Defining (Creating) a Function

Table of Contents

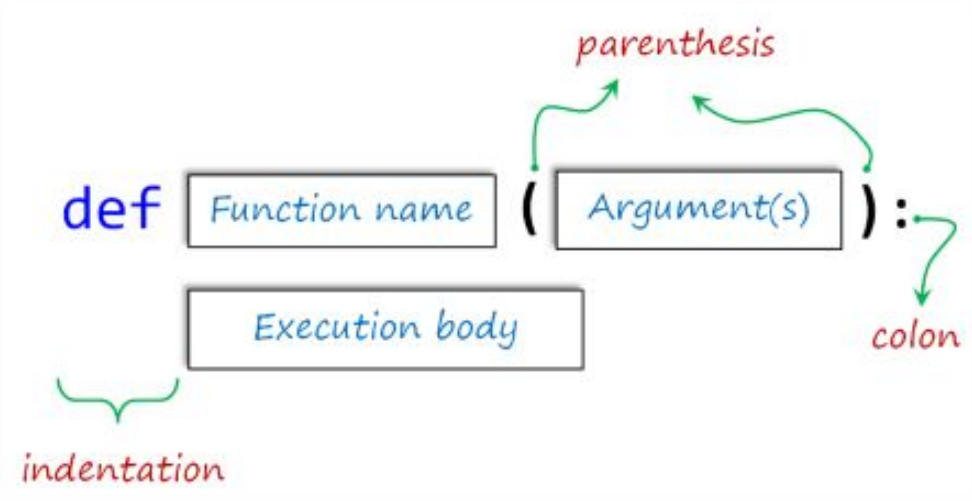
1 Introduction

How was the **pre-class** content?

Students, drag the icon!

Introduction (review)

- The keyword `def` introduces the name of the function.
- It must be followed by the function name and the parenthesized list of formal arguments.
- The statements that form the body of the function start at the next line and must be indented (leave four spaces).



2 Main Principles of 'Defining'

Main Principles of 'Defining' (review)

- The basic **formula syntax** of user-defined function is:

```
def function_name(arguments) :
    execution body
```

- Variables should be (traditionally) written in **lowercase** with **underscores** between words.
- Argument lists are optional, but the parentheses.
- A colon 👉 **:** follows the closing parenthesis.
- The codes must be indented under `def` statement.

Main Principles of 'Defining' (review)

- Let's give an example by leaving the parentheses empty.

```
1 def motto() :
2     print("Don't hesitate to reinvent yourself!")
3
4 motto()  # it takes no argument
```

What is the output? Try to figure out in your mind...

Main Principles of 'Defining' (review)

- Let's give an example by leaving the parentheses empty.

```
1 def motto() :
2     print("Don't hesitate to reinvent yourself!")
3
4 motto()  # it takes no argument
```

```
1 Don't hesitate to reinvent yourself!
```

Main Principles of 'Defining' (review)

- Let's grasp the matter with a pre-class example :

👇

```
1 def first_function(argument_1, argument_2) :
2     print(argument_1**2 + argument_2**2)
```

$$\texttt{argument\_1}^2 + \texttt{argument\_2}^2$$

sum of the squares of arguments

Main Principles of 'Defining' (review)

- Let's call and use `first_function`.

```
1  first_function(2, 3)  # here, the values (2 and 3) are
       allocated to the arguments
```

- In the example above, the values (2 and 3) are allocated to the arguments provided at the function call in parentheses.

Main Principles of 'Defining' (review)

- Let's call and use `first_function`.

```
1  first_function(2, 3)  # here, the values (2 and 3) are
       allocated to the arguments
```

```
1  13
```

- In the example above, the values (2 and 3) are allocated to the arguments provided at the function call in parentheses.

Main Principles of 'Defining' (review)

- Let's define the multiplying function `multiply(a, b)`.

```
1  def multiply(a, b) :
2      print(a * b)
3
4  multiply(3, 5)
5  multiply(-1, 2.5)
6  multiply('amazing ', 3)  # it's really amazing, right?
```

What is the output? Try to figure out in your mind...

Students, write your response!

Main Principles of 'Defining'

- Let's define the multiplying function `multiply(a, b)`.

```
1  def multiply(a, b) :
2      print(a * b)
3
4  multiply(3, 5)
5  multiply(-1, 2.5)
6  multiply('amazing ', 3)  # it's really amazing, right?
```

```
1  15
2  -2.5
3  amazing amazing amazing
```

Main Principles of 'Defining'

- **Task** :
  - Define a function named `add` to sum two numbers and print the result.

