🚀 21-Day Python Challenge

Day 13 – Exception Handling

🎯 Learning Objectives

By the end of today's class, students will be able to:

- Understand what an **Exception** is.
- Learn why programs crash.
- Use **try**.
- Use **except**.
- Use **finally**.
- Build programs that handle errors without crashing.

📓 What is an Exception?

An **Exception** is an error that happens while the program is running.

Normally,

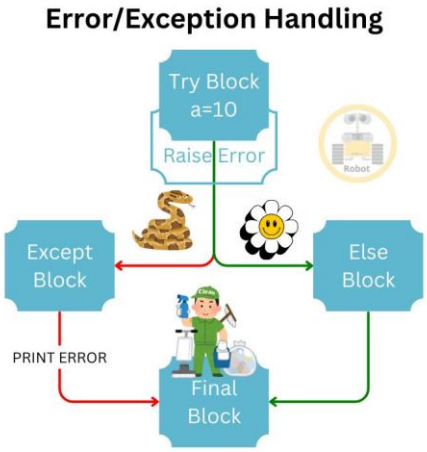
```
No Error
↓
Program Runs
↓
Output
```

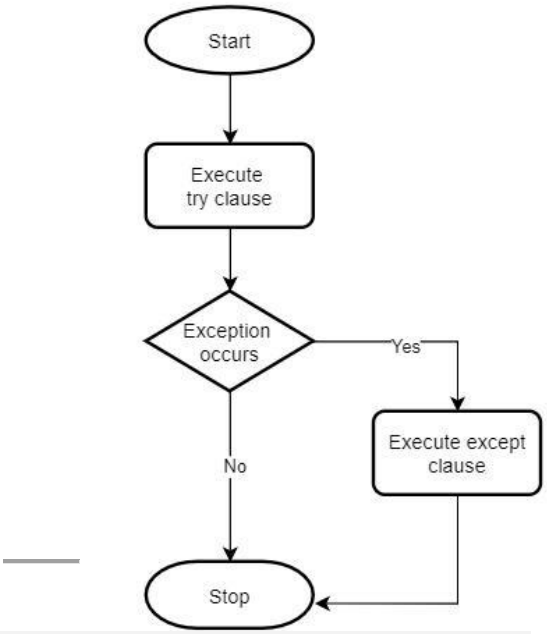
But sometimes,

```
Program
↓
Error Occurs
↓
Program Stops ✕
```

Python gives an error message and the program stops.

📷 Picture 1 – Program with Error





What is Exception

An Exception is an unwanted or unexpected event that occurs during the execution of a program (i.e., at runtime) and disrupts the normal flow of the program's instructions. It occurs when something unexpected things happen, like accessing an invalid index, dividing by zero, or trying to open a file that does not exist.

```
START
    SET loan_amount = 10000
    SET months = 0  // Invalid input
    COMPUTE installment = loan_amount / months
    PRINT installment
END
```

Without Exception Handling

If months is 0, the program will throw an error (e.g., Division by Zero error) and crash.

Real-Life Example

Imagine riding a bicycle.

```
Road is Clear
↓
Ride Continues
↓
Destination
```

But,

```
Road
↓
Stone
↓
Handle Carefully
↓
Continue Riding
```

Exception Handling is like carefully crossing the stone instead of falling.

Why Exception Handling?

Without Exception Handling

```
Program
↓
Error
↓
Stops ✕
```

With Exception Handling

```
Program
↓
```

```
Error

↓

Handled

↓

Program Continues ✅
```

📄 try Block

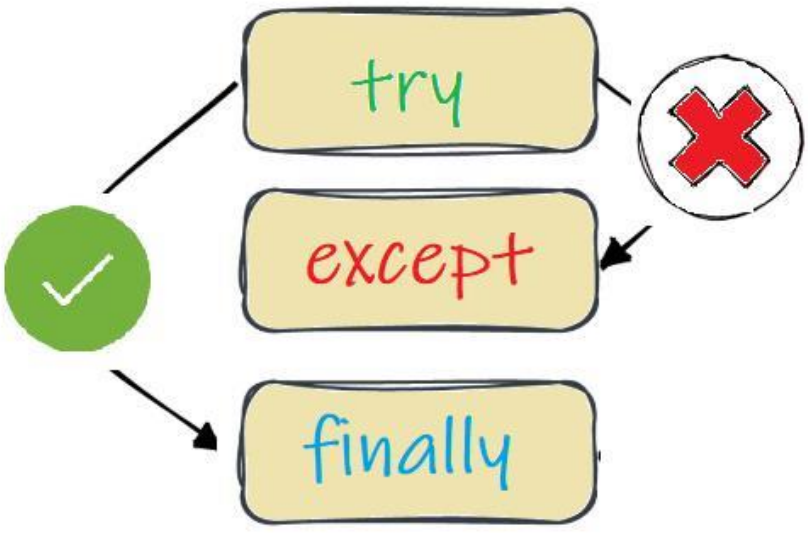
The **try** block contains code that may produce an error.

