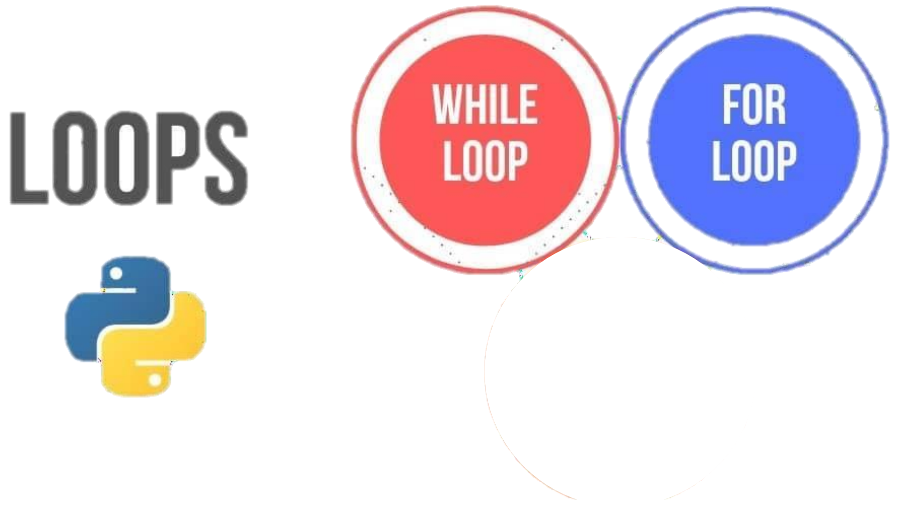
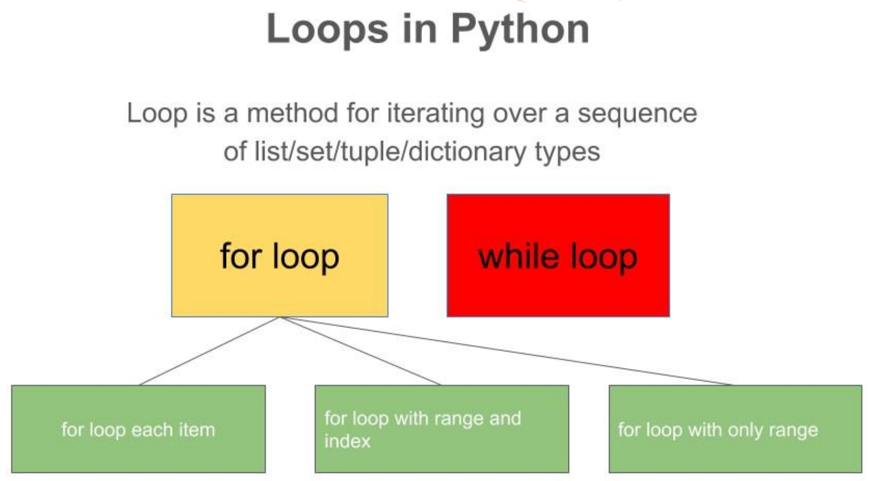
Introduction to Python Programming

Introduction to Loops in Python

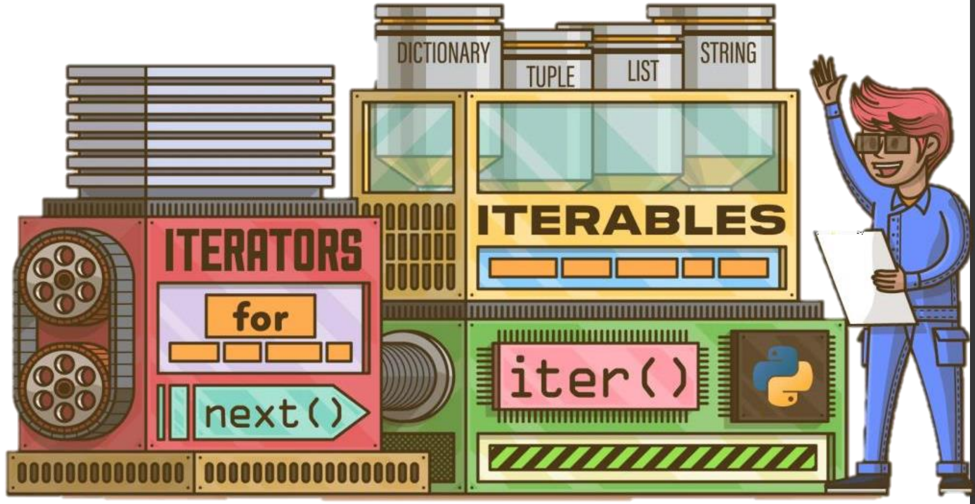
- **What are Loops?**
- Loops are programming structures that repeat a sequence of instructions until a specific condition is met. In Python, loops allow us to iterate over data structures or perform repetitive tasks efficiently.
- **Why Use Loops?**
- Loops help reduce redundancy, automate repetitive tasks, and make the code more efficient and readable.
- **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.

