TECHNION
Azrieli Continuing Education and External Studies Division

Module 5.7.1: Importing Libraries

Files

We learned how to deal with files.

Now we will learn how to deal with a *specific* type of file:

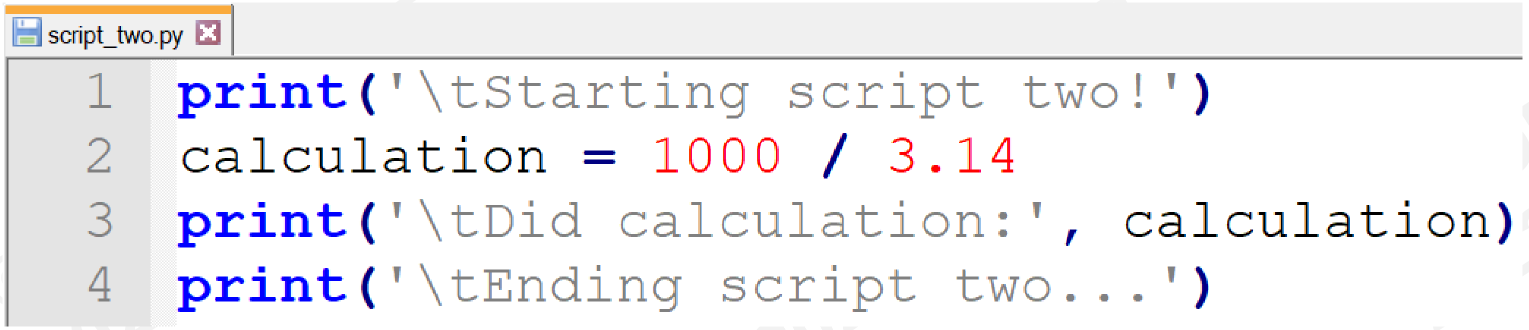
A ***Python*** file!

Python files are text files, filled with Python code.

`import`

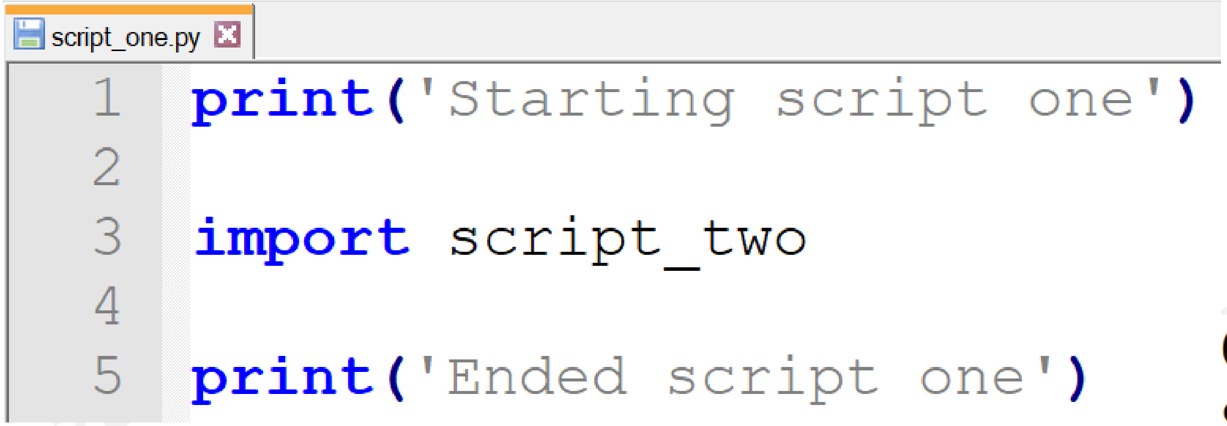
We want the ability to use (execute) the code in a different Python script.

We will do this by using the **`import`** statement.



script_two.py

```python
print('\tStarting script two!')
calculation = 1000 / 3.14
print('\tDid calculation:', calculation)
print('\tEnding script two...')
```



script_one.py

```python
print('Starting script one')

import script_two

print('Ended script one')
```

```
C:\>python C:\Python\script_one.py
Starting script one
        Starting script two!
        Did calculation: 318.471337
        Ending script two...
Ended script one
```

TECHNION
Azrieli Continuing Education and External Studies Division

Using `import`

When using **`import`**, the *whole file* is executed, line-by-line.

