LEARN RISE SHINE
NAVTTC

Summer of Code

Artificial Intelligence

(Machine Learning & Deep Learning)

Instructor
Wajahat Ullah
- *Research Assistant* (DIP Lab)

Duration
03 Months
(September – November)

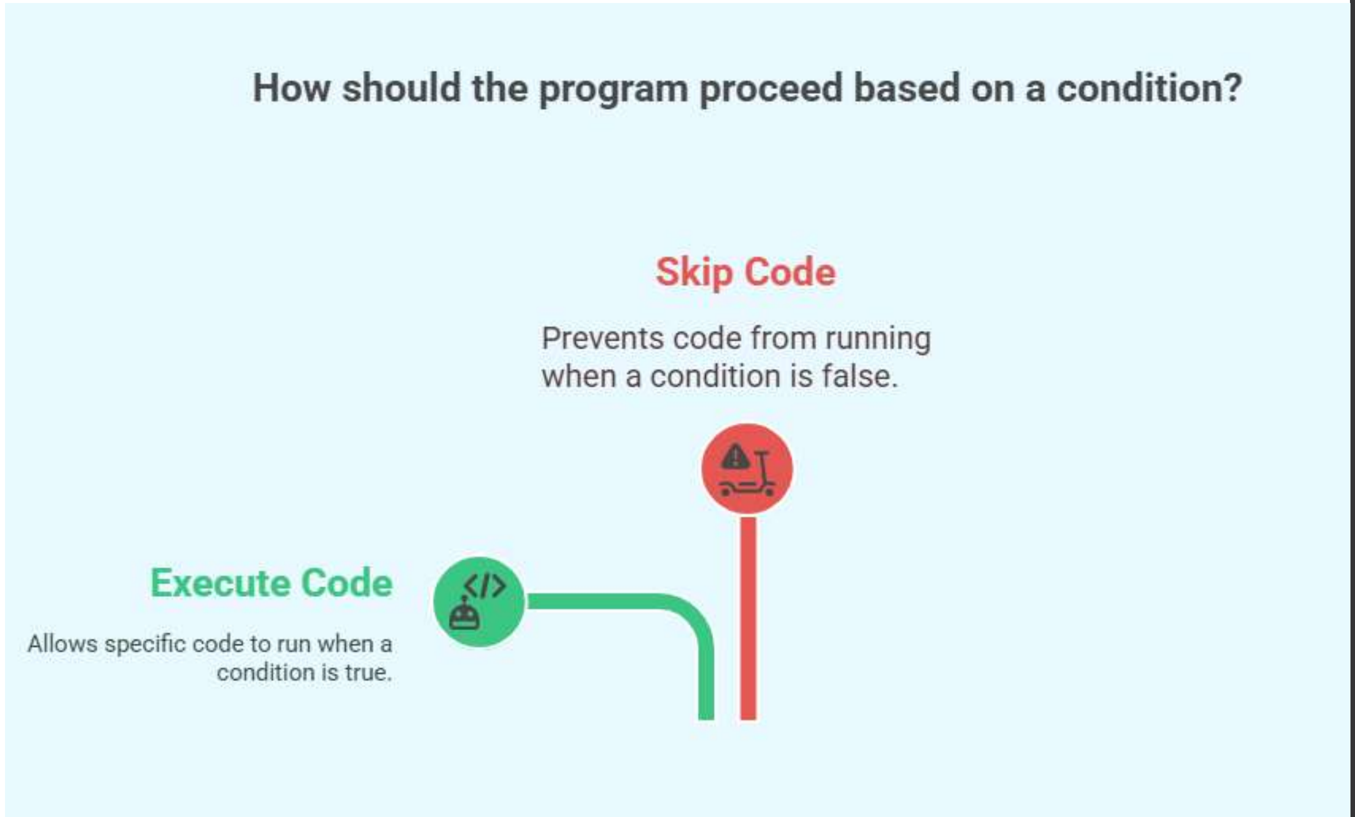
Day 05 – Python Fundamentals (Conditionals and Loops)

Objectives:

- What are Conditional Statements?
- What are Loops?
- Python Indentation
- What are Iterables?

Conditional Statements

- **What Are Conditional Statements?**
  - Conditional statements allow a program to make decisions based on whether a condition is **True** or **False**.
- **Why Use Conditions?**
  - Control the flow of execution.
  - Executes code only when specific conditions are met.
  - Skip or branch logic based on outcomes.
- **Real-Life Example**: Traffic light
  - If the light is green → cars go.
  - If the light is red → cars stop.


How should the program proceed based on a condition?
Skip Code
Prevents code from running when a condition is false.
Execute Code
Allows specific code to run when a condition is true.

Conditions in Python

- A condition is an expression that evaluates to True or False.

