Python Plus

- Functions
- Modules & Packages
- Errors & Exceptions Handling

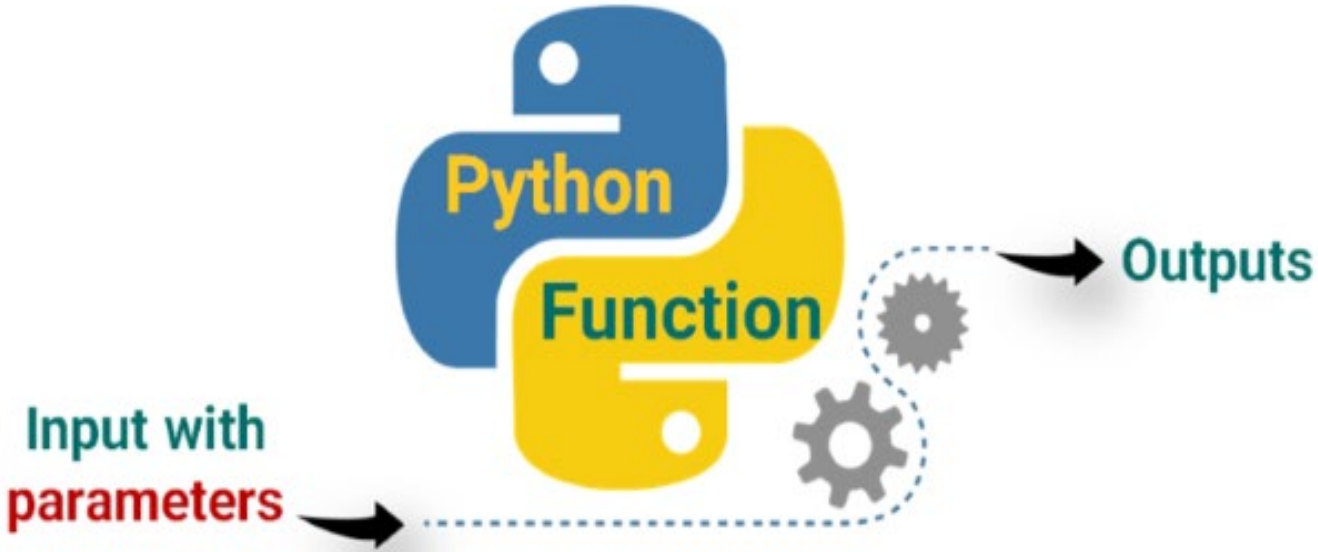
Acquaintance with Functions

Table of Contents

- Introduction
- Calling a Function
- Built-in Functions

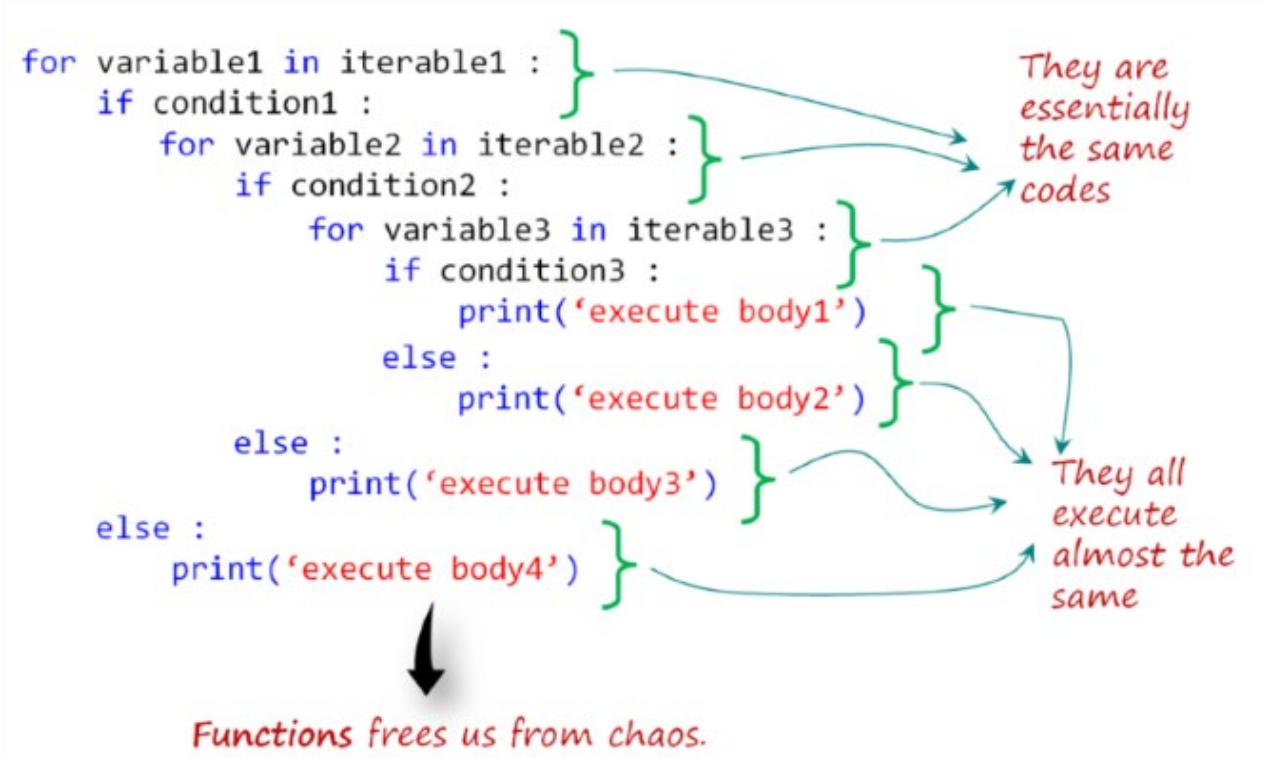
1 Introduction to Functions

Introduction



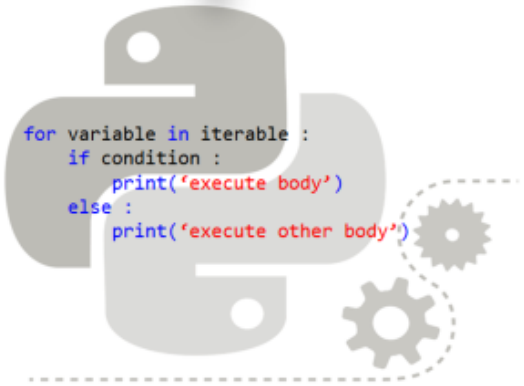
Introduction (review)

- Functions free us from chaos.



Introduction (review)

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




```
my_function(iterable)
```

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

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

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

2 Calling a Function

Calling `print()` Function (review)

- Calling `print()` function :

```
print("Say : I love you!")
```

name of the function

argument of the function

Calling `print()` Function (review)

- Take a look at this pre-class example

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

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()
```

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])
```

Built-in Functions

- Leaking snow water in your ear.

Built-in Functions

- `filter(function, iterable)`.

filter() Parameters

`filter()` method takes two parameters:

- **function** - function that tests if elements of an iterable return true or false
  If None, the function defaults to Identity function - which returns false if any elements are false
- **iterable** - iterable which is to be filtered, could be sets, lists, tuples, or containers of any iterators

Built-in Functions

- `filter(function, iterable)`.

```
1  listA = ["susan", "tom", False, 0, "0"]
2
3  filtered_list = filter(None, listA)
4
5  print("The filtered elements are : ")
6  for i in filtered_list:
7      print(i)
8
```

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

Built-in Functions

- **`filter(function, iterable)`.**

```
1  listA = ["susan", "tom", False, 0, "0"]
2
3  filtered_list = filter(None, listA)
4
5  print("The filtered elements are : ")
6  for i in filtered_list:
7      print(i)
8
```

With **`filter()`** function as **`None`**, the function defaults to Identity function, and each element in **`listA`** is checked if it's **`True`**.

Output

```
The filtered elements are :
susan
tom
0
```

Built-in Functions

- `enumerate(iterable, start=0)`.

enumerate() Parameters

`enumerate()` method takes two parameters:

- **iterable** - a sequence, an iterator, or objects that supports iteration
- **start** (optional) - `enumerate()` starts counting from this number. If `start` is omitted, `0` is taken as `start`.

