Python Plus

- Functions
- Modules & Packages
- Working with Files
- Errors & Exceptions Handling

Acquaintance with Functions

CLARUSWAY©
WAY TO REINVENT YOURSELF

Table of Contents

- Introduction
- Calling a Function
- Built-in Functions

1 Introduction to Functions

CLARUSWAY©
WAY TO REINVENT YOURSELF

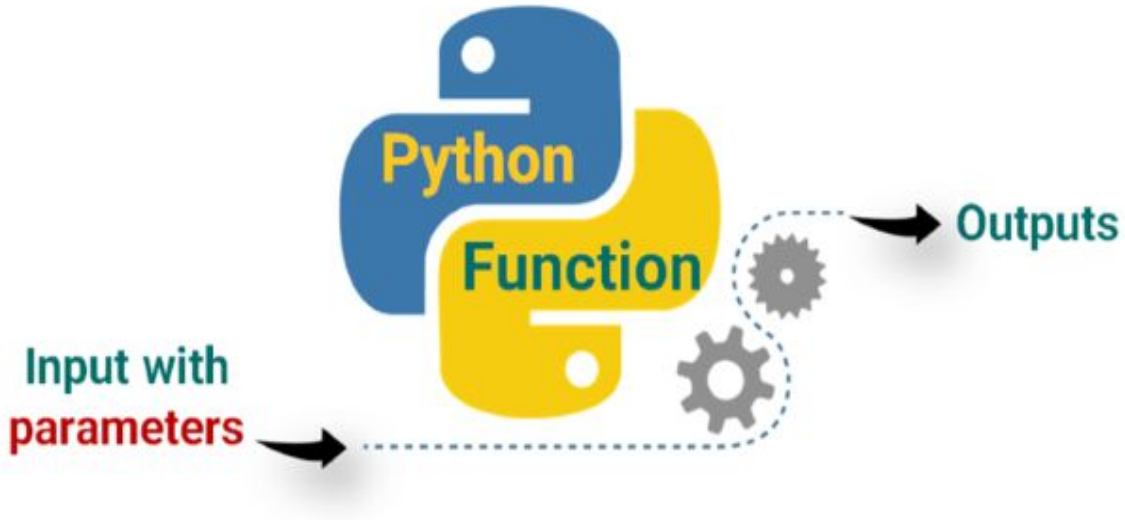
What do you know about functions in Python?

Type at least 3 things...

Students, write your response!

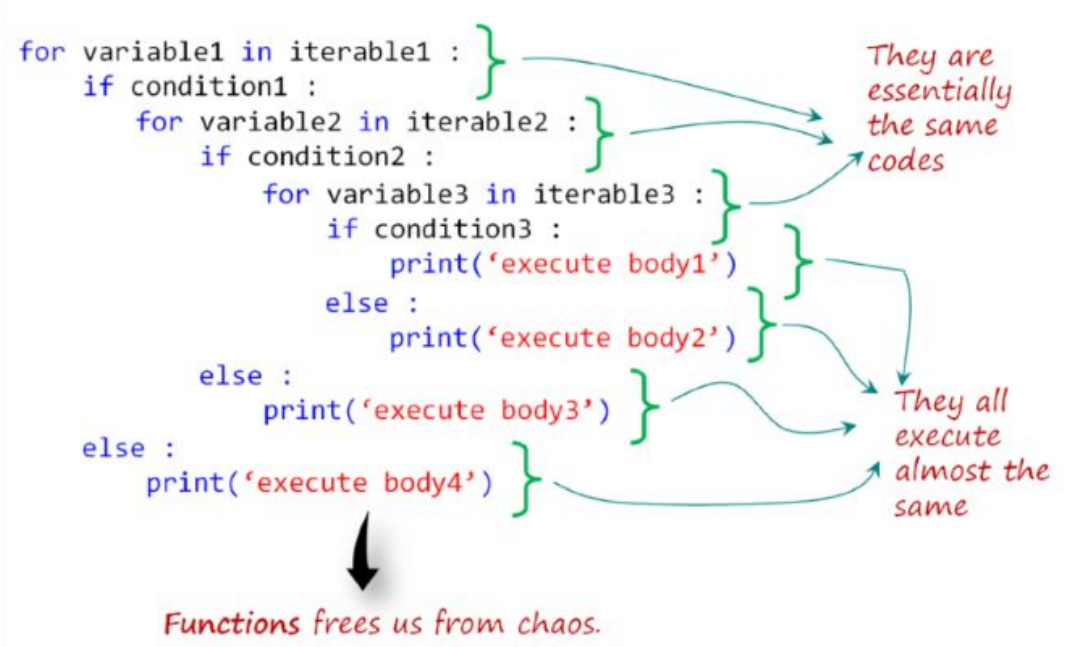
Pear Deck Interactive Slide
Do not remove this bar

