# Acquaintance with Functions

# Table of Contents

1

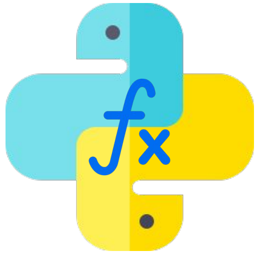
## Introduction to Functions

## What do you know about functions in Python?

Type at least 3 things...

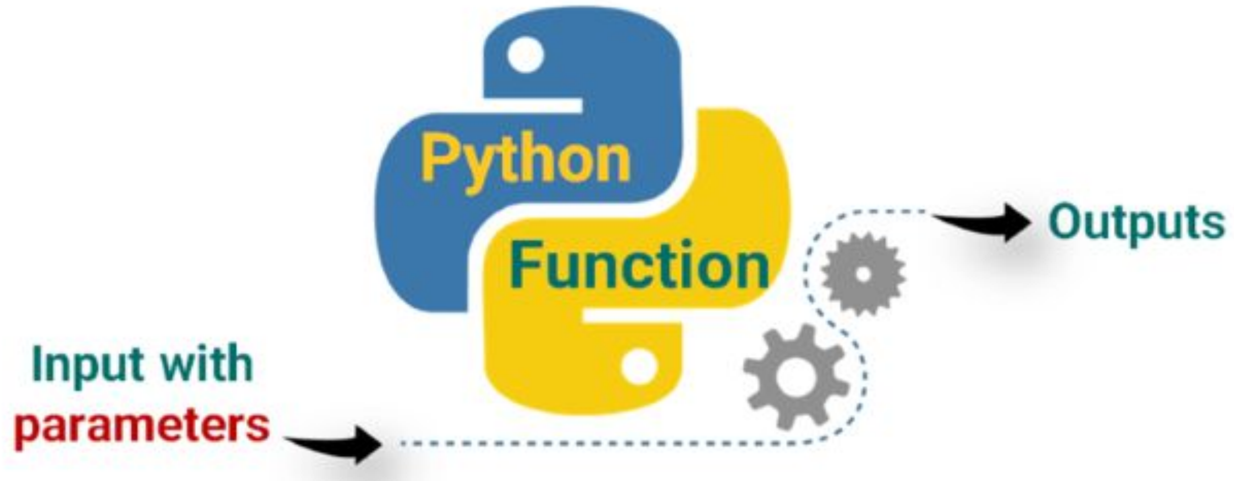
# Introduction

- Basically, a **function** is a block of code that executes some logic for you, e.g. *prints* a text, *deletes* some data or *square* a number. In other words, a function is **a piece of code that only runs when it is called**.
- Functions in Python provide organized, reusable and modular code to perform a set of specific actions. Functions simplify the coding process, prevent redundant logic, and make the code easier to follow.



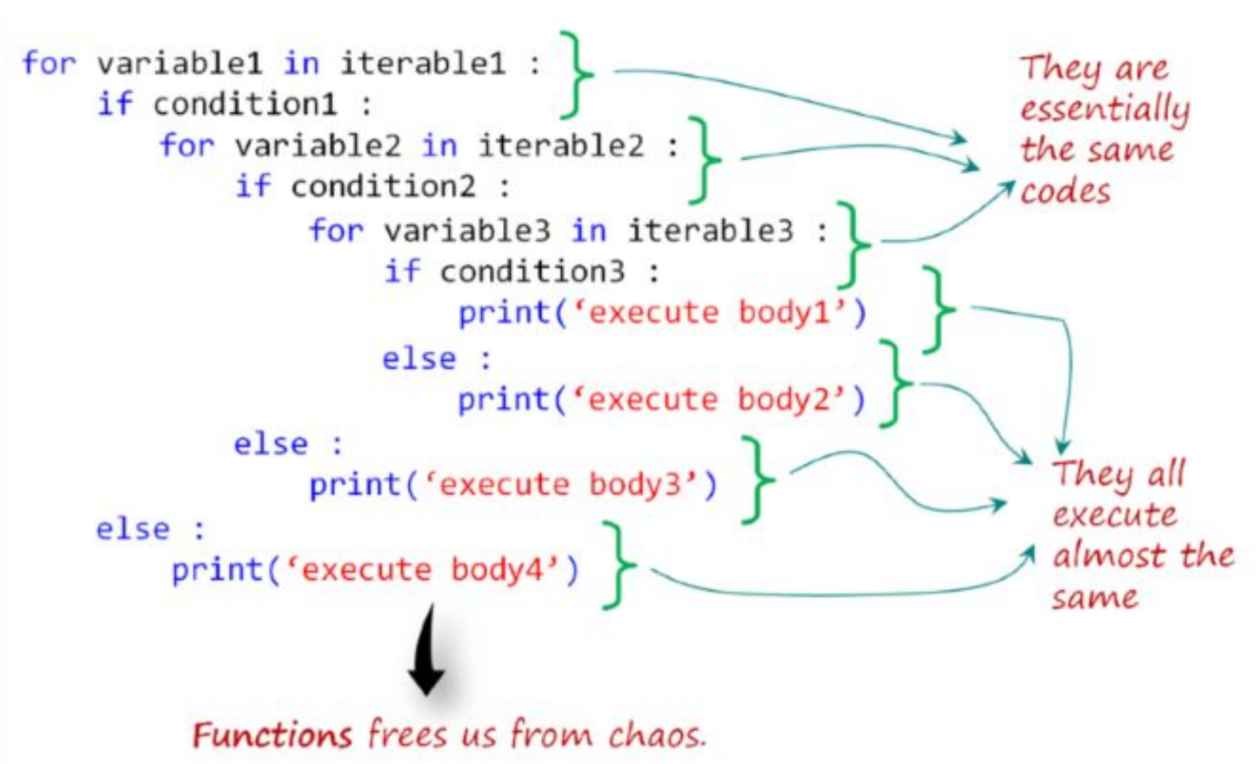
# Introduction (review)

- You can enter or input data, known as **arguments**, into a function and it returns/outputs something good that you want.



# Introduction (review)

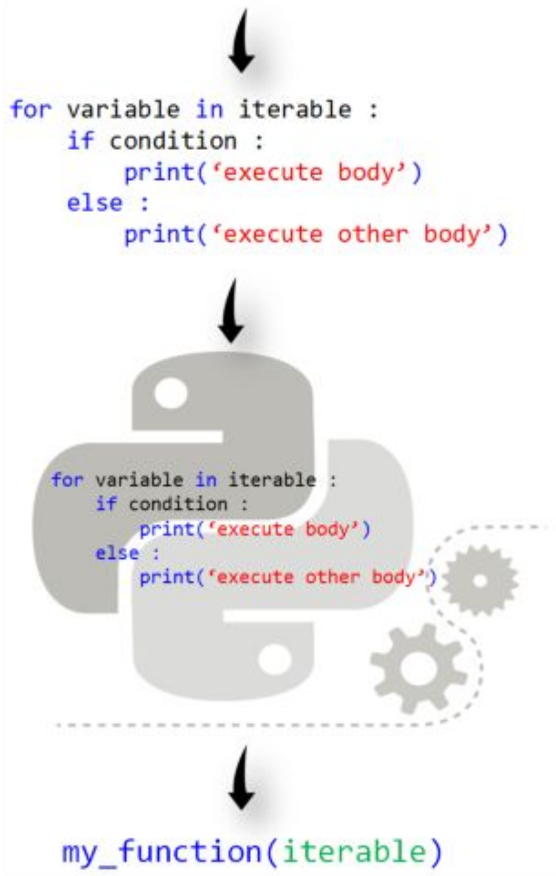
- Functions free us from chaos.



```
for variable1 in iterable1 :
    if condition1 :
        for variable2 in iterable2 :
            if condition2 :
                for variable3 in iterable3 :
                    if condition3 :
                        print('execute body1')
                    else :
                        print('execute body2')
            else :
                print('execute body3')
    else :
        print('execute body4')
```

They are essentially the same codes
They all execute almost the same
Functions frees us from chaos.

