Control Flow

Python Course · Module 4

ElnurBDa

Control Flow — Overview

Control flow decides **how a program behaves**:

- Conditional statements
- Logical operators
- Loops (`for`, `while`)
- Loop control (`break`, `continue`, `pass`)

This module teaches how Python makes decisions and repeats actions.

Conditional Statements

Basic structure:

```
if condition:
    block
```

A condition must evaluate to **True** or **False**.

Example:

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

———————————— [finished] ————————————

```
Adult
```

---

if / else

```
if condition:
    action_if_true
else:
    action_if_false
```

Example:

```
temperature = 10

if temperature > 15:
    print("Warm")
else:
    print("Cold")
```

———————— [finished] ————————

```
Cold
```

if / elif / else

Used when multiple conditions must be checked in order.

```
grade = "?"
score = 75

if score >= 90:
    grade = "A"
elif score >= 80:
    grade = "B"
elif score >= 70:
    grade = "C"
else:
    grade = "D"

print(grade)
```

———————— [finished] ————————

```
C
```

Logical Operators in Conditions

```
and   # both True
or    # one True
not   # invert
```

Examples:

```
age = 20
has_id = True

age >= 18 and has_id
age < 18 or has_id
not has_id
```

Nested Conditions

```
age = 25
citizen = True

if age >= 18:
    if citizen:
        print("Eligible")
    else:
        print("Not eligible")
```

———————————— [finished] ————————————

```
Eligible
```

Avoid deep nesting when possible.

---

for Loop — Basics

Iterates over a sequence:

```
for x in [1, 2, 3]:
    print(x)
```

———————————— [finished] ————————————

```
1
2
3
```

Strings:

```
for char in "Python":
    print(char)
```

———————————— [finished] ————————————

```
P
y
t
h
o
n
```

---

for Loop with range()

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

```
———————————————— [finished] ————————————————
```

```
0
1
2
3
4
```

Custom range:

```
for i in range(2, 10, 2):
    print(i)
```

```
———————————————— [finished] ————————————————
```

```
2
4
6
8
```

---

while Loop

Repeats while a condition is **True**.

```
count = 0
while count < 5:
    print(count)
    count += 1
```

———————————— [finished] ————————————

```
0
1
2
3
4
```

Loop Control — break

Stops the loop immediately.

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

———————————— [finished] ————————————

```
0
1
2
3
4
```

Loop Control — continue

Skips current iteration.

```
for i in range(5):
    if i % 2 == 0:
        continue
    print(i)
```

———————————— [finished] ————————————

```
1
3
```

Loop Control — pass

`pass` does nothing — placeholder.

```
for i in range(3):
    pass
```

──────────────── [finished] ────────────────

Loop Patterns & Comprehensions

Often we use loops to **build new lists** from old ones.

Classic pattern:

```
nums = [1, 2, 3, 4, 5]
evens = []

for n in nums:
    if n % 2 == 0:
        evens.append(n)

print("evens:", evens)
```

———————— [finished] ————————

```
evens: [2, 4]
```

Same logic using a **list comprehension** (from Module 3):

```
nums = [1, 2, 3, 4, 5]
evens = [n for n in nums if n % 2 == 0]
print("evens:", evens)
```

———————— [finished] ————————

```
evens: [2, 4]
```

Use whichever is **clearer** to you and your team; for multiple complex steps, prefer the classic loop.

Example

```python
total = 0

for i in range(1, 6):
    if i % 2 == 0:
        total += i
    else:
        continue

print("Sum of even numbers:", total)
```

———————————— [finished] ————————————

```
Sum of even numbers: 6
```

Mini Task

Write a script that:

1. Asks the user for a number
2. Prints all numbers from 1 to that number
3. Prints only **even numbers**
4. Skips odd numbers using `continue`
5. Stops entirely if number exceeds 100 using `break`