Main Principles of 'Defining' (review)

- **The code can be like :**

```
1  def add(a, b):
2      print(a + b)
3
4  add(-3, 5)
5
```

Output

```
2
```

Main Principles of 'Defining'

- **Task** :
  - Define a function named `calculator` to calculate four math operations with two numbers and print the result.

```
1
2 calculator(88, 22, "+")
3
```

Output

```
110
```

Main Principles of 'Defining'

- **The code might be like :**

```
1  def calculator(x, y, o):
2      if o == "+" :
3          print(x + y)
4      elif o == "-" :
5          print(x - y)
6      elif o == "*" :
7          print(x * y)
8      elif o == "/" :
9          print(x / y)
10     else : print("enter valid arguments!")
11
```

3 Execution of a Function

Execution of a Function

- The functions you have seen so far did not return any types or values but executed some actions.
- In order to use the output and data types generated by the functions in our next program flow, we need to define our function with the keyword `return`.

Execution of a Function (review)

- Let's take a look at the example below:

- `print`
- `return`

```
def multiply_1(a, b) :
    print(a * b)  # it prints something

multiply_1(10, 5)
```

Execution of a Function (review)

- Let's take a look at the example below:

- print
- return

```
def multiply_1(a, b) :
    print(a * b)  # it prints something

multiply_1(10, 5)
```

```
50
```

Execution of a Function (review)

- Let's take a look at the example below:

- print
- return

```
def multiply_2(a, b) :
    return(a * b)  # returns any numeric
data type value

print(multiply_2(10, 5))
```

Execution of a Function (review)

- Let's take a look at the example below:

- print
- return

```
def multiply_2(a, b) :
    return(a * b)  # returns any numeric
data type value

print(multiply_2(10, 5))
```

```
50
```

Execution of a Function (review)

- The first function just prints some data what you passed into. The second one generates a numeric type value. If you check their types you will see :

```
1  print(type(multiply_1(10, 5)))
2  print(type(multiply_2(10, 5)))
```

Execution of a Function (review)

- The first function just prints some data what you passed into. The second one generates a numeric type value. If you check their types you will see :

```
1  print(type(multiply_1(10, 5)))
2  print(type(multiply_2(10, 5)))
```

```
1  50
2  <class 'NoneType'>
3  <class 'int'>
```

Execution of a Function (review)

- We can't use the result of the first function since it is `NoneType` data. But, the second one is **integer data** that we can use it in the future when we need it.

```
1  shadow_var = print("It can't be assigned to any variable")
2  print(shadow_var)  # NoneType value can't be used
```

Execution of a Function (review)

- We can't use the result of the first function since it is `NoneType` data. But, the second one is **integer data** that we can use it in the future when we need it.

```
1  shadow_var = print("It can't be assigned to any variable")
2  print(shadow_var)  # NoneType value can't be used
```

```
1  It can't be assigned to any variable
2  None
```

Main Principles of 'Defining'

- **Task** :
  - Define a function named `calculator` to calculate four math operations with two numbers and `return` the result.

```
1
2  print(calculator(-12, 2, "+"))
3
```

Output

```
-10
```

Main Principles of 'Defining'

- **The code might be like :**

```
1  def calculator(x, y, o):
2      if o == "+" :
3          return(x + y)
4      elif o == "-" :
5          return(x - y)
6      elif o == "*" :
7          return(x * y)
8      elif o == "/" :
9          return(x / y)
10     else : return ("enter valid arguments!")
11
```

Main Principles of 'Defining'

- **Task** :
  - Define a function named `absolute_value` to calculate and `return` absolute value of the entered number.

```
1 print(absolute_value(3.3))
2 print(absolute_value(-4))
3
```

Output

```
3.3
4
```

Students, write your response!

Main Principles of 'Defining'

- **The code might be like :**

```
1  def absolute_value(num):
2      """This function returns the absolute
3      value of the entered number"""
4
5      if num >= 0:
6          return num
7      else:
8          return -num
9
10 print(absolute_value.__doc__)
11
```

By the way, we can display the **docstring** of this function

Output

```
This function returns the absolute
    value of the entered number
```

THANKS!

Any questions?

You can find me at:

-
- andy@clarusway.com