# Introduction (review)



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

```
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

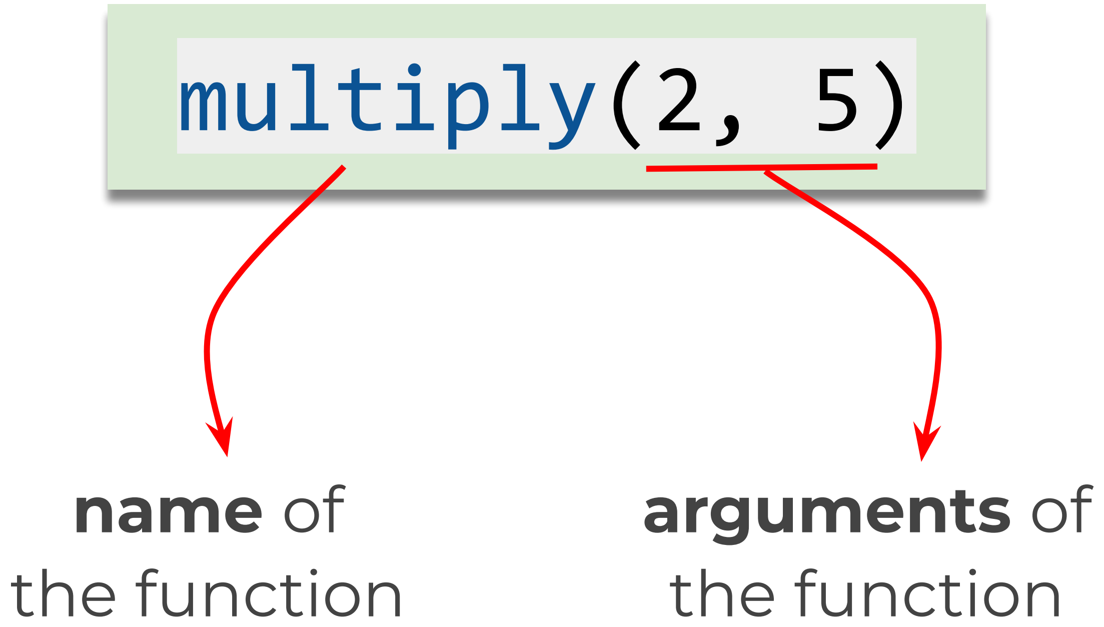
CLARUSWAY©
WAY TO REINVENT YOURSELF

2

## Calling a Function

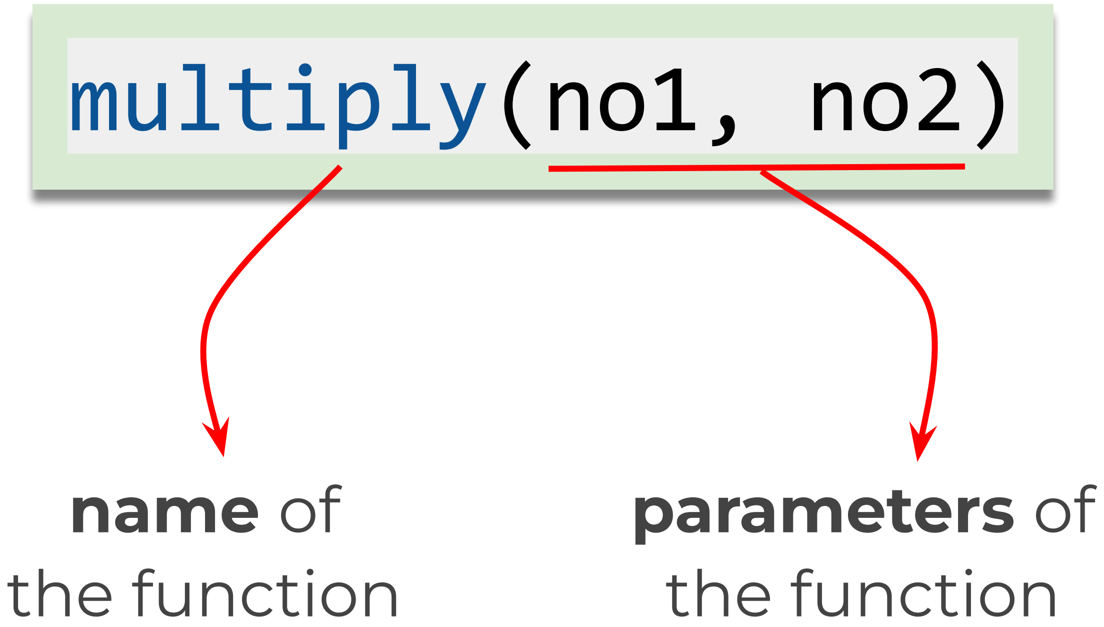
## Calling a Function Means Using It(review)

- Reading a function is very easy in Python.



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

# Calling a Function Means Using It(review)



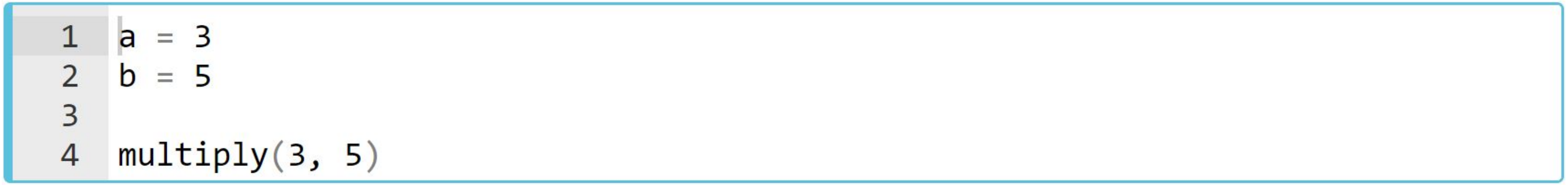
```
multiply(no1, no2)
```

CLARUSWAY©
WAY TO REINVENT YOURSELF

## Calling a Function Means Using It(review)

- If you want to multiply two numbers, you can just write the name of that function and the numbers (arguments) inside the parenthesis.

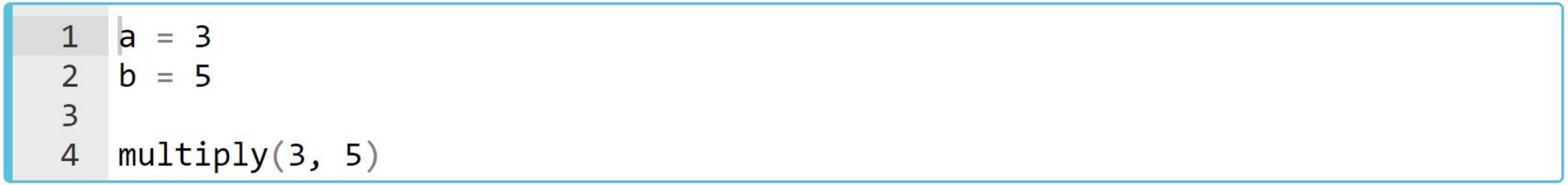
👇



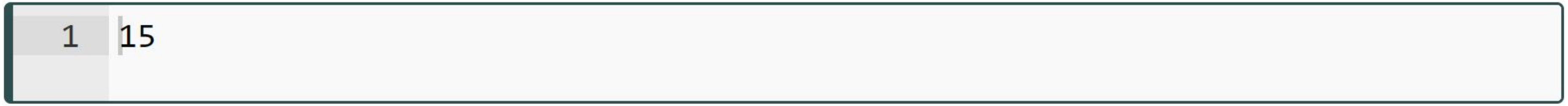
```python
a = 3
b = 5

multiply(3, 5)
```

## Calling a Function Means Using It(review)

- If you want to multiply two numbers, you can just write the name of that function and the numbers (arguments) inside the parenthesis.



