

Python for Data Science

Agenda

Introduction to Python

Intermediate Python

Python Data Science Toolbox

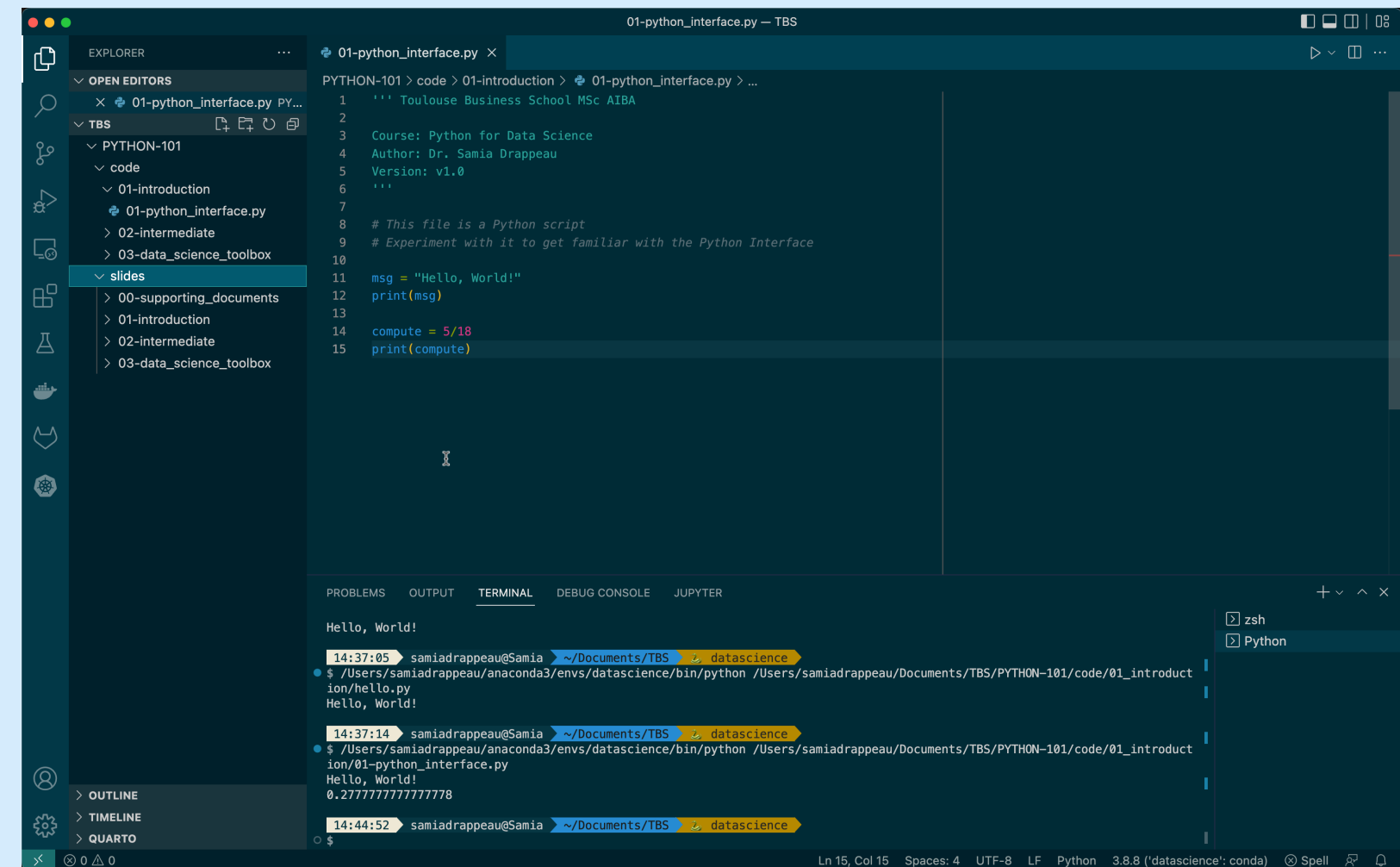


Expectations

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 for 'PYTHON-101', 'code', '01-introduction', '02-intermediate', '03-data_science_toolbox', and 'slides'. The '01-python_interface.py' file is open in the editor. The code in the file is as follows:

```
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)
```

The TERMINAL panel at the bottom shows the output of the script execution:

```
Hello, World!
14:37:05 samiadrappau@Samia ~/Documents/TBS data science
$ /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 data science
$ /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 data science
$
```

The status bar at the bottom indicates the current file is '01-python_interface.py' at line 15, column 15, using UTF-8 encoding, LF line endings, and Python 3.8.8.

