



# Class in Python



# Class in Python

Same data can be package into the class as in the following way.

```
class student:  
    name = ""  
    matricMarks = 0.0  
    fscMarks = 0.0  
    ecatMarks = 0.0  
    merit = 0.0
```



# Object in Python



# Object in Python

A Variable/Object of the class can be created as:

```
class student:  
    name = ""  
    matricMarks = 0.0  
    fscMarks = 0.0  
    ecatMarks = 0.0  
    merit = 0.0
```

```
s = student()
```

# Accessing data members in Python

If you need to update part of any information you can use dot notation to access the sub component of the data.

```
class student:  
    name = ""  
    matricMarks = 0.0  
    fscMarks = 0.0  
    ecatMarks = 0.0  
    merit = 0.0
```

```
s = student()  
s.name = "Ahmad"  
s.matricMarks = 1009  
s.fscMarks = 972  
s.ecatMarks = 280
```



# Behaviour in Python



# Defining Behaviour in Python


If you need to make a function/behaviour in a class, the foremost parameter of the method should always have to be self.

```
class student:
    name = ""
    matricMarks = 0.0
    fscMarks = 0.0
    ecatMarks = 0.0
    merit = 0.0
    def calculateMerit(self):
        merit = ((0.25 * self.matricMarks/1100) + (0.45
            * self.fscMarks/1100) + (0.3 * self.ecatMarks/400))
        return merit
```

# Defining Behaviour in Python

If you need to make a function/behaviour in a class, the foremost parameter of the method should always have to be self.

```
class student:
    name = ""
    matricMarks = 0.0
    fscMarks = 0.0
    ecatMarks = 0.0
    merit = 0.0
    def calculateMerit(self):
        merit = ((0.25 * self.matricMarks/1100) + (0.45
            * self.fscMarks/1100) + (0.3 * self.ecatMarks/400))
        return merit
```





# Calling a Behaviour in Python

The self variable points to an instance of the class student when the method is invoked

```
class student:
    name = ""
    matricMarks = 0.0
    fscMarks = 0.0
    ecatMarks = 0.0
    merit = 0.0
    def calculateMerit(self):
        merit = ((0.25 * self.matricMarks/1100) + (0.45
            * self.fscMarks/1100) + (0.3 * self.ecatMarks/400))
        return merit
```

```
s = student()
s.name = "Ahmad"
s.matricMarks = 1009
s.fscMarks = 972
s.ecatMarks = 280
print(s.calculateMerit())
```



# Constructor in Python



# Constructor in Python

Default Constructor is defined in the following way

```
class student:  
    name = ""  
    matricMarks = 0.0  
    fscMarks = 0.0  
    ecatMarks = 0.0  
    merit = 0.0  
  
    def __init__(self):  
        print("Default Constructor")
```

# Constructor in Python

Parameterized Constructor is defined in the following way

```
class student:
    name = ""
    matricMarks = 0.0
    fscMarks = 0.0
    ecatMarks = 0.0
    merit = 0.0

    def __init__(self, name, matricMarks, fscMarks, ecatMarks):
        self.name = name
        self.matricMarks = matricMarks
        self.fscMarks = fscMarks
        self.ecatMarks = ecatMarks
```

# Constructor in Python

Python does not support explicit multiple constructors. If multiple `__init__` methods are written, then the latest one overwrites all the previous constructors

```
class student:
    name = ""
    matricMarks = 0.0
    fscMarks = 0.0
    ecatMarks = 0.0
    merit = 0.0

    def __init__(self, name, matricMarks, fscMarks, ecatMarks):
        self.name = name
        self.matricMarks = matricMarks
        self.fscMarks = fscMarks
        self.ecatMarks = ecatMarks
```



# SignIn SignUp in Python



# Working Example: Vision

Make a **SignIn** and **SignUp** Application that will check if the user is stored in the file then it will allow it to LogIn. If the user SignUp then the record is stored in the file in comma separated format.



# MUser Class

```
class MUser:
    userName = ""
    userPassword = ""
    userRole = ""

    def __init__(self, userName, userPassword, userRole):
        self.userName = userName
        self.userPassword = userPassword
        self.userRole = userRole

    def isAdmin(self):
        if (self.userRole == "Admin"):
            return True
        else:
            return False
```



# MUserDL

```
import os.path
from MUser import MUser

class MUserDL:
    usersList = []

    @staticmethod
    def addUserIntoList(user):
        MUserDL.usersList.append(user)

    @staticmethod
    def SignIn(user):
        for storedUser in MUserDL.usersList:
            if(storedUser.userName == user.userName and
storedUser.userPassword == user.userPassword):
                return storedUser
        return None

    @staticmethod
    def parseData(line):
        line = line.split(",")
        return line[0], line[1], line[2]
```

```
@staticmethod
def readDataFromFile(path):
    if (os.path.exists(path)):
        fileVariable = open(path, 'r')
        records = fileVariable.read().split("\n")
        fileVariable.close()
        for line in records:
            userName, userPassword, userRole =
MUserDL.parseData(line)
            user = MUser(userName, userPassword,
userRole)

            MUserDL.addUserIntoList(user)
        return True
    else:
        return False

    @staticmethod
    def storeUserIntoFile(user, path):
        file = open(path, 'a')
        file.write("\n" + user.userName + "," +
user.userPassword + "," + user.userRole)
        file.close()
```



# MUserUI

```
from MUser import MUser

class MUserUI:

    @staticmethod
    def menu():
        print("1. SignIn")
        print("2. SignUp")
        print("Enter Option")
        option = int(input())
        return option

    @staticmethod
    def printList(usersList):
        for user in usersList:
            print(user.userName,
user.userPassword, user.userRole)
```

```
@staticmethod
def TakeInputFromConsole():
    userName = input("Enter UserName")
    userPassword = input("Enter UserPassword")
    userRole = input("Enter UserRole");
    user = MUser(userName, userPassword, userRole)
    return user

@staticmethod
def takeInputwithOutRole():
    userName = input("Enter UserName")
    userPassword = input("Enter UserPassword")
    user = MUser(userName, userPassword, None)
    return user
```

# Main

```
#    Calling Main Function as the Code starts executing
if __name__ == "__main__":
    main()
```

```
#    Including Libraries
from MUserDL import MUserDL
from MUserUI import MUserUI
import os
from time import sleep

#    Defining Main Function
def main():
    path = "Data.txt"
    MUserDL.readDataFromFile(path)
    option = 0
    while (option != 3):
        os.system("cls")
        option = MUserUI.menu()
        if (option == 1):
            user = MUserUI.takeInputwithOutRole()
            user = MUserDL.SignIn(user)
            if (user != None):
                if (user.isAdmin()):
                    print("This is Admin")
                    #Admin Menu
                else:
                    print("This is User")
                    #User Menu
                    sleep(3)
            elif (option == 2):
                user = MUserUI.TakeInputFromConsole()
                MUserDL.addUserIntoList(user)
                MUserDL.storeUserIntoFile(user, path)
```