```
a = 3
b = 5

multiply(3, 5)
```

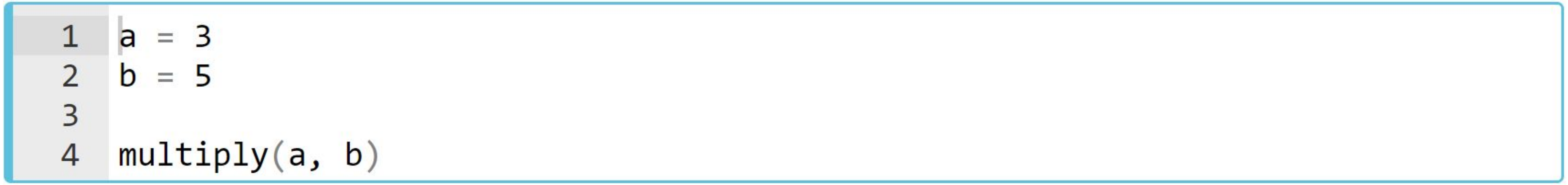


```
15
```

CLARUSWAY©
WAY TO REINVENT YOURSELF

## Calling a Function Means Using It(review)

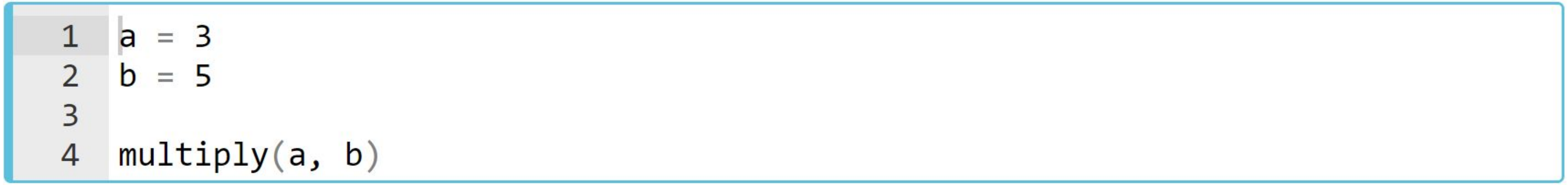
- If you want to multiply two numbers, you can just write the name of that function and the numbers (arguments) inside the parenthesis.



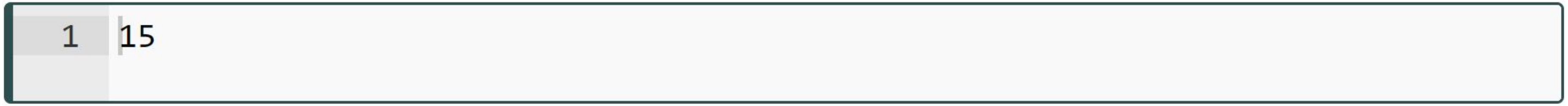
```
a = 3
b = 5

multiply(a, b)
```

# Calling a Function Means Using It(review)

- If you want to multiply two numbers, you can just write the name of that function and the numbers (arguments) inside the parenthesis.



```
a = 3
b = 5

multiply(a, b)
```



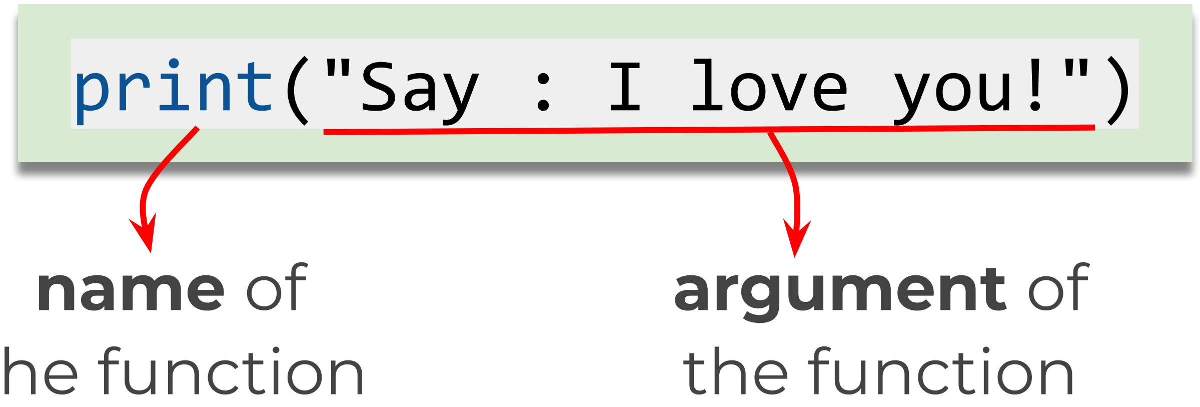
```
15
```

CLARUSWAY©
WAY TO REINVENT YOURSELF

## Calling `print()` Function (review)

- What we do is solely ;
  - writing its name and
  - adding parentheses after it

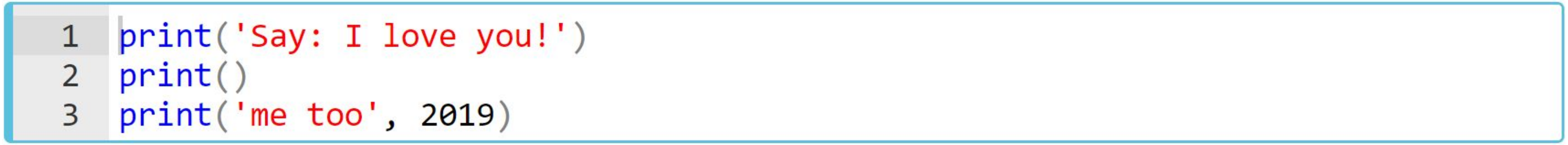
to call the `print()` function in your code.



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

# Calling `print()` Function (review)

- Take a look at this pre-class example 👇



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

## Calling `print()` Function (review)

- Take a look at the example 👇

```python
print('Say: I love you!')
print()
print('me too', 2019)
```

```
Say: I love you!

me too 2019
```

## 3 Built-in Functions

## Built-in Functions (review)

- If you are considering a function which may do something that you want, it probably exists. You just need to be aware of its existence.
- There are a range of functions and types built into the Python interpreter, so they are always usable.

In the latest version Python 3.9,
# of built-in functions

→ **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

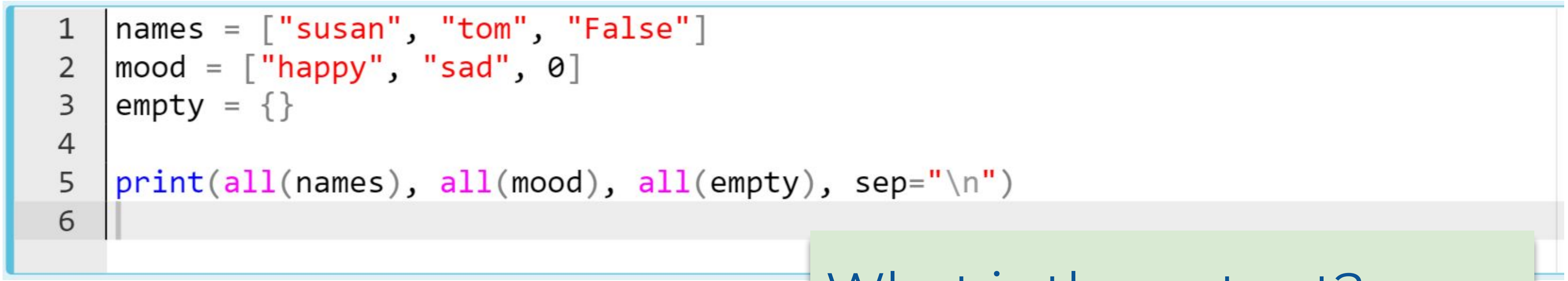
## Built-in Functions

- We ***assume*** that you take a look at the ***built-in functions*** mentioned in the ***pre-class*** content.

- Let’s take a look at several examples of them.

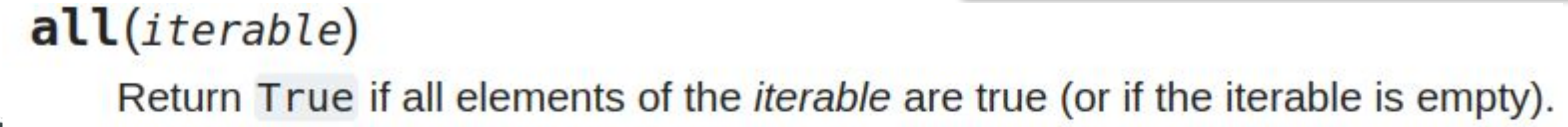
# Built-in Functions

- `all()` function.