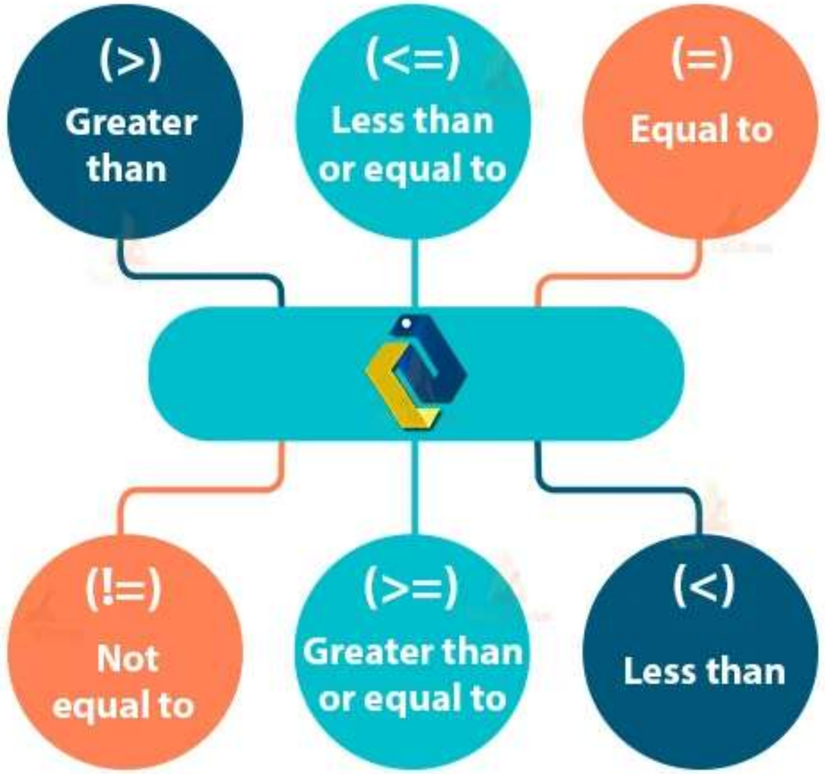
Common Comparison Operators:

Operator	Meaning
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Example:

```
x = 10
if x > 5:
    print(“x is greater than 5”)
```

- In this example, ***x > 5*** is the condition being evaluated.



Logical Operators in Python

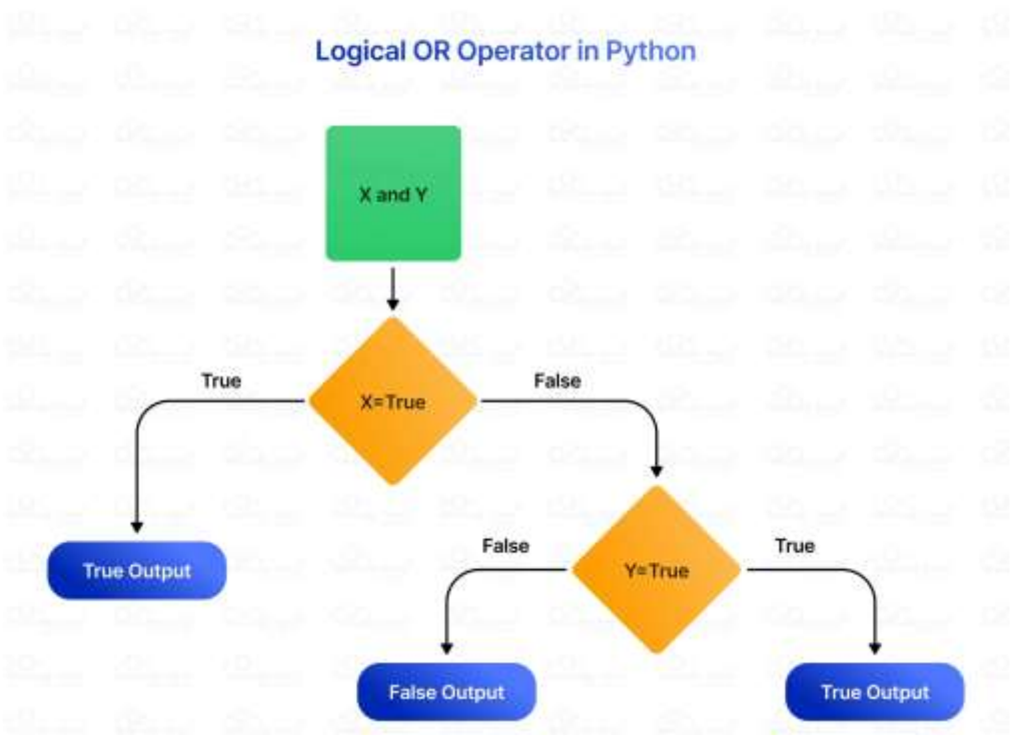
Logical operators are used to combine multiple conditions and return a True or False result.

Types of Logical Operators:

Operator	Description
and	True if both conditions are True
or	True if at least one condition is True
not	Reverses the result (True ↔ False)

Examples:

```
# and operator
age = 20
if age > 18 and age < 30:
   print("You're a young adult")

# or operator
grade = 'B'
if grade == 'A' or grade == 'B'
      print("You passed")
```



if Statements

Purpose:

- if statements allow you to execute code only when a specific condition is True.

Syntax:

```
if condition:
    # code block to execute if condition is True
```

- The condition follows the if keyword and ends with a colon :.
- The code block is indented and runs only if the condition is True.
- If the condition is False, the block is skipped.