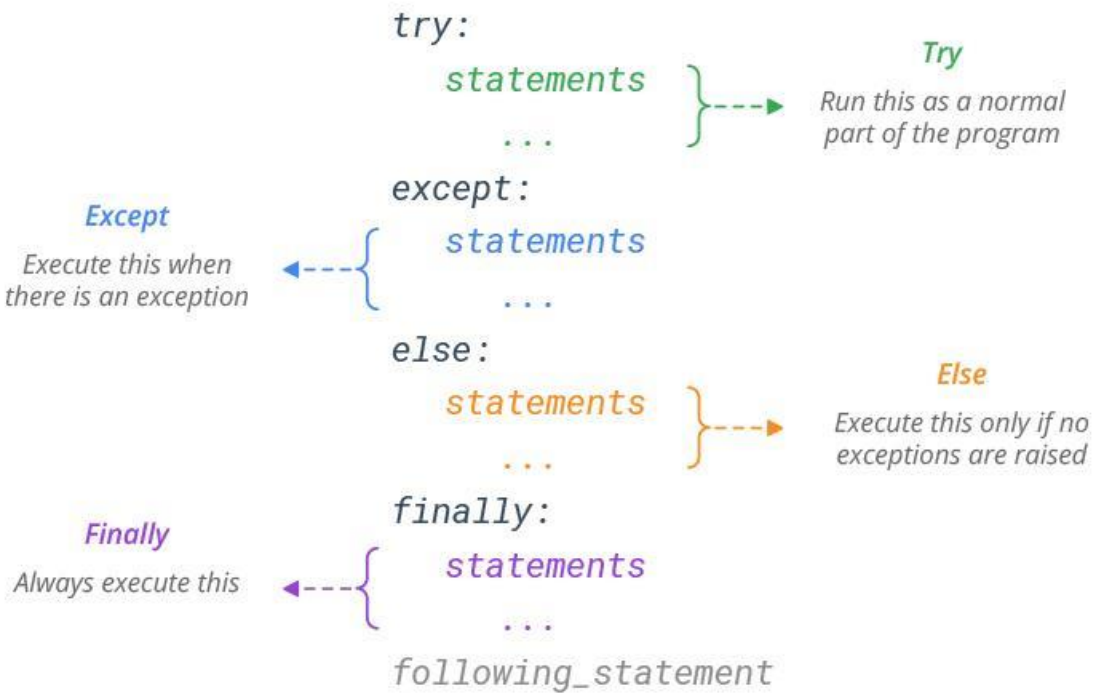
```
try:
    print(10/0)
```

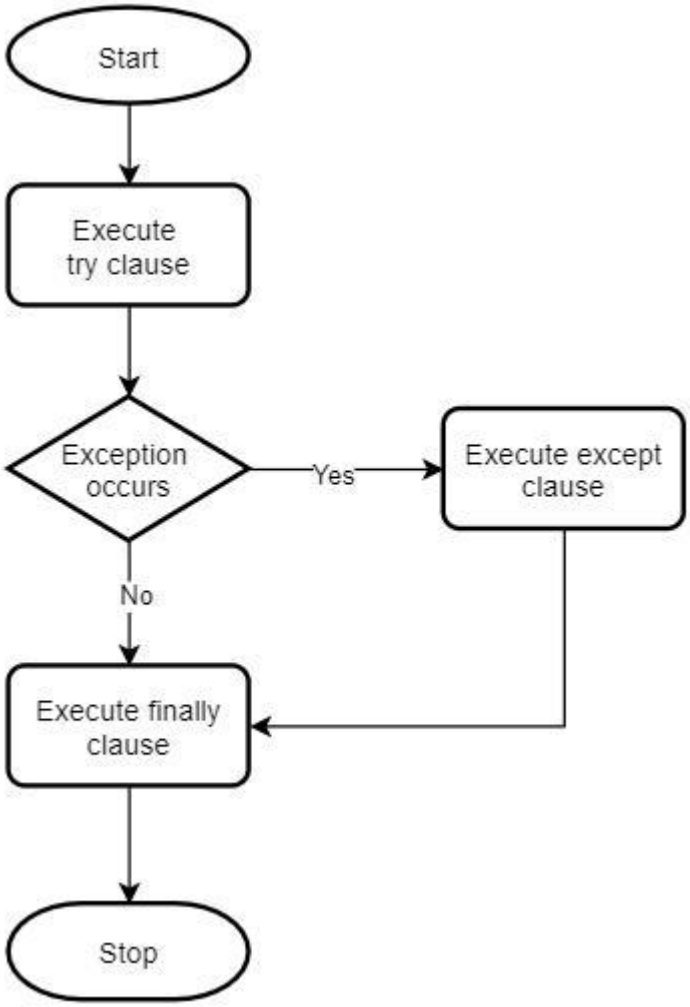
Python checks this code.

