1- Statements: تعليمة برمجية

Python statements are instructions that the Python interpreter executes. A Python program consists of multiple statements, and these statements perform tasks such as assigning values to variables, controlling the flow of the program, performing loops, handling exceptions, and more.

Let's explore various Python statements with explanations and examples.

6.1. Conditional Statements (if, elif, else) تعليمة شرطية

Conditional statements allow you to control the flow of the program based on conditions. The **if** statement evaluates a condition, and if it is true, the corresponding block of code is executed. You can chain multiple conditions using **elif**, and provide a fallback using **else**.

Example:

python
```
age = 18

if age >= 18:
   print("You are an adult.")
elif age >= 13:
   print("You are a teenager.")
else:
   print("You are a child.")
```

Output:

```
You are an adult.
```

Explanation:

- The program checks the value of age. Since age is 18, the condition age >= 18 is true, so it prints "You are an adult.".

6.2. Looping Statements تعليمة التكرار

6.2.1. for Loop

A **for loop** is used to iterate over a sequence (such as a list, string, or range) and perform a block of code multiple times.

Example:

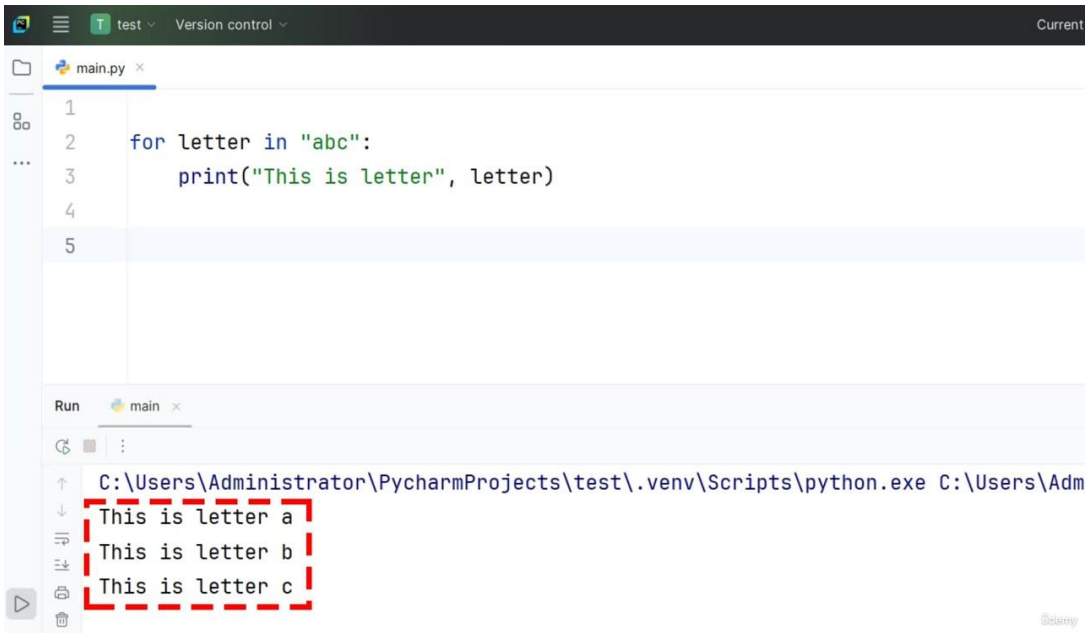
```
python
fruits = ["apple", "banana", "cherry"]

for fruit in fruits:
    print(fruit)
```

Output:

```
apple
banana
cherry
```

Explanation:

- The for loop iterates through the fruits list, and each fruit is printed on a new line.



6.2.2. while Loop

A **while loop** repeats a block of code as long as a given condition is true.