Example:

```
num = 10
if num > 5:
    print("The number is greater than 5")
```

- In this example, the message is printed only if *num > 5*

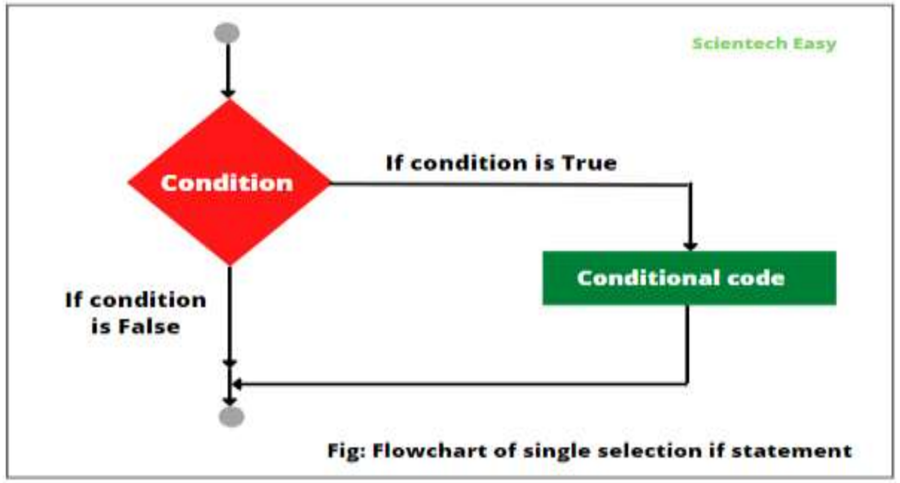
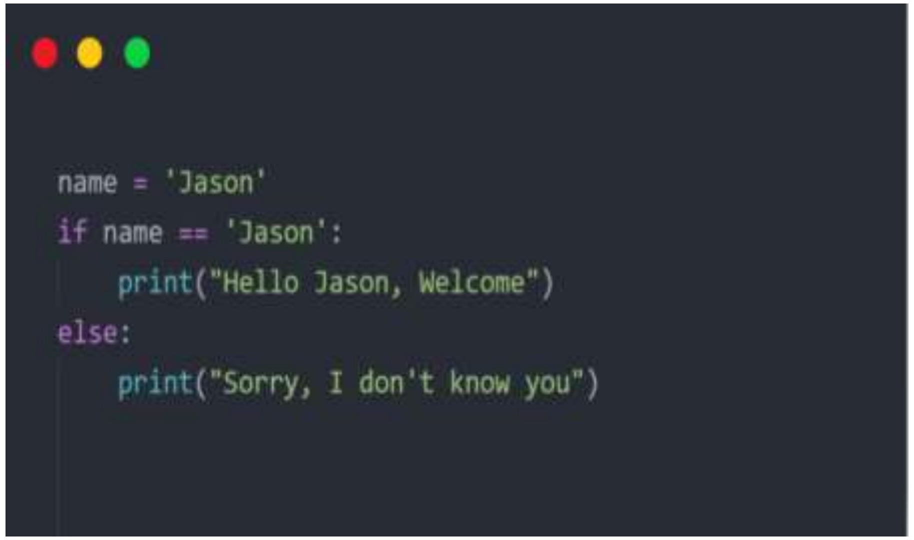


Fig: Flowchart of single selection if statement



```
name = 'Jason'
if name == 'Jason':
    print("Hello Jason, Welcome")
else:
    print("Sorry, I don't know you")
```

if-else Statements

Purpose:

- if-else statements allow a program to take two different actions based on whether a condition is True or False

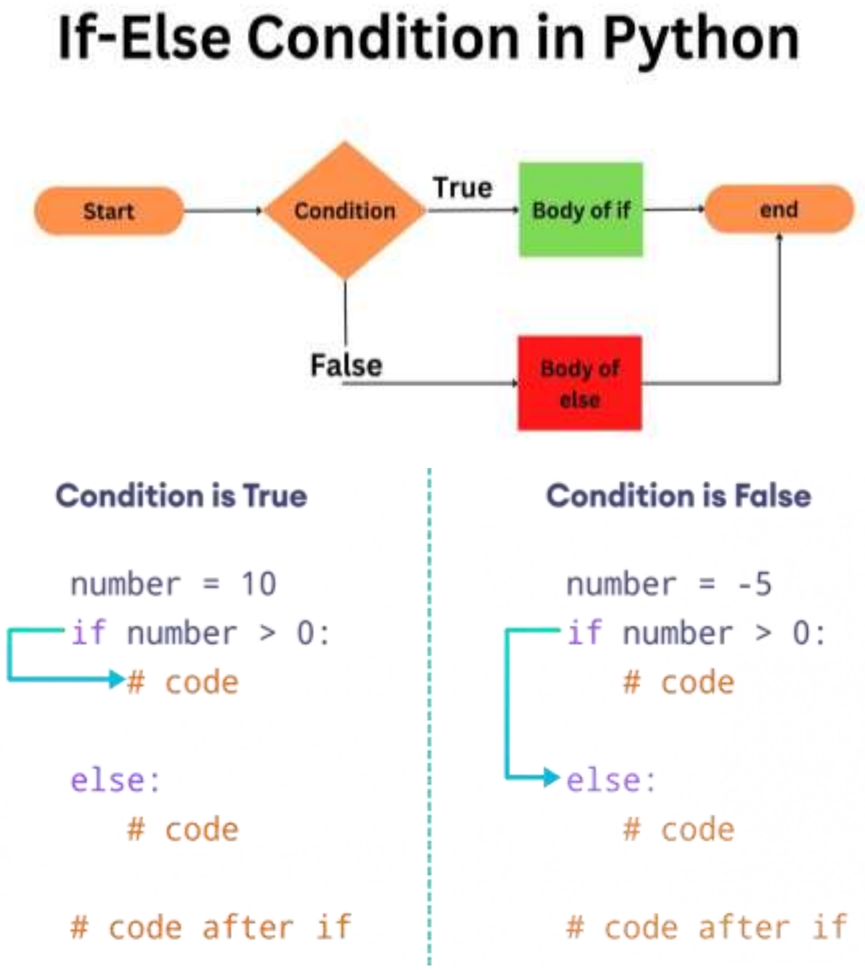
How it Works:

- The **if** block executes when the condition is True.
- The **else** block executes when the condition is False.
- This ensures that one of the two blocks will always run.