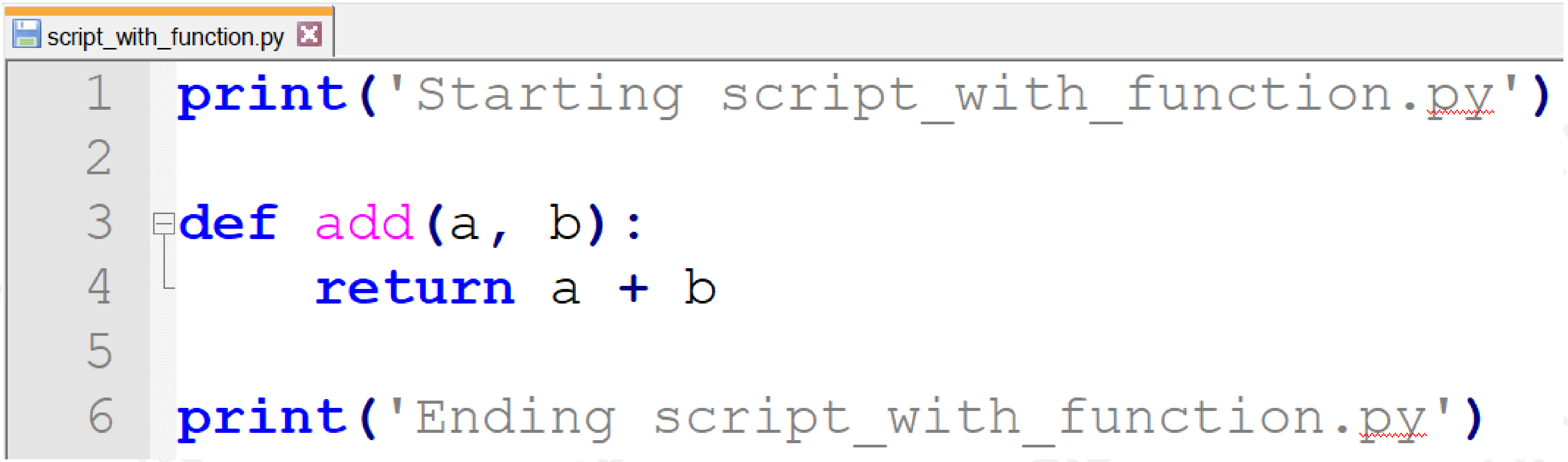
This means that the current script is paused until all lines of the other script have finished executing. Only then, the current script continues to run.

You can only import scripts that are in the *same directory* (folder) as your script.

Using Imported Functions

After importing, we can now use functions defined in our imported script!

We do this just the same as we use methods – by using a dot.



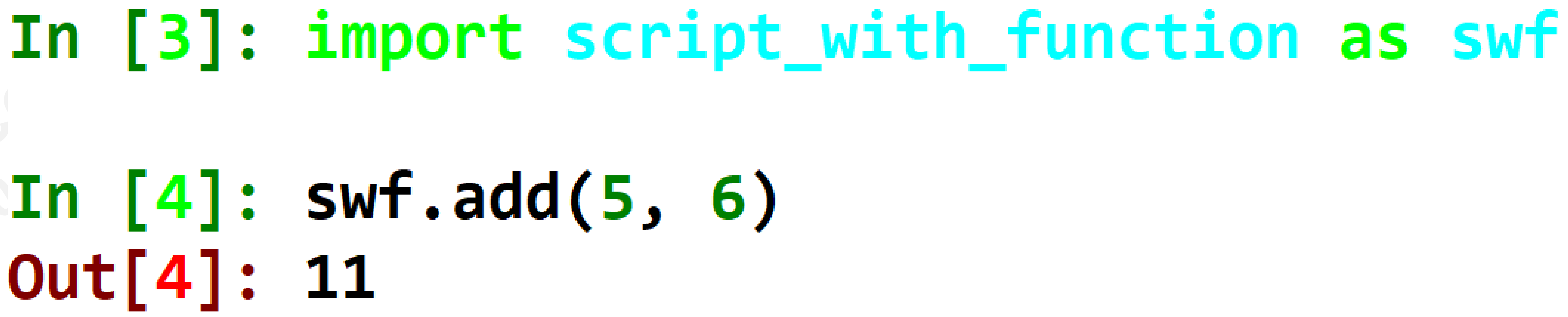
script_with_function.py

```python
print('Starting script_with_function.py')

def add(a, b):
    return a + b

print('Ending script_with_function.py')
```

```
In [1]: import script_with_function
Starting script_with_function.py
Ending script_with_function.py

In [2]: script_with_function.add(1, 2)
Out[2]: 3
```

Changing The Imported Script's Name

We can use the **as** keyword to give the imported script a nickname.