Purpose



Objective 1

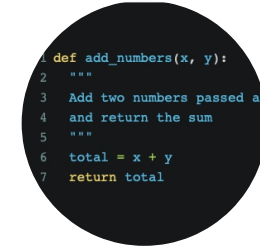
Master the basics of
data analysis

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

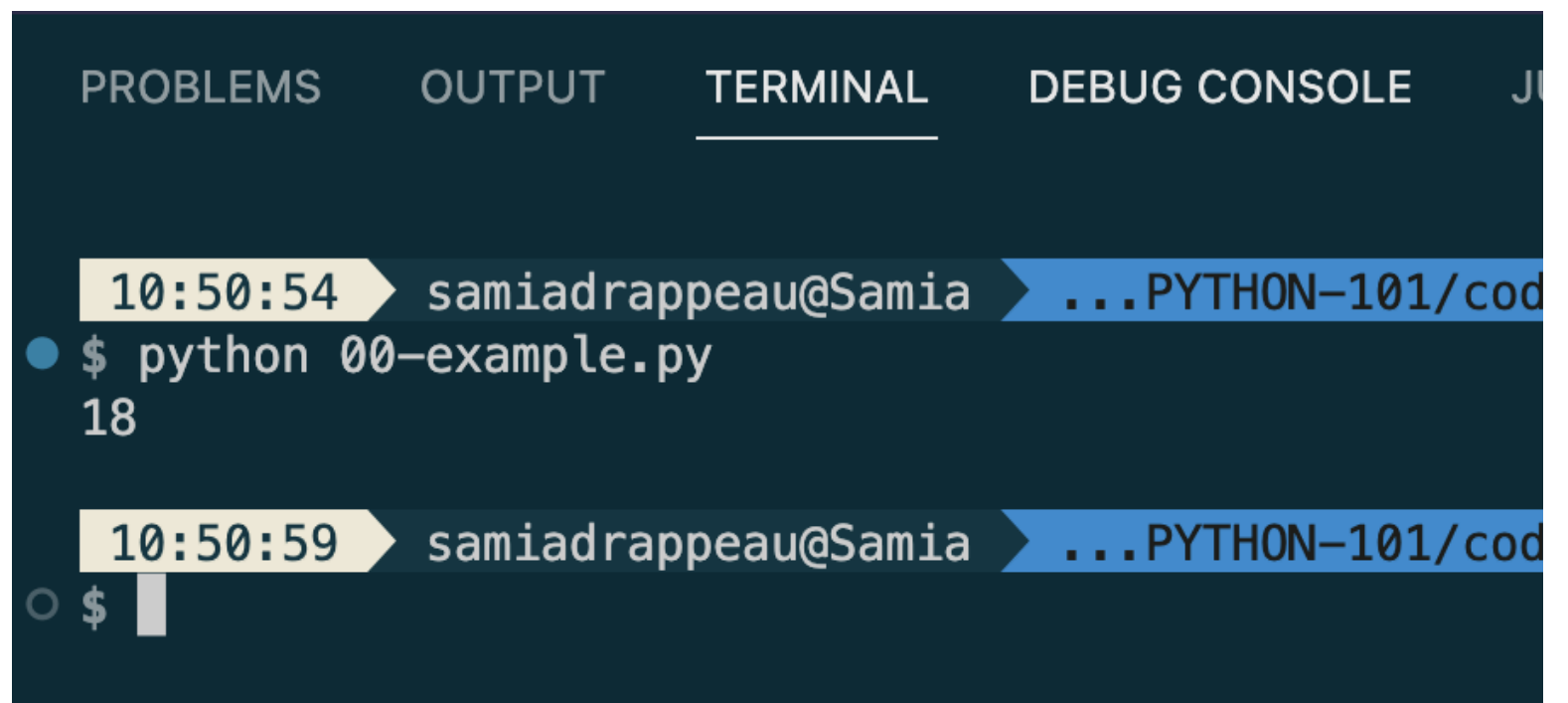
Source code

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 shows a timestamp '10:50:54', the username 'samiadrappeau@Samia', and the directory '...PYTHON-101/cod'. It shows a command '\$ python 00-example.py' being executed, followed by the output '18'. The second session shows a timestamp '10:50:59', the same username and directory, and a command '\$' followed by a cursor, indicating it is ready for input.

```
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

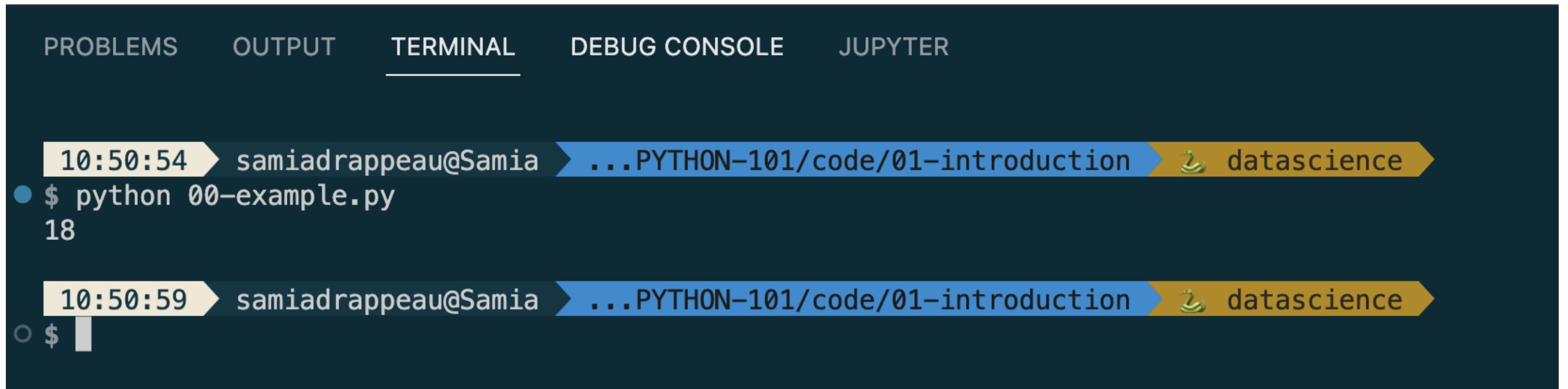
```
00-example.py

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 underlined), DEBUG CONSOLE, and JUPYTER. Below the tabs, there are two terminal sessions. The first session starts with a timestamp '10:50:54', a username 'samiadrappeau@Samia', and a path '...PYTHON-101/code/01-introduction'. It shows a command prompt '\$' followed by the command 'python 00-example.py' and the output '18'. The second session starts with a timestamp '10:50:59', the same username and path, and shows a command prompt '\$' followed by a cursor. On the right side of the terminal, there is a yellow arrow pointing right with a Python logo icon and the text 'datascience'.

```
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

```
array([[ 1.73,  1.68,  1.71,  1.89,  1.79],  
       [ 65.4,  59.2,  63.6,  88.4,  68.7]])
```

	0	1	2	3	4	
0	1.73	1.68	1.71	1.89	1.79	0
1	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],
       [0.04, 60.50],
```

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

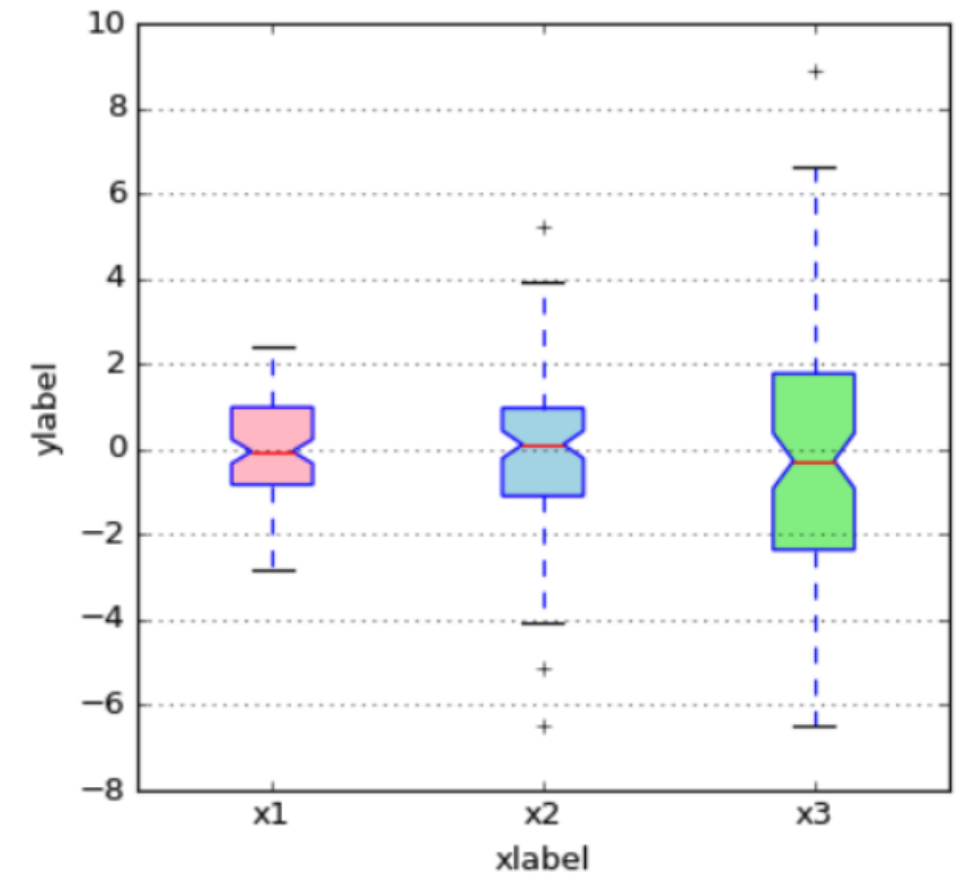
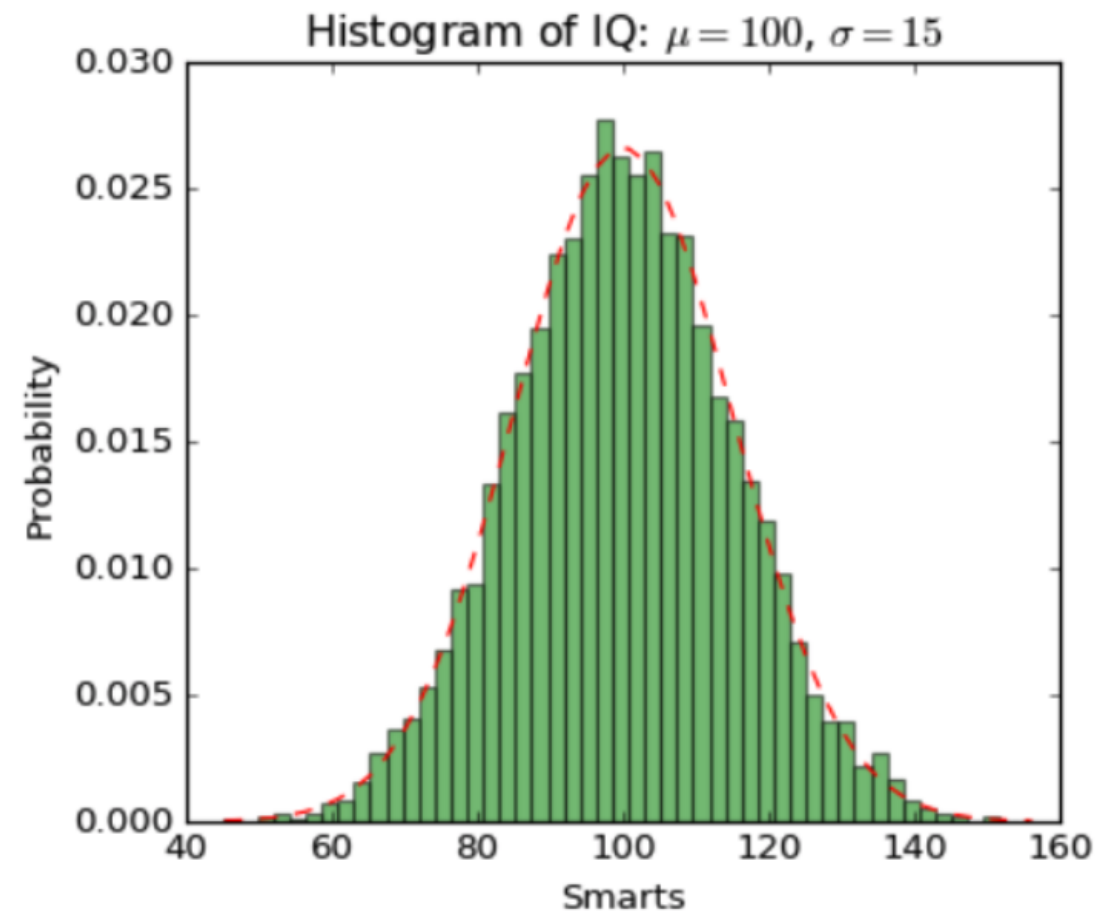
Let's practice

