Python for DevOps – Conditional Statements

Introduction to Conditional Statements

In real life, we constantly make decisions:

- If it's raining, take an umbrella.
- If traffic is clear, drive fast, else take an alternate route.
- If marks ≥ 50, student passes; else fails.

In Python, **conditional statements** help programs **decide what action to take based on conditions**.
They allow the program to **branch logic** instead of executing line by line blindly.

Why Conditional Statements Matter in DevOps?

In **DevOps automation scripts**, decisions often need to be made based on:

- Whether a server is **up or down**.
- If a **service is running**, take one action; else restart it.
- If a **deployment succeeds**, notify the team; else rollback.
- If CPU usage is above 80%, trigger an **alert**.

Example Scenarios:

1. **Server Health Check**
    - *If* server is reachable → print "Server Active"
    - *Else* print "Server Down"
2. **Log File Monitoring**
    - *If* error keyword found → send alert mail.
    - *Else* print "No issues detected."

Thus, **conditional statements** make scripts intelligent and dynamic, which is critical in automation.

Types of Conditional Statements in Python

(a) The if Statement

The simplest conditional form. Executes a block **only if** a condition is true.

Syntax:

```
if condition:
    # code block if condition is true
```

Example (Python):

```
age = 20
if age >= 18:
    print("You are eligible to vote.")
```

(b) The if-else Statement

Adds an **alternative action** if the condition is false.

Syntax:

```
if condition:
    # block when condition is true
else:
    # block when condition is false
```

Example (Python):

```
marks = 45
if marks >= 50:
    print("Pass")
else:
    print("Fail")
```

(c) The if-elif-else Ladder

Used when there are **multiple conditions**.
Python evaluates conditions one by one until it finds the first true one.

Syntax:

```
if condition1:
   # block1
elif condition2:
   # block2
else:
   # block3
```

Example (Python):

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

DevOps Real-Time Scenarios

**(These are illustrative only. They show *where conditionals fit into DevOps*.
They cannot run on your laptops without real servers/pipelines)**

Scenario 1: Service Status Check

```
service_running = False

if service_running:
   print("Service is running smoothly.")
else:
   print("Service is down. Restart required.")
```

Why DevOps Relevant?

- Automating server monitoring scripts.
- Automatically checking health of Jenkins, Docker, or Kubernetes pods.

Scenario 2: Deployment Validation

```
build_status = "SUCCESS"

if build_status == "SUCCESS":
   print("Deployment can proceed.")
elif build_status == "FAILED":
   print("Deployment stopped. Fix build errors.")
else:
   print("Unknown status. Check manually.")
```

Why DevOps Relevant?

- Automates **decision-making in CI/CD pipelines**.
- Saves engineers from manual validation of builds.

Python Practice Tasks

Task 1: Number Check

Write a program to check if a given number is **positive, negative, or zero**.

Task 2: Even or Odd

Ask the user for a number and print whether it's **even or odd**.

Task 3: Simple Calculator

Take two numbers and an operator (+, -, *, /) from the user.
Use **if-elif-else** to perform the correct calculation.

Task 4: Voting Eligibility

Input age from the user.

- If age ≥ 18 → print "Eligible to Vote"
- Else → print "Not Eligible"

Task 5: Grade Evaluator

Ask user for marks:

- ≥90 → Grade A
- ≥75 → Grade B
- ≥50 → Grade C
- Else → Fail

Python for DevOps – Loops in Python

Introduction to Loops

In programming, loops allow us to **repeat a block of code** multiple times without rewriting it.

Why Loops Matter in DevOps?

In **DevOps**, you often need to repeat tasks:

- Checking **all servers in a cluster**.
- Iterating through **log files** to detect errors.
- Deploying an application to **multiple environments** (dev → test → prod).
- Monitoring resource usage for **all containers** in Kubernetes.

Example: Instead of manually pinging 100 servers, a loop can do it in seconds.

Types of Loops in Python

(a) for Loop

Used when you know **how many times** you want to repeat.

Syntax:

```
for variable in sequence:
    # code block
```

Example (Python):

```
for i in range(5):
    print("Iteration:", i)
```

Prints numbers 0 to 4.

(b) while Loop

Used when you **don't know in advance** how many times to run, but continue until a condition is false.

Syntax:

```
while condition:
    # code block
```

Example (Python):

```
count = 1
while count <= 5:
    print("Count:", count)
    count += 1
```

Prints numbers 1 to 5.

Loop Control Statements

- break → exits loop immediately.
- continue → skips current iteration and continues.
- pass → does nothing (placeholder).

Example:

```
for num in range(1, 6):
    if num == 3:
        continue
    print(num)
```

Skips 3 and prints 1, 2, 4, 5.

DevOps Real-Time Scenarios with Loops

**(These are illustrative only. They show *where conditionals fit into DevOps*.
They cannot run on your laptops without real servers/pipelines)**

Scenario 1: Checking Multiple Servers

```
servers = ["server1", "server2", "server3"]
for s in servers:
    print("Pinging", s)
    # In real scripts, use subprocess to ping
```

Relevance: Automating server health checks in a loop.

Scenario 2: Monitoring Log File Until Error Appears

```
error_found = False
line_number = 1
while not error_found and line_number <= 10:
    log = "Line " + str(line_number)
    print("Checking:", log)
    if "error" in log.lower():
        error_found = True
    line_number += 1
print("Scan completed")
```

Relevance: Automating log scanning in CI/CD pipelines.

Scenario 3: Restarting a Service Until Successful

```
attempt = 1
success = False
while not success and attempt <= 3:
    print("Attempt:", attempt, "- Starting service...")
    # In real world: check return code
    if attempt == 2:
```

```
        success = True
        print("Service started successfully.")
    attempt += 1
```

Relevance: Auto-retrying failed deployments or restarts.

Python Practice Tasks

Task 1: Sum of Numbers

Write a program using a for loop to calculate the **sum of numbers from 1 to 50**.

Task 2: Multiplication Table

Take a number from the user and print its **multiplication table (1–10)** using a loop.

Task 3: Factorial Calculator

Use a while loop to find the **factorial of a number**.

Task 4: Print Patterns

Use nested loops to print:

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

Task 5: Prime Number Checker

Ask user for a number and check if it's **prime or not** using a loop.