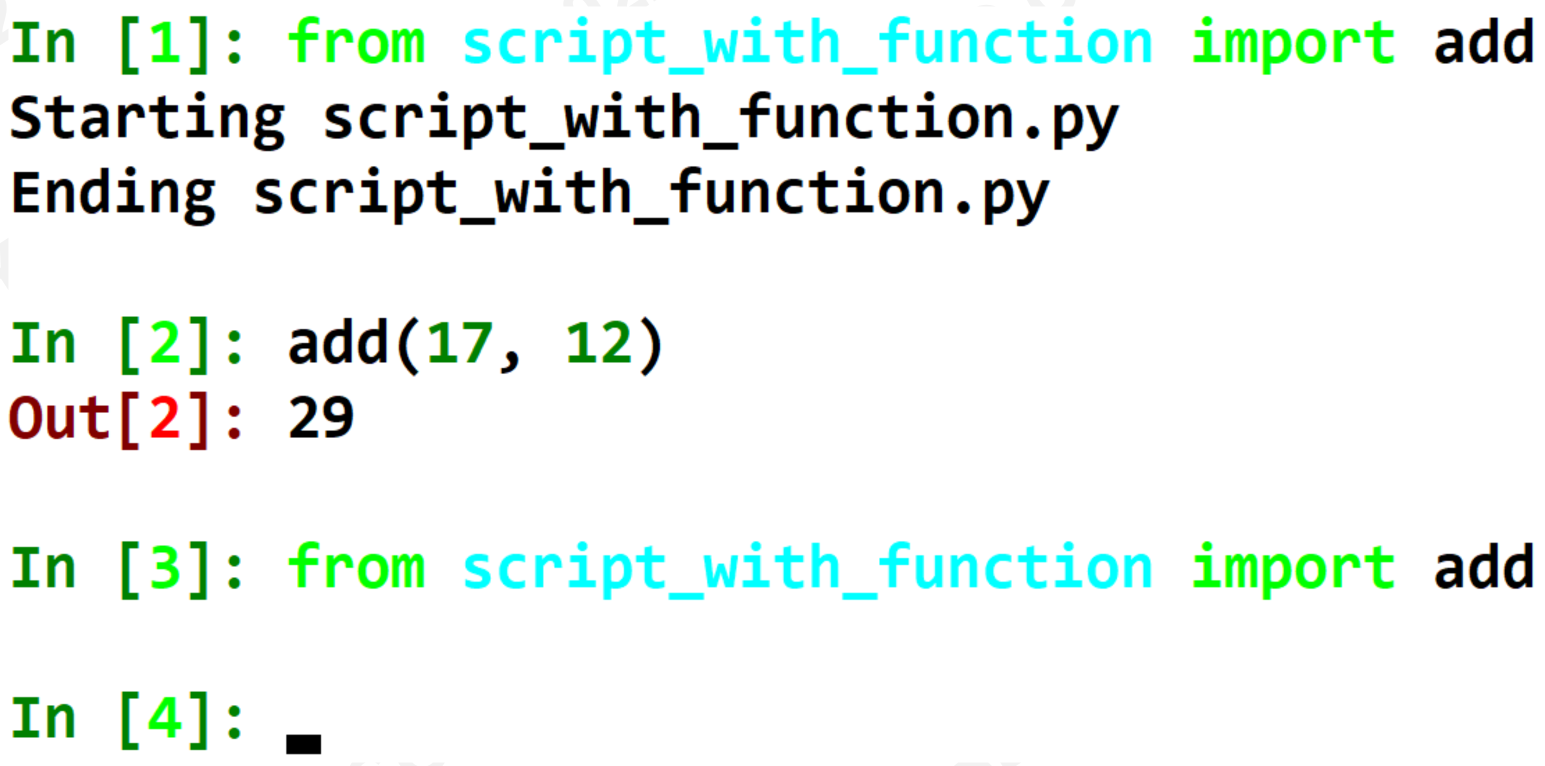


```
In [3]: import script_with_function as swf

In [4]: swf.add(5, 6)
Out[4]: 11
```

Importing a Single Function

We can use the **`from`** keyword to import a specific object from our script.

