

Python for Data Science

Agenda

Introduction to Python

Intermediate Python

Python Data Science Toolbox

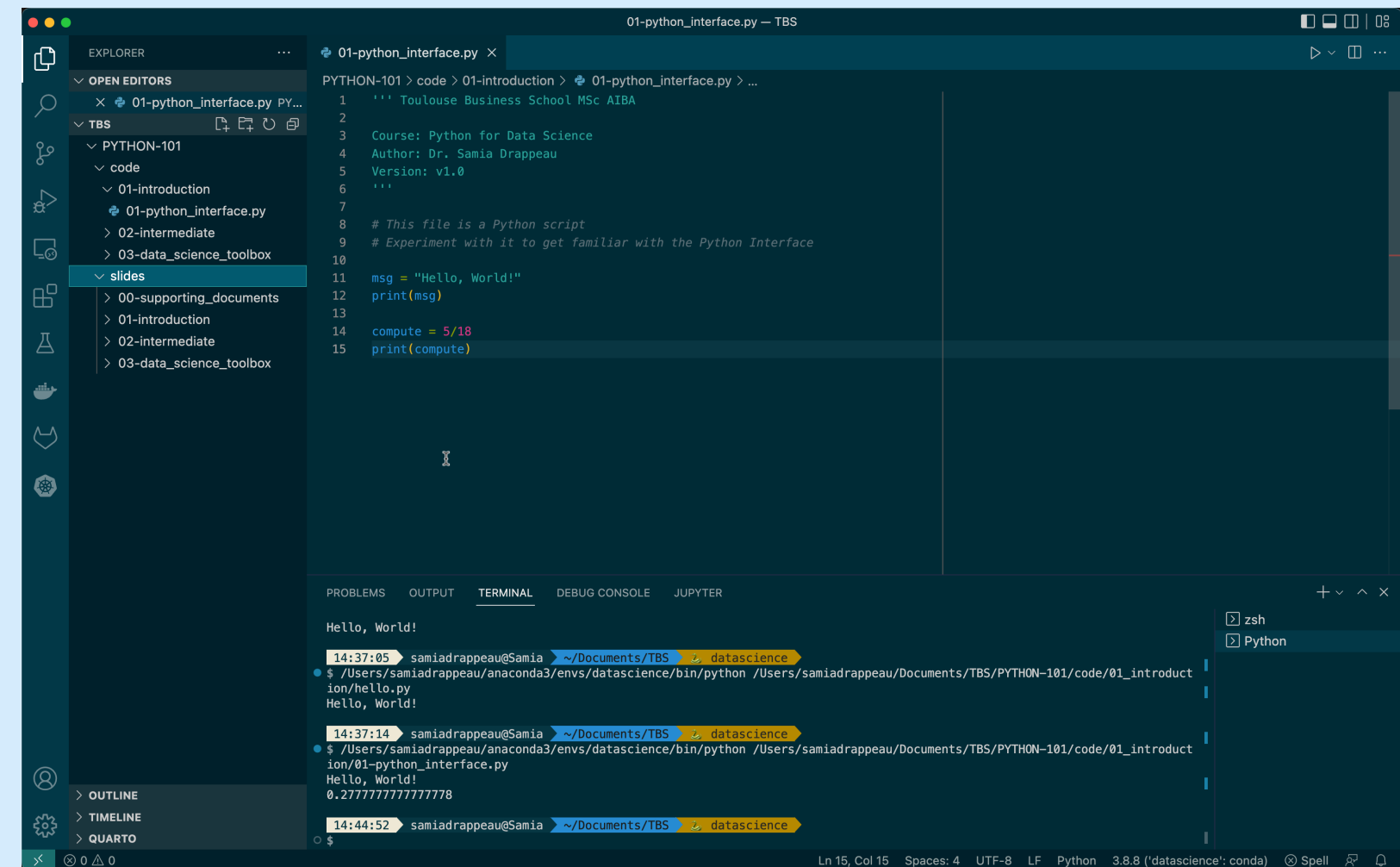


Expectations

This is a Hand-on course

Share question as we go

Slides and scripts will be available at the end of the lecture



The screenshot displays the Visual Studio Code (VS Code) interface. The Explorer panel on the left shows a project structure with folders 'PYTHON-101', 'code', and 'slides'. The 'code' folder is expanded, showing '01-python_interface.py'. The main editor window displays the content of '01-python_interface.py', which includes a docstring with course information and a Python script that prints 'Hello, World!' and calculates 5/18. The Terminal panel at the bottom shows the output of running the script, displaying 'Hello, World!' and the result of the division, 0.2777777777777778. The status bar at the bottom indicates the file is at line 15, column 15, using UTF-8 encoding, and is a Python file in a conda environment.

```
01-python_interface.py
1 ''' Toulouse Business School MSc AIBA
2
3 Course: Python for Data Science
4 Author: Dr. Samia Drappeau
5 Version: v1.0
6 '''
7
8 # This file is a Python script
9 # Experiment with it to get familiar with the Python Interface
10
11 msg = "Hello, World!"
12 print(msg)
13
14 compute = 5/18
15 print(compute)
```

```
14:37:05 samiadrappau@Samia ~/Documents/TBS
$ /Users/samiadrappau/anaconda3/envs/datascience/bin/python /Users/samiadrappau/Documents/TBS/PYTHON-101/code/01_introduction/hello.py
Hello, World!

14:37:14 samiadrappau@Samia ~/Documents/TBS
$ /Users/samiadrappau/anaconda3/envs/datascience/bin/python /Users/samiadrappau/Documents/TBS/PYTHON-101/code/01_introduction/01-python_interface.py
Hello, World!
0.2777777777777778

14:44:52 samiadrappau@Samia ~/Documents/TBS
$
```

