**Section 6: Methods and functions**

**11.02.**

**41. Methods and the Python Documentation**

help(object.method) can be used to get info about the method, but method w/0 ()

Also, can be used SHIFT+TAB after writing object\_name.

**46. Tuple Unpacking with Python Functions**

We can return multiple values in Python and use the tuple unpacking for the returned results

**14.02.**

**49. \*args and \*\*kwargs in Python**

args – non keyword arguments

\*args will embed a tuple of arbitrary arguments

kwargs – key word arguments

\*\*kwargs will embed a dictionary of key-value pairs

Args and kwargs naming are a convention of PEP8

Both args and kwargs will let us use as many arguments as wanted

If we set in a function definition the args first and then kwargs, we cannot send position arguments after key word arguments

\*kwargs in a function give us the keys

We cannot use \*args after \*\*kwargs in a function

After \*args we can use keyword arguments before or after \*\*kwargs

[**https://www.programiz.com/python-programming/function**](https://www.programiz.com/python-programming/function)

! In a function we can access the doctring in the following way: function\_name.\_\_doc\_\_

[**https://levelup.gitconnected.com/5-types-of-arguments-in-python-function-definition-e0e2a2cafd29**](https://levelup.gitconnected.com/5-types-of-arguments-in-python-function-definition-e0e2a2cafd29)

A function has formal parameters that are in the definition of the function

A function has actual parameters or arguments that are passed when the function is called

In Python there are multiple types of arguments:

Default arguments

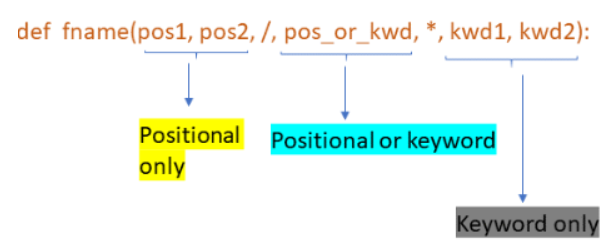
Positional arguments

Keyword arguments

Arbitrary positional arguments

Arbitrary keyword arguments

There is a separation of formal parameters in the function definition to increase readability:



In this way the position and keyword only arguments are enforced

**15.02.**

**50. Function Practice Exercises – Overview**

.join(iterable) returns a string in which between every 2 elements a specific string is positioned

For example ‘#’.join([‘a’, ‘b’, ‘c’]) will give ‘a#b#c’

**16.02.**

**53. Function Practice - Solutions Level Two**

In many cases you can use slicing instead of taking individual elements

For example: l[0] == 2 and l[1] == 3 is the same as l[:2] == [2, 3]

**54. Function Exercise Solutions - Challenge Problem**

If we are searching for a sequence inside a list, we can store the sequence in another list

When an element is found, we pop if from the list

The primes searching process can be speed-up if use the list of found primes until that moment

There can be used an else statement even for a for loop

It will be executed when the for loop ends, but not when there was an exit due to a break

Else statements can be user for a for loop to check if the exit was caused by a break or not

**55. Lambda Expressions, Map, and Filter Functions**

Map(function, iterable) -> applies the function on each element

Filter(function, iterable) -> function must return true/false and filters the elements

Reduce(function, iterable) -> combines all the elements based on function and returns a value

Reduce can be found just in functools library, not in the default one

Lambda expression: lambda x: x + 1

We can assign a lambda expression to a variable that can be called

For example: f = lambda x: x + 1; f(10)

Lambda function must not be that complex and accept multiple parameters

Map, filter and reduce can be applied on lists, tuples, sets and dictionaries

With dictionaries we can use: dict(map(lambda x: (x[0] + 1, x[1] \* 10), iterable.items()))

**17.02.**

**56. Nested Statements and Scope**

LEGB rule as the order for scoping in Python:

L: Local – names assigned in any way within a function and not declared global in that function

E: Enclosing function locals – names in the local scope of any enclosing func from inner to outer

G: Global (module) – assigned at the top-level of a module or declared global within the func.

B: Built-in (Python) – open, range, SyntaxError and so on

To change the value of a global var, then at the top of the func. use “global var\_name”

We can’t use “global var\_name” if the var\_name was defined just in an outer enclosing function

If the var is numeric or boolean and it was passed as a parameter, we cannot change its value

**Check the data type hierarchy and the complex data type**

[**https://python-course.eu/python-tutorial/passing-arguments.php**](https://python-course.eu/python-tutorial/passing-arguments.php)