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

Built-in Functions

- `enumerate(iterable, start=0)`.

```
1  grocery = ['bread', 'water', 'olive']
2  enum_grocery = enumerate(grocery)
3
4  print(type(enum_grocery))
5
6  print(list(enum_grocery))
7
8  enum_grocery = enumerate(grocery, 10)
9  print(list(enum_grocery))
10
```

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

Built-in Functions

- `enumerate(iterable, start=0)`.

```
1  grocery = ['bread', 'water', 'olive']
2  enum_grocery = enumerate(grocery)
3
4  print(type(enum_grocery))
5
6  print(list(enum_grocery))
7
8  enum_grocery = enumerate(grocery, 10)
9  print(list(enum_grocery))
```

Output

```
<class 'enumerate'>
[(0, 'bread'), (1, 'water'), (2, 'olive')]
[(10, 'bread'), (11, 'water'), (12, 'olive')]
```

Built-in Functions

- `max(iterable), min(iterable).`

```
1  number = [-222, 0, 16, 5, 10, 6]
2  largest_number = max(number)
3  smallest_number = min(number)
4
5  print("The largest number is:", largest_number)
6  print("The smallest number is:", smallest_number)
7
```

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

Built-in Functions

- `max(iterable), min(iterable).`

```
1  number = [-222, 0, 16, 5, 10, 6]
2  largest_number = max(number)
3  smallest_number = min(number)
4
5  print("The largest number is:", largest_number)
6  print("The smallest number is:", smallest_number)
7
```

Output

```
The largest number is: 16
The smallest number is: -222
```

Built-in Functions

- `sum(iterable, start)`.

```
1  numbers = [2.5, 30, 4, -15]
2
3  numbers_sum = sum(numbers)
4  print(numbers_sum)
5
6  numbers_sum = sum(numbers, 20)
7  print(numbers_sum)
8
```

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

sum() Parameters

- **iterable** - iterable (list, tuple, dict, etc). The items of the iterable should be numbers.
- **start** (optional) - this value is added to the sum of items of the iterable. The default value of `start` is 0 (if omitted)

Built-in Functions

- `sum(iterable)`.

```
1  numbers = [2.5, 30, 4, -15]
2
3  numbers_sum = sum(numbers)
4  print(numbers_sum)
5
6  numbers_sum = sum(numbers, 20)
7  print(numbers_sum)
8
```

Output

```
21.5
41.5
```

Defining (Creating) a Function

Table of Contents

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

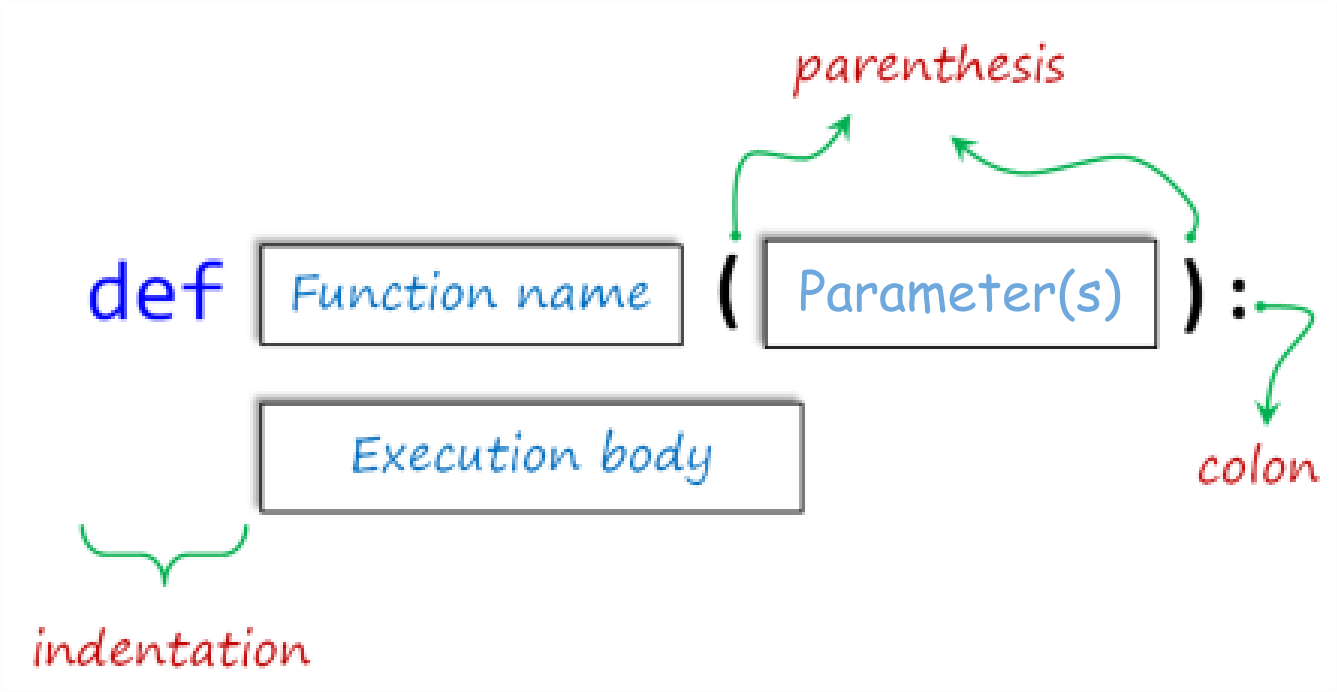
1 Main Principles of 'Defining'