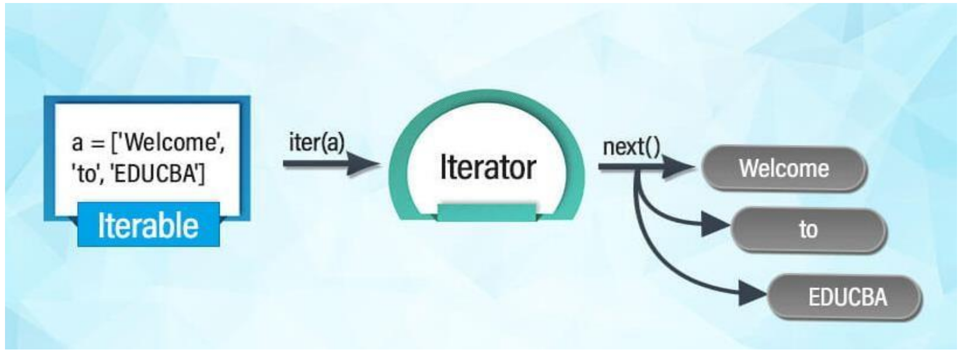


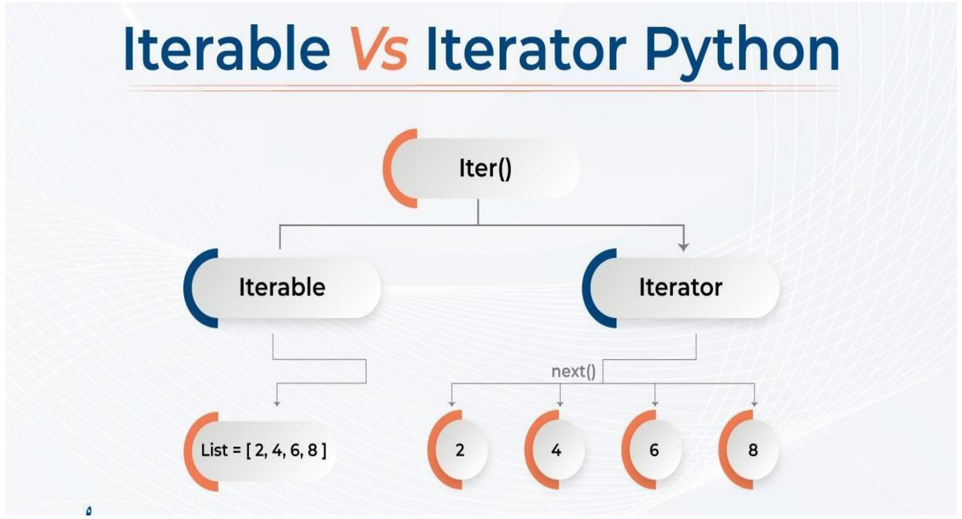

Loops in Python
Loop is a method for iterating over a sequence of list/set/tuple/dictionary types
for loop
while loop
for loop each item
for loop with range and index
for loop with only range

Iterating with Python Loops

- **What Does Iterating Mean?**
- Iterating refers to the process of accessing each item in a collection (like a list or a tuple) one by one. It is a fundamental aspect of loops in Python.
- **How Loops Perform Iteration:**
- Loops can be used to iterate over various Python data structures such as lists, dictionaries, sets, and even strings.
- **Key Points to Remember:**
- Loops allow sequential access to items in an iterable.
- Use loops to access or modify elements in lists, strings, and other sequences.

