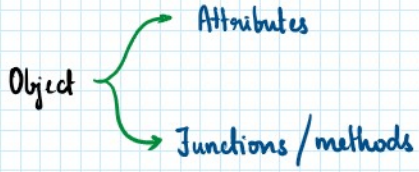
3. Object Oriented Programming

01 December 2023 12:19



CLASS

- Methodology to create entities / objects
- No memory allocation for classes
- Function with the same name as class → constructor function → __init__(self) called
- self.__class__.__name__ → DOUBT

```
class Student:
    def __init__(self):
        #instance variables/attributes/properties (always inside init)
        self.name="abc"
        self.age=18
        self.srn="PES12023[redacted]"

    def disp(self):
        print(self.name,self.age,self.srn,sep="\n")

s1=Student()
s1.name="Advaith"
Student.disp(s1)
```

→ can also be called as s1.disp()

```
Advaith
18
PES12023[redacted]
```

```
print(type(s1))
```

```
<class '__main__.Student'>
```

```
class Student:
    def __init__(self, name, age, srn):
        #instance variables/attributes/properties (always inside init)
        #instance variables are always of the form self.[attribute]
        self.name=name
        self.age=age
        self.srn=srn
        name1="Student" #NOT an instance variable; s1.name1 returns an error

    def disp(self):
        print(self.name,self.age,self.srn,sep="\n")

s1=Student("advaith",18,"PES12023...")
s1.name="Advaith"
Student.disp(s1)
print(type(s1),end="\n\n")
```

→ s1.disp()

```
Advaith
18
PES12023...
<class '__main__.Student'>
```

- __del__ → destructor → called automatically for all user defined classes at the end of program execution

↓

Can also be called in between using the del keyword

```python
class Student:
    def __init__(self, name, age, srn):
        #instance variables/attributes/properties (always inside init)
        #instance variables are always of the form self.[attribute]
        self.name=name
        self.age=age
        self.srn=srn
        name1="Student" #NOT an instance variable; s1.name1 returns an error

    def disp(self):
        print(self.name,self.age,self.srn,sep="\n")

    def __del__(anyvariable):
        print(anyvariable)
        print("in destructor")

s1=Student("advaith",18,"PES12023...")
s1.name="Advaith"
s1.disp()
s2=Student("Advaith2",18,"PES12023...")
s2.disp()
del s2
print("Program ends here")
```

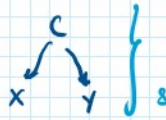
```
E:\PES\Sem 1\Python\OOP>python OOP.py
Advaith
18
PES12023...
Advaith2
18
PES12023...
<__main__.Student object at 0x0000021C246566C0>
in destructor
Program ends here
<__main__.Student object at 0x0000021C24656690>
in destructor
```

NOTE: wasn't getting printed in IDLE

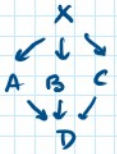
FEATURES OF OOP

1. All entities → viewed as an object
2. Encapsulation: Certain methods and attributes are put together in every class
3. Abstraction: The way you understand an object's nature
4. Data hiding: Private-like variables
5. Polymorphism: Overriding
6. Inheritance: B(A) → B is a child of A; C(B) → Multilevel inheritance

Multiple inheritance: multiple parents, single child



Hierarchical inheritance: single parent



Hybrid/diamond inheritance: mixture of other types

→ "is a" relationship

composition

→ "has a" relationship.

Object of one class is used as an instance variable of another class

STATIC METHODS

Methods / functions that are only accessible by class name, not by class objects.

"self" is never passed into the definition

```python
class Student:
    count = 0 #class variable
    def __init__(self, name, age, srn):
        #instance variables/attributes/properties (always inside init)
        #instance variables are always of the form self.[attribute]
        self.name=name
        self.age=age
        self.srn=srn
```

```python
class Student:
    count = 0 #class variable
    def __init__(self, name, age, srn):
        #instance variables/attributes/properties (always inside init)
        #instance variables are always of the form self.[attribute]
        self.name=name
        self.age=age
        self.srn=srn
        name1="Student" #NOT an instance variable; s1.name1 returns an error
        Student.count+=1

    def disp(self):
        print(self.name,self.age,self.srn,sep="\n")

    def __del__(anyvariable):
        print(anyvariable)
        print("in destructor")

    def disp_count(): #static method; accessed using class, not class objects
        print(Student.count)

s1=Student("advaith",18,"PES12023...")
s1.name="Advaith"
s1.disp()
s2=Student("Advaith2",18,"PES12023...")
s2.disp()
Student.disp_count()
s1.disp_count()
del s2
print("Program ends here")
```

```
Advaith
18
PES12023...
Advaith2
18
PES12023...
2
Traceback (most recent call last):
  File "E:\PES\Sem 1\Python\OOP\OOP.py", line 28, in <module>
    s1.disp_count()
TypeError: Student.disp_count() takes 0 positional arguments but 1 was given
```

```python
    @staticmethod #decorator that avoids error when someone tries to call static method using class object
    def disp_count(): #static method; accessed using class, not class objects
        print(Student.count)

s1=Student("advaith",18,"PES12023...")
s1.name="Advaith"
s1.disp()
s2=Student("Advaith2",18,"PES12023...")
s2.disp()
#Student.disp_count()
s1.disp_count()
del s2
print("Program ends here")
```

```
Advaith
18
PES12023...
Advaith2
18
PES12023...
2
<__main__.Student object at 0x000002DF4D111250>
in destructor
Program ends here
```