**Control Statements & Loops**

**✅ A. INTRODUCTION TO CONTROL STATEMENTS**

**🔹 What Are Control Statements?**

Control statements are Python’s **decision-making structures** — they allow your script to **respond dynamically** to conditions or events.

Think of it like:

"If something is true, do this. Otherwise, do something else."

You control **what runs**, **how often**, and **when to stop**.

**✅ B. CONDITIONAL STATEMENTS: if, elif, else**

**🔹 Basic if Statement**

value = 42

if value > 10:

print("Value is greater than 10")

🔍 **Explanation**:

* The condition value > 10 is evaluated.
* If **True**, the block indented below if runs.
* Otherwise, it’s skipped.

**🔹 Full if-elif-else Ladder**

score = 85

if score >= 90:

print("Grade: A")

elif score >= 75:

print("Grade: B")

elif score >= 60:

print("Grade: C")

else:

print("Grade: F")

🔍 **Explanation**:

* Python evaluates **from top to bottom**.
* First True condition gets executed, and the rest are **skipped**.
* If **none** of the conditions are True, else runs.

🚨 **Only one block ever runs** in an if-elif-else chain.

**🔹 DevOps Example: Choose Environment Based on Tag**

tag = "v1.0.0-prod"

if "dev" in tag:

env = "development"

elif "staging" in tag:

env = "staging"

elif "prod" in tag:

env = "production"

else:

env = "unknown"

print("Deploying to:", env)

**🔹 Logical Operators**

region = "us-west-1"

zone = "a"

if region == "us-west-1" and zone == "a":

print("Targeting us-west-1a")

and, or, and not can be used to combine or negate conditions.

**✅ C. NESTED IF STATEMENTS**

**🔹 Example:**

x = 12

if x > 10:

if x < 20:

print("x is between 10 and 20")

🔍 **Best Practice**:  
Avoid deep nesting if you can **flatten** the logic:

if 10 < x < 20:

print("x is between 10 and 20")

**✅ D. THE match-case STATEMENT (Python 3.10+)**

**🔹 Basic Pattern Matching**

code = 404

match code:

case 200:

print("Success")

case 404:

print("Not Found")

case 500:

print("Server Error")

case \_:

print("Unknown status")

🔍 **Explanation**:

* Replaces long if-elif chains.
* case \_ acts as the default.
* Cleaner for large comparisons (HTTP codes, menu selections, etc.).

**✅ E. LOOPS IN PYTHON**

Loops are **structures that repeat** a block of code until a condition is met.

**✅ F. THE for LOOP**

**🔹 Loop Over a List**

tools = ["Git", "Jenkins", "Vault"]

for tool in tools:

print(f"Installing {tool}...")

🔍 Explanation:

* for tool in tools: iterates over the list tools.
* Each item is assigned to tool, and the indented block executes.

**🔹 Using range()**

for i in range(1, 6):

print(i)

**range(start, stop)** generates numbers from start to stop-1.

**🔹 DevOps Example: Apply Helm Chart to Multiple Namespaces**

namespaces = ["dev", "qa", "prod"]

for ns in namespaces:

print(f"helm upgrade --install myapp ./chart -n {ns}")

**🔹 Looping Over Dictionary Items**

versions = {"python": "3.11", "node": "20", "java": "17"}

for key, val in versions.items():

print(f"{key.upper()} version is {val}")

**✅ G. THE while LOOP**

**🔹 Syntax:**

i = 0

while i < 5:

print("Count:", i)

i += 1

🔍 Explanation:

* Loop runs as long as the **condition is True**
* Update variables **inside** to avoid infinite loops

**🔹 DevOps Example: Retry SSH Until Successful**

import time

success = False

attempts = 0

while not success and attempts < 5:

attempts += 1

print(f"Attempt {attempts}: Trying SSH connection...")

success = try\_ssh()

if not success:

time.sleep(2)

else:

print("SSH succeeded!" if success else "Failed after 5 attempts")

**✅ H. LOOP CONTROL STATEMENTS**

**🔹 break**

for i in range(10):

if i == 3:

break

print(i) # Output: 0 1 2

🔍 Exits the **entire loop** when condition is met.

**🔹 continue**

for i in range(5):

if i == 2:

continue

print(i) # Output: 0 1 3 4

🔍 Skips the current iteration, **continues with the next one**.

**🔹 pass**

for i in range(5):

if i == 2:

pass # Placeholder

print(i)

🔍 A **no-op** statement — use it when code is **to be written later**.

**✅ I. else CLAUSE WITH LOOPS**

**🔹 Rare but powerful**

for i in range(3):

print(i)

if i == 10:

break

else:

print("Loop completed successfully")

🔍 else block only runs if the **loop is not broken**

**✅ J. NESTED LOOPS**

**🔹 Example:**

for i in range(3):

for j in range(2):

print(f"i={i}, j={j}")

Be cautious — can become inefficient quickly (O(n²)).

**🔹 DevOps Example: Deploy Multiple Apps in Multiple Namespaces**

apps = ["frontend", "backend"]

namespaces = ["dev", "staging"]

for app in apps:

for ns in namespaces:

print(f"kubectl apply -f {app}.yaml -n {ns}")

**✅ K. LOOP COMPREHENSIONS**

**🔹 List Comprehension**

squares = [x \* x for x in range(5)]

print(squares) # [0, 1, 4, 9, 16]

With condition:

even\_squares = [x \* x for x in range(10) if x % 2 == 0]

Clean, compact, Pythonic.

**✅ L. HANDY LOOP UTILITIES**

**🔹 enumerate()**

tools = ["Git", "Docker", "Vault"]

for i, tool in enumerate(tools):

print(f"{i}: {tool}")

**🔹 zip()**

names = ["App1", "App2"]

versions = ["v1", "v2"]

for name, version in zip(names, versions):

print(f"{name} runs on {version}")

**🔹 all() and any()**

checks = [True, True, False]

print(all(checks)) # False (not all True)

print(any(checks)) # True (at least one True)

**✅ M. BEST PRACTICES**

1. ✅ Keep conditionals clear and short
2. ✅ Use comprehensions over verbose loops where possible
3. ✅ Use match-case for multiple branches (Python 3.10+)
4. ✅ Prefer for over while unless loop control requires condition
5. ✅ Always add loop termination logic to avoid infinite loops
6. ✅ Don’t overuse nested loops – flatten logic where possible

**✅ N. SUMMARY**

| **Concept** | **Description** |
| --- | --- |
| if/elif/else | Basic decision making |
| match-case | Structured pattern matching (3.10+) |
| for loop | Iterate over a sequence (list, dict, etc.) |
| while loop | Loop until a condition is met |
| break | Exit the loop |
| continue | Skip current iteration |
| pass | Placeholder code |
| else on loop | Executes only if loop wasn’t interrupted |
| Comprehension | Clean 1-line loops for collections |