A brief introduction to Python - part 2, functions and packages

*By Nicola Romanò - last updated on **25 July 2024***

This is the second part of a brief introduction to Python. In this part we will cover functions and packages.

1. Functions

Functions are a way to group pieces of code together so they can be reused.

Functions are defined using the `def` keyword. Functions can accept arguments and return values. For example:

What does the following code do? Try to understand it and guess the output before running it.

```
def square(x):
    return x**2

print(square(5))
```

Other functions might not have any arguments and/or return values. For example this function prints "Hello!", but does not return anything nor accept any arguments:

```
def say_hello():
    print("Hello!")

say_hello()
```

You can return multiple values from a function:

```
def get_min_max(numbers):
    return min(numbers), max(numbers)

min, max = get_min_max([1, 2, 3, 4, 5])
print(f"The minimum value is: {min}, the maximum is: {max}")
```

Exercise 1

1. Write a function that takes a list of numbers and returns the sum of all the numbers in the list.

2. Write a function that takes a string and returns the string reversed. (Hint: you can access the characters of a string using the square brackets `[]`, like you would with a list)

It is good practice to include **docstrings** in your functions. This is a string that describes what the function does. For example:

```
def square(x):
     """Return the square of x.
     Parameters
         x : int or float - the number to square
     Returns
         int or float - the square of x
     """
     return x**2
```

Default arguments

Sometimes you want to have default values for some arguments. You can do this by assigning a value to the argument in the function definition. You can then call the function without providing a value for that argument, if you want to use the default value.

```
def get_top_n(numbers, n=3):
    """
    Return the top n numbers in the list.
    Parameters
        numbers : list - the list of numbers
        n : int - the number of top numbers to return
    Returns
        list - the top n numbers in the list
    """

    # Sort the numbers in descending order
    numbers.sort(reverse=True)
    return numbers[:n]
```

What would the following code output?

```
numbers = [5, 38, 22, 1, 3, 83, 44, 12]
print(get_top_n(numbers))
print(get_top_n(numbers, 5))
```

Important: default arguments should always come after non-default arguments in the function definition. The following code would raise an error:

```
def get_top_n(n=3, numbers): # Should be def get_top_n(numbers, n=3):
    ...
```

2. Importing external libraries

Sometimes you need to do some complex operations that are not available in the standard Python library. Fortunately, Python has a large ecosystem of libraries that can be used in your code. You can import a library by using the `import` keyword. For example to import the `math` library you can use:

```
import math
```

All of the functions and variables in the `math` library are now available to you. For example:

```
print(math.sqrt(16)) # Square root of 16
print(math.pi)
```

Sometimes the name of the library is long or not descriptive enough. In this case you can use an alias to refer to the library:

```
import math as m
print(m.sqrt(16))
```

Certain libraries are very complex and contain many functions. If you just need one or two of the functions, you can import just those. You can then use the function names without the library name.

```
from math import sqrt, pi
print(sqrt(16))
print(pi)
```

There are thousands of libraries available for Python. A good place to find libraries is the Python Package Index (PyPI). Generally you can install a library using the `pip` command. For example to install the `numpy` library (a library for numerical operations which we will use extensively in the course!) you can use:

```
pip install numpy
```

3. Writing your own modules and libraries

You can also write your own libraries and modules. A module is a file containing Python code. You can import a module in the same way you import a library. For example, if you have a file called `math_functions.py` containing the following code (this is an extremely simple example, this would not be a very useful module!):

```
def add(x, y):
    return x + y

def subtract(x, y):
    return x - y
```

You can import this module in another file located in the same directory:

```
import math_functions as math_fn

print(math_fn.add(5, 3))
print(math_fn.subtract(5, 3))
```

You can also import just the functions you need:

```
from math_functions import add

print(add(5, 3))
```

x

Writing your own modules is a good way to organise your code and make it more readable and maintainable.

A package is a collection of modules. Creating packages (e.g. those that can be installed using `pip`) is a bit more complex, and outside the scope of this document but if you are interested you can find more information in the Python documentation.

4. Conclusion

In this document we have covered functions and packages in Python. These are fundamental concepts that are important to understand in order to write clean and maintainable code.

You are free to:

Share — copy and redistribute the material in any medium or format

Adapt — remix, transform, and build upon the material
for any purpose, even commercially.

Under the following terms:

Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

ShareAlike — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.