Intermediate Python

Matplotlib

Data Visualization

- Very important in Data Analysis
 - Explore data
 - Report insights

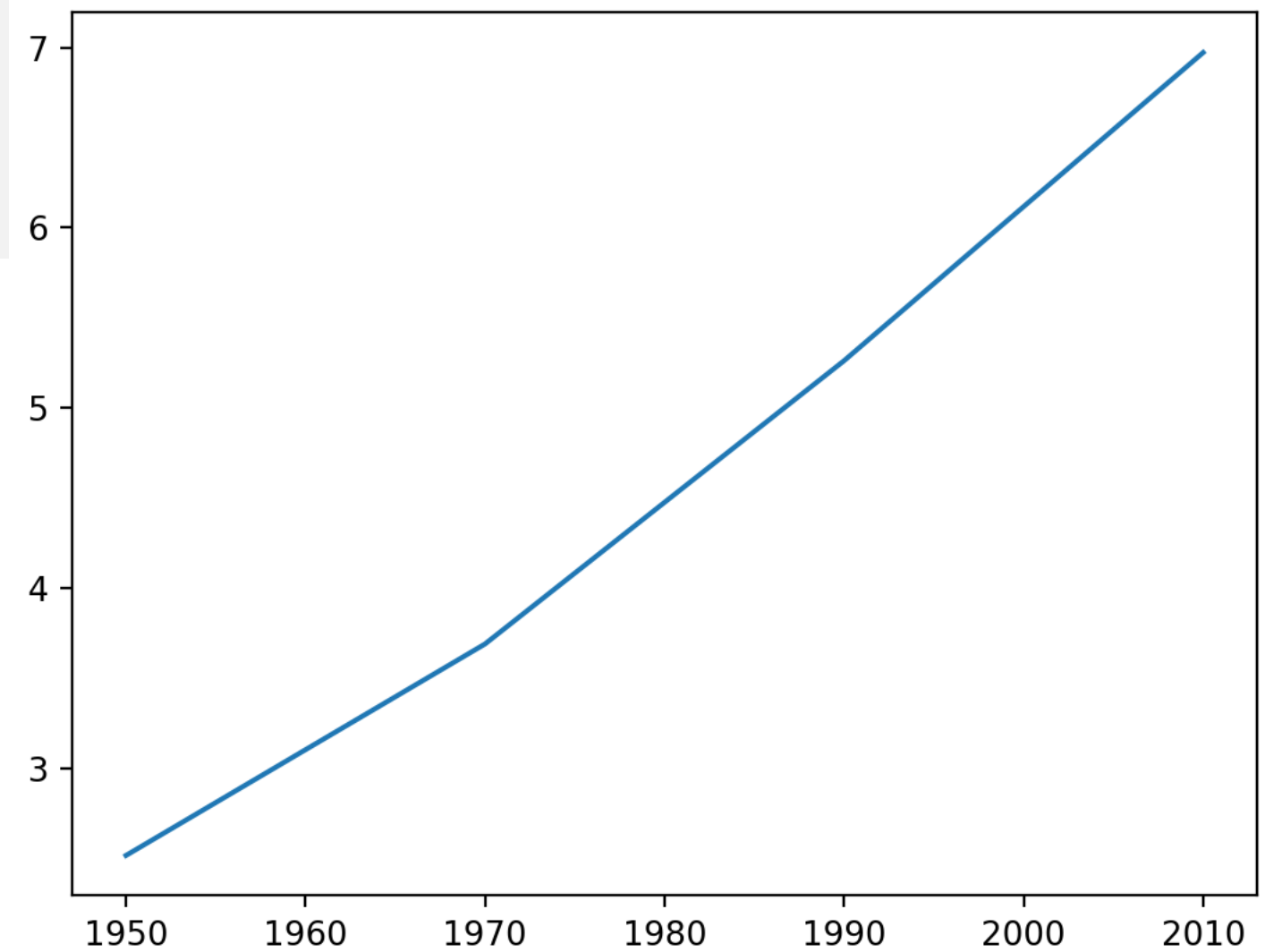


Matplotlib

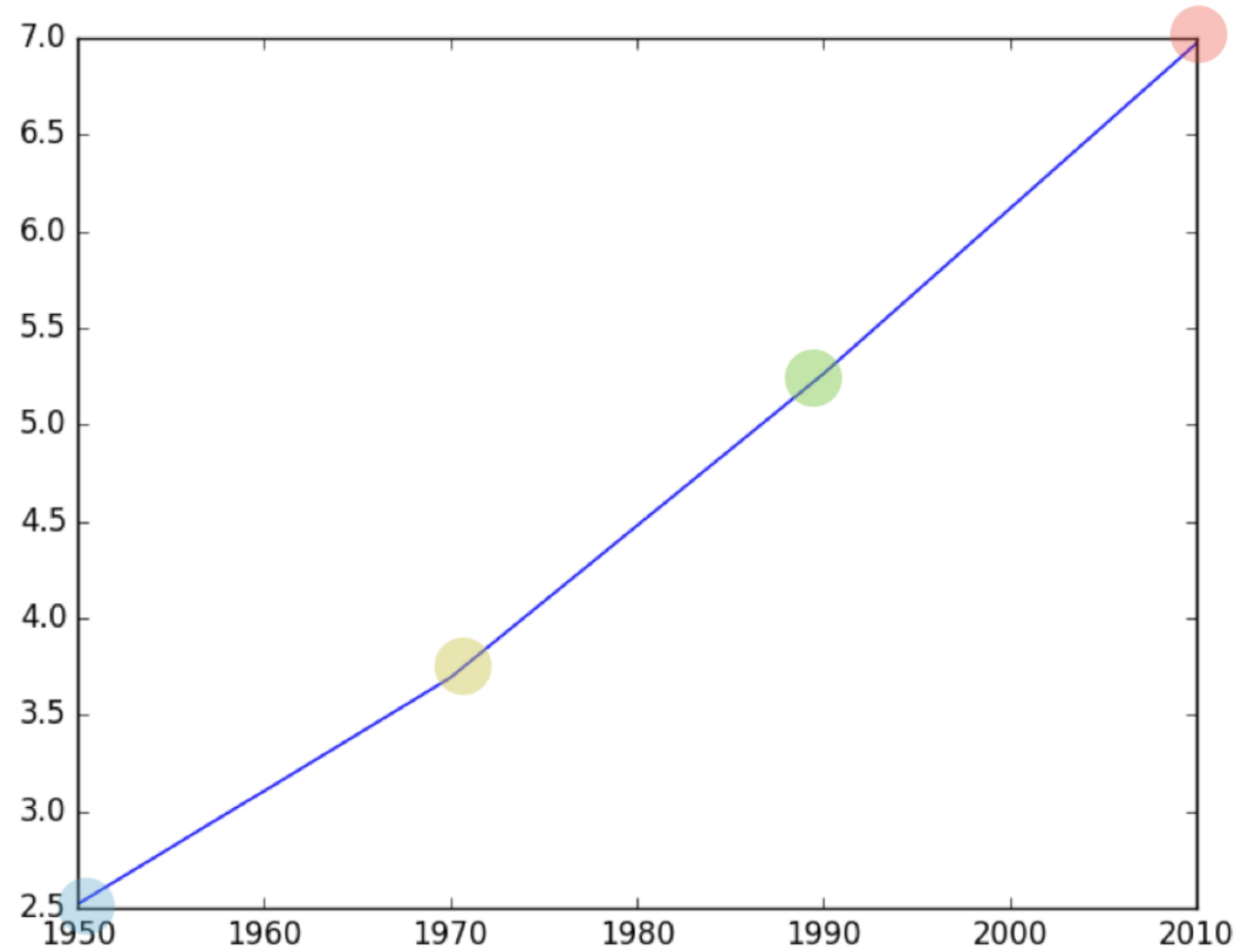
```
import matplotlib.pyplot as plt
year = [1950, 1970, 1990, 2010]
pop = [2.519, 3.692, 5.263, 6.972]
plt.plot(year, pop)
plt.show()
```


Matplotlib

```
import matplotlib.pyplot as plt
year = [1950, 1970, 1990, 2010]
pop = [2.519, 3.692, 5.263, 6.972]
plt.plot(year, pop)
plt.show()
```



Matplotlib



```
year = [1950 , 1970 , 1990 , 2010]  
pop  = [2.519, 3.692, 5.263, 6.972]
```

Matplotlib

Scatter plot

```
import matplotlib.pyplot as plt
year = [1950, 1970, 1990, 2010]
pop = [2.519, 3.692, 5.263, 6.972]
plt.plot(year, pop)
plt.show()
```