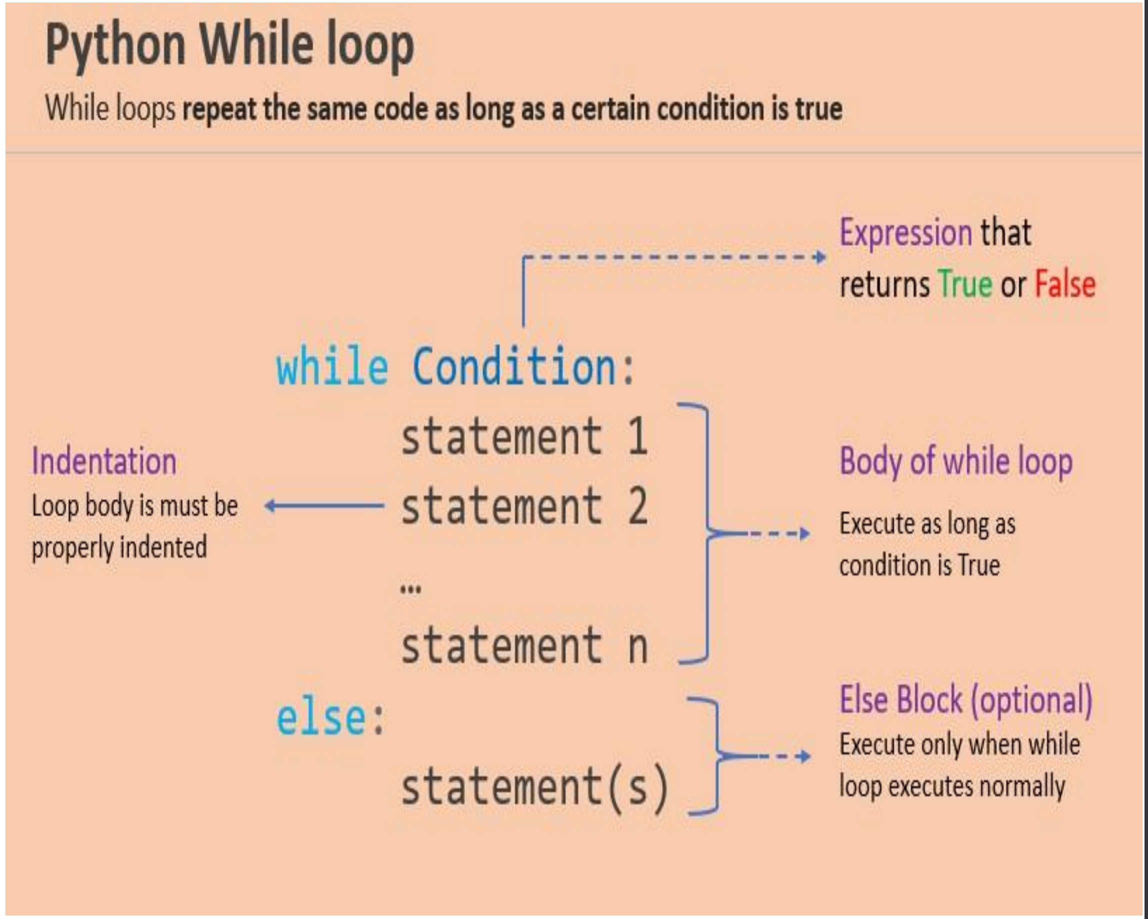






The while Loop

- **How while Loops Work:**
- The while loop runs as long as the condition is true. If the condition becomes false, the loop stops.
- **Useful Scenarios:**
- When you don’t know the number of iterations beforehand (e.g., reading data from a file until it ends).
- **Example with Explanation:**
- 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.