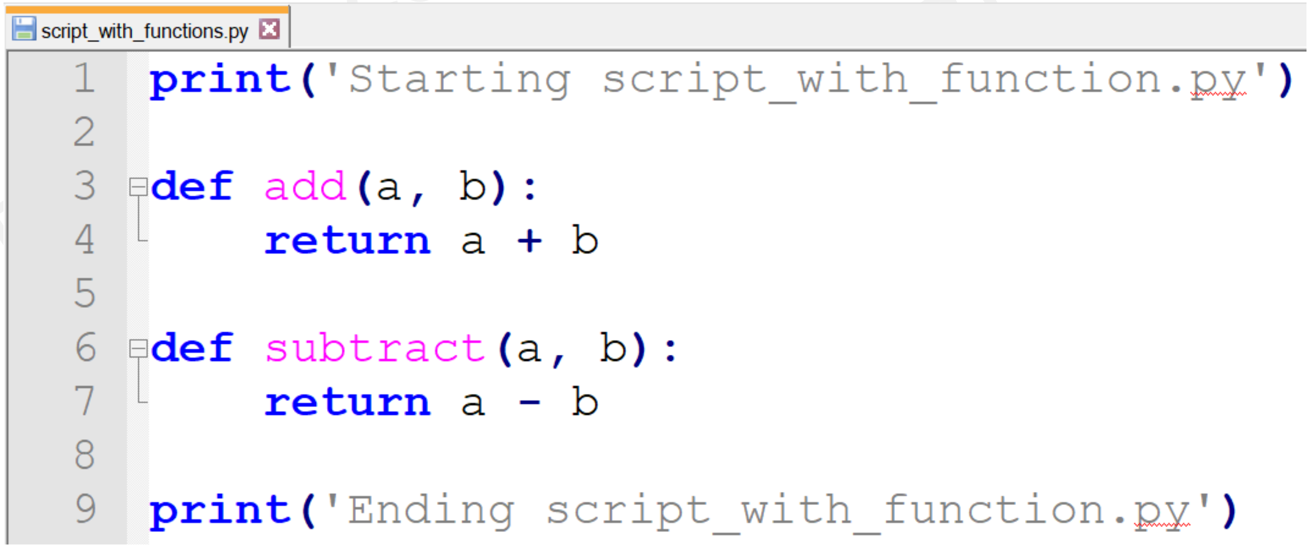
Once imported, you cannot import a script again. You will need to restart *ipython* to re-import a module.



```
In [1]: from script_with_function import add
Starting script_with_function.py
Ending script_with_function.py

In [2]: add(17, 12)
Out[2]: 29

In [3]: from script_with_function import add

In [4]: 
```

Importing Multiple Functions

Separate imported names with commas to import multiple items from your script.



script_with_functions.py

```
print('Starting script_with_function.py')

def add(a, b):
    return a + b

def subtract(a, b):
    return a - b

print('Ending script_with_function.py')
```

```
In [1]: from script_with_functions import add, subtract
Starting script_with_function.py
Ending script_with_function.py

In [2]: add(3, -3)
Out[2]: 0

In [3]: subtract(3, -3)
Out[3]: 6
```

Azrieli Continuing Education and External Studies Division

Using as with from

```
In [1]: from script_with_functions import add as plus, subtract as minus
Starting script_with_functions.py
Ending script_with_functions.py

In [2]: plus(4, 4)
Out[2]: 8

In [3]: minus(4, 4)
Out[3]: 0
```

