Conditional Statements

Making decisions based on conditions

Python Programming

What are Conditional Statements?

تسمح باتخاذ قرارات بناءً على شروط.

`if / else`

Two cases:
True → block A
False → block B

`if / elif / else`

Multiple cases:
Chains several conditions

`Nested if`

Decisions inside decisions

0. The if Statement (No else)

When the condition is **True**, the block runs. When **False**, Python simply skips it , no alternative block is needed.

Syntax

```
if condition:
    # run bloc only if True
# continues here always
```

How it works

True -> block executes, then continues
False -> block is skipped, continues

No else needed , Python moves on automatically.

Check Passing Grade

```
if Grade >= 10:
    print("Congrats!")
```

Negative Number Warning

```
if Num < 0:
    print("Negative!")
```

Word Contains Letter

```
if "a" in Word:
    print("has a")
```

0. The if Statement (No else)

When the condition is **True**, the block runs. When **False**, Python simply skips it , no alternative block is needed.

```
Algorithm equation_1er;
Var a, b, x: real;
begin
  read(a, b);
  if  (a <> 0)  then
      x := -b/a;
     write('the solution x =
',x);
  endif
end.
```

```
a = float(input("Enter a: "))
b = float(input("Enter b: "))

if a != 0:
    x = -b / a
    print("The solution x =", x)
```

1. The if / else Statement

Simple two-way decision

Syntax

```
if condition:
    # bloc 1 if True
else:
    # bloc 2 if False
```

Example — Pass or Fail

```
Grade = float(input(“enter your grade”))
if Grade >= 10:
    print('Passed')
else:
    print('Failed')
```

Even or Odd

```
if Num % 2 == 0:
    print("even")
else:
    print("odd")
```

Absolute Value

```
if Num >= 0:
    abs_val = Num
else:
    abs_val = -Num
print(abs_val)
```

1.2 Compound Conditions — and / or

Combine multiple conditions using logical operators

and

Both conditions must be True

```
if Num >= 1 and Num <= 100:
    print("in range")
```

or

At least one condition must be True

```
if day == "saturday" or day == "friday":
    print("weekend")
```

2. Multiple Cases — if / elif / else

When more than two outcomes are needed

Syntax

```
if condition1:
    block1
elif condition2:
    block2
elif condition3:
    block3
else:
    blockN
```

Example : Score → Grade

```
if Score >= 18:  print("Excellent")
elif Score >= 16: print("Very Good")
elif Score >= 14: print("Good")
elif Score >= 10: print("Acceptable")
else:       print("Fail")
```

Positive / Negative / Zero

```
if num > 0:  print("positive")
elif num < 0: print("negative")
else:         print("zero")
```

Age Group

```
if Age < 13:  print("Child")
elif Age <= 18: print("Teen")
elif Age < 60:  print("Adult")
else:         print("Senior")
```

3. Nested if Statements

Decisions inside decisions

A nested if places an if statement inside another if, allowing multi-level condition checks.

Syntax

```
if condition1:
    if condition2:
        code1
    else:
        code2
else:
    code3
```

Example : Maximum of 3 Numbers

```
if Num1 > Num2:
    if Num1 > Num3: print(Num1)
    else:           print(Num3)
else:
    if Num2 > Num3: print(Num2)
    else:           print(Num3)
```

Example : Age & Gender Check

```
if Age >= 18:
    if Gender == "female": print("Adult female")
    else:                  print("Adult male")
else: print("Minor")
```

Loops in Python

Repeat code efficiently without duplication

Python Programming

Why Do We Need Loops?

Loops repeat a block of code multiple times , without writing the same instructions again and again.

Without a Loop

```
print(1)
print(2)
print(3)
# ... up to 100
# 100 lines!
```

With a Loop

```
for i in range(1, 101):
    print(i)

# Just 2 lines!
```

I. The for Loop & range()

Iterate over sequences of numbers

range(stop)

Starts at 0, ends at stop−1

```
for i in range(11):
    print(i)  # 0..10
```

range(start, stop)

Starts at start, ends at stop−1

```
for i in range(1, 11):
    print(i)  # 1..10
```

range(start, stop, step)

step controls the increment

```
for i in range(1, 11, 2):
    print(i)  # 1,3,5,7,9
```

Example : Print even numbers below N:

```
for i in range(0, Num, 2):
    print(i)
```

for Loop — Iterating Over Strings

Print each character of a string

The for loop can directly iterate over characters in a string, or use range(len(text)) for index-based access.

