LEVEL 3 - Control Flow (Conditions + Loops)

1. Conditional Statements in Python

Conditional statements allow a program to **make decisions**.
They check conditions (True/False) and execute code based on the result.

In real life, we use conditions every day:

- *"If it rains, take an umbrella."*
- *"If you score above 90, you get an A grade."*

Python does the same using **if, else, elif**, and other decision-making tools.

1.1 The if Statement

The **if** statement runs a block of code only when the condition is **True**.

Syntax

```
if condition:
  statement
```

Example

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

If the condition is False, Python simply skips the block.

1.2 The if–else Statement

Use **if–else** when there are only two possible outcomes.

Syntax

```
if condition:
  statement1
else:
  statement2
```

Example

```
marks = 40
if marks >= 50:
    print("Pass")
else:
    print("Fail")
```

1.3 The elif (Else If) Statement

Use **elif** when you have **multiple conditions**.

Syntax

```
if condition1:
    statement1
elif condition2:
    statement2
elif condition3:
    statement3
else:
    default_statement
```

Example

```
score = 85
if score >= 90:
    print("Grade A")
elif score >= 80:
    print("Grade B")
elif score >= 70:
    print("Grade C")
```

```
else:
  print("Needs Improvement")
```

1.4 Nested Conditions

The condition **inside another condition** is called a nested condition.

Example

```
age = 20
has_id = True
if age >= 18:
  if has_id:
    print("Entry allowed")
  else:
    print("ID required")
else:
  print("Not eligible")
```

Nested conditions help check **multiple dependent rules**.

1.5 Boolean Logic in Conditions

Python uses **logical operators** to combine multiple conditions:

Operator	Meaning	Example
and	Both conditions must be True	age > 18 and is_student
or	At least one must be True	age > 18 or has_permission
not	Reverses the condition	not is_raining

Example

```
age = 19
is_student = True
if age >= 18 and is_student:
    print("Discount allowed")
```

1.6 Comparison Operators in Conditions

Comparison operators evaluate to **True or False**.

Operator	**Meaning**	**Example**
==	equal to	a == b
!=	not equal to	a != b
>	greater than	a > b
<	less than	a < b
>=	greater than or equal to	a >= b
<=	less than or equal to	a <= b

Example

```
temp = 30

if temp != 25:

    print("Temperature is not 25")
```

1.7 Membership Operators (in, not in)

Used to check whether a value **exists in a collection** (string, list, tuple).

Operators

Operator	**Meaning**
in	checks presence
not in	checks absence

Example

```
fruits = ["apple", "mango", "banana"]

if "apple" in fruits:

    print("Apple is available")
```

```
if "kiwi" not in fruits:
    print("Kiwi is not available")
```

1.8 Identity Operators (is, is not)

Used to compare memory identity — whether two variables refer to the same object.

Operators

Operator	Meaning
is	same object in memory
is not	not the same object

Example

```
a = None
if a is None:
    print("No value assigned")
```

Useful for None checks.

1.9 Shorthand if (Optional but Useful)

Python allows writing **short conditions in one line**.

Single-line if

```
age = 20
if age > 18: print("Adult")
```

Shorthand if–else

```
marks = 80
result = "Pass" if marks >= 50 else "Fail"
print(result)
```

This is helpful for **clean, compact expressions**.

2. Loops in Python — Complete Beginner-Friendly Notes

Loops allow your program to **repeat actions automatically**, without writing the same code again and again.
They are essential for automation, patterns, calculations, games, AI logic, data processing, and more.

Python has **two main loops**:

- for loop → repeats a fixed number of times
- while loop → repeats until a condition becomes False

2.1 What Is a Loop?

A loop is a structure that allows you to run a block of code multiple times.

Real-Life Examples:

- Repeating "Good Morning" 10 times
- Counting from 1 to 100
- Checking every student's marks
- Printing stars in a pattern
- Sending 100 messages automatically

2.2 The for Loop

The for loop repeats code **for each item** in a sequence (numbers, list, string, range, etc.).

Basic Syntax

```
for variable in sequence:

  # repeated code
```

Example

```
for i in range(5):

  print("Hello")
```

Output:

```
Hello
Hello
Hello
```

```
Hello
Hello
```

2.3 The range() Function

range() creates a sequence of numbers for loops.

1. range(n)

Generates numbers from 0 to n-1.

```
for i in range(5):
    print(i)
```

Output: 0 1 2 3 4

2. range(start, end)

Generates numbers from **start to end-1**.

```
for i in range(2, 7):
    print(i)
```

Output: 2 3 4 5 6

3. range(start, end, step)

Step controls the gap between numbers.

```
for i in range(1, 11, 2):
    print(i)
```

Output: 1 3 5 7 9

Reverse counting:

```
for i in range(10, 0, -1):
    print(i)
```

2.4 The While Loop

A while loop repeats code **as long as the condition is True**.

Basic Syntax

```
while condition:
   # repeated code
```

Example: count from 1 to 5

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

2.5 Difference Between for and while

Feature	for loop	while loop
Best for	Known repetitions	Unknown repetitions
Uses	range(), sequences	Conditions
Risk	Very safe	Can create infinite loops

2.6 The Loop else Clause

else runs **only when the loop completes normally** (not stopped by break).

Example

```
for i in range(3):
   print(i)
else:
   print("Loop finished!")
```

2.7 Nested Loops

A loop inside another loop.