Purpose



Objective 1

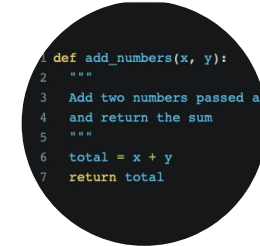
Master the basics of
data analysis

Series

	oranges
0	0
1	3
2	7
3	2

Objective 2

Create data
visualisation &
manipulating
DataFrame



Objective 3

Write your own
function

Introduction to Python

Programming Basic

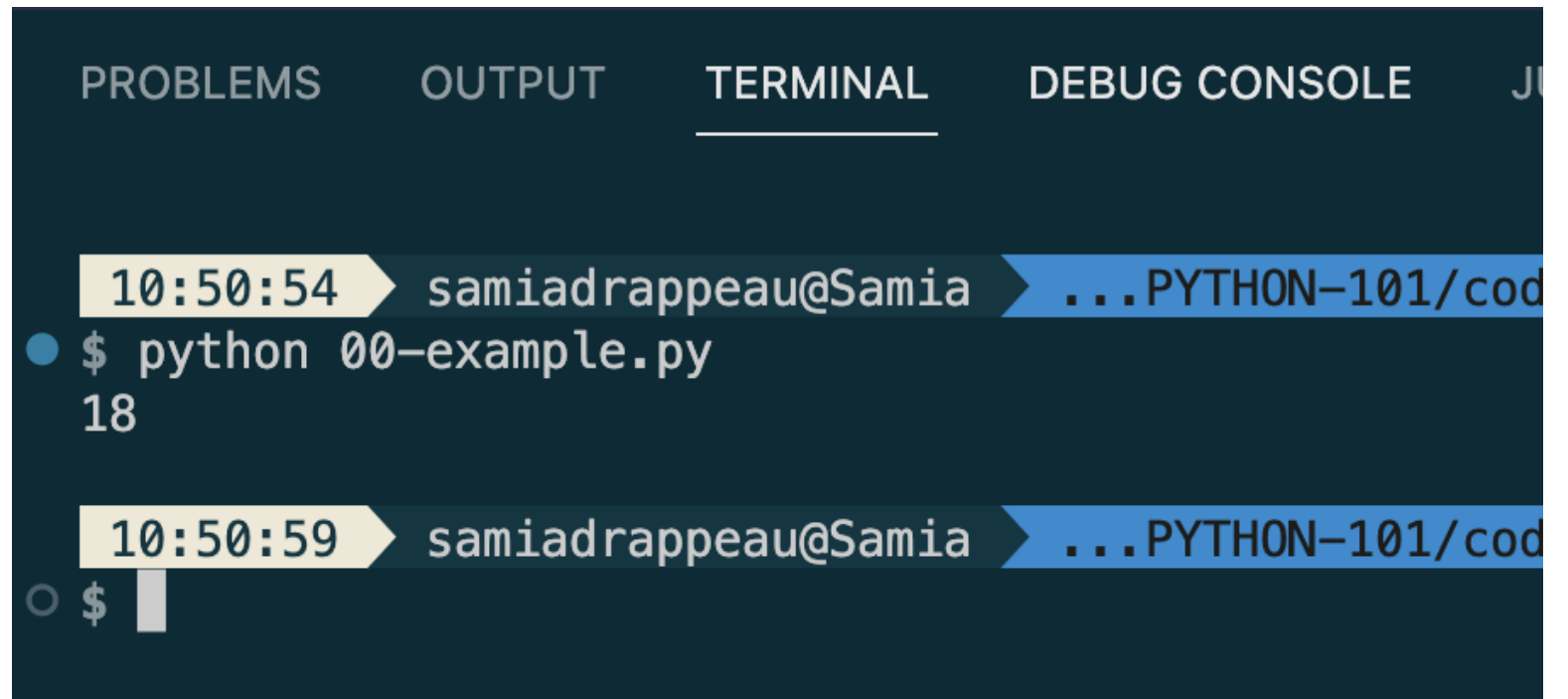
Source code

Syntax

Output

Console

```
17 def main():
18     x = 10
19     y = 8
20     sum = add_numbers(x, y)
21     print(sum)
22
23 if __name__ == '__main__':
24     main()
```



The screenshot shows a terminal window with a dark blue background. At the top, there are tabs labeled 'PROBLEMS', 'OUTPUT', 'TERMINAL' (which is active and underlined), 'DEBUG CONSOLE', and 'Jupyter'. Below the tabs, there are two terminal sessions. The first session