Matplotlib

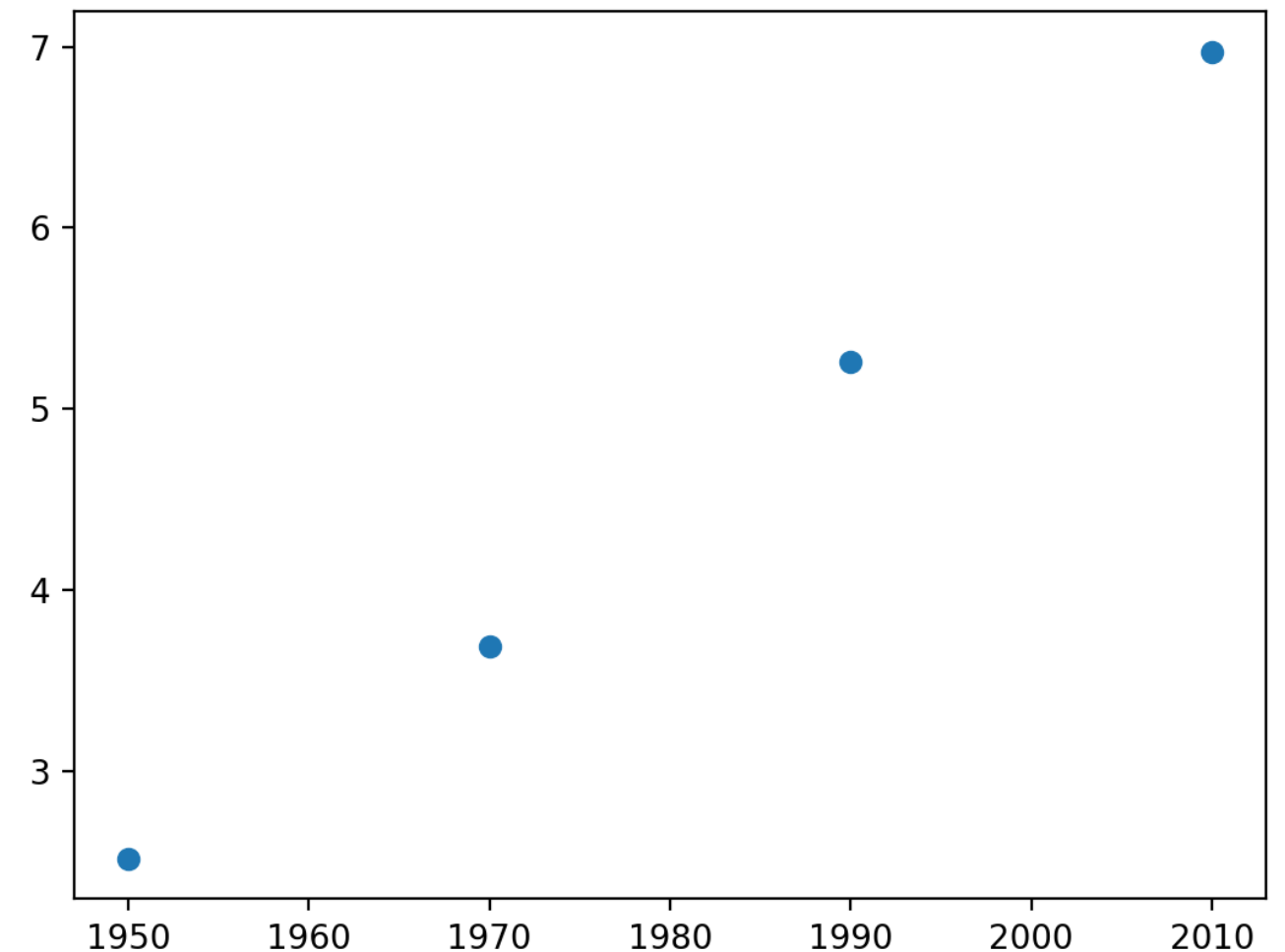
Scatter plot

```
import matplotlib.pyplot as plt
year = [1950, 1970, 1990, 2010]
pop = [2.519, 3.692, 5.263, 6.972]
plt.scatter(year, pop)
plt.show()
```

Matplotlib

Scatter plot

```
import matplotlib.pyplot as plt
year = [1950, 1970, 1990, 2010]
pop = [2.519, 3.692, 5.263, 6.972]
plt.scatter(year, pop)
plt.show()
```



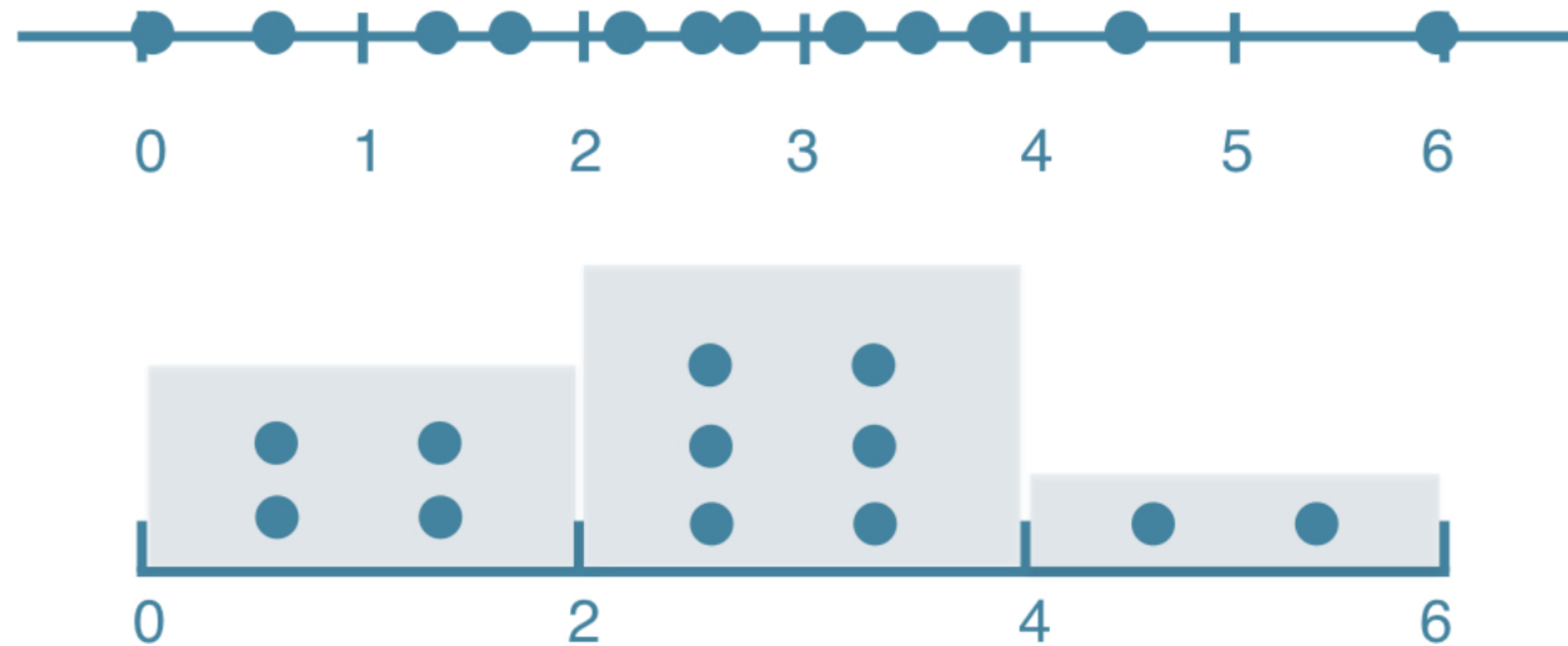
- 1- Open VSC
- 2- Open *PYTHON-101/code/02-intermediate/01-basic_plots.py*
- 3- Play with it

Let's practice

Matplotlib

Histogram

- Explore Dataset
- Get idea about distribution



Matplotlib

Histogram

```
import matplotlib.pyplot as plt
```

```
help(plt.hist)
```

Help on function hist in module matplotlib.pyplot:

```
hist(x, bins=None, range=None, density=False, weights=None,
      cumulative=False, bottom=None, histtype='bar', align='mid',
      orientation='vertical', rwidth=None, log=False, color=None,
      label=None, stacked=False, *, data=None, **kwargs)
```

Plot a histogram.

Compute and draw the histogram of *x*. The return value is a tuple (*n*, *bins*, *patches*) or (*n*₀, *n*₁, ...], *bins*, [*patches*₀, *patches*₁, ...]) if the input contains multiple data.

