Python Fundamentals day 44

Today's Agenda

- Object orientation

Object orientation

So far we have seen the structured style of programming language where we used different functions to perform operations. Now it's time to have a look at providing solutions using the classes and the objects.

For example: Imagine you are standing in a queue for getting the food every time. So suddenly you have a eureka moment and plan to have a mobile application into which all the food items are present. All you have to do is click on whichever food item you like and get it delivered at your doorstep. So this person goes to a software company and proposes this idea, where upon clicking on the food item it should ask from which store you want to purchase it and then track down your location after which mode of payment is selected and within few minutes you can enjoy the food at your desired location.

For all this to work software developer should see this in an object oriented perspective where different objects are interacting with each other, in this case customer, restaurant and delivery agent are the objects interacting with one another via app.

For software to recognize real time entities as objects it should know the state and behaviour of objects.

Customer

State:

- Name
- Email
- Phone_num
- Address

Behaviour:

- Sel_restaurant()
- Sel_item()
- Sel_address()
- Place_order()
- Cancel_order()
- Make_payment()
- Avail_discount()

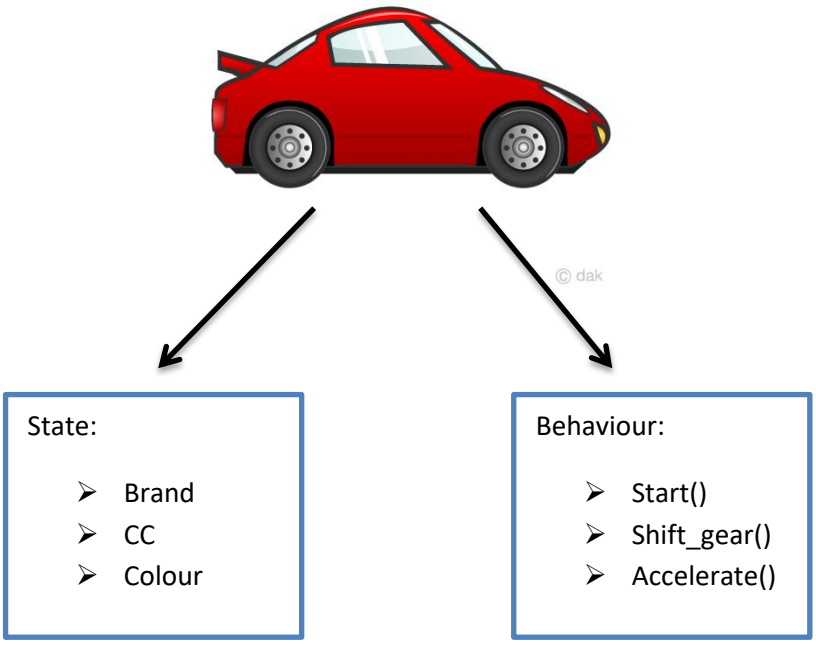
State:

- Name
- Address
- Ratings
- Item

Behaviour:

- Confirm_order()
- Add_item()
- Del_item()

Now let us see how to implement these in python considering the following example




```
class Car():
    #states of object
    def __init__(self):
        self.brand='BMW'
        self.cc=2100
        self.color='blue'

    #behaviours of object
    def start_engine(self):
        print('engine is starting')

    def shift_gear(self):
        print('shifting gear')

    def accelerate(self):
        print('car is accelerating')

def main():
    c1=Car()
    print(c1.brand)
    print(c1.cc)
    print(c1.color)
    c1.start_engine()
    c1.shift_gear()
    c1.accelerate()

if __name__=='__main__':
    main()
```

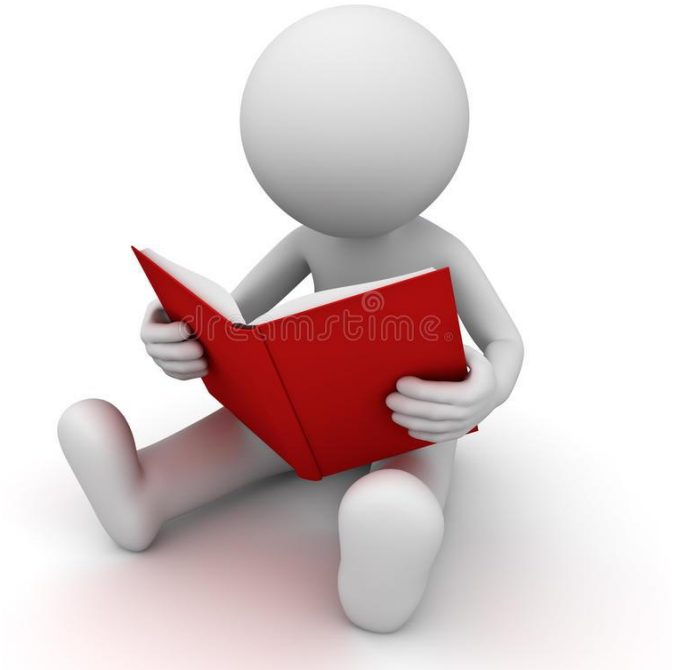
Output:

```
In [3]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')
BMW
2100
blue
engine is starting
shifting gear
car is accelerating
```

You must be wondering why should we pass self as the argument and is it mandatory to use it?

Certainly using self as argument is mandatory because internally c1 object is passed as argument which contains the different states of object. So, not using self will throw an error. But replacing self with any other character will definitely work. The only issue will be not following the conventions of python.

We know that there exist multiple objects in real life. So let us see if the above code could have another object in it



```
class Car():
    #states of object
    def __init__(self):
        self.brand='BMW'
        self.cc=2100
        self.color='blue'

    #behaviours of object
    def start_engine(self):
        print('engine is starting')

    def shift_gear(self):
        print('shifting gear')

    def accelerate(self):
        print('car is accelerating')

def main():
    c1=Car()
    print(c1.brand)
    print(c1.cc)
    print(c1.color)
    c1.start_engine()
    c1.shift_gear()
    c1.accelerate()

    c2=Car()
    print(c2.brand)
    print(c2.cc)
    print(c2.color)
    c2.start_engine()
    c2.accelerate()
    c2.shift_gear()

if __name__=='__main__':
    main()
```

Object:

```
In [4]: runfile('C:/Users/rooman/OneDrive/Desktop/python/
test.py', wdir='C:/Users/rooman/OneDrive/Desktop/python')
BMW
2100
blue
engine is starting
shifting gear
car is accelerating
BMW
2100
blue
engine is starting
car is accelerating
shifting gear
```