Example: table

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

2.8 Infinite Loops (Important Awareness)

A loop that **never stops**.

Example

```
while True:
  print("Running forever")
```

Infinite loops are used in:

- Games
- Robotics
- Sensors
- Servers

But beginners must avoid them accidentally.

2.9 Input Validation Loops

Used to make sure the user enters the correct data.

Example: ask age until valid

```
age = input("Enter age: ")
while not age.isdigit():
  print("Invalid input! Enter numbers only.")
  age = input("Enter age: ")
print("Your age is", age)
```

2.10 Real-Life Mini Examples

1. Print even numbers

```
for i in range(2, 21, 2):
  print(i)
```

2. Countdown timer

```
i = 5
while i > 0:
    print(i)

    i -= 1
```

3. Sum of first 10 numbers

```
total = 0

for i in range(1, 11):

    total += i

print("Sum =", total)
```

3. Loop Control Statements

Loop control statements allow you to change the normal flow of a loop. Instead of executing every iteration, you can skip an iteration, stop early, or create a placeholder for future code.
They are extremely important in writing clean, efficient, and logical programs.

The three control statements are:

1. **break** – stops the loop immediately
2. **continue** – skips the current iteration
3. **pass** – does nothing (acts as a placeholder)

3.1 break — Stop the Loop Immediately

Definition:
break is used to *terminate the loop instantly*, even if the loop condition is still true.

When to use:

- You found what you were searching for
- You want to stop a loop early
- Exiting a menu or game when user chooses “quit”

Example — Stop at the first multiple of 7

```
for i in range(1, 20):

    if i % 7 == 0:
        print("First multiple of 7 is:", i)

        break
```

3.2 continue — Skip to the Next Iteration

Definition:
continue *skips the remaining code* in the current iteration and moves to the next.

When to use:

- Skip unwanted values
- Skip invalid input
- Skip even/odd numbers

Example — Print only odd numbers

```
for i in range(1, 11):

  if i % 2 == 0:

    continue

  print(i)

Output → 1 3 5 7 9
```

3.3 pass — Do Nothing (Placeholder)

Definition:
pass is used when a statement is required but you don’t want any action yet.

When to use:

- Writing functions/classes you will complete later
- Creating loop structures without logic yet
- Avoiding syntax errors

Example

```
for i in range(5):

  pass   # To be filled later
```

4. Pattern Programming

Patterns help students understand **loops**, **logic**, and **nested iterations**.
They build strong logical thinking and visual understanding.

4.1 Star Patterns

Pattern 1: Increasing Triangle

```
*

**

***

****
```

```
for i in range(1, 5):

    print("*" * i)
```

Pattern 2: Square Pattern

```
****

****

****

****
```

```
for i in range(4):

    print("*" * 4)
```

4.2 Pyramid Patterns

Center Pyramid

```
   *

  ***

 *****

*******
```

Logic

- Print spaces → decreasing
- Print stars → increasing odd numbers

```
rows = 4

for i in range(1, rows + 1):

   print(" " * (rows - i) + "*" * (2*i - 1))
```

4.3 Reverse Patterns

Reverse Triangle

```
****

***

**

*
```

```
for i in range(4, 0, -1):

   print("*" * i)
```

4.4 Number Patterns

Pattern: Increasing Numbers

```
1

12

123

1234
```

```
for i in range(1, 5):

   for j in range(1, i+1):

      print(j, end="")

   print()
```

Pattern: Same Number Repeated

```
1

22

333

4444
```

```
for i in range(1, 5):
    print(str(i) * i)
```

4.5 How Pattern Logic Works (Beginner-Friendly)

Every pattern requires two things:

1. Outer Loop → Number of rows

```
for row in range(rows):
```

2. Inner Loop → What to print (stars, spaces, numbers)

```
for col in range(row + 1):
```

Remember

- **Spaces first**
- **Stars or numbers next**
- Combine using print()

This builds strong foundational logic.

5. Mini Projects

Now we combine everything: conditional statements, loops, and control statements.

5.1 Pattern Generator (User Chooses Pattern)

```
choice = int(input("1: Triangle, 2: Reverse Triangle: "))

rows = int(input("Enter number of rows: "))

if choice == 1:

    for i in range(1, rows + 1):
        print("*" * i)

elif choice == 2:

    for i in range(rows, 0, -1):

        print("*" * i)

else:

    print("Invalid choice!")
```

5.2 Prime / Composite Checker

Logic

A number is **prime** if it has **no divisors except 1 and itself**.

```
num = int(input("Enter a number: "))

if num <= 1:

    print("Not Prime")

else:

    for i in range(2, num):

        if num % i == 0:

            print("Composite Number")

            break

    else:

        print("Prime Number")
```

5.3 Menu-Driven Program

```
while True:

    print("\nMENU")

    print("1. Add")

    print("2. Subtract")

    print("3. Exit")
    choice = input("Enter choice: ")
    if choice == "1":
        a = int(input("A: "))
        b = int(input("B: "))
        print("Result:", a + b)
    elif choice == "2":
        a = int(input("A: "))
        b = int(input("B: "))
        print("Result:", a - b)
    elif choice == "3":
        print("Goodbye!")
        break
    else:
        print("Invalid choice! Try again.")
```

5.4 Multiplication Table Generator

```
n = int(input("Enter a number: "))
for i in range(1, 11):
    print(f"{n} x {i} = {n * i}")
```