starts with a timestamp '10:50:54' and the user 'samiadrappeau@Samia'. The prompt is '...PYTHON-101/cod'. The command entered is '\$ python 00-example.py', and the output is '18'. The second session starts with a timestamp '10:50:59' and the same user. The prompt is '...PYTHON-101/cod'. The command entered is '\$', and the cursor is visible on the next line.

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE  Jupyter

10:50:54 samiadrappeau@Samia ...PYTHON-101/cod
● $ python 00-example.py
18

10:50:59 samiadrappeau@Samia ...PYTHON-101/cod
○ $
```

Programming Basic

Python script

- It is a text file with extension ".py"
- It has a list of Python commands
- Use "print()" to generate output from script

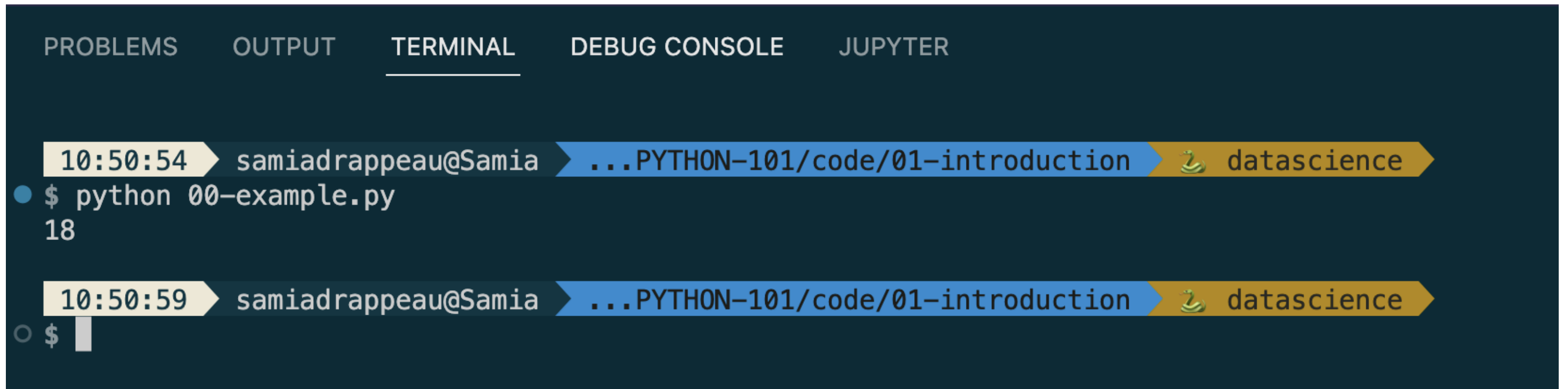
```
00-example.py