Example:

```
num = 3
if num > 5:
   print("The number is greater than 5")
else:
   print("The number is not greater than 5")
```

- Here, since *num* is not greater than 5, the else block runs and prints the second message



```
number = 10
if number > 0:
    # code

else:
    # code

# code after if
```

```
number = -5
if number > 0:
    # code

else:
    # code

# code after if
```

if-elif-else Statements

Purpose:

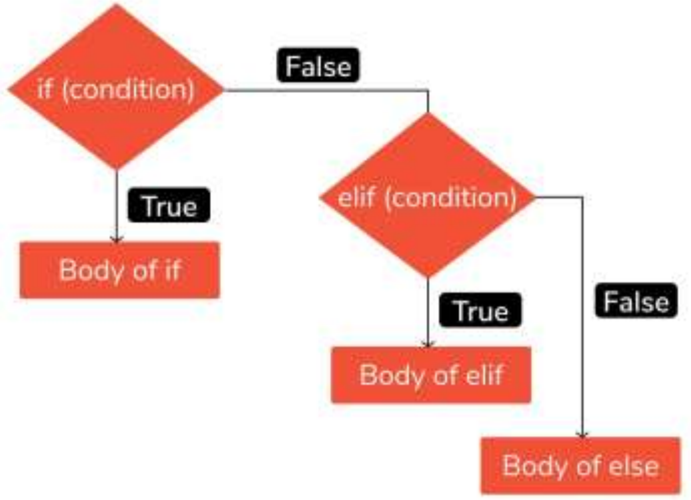
- Used when you need to evaluate multiple conditions and execute different blocks of code based on which condition is True.

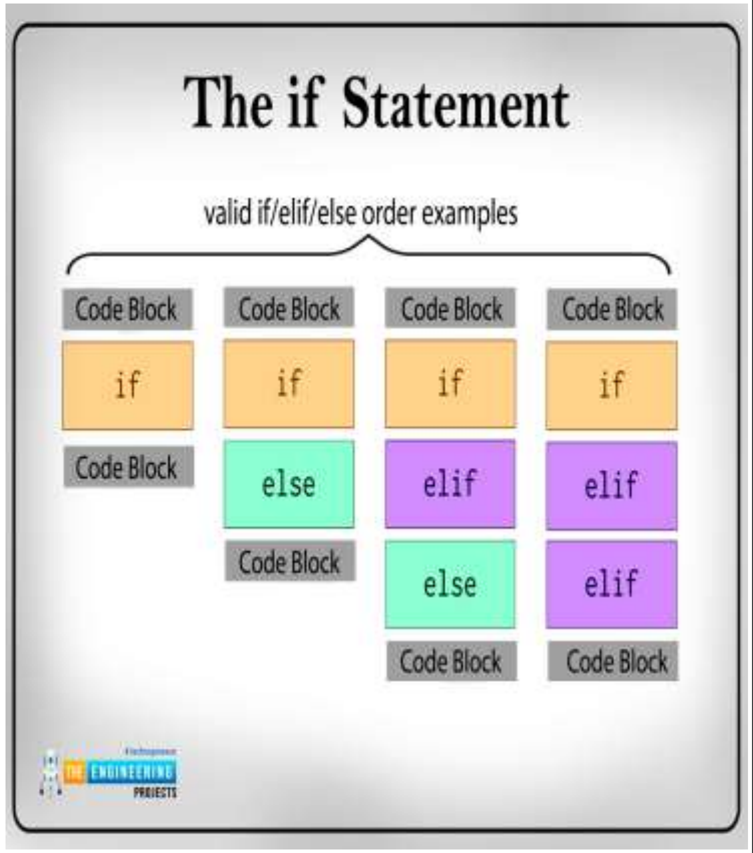
How it Works:

- The program checks the first condition with if.
- If True, it runs that block and skips the rest.
- If False, it checks the next condition using elif (else if).
- You can have multiple elif blocks.
- If none of the conditions are True, the else block is runs as a fallback.

- **Example**

```
score = 75
if score >= 90:
    print(“Grade: A”)
elif score >= 80:
    print(“Grade: B”)
elif score >= 70:
    print(“Grade: C”)
else:
    print(“Grade: D”)
```





Introduction to Loops

Definition:

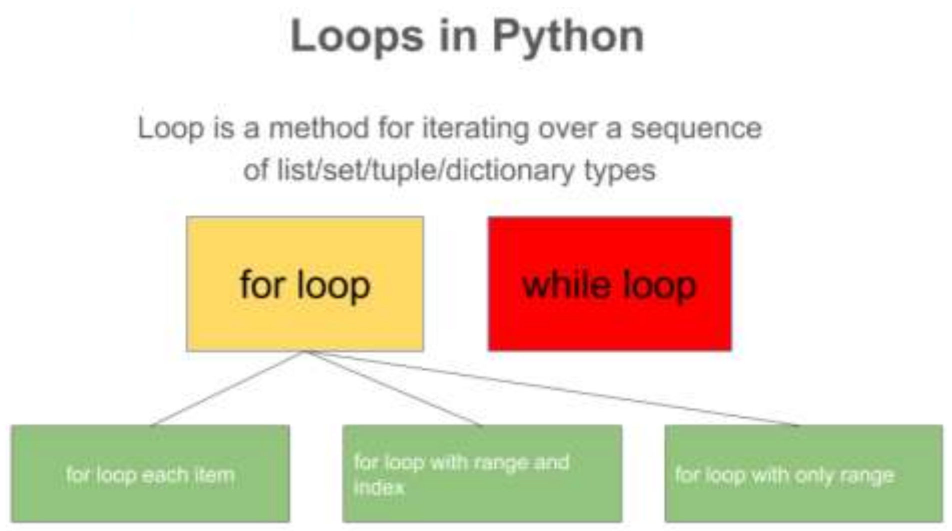
- Loops are control structures used to repeat a block of code until a specific condition is met.