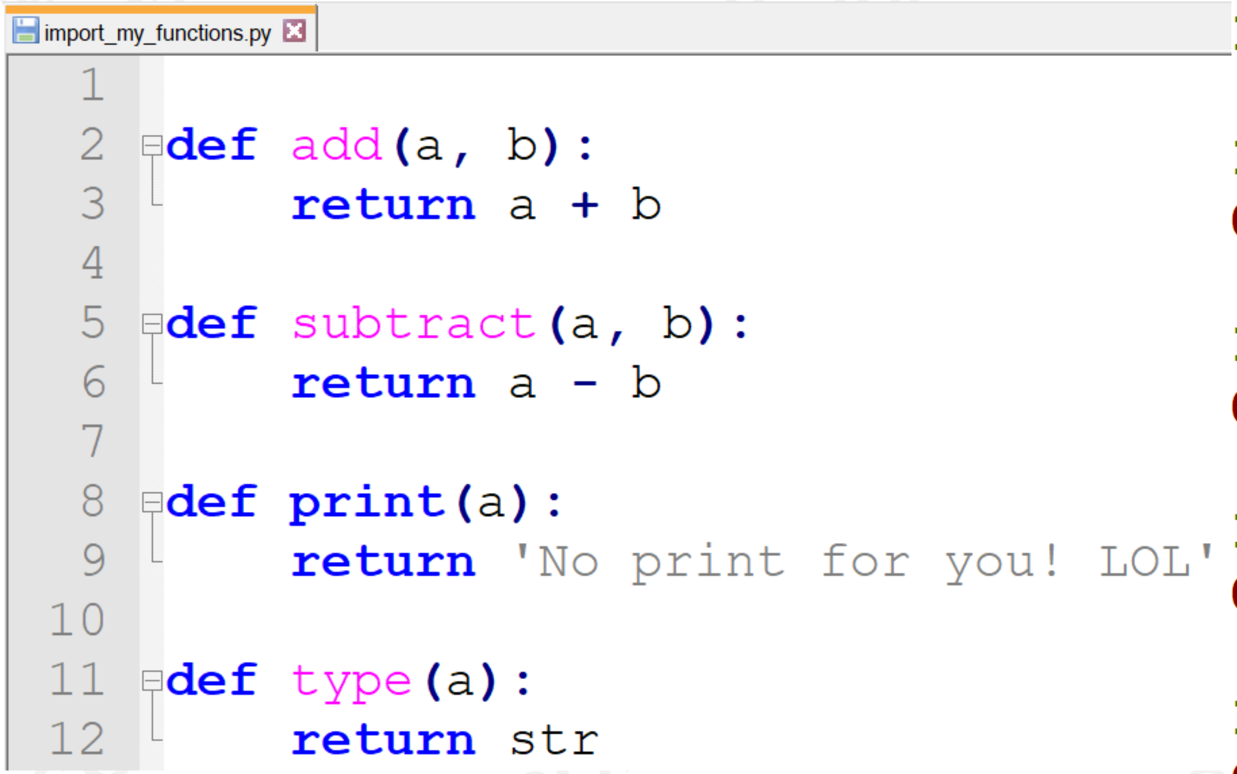
```
In [4]: add(1, 1)
-------------------------------------------

NameError

<ipython-input-4-d6a28b0c7f0b> in <module>
----> 1 add(1, 1)

NameError: name 'add' is not defined
```

from module_name import *

- Using * instead of specific function names imports all the functions in the other script.
- This sounds useful, but it is very bad to do.
- Let me explain!