1 ''' Toulouse Business School MSc AIBA
2
3 Course: Python for Data Science
4 Author: Dr. Samia Drappeau
5 Version: v1.0
6 '''
7
8 def add_numbers(x, y):
9     """
10     Add two numbers passed as arguments
11     and return the sum
12     """
13     total = x + y
14     return total
15
16
17 def main():
18     x = 10
19     y = 8
20     sum = add_numbers(x, y)
21     print(sum)
22
23 if __name__ == '__main__':
24     main()
```

Programming Basic

Python execution

- Python is a *interpreted* language
- Command is:
> *python script.py*



The screenshot shows a terminal window with a dark blue background. At the top, there are five tabs: PROBLEMS, OUTPUT, TERMINAL (which is active and underlined), DEBUG CONSOLE, and JUPYTER. Below the tabs, there are two terminal sessions. The first session starts at 10:50:54, with the user 'samiadrappeau@Samia' in the directory '...PYTHON-101/code/01-introduction'. The user runs the command '\$ python 00-example.py', and the output is '18'. The second session starts at 10:50:59, with the same user and directory. The prompt '\$' is visible, but no command has been entered yet.

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE  JUPYTER

10:50:54 samiadrappeau@Samia ...PYTHON-101/code/01-introduction datascience
• $ python 00-example.py
18

10:50:59 samiadrappeau@Samia ...PYTHON-101/code/01-introduction datascience
○ $
```


- 1- Open VSC
- 2- Open *PYTHON-101/code/01-introduction/01-python_interface.py*
- 3- Play with it

Let's practice

Variables & Types

Variables

- Definition is done with specific, case-sensitive name
- Call up its value by typing the variable name

04-variable.py

```
1 ''' Toulouse Business School MSc AIBA
2
3 Course: Python for Data Science
4 Author: Dr. Samia Drappeau
5 Version: v1.0
6 '''
7
8 """
9 Without Variables
10 """
11 # How much is your $100 worth after 7 years?
12 print(100 * 1.1**7)
13
14
15 """
16 With Variables
17 """
18 initial_invest = 100
19 roi = 1.1 #10%
20 duration = 7
21 print(initial_invest * roi**duration)
22 # > 194.87
```

Variables & Types

Reproducibility

04-variable.py

```
1 ''' Toulouse Business School MSc AIBA
2
3 Course: Python for Data Science
4 Author: Dr. Samia Drappeau
5 Version: v1.0
6 '''
7
8 """
9 Without Variables
10 """
11 # How much is your $100 worth after 7 years?
12 print(100 * 1.1**7)
13
14
15 """
16 With Variables
17 """
18 initial_invest = 100
19 roi = 1.1 #10%
20 duration = 7
21 print(initial_invest * roi**duration)
22 # > 194.87
```

Variables & Types

Reproducibility

```
14
15 """
16 With Variables
17 """
18 initial_invest = 100
19 roi = 1.2 #20% <---
20 duration = 7
21 print(initial_invest * roi**duration)
22 # > 358.32
```