If there is no error, it continues normally.

📷 Picture 2 – try Block




try:
statements
...
Try
Run this as a normal part of the program
except:
Except
Execute this when there is an exception
statements
...
else:
Else
Execute this only if no exceptions are raised
statements
...
finally:
Finally
Always execute this
statements
...
following_statement


Start
Execute try clause
Exception occurs
Yes
Execute except clause
No
Execute finally clause
Stop

Example 1

```
try:
    print(10/2)

except:
    print("Error")
```

Output

```
5.0
```

No error occurred.

📄 except Block

If an error happens, Python runs the **except** block.

Example

```
try:
    print(10/0)

except:
    print("Cannot Divide by Zero")
```

Output

```
Cannot Divide by Zero
```

Instead of crashing, Python displays our message.

Step-by-Step Flow

```
try

↓

10 / 0

↓

Error?

↓

Yes

↓

except Runs

↓

Program Continues
```

Example 2

```
try:
    number = int(input("Enter Number : "))
    print(100 / number)
except:
    print("Invalid Input")
```

Possible Output

```
Enter Number : 0
Invalid Input
```

📄 finally Block

The **finally** block always runs.

Whether there is an error or not.

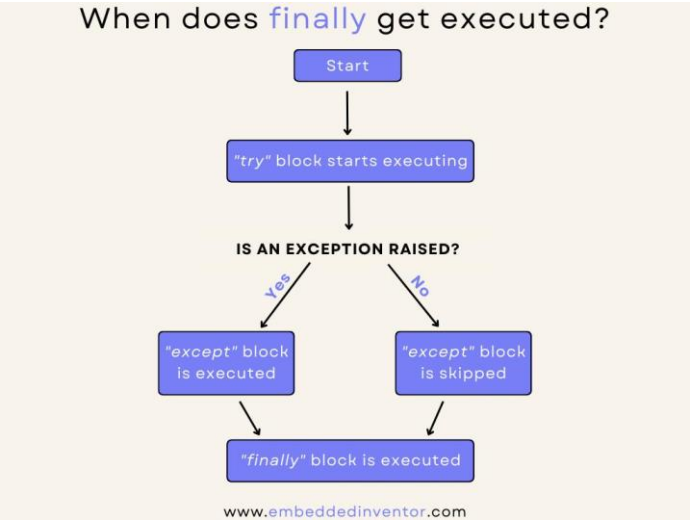
Example

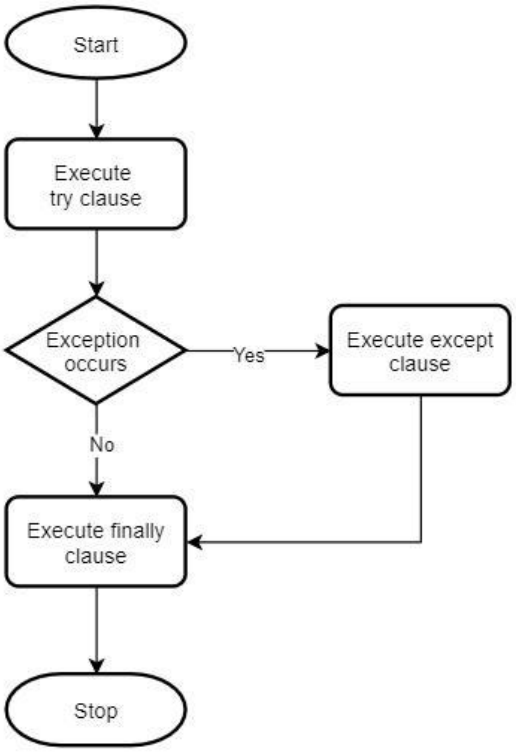
```
try:
    print(10/2)
except:
    print("Error")
finally:
    print("Program Finished")
```