Introduction

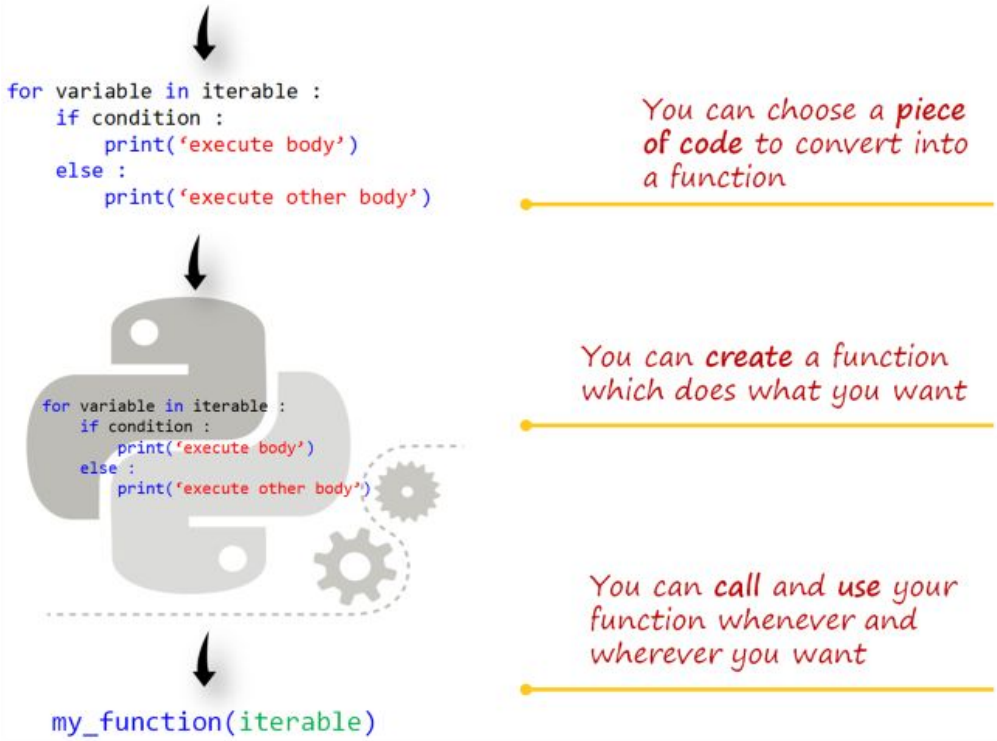


Introduction (review)

- Functions free us from chaos.



Introduction (review)



2 Calling a Function

CLARUSWAY©
WAY TO REINVENT YOURSELF

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

CLARUSWAY©
WAY TO REINVENT YOURSELF

Calling a Function Means Using It(review)

- Calling a function :

👇

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


Calling a Function Means Using It(review)

- Calling a function :

👇

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

```
1  15
```

Calling a Function Means Using It(review)

- Calling a function .

👇

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


Calling a Function Means Using It(review)

-

👇

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

```
1  15
```

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 3.9.1

69

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])
```

As mentioned in the **pre-class** content, I took a look at the ***built-in functions*** in the official Python docs.

True

False

Students choose an option

Pear Deck Interactive Slide
Do not remove this bar

Built-in Functions

- Leaking snow water in your ear.

Built-in Functions

- `all()` function.

```
1  names = ["susan", "tom", "False"]
2  mood = ["happy", "sad", 0]
3  empty = {}
4
5  print(all(names), all(mood), all(empty), sep="\n")
6
```

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

Students, write your response!

Built-in Functions

- `all()` function.

`all()` method returns:

Truth table for all()

When	Return Value
All values are true	True
All values are false	False
One value is true (others are false)	False
One value is false (others are true)	False
Empty Iterable	True

Built-in Functions

- `all()` function.

`all()` method returns:

- **True** - If all elements in an iterable are true
- **False** - If any element in an iterable is false

```
1  names = ["susan", "tom", "False"]
2  mood = ["happy", "sad", 0]
3  empty = {}
4
5  print(all(names), all(mood), all(empty), sep="\n")
6
```

Output

```
True
False
True
```


Built-in Functions

- `any()` function.

```
1  listA = ["susan", "tom", False]
2  listB = [None, (), 0]
3  empty = {}
4
5  print(any(listA), any(listB), any(empty), sep="\n")
6
```

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

Built-in Functions

- any() function

Condition	Return Value
All values are true	True
All values are false	False
One value is true (others are false)	True
One value is false (others are true)	True
Empty Iterable	False

```
True
False
False
```


Built-in Functions

- any() function.

The `any()` function returns a boolean value:

- `True` if at least one element of an iterable is true
- `False` if all elements are false or if an iterable is empty

```
1  listA = ["susan", "tom", False]
2  listB = [None, (), 0]
3  empty = {}
4
5  print(any(listA), any(listB), any(empty), sep="\n")
6
```

Output

```
True
False
False
```

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
```

Output

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

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

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...

Students, write your response!

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
```

Built-in Functions

- **round(numbers, ndigits)**.

```
1  print(round(12))
2  print(round(10.8))
3  print(round(3.665, 2))
4  print(round(3.675, 2))
5
```

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

round() Parameters

The `round()` function takes two parameters:

- **number** - the number to be rounded
- **ndigits (optional)** - number up to which the given number is rounded; defaults to 0

Built-in Functions

- **round(numbers, ndigits)**.

```
1  print(round(12))
2  print(round(10.8))
3  print(round(3.665, 2))
4  print(round(3.675, 2))
5
```

Output

```
12
11
3.67
3.67
```

Built-in Functions

- `round(numbers, ndigits)`.

```
1 print(round(12))
2 print(round(10.8))
3 print(round(3.665, 2))
4 print(round(3.675, 2))
5
```

Homework

```
3.674999999999999822364316059974953532218933105468 75
```

```
12
11
3.67
3.67
```


THANKS!

End of the Lesson

(Acquaintance with Func)

next Lesson

Defining a Function

click above