04-variable.py

```
1 ''' Toulouse Business School MSc AIBA
2
3 Course: Python for Data Science
4 Author: Dr. Samia Drappeau
5 Version: v1.0
6 '''
7
8 """
9 Without Variables
10 """
11 # How much is your $100 worth after 7 years?
12 print(100 * 1.1**7)
13
14
15 """
16 With Variables
17 """
18 initial_invest = 100
19 roi = 1.1 #10%
20 duration = 7
21 print(initial_invest * roi**duration)
22 # > 194.87
```

Variables & Types

Python Type

- Numbers: *float* or *int*
- Text: *str*
- Boolean: *bool*
- Different type makes different behaviour for a same operator

```
18 initial_invest = 100
19 roi = 1.2 #20% <---
20 duration = 7
21 print(initial_invest * roi**duration)
22 # > 358.32
23
24 type(roi)
25 # > float
26
27 type(duration)
28 # > int
29
30 text1 = "Return on Investment"
31 text2 = 'This works too'
32 type(text1), type(text2)
33 #> str, str
34
35 is_ok = False
36 type(is_ok)
37 #> bool
```

- 1- Open VSC
- 2- Open *PYTHON-101/code/01-introduction/04-variables.py*
- 3- Play with it

Let's practice

Python List

- Name a collection of values
- Can contain any type
- Can contain different types
- Can even contain a list!
- Has specific functionality & behaviour



00-list.py

```
1 ''' Toulouse Business School MSc AIBA
2
3 Course: Python for Data Science
4 Author: Dr. Samia Drappeau
5 Version: v1.0
6 '''
7
8 saving1 = 100
9 saving2 = 200
10 saving3 = 500
11 saving4 = 1000
12
13 client_savings = [100, 200, 500, 1000]
14
15 # Different types of variable
16 client_savings = ["client1", 100, "client2", 200, "client3", 500, "client4", 1000]
17
18 # List of lists
19 client_savings2 = [
20     ["client1", 100],
21     ["client2", 200],
22     ["client3", 500],
23     ["client4", 1000]
24 ]
25
26 type(client_savings)
27 # > list
28
29 type(client_savings2)
30 # > list
```

- 1- Open VSC
- 2- Open *PYTHON-101/code/01-introduction/05-list.py*
- 3- Play with it

Let's practice

Python List

Subsetting lists

- use *index* to access a value of the list
- *index* starts at **0**
- **-1** accesses the last element of the list

```
1 ''' Toulouse Business School MSc AIBA
2
3 Course: Python for Data Science
4 Author: Dr. Samia Drappeau
5 Version: v1.0
6 '''
7
8 client_savings = [100, 200, 500, 1000]
9 print(client_savings)
10 # > [100, 200, 500, 1000]
11
12 print(client_savings[2])
13 #> 500
14
15 print(client_savings[-1])
16 #> 1000
17
18 print(client_savings[3])
19 #> 1000
```

Python List

List slicing

- [start:end]
- *start* is **inclusive**
- *end* is **exclusive**
- *start* & *end* can be omitted



07-slicelist.py

```
1 ''' Toulouse Business School MSc AIBA
2
3 Course: Python for Data Science
4 Author: Dr. Samia Drappeau
5 Version: v1.0
6 '''
7
8 client_savings = [100, 200, 500, 1000, 50000]
9 print(client_savings[1:3])
10 #> [200, 500]
11
12 print(client_savings[:2])
13 #> [100, 200]
14
15 print(client_savings[1:])
16 #> [200, 500, 1000, 50000]
```

- 1- Open VSC
- 2- Open *PYTHON-101/code/01-introduction/06-sublist.py*
- 3- Play with it

