What is class?

→ Blueprint for creating objects (instances)

It allows you to bundle

- Data (Var/ attributes)
- Behavior (fun/ methods)

} Class.

Syntax:

class ClassName:
    pass

Keyword → Name of class → Rec to use Camelcase

Ex:

class Car:
    pass.

3. __init__ Method (Constructor)

__init__ runs automatically when an object is created.

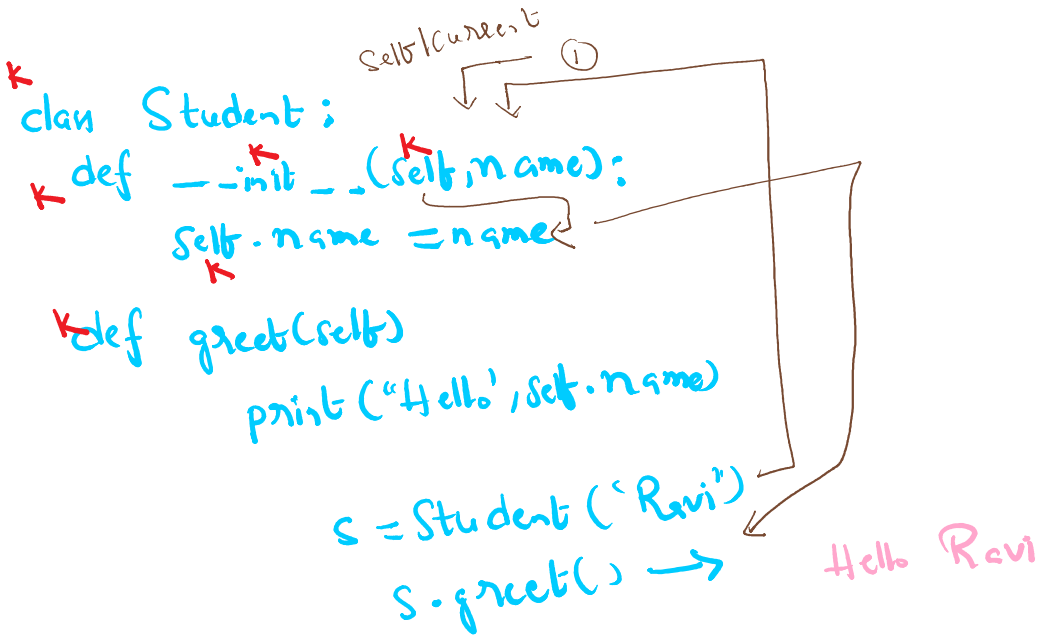
```
class Car:
    def __init__(self, brand, year):
        self.brand = brand
        self.year = year

c1 = Car("Toyota", 2020)
print(c1.brand)
```

4. Understand self

→ self refers current object

→ It is required for instance methods.

→ It is automatically passed.



5. Instance Var (or) vs Class Variable

↳ Belong to each object

↳ Shared by all objects.

```
class Dog:
    def __init__(self, name):
        self.name = name

d = Dog("Hello")
d.name()
```

```
class Dog:
    species = "Golden"
    def __init(self,name)
        self.name = name

print(Dog.species)
```

6. Types of Methods

① Instance methods

② Class Methods — Use decorator called

③ static methods

① Instance Methods

→ Works with object data

Key points:

→ First parameter self.

→ Can access and modify instance var

→ Called using an object

② Class methods

Use decorator called @ classmethod.

③ Static methods

Use decorator @staticmethod.

```
class Student:
    def __init__(self, name, marks):
        self.name = name
        self.marks = marks

    def display(self):
        print("Name:", self.name)
        print("Marks:", self.marks)
```

⇒ s1.display()
↳ Student.display(s1)
self refers to s1

```
s1 = Student("Vijay", 98)
s1.display()
```

self → s1

Name: Vijay
Marks: 98

Class Methods:

Class methods work with class variables not instance variables.

Key points:

→ Use decorator @classmethod.

→ First parameter is cls

→ Can modify class-level data.

→ Called using class name or objects.

```
class Employee:
    Company = "ABC Limited"

    def __init__(self, name):
        self.name = name
```

⇒ `Employee.change_Company("xyz Limited")`

Now all employees share updated Company name.

```
@classmethod
def change_Company(cls, new_name):
    cls.Company = new_name
```

When to use?

→ class vari ✓

→ Alternative constructors

→ factory methods — ✓

3. Static Methods

Static methods don't access instance or class data.

→ Key points

→ Use decorator @staticmethod

→ No self or cls.

→ Behave like normal fun inside class namespace.

When to use?

→ Utility classes

→ Logic that doesn't depend on Object/class.