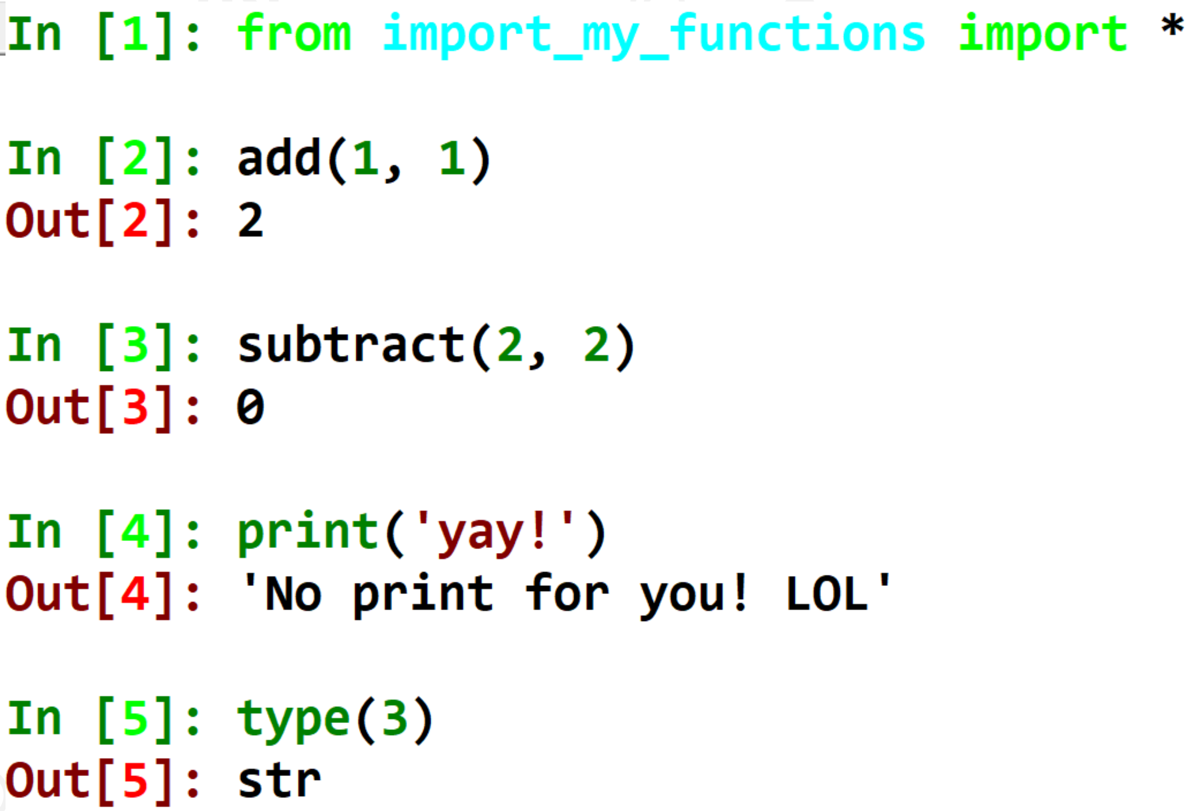
This is my script. I want to easily import *add* and *subtract*, so I use *.

The functions will override my own function! And I can't do anything about it!



import_my_functions.py

```python

def add(a, b):
    return a + b

def subtract(a, b):
    return a - b

def print(a):
    return 'No print for you! LOL'

def type(a):
    return str
```



```
In [1]: from import_my_functions import *

In [2]: add(1, 1)
Out[2]: 2

In [3]: subtract(2, 2)
Out[3]: 0

In [4]: print('yay!')
Out[4]: 'No print for you! LOL'

In [5]: type(3)
Out[5]: str
```

TECHNION
Azrieli Continuing Education and External Studies Division

So…

So never use “`from module import *`”!

What is __name__

Modules

__name__

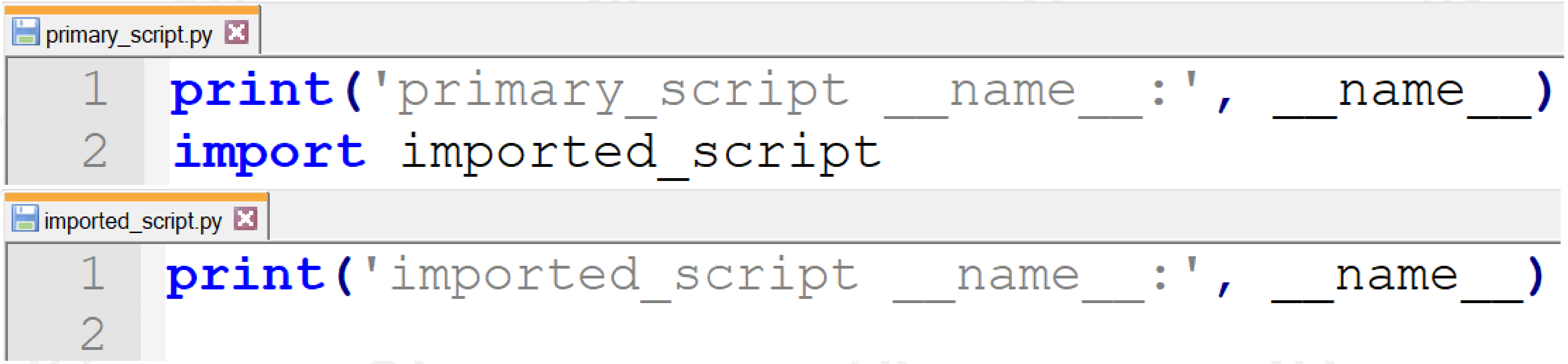
The variable __name__ is a built-in variable, that holds the name of the running script.

- If the script is running because the user decided to run it – __name__ will be "__main__", because it is the primary script the user decided to run.
- If the script is running because it is being *imported* from a different script, __name__ will be the script name.