```python
names = ["susan", "tom", "False"]
mood = ["happy", "sad", 0]
empty = {}

print(all(names), all(mood), all(empty), sep="\n")

```

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

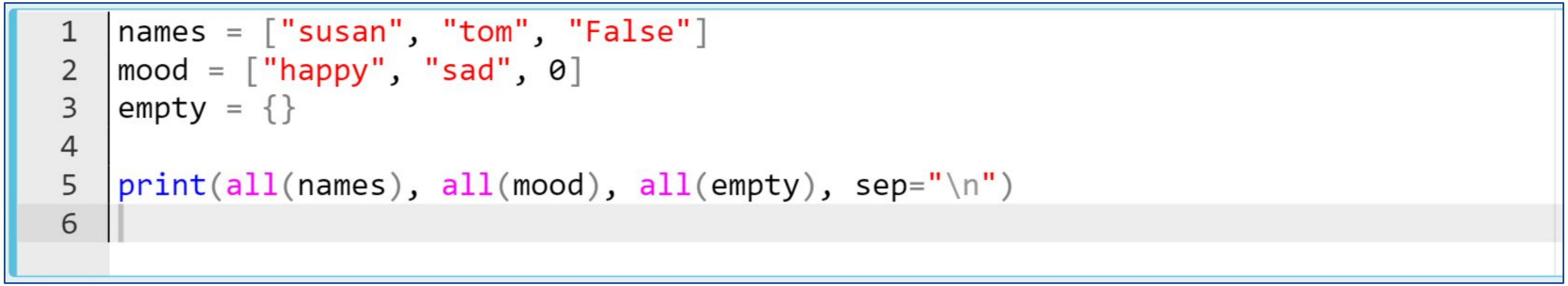


**all**(*iterable*)

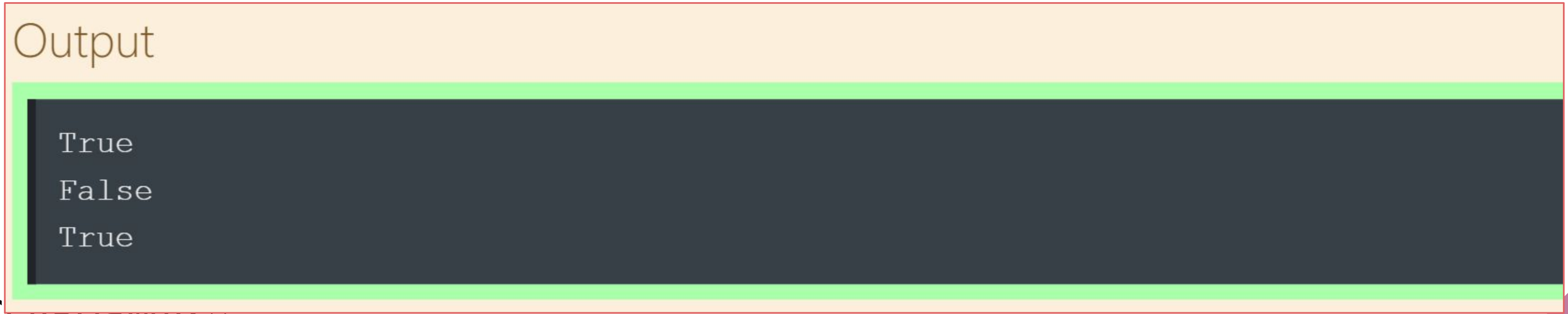
Return `True` if all elements of the *iterable* are true (or if the iterable is empty).

# Built-in Functions

- `all()` function.



```python
names = ["susan", "tom", "False"]
mood = ["happy", "sad", 0]
empty = {}

print(all(names), all(mood), all(empty), sep="\n")
```

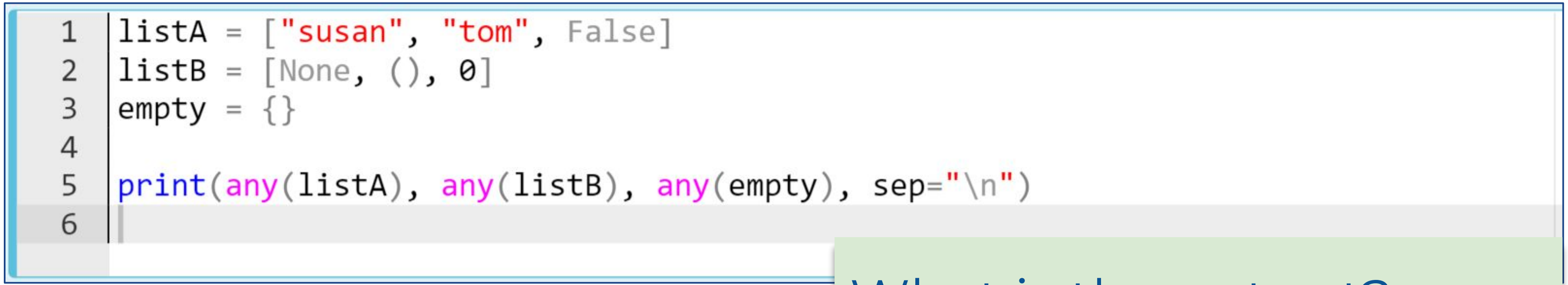


Output

```
True
False
True
```

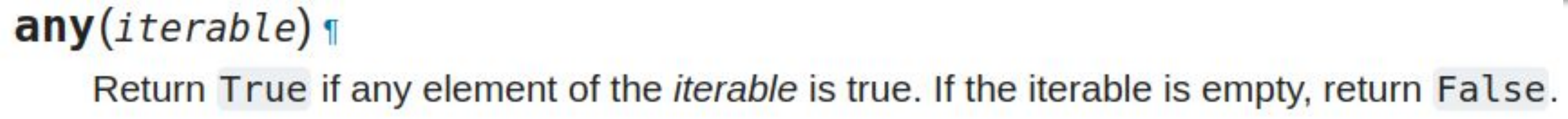
# Built-in Functions

- `any()` function.



```python
listA = ["susan", "tom", False]
listB = [None, (), 0]
empty = {}

print(any(listA), any(listB), any(empty), sep="\n")
```

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

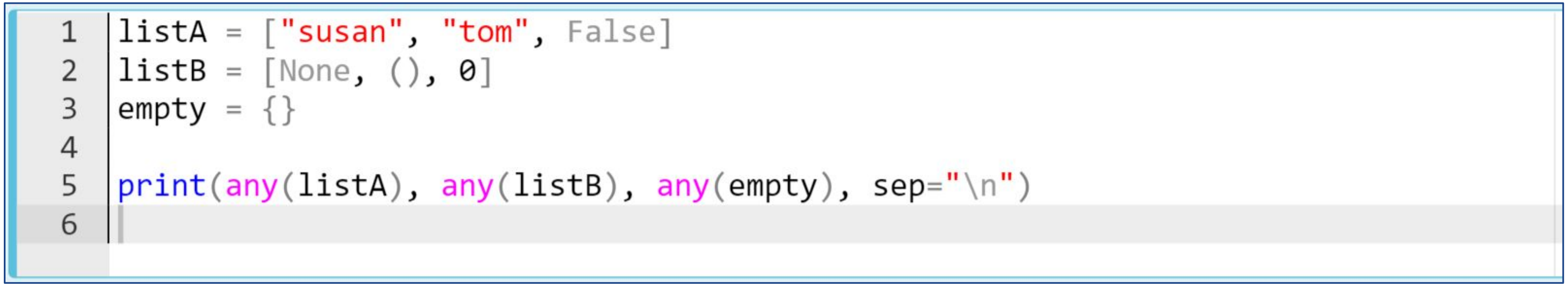


**any**(*iterable*) ¶

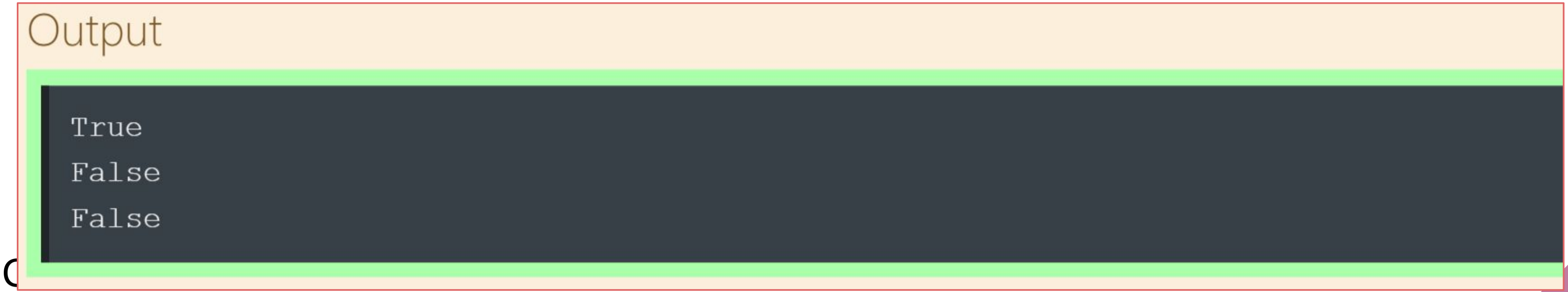
Return `True` if any element of the *iterable* is true. If the iterable is empty, return `False`.

# Built-in Functions

- `any()` function.