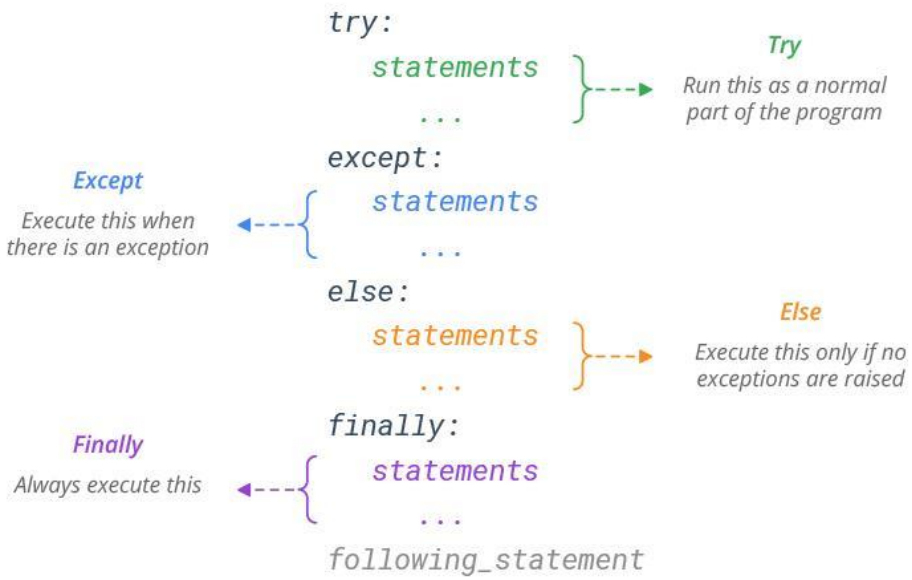
Output

```
5.0
Program Finished
```

📷 Picture 3 – finally Block







Real-Time Example

ATM Machine

```
Insert ATM Card

↓

Enter PIN

↓

Correct?

↓

Yes → Withdraw Money

↓

Remove Card

----------------

Wrong PIN

↓

Show Error

↓

Remove Card
```

Notice: **Card is always returned.**

That is similar to **finally**.

Practical Program 1

```
try:

    a = int(input("First Number : "))
    b = int(input("Second Number : "))

    print("Answer =", a/b)

except:

    print("Cannot Divide by Zero")

finally:

    print("Calculator Closed")
```

Practical Program 2

```
try:

    age = int(input("Enter Age : "))

    print("Age =", age)

except:

    print("Please Enter Numbers Only")

finally:

    print("Thank You")
```

Mini Project

Safe Calculator

```
print("===== SAFE CALCULATOR =====")

try:

    num1 = int(input("First Number : "))
    num2 = int(input("Second Number : "))

    print("Addition =", num1 + num2)
    print("Division =", num1 / num2)

except:

    print("Invalid Input or Cannot Divide by Zero")

finally:

    print("Program Completed Successfully")
```

Interview Questions

1. What is an Exception?
2. Why do programs crash?
3. What is try?
4. What is except?
5. What is finally?
6. Does finally always execute?

🌟 Day 13 Success Message

Today you learned:

- ✅ try
- ✅ except
- ✅ finally

"Good programmers don't just write programs—they prepare their programs to handle unexpected situations."

🚀 21-Day Python Challenge

Day 14 – Object-Oriented Programming (OOP)

🎯 Learning Objectives

You will learn

- What is OOP?
- What is a Class?
- What is an Object?
- What is a Constructor?
- Create simple classes.

📄 What is OOP?

Object-Oriented Programming (OOP) is a way of organizing a program using **Classes** and **Objects**.

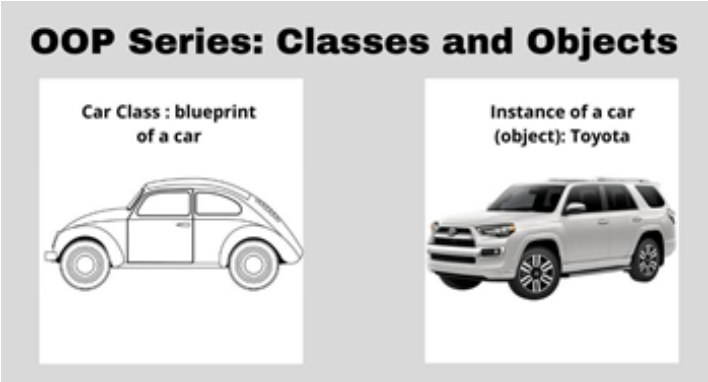
Real-life objects include:

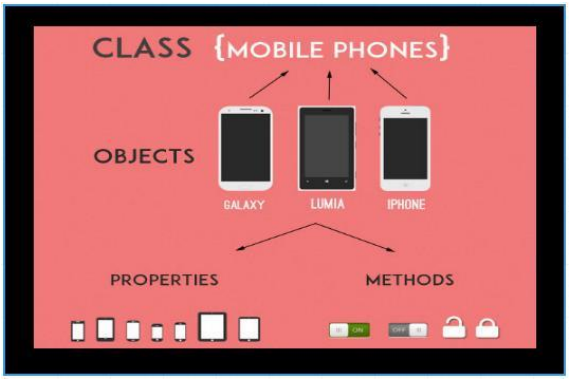
- Student
- Mobile Phone
- Car
- Laptop
- Fan
- Television