Purpose:

- Automate repetitive tasks and reduce redundancy.

Types of Loops:

- **while Loop**: Repeats as long as a condition is true.
- **for Loop**: Iterates over items of a sequence like lists, tuples, or strings.

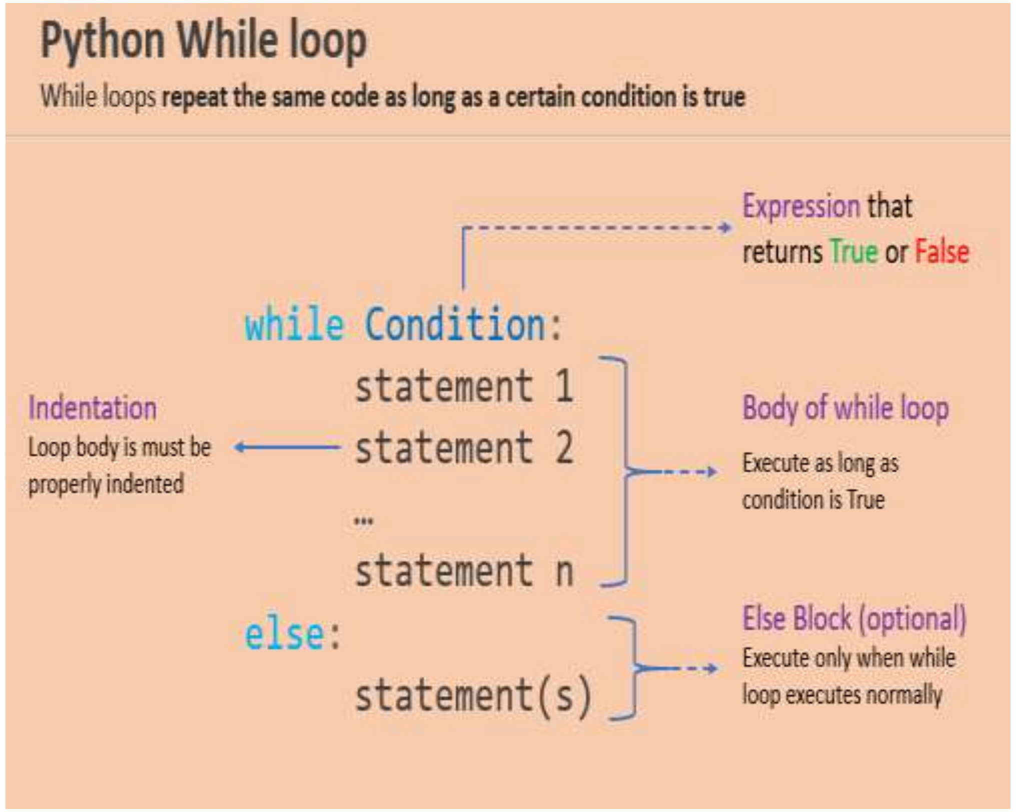

Loops in Python
Loop is a method for iterating over a sequence of list/set/tuple/dictionary types
for loop
while loop
for loop each item
for loop with range and index
for loop with only range

while Loop

- **How it Works:**
  - The while loop runs as long as the condition is true. If the condition becomes false, the loop stops.
- **Use Case:**
  - Ideal when the number of iterations is not known in advance.
- **Example:**

```
count = 0
while count < 3:
   print("Counting:", count)
   count += 1
```

- The loop checks if *count* is less than 3. If true, it prints the value and increases *count* by 1.
- Prints Counting: 0, Counting: 1, Counting: 2. Stops when count == 3.



Python While loop
While loops repeat the same code as long as a certain condition is true
Expression that returns True or False
while Condition:
statement 1
statement 2
...
statement n
else:
statement(s)
Indentation
Loop body is must be properly indented
Body of while loop
Execute as long as condition is True
Else Block (optional)
Execute only when while loop executes normally

