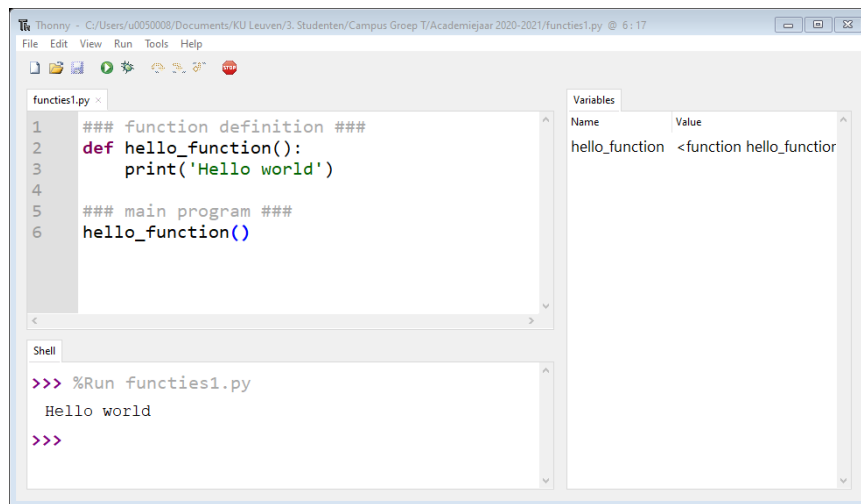


Exercise Session 3: Functions

Make a clear distinction between the **function definition** and the **function call**. You can do this by adding comments. When you write a `#` in your program, what you write after it on that line will be considered a comment and not code (light gray). Use this to make a distinction between two sections: a section where you write function definitions, and the main program where you make function calls. For example, for a very simple function with no parameters, which just prints “Hello world” in the shell window:



The screenshot shows a Python IDE window titled 'Thonny'. The main editor displays a file named 'functies1.py' with the following code:

```
1  ### function definition ###
2  def hello_function():
3      print('Hello world')
4
5  ### main program ###
6  hello_function()
```

On the right side, there is a 'Variables' panel showing a table with two columns: 'Name' and 'Value'. It contains one entry: 'hello_function' with the value '<function hello_function>'. At the bottom, there is a 'Shell' panel showing the execution of the code:

```
>>> %Run functies1.py
Hello world
>>>
```

Exercise 1: Parameter-based greeting

The real added value of functions becomes apparent once you add a parameter. Rewrite the **definition** of the `hello_function()` function so that it accepts a name parameter, which is the name of the person to be greeted. The function mixes the name into the message. When **calling** the function you will also have to pass a value as a parameter.

You can also call the function several times in the “main” so that, in this case, you can also greet several people. You repeat the function **call** with different concrete values for the parameter. At the top of your code, the function **definition** remains unchanged.

Check whether you get the correct output:

```
### main program ###
hello_function('Hannelore')
hello_function('Nigel')
```

Printed output in the shell window:

```
Shell
>>> %Run functies1.py
Hello Hannelore
Hello Nigel
>>>
```

Exercise 2: Greeting based on two parameters

A function can also have more than one parameter. Write a function `hello_to()` that accepts two parameters. The first is again the name of the person to be greeted. The second parameter is where we greet the person. For example, the function call

```
hello_to("Stef", "Group T")
```

should give the printed output:

```
Hello Stef, and welcome to Group T.
```

You can print several strings together in two ways: `print("text", "more text")` will print the two parts, separated by a space. `print("text" + "more text")` will concatenate the strings (without spaces in this case) and print.

Exercise 3: Area of the circle as a function

In session 2 you wrote the code to calculate and print the area of a circle. You hadn't learned to work with functions and function parameters at that point. Write a function `calculate_area()` that calculates the area of a circle with the radius of the circle as a parameter and prints in the Shell. In the main program you call the function in the following way:

```
calculate_area(10)
```

Note that you don't pass the (rounded) value of pi as a parameter!

Exercise 4: Greeting based on name length

You can do more in a function than just print text. For example, you can add an if-else structure to it. Write a function `greet_based_on_name_length()` that accepts a name parameter. The function decides which greeting is printed based on the length of the name: "Hello <name>, you have quite a long name" or "Hello <name>, you have a short name". We consider names with less than 4 letters to be "short".

Exercise 5: Greeting based on name length, as a parameter

Adjust the function of exercise 4 so that you can decide what length you consider "short".

Exercise 6: Full greeting or not?

Write a function `greet_full_or_not()` that accepts two parameters. The first is again the name of the person to be greeted. The second parameter is a Boolean that specifies whether or not the greeting should be “full”. If not (False) the greeting is simply “Hello <name>”, if yes (True) the greeting is “Hello <name>, I hope you have a pleasant day”.

You should be able to write this function *without* writing anything of the form `== True` or `== False` in the if structure (!).

Exercise 7: Alcohol testing

The Leuven police wants to check for drunk cyclists. The fines increase according to the alcohol percentage in your blood (blood alcohol content). Come up with a solution to calculate the fine to be paid for a given alcohol percentage. Write a function `calculate_fine()` that prints the fine in the Shell based on the blood alcohol content, which you pass as a parameter:

- Alcohol percentage < 0.5‰ : no fine
- 0.5‰ to 0.8‰: €50 fine
- Larger than 0.8‰ and up to 1.5‰: the fine increases linearly from €50 at 0.8‰ to €150 at 1.5‰. You can use the following formula to calculate the fine in this case¹:
$$50 + (((\text{alcoholpercentage} - 0.8) / 0.7) * 100)$$
- If alcohol percentage > 1.5‰: €250 fine and printed message: “you are in mortal danger”
- If an incorrect (negative) value is entered: “Error, negative value detected”

In these rules the alcohol percentage is expressed in per mille, in your code you can translate this into a float: 0.5‰ should be passed as simply the number 0.5.

Exercise 8: Movie access via nested ifs

1.

Write a `movie_access()` function that accepts two parameters. The first is the age of the visitor. The second parameter is a Boolean that specifies whether the movie is suitable for adults or not. Implement the following flowchart using a nested if-structure (= an if-statement within another if-statement). You are NOT allowed to use an `elif` in your code.

2.

Now write a function `movie_access_alternate()` that does the same thing, but this time using an `elif` instead of a nested if-structure.

Complex if-structures can usually be written in both ways (or even a combination). As long as your code does what it’s supposed to, it doesn’t matter which approach you use. In the future, the choice is yours.

¹ Try to figure out how we came up with this formula, it’s not that complex!

