Python Notes — Conditional Statements & Control Structures

---

1. Introduction

In Python, **control structures** determine **the flow of execution** — which statements run, how many times, and under what conditions.

There are two main types:

1. **Conditional Statements** → Execute code **based on conditions**.
2. **Loops (Control Structures)** → Execute code **repeatedly** until a condition is met.

---

2. Conditional Statements

Conditional statements are used to **make decisions** in code.
They allow your program to **choose different paths** based on logical tests.

Syntax:

```
if condition:
    # code executes if condition is True
elif another_condition:
    # code executes if above is False but this is True
else:
    # code executes if all above are False
```

Example 1: Simple If

```
age = 20
```

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

Output:

```
You are eligible to vote.
```

Example 2: If-Else

```
num = 7
if num % 2 == 0:
    print("Even number")
else:
    print("Odd number")
```

Output:

```
Odd number
```

Example 3: If-Elif-Else

Used when you want to check **multiple conditions**.

```
marks = 82
if marks >= 90:
    print("Grade A")
elif marks >= 75:
    print("Grade B")
elif marks >= 60:
    print("Grade C")
else:
    print("Fail")
```

Output:

```
Grade B
```

Example 4: Nested If

One `if` statement inside another — used for multi-level conditions.

```
x = 10
if x > 0:
    if x % 2 == 0:
        print("Positive even number")
    else:
        print("Positive odd number")
else:
    print("Negative number")
```

Output:

```
Positive even number
```

---

3. Control Structures (Loops)

Loops are used to **repeat a block of code** as long as a condition remains true.

A. `for` Loop

Used to iterate over a **sequence** (like a list, tuple, or range).

Syntax:

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

Example:

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

Output:

```
Number: 1
Number: 2
Number: 3
Number: 4
Number: 5
```

B. `while` Loop

Repeats code **while a condition is True**.

Syntax:

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

Example:

```
x = 1
while x <= 5:
    print(x)
    x += 1
```

Output:

```
1
2
3
4
5
```

---

4. Loop Control Statements

These help **control the flow** inside loops.

Statement	Description	Example
`break`	Terminates the loop immediately	Stop loop at a condition
`continue`	Skips current iteration, continues next	Skip specific values
`pass`	Placeholder — does nothing	Used for empty blocks

Example — `break`

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

Output:

```
0
1
2
3
4
```

Example — `continue`

```
for i in range(6):
    if i == 3:
        continue
    print(i)
```

Output:

```
0
1
```

```
2
4
5
```

Example — `pass`

```
for i in range(5):
    pass   # placeholder for future logic
```

---

5. The `range()` Function

The `range()` function generates a sequence of numbers, useful in loops.

Syntax:

```
range(start, stop, step)
```

Examples:

```
range(5)         # 0 to 4
range(1, 6)      # 1 to 5
range(2, 10, 2)  # 2, 4, 6, 8
```

---

6. Combining Loops with Conditions

```
for i in range(1, 11):
    if i % 2 == 0:
        print(i, "is even")
    else:
        print(i, "is odd")
```

---

7. Nested Loops

A loop inside another loop.

```
for i in range(1, 4):
    for j in range(1, 3):
        print(i, j)
```

Output:

```
1 1
1 2
2 1
2 2
3 1
3 2
```

---

8. Infinite Loop Example (Be Careful!)

```
while True:
    print("Runs forever!")
    break  # stop manually or add a condition
```