Matplotlib

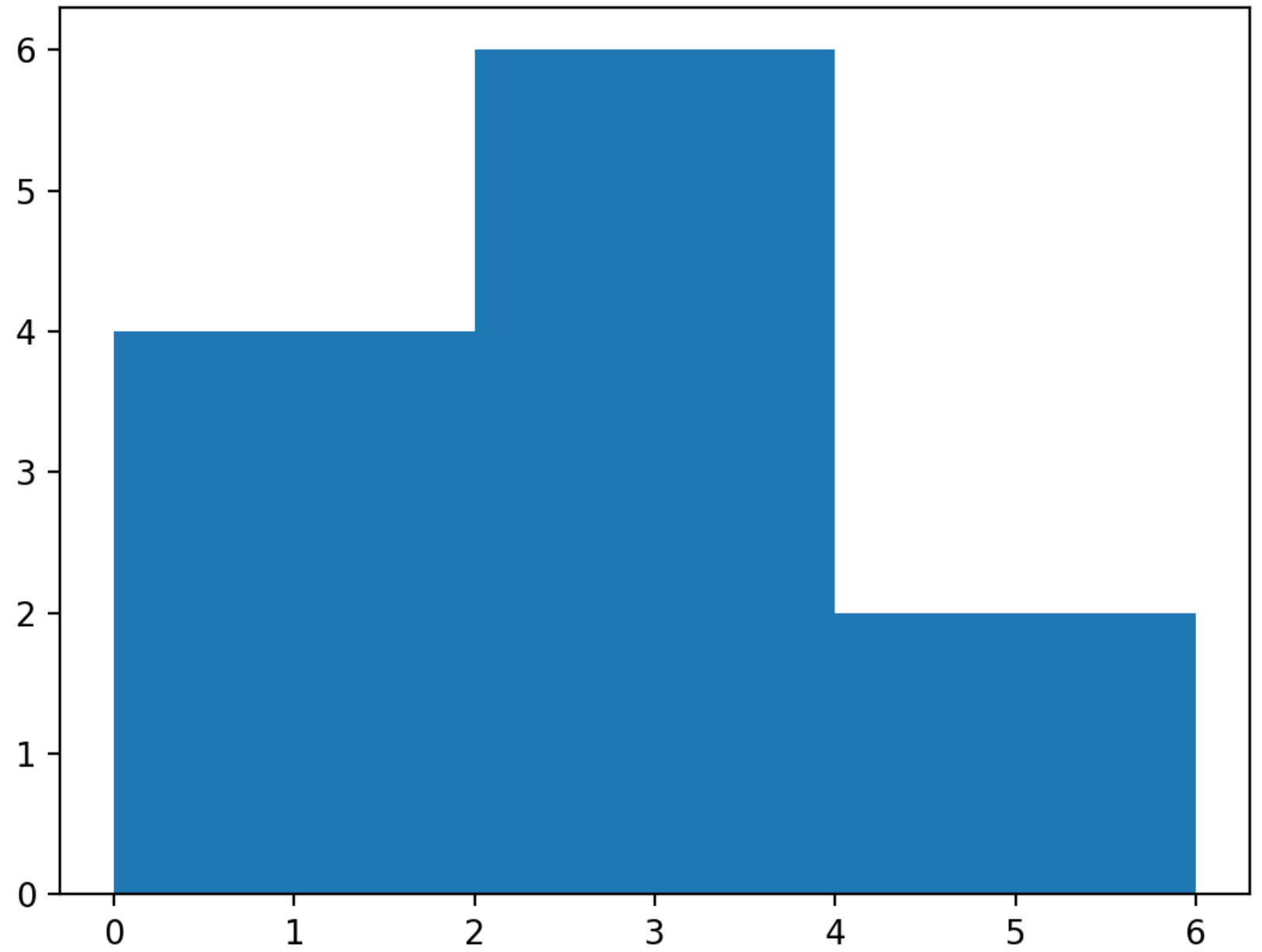
Histogram

```
import matplotlib.pyplot as plt
values = [0,0.6,1.4,1.6,2.2,2.5,2.6,3.2,3.5,3.9,4.2,6]
plt.hist(values, bins=3)
plt.show()
```

Matplotlib

Histogram

```
import matplotlib.pyplot as plt
values = [0, 0.6, 1.4, 1.6, 2.2, 2.2, 2.2]
plt.hist(values, bins=3)
plt.show()
```



- 1- Open VSC
- 2- Open *PYTHON-101/code/02-intermediate/02-histogram.py*
- 3- Play with it

Let's practice

Matplotlib

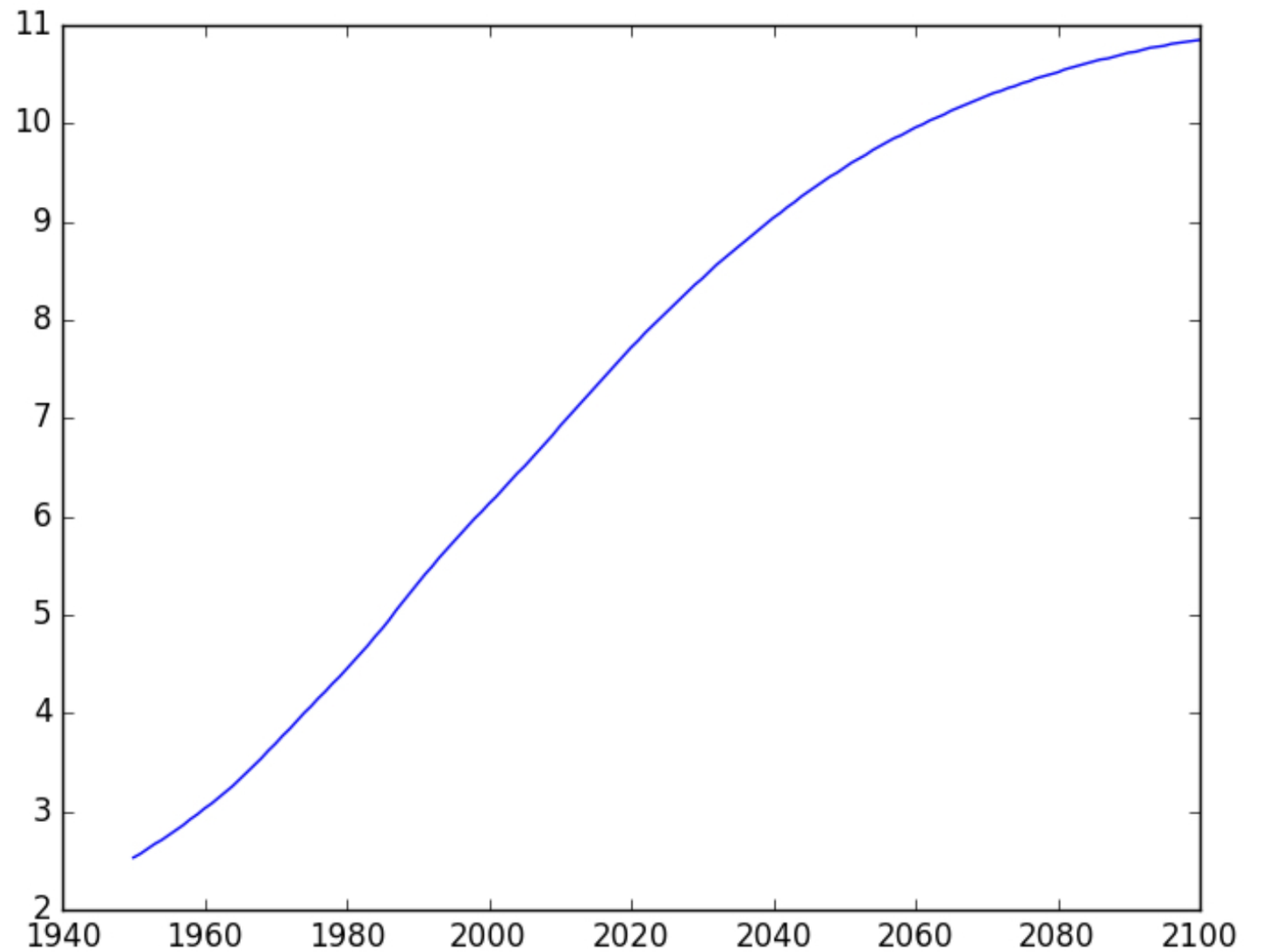
Customization

- Many option
 - Different plot types
 - Many customization
- Choice depend on:
 - Data
 - Story we want to tell

Matplotlib

Customization

```
import matplotlib.pyplot as plt
year = [1950, 1951, 1952, ..., 2100]
pop = [2.538, 2.57, 2.62, ..., 10.85]
plt.plot(year, pop)
plt.show()
```



