



Acquaintance with Functions



Table of Contents



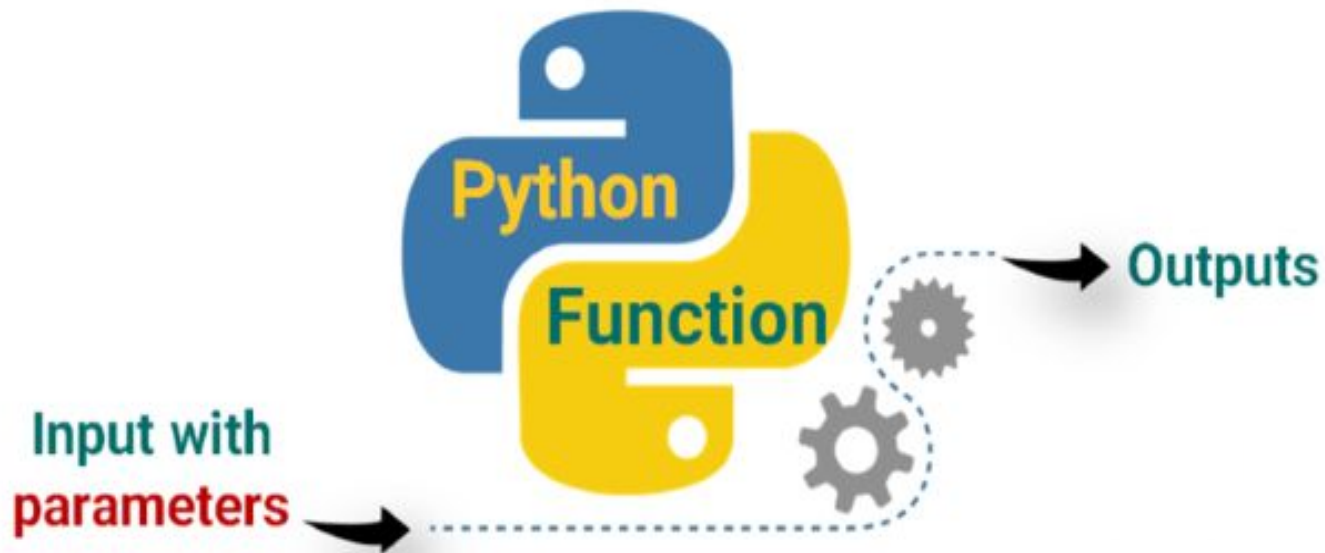
- ▶ Introduction
- ▶ Calling a Function
- ▶ Built-in Functions