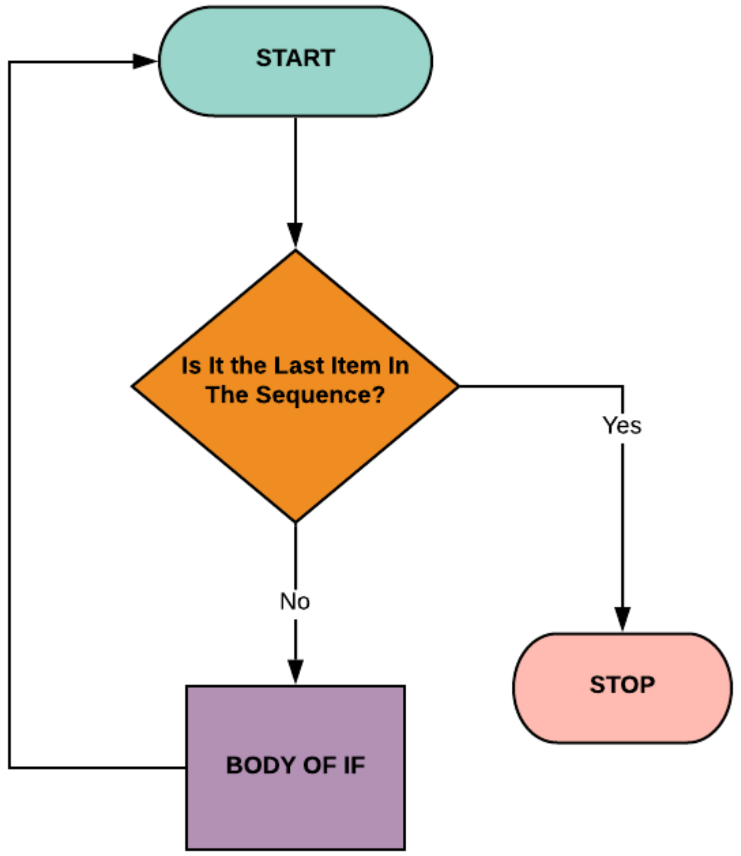
for Loop

- **How it Works:**
  - The for loop iterates items in a sequence (list, tuple, string, etc.).
- **Use Case:**
  - Best when the number of iterations is know or fixed.
- **Example 1 – List:**

```
colors = [“red”, “green”, “blue”]
for color in colors:
    print(“Color:”, color)
```

- **Example 2 – Dictionary:**

```
data = {“name”: “John”, “age”: 25}
for key, value in data.items():
    print(f“{key}: {value}”)
```



range() Function

- Generates a sequence of numbers, commonly used with for loops.
- **Syntax:**

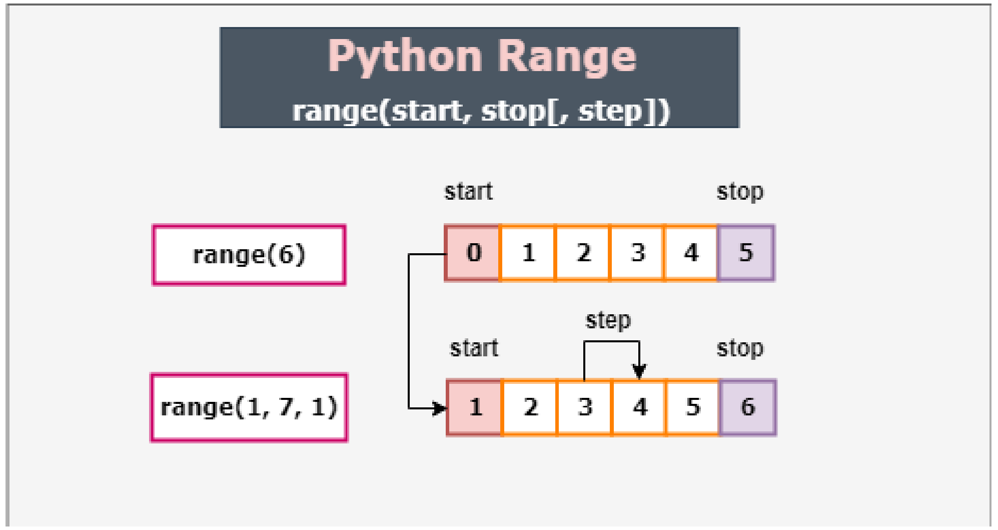
  range(start, stop, step)

- **Parameters of range():**
  - **start:** Starting number (default is 0).
  - **stop:** End number (non-inclusive).
  - **step:** Increment (default is 1).
- **Examples:**
  - **Standard Loop:**

```
for i in range(1, 6):
    print(i)  # Prints 1 to 5
```

  - **Reverse Loop:**

```
for i in range(5, 0, -1):
    print(i)  # Prints 5 to 1
```



break Statement

- **Purpose:**
  - Immediately stops the loop when a condition is met.
- **Example:**

```
for number in range(10):
    if number == 5:
        break
    print(number)
```

- Prints 0 to 4. Stops at 5.

