Methods are the function that are defined inside a class.

· Creating and work with methods in Python is the same way that you do functions, except that methods are always associated with a class.

· We can create two kinds of methods: those associated with the class itself and those associated with an instance of a class.

Creating a Class Method : A class method is one that you execute directly from the class without creating an instance of the class.

· To create methods that execute from the class, such as the functions you used with the str class in order to modify strings.

Example:

```
class Myclass:
    def One():
    print ("Welcome to Digital Lync")
Myclass.One()
```

· Example class contains a single defined attribute, One().

· This cannot accept any arguments and does not return any values.

· Notice that here we haven’t created any instance to class as methods is available for use.

Output:

```
Welcome to Digital Lync
```

Creating Instance Method : An instance method is one that is part of the individual instances.

· Instance methods are used to manipulate the data that the class manages.

· Instance methods cannot be used until we instantiate an object from the class.

· All instance methods accept a single argument as a minimum (self).

· The self argument points at the particular instance that the application is using to manipulate data.

· Without the self argument, the method wouldn’t know which instance data to use. However, self isn’t considered an accessible argument — the value for self is supplied by Python, and you can’t change it as part of calling the method.

Example:

```
class Myclass:
    def One():
    print ("Welcome to Digital Lync")
I = Myclass()
I.One()
```

· Above example will return a error as “TypeError: One() takes 0 positional arguments but 1 was given”.

· To overcome this error just provide built-in self argument in the One method.

```
def One(self):
```

Output:

```
Welcome to Digital Lync
```

<u>By using Return</u> : We can use return for getting the output from the class.

Example:

```
class Myclass:
    def One(self):
    return ("Welcome to Digital Lync")
I = Myclass()
print(I.One())
```

Ouput: Welcome to Digital Lync

<u>Static Methods</u> : The method was marked with a @staticmethod decorator to make it as a static method.

· This type of method will not take self argument and class arguments.

· A static method can neither modify object state nor class state. Static methods are restricted in what data they can access – and they're primarily a way to namespace your methods.

Example:

```
class Myclass:
    @staticmethod
    def staticmethod():
        return ("This is a static method")
obj = Myclass()
print(obj.staticmethod())
```

Output:

```
This is a static method
```

· This is a static method which work fine in class without passing class arguments and self argument.

· Let us discuss another example which works on three different methods.

Example:

```
class Myclass:
    def method(self):
        return "instance method called",self
    @classmethod
    def classmethod(cls):
        return "class method called",cls
    @staticmethod

    def staticmethod():

        return "This is a static method"

obj = Myclass()

print(obj.method())

print(obj.classmethod())

print(obj.staticmethod())
```

· @static method and @class method decorators are available in python 2.4 for those using python2.x version.

· Calling these method is also same as normal calling.

Output :

```
('instance method called', <__main__.Myclass object at 0x00000175A955C4E0>)

('class method called', <class '__main__.Myclass'>)

This is a static method
```

· Another way through which we can call these methods is:

```
print(obj.classmethod())

print(obj.staticmethod())

print(obj.method())
```

Output:

```
('class method called', <class '__main__.Myclass'>)

This is a static method

Traceback (most recent call last):

    File "classes.py", line 26, in <module>

        print(Myclass.method())
```

TypeError: method() missing 1 required positional argument: 'self'

· Here it will give error at obj.method() because as we specified and argument (self) then it consider it as an argument and return an error to enter the value.