3.4 FLOW OF EXECUTION IN A FUNCTION CALL

Let us now talk about how the control flows (*i.e.*, the flow of execution of statements) in case of a function call. You already know that a function is called (or invoked, or executed) by providing the function name, followed by the values being sent enclosed in parentheses. For instance, to invoke a function whose header looks like :

```
def sum (x, y) :
```

the *function call statement* may look like as shown below :

```
sum (a, b)
```

where *a*, *b* are the values being passed to the function *sum( )*.

Let us now see what happens when Python interpreter encounters a function call statement.

The *Flow of Execution* refers to the order in which statements are executed during a program run.

FLOW OF EXECUTION

The ***Flow of Execution*** refers to the order in which statements are executed during a program run.

Recall that a block is a piece of Python program text that is executed as a unit (denoted by line indentation). A **function body** is also a block. In Python, a block is executed in an **execution frame**.

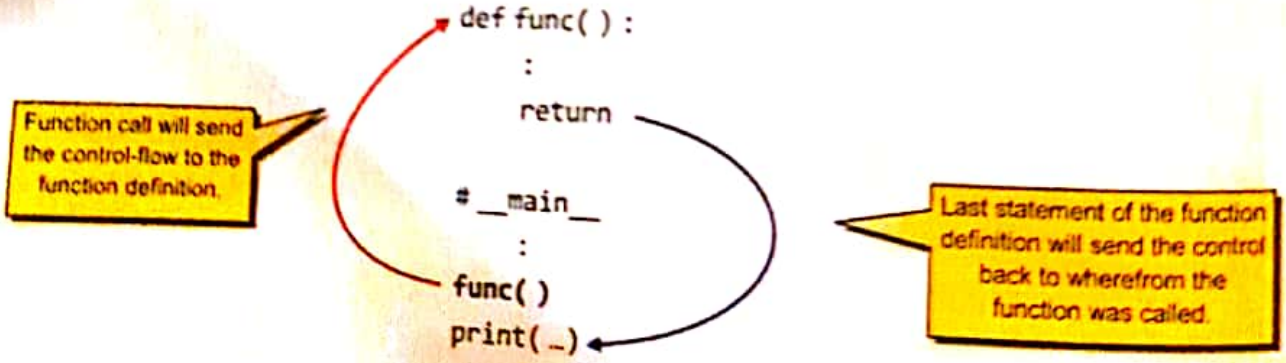
NOTE

The Flow of Execution refers to the order in which statements are executed during a program run.

An execution frame contains :

- some internal information (used for debugging)
- name of the function
- values passed to function
- variables created within function
- information about the next instruction to be executed.

Whenever a function call statement is encountered, an *execution frame* for the called function is created and the control (program control) is transferred to it. Within the function's execution frame, the statements in the function-body are executed, and with the *return statement* or the last statement of function body, the control returns to the statement wherefrom the function was called, *i.e.*, as :



Let us now see how all this is done with the help of an example. Consider the following program 3.1 code.

