Day-9: Understanding Control Flow & Loops in Python

1. Conditional Statements

Conditional statements allow programs to make decisions based on specific conditions. These statements control the flow of execution in a program.

1.1 if-else Statement

The if-else statement executes one block of code if the condition is True and another block if it is False.

Syntax:

```
if condition:
    # Code to execute if condition is True
else:
    # Code to execute if condition is False
```

Example:

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

Explanation: If age is 18 or more, the message "Eligible to vote" is printed; otherwise, "Not eligible to vote" is displayed.

1.2 Nested if Statement

A nested if statement means an if statement inside another if statement. This is useful for making more complex decisions.

Syntax:

```
if condition1:
    if condition2:
```

```
    # Code to execute if both conditions are True
```

Example:

```
num = 10
if num > 0:
  print("Positive number")
  if num % 2 == 0:
    print("Even number")
```

Explanation: The program first checks if the number is positive, and if true, it further checks if it's even.

1.3 if-elif-else Statement

The if-elif-else statement is used when multiple conditions need to be checked sequentially.

Syntax:

```
if condition1:
  # Code to execute if condition1 is True
elif condition2:
  # Code to execute if condition1 is False and condition2 is True
else:
  # Code to execute if none of the above conditions are True
```

Example:

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

```
    print("Grade: C")
```

Explanation: The program assigns a grade based on the marks obtained.

1.4 Ternary Operator (One-line if-else)

The ternary operator allows writing an if-else condition in a single line.

Syntax:

```
result = value1 if condition else value2
```

Example:

```
age = 20

status = "Adult" if age >= 18 else "Minor"

print(status)
```

Explanation: If age is 18 or more, status is set to "Adult"; otherwise, it is set to "Minor".

1.5 match-case (Case Statement)

Introduced in Python 3.10, the match-case statement works like a switch-case in other languages, allowing pattern-based execution.

Syntax:

```
match value:
    case pattern1:
        # Code to execute if value matches pattern1
    case pattern2:
        # Code to execute if value matches pattern2
    case _:
        # Default case if no match is found
```

Example:

```
day = "Monday"
```

```
match day:
  case "Monday":
    print("Start of the workweek")
  case "Friday":
    print("Weekend is near")
  case _:
    print("Regular day")
```

Explanation: The program prints different messages based on the day of the week.

---

2. Loops

Loops are used to execute a block of code multiple times.

2.1 for Loop

A for loop iterates over a sequence (like a list, tuple, string, or range).

Syntax:

```
for item in sequence:
  # Code to execute for each item
```

Example:

```
for i in range(1, 6):
  print(i)
```

Explanation: This loop prints numbers from 1 to 5.

2.2 while Loop

A while loop continues execution as long as a given condition remains True.

Syntax:

```
while condition:
  # Code to execute while condition is True
```

Example:

```
count = 0
while count < 5:
    print(count)
    count += 1
```

Explanation: The loop prints numbers from 0 to 4, incrementing count each time.

---

3. Loop Control Statements

Loop control statements modify loop behavior.

3.1 break Statement

The break statement exits the loop immediately.

Example:

```
for i in range(1, 10):
    if i == 5:
        break
    print(i)
```

Explanation: The loop stops execution when i reaches 5.

3.2 continue Statement

The continue statement skips the current iteration and moves to the next one.

Example:

```
for i in range(1, 10):
    if i % 2 == 0:
        continue
    print(i)
```

Explanation: Only odd numbers are printed since even numbers are skipped.

3.3 else Clause in Loops

The else block in a loop executes only if the loop completes normally (i.e., without encountering a break).

Example:

```
for i in range(1, 6):
    print(i)
else:
    print("Loop completed successfully")
```

Explanation: Since there is no break, the else block executes.

Key Takeaways

- **Conditional statements** control the flow of execution based on conditions.
- **Loops** are used to execute repetitive tasks efficiently.
- **Loop control statements** (break, continue, else) help modify loop behavior.
- **Choose the right loop:** Use for loops when iterating over a sequence and while loops when the number of iterations is unknown.