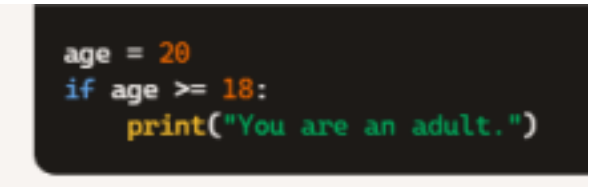
Control Structure:

Python Control Flow: if, else, and elif

✅ 1. `if` Statement

Used to execute a block of code only if a condition is true.

- The `if` statement is used to test a condition.
- If the condition evaluates to `True`, the indented block of code runs.
- It forms the foundation of decision-making in Python.
- Conditions are usually comparisons using operators like `==`, `>`, `<`, etc.
- Only one `if` block runs per condition check.



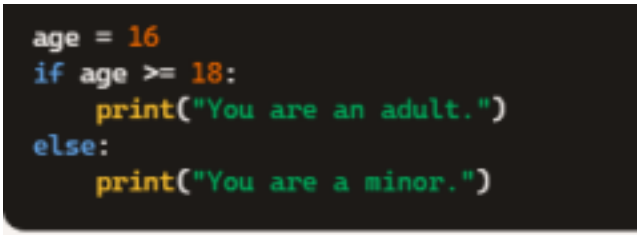
```
age = 20
if age >= 18:
    print("You are an adult.")
```

If the condition `age >= 18` is true, the message is printed.
Indentation is crucial in Python to define the block under `if`.

✅ 2. `else` Statement

Provides an alternative block of code if the `if` condition is false.

- The `else` block follows an `if` and runs when the `if` condition is `False`. ● It acts as a fallback or default action.
- No condition is attached to `else`; it simply catches all remaining cases.
- It must be paired with an `if` or `elif`.
- Useful for handling unexpected or catch-all scenarios.



```
age = 16
if age >= 18:
    print("You are an adult.")
else:
    print("You are a minor.")
```

If the `if` condition fails, the `else` block runs.

✅ 3. `elif` Statement (short for "else if")

Used to check multiple conditions sequentially.

- `elif` stands for "else if" and allows checking multiple conditions.
- It follows an `if` and precedes an optional `else`.
- Only the first `True` condition among `if`/`elif` blocks is executed. ● You can use multiple `elif` blocks to handle different cases.
- It makes code cleaner than nesting multiple `if` statements.

```
marks = 85

if marks >= 90:
    print("Grade: A")
elif marks >= 80:
    print("Grade: B")
elif marks >= 70:
    print("Grade: C")
else:
    print("Grade: F")
```

- Python checks each condition in order.
- The first condition that evaluates to `True` is executed, and the rest are skipped.

4. Nested `if` Statements

An `if` inside another `if` to check multiple layers of conditions.

```
if x > 0:
    if x < 10:
        print("Single-digit positive")
```

5. Ternary Conditional Operator

A one-line shorthand for `if-else`.

```
result = "Even" if x % 2 == 0 else "Odd"
```

Example :

```python
temperature = 30

if temperature > 35:
    print("It's very hot!")
elif temperature > 25:
    print("It's warm.")
elif temperature > 15:
    print("It's cool.")
else:
    print("It's cold.")
```

Simple for Loop in Python

- A `for` loop is used to iterate over a sequence like a list, tuple, or string.
- It executes a block of code for each item in the sequence. ● The loop variable takes the value of each item one by one. ● It's commonly used for processing collections or repeating actions. ● The loop ends when all items have been visited.

✅ Example

```python
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)
```

for Loop Using range()

- The `range()` function generates a sequence of numbers.
- It's often used with `for` loops to repeat actions a specific number of times. ● Syntax: `range(start, stop, step)` — start is inclusive, stop is exclusive. ● If only one argument is given, it's treated as the stop value.
- Useful for loops with numeric counters or index-based iteration.

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

while Loop in Python

- A `while` loop repeatedly executes a block of code as long as a condition is `True`.
- It checks the condition before each iteration.

- If the condition becomes `False`, the loop stops.
- Useful when the number of iterations is not known in advance.
  - Be cautious of infinite loops—ensure the condition will eventually become `False`.

✅ Example

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

This loop prints numbers from 1 to 5. The variable `count` increases each time, eventually breaking the loop.

continue Statement

- `continue` skips the current iteration and moves to the next one.
- It's used inside loops to bypass certain conditions.
- The loop doesn't terminate—it just skips that cycle.
- Often used to ignore unwanted values or cases.