Let's practice

Python List

Manipulation

- change element value
- add a new element
- remove a element

```
8 client_savings = [100, 200, 500, 1000, 50000]
9 print(client_savings)
10 # > [100, 200, 500, 1000, 50000]
11
12 # CHANGING VALUE
13 client_saving[3] = 2000
14 print(client_savings)
15 #> [100, 200, 500, 2000, 50000]
16
17 client_saving[0:2] = [300, 600]
18 print(client_savings)
19 #> [300, 600, 500, 2000, 50000]
20
21
22
23 # ADD ELEMENT
24 new_client_savings = client_savings + [2345, 50090]
25 print(new_client_savings)
26 #> [200, 500, 1000, 50000, 2345, 50090]
27
28
29 # REMOVE ELEMENT
30 del new_client_savings[2]
31 print(new_client_savings)
32 #> [200, 500, 50000, 2345, 50090]
```

- 1- Open VSC
- 2- Open *PYTHON-101/code/01-introduction/07-list_manipulation.py*
- 3- Play with it

Let's practice

Python Function

- Nothing new
- *type()*
- It's a piece of reusable code
- It solves a particular task
- Call functions instead of writing code yourself

Python Function

- max()
- min()
- round()
- and many others!

```
8 client_savings = [100, 200, 500, 1000, 50000]
9
10 # Function: max()
11 max(client_savings)
12 # > 50000
13
14 # Function: round()
15 round(1.86, 1)
16 # > 1.9
17
18 # Documentation of a function: help()
19 help(round)
```

Help on built-in function round in module builtins:

`round(number, ndigits=None)`

Round a number to a given precision in decimal digits.

The return value is an integer if ndigits is omitted or None.

Otherwise the return value has the same type as the number. ndigits may be negative.

Python Function

How to find a function?

- standard task probably already has a function
- ***Internet is your friend***

```
8 client_savings = [100, 200, 500, 1000, 50000]
9
10 # Function: max()
11 max(client_savings)
12 # > 50000
13
14 # Function: round()
15 round(1.86, 1)
16 # > 1.9
17
18 # Documentation of a function: help()
19 help(round)
```

Help on built-in function round in module builtins:

```
round(number, ndigits=None)
```

Round a number to a given precision in decimal digits.

The return value is an integer if ndigits is omitted or None.

Otherwise the return value has the same type as the number. ndigits may be negative.

- 1- Open VSC
- 2- Open *PYTHON-101/code/01-introduction/08-function.py*
- 3- Play with it

Let's practice

Python Method

- Methods are functions that are specific to Python objects
- Objects are: float, string, list, etc

	type	examples of methods
Object	str	capitalize() replace()
Object	float	bit_length() conjugate()
Object	list	index() count()

Python Method

List methods

```
8 client_savings = [100, 200, 500, 100, 50000]
9
10 # List Method: index()
11 client_savings.index(200)
12 # > 1
13
14 # List Method: count()
15 client_savings.count(100)
16 # > 2
```

Python Method

String methods

```
8 name = 'liz'
9
10 # Str Method: capitalize()
11 name.capitalize()
12 # > 'Liz'
13
14 # Str Method: replace()
15 name.replace("z", "sa")
16 # > 'lisa'
```

- 1- Open VSC
- 2- Open *PYTHON-101/code/01-introduction/09-method.py*
- 3- Play with it

Let's practice

Python Package

- Package is a directory of Python scripts
- Each script of a package is called a module
- Modules specify functions, methods and new Python types aimed at solving particular problems
- Packages for data science:
 - **NumPy** to efficiently work with arrays
 - **Pandas** for data analysis
 - **Matplotlib** for data visualization
 - **scikit-learn** for machine learning
- Not all these packages are available in Python by default

Python Package

Install package

- <https://pip.pypa.io/en/stable/installation/>
- Command to install a given package:
pip install <package>
example: *pip install numpy*

Python Package

Import package


- *import numpy*

or

import numpy as np ← Better!

Python Package

Import module

- *from numpy import array*
then, in code:
array()
or
- *import numpy as np*
then, in code:
np.array()  Better!