```python
listA = ["susan", "tom", False]
listB = [None, (), 0]
empty = {}

print(any(listA), any(listB), any(empty), sep="\n")

```



Output

```
True
False
False
```

# Built-in Functions

- `filter(function, iterable)`.

filter() is used to filter a group of data (iterable) according to a certain criterion(or function).

- Construct an iterator from those elements of ***iterable*** for which *function* returns **true**.

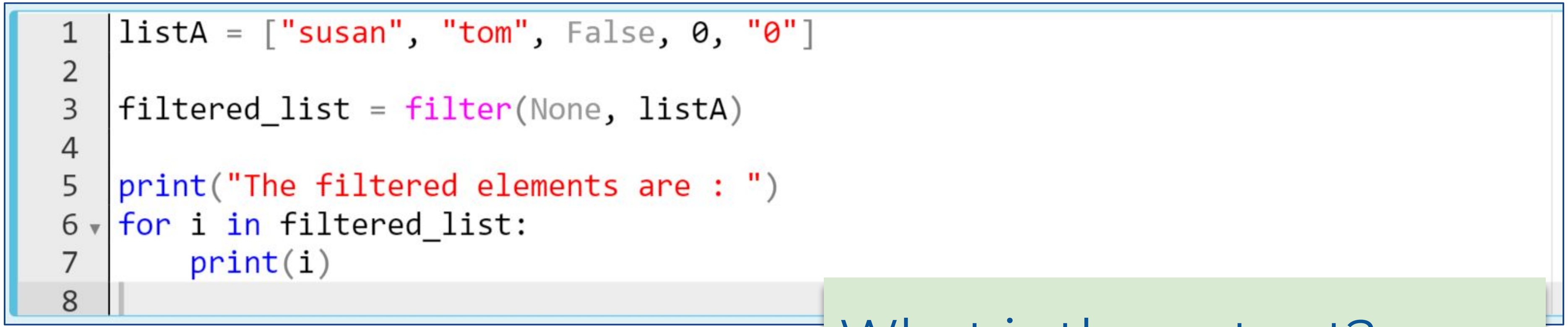
- Note:

  if you pass **None** to function, then filter() uses the identity function and yields all the elements of iterable that evaluate to True:

CLARUSWAY©
WAY TO REINVENT YOURSELF

# Built-in Functions

- `filter(function, iterable)`.



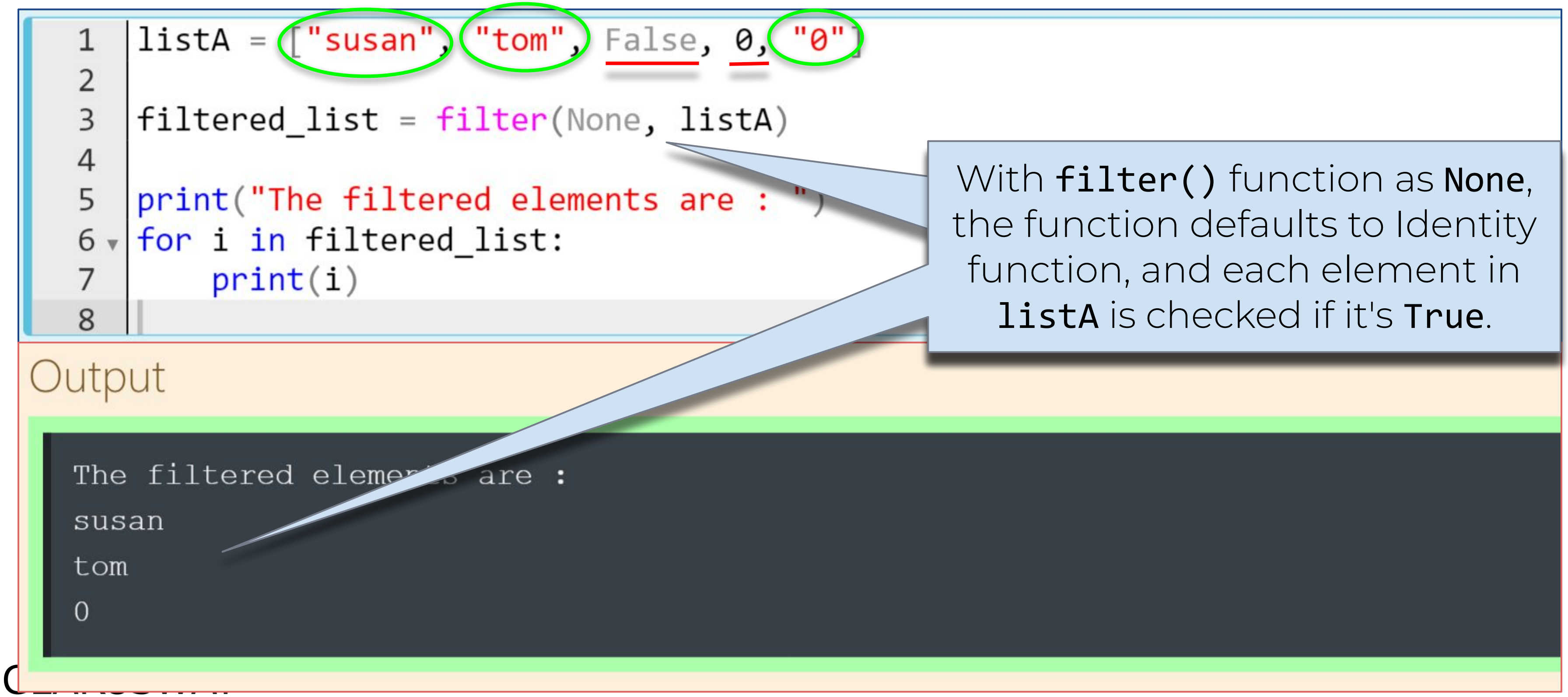
```
listA = ["susan", "tom", False, 0, "0"]

filtered_list = filter(None, listA)

print("The filtered elements are : ")
for i in filtered_list:
    print(i)

```

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

CLARUSWAY©
WAY TO REINVENT YOURSELF

# Built-in Functions

- `filter(function, iterable)`.