Python While loop

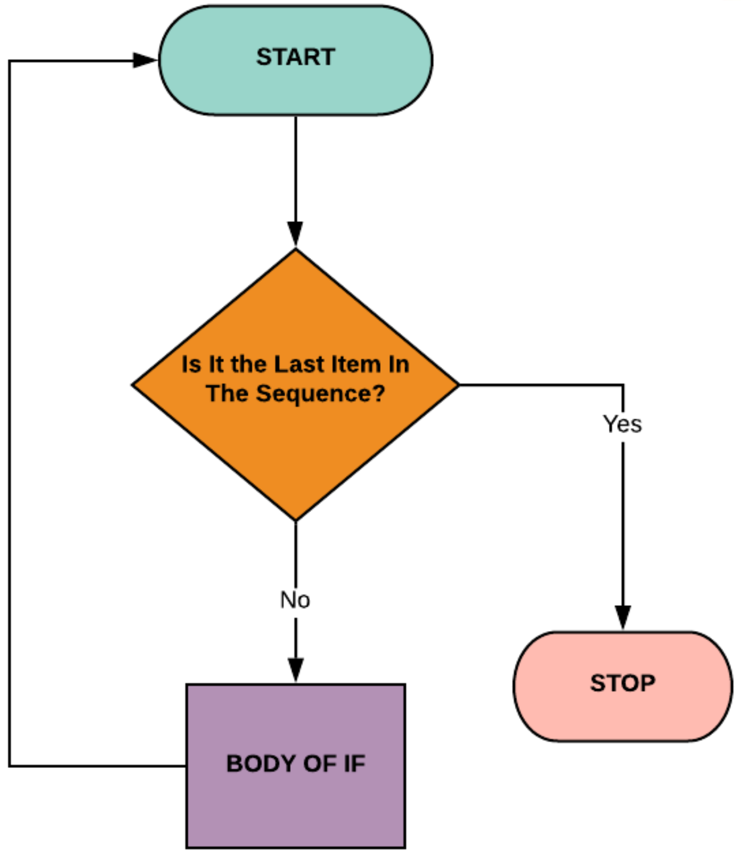
While loops **repeat the same code as long as a certain condition is true**

```
while Condition:
    statement 1
    statement 2
    ...
    statement n
else:
    statement(s)
```

Expression that returns True or False
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