- 1- Open VSC
- 2- Open *PYTHON-101/code/01-introduction/10-packages.py*
- 3- Play with it

Let's practice

NumPy

- Numeric Python
- Package that, among others, provides an alternative to the regular python list: the **NumPy array**

```
import numpy as np
height = [1.85, 1.79, 1.67]
np_height = np.array(height)
```

NumPy

```
import numpy as np
height = [1.73, 1.68, 1.71, 1.89, 1.79]
np_height = np.array(height)
```

```
weight = [65.4, 59.2, 63.6, 88.4, 68.7]
np_weight = np.array(weight)
```

```
bmi = np_weight/np_height**2
bmi
# > array([21.85171573, 20.97505669, 21.75028214,
24.7473475 , 21.44127836])
```

NumPy

If we use list instead of NumPy Array

```
weight/height**2  
# > TypeError: unsupported operand type(s) for ** or  
pow(): 'list' and 'int'
```

NumPy

Subsetting

```
bmi  
# > array([21.85171573, 20.97505669, 21.75028214,  
24.7473475 , 21.44127836])
```

```
bmi > 23  
# > array([False, False, False,  True, False])
```

```
bmi[bmi>23]  
# > array([24.7473475])
```

- 1- Open VSC
- 2- Open *PYTHON-101/code/01-introduction/11-numpy.py*
- 3- Play with it

Let's practice

2D-NumPy

```
import numpy as np
np_height = np.array([1.73, 1.68, 1.71, 1.89, 1.79])
np_weight = np.array([65.4, 59.2, 63.6, 88.4, 68.7])
```

```
type(np_height)
# > numpy.ndarray
```

```
type(np_weight)
# > numpy.ndarray
```


2D-NumPy

```
np_2d = np.array([[1.73, 1.68, 1.71, 1.89, 1.79],  
                  [65.4, 59.2, 63.6, 88.4, 68.7]])  
  
np_2d.shape  
# > (2, 5) # 2 rows, 5 columns
```

2D-NumPy

Subsetting

	0	1	2	3	4	
array([[1.73,	1.68,	1.71,	1.89,	1.79],	0
[65.4,	59.2,	63.6,	88.4,	68.7]])	1

```
np_2d[0]
```

```
# > array([1.73, 1.68, 1.71, 1.89, 1.79]) # 1st row
```

2D-NumPy

Subsetting

```
      0      1      2      3      4
array([[ 1.73,  1.68,  1.71,  1.89,  1.79],  0
       [ 65.4,  59.2,  63.6,  88.4,  68.7]])  1
```

```
np_2d[:, 1:3]
# > array([[ 1.68,  1.71],
          [ 59.2 ,  63.6 ]]) # All the rows, columns 2 and 3
```

index starts at 0!

- 1- Open VSC
- 2- Open *PYTHON-101/code/01-introduction/12-2D_numpy.py*
- 3- Play with it

Let's practice

NumPy: basic statistics

Data Analysis

```
import numpy as np
np_city = ... # Implementation left out; 5000 rows, 2
columns
np_city
```

```
array([[1.64, 71.78],
       [1.37, 63.35],
       [1.6 , 55.09],
       ...,
       [2.04, 74.85],
       [2.04, 62.52],
```

NumPy: basic statistics

```
np.mean(np_city[:, 0])  
# > 1.7472
```

```
np.median(np_city[:, 0])  
# > 1.75
```

NumPy: basic statistics

```
np.corrcoef(np_city[:, 0], np_city[:, 1])  
# > array([[ 1.          , -0.01802],  
           [-0.01803,  1.          ]])
```

```
np.std(np_city[:, 0])  
# > 0.1992
```

NumPy: basic statistics

- *sum()*
- *sort()*
- and many other!
- You can even generate data with NumPy:

np.random.normal()

standard dev

mean

nb of samples

```
height = np.round(np.random.normal(1.75, 0.20, 5000), 2)
weight = np.round(np.random.normal(60.32, 15, 5000), 2)
np_city = np.column_stack((height, weight))
```


- 1- Open VSC
- 2- Open *PYTHON-101/code/01-introduction/13-numpy_stats.py*
- 3- Play with it

Let's practice