Scanned with CamScanner

Program 3.1 *Program to add two numbers through a function*

To see Working of a function in action

Scan QR Code

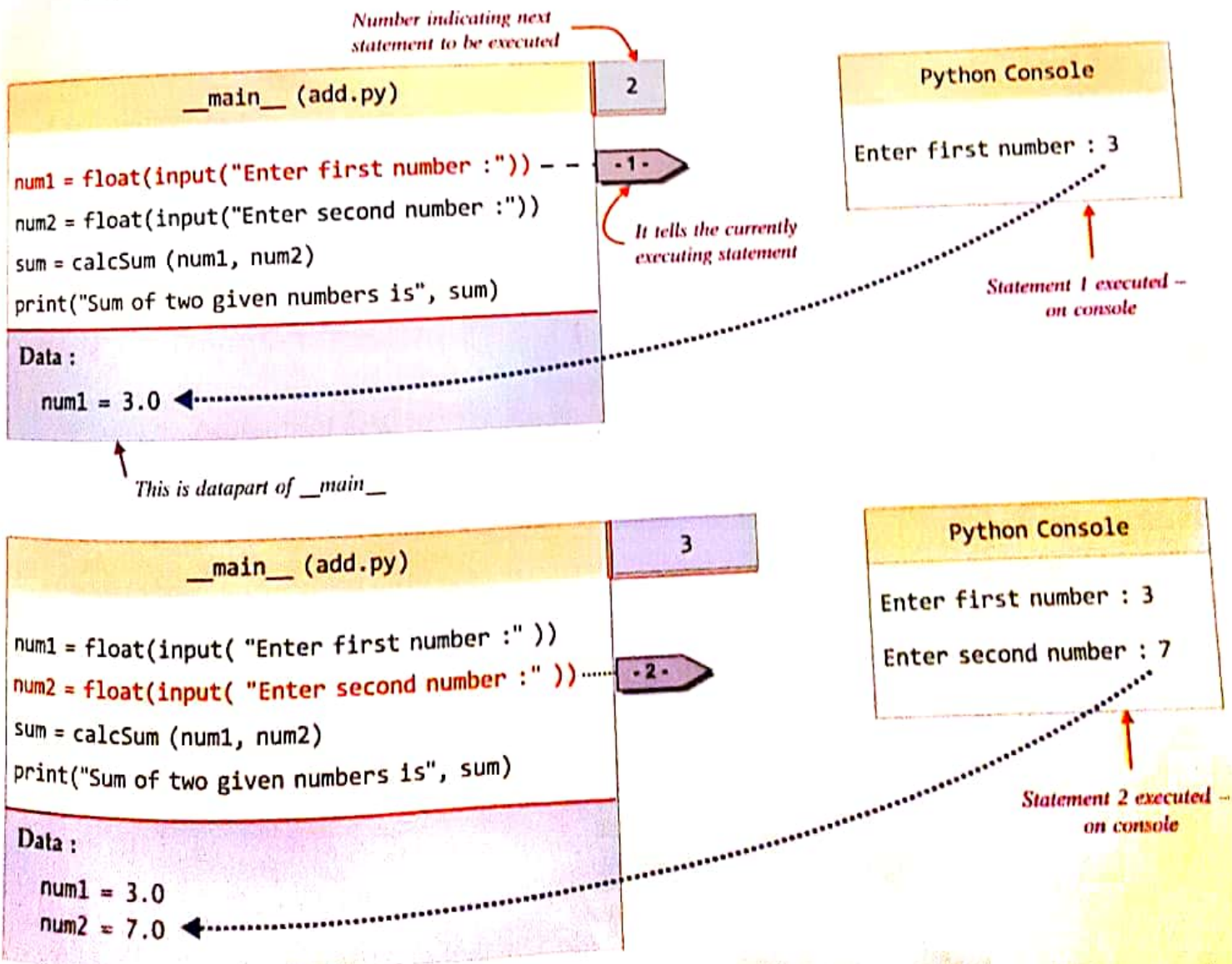
```
# program add.py to add two numbers through a function
def calcSum (x, y) :
    s = x + y                    # statement 1
    return s                     # statement 2

num1 = float(input( "Enter first number :" ) )      # 1 (statement 1)
num2 = float(input( "Enter second number :" ) )     # 2 (statement 2)
sum = calcSum(num1, num2)                           # 3 (statement 3)
print("Sum of two given numbers is", sum)           # 4 (statement 4)
```

Program execution begins with first statement of *__main__* segment. (def statements are also read but ignored until called. It will become clear to you in a few moments. Just read on.)