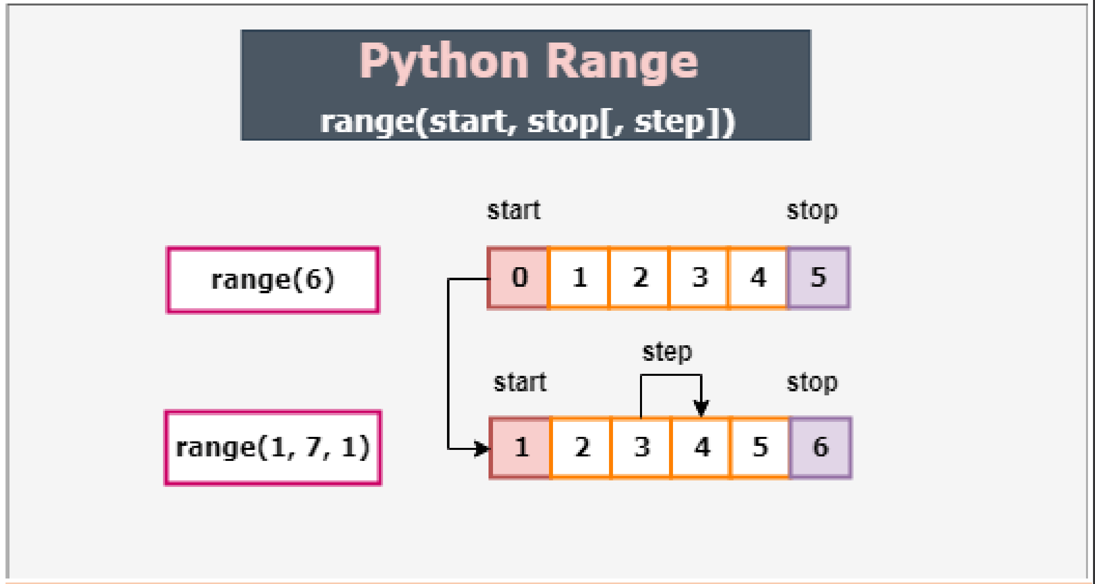
The for Loop

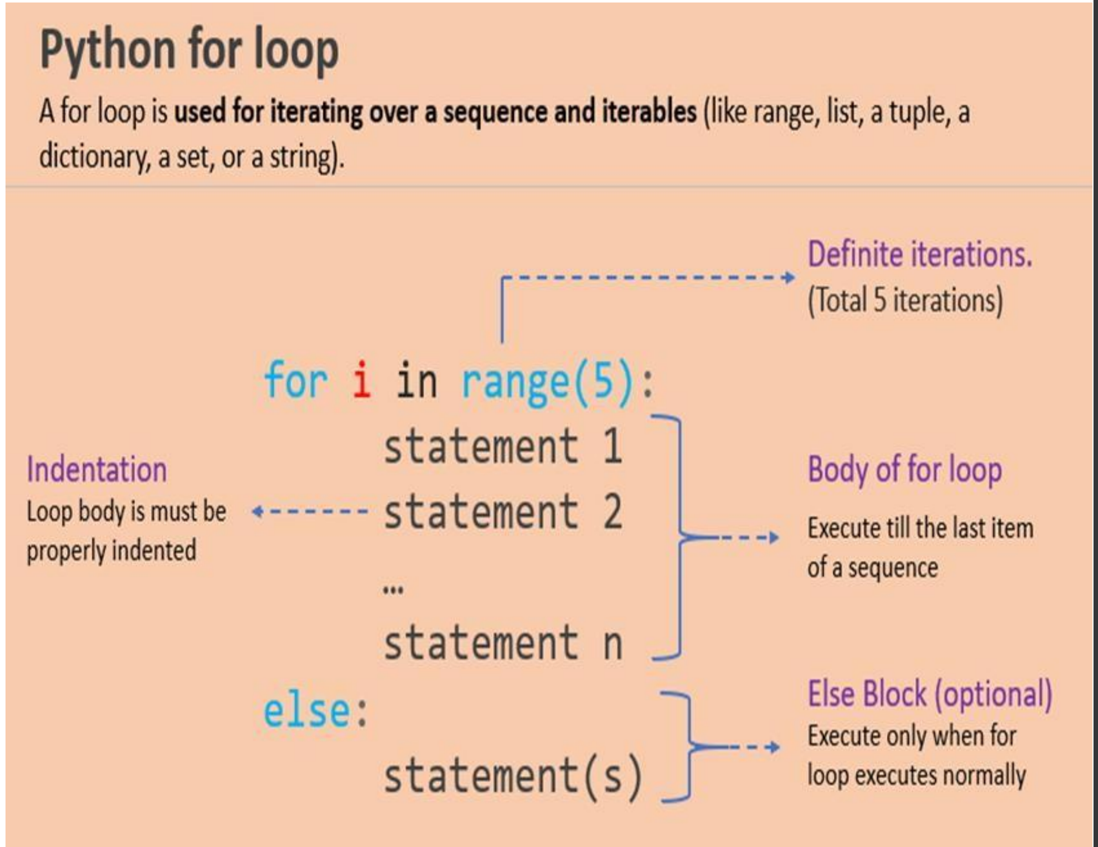
- **How for Loops Work:**
- The for loop iterates over each item of an iterable (like a list or string) until all items are processed.
- **When to Use for Loops:**
- When you know the number of iterations or need to iterate over each item in a collection.
- **Example with Explanation:**
- colors = ["red", "green", "blue"]
- for color in colors:
-     print("Color:", color)
- **Using for Loops with Dictionaries:**
- data = {"name": "John", "age": 25}
- for key, value in data.items():
-     print(f"{key}: {value}")



The range() Function

- **Purpose of range():**
- range() generates a sequence of numbers, making it useful for looping a specific number of times.
- **Parameters of range():**
- **start:** Starting number (default is 0).
- **stop:** End number (non-inclusive).
- **step:** Increment (default is 1).
- Example with range():
- for i in range(1, 6):
-     print(i)
- This prints numbers from 1 to 5. The loop ends before reaching 6.
- Additional Use Case –Reverse Iteration:
- for i in range(5, 0, -1):
-     print(i)




Python for loop
Definite iterations.
(Total 5 iterations)
Indentation
Loop body is must be properly indented
Body of for loop
Execute till the last item of a sequence
Else Block (optional)
Execute only when for loop executes normally

A for loop is **used for iterating over a sequence and iterables** (like range, list, a tuple, a dictionary, a set, or a string).

```
for i in range(5):
      statement 1
      statement 2
      ...
      statement n
else:
      statement(s)
```

The break Statement

- **What Does break Do?**
- The break statement allows you to exit a loop prematurely. It’s commonly used to stop loops based on a condition.
- **When to Use break:**
- When a specific condition is met, and you need to stop the loop immediately (e.g., finding an element in a list).
- **Example with break:**
- for number in range(10):
-     if number == 5:
-         break
-     print(number)
- This prints numbers from 0 to 4. The loop stops when it reaches 5.