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

break Statement

- `break` immediately exits the loop, regardless of the condition.
- It's used to stop the loop when a specific condition is met. ● No further iterations are executed after `break`.
- Common in search operations or early exits.
- Can be used in both `for` and `while` loops.

```
for i in range(1, 6):
    if i == 4:
        break
    print(i)
```

Output: 1, 2, 3 (stops at 4)

else with Loops

● The `else` block runs after the loop finishes normally (not via `break`).
● It's useful for post-loop actions like confirming completion. ● Often paired with search loops to detect if an item was found. ● If `break` is triggered, `else` is skipped.
● Works with both `for` and `while` loops.

Infinite loop

Here's a simple example of an **infinite loop** in Python using a `while` loop:

```python
while True:
    print("This will run forever!")
```

🧠 How It Works:

- `while True` creates a condition that is always `True`.
- The loop never ends unless you manually stop it (e.g., with `Ctrl+C` in a
- You can also break the loop using a condition:

```python
while True:
    user_input = input("Type 'exit' to stop: ")
    if user_input == "exit":
        break
```

This version runs infinitely until the user types `"exit"`.

Nested Loops

- A **nested loop** is a loop inside another loop.
- The **inner loop** runs completely every time the **outer loop** runs once.
- Useful for working with matrices, grids, or combinations. ● You can nest `for` loops, `while` loops, or mix them.
- Be mindful of performance — nested loops can grow exponentially.

Example: Nested `for` Loop

```python
for i in range(1, 4):          # Outer loop
    for j in range(1, 4):      # Inner loop
        print(f"i={i}, j={j}")
```

Output:

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

Example: Multiplication Table (3x3)

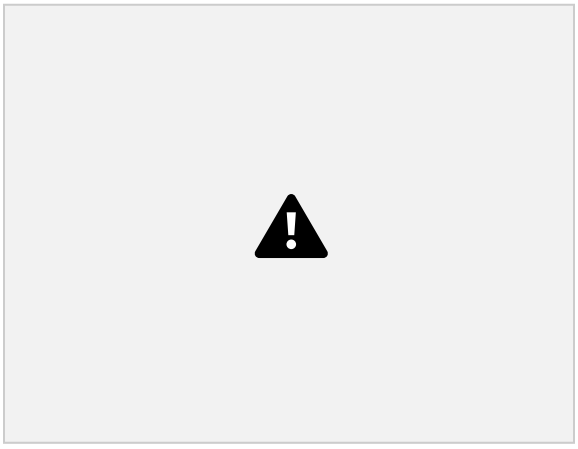
```python
for i in range(1, 4):
    for j in range(1, 4):
        print(i * j, end="\t")
    print()
```

Output:

```
1   2   3
2   4   6
3   6   9
```

Pattern Example

Pattern Example



Output

Practical Python Assignments

✅ Loops

1. **Sum of First N Natural Numbers**
    - Input: `n`
    - Use a `for` loop to calculate the sum.
    - Bonus: Try with a `while` loop too.
2. **Multiplication Table Generator**
    - Input: `number`

- Output: Print table from 1 to 10 using a `for` loop.

3. **Factorial Calculator**
   - Input: `n`
   - Use a `while` loop to compute `n!`

✅ Conditional Statements

4. **Grade Calculator**
   - Input: `marks`
   - Use `if-elif-else` to assign grades (A, B, C, D, F).
5. **Even or Odd Checker**
   - Input: `number`
   - Output: Print whether it's even or odd.
6. **Leap Year Checker**
   - Input: `year`
   - Output: Print whether it's a leap year.

⭐ Pattern Printing

7. **Right-Angled Triangle**
   - Input: `rows`
   - Output: Print `*` pattern.
8. **Pyramid Pattern**
   - Input: `rows`
   - Output: Print centered pyramid using nested loops.
9. **Number Triangle**

Output:
Code
1
12
123
1234

-

✅ Loop Manipulation

10. **Skip Multiples of 3**
    - Print numbers from 1 to 20, skip multiples of 3 using `continue`.
11. **Stop at First Negative**
    - Input: list of numbers

- Use `break` to stop when a negative number is found.

12. **Search with Loop-Else**
    - Input: list and target
    - Use `for-else` to print “Found” or “Not Found”.