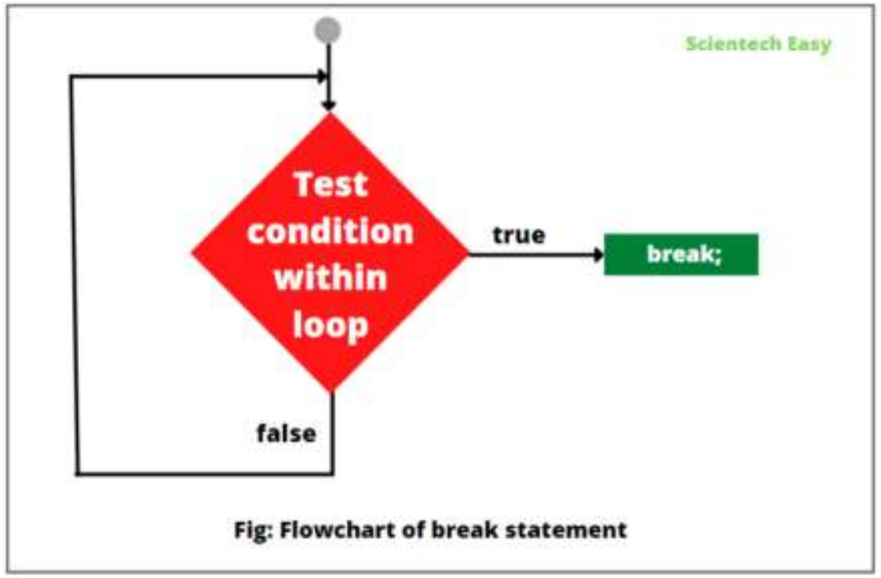
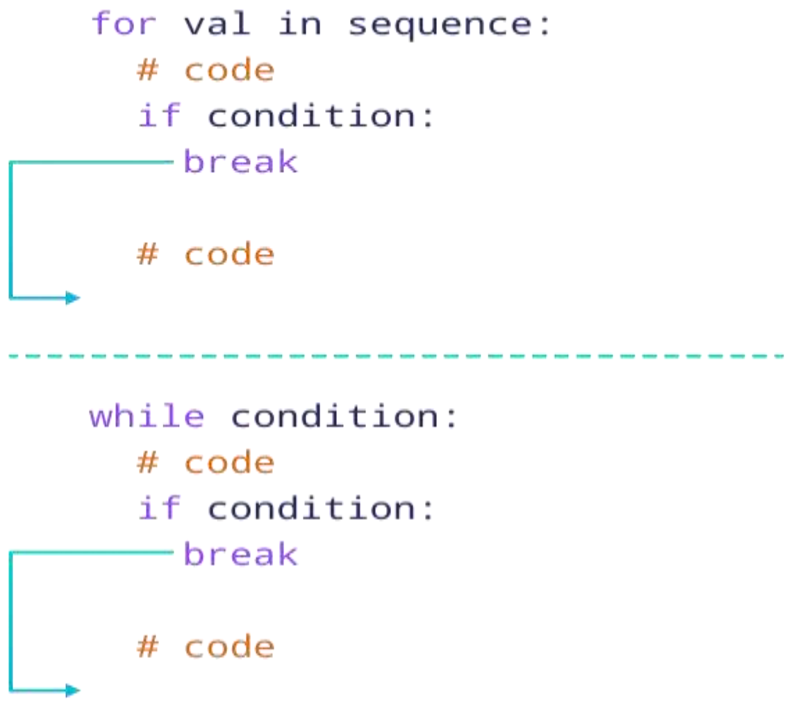


Fig: Flowchart of break statement



```
for val in sequence:
    # code
    if condition:
        break

    # code
```

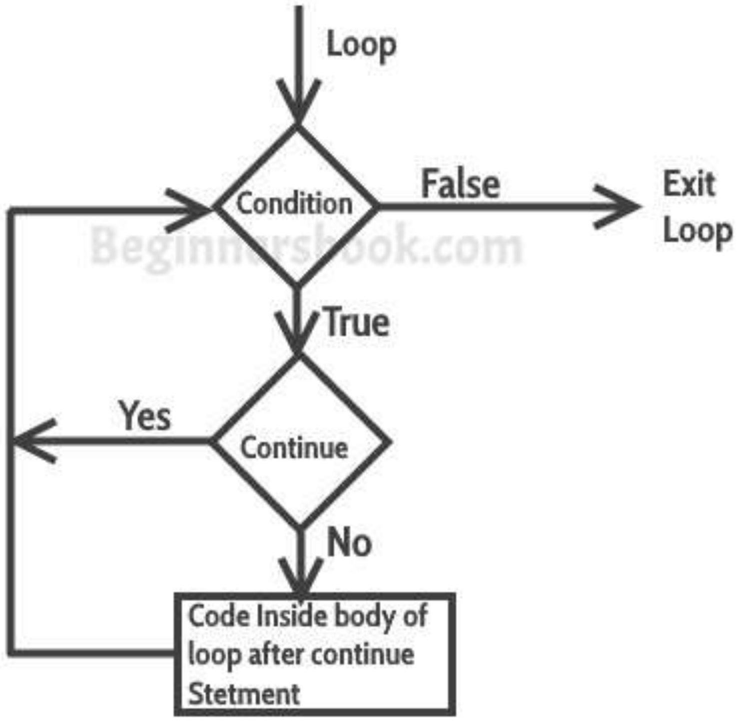
```
while condition:
    # code
    if condition:
        break

    # code
```

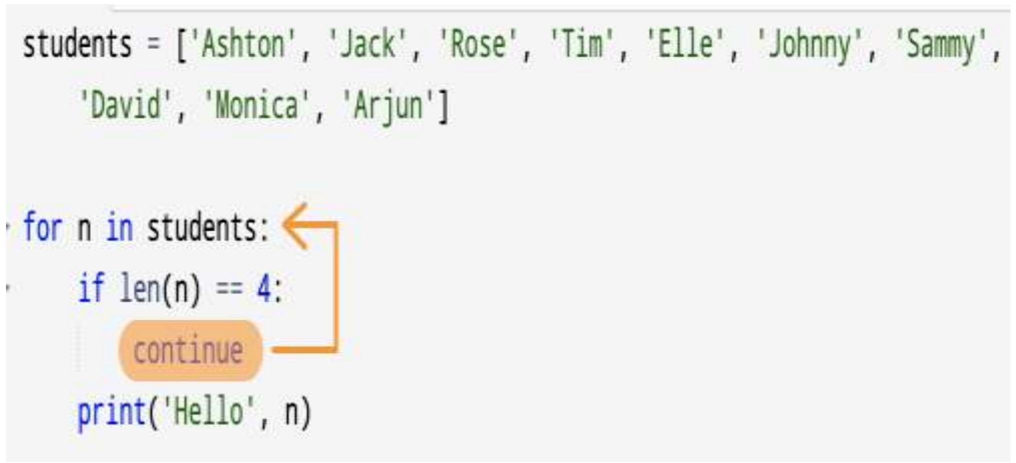
continue statement

- Skips the current iteration and moves to the next one.
- It doesn't terminate the loop but simply skips the remaining code for that iteration.
- **When to Use continue:**
  - To skip unwanted iterations (e.g., skipping certain values in a dataset).
- **Example:**

```
for i in range(5):
    if i == 2:
        continue
    print(i)
```

- Skips 2. Prints 0, 1, 3, 4.