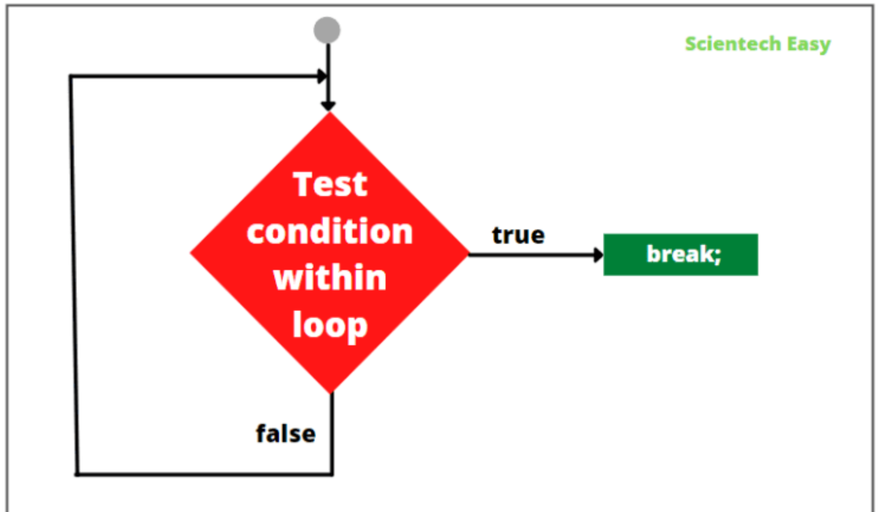
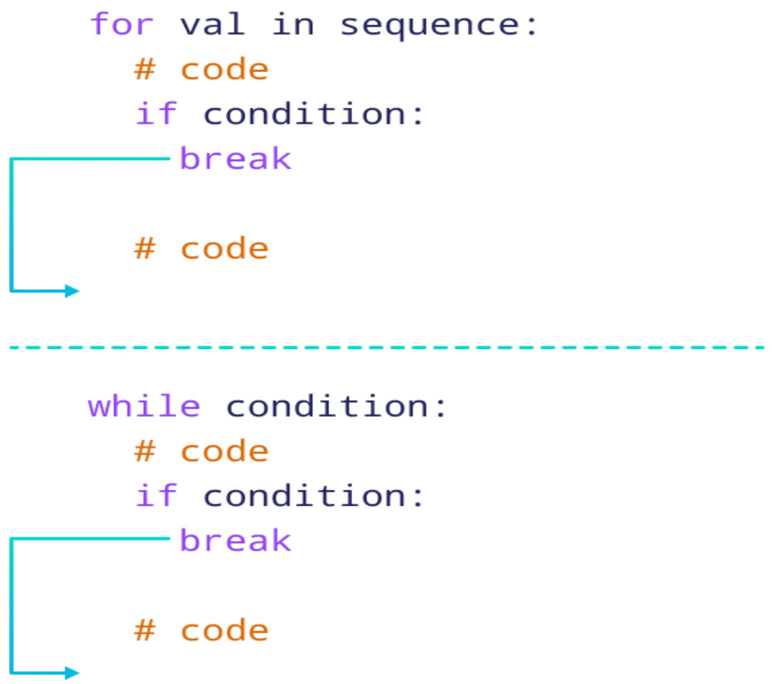