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

Students, drag the icon!

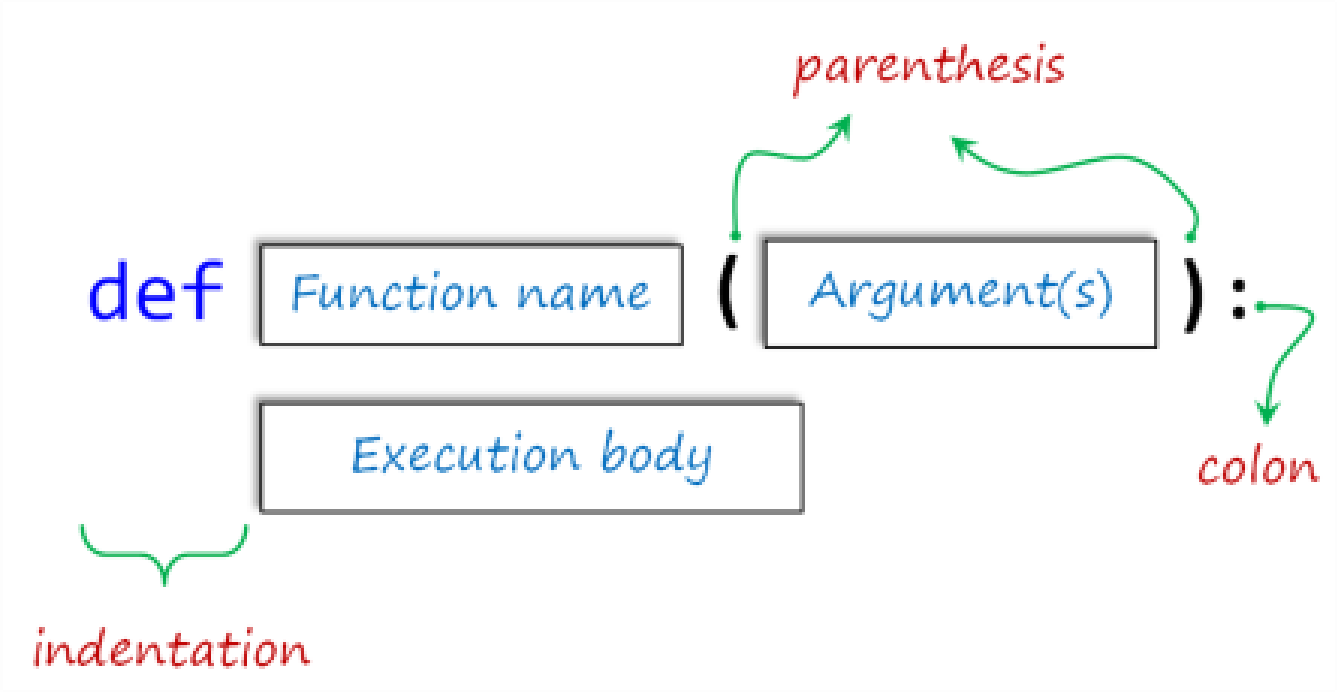
Main Principles of 'Defining' (review)

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



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)
```

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

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 :**

```
def calculator(x, y, opr):
    if opr == "+" :
        print(x + y)
    elif opr == "-" :
        print(x - y)
    elif opr == "*" :
        print(x * y)
    elif opr == "/" :
        print(x / y)
    else :
        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.
  - You can add docstring for an explanation.

```
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
```

THANKS!

End of the Lesson

(Defining a Function)

next Lesson

The Matter of Arguments

click above