Example:

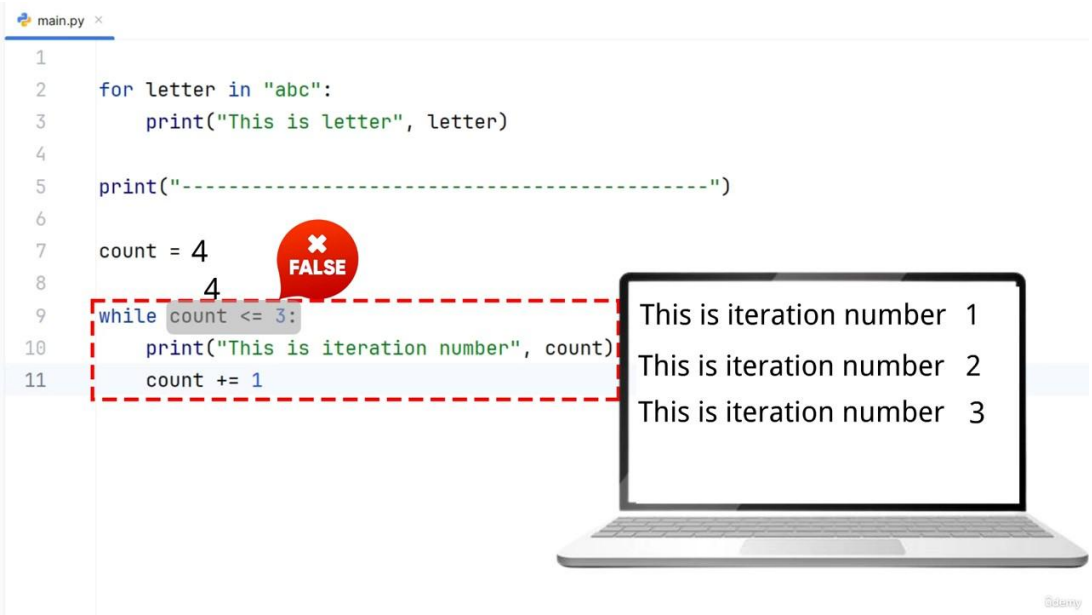
```
python
i = 1

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

Output:

```
1
2
3
4
5
```

Explanation:

- The while loop prints the value of i as long as i is less than or equal to 5. After each iteration, i is incremented by 1 using i += 1.



Example — Counting loop

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

Output:

1
2
3
4
5

Example — Loop through a string

```
word = "Python"

for letter in word:
    print(letter)
```

Output:

P
y
t
h

o
n

Example — Nested loops (حلقة داخل حلقة)

```
for i in range(2):       # outer loop
   for j in range(3):    # inner loop
      print(i, j)
```

Output:

```
0 0
0 1
0 2
1 0
1 1
1 2
```

6.3. break **and** continue **Statements**

6.3..1. break Statement

The **break statement** is used to exit a loop prematurely when a specific condition is met.

Example:

```
python
for i in range(1, 10):
   if i == 5:
      break
   print(i)
```

Output:

```
1
2
3
4
```

Explanation:

- The loop stops when i becomes 5 due to the break statement, so the numbers from 1 to 4 are printed.

6.3.2. continue Statement

The **continue statement** skips the current iteration of the loop and moves on to the next iteration.

Example:

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

Output:

1
2
4
5

Explanation:

- When i is 3, the continue statement is executed, skipping the print statement for that iteration. The numbers 1, 2, 4, and 5 are printed.

6.4. Function Definition (def) الدوال

What is a Function? (ما هي الدالة؟)

- A **function** in Python is a reusable block of code that performs a specific task.
- It is defined using the keyword **`def`**.
- Functions help **organize**, **reuse**, and **simplify** code.

Structure (البنية):

```
def function_name(parameters):
    # code block
    return result
```

Example:

python
def greet(name):
  print(f"Hello, {name}!")

greet("Alice")
greet("Bob")

Output:

Hello, Alice!
Hello, Bob!

Explanation:

- The greet() function takes a single argument name and prints a greeting message. It is called twice with the arguments "Alice" and "Bob".

Example 5 — Function with Default Parameter (وسيط افتراضي)

def greet(name="Guest"):
  print("Welcome,", name)

Usage:

greet("Sara")
greet()

Output:

Welcome, Sara
Welcome, Guest

7. return Statement

The **return statement** is used in a function to return a value to the caller and terminate the function.

Example:

```
python
def add(a, b):
    return a + b

result = add(3, 5)
print(result)
```

Output:

8

Explanation:

- The add() function takes two arguments, a and b, and returns their sum. The result of the function call is stored in the result variable and printed.

Example 6 — Function Returning Multiple Values (عدة نتائج)

```
def calculate(a, b):
    sum_ = a + b
    diff = a - b
    return sum_, diff
```

Usage:

```
s, d = calculate(10, 5)
print("Sum:", s)
print("Difference:", d)
```

Output:

Sum: 15
Difference: 5

8. pass Statement

The **pass statement** is a placeholder for future code. It does nothing and is used to define empty blocks of code.

Example:

python

```
def my_function():
    pass  # Code to be added later

class MyClass:
    pass  # Class definition to be added later
```

Output:

- There is no output since the pass statement does nothing, but it prevents syntax errors when you need to define an empty function or class.

11. import Statement

The **import statement** allows you to include modules and use their functions and variables.

Example:

```
python
import math

result = math.sqrt(16)
print(result)
```

Output:

```
4.0
```

Explanation:

- The import math statement imports the math module, allowing you to use its sqrt() function to calculate the square root of 16.

12. assert Statement

The **assert statement** is used for debugging purposes. It tests if a condition is true, and if it is not, it raises an AssertionError.

Example:

```
python
x = 10
assert x > 5, "x should be greater than 5"

y = 2
# assert y > 5, "y should be greater than 5"  # Uncommenting this will raise an AssertionError
```

Output:

- There is no output if the assertion is true. If the assertion is false, an AssertionError is raised.

13. List Comprehension

List comprehension provides a concise way to create lists based on existing lists or other iterables.

Example:

python
```
squares = [x ** 2 for x in range(5)]
print(squares)
```
Output:

[0, 1, 4, 9, 16]

Explanation:

- The list comprehension [x ** 2 for x in range(5)] generates a list of squares of numbers from 0 to 4.

14. with Statement (Context Manager)

The **with statement** is used for resource management, such as opening and closing files automatically, even if an error occurs.

Example:

python
Code
```
with open("example.txt", "w") as file:
   file.write("Hello, World!")
```
Output:

- The with statement automatically closes the file after writing "Hello, World!" to example.txt. There is no explicit need to call file.close().

Exercises:

1. **List (القوائم)**
```
fruits = ["apple", "banana", "cherry"]

# Check if an item exists
if "banana" in fruits:
   print("Yes, banana is in the list!")

# Check if list is empty
if not fruits:
   print("The list is empty")
else:
   print("The list has items")
```
Explanation:

- in checks if an element exists in the list.
- not fruits checks if the list is empty.

2. **Tuple (الصفوف)**
```
coordinates = (10, 20)
```

```
# Check a specific value
if coordinates[0] == 10:
   print("The x coordinate is 10")
```

Explanation:
Tuples are immutable, but we can still use if to check their contents.

5. **String (السلاسل النصية)**

```
name = "Python"

# Check if substring exists
if "th" in name:
   print("Found 'th' inside name")

# Check start of string
if name.startswith("Py"):
   print("Starts with Py")
```

Explanation:
Strings behave like sequences, so we use in, startswith()

Example — Function Using a Loop Inside

```
def print_list(items):

   for i in items:

       print(i)
```

Usage:

```
python

print_list(["apple", "banana", "cherry"])
```

Output:

```
apple

banana

cherry
```

Example 8 — Function with Conditional Logic

```
def check_even(number):
   if number % 2 == 0:
      return "Even"
```

```
    else:
        return "Odd"
```

Usage:

```
print(check_even(7))
```

Output:

```
Odd
```

Conclusion

Python statements are the fundamental building blocks of a Python program. They include assignment, control flow, looping, function definition, and exception handling, among others. Each type of statement serves a specific purpose and allows you to control the flow and behavior of your program.