

Ch 10 : Object Oriented Programming

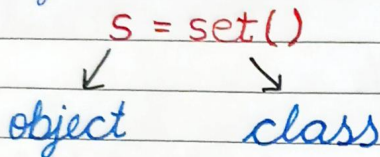
Object oriented programming focuses on the reusability of code.

Class

A class is a blueprint or a layout for creating objects. We use classes. eg - set()

Object

An object is an occurrence of a class



Object of any class uses the methods of that class. All attributes and methods will be used of the class in the object.

Syntax

class Sample :

Functions / variables

Attribute

class BoatTail :

price = "\$ 28M" → an attribute of this class
company = "Rolls Royce"

elon = BoatTail() → New instance of the class.

`elon.owner = "Elon Musk"` → New attribute of this instance

`elon.price = "$30M"` → price attribute replaced

`print(elon.price)` → "\$30M"

`BoatTail.carColour = "Black"` → New attribute of class
`print(BoatTail.carColour)` → Black

- * This attribute will be assigned to all new objects created.

'self' parameter

self is a parameter passed into the function of a class.

class BoatTail:

price = "\$28M"

company = "RR"

def carTesting(self):

return "Car is tested"

`elon = BoatTail()`

`print(elon.carTesting())` → "Car is tested"

- * If we didn't use the self, it would produce an error saying an unknown parameter is given.

self is the object on which the function will be used.


```
def receipt(self):
```

```
    return f"Thank you Mr. {self.owner} for buying the  
car.\n It costs {self.price}"
```

* Here, `self.price` means `elon.price` and `self.owner` is `elon.owner`

`print(elon.__dict__)` → Returns all the attributes with values.

□ `--init--` constructor

`--init--()` is a special function used in classes

```
class BoatTail:
```

```
    def __init__(self, price, owner):
```

```
        self.amount = price
```

```
        self.name = owner
```

```
        print(f"Price is {self.amount} \n The owner is {self.name}  
        \n Thanks")
```

```
elon = BoatTail("$30M", "Elon Musk")
```

```
    ↳ self = elon
    ↳ price = "$30M"
    ↳ owner = "Elon Musk"
```

Static Function

When we don't want any parameter in a function of classes, we use static method.

`@staticmethod` → decorator

```
def greet():  
    print("Happy birthday")
```

* We need not have to use self here.

* We can add other parameters just as we do normally and work on them.

□ `if __name__ == "__main__":`

`--name--` is a special variable that is set to `"__main__"` by default. It represents the module name.

If we use this condition and write some code, it will be ran only if the same code is ran from command line. If this module is imported into other code, the code under this snippet won't be executed.

□ global keyword

global keyword is used to modify variable outside current level.