Matplotlib

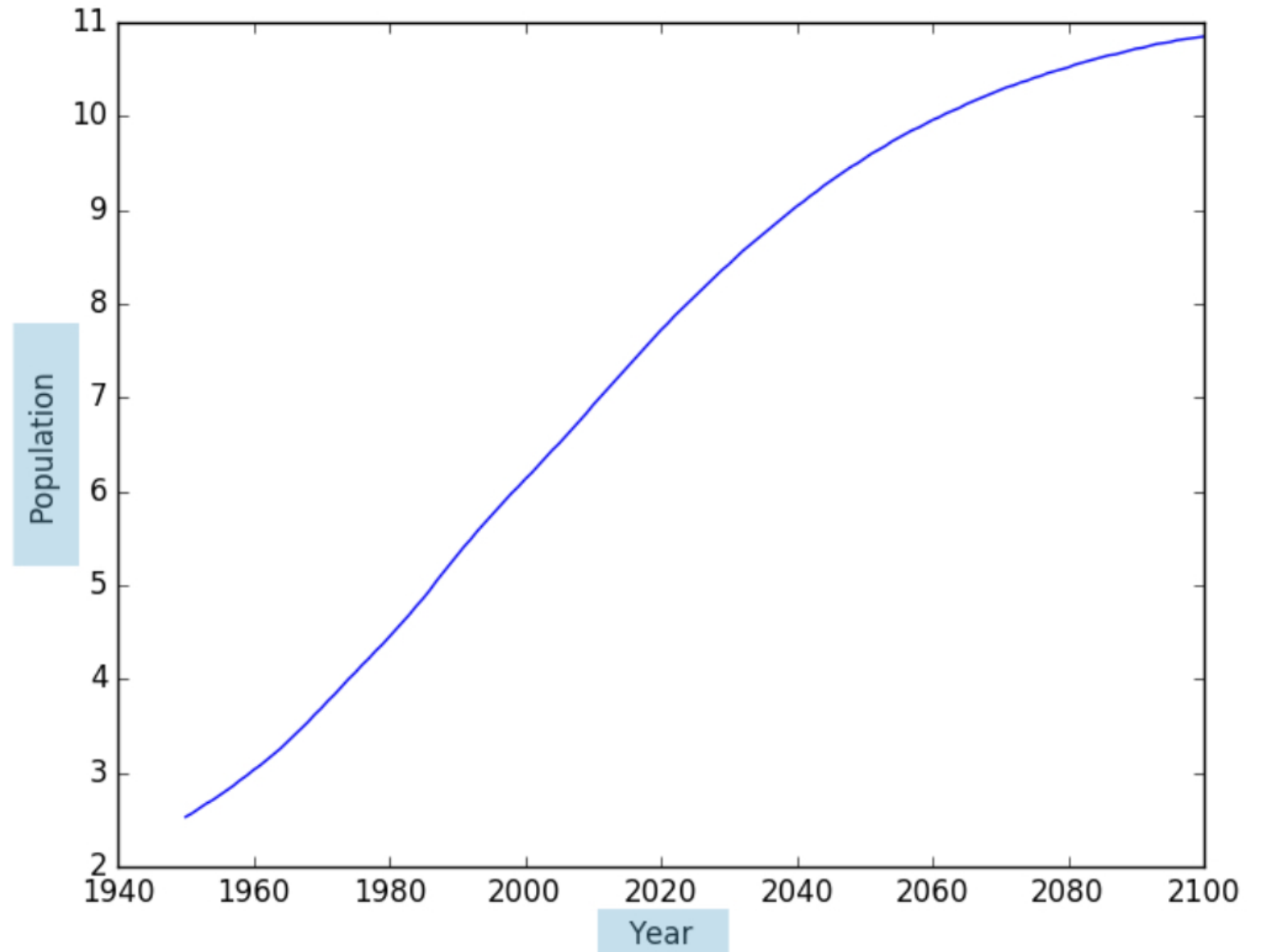
Customization

```
import matplotlib.pyplot as plt
year = [1950, 1951, 1952, ..., 2100]
pop = [2.538, 2.57, 2.62, ..., 10.85]

plt.plot(year, pop)

plt.xlabel('Year')
plt.ylabel('Population')

plt.show()
```



Matplotlib

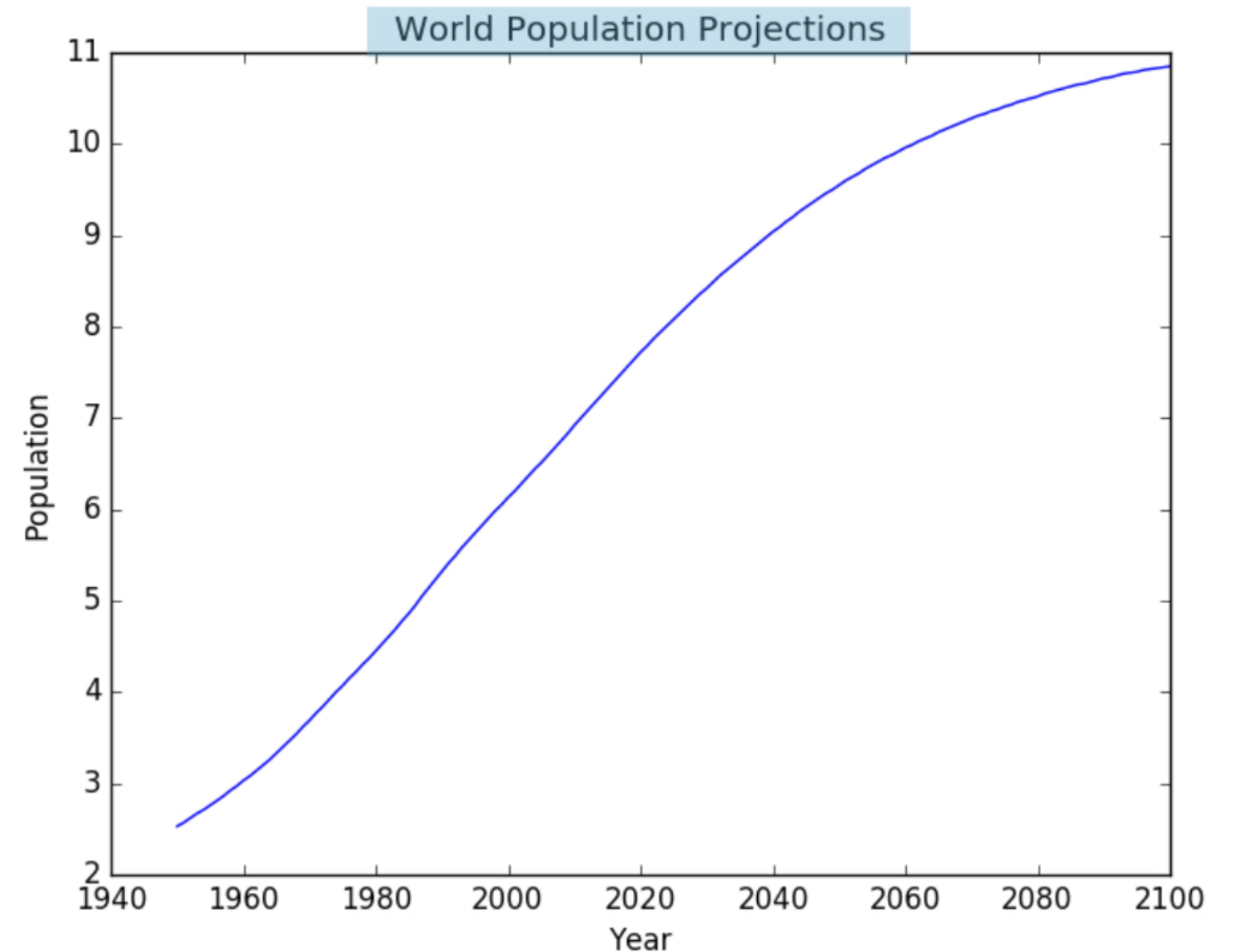
Customization

```
import matplotlib.pyplot as plt
year = [1950, 1951, 1952, ..., 2100]
pop = [2.538, 2.57, 2.62, ..., 10.85]

plt.plot(year, pop)

plt.xlabel('Year')
plt.ylabel('Population')
plt.title('World Population Projections')

plt.show()
```