```
listA = ["susan", "tom", False, 0, "0"]

filtered_list = filter(None, listA)

print("The filtered elements are : ")
for i in filtered_list:
    print(i)

```

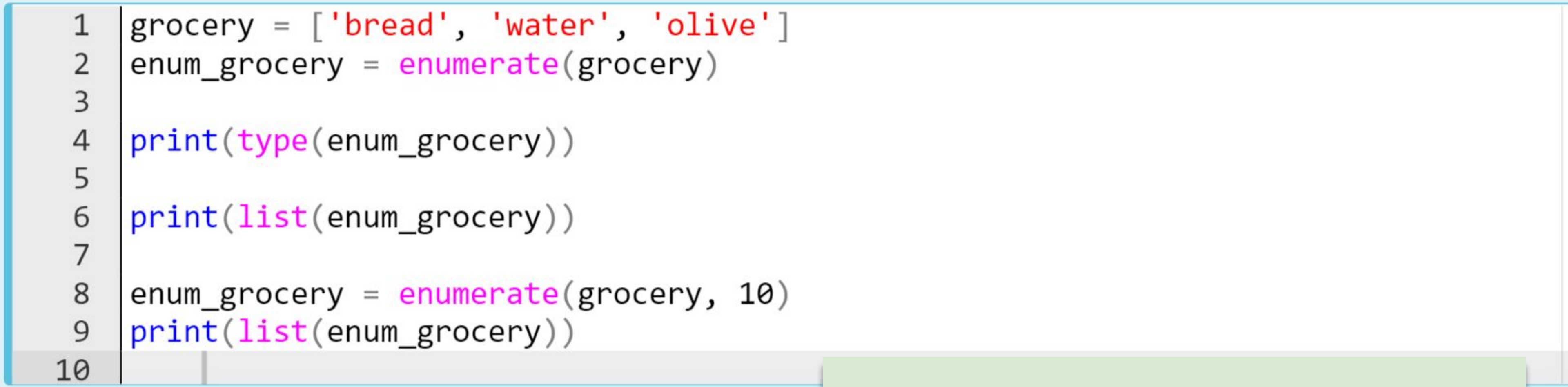
Output

```
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)`.



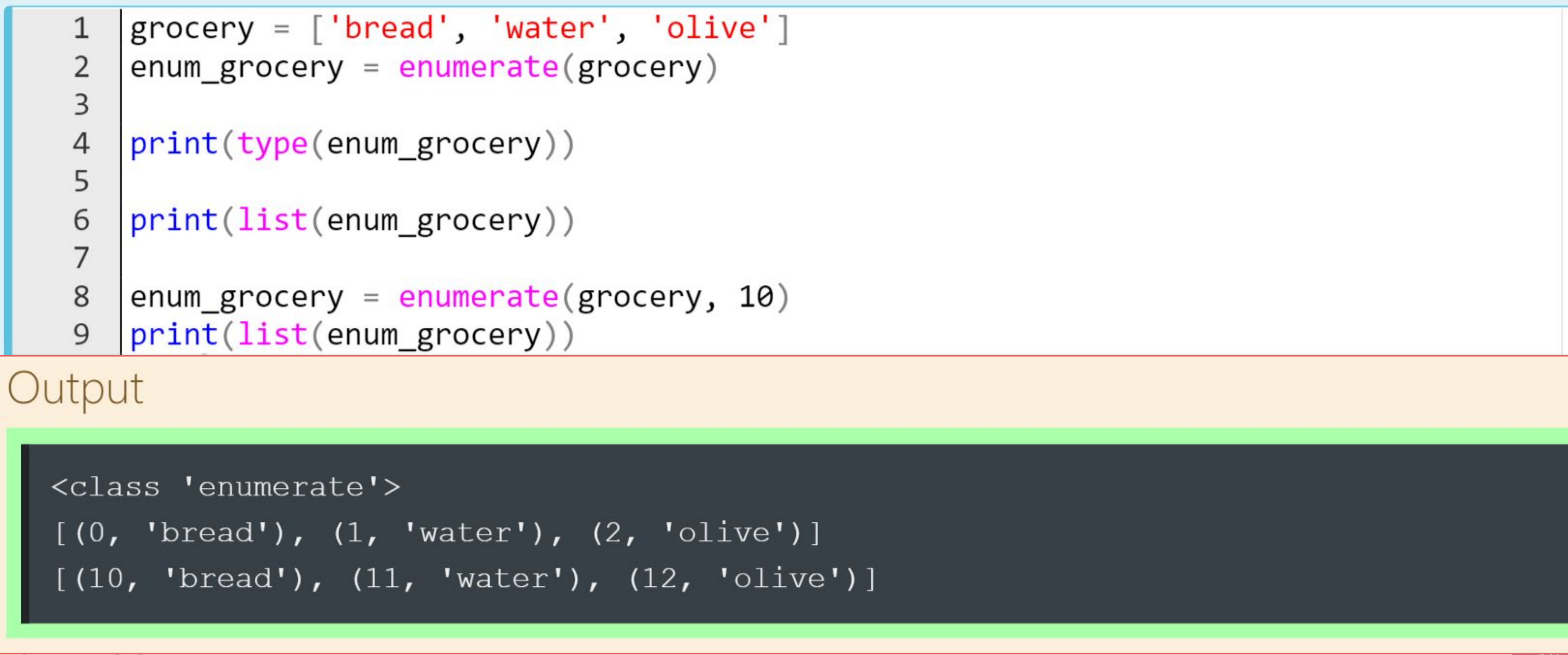
```
grocery = ['bread', 'water', 'olive']
enum_grocery = enumerate(grocery)

print(type(enum_grocery))

print(list(enum_grocery))

enum_grocery = enumerate(grocery, 10)
print(list(enum_grocery))

```

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

CLARUSWAY©
WAY TO REINVENT YOURSELF

# Built-in Functions

- `enumerate(iterable)`.



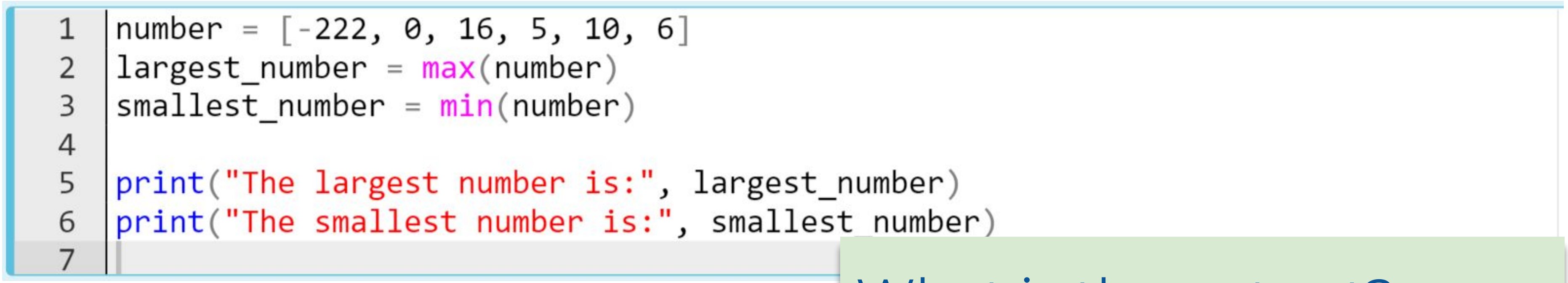
```python
grocery = ['bread', 'water', 'olive']
enum_grocery = enumerate(grocery)

print(type(enum_grocery))

print(list(enum_grocery))

enum_grocery = enumerate(grocery, 10)
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).`



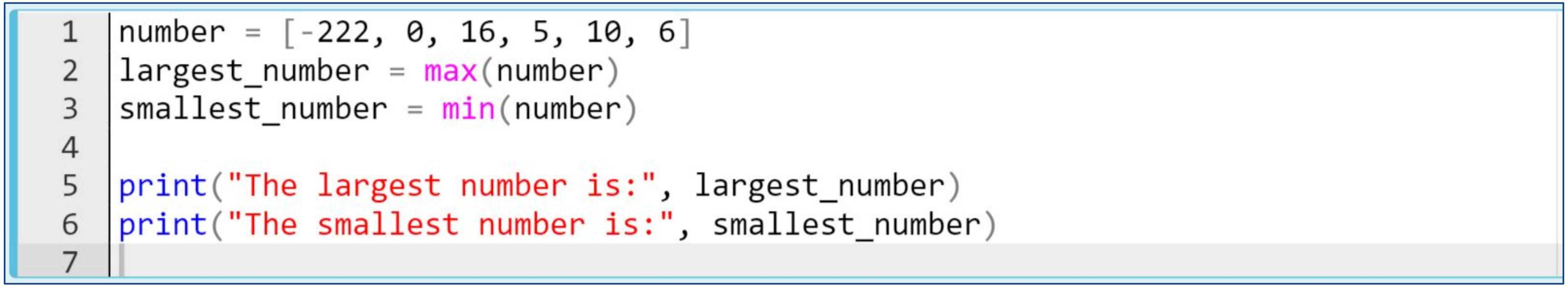
```
number = [-222, 0, 16, 5, 10, 6]
largest_number = max(number)
smallest_number = min(number)