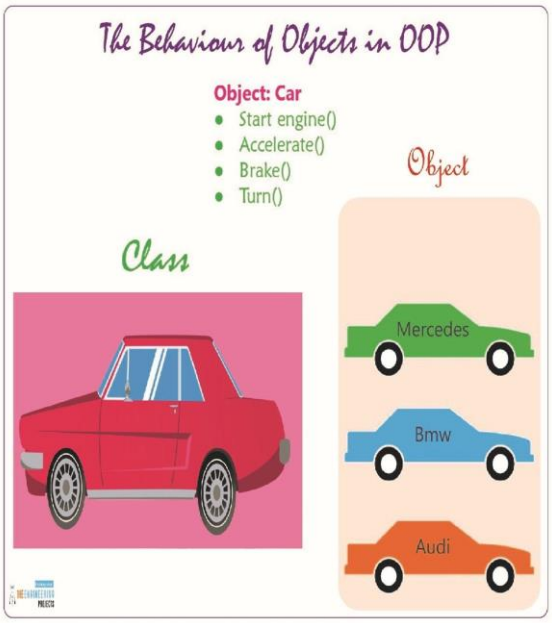
Each object has:

- Data (Properties)
- Actions (Functions)

📷 Picture 4 – Real World Objects







What is a Class?

A **Class** is a blueprint or template.