Let’s print __name__ out.

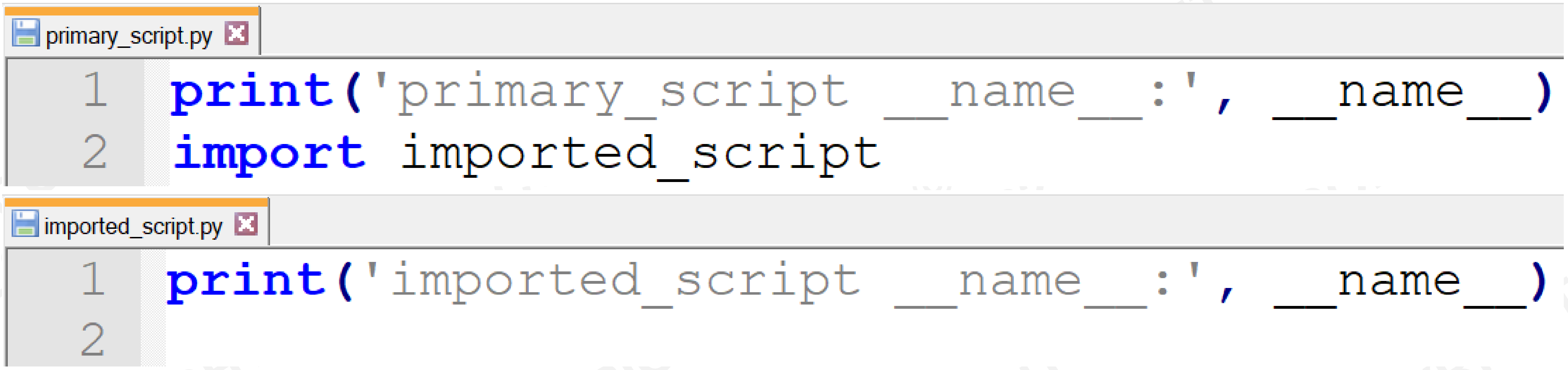
Once from a script that we will run directly.

Once from a script that will be run from a different script’s **import**.



```
print('primary_script __name__:', __name__)
import imported_script
```

```
print('imported_script __name__:', __name__)
```


primary_script.py

```python
print('primary_script __name__:', __name__)
import imported_script
```

imported_script.py

```python
print('imported_script __name__:', __name__)
```

```
C:\Python>python primary_script.py
primary_script __name__: __main__
imported_script __name__: imported_script
```

Why We Care

Let’s say I wrote a script.