Fig: Flowchart of break statement

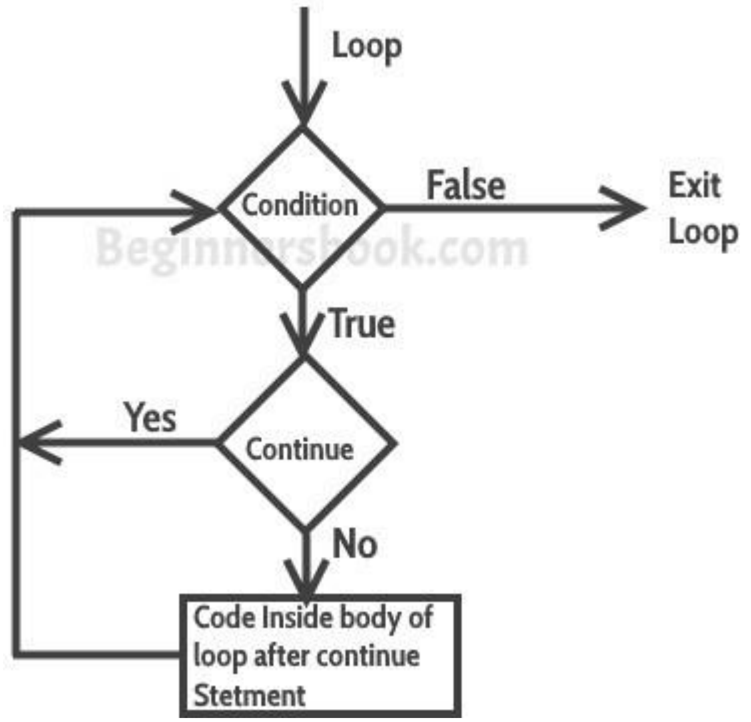


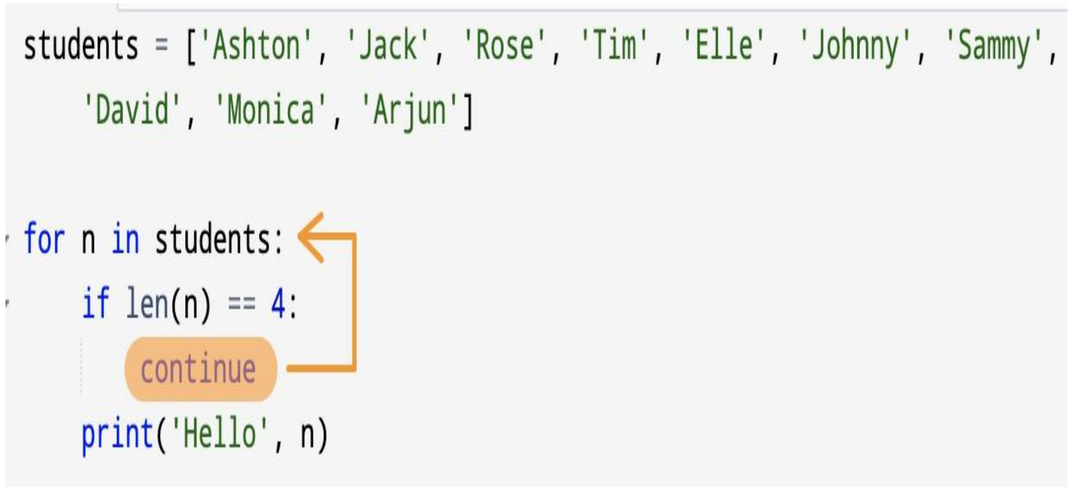
```
for val in sequence:
  # code
  if condition:
    break

  # code
```

