Statement Flow Control in Python

1. Types of Statements in Python

In programming, a statement is an instruction that the Python interpreter can execute. I is of three types

(i)Empty Statement: A statement that does nothing. It is used where syntax requires a statement but no action needs to be performed. It is represented by the **pass** keyword.

Example:

```
if x > 100:
    pass  # Empty statement; logic to be added later
else:
    print("x is small")
```

(ii)Simple Statement: A single logical line of code, usually containing one instruction.

Examples:

```
x = 10
print("Hello, World!")
name = input("Enter your name: ")
```

(iii)Compound Statement (or Block of Code): A statement that contains other statements within it. (a group of statements).

It consists of a header line (starting with a keyword like if, for, while, def) and a suite/block of indented statements.

Example:

```
if age >= 18:  # Header line
    print("You are an adult.")  # Indented block
    print("You can vote.")      # Indented block
```

2. Types of Statement Flow Control:

The order in which statements are executed in a program is called flow control. There are three basic types:

(i) Sequence: Statements are executed one after another in the order they are written, from top to bottom. This is the default flow.

(ii) Selection (or Decision Making): Used to execute a set of statements only if a certain condition is true. Allows the program to choose between different paths.

Keywords **if, else, elif** are used.

(iii) Repetition / Looping: It is used to execute a block of code repeatedly as long as a condition is true. Used for tasks that require repetition.

Keywords **for, while** are used.

3. The Conditional Statements: Thes are used for selection-based flow of control.

(i) if statement: It executes a block of code only if a condition is True.

Syntax:

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

Example:

```
num = 10
if num > 0:
    print("Number is positive")
```

(ii) if-else statement: It provides an alternative path. Executes one block if the condition is True, and another block if it is False.

Syntax:

```
if condition:
    # block if condition is True
else:
    # block if condition is False
```

Example:

```
age = 16
if age >= 18:
    print("Eligible to vote")
else:
    print("Not eligible to vote")
```

(iii) if-elif-else statement: It is used to check multiple conditions in sequence.

As soon as one condition is found True, its block is executed, and the rest are skipped.

Syntax:

```
if condition1:
    # block if condition1 is True
elif condition2:
    # block if condition2 is True
elif condition3:
    # block if condition3 is True
else:
    # block if none of the above are True
```

Example:

```
marks = 85
if marks >= 90:
    print("Grade A")
elif marks >= 75:
    print("Grade B")
elif marks >= 50:
    print("Grade C")
else:
    print("Grade F")
```

4. Nested if: It means an if or if-else statement inside another if or else statement.

It is used for more complex decision-making scenarios.

Example:

```
num = 15
if num >= 0:
    if num == 0:  # This if is nested inside the outer if
        print("Zero")
    else:
        print("Positive number")
else:
    print("Negative number")
```

5. The range() Function: The range() function generates a sequence of numbers. It is commonly used to control the number of iterations in a for loop.

Syntax:

(a) range(stop) → Generates numbers from 0 to stop-1.

Ex: range(5) gives: 0, 1, 2, 3, 4

(b) range(start, stop) → Generates numbers from start to stop-1.

Ex: range(2, 6) gives: 2, 3, 4, 5

(c) range(start, stop, step) → Generates numbers from start to stop-1, incrementing by step.

Example:

* range(1, 10, 2) gives: 1, 3, 5, 7, 9
* range(5, 0, -1) gives: 5, 4, 3, 2, 1

6. for Loop: A for loop is used for iterating over a sequence (like a list, tuple, string, or the sequence generated by range()).

It is often called a definite loop because the number of iterations is known (based on the length of the sequence).

Syntax:

```
for variable in sequence:
    # body of the for loop
```

Examples:

```
# Iterating through a list
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)

# Using range() to repeat an action 5 times
for i in range(5):
    print("Hello")          # Prints "Hello" 5 times

# Calculating sum of first 10 natural numbers
sum = 0
for num in range(1, 11):
    sum += num
print("Sum is:", sum)
```

7. while Loop: A while loop repeats a block of code as long as a given condition is True. It is often called an indefinite loop because the number of iterations is not known in advance; it depends on when the condition becomes False.

Important: Ensure the loop condition eventually becomes False to avoid an infinite loop.

Syntax:

```
while condition:
    # body of the while loop
```

Examples:

```
# Countdown from 5 to 1
count = 5
while count > 0:
    print(count)
    count -= 1  # Decrement count; crucial to avoid infinite loop
print("Blastoff!")

# Taking user input until a specific value is entered
num = int(input("Enter a number (0 to stop): "))
while num != 0:
    print("You entered:", num)
    num = int(input("Enter a number (0 to stop): "))
```

8. Jump statements (break and continue): These statements alter the normal flow of a loop.

(i) break Statement: It is used to exit (terminate) the loop immediately. The program control moves to the statement immediately after the loop.

Example:

```
for i in range(1, 10):
    if i == 5:
        break  # Loop stops when i becomes 5
    print(i)
# Output: 1 2 3 4
```

(ii) continue Statement: It is used to skip the rest of the code inside the loop for the current iteration and move to the next iteration.

Example:

```
for i in range(1, 6):
    if i == 3:
        continue                    # Skip printing 3
    print(i)
# Output: 1 2 4 5
```

9. Nested Loop: A loop inside the body of another loop is called a nested loop.

For every single iteration of the outer loop, the inner loop completes all its iterations.

Example: Pattern Printing (Very Common)

 # Printing a 3x3 grid of stars

```
for i in range(3):      # Outer loop for rows
    for j in range(3):  # Inner loop for columns
        print("*", end=" ")
    print()  # Moves to the next line after each row
```

Output:

```
* * *
* * *
* * *
```

Example: Multiplication Table

```
for i in range(1, 3):       # Outer loop for first number
    for j in range(1, 4):   # Inner loop for second number
        print(i, "x", j, "=", i * j)
    print("---")  # Separator after each table
```