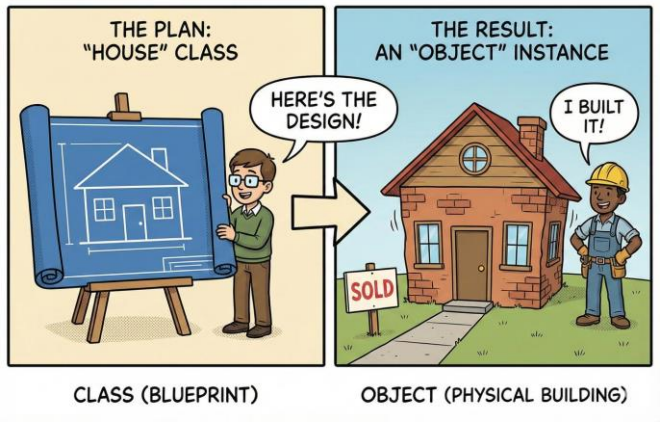
Example

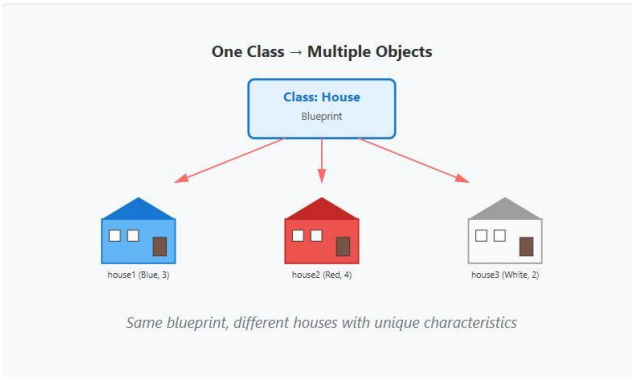
```
House Plan

↓

Many Houses
```

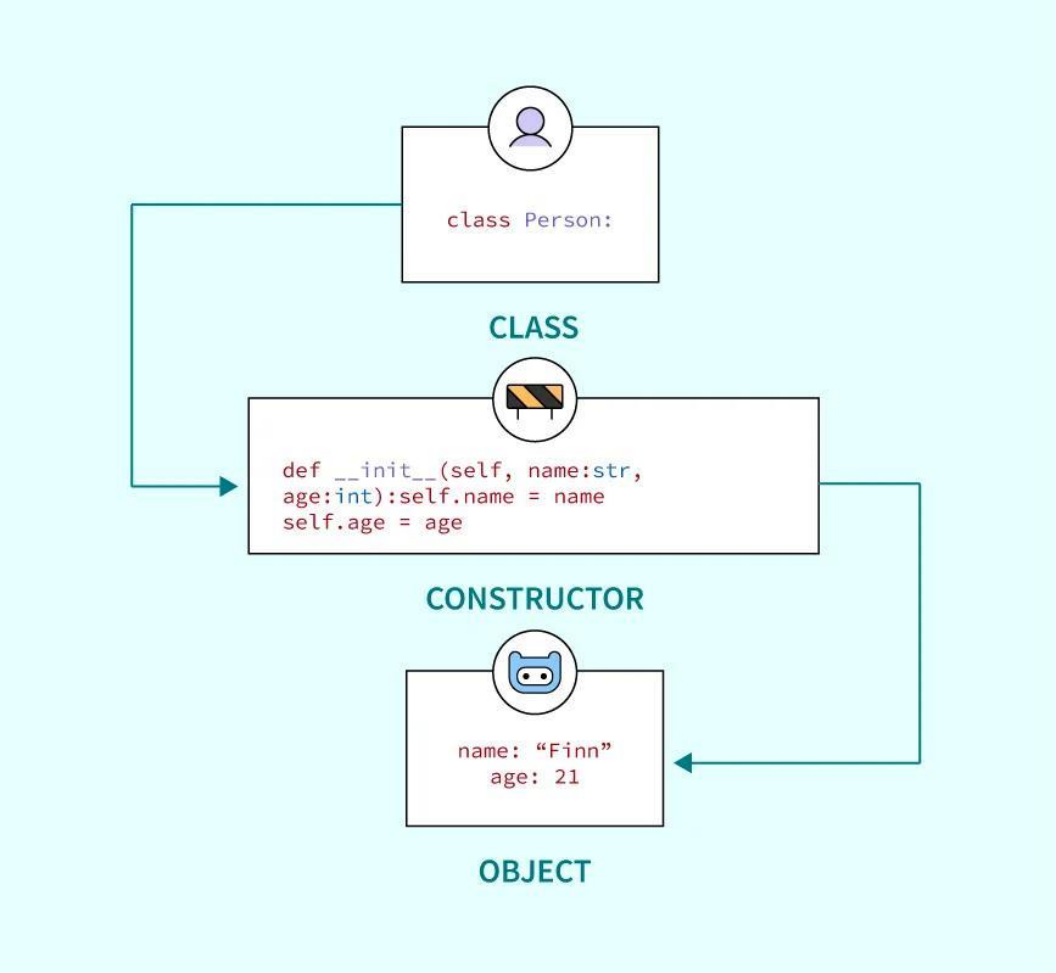
The plan is the **Class**.

Each house built is an **Object**.

📷 Picture 5 – Class and Objects







Python Class

```
class Student:

    pass
```

Here,

```
Student

↓

Class
```

No object has been created yet.

What is an Object?

An Object is a real instance created from a class.

```
class Student:

    pass

student1 = Student()

print(student1)
```

Picture Explanation

```
Student Class

↓

Student1 Object

↓

Student2 Object

↓

Student3 Object
```

One class can create many objects.

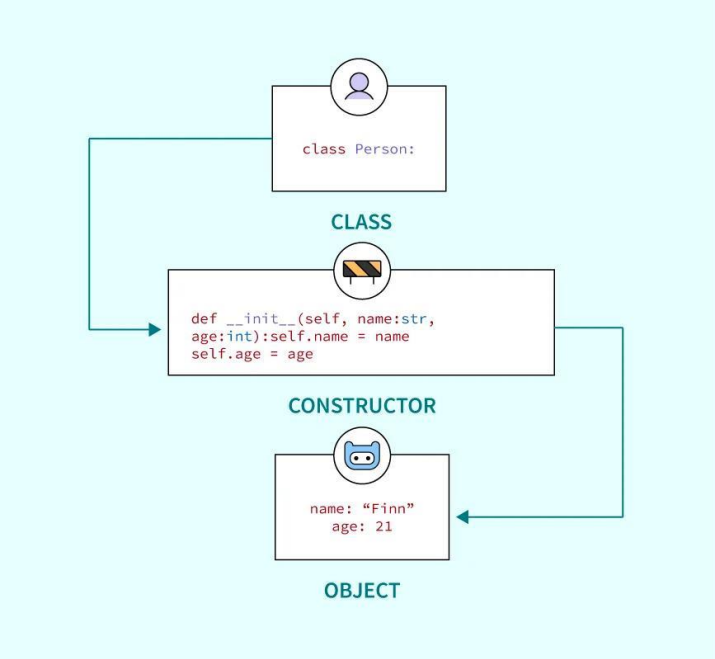
Constructor

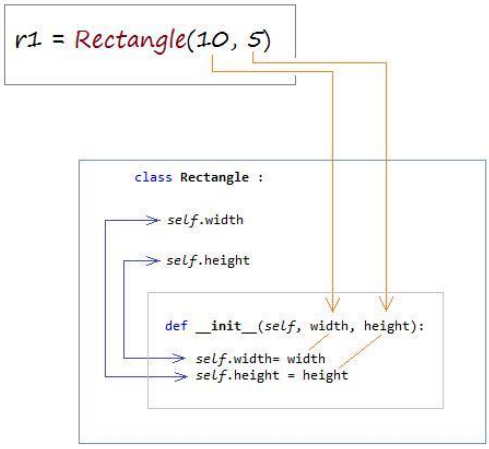
A Constructor automatically runs when an object is created.