1

Introduction to Functions

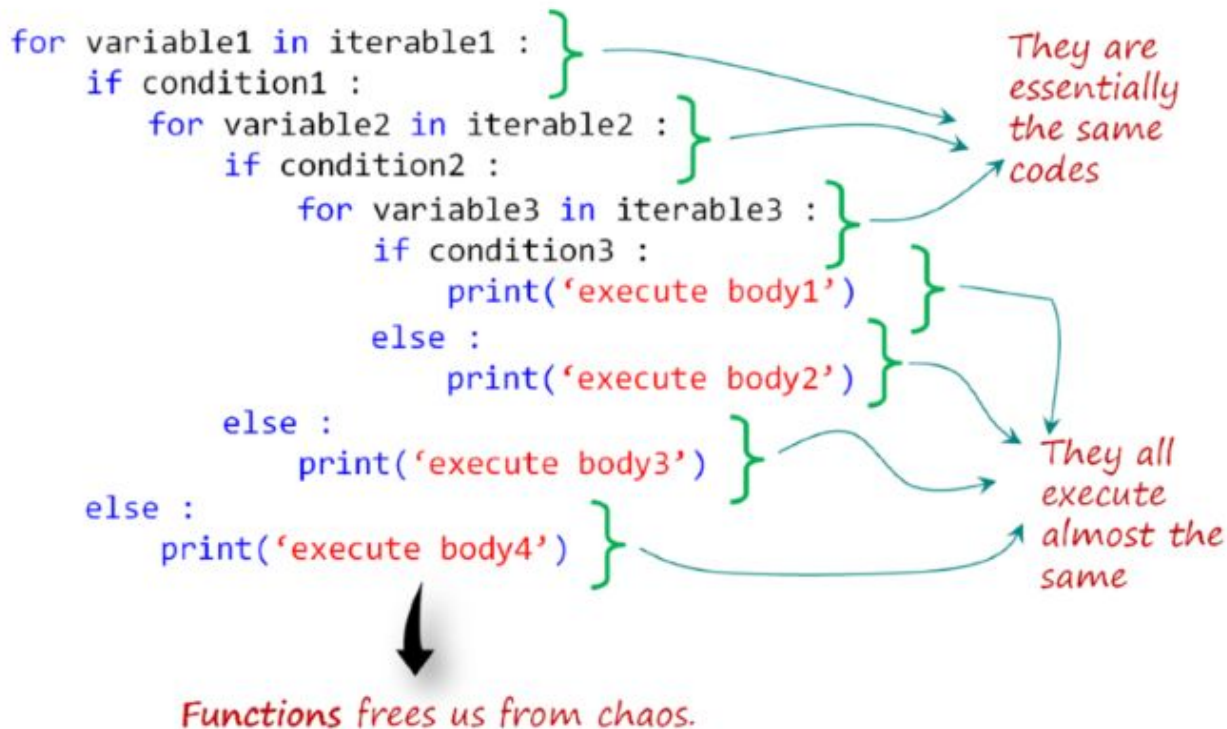
Introduction





Introduction (review)

- Functions free us from chaos.

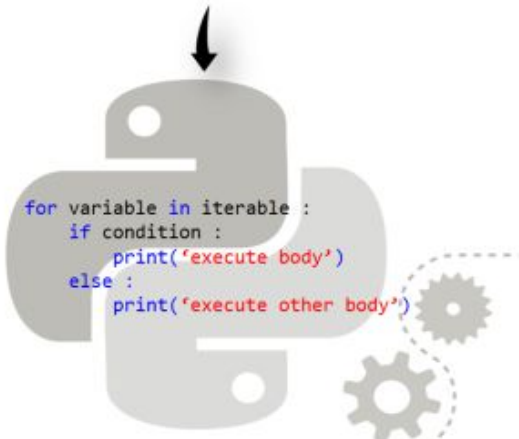




Introduction (review)

```
for variable in iterable :  
    if condition :  
        print('execute body')  
    else :  
        print('execute other body')
```

You can choose a *piece of code* to convert into a function



You can *create* a function which does what you want

```
my_function(iterable)
```

You can *call and use* your function whenever and wherever you want



2

Calling a Function

Calling a Function Means Using It(review)



- ▶ Reading a function is very easy in Python.

```
multiply(2, 5)
```

name of
the function

arguments of
the function

Calling a Function Means Using It(review)



```
1 a = 3
2 b = 5
3
4 multiply(3, 5)
4 multiply(a, b)
```

```
1 15
```



Calling `print()` Function (review)



- Take a look at the example 

```
1 print('Say: I love you!')  
2 print()  
3 print('me too', 2019)
```

```
1 Say: I love you!  
2  
3 me too 2019
```



3

Built-in Functions



Built-in Functions (review)

- ▶ The number of built-in functions :

In the latest version
Python 3.11



71



Built-in Functions (review)

- So far we have learned 

```
print(), int(), list(), input(), range()
```

- Some of them return bool type 

```
all(iterable), any(iterable), callable(object)
```



Built-in Functions (review)

- ▶ Some of them help you convert data types 

```
bool(), float(), int(), str()
```

- ▶ For creating and processing the collection types. 

```
dict(), list(), tuple(), set(), len(), zip(),  
filter(function, iterable), enumerate(iterable)
```



Built-in Functions (review)

- ▶ Some others tackle numbers. 

```
max(), min(), sum(), round()
```

- ▶ The others are built for special purposes. 

```
map(function, iterable, ...), eval(expression[,  
globals[, locals[]]), sorted(iterable), open(),  
dir([object]), help([object])
```



