Courses (/student) / Web Development 1 (Wien) (/student/course/5727275479728128) / 12) Functions

12) Functions

Agenda

- Intro to functions
- Reusability
- Importing functions
- Function input (parameters)
- Function output (return)

Functions help us to better **structure** our code and made some parts **reusable.**

Let's create a simple example to see how functions look like. Create a new Python file called `functions.py` and add this code in it:

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

This is a simple function that prints out the string "Hello!". Let's run the Python program...

What happens?

Nothing!

Why?

Because we only made the definition of the function (hence the `def` keyword). We haven't called it yet. So let's call it:

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


say_hello()
```

What happens if we run the program now?

We get a `"Hello!"` written in our terminal.

Reusability

Since functions can be reused, we can call a single functions many times. Add five `say_hello()` calls in your code:

```
say_hello()
say_hello()
say_hello()
say_hello()
say_hello()
```

Now run the program. What happens?

We get five `"Hello!"` lines in our Terminal. Because we called the function 5 times in our code.

Importing functions

We can even import a function from another Python file. Let's see how this goes.

Important: First **delete** all the `say_hello()` calls in the `functions.py` file. It should contain **only** the function defintion:

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

Then **create a new Python file** and name it `hello.py` .

Inside this new file **call** the say_hello function from functions.py:

```
say_hello()
```

As you can see, `say_hello()` is underlined red in PyCharm, which symbolizes an error. This is because we have to add the **import** statement at the top of the hello.py file:

```
from functions import say_hello
```

Now run the `hello.py` program. As you can see, the Terminal prints out `"Hello!"` .

If you remember, we already imported functions before, for example:

```
from random import randint
```

Or this one:

```
from datetime import datetime
```

Both `randint` and `datetime` are functions that someone esle created and we can use them in our projects (they are part of the Python standard functions and come pre-installed together with the Python installation).

Function input

But functions can do much more than just print the same string every time.

We can make the same function print out **different** strings each time we call it. Let's see how!

Functions can accept data as an input. So far our `say_hello()` function does not accept any input. But we can rewrite it in a way that it would.

Change the function like this:

```
def say_hello(name):
    print("Hello {0}!".format(name))
```

What we've added to our function is called a **parameter** and it's a way of entering data in a function. In our case, the parameter is a **variable** called `name` and it's a **string**.

Let's run the `hello.py` file again. What happens? **An error!**

That's because we have to also change our function call. Currently our function call looks like this:

```
say_hello()
```

But since we've updated our function, it wants to have some data inserted. More precisely, a paramater called `name`.

So let's add the name parameter in our function call:

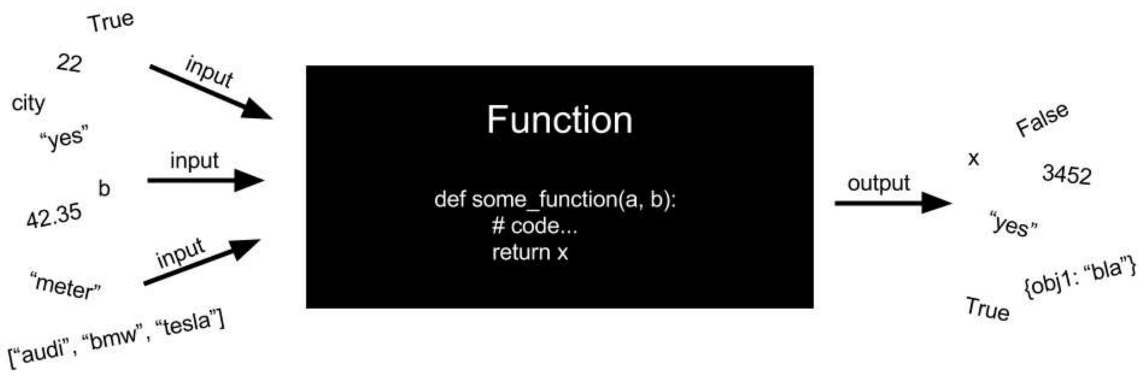
```
say_hello(name="Matt")
```

Let's run the `hello.py` file now. What do we get in the Terminal? "Hello Matt!". Yaaaay! :)

> *See the complete example here (https://github.com/smartninja/wd1-py3-exercises/tree/master/lesson-12/simple-example).*

Function output (a.k.a "return")

But functions don't only receive data (if they want). They can also output (or **return**) data.



Let's change our function again. Instead of printing a hello string, we will return it to the function call:

```
def say_hello(name):
    return "Hello {0}!".format(name)
```

Let's run the `hello.py` again. What happens?

Nothing. At least nothing gets printed in the Terminal.

It's because we've removed the `print()` function from our code.

But we can put it back - not in the function, but where we call the code. In the `hello.py` file.

Let's first save the function result in a variable:

```
text = say_hello(name="Matt")
```

This variable now contains a string `"Hello Matt!"`.

Next let's print the variable in the Terminal:

```
print(text)
```

Let's run the program... Success! The `"Hello Matt!"` string is again visible in our Terminal!

> *The shorter version would be to write everything in one line:* `print(say_hello(name="Matt"))`*. Use whichever you prefer more.*
>
> *See the complete example here (https://github.com/smartninja/wd1-py3-exercises/tree/master/lesson-12/simple-example-return).*

Exercise 12.1: The sum of two numbers

Let's create a function that will take two numbers as parameters and add them together.

Call this function many times in your program, each time with different parameters.

Exercise length: 5-10 minutes. After the time is up, the instructor will present the solution to the students.

▶ Click to see the solution.

Q&A

Any question?

If there's enough time after Q&A, students can start working on their homework.

Homework 12.2: Make the guessing game more modular

Make our "Guess the secret number" game more modular.

Currently, the game looks like this:

```
import datetime
import json
import random

secret = random.randint(1, 30)
attempts = 0

with open("score_list.txt", "r") as score_file:
    score_list = json.loads(score_file.read())

print("Top scores: " + str(score_list))

for score_dict in score_list:
    print(str(score_dict["attempts"]) + " attempts, date: " + score_dict.get("date"))

while True:
    guess = int(input("Guess the secret number (between 1 and 30): "))
    attempts += 1

    if guess == secret:
        score_list.append({"attempts": attempts, "date": str(datetime.datetime.now())})
        with open("score_list.txt", "w") as score_file:
            score_file.write(json.dumps(score_list))

        print("You've guessed it - congratulations! It's number " + str(secret))
        print("Attempts needed: " + str(attempts))
        break
    elif guess > secret:
        print("Your guess is not correct... try something smaller")
    elif guess < secret:
        print("Your guess is not correct... try something bigger")
```

Use functions to make the game more modular.

Also try to add another while loop so the user can play many rounds of the game without having to re-run the program each time.

Hint, the game would look something like this:

```
while True:
    selection = input("Would you like to A) play a new game, B) see the best scores, or C) quit

    if selection == "A":
        play_game()
    elif selection == "B":
        for score_dict in get_top_scores():
            print(str(score_dict["attempts"]) + " attempts, date: " + score_dict.get("date"))
    else:
        break
```

And you'll probably need three functions:

- `play_game()`
- `get_score_list()`
- `get_top_scores()`

But like with anything in programming, there are many ways on how to do this homework. So you can do it in some other way.

> *Solution (https://github.com/smartninja/wd1-py3-exercises/tree/master/lesson-12/guess-secret-number).*

Optional task: game levels

You can let user choose the level of the game:

```
play_game(level="hard")
```

or

```
play_game(level="easy")
```

The easy level has suggestions "try something smaller" or "try something bigger". The hard level does not have these suggestions, it only tells you if your guess is right or wrong.

> *Solution (https://github.com/smartninja/wd1-py3-exercises/tree/master/lesson-12/guess-secret-number-levels).*

Bonus homework 12.3: Geography Quiz

A local gaming company contacted you to build a game for them. It is a simple **geography quiz** where a user has to **guess the capital city of some country**:

```
> Game: What is the capital city of Slovenia?
User: Ljubljana
> Game: This is correct!
```

Build this game, push the code to your GitHub and share it on the forum! :)

▶ Hint

Bonus: A problem with imports

▶ Click to see the bonus content.