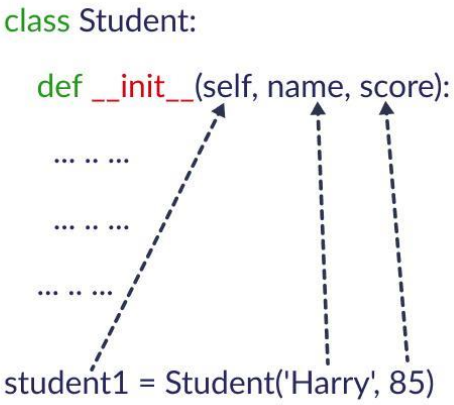
Python Constructor is

```
__init__()
```

Picture 6 – Constructor







Example 1

```
class Student:

    def __init__(self):

        print("Student Object Created")

student1 = Student()
```

Output

```
Student Object Created
```

Understanding `self`

Think of **self** as "**this object**".

When we create

```
student1 = Student()
```

Python internally sends

```
student1

↓

self
```

so the constructor knows which object it is working with.

Example 2

```
class Student:

    def __init__(self, name):

        self.name = name

student1 = Student("Rahul")

print(student1.name)
```

Output

```
Rahul
```

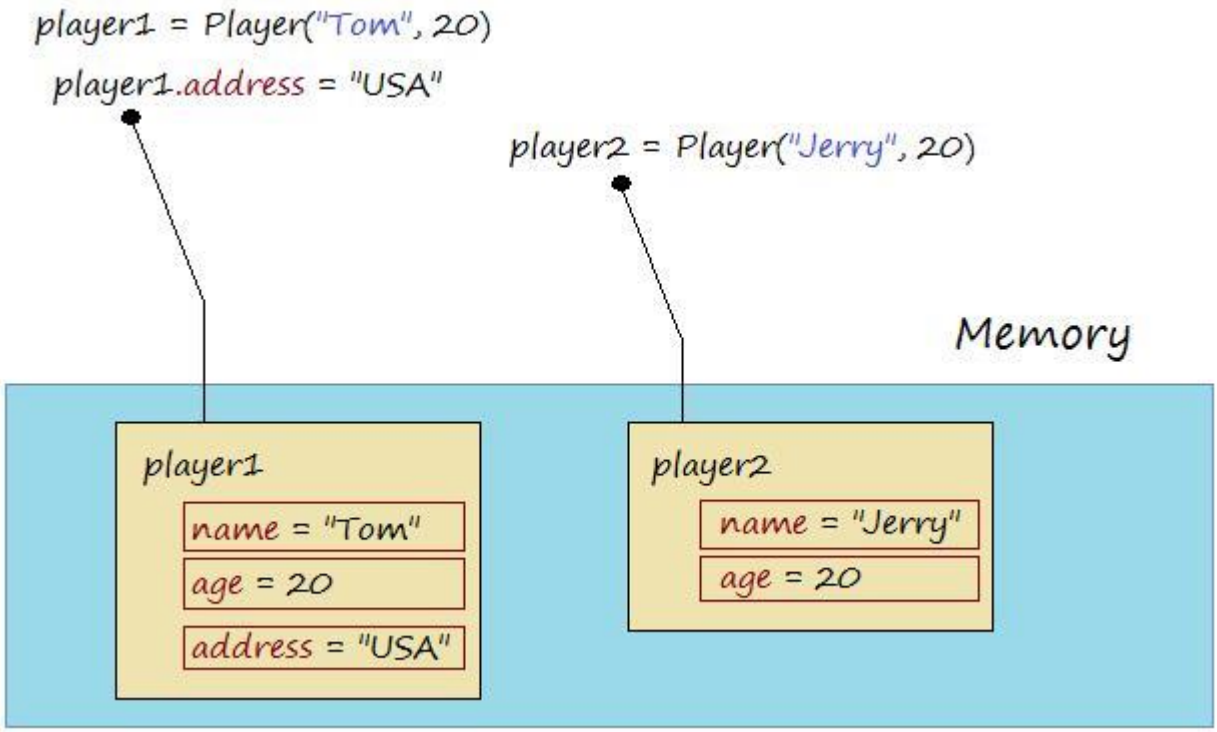
Example 3

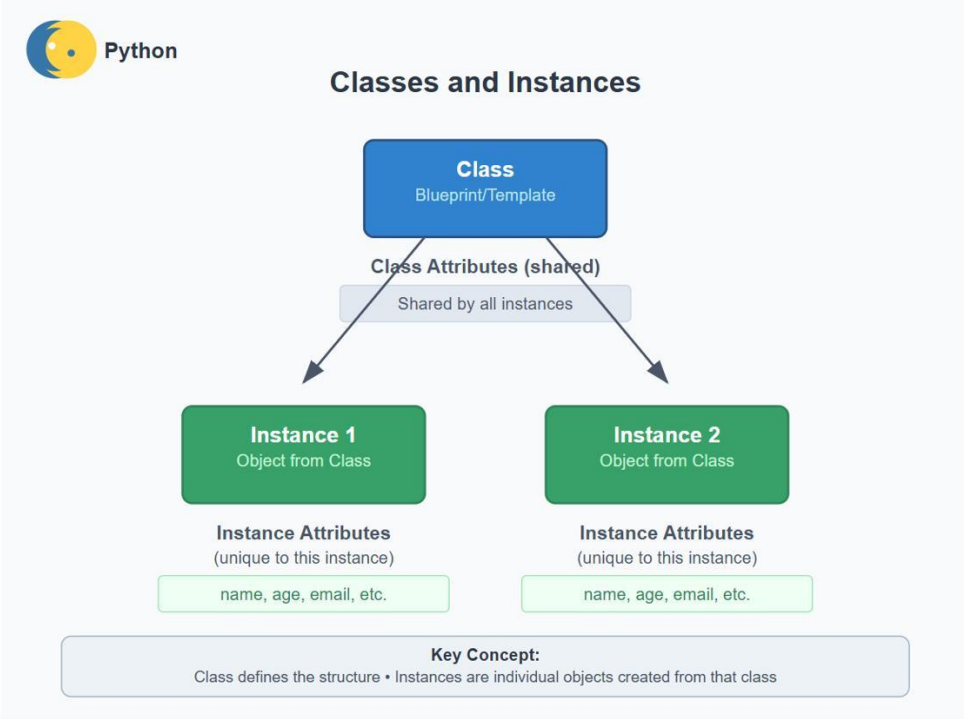
```
class Student:

    def __init__(self, name, marks):

        self.name = name
        self.marks = marks

    def display(self):

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

student1 = Student("Rahul",92)

student1.display()
```

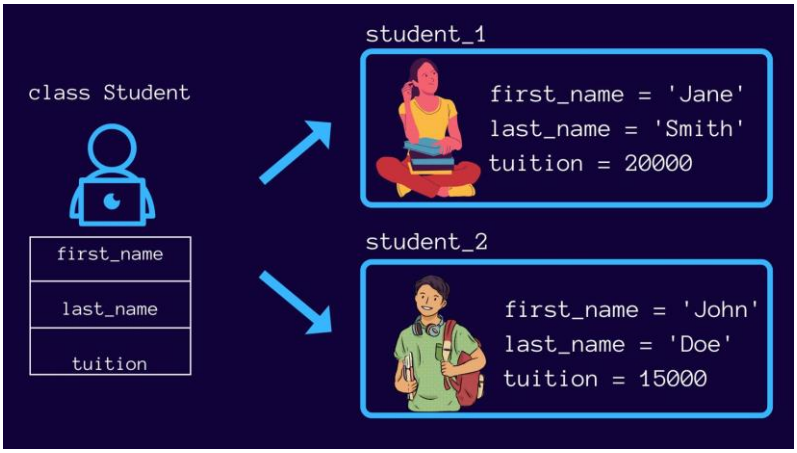
Output

```
Name : Rahul

Marks: 92
```

📷 Picture 7 – Object in Memory








```
Student

↓

Object

↓

Name = Rahul

Marks = 92

↓

Display()
```

Real-Time Program

Student Information

```
class Student:

    def __init__(self, name, age):

        self.name = name
        self.age = age

    def show(self):

        print("Student Name :", self.name)
        print("Age :", self.age)

name = input("Enter Name : ")