Matplotlib

Customization

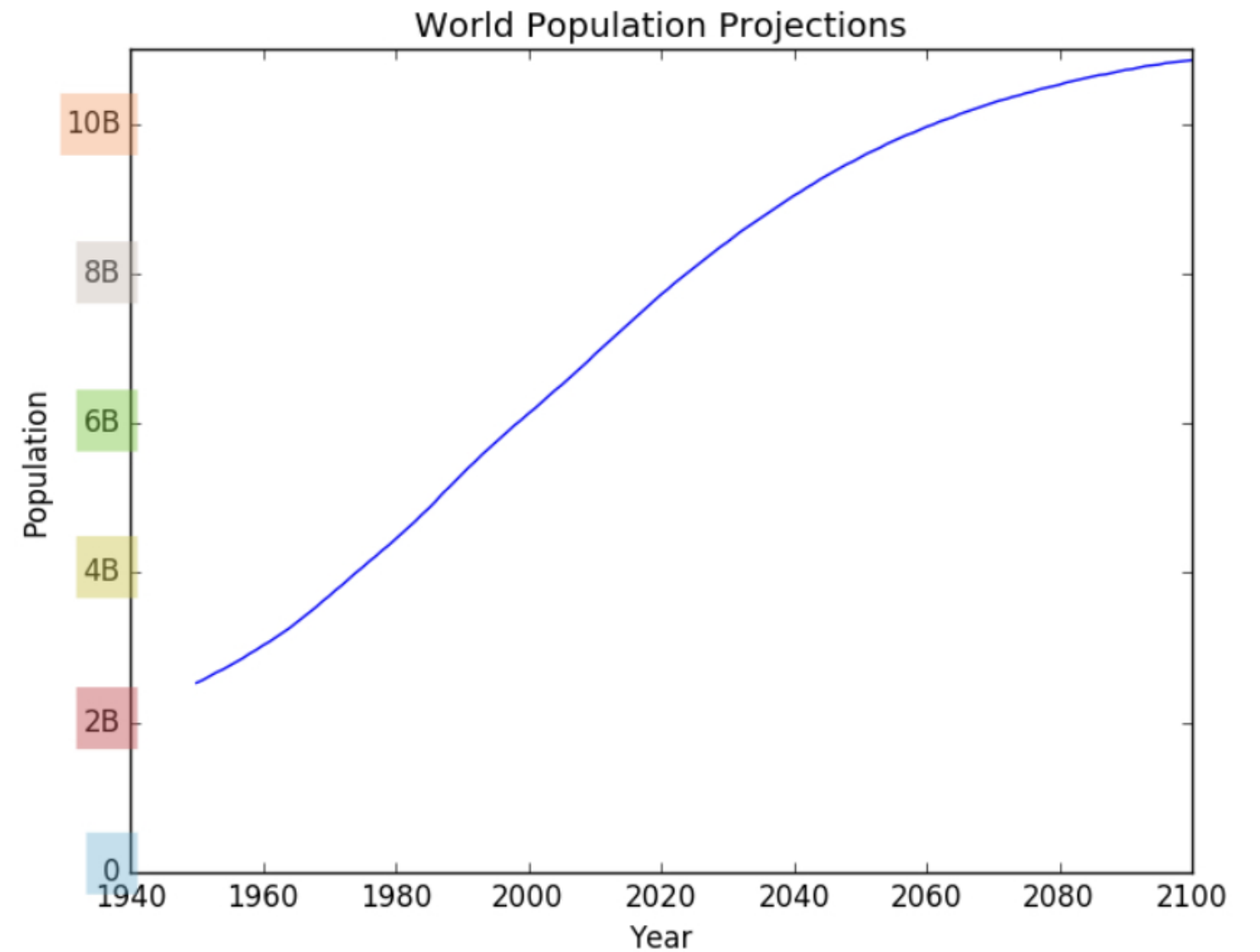
```
import matplotlib.pyplot as plt
year = [1950, 1951, 1952, ..., 2100]
pop = [2.538, 2.57, 2.62, ..., 10.85]

plt.plot(year, pop)

plt.xlabel('Year')
plt.ylabel('Population')
plt.title('World Population Projections')

plt.yticks([0, 2, 4, 6, 8, 10],
           ['0', '2B', '4B', '6B', '8B', '10B'])

plt.show()
```



Matplotlib

Customization

```
import matplotlib.pyplot as plt
year = [1950, 1951, 1952, ..., 2100]
pop = [2.538, 2.57, 2.62, ..., 10.85]

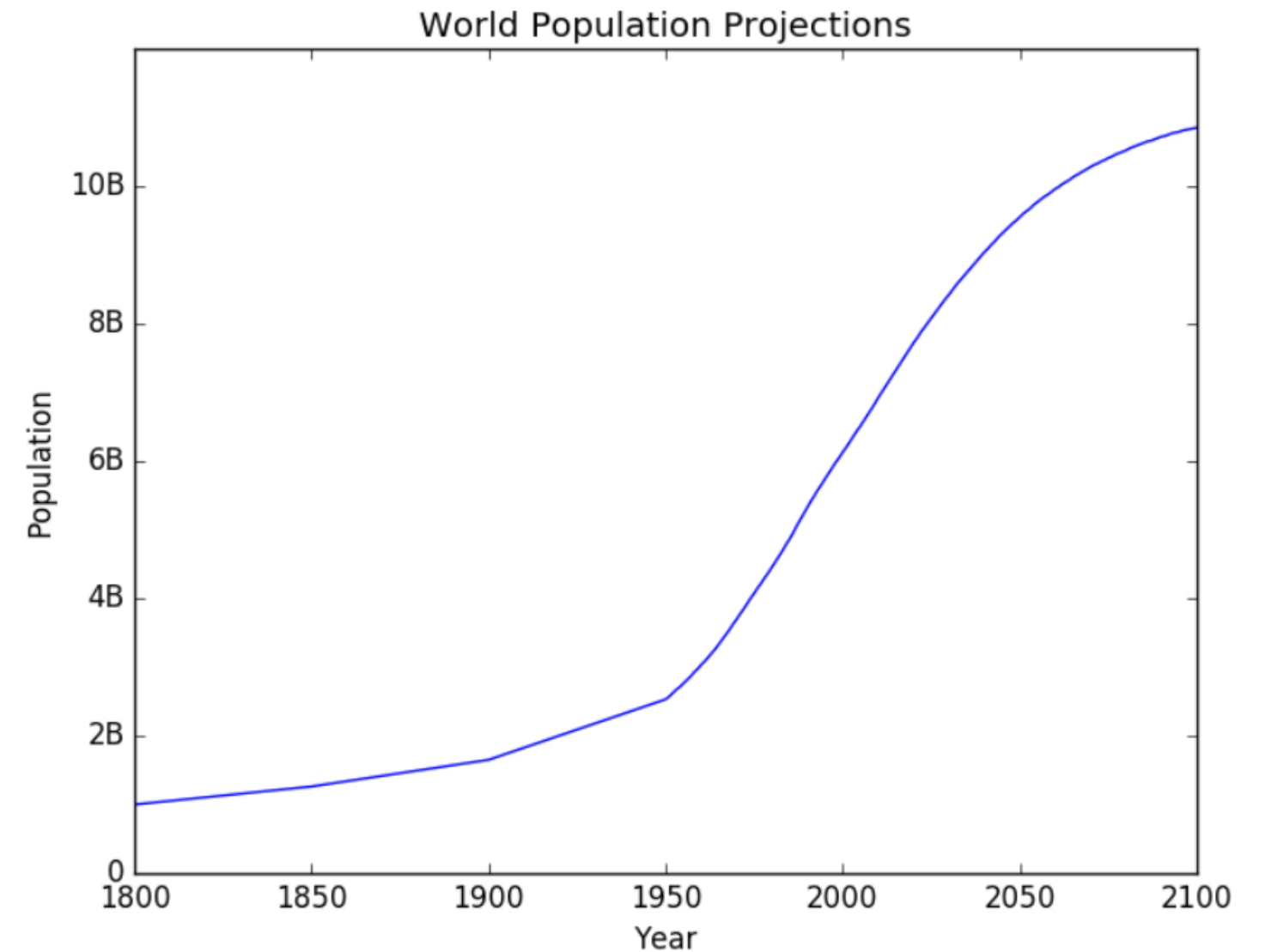
# Add more data
year = [1800, 1850, 1900] + year
pop = [1.0, 1.262, 1.650] + pop

plt.plot(year, pop)

plt.xlabel('Year')
plt.ylabel('Population')
plt.title('World Population Projections')

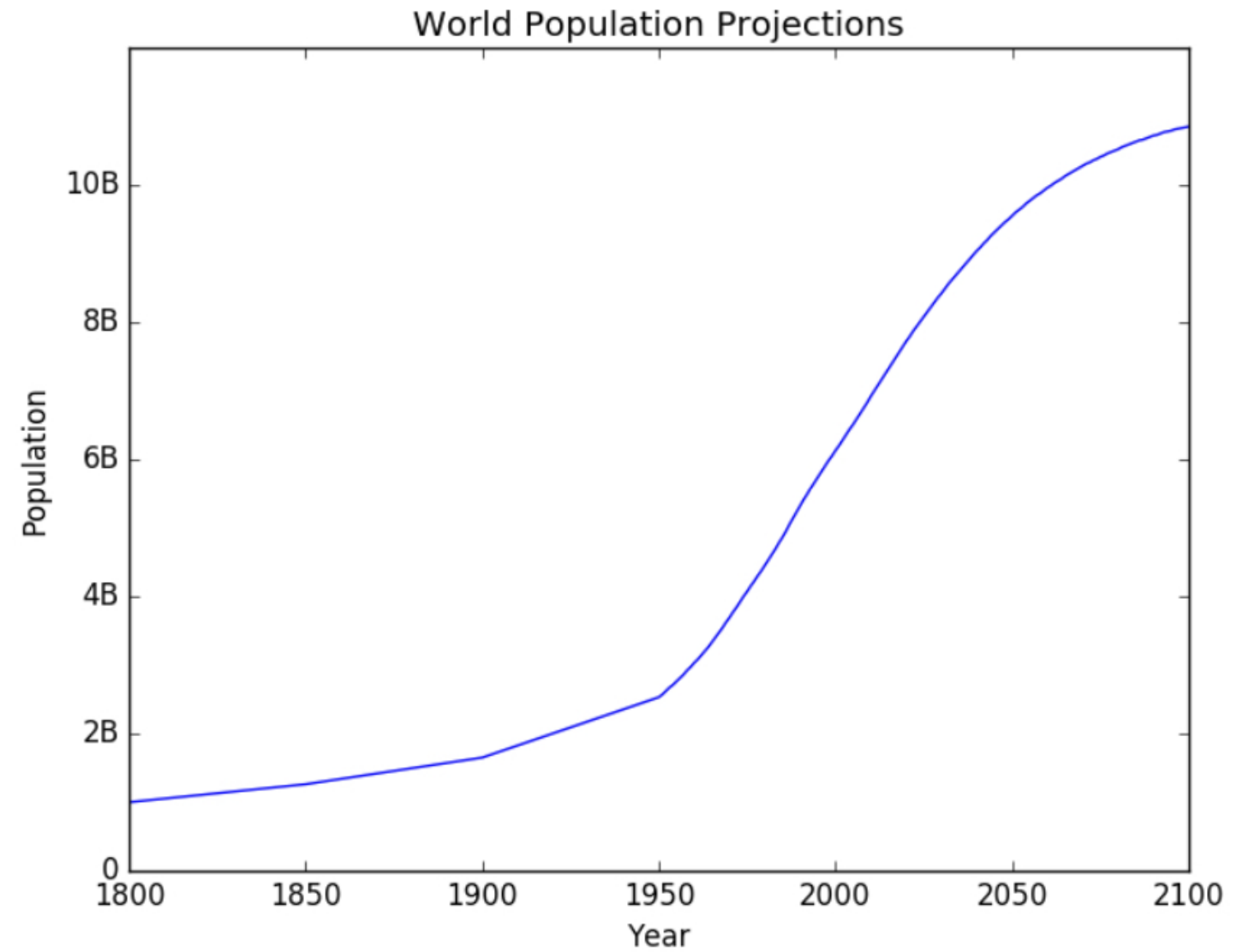
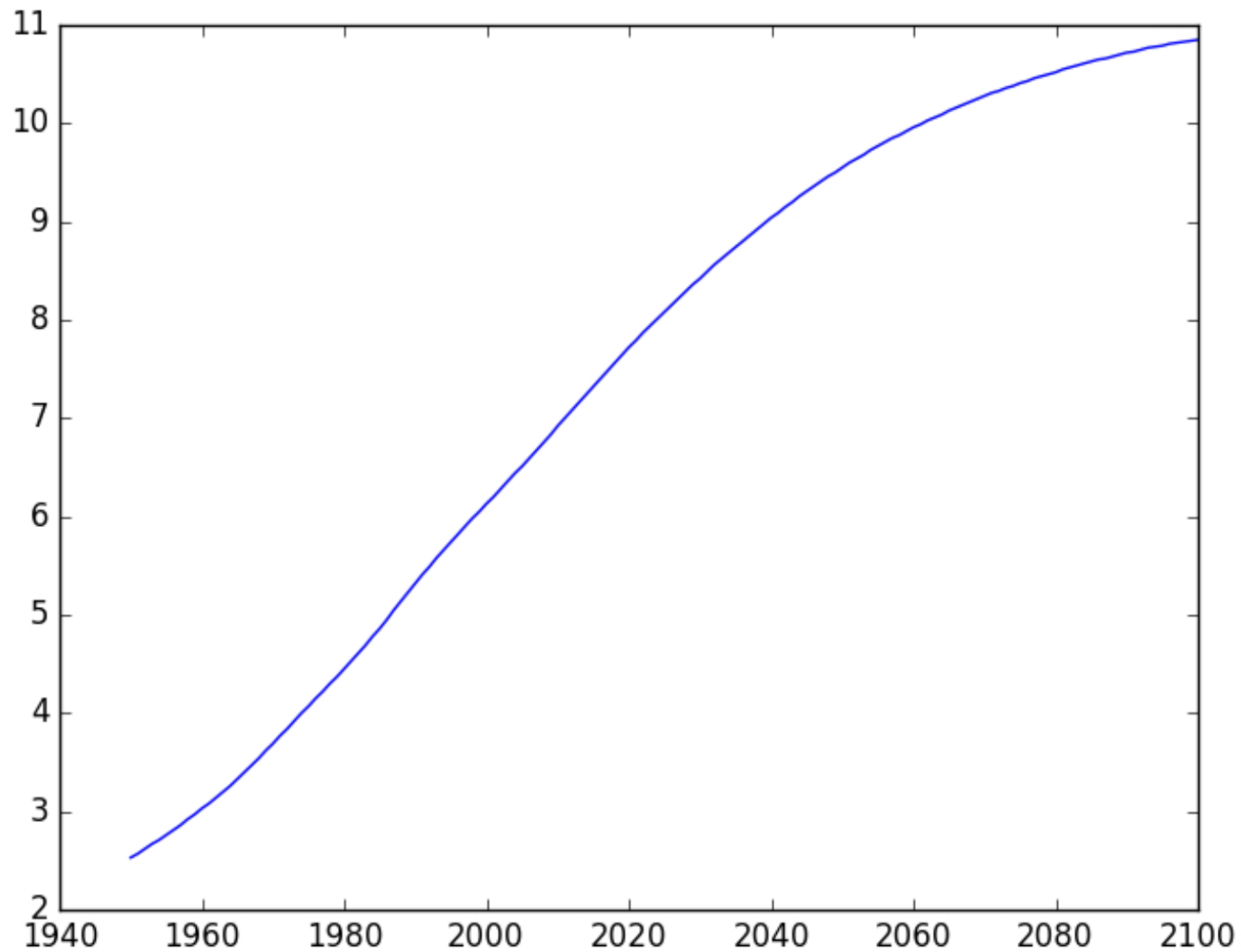
plt.yticks([0, 2, 4, 6, 8, 10],
           ['0', '2B', '4B', '6B', '8B', '10B'])

plt.show()
```



