Isha Dhiman

AI/ML Enthusiast | Intern @CodroidHub | First-Year Engineering Student

Artificial Intelligence/Machine Learning UpSkills Notebook

From Basics to Real-World — Starts Your ML Journey

What Are Control Flow Statements?

- **Control flow statements** are special instructions in Python that control the order in which your code runs.
- They decide which block of code should run next, based on conditions or repetition.

Types of Control Flow Statements

1️⃣ Conditional Statements

Conditional statements let your program make decisions by running different blocks of code based on whether a condition is True or False.

Types of Conditional Statements :

- **if statement**

Runs a block of code only if a given condition is True.

Example:

In [1]:
```
x = 10
if x > 5:
    print("x is greater than 5")
```

```
x is greater than 5
```

- **if-else Statement**

Runs one block of code if the condition is True, otherwise runs the else block.

Example:

In [2]:
```
age = 16
if age >= 18:
    print("You can vote.")
else:
    print("You are not eligible to vote yet.")
```

```
You are not eligible to vote yet.
```

- **if-elif-else ladder**

Checks multiple conditions in order, and runs the block for the first True condition.

Example:

In [3]:
```
marks = 75

if marks >= 90:
    print("Grade A")
elif marks >= 75:
    print("Grade B")
elif marks >= 60:
```

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

```
Grade B
```

- **Nested if**

An if statement inside another if statement, useful for more detailed checks.

Example:

In [4]:
```
num = 8

if num >= 0:
    if num % 2 == 0:
        print("Positive and even")
    else:
        print("Positive and odd")
else:
    print("Negative number")
```

```
Positive and even
```

Loops

Loops let your program repeat a block of code multiple times until a condition is met or over a sequence.

Types of loops :

- **While loop**

  Repeats a block of code as long as a condition is True.

Example:

In [6]:
```
count = 0

while count < 3:
    print("Count is:", count)
    count += 1
```

```
Count is: 0
Count is: 1
Count is: 2
```

- **for loop**

  Repeats a block of code for each item in a sequence (like a list, tuple, string, or range).

Example:

In [7]:
```
for i in range(1, 6):
    print(i)
```

```
1
2
3
4
5
```

- **Nested loops**

  A loop inside another loop, useful for working with grids, tables, or patterns.

In [8]:
```
for i in range(1, 4):
    for j in range(1, 4):
        print(f"i={i}, j={j}")
```

```
i=1, j=1
i=1, j=2
i=1, j=3
i=2, j=1
i=2, j=2
i=2, j=3
i=3, j=1
i=3, j=2
i=3, j=3
```

Loop Control Statements

Special statements that control how loops behave — they can stop a loop, skip steps, or do nothing.

Types of Loop Control Statements :

- **break**

Stops the loop immediately, even if the condition or sequence is not finished.

Example:

In [11…
```
for i in range(10):
    if i == 5:
        break
    print(i)
```

```
0
1
2
3
4
```

- **continue**

Skips the current iteration and moves to the next loop cycle.

Example:

In [12…
```
for i in range(5):
    if i == 2:
        continue
    print(i)
```

```
0
1
3
4
```

- **pass**

Does nothing — it acts as a placeholder when a statement is required syntactically but no action is needed.

Example:

In [13…
```
for letter in "Python":
    if letter == "h":
        pass  # Placeholder, does nothing
    else:
        print("Current Letter:", letter)
```

```
Current Letter: P
Current Letter: y
Current Letter: t
Current Letter: o
Current Letter: n
```

The range() function

The range() function generates a sequence of numbers — it's often used with for loops.

Syntax:

range(stop)

range(start, stop)

range(start, stop, step)

Examples:

1️⃣Basic range:

In [18…
```
for i in range(5):
```

```
    print(i)
```

```
0
1
2
3
4
```

2️⃣Range with start and stop:

In [20…
```
for i in range(2, 6):
    print(i)
```

```
2
3
4
5
```

3️⃣Range with step:

In [21…
```
for i in range(1, 10, 2):
    print(i)
```

```
1
3
5
7
9
```

"else" Clauses on Loops

In Python, loops can have an else block that runs only if the loop finishes normally (not terminated by break).

Example:

In [23…
```
for i in range(5):
    print(i)
else:
    print("Loop finished without break.")
```

```
0
1
2
3
4
Loop finished without break.
```

With break:

In [24…
```
for i in range(5):
    if i == 3:
        break
    print(i)
else:
    print("Loop finished without break.")
```

```
0
1
2
```

"match" Statement (Structural Pattern Matching)

The match statement (introduced in Python 3.10) is like a switch in other languages. It matches a value against patterns.

In [ ]:
```
#Syntax:
match variable:
    case pattern1:
        # Code block
    case pattern2:
        # Code block
    case _:
        # Default case
```

Example:

In [28…
```
status_code = 404

match status_code:
    case 200:
        print("OK")
```

```
    case 404:
        print("Not Found")
    case 500:
        print("Server Error")
    case _:
        print("Unknown Status")
```

```
Not Found
```

Summary

In this notebook, you learned how Python controls the flow of a program using different tools:

1️⃣ Conditional Statements

Use if, elif, else, and nested if to make decisions and run code only when certain conditions are true.

2️⃣ Loops

Use while and for loops to repeat actions automatically.

Use nested loops to handle patterns or multiple layers of repetition.

3️⃣ Loop Control Statements

break stops a loop early.

continue skips the current loop step and goes to the next one.

pass does nothing but keeps your code valid — useful as a placeholder.

4️⃣ range() Function

Generates a sequence of numbers — useful with for loops.

5️⃣ else with Loops

Runs only if the loop completes normally without break.

6️⃣ match Statement

A modern way to handle multiple conditions with cleaner, pattern-based matching (like switch in other languages).

Connect @
Mail (Isha Dhiman): dhimanisha177@gmail.com
Contact: +91-9253165167

Isha Dhiman

AI/ML Enthusiast | Intern @CodroidHub | First-Year Engineering Student

Isha Dhiman