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

  # code
```

The continue statement

- **What Does continue Do?**
- The 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 with continue
- for i in range(5):
  - if i == 2:
  - continue
- print(i)
- The loop prints all numbers except 2. When i == 2, the loop skips that iteration.

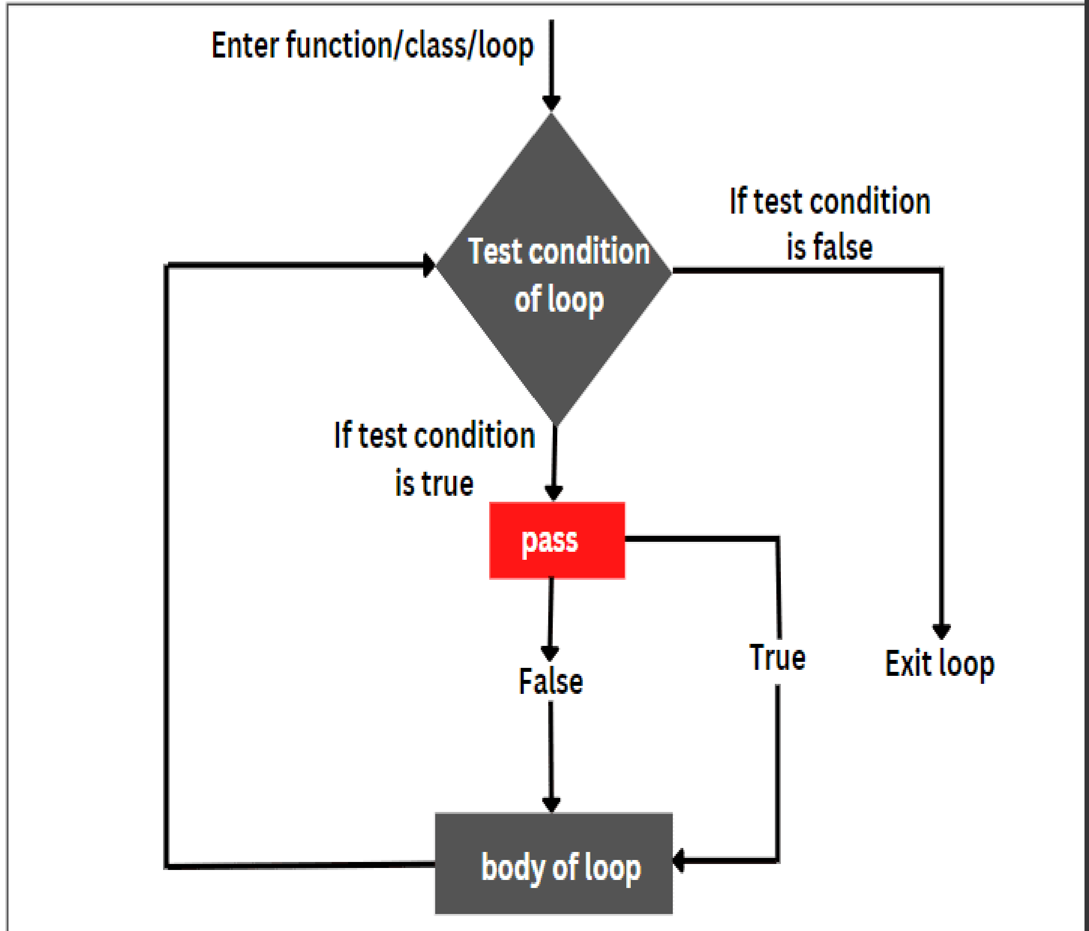




```
students = ['Ashton', 'Jack', 'Rose', 'Tim', 'Elle', 'Johnny', 'Sammy',
    'David', 'Monica', 'Arjun']

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

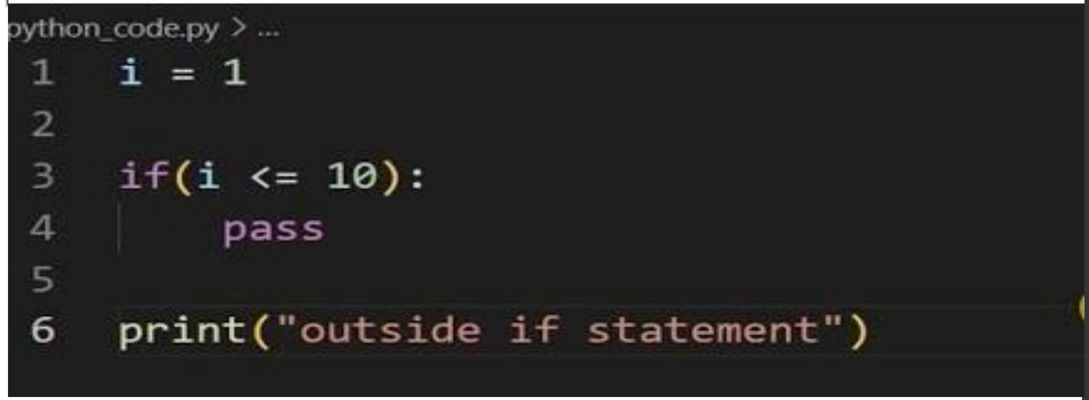
The pass statement

- The pass Statement
- What is pass Used For?
- 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.
- **When to Use pass**:
- In places where code is required syntactically, but you don't want to execute anything yet (e.g., in a try block without an exception handler yet).
- **Example with pass:**
- for i in range(5):
  - if i < 3:
  - pass
- print(i)
- This prints all numbers from 0 to 4, but pass does nothing when i is less than 3.



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



```
python_code.py > ...
i = 1

if(i <= 10):
    pass

print("outside if statement")
```