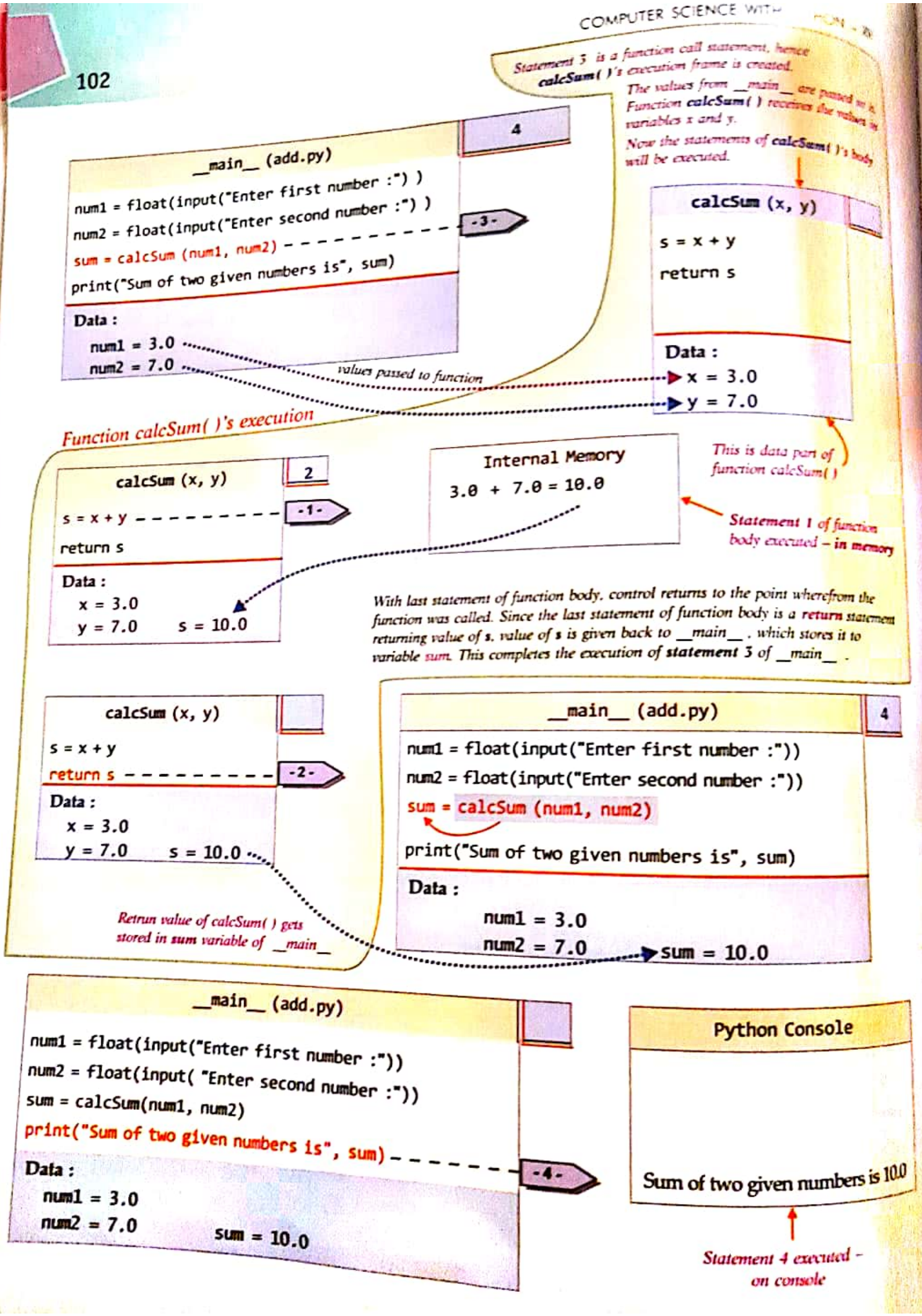
(Please note that in the following lines, we have put up some execution frames for understanding purposes only; these are not based on any standard diagram.)

NOTE

Program execution begins with first statement of *__main__* segment.



Scanned with CamScanner



Scanned with CamScanner

So we can say that for above program the statements were executed as :

main.1 → main.2 → main.3 → calcSum.1 → calcSum.2 → main.3 → main.4

(As you can see that we have shown a statement as its <segment-name>.<statement-number>)

Now that you know how functions are executed internally, let us discuss about *actual flow of execution*.

In a program, Python starts reading from line 1 downwards. Statements are executed one at a time, in order from top to bottom. While executing a program, Python follows these guidelines :

- Execution always begins at the first statement of the program.
- Comment lines (lines beginning with a #) are ignored, *i.e.*, not executed. All other non-blank lines are executed.
- If Python notices that it is a function definition, (**def statements**) then Python just executes the function header line to determine that it is proper function header and skips/ignores all lines in the function body.
- The statements inside a function-body are not executed until the function is called.
- In Python, a function can define another function inside it. But since the inner function definition is inside a function-body, the inner definition isn't executed until the outer function is called.
- When a code-line contains a *function-call*, Python first jumps to the function header line and then to the first line of the function body and starts executing it.
- A function ends with a **return** statement or the last statement of function body, whichever occurs earlier.
- If the called function returns a value *i.e.*, has a statement like return <variable/value/expression> (*e.g.*, **return a** or **return 22/7** or **return a + b** etc.) then the control will jump back to *the function call statement* and completes it (*e.g.*, if the returned value is to be assigned to variable or to be printed or to be compared or used in any type of expression etc. ; whole function call is replaced with the return value to complete the statement).
- If the called function does not return any value *i.e.*, the **return** statement has no variable or value or expression, then the control jumps back to the line following the function call statement.

If we give line number to each line in the program then flow of execution can be represented just through the line numbers, *e.g.*,

To see Flow of Execution in action

Scan QR Code

```
1.  # program add.py to add two numbers through a function
2.  def calcSum (x, y) :
3.          s = x + y              # statement 1
4.          return s               # statement 2
5.
6.  num1 = float(input("Enter first number :" ))      # 1 (statement 1)
7.  num2 = float(input("Enter second number :" ))     # 2 (statement 2)
8.  sum = calcSum (num1, num2)                        # 3 (statement 3)
9.  print("Sum of two given numbers is", sum)         # 4 (statement 4)
```

Scanned with CamScanner