```
class MyClass:
    x =10

new_object = MyClass
new_object_assigned = MyClass()
print(type(new_object))
print(type(new_object_assigned))
```

does **not** create an object/instance. It assigns the class itself to `new_object`.
Class Reference vs Object/ instance

- Blueprint for creating objects. It groups data and functions

Important Concepts

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

    def introduce(self):
        return f"Hi, I am {self.name} and I am {self.age} years old."
first_person = PersonClass("Anish",25)
print(first_person.name)
print(first_person.age)
print(first_person.introduce())
```

- self refers to the current object. Can be written anything
- `__init__` is a constructor which runs itself.

Calling function with self

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

  def greet_student(self):
    return "Hello, " + self.name

  def welcome_student(self):
    message = self.greet_student()
```

```
    print(message + "! Welcome to Swastik College.")

first_person = BachelorsOfComputerApplication("Shyam")
first_person.welcome_student()
```

Class Property and Instance Property

```
class Dog:
    species = "Canine"   # class attribute

    def __init__(self, name):
        self.name = name  # instance attribute

dog1 = Dog("Tommy")
dog2 = Dog("Bruno")

print(dog1.name)      # Tommy
print(dog2.name)      # Bruno
print(dog1.species)   # Canine
print(dog2.species)   # Canine
```

can be called by `self.species = "Canine"` as well

Inheritance

```
class Animal:
    def __init__(self):
        print("This animal eats food.")

class Dog(Animal):
    def __init__(self):
        print("Dog barks.")

dog = Dog()
```

Calling constructor of the parent function:

```
class Dog(Animal):
    def __init__(self):
        Animal.__init__(self)
        print("Dog barks.")
```

can be replaced by
`super().__init__()`

String printing in class

Done by `__str__`

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

    def __str__(self):
        return f"Person's name is {self.name}"

p = PersonClass("Anish")
print(p)
```

Encapsulation

```
class BankAccount:
    def __init__(self, balance):
        self.__balance = balance

    def show_balance(self):
        print(self.__balance)

account = BankAccount(1000)
account.show_balance()
account.__balance
```

```
> Try with protected attribute
```

```
class BankAccount:
    def __init__(self, account_holder, balance):
        self.account_holder = account_holder
        self.__balance = balance   # private variable

    def deposit(self, amount):
        self.__balance += amount
        print("Amount deposited:", amount)

    def withdraw(self, amount):
        if amount <= self.__balance:
```

```
            self.__balance -= amount
            print("Amount withdrawn:", amount)
        else:
            print("Insufficient balance")

    def show_balance(self):
        print("Current Balance:", self.__balance)

    def __str__(self):
        return f"{self.account_holder} is the account holder"

first_account = BankAccount("Anish", 5000)
print(first_account)
first_account.deposit(2000)
first_account.withdraw(1000)
first_account.show_balance()
```

#WAP to inherit Person from student

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

    def display_person_info(self):
        print("Name:", self.name)
        print("Address:", self.address)


class Student(Person):
    def __init__(self, name, address, roll_no, semester):
        super().__init__(name, address)
        self.roll_no = roll_no
        self.semester = semester

    def display_student_info(self):
        self.display_person_info()
        print("Roll Number:", self.roll_no)
        print("Semester:", self.semester)


student = Student("Nischal", "Bhaktapur", 21, "BCA 5th Semester")
student.display_student_info()
```

Polymorphism

```python
class Esewa:
    def pay(self, amount):
        print(f"Paid Rs. {amount} using Esewa")


class Khalti:
    def pay(self, amount):
        print(f"Paid Rs. {amount} using Khalti")


class Bank:
    def pay(self, amount):
        print(f"Paid Rs. {amount} using Bank Transfer")


def make_payment(payment_method, amount):
    payment_method.pay(amount)


esewa = Esewa()
khalti = Khalti()
bank = Bank()


make_payment(esewa, 1000)
make_payment(khalti, 1500)
make_payment(bank, 2000)
```

#WAP to calculate total % and grade

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

    def calculate_total(self):
        return sum(self.marks)

    def calculate_percentage(self):
        return self.calculate_total() / len(self.marks)

    def calculate_grade(self):
        percentage = self.calculate_percentage()
```

```
        if percentage >= 80:
            return "A"
        elif percentage >= 60:
            return "B"
        elif percentage >= 40:
            return "C"
        else:
            return "Fail"

    def display_result(self):
        print("Name:", self.name)
        print("Roll Number:", self.roll_no)
        print("Total Marks:", self.calculate_total())
        print("Percentage:", self.calculate_percentage())
        print("Grade:", self.calculate_grade())


marks = [78, 85, 69, 90, 74]

student = Student("Swastik", 10, marks)
student.display_result()
```