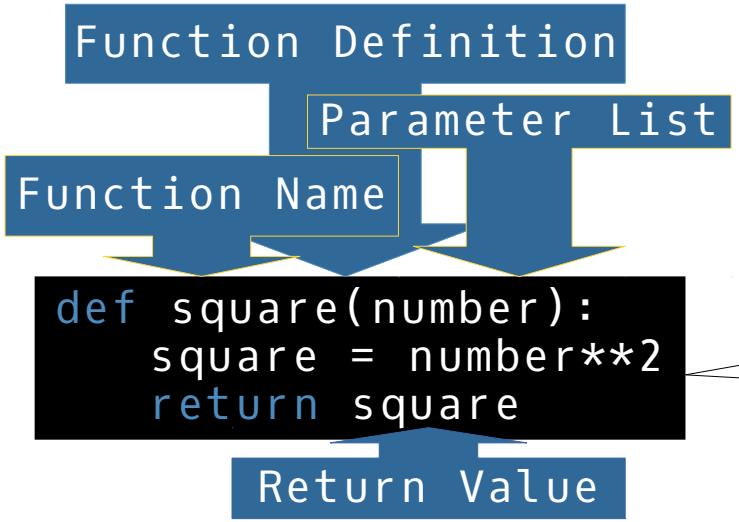
Functions

What's a function?




```
"""
    Your interesting program
"""
a   = 5
b   = 10

aa = square(a)
bb = square(b)

print(f"{aa}, {bb}")
```

Function Call

Function Call

Important: Variables Defined in the function can only be used inside of the function they are declared in.

Advanced Functions

```python
def square_sum(array: list, weights: (list, float) = 1, normalise: bool = False) -> float:
    """
    This function computes the square sum of an array. You can pass a second array with the same length as the first array. The second array will be used as weights.

    Parameters
    ----------
    array : list of float
        An array of numbers.
    weights : list of float or float, optional
        An array of numbers or a single number. Will be used to compute the weighted sum of the parameter array. If the parameters weights is shorter than the array parameter, the elements of weights will be repeated until the lengths match. If the parameter weights is to long, weights without matching element in the array parameter will be ignored. If weights is a single number, it will be converted into a list. The default is 1.
    normalise : bool, optional
        If true the array of weights will be normalised with the sum of the weights. The default is False.

    Returns
    -------
    square_sum : float
        The weighted square sum of the the passed array..

    """

    # Check the type of the weights
    if not isinstance(weights, list):
        # Convert the parameter weights into an array of the length len(array)
        weights = [ weights for number in array ]

    # Check the length of the array
    # If the array is longer than the weights array make them the same length
    elif len(weights) < len(array):
        # Create a list of indices
        # There will be one index for every element in the parameter array
        # repeated_indices contains repeating indices of the parameter weights
        repeated_indices = [ index % len(weights) for index, number in enumerate(array) ]
        # Use repeated_indices to elongate the weights list
        weights = [ weights[index] for index in repeated_indices ]

    # Normalise the weights
    if normalise:
        weights = [ weight / sum(weights) for weight in weights ]

    # Calculate the weighted square sum
    square_sum = sum([ weight * number**2 for weight, number in zip(weights, array) ])

    # Return the result
    return square_sum
```

```
In [69]: help(square_sum)
Help on function square_sum in module __main__:

square_sum(array: list, weights: (<class 'list'>, <class 'float'>) = 1, normalise: bool = False) -> float
    This function computes the square sum of an array. You can pass a second array with the same length as the first array. The second array will be used as weights.

    Parameters
    ----------
    array : list of float
        An array of numbers.
    weights : list of float or float, optional
        An array of numbers or a single number. Will be used to compute the weighted sum of the parameter array. If the parameters weights is shorter than the array parameter, the elements of weights will be repeated until the lengths match. If the parameter weights is to long, weights without matching element in the array parameter will be ignored. If weights is a single number, it will be converted into a list. The default is 1.
    normalise : bool, optional
        If true the array of weights will be normalised with the sum of the weights. The default is False.

    Returns
    -------
    square_sum : float
        The weighted square sum of the the passed array..
```

```
In [70]: square_sum([1,2,3,4,5,6,7])
Out[70]: 140

In [71]: square_sum([1,2,3,4,5,6,7], weights = 0.4)
Out[71]: 56.0

In [72]: square_sum([1,2,3,4,5,6,7], weights = [0.4, 0.6])
Out[72]: 67.19999999999999

In [73]: square_sum([1,2,3,4,5,6,7], weights = [0.4, 0.6, 0.1, 0.2, 0.9, 0.1, 0.2])
Out[73]: 42.8

In [74]: square_sum([1,2,3,4,5,6,7], weights = [0.4, 0.6, 0.1, 0.2, 0.9, 0.1, 0.2], normalise = True)
Out[74]: 17.119999999999997

In [75]: square_sum([1,2,3,4,5,6,7], True)
Out[75]: 140

In [76]: square_sum([1,2,3,4,5,6,7], normalise = True)
Out[76]: 20.0
```

```
In [77]: square_sum(2)
Traceback (most recent call last):

  File "<ipython-input-77-46ee8a2dbdbf>", line 1, in <module>
    square_sum(2)

  File "/home/jonas/Dokumente/pythonBootcamp2021/code/example_01_functions.py", line 36, in square_sum
    # Use repeated_indices to elongate the weights list

TypeError: 'int' object is not iterable
```

Yielding Generators

```
>>> def generate_squares(numbers):
...     for i in numbers:
...             if i % 2 == 0: yield i**2
...
>>> squares = generate_squares([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> squares
<generator object generate_squares at 0x7ff657321a50>
>>> for i in squares:
...     print(i)
...
0
4
16
36
64
>>>
```

```
>>> def list_squares(numbers):
...     array = []
...     for i in numbers:
...             if i % 2 == 0: array.append(i**2)
...     return array
...
>>> squares = list_squares([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> squares
[0, 4, 16, 36, 64]
>>> for i in squares:
...     print(i)
...
0
4
16
36
64
>>>
```

Lambda Functions

```
lambda parameter : expression
```

```
>>> l = lambda a, b : a**b
>>> l(3, 4)
81
>>> def multiplier(factor):
...     return lambda x : factor * x
...
>>> doubler = multiplier(2)
>>> doubler(5)
10
>>> trippler = multiplier(3)
>>> trippler(5)
15
>>>
```

Callback Functions

```python
def some_computation_taking_unknown_time(array, callback):
    print("DOING SOME COMPUTATION")
    result = sum(array)
    callback(result)
    print("DONE COMPUTING")

some_computation_taking_unknown_time(range(0, 1000), lambda res : print(f"Callback function says '{res}'."))

print("-----------------------------------------------")

def some_other_callback_function(result):
    print(f"The other callback function also says '{result}'.")

some_computation_taking_unknown_time(range(0, 1000), some_other_callback_function)
```

```
DOING SOME COMPUTATION
Callback function says '499500'.
DONE COMPUTING
-----------------------------------------------
DOING SOME COMPUTATION
The other callback function also says '499500'.
DONE COMPUTING
```

Exercise 8: Parsing Files

Convert the script from exercise 7 into a function parsing files.
The function takes the path to the file as parameter.
The return value is the parsed data.
Keep in mind: The headers of the files are different.