print("The largest number is:", largest_number)
print("The smallest number is:", smallest_number)
```

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

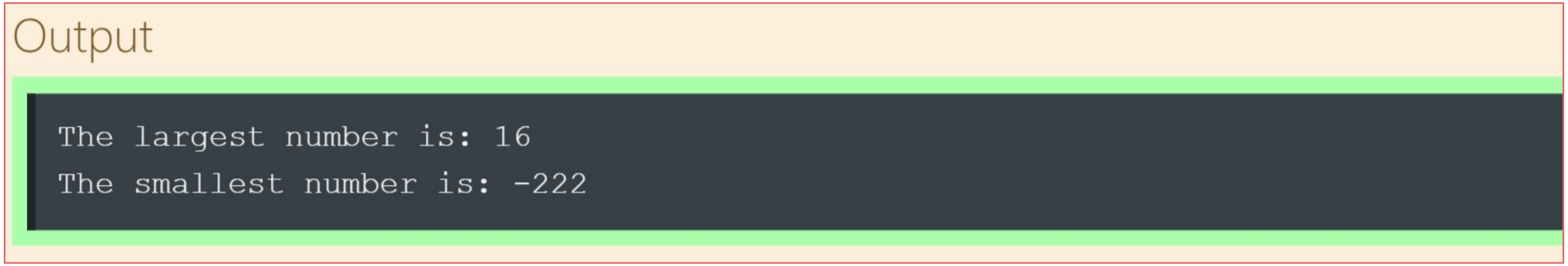
Students, write your response!

# Built-in Functions

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



```python
number = [-222, 0, 16, 5, 10, 6]
largest_number = max(number)
smallest_number = min(number)

print("The largest number is:", largest_number)
print("The smallest number is:", smallest_number)

```

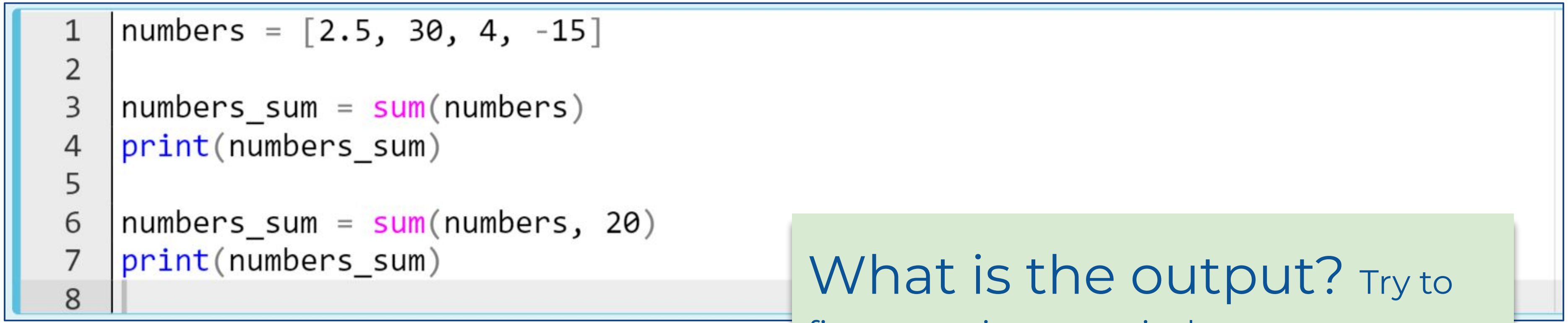


Output

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

## Built-in Functions

- `sum(iterable)`.

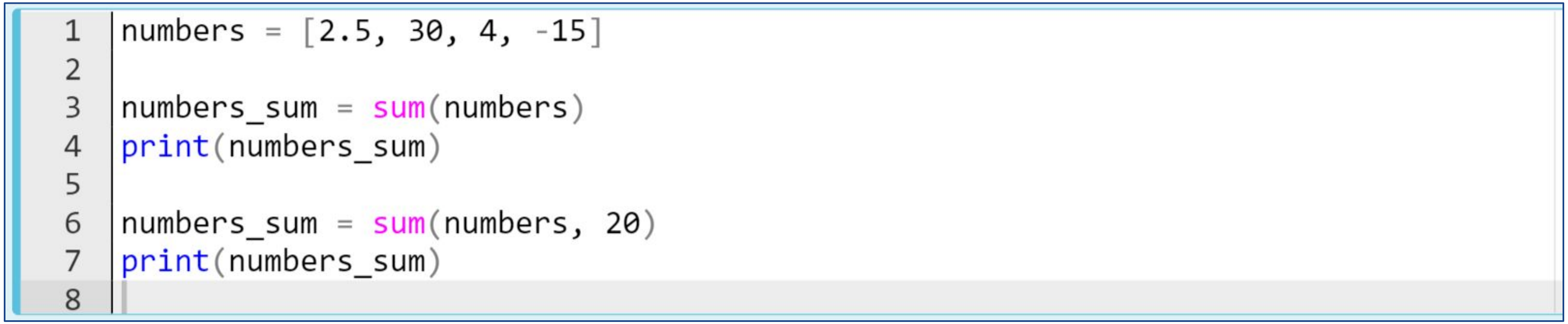


```
numbers = [2.5, 30, 4, -15]

numbers_sum = sum(numbers)
print(numbers_sum)

numbers_sum = sum(numbers, 20)
print(numbers_sum)

```

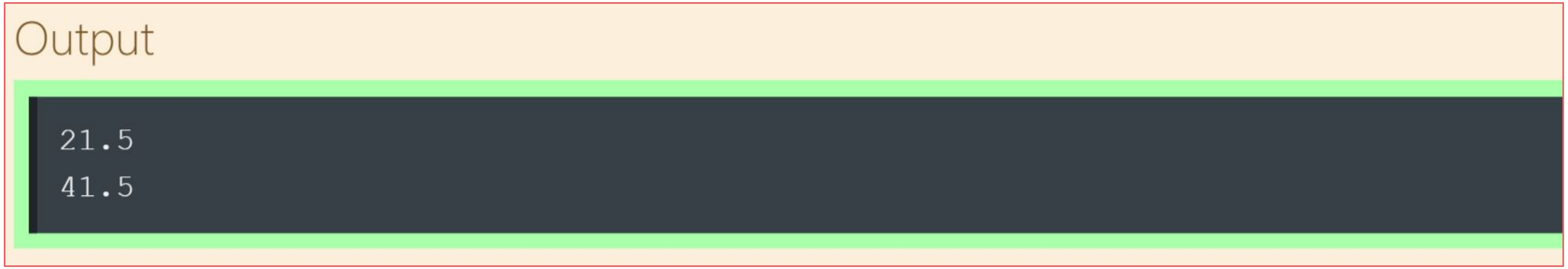
What is the output? Try to figure out in your mind...

# Built-in Functions

- `sum(iterable)`.