Defining (Creating) a Function



Table of Contents



- ▶ Main Principles of 'Defining'
- ▶ Execution of a Function



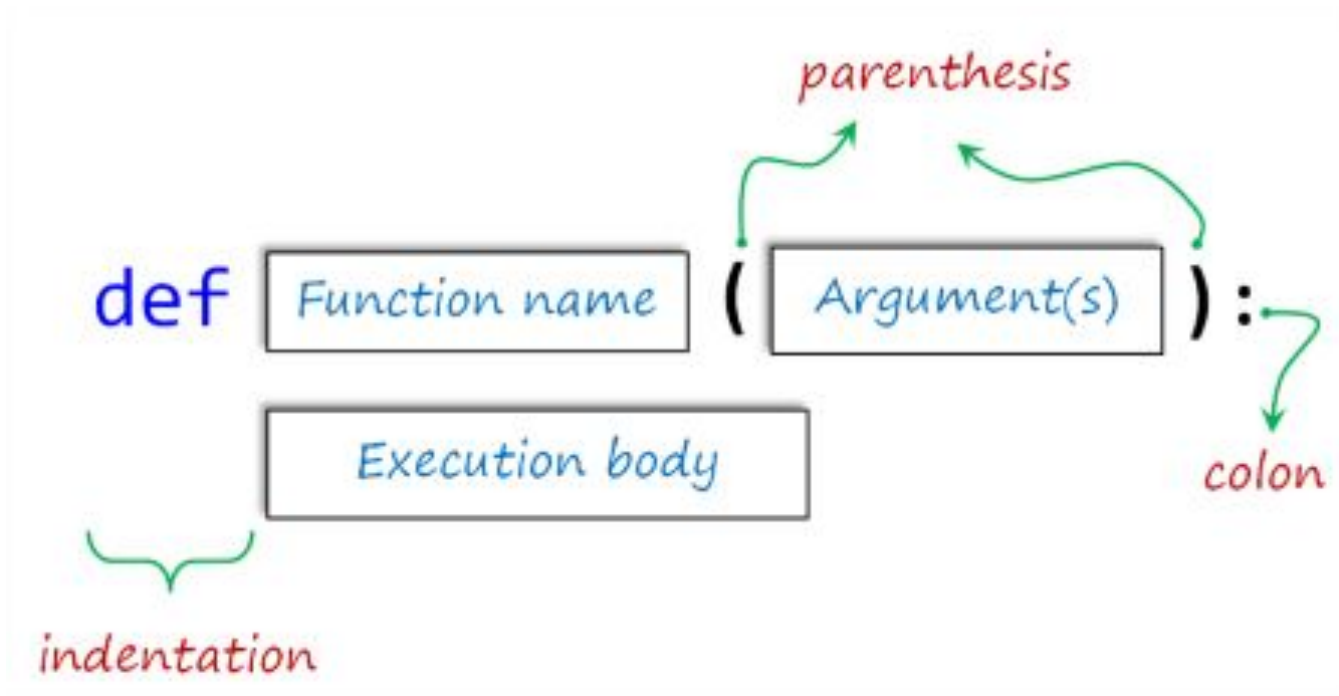
1

Main Principles of 'Defining'



Main Principles of 'Defining' (review)

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





Main Principles of 'Defining' (review)

- ▶ Defining a simple function 

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

$$\text{argument_1}^2 + \text{argument_2}^2$$



▶ 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
```



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
```



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...



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' (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'

▶ **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.
- Warn user in case of wrong entry : `"Enter valid arguments"`

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

Output

```
110
```



Main Principles of 'Defining'

- **The code might be like :**

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



2

Execution of a Function



Execution of a Function (review)

- ▶ The result of a function :

- `print`
- `return`



```
def multiply_1(a, b) :  
    print(a * b)  # it prints something  
  
multiply_1(10, 5)
```




Execution of a Function (review)

- ▶ The result of a function :

- `print`
- `return`



```
def multiply_1(a, b) :  
    print(a * b) # it prints something  
  
multiply_1(10, 5)
```

50



Execution of a Function (review)

- ▶ The result of a function :

- `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)

- ▶ The result of a function :

- `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)

- Compare the usage options :

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



Execution of a Function (review)

- ▶ The outputs are :

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

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



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
```


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
```