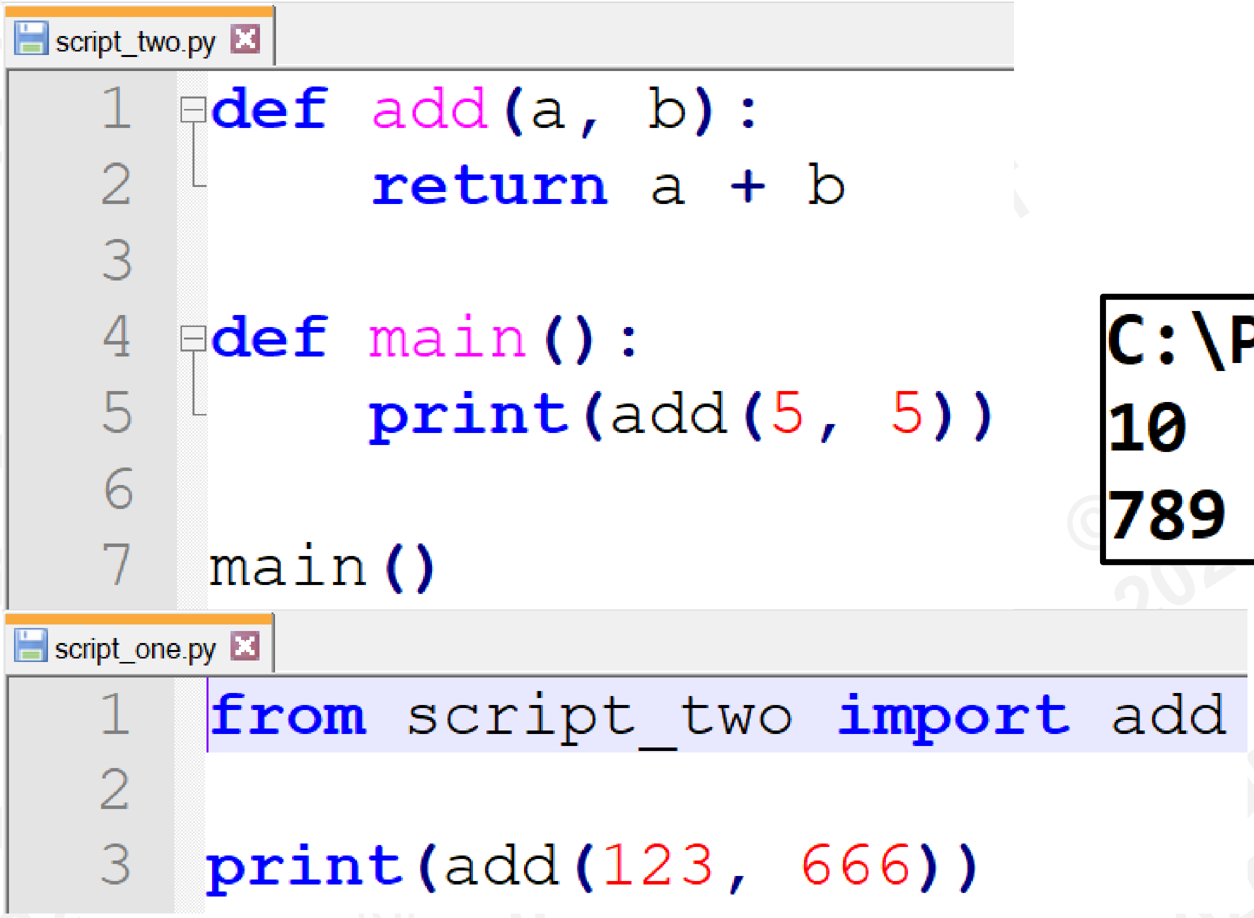
I wrote a function that is useful for my script, but it also can be useful for other scripts.

I use my function in my ***main*** function.

I want to give other scripts the ability to import my function and use it on their own, without running my ***main*** function (which will print and run things that are only important to the current function).

For this reason, only if __name__ is “__main__” (which means that only if the user specifically chose to run this script) will my ***main*** function activate.

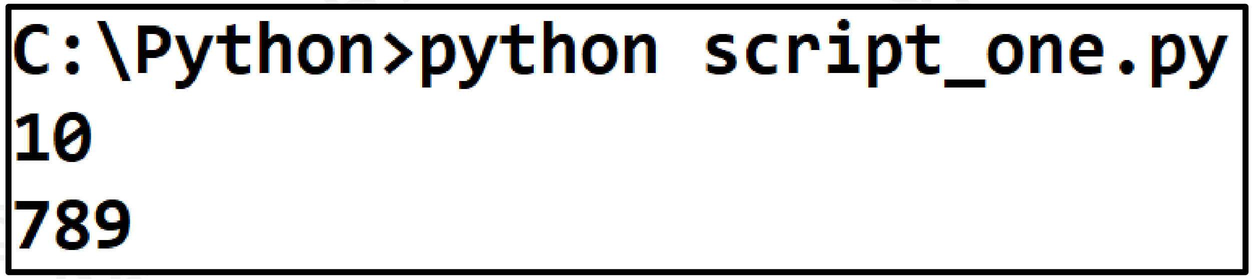
When you don't use `if __name__`...


script_two.py

```python
def add(a, b):
    return a + b

def main():
    print(add(5, 5))

main()
```

script_one.py

```python
from script_two import add

print(add(123, 666))
```



```
C:\Python>python script_one.py
10
789
```

Even though I only wanted to print the sum 789, the sum 10 was printed because main() was called in the other script.

TECHNION
Azrieli Continuing Education and External Studies Division

External Libraries

Modules

TECHNION
Azrieli Continuing Education and External Studies Division

External Libraries

There are many modules that are built in to Python. They are **VERY** useful. These make up what is called the “Standard Library”.

We can access them by using the **`import`** statement. We will learn how to use them in one of the coming lessons!

Laziness

Many programmers don’t learn these modules because they are too lazy to read about them. This is a big mistake.

10 minutes of reading about a module can save you hours and hours of programming!

You would never believe what absurd code programmers come up with, just because they didn’t know a simple, efficient function existed in one of the builtin modules of the Standard Library.