age = int(input("Enter Age : "))

student = Student(name, age)

student.show()
```

Mini Project

Employee Information

```
class Employee:

    def __init__(self, name, salary):

        self.name = name
        self.salary = salary

    def display(self):

        print("Employee Name :", self.name)
        print("Salary :", self.salary)

name = input("Enter Employee Name : ")
salary = int(input("Enter Salary : "))

employee = Employee(name, salary)

employee.display()
```

Class vs Object

Class	Object
Blueprint	Real Object
Template	Instance
Defines Data	Uses Data

Assignment

Write programs to:

1. Create a Student class.
2. Create an Employee class.
3. Use a constructor.
4. Store name and age.
5. Display student details.

Interview Questions

1. What is OOP?
2. What is a Class?
3. What is an Object?
4. What is a Constructor?
5. What is `__init__()`?
6. What is `self`?

🌟 Day 14 Success Message

Congratulations! 🎉

Today you learned one of the most powerful concepts in Python—**Object-Oriented Programming (OOP)**.

You explored:

- ✅ Classes
- ✅ Objects
- ✅ Constructors
- ✅ `self`
- ✅ Creating real-world programs using OOP

"A class is like a blueprint, and an object is the real thing built from that blueprint. Mastering OOP helps you build larger, cleaner, and more professional software applications."

🚀 **Next Lesson (Day 15): Inheritance, Polymorphism, and Encapsulation – the core principles that make OOP even more powerful.**