```python
numbers = [2.5, 30, 4, -15]

numbers_sum = sum(numbers)
print(numbers_sum)

numbers_sum = sum(numbers, 20)
print(numbers_sum)
```

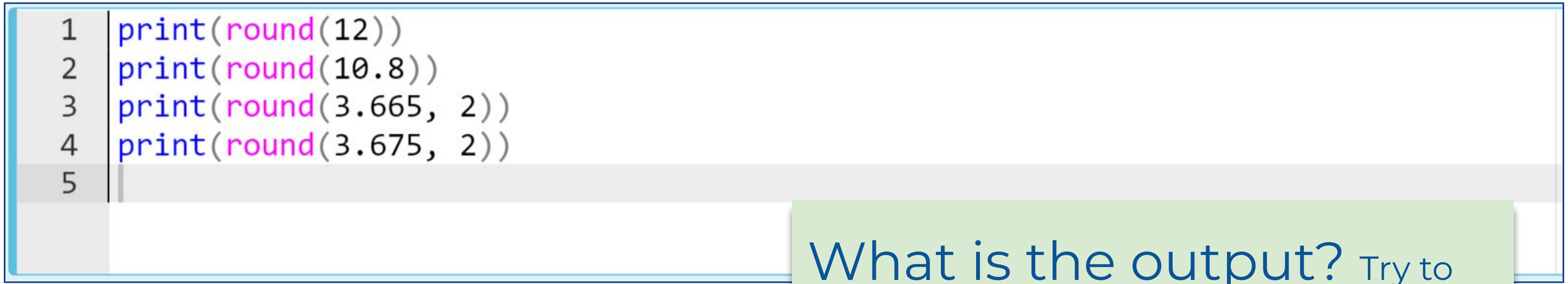


Output

```
21.5
41.5
```

# Built-in Functions

- `round(numbers, ndigits)`.



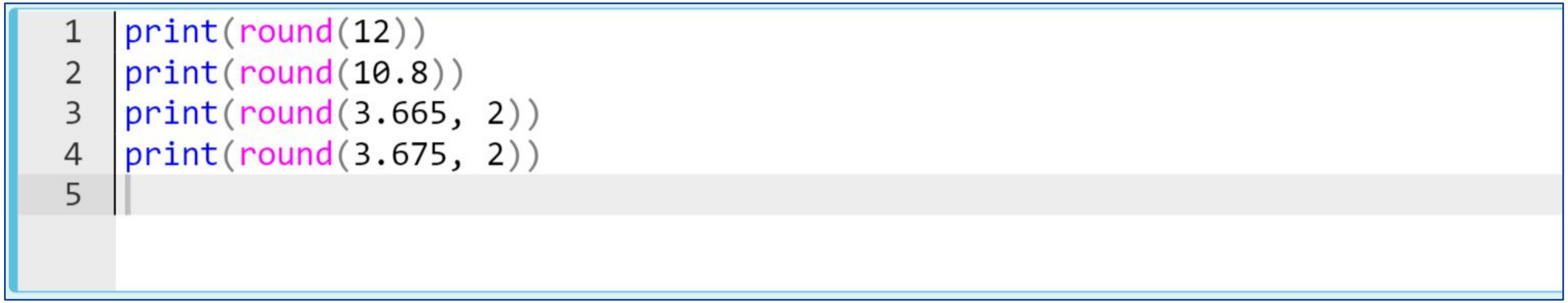
```
print(round(12))
print(round(10.8))
print(round(3.665, 2))
print(round(3.675, 2))
```

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

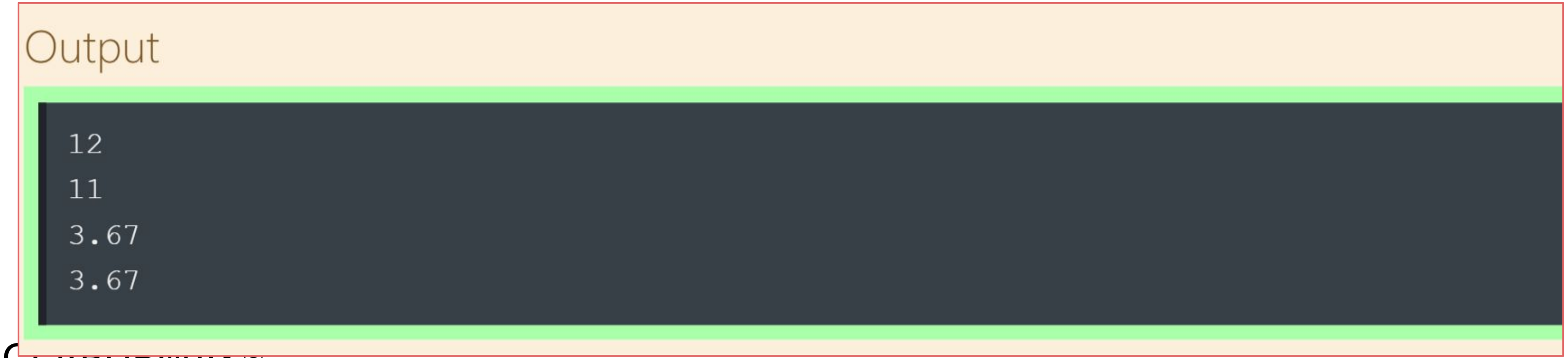
Students, write your response!

# Built-in Functions

- `round(numbers, ndigits)`.



```python
print(round(12))
print(round(10.8))
print(round(3.665, 2))
print(round(3.675, 2))
```

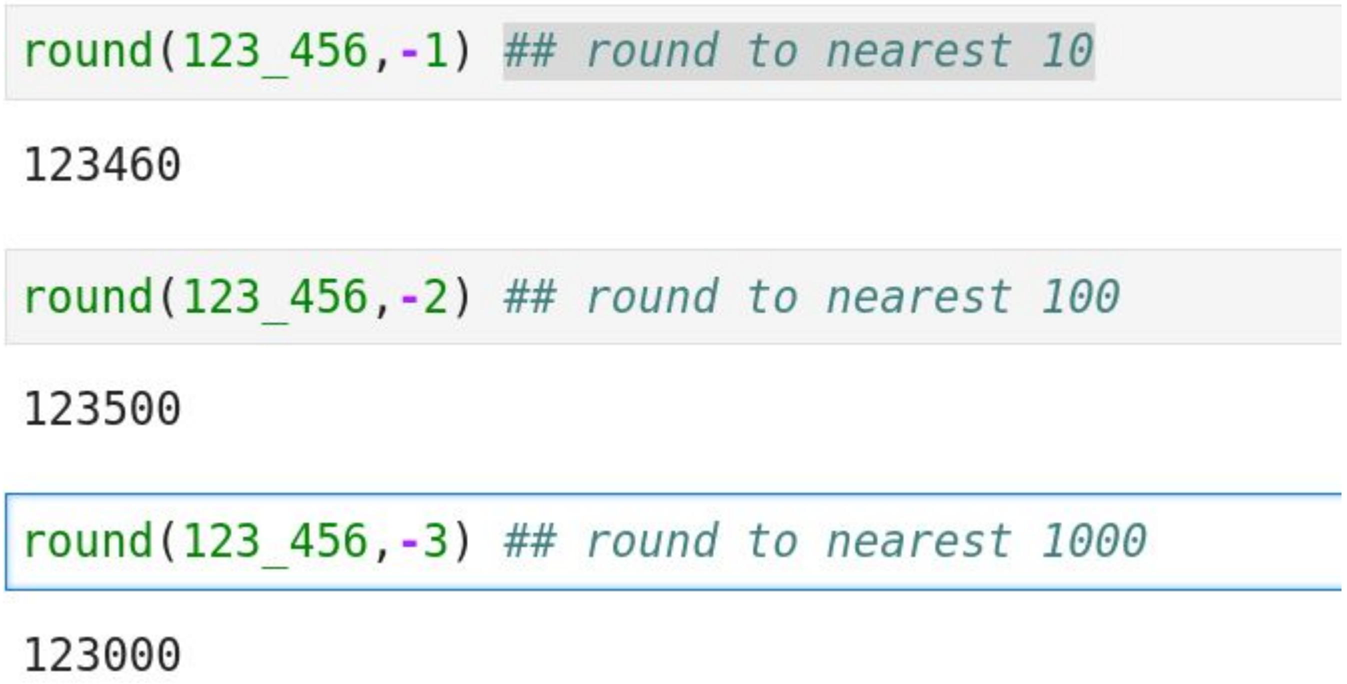


Output

```
12
11
3.67
3.67
```

# Built-in Functions

- `round(numbers, ndigits)`.



```
round(123_456,-1) ## round to nearest 10
```

```
123460
```

```
round(123_456,-2) ## round to nearest 100
```

```
123500
```

```
round(123_456,-3) ## round to nearest 1000
```

```
123000
```

# THANKS!

## Any questions?

You can find me at:

- andy@clarusway.com