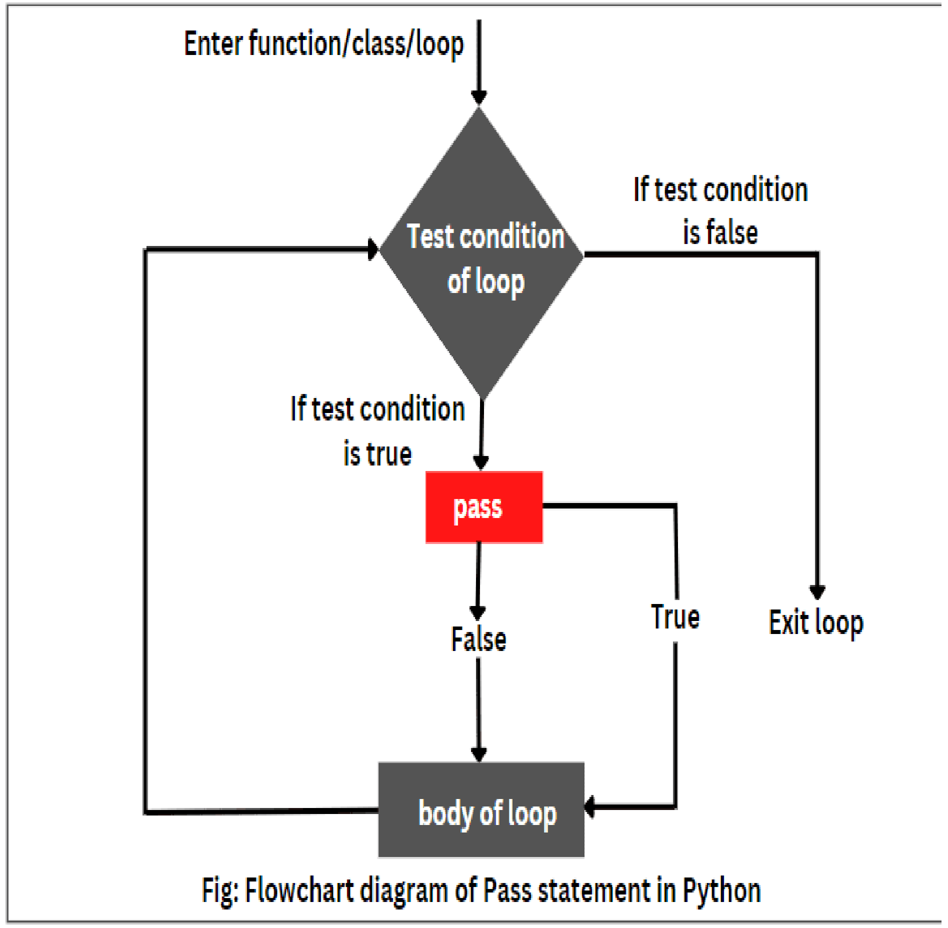
```
students = ['Ashton', 'Jack', 'Rose', 'Tim', 'Elle', 'Johnny', 'Sammy',
    'David', 'Monica', 'Arjun']

for n in students:
    if len(n) == 4:
        continue
    print('Hello', n)
```

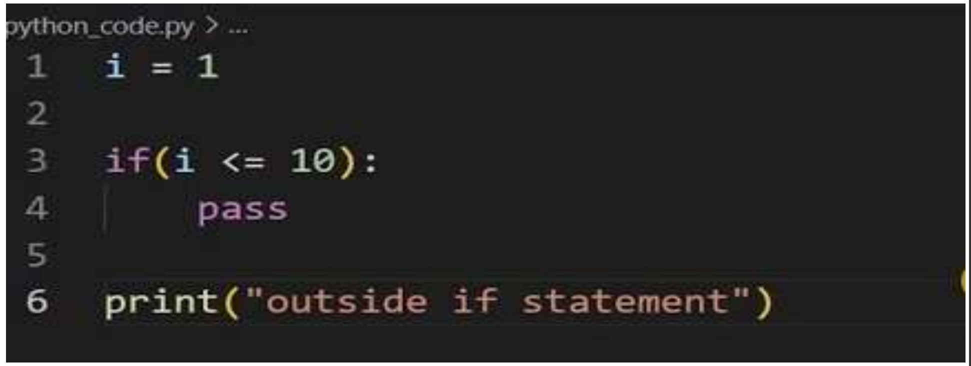
pass statement

- **Purpose:**
  - Does nothing, used where code is syntactically required but not yet implemented.
  - The pass statement is a null operation; nothing happens when it is executed. It's used as a placeholder for code you'll add later.
- **Example with pass:**

```
for i in range(5):
    if i < 3:
        pass
print(i)
```

- *pass* does nothing; all numbers 0 to 4 are printed.


Enter function/class/loop
Test condition of loop
If test condition is false
If test condition is true
pass
False
True
Exit loop
body of loop
Fig: Flowchart diagram of Pass statement in Python



```
i = 1

if(i <= 10):
    pass

print("outside if statement")
```

Happy Coding