Solution 1 : Direct Iteration

```
Text = input('enter a text')

for c in Text:
    print(c)
```

Solution 2 : Index-Based

```
Text = input('enter a text')

for i in range(len(Text)):
    print(Text[i])
```

II. The while Loop

Repeat as long as a condition is True

Syntax

```
while condition:
    # code runs while
    # condition is True
```

Key Points

- Condition checked before each iteration
- Must update a variable to avoid infinite loop
- Use when the number of iterations is unknown

Example : Countdown from N to 0 (for vs while)

```
# Using for
for i in range(Num, -1, -1):
    print(i)
```

```
# Using while
i = Num
while i >= 0:
    print(i)
    i = i-1
```

Python Programming

Modularity, Functions & Debugging

Write cleaner, reusable, and error-free code

I

The Problem

What happens when we repeat the same code?

Running the Same Instructions Multiple Times

تخيّل أننا نريد تحية 3 طلاب، ثم طباعة فاصل، ثم عرض درجاتهم. نقوم بكتابة نفس المقطع ثلاث مرات:

```
# Student 1
print("Hello, Alice!")
print("----------")
print("Score: 18")

# Student 2
print("Hello, Bob!")
print("----------")
print("Score: 15")

# Student 3
print("Hello, Sara!")
print("----------")
print("Score: 12")
```

The Solution : Functions!

الدالة تقوم بتجميع المقطع المتكرر في وحدة واحدة ذات اسم. نقوم بتعريفها مرة واحدة، ثم نستدعيها كلما احتجنا إليها.

```
def greet_student(name, score):
    print("Hello, " , name)
    print("----------")
    print("Score: " , score)


greet_student("Alice", 18)
greet_student("Bob",   15)
greet_student("Sara",  12)
```

Benefits ✓

- يتم تعريفها مرة واحدة، واستدعاؤها 3 مرات
- تعمل حتى مع 100 طالب
- سهلة القراءة

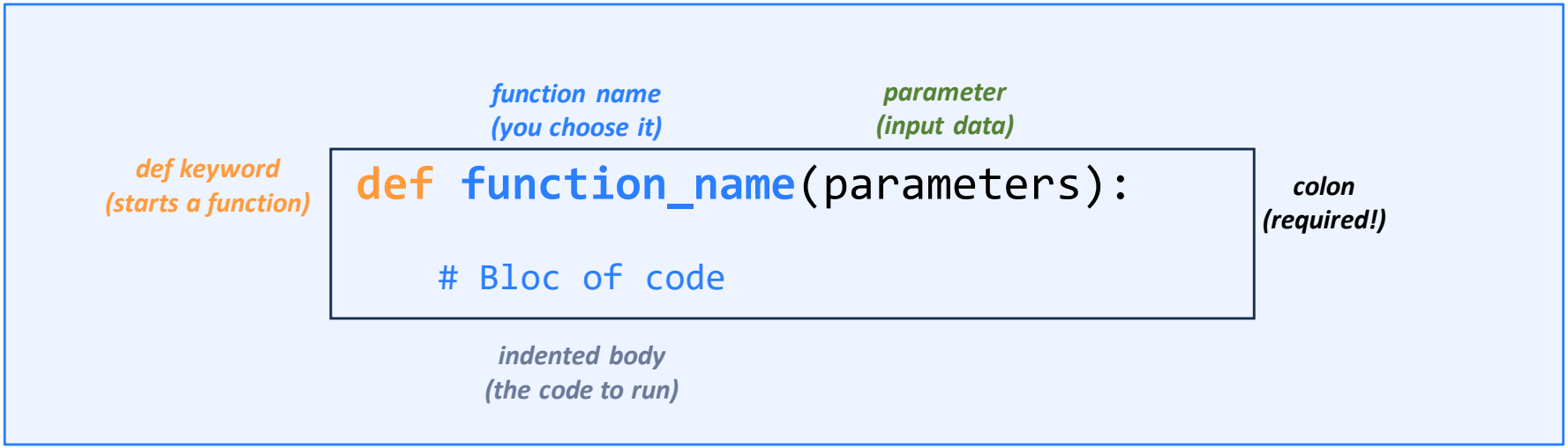
This is modularity!

II

Functions

Define once. Call anywhere. Reuse forever.

Defining a Function



Return Values

Getting a result back from a function

الكلمة المفتاحية **return** تقوم بإرجاع قيمة إلى المكان الذي تم فيه استدعاء الدالة. بدون **return**، تقوم الدالة فقط بتنفيذ بعض العمليات ولا تُرجِع أي قيمة.

Without return

```
def add(a, b):
    result = a + b
    print(result)

x = add(3, 5)
print(x)  # None ✗
```

With return

```
def add(a, b):
    result = a + b
    return result

x = add(3, 5)
print(x)  # 8 ✓
```

Return Values — Practical Examples

Using the result of a function in your program

Square of a Number

```
def square(n):
    return n * n

result = square(5)
print(result)  # 25
```

Maximum of Two Numbers

```
def maximum(a, b):
    if a > b:
        max=a
    else:
        max=b
    return max

print(maximum(8,3))
# 8
```