OOPs

We were writing our code in a procedural way. This is not how the real-world programing is.

In 1970, we had languages like, C, Cobol, Pascal, Fortran. People wrote huge files having 10,000 line of codes. A very big file having so many functions. It was not east to debug the file. Then a Danish Computer Scientist, name **dBjarne Stroustrup** developed C++ as an extension of C language. It was a better version of C. in this he introduced ***classes***. This helps developers to separate the code into separate files. The whole 10,000 lines code is separated into these classes so a new developer who has joined recently can look into these classes and easily understand it.

Since 1970, developers have been trying to modelling the real world into code. Ex: airport

Steps:

- Enter the building
- Security check
- Boarding
- Fly
- Land
- Baggage collection
- Leave

In all these steps, we see the use of computers. Even when we book a flight, we use a computer. There is an inch of programming. So, now-a-days programming is to solve the real-world problems. For this reason, classes were used.

Classes are tools which helps us to model real world into code.

To model a student, we need to mention the **attributes** of the students and then the **actions (Methods)** which the student can perform.

Class Student: has two things, **attributes** and **methods**

Ex: Attributes of an Airplane

- Wheels
- Color

- Brand
- Wings
- Seats
- Engine

Methods of an Airplane

- Flying
- Take-off
- Emergency landing
- Landing

It’s good to thinking about classes for every program we write.

The number one rule before coding is to make sure it looks beautiful once you have finished. Be very professional. Keep your code clean and make sure that other developers will understand the code that you’ve written. Feel every single word that you type, let your code breath.

Rule #1: - each and every **class** should be in a **separate** file irrespective of its length.

File name = Car_Manager
class name = CarManager

A file named as main.py is the file from which your code starts. So, lets save a file as student.py. it should have a class of student. It is written as

class Student:

pass → ignore it, will be filled later.

whenever you create a class, it should be dunder method.

__init__: this is pronounced as dunder init also called a constructor. Any calss you make, it’s a good practice.

```
class Student:
    def __init__(self):
        pass
```

All the attributes of a class will be written in the dunder init. The first argument is self and it means the student. Self is a good practice.

To create an attribute, type **self.name = "Shyam"**

```
class Student:
    def __init__ (self):
        """
        attrs:
        name: denotes the name of the student.
        age: denotes the age of the student.
        birth_year: denotes the birth year of the student.
        """
        self.name = "Shyam"
        self.age = 10
        self.birth_year = 1997
```

For now, we have given 3 attributes to the class student.

Student's function is to read. So,

```
def read (self):
    print ("student is reading!!")
```

Whenever we use def read (), it will print that the student is reading.

With "print" if we put an 'f' before a string and put an attribute in it, it will replace it.

```
def read (self):
    # read: read a method of studen class.
    print (f "student{self.name} is reading!!")
```

'f' stands for formatting.

Student can write as well. So,

```
def write (self):
    # write: write is a method of student class.
    print (f "student{self.name} is writing!!")
```

Now, in my main.py file, I want the student class. So, I will import the class student from a different file.

```
from .student import Student # importing
```

This means we are loading the code from a different file. '.' is the current directory.

Always start your code under,

```
if __name__ == "__main__":
    pass
```

The other benefits would be when you are importing modules.

Understanding:

We have created a class student. We have given it some attributes and methods to the class students. But we have only declared the class.

- We have not yet created an instance of this class,
- Or called the class
- Or yet made an object of this class.

We call a class by its name and (). Hence,

```
from student import Student # importing

if __name__ == "__main__":
    kunal = Student()  # we are making an object of the type student.

    print(kunal.age)
    print(kunal.name)
    print(kunal.birth_year)
```

OUTPUT:
10
Shyam
1997

Now kunal is object of the class student and kunal will have a name, age and birth_year. We can call kunal as **instance** or **object** or **type** of **class student**.

The same way as we call a function is the same way we call the method.

```
from student import Student

if __name__ == "__main__":
    kunal = Student()

    print(kunal.age)
    print(kunal.name)
    print(kunal.birth_year)
    kunal.read()
    kunal.write()
```

OUTPUT:

```
10
Shyam
1997
student Shyam is reading!!
student Shyam is writing!!
```

Let us create few parameters.

IN **student.py**

```
class Student:
    def __init__(self, name, age, birth_year):
        """
        attrs:
        name: name of the student.
        age: age of the student.
        birth_year:  birth year of the student.
        """
        self.name = name
        self.age = age
        self.birth_year = birth_year
```

OUTPUT:

```
25
Kunal
1998
student Kunal is reading!!
student Kunal is writing!!
```

IN **main.py**

```
from student import Student

if __name__ == "__main__":
    kunal = Student("Kunal", 25, 1998)

    print(kunal.age)
    print(kunal.name)
    print(kunal.birth_year)
```

If we add another student;

```
from student import Student

if __name__ == "__main__":
    kunal = Student("Kunal", 25, 1998)
    abhijit = Student("Abhijit Sharma", 24, 2001)

    print(kunal.age)
    print(kunal.name)
    print(kunal.birth_year)
    kunal.read()
    kunal.write()

    print(abhijit.age)
    print(abhijit.name)
    print(abhijit.birth_year)
    abhijit.read()
    abhijit.write()
```

OUTPUT:

```
25
Kunal
1998
student Kunal is reading!!
student Kunal is writing!!
24
Abhijit Sharma
2001
student Abhijit Sharma is reading!!
student Abhijit Sharma is writing!!
```

INPUT:

```
from student import Student # importing

if __name__ == "__main__":

    #Student Kunal
    kunal = Student("Kunal", 25, 1998)

    print(f"{kunal.name} age: {kunal.age} birth_year: {kunal.birth_year}")
    kunal.read()
    kunal.write()
    kunal.give_mcq("tuples")

    # Student Abhijit
    abhijit = Student("Abhijit Sharma", 24, 2001)

    print(f "{abhijit.name} age: {abhijit.age} birth_year: {abhijit.birth_year}")
    abhijit.read()
    abhijit.write()
    abhijit.give_mcq("oops")
```

OUTPUT:

Kunal age: 25 birth_year: 1998

student Kunal is reading!!

student Kunal is writing!!

student Kunal gave mcq on topic tuples

Abhijit Sharma age: 24 birth_year: 2001

student Abhijit Sharma is reading!!

student Abhijit Sharma is writing!!

student Abhijit Sharma gave mcq on topic oops

We can add another method, “get_marks”

in file student.py

```
    def get_marks(self):
        """
        get_marks: returns soms random marks.
         """
        return random.randint(1,100)
```

in main.py

```
print(f"marks got by abhijit {abhijit.get_marks()}")
```

OUPUT:

marks got by abhijit 45

IN **student.py**

```
    def get_marks(self):
        """
        get_marks: returns soms random marks.
         """
        subjects = ["maths", "geography", "sociology"]
        random_subject = subjects[random.randint(0,len(subjects)-1)]
        return random_subject, random.randint(1,100)
```

IN **main.py**

```
subject, marks = abhijit.get_marks()
    print(f "abhijit got {marks} in {subject}")
```

first run → abhijit got 17 in sociology

second run → abhijit got 71 in geography

third run → abhijit got 59 in geography

fourth run → abhijit got 6 in geography