Matplotlib

Customization



- 1- Open VSC
- 2- Open *PYTHON-101/code/02-intermediate/03-customization.py*
- 3- Play with it

Let's practice

Dictionary

```
pop = [30.55, 2.77, 39.21]  
countries = ["afghanistan", "albania", "algeria"]  
ind_alb = countries.index("albania")  
ind_alb
```

1

```
pop[ind_alb]
```

2.77

Dictionary

- Not convenient
- Not intuitive

```
pop = [30.55, 2.77, 39.21]
countries = ["afghanistan", "albania", "algeria"]
ind_alb = countries.index("albania")
ind_alb
```

1

```
pop[ind_alb]
```

2.77

Dictionary

```
pop = [30.55, 2.77, 39.21]  
countries = ["afghanistan", "albania", "algeria"]
```

```
world = {"afghanistan": 30.55, "albania": 2.77, "algeria": 39.21}
```

Dictionary

```
pop = [30.55, 2.77, 39.21]  
countries = ["afghanistan", "albania", "algeria"]
```

```
world = {"afghanistan": 30.55, "albania": 2.77, "algeria": 39.21}
```

Dictionary

```
pop = [30.55, 2.77, 39.21]
```

```
countries = ["afghanistan", "albania", "algeria"]
```

```
world = {"afghanistan": 30.55, "albania": 2.77, "algeria": 39.21}
```


Dictionary

```
pop = [30.55, 2.77, 39.21]
countries = ["afghanistan", "albania", "algeria"]

world = {"afghanistan": 30.55, "albania": 2.77, "algeria": 39.21}
```

Dictionary

```
pop = [30.55, 2.77, 39.21]
countries = ["afghanistan", "albania", "algeria"]

world = {"afghanistan": 30.55, "albania": 2.77, "algeria": 39.21}

world['albania']
```

2.77

- 1- Open VSC
- 2- Open *PYTHON-101/code/02-intermediate/04-dictionary.py*
- 3- Play with it

Let's practice

Dictionary

Add value

```
world['sealand'] = 0.000027  
world
```

```
{"afghanistan": 30.55, "albania": 2.77, "algeria": 39.21,  
"sealand": 2.7e-05}
```

Dictionary

Add value

```
world['sealand'] = 0.000027  
world
```

```
{"afghanistan": 30.55, "albania": 2.77, "algeria": 39.21,  
"sealand": 2.7e-05}
```

```
"sealand" in world
```

```
True
```

Dictionary

Change value

```
world['sealand'] = 0.000028  
world
```

```
{"afghanistan": 30.55, "albania": 2.77, "algeria": 39.21,  
"sealand": 2.8e-05}
```

Dictionary

Remove value

```
del world['sealand']  
world
```

```
{"afghanistan": 30.55, "albania": 2.77, "algeria": 39.21}
```

Dictionary versus List

List	Dictionary
Select, update, and remove with <code>[]</code>	Select, update, and remove with <code>[]</code>
Indexed by range of numbers	Indexed by unique keys
Collection of values — order matters, for selecting entire subsets	Lookup table with unique keys

- 1- Open VSC
- 2- Open *PYTHON-101/code/02-intermediate/04-dictionary.py*
- 3- Play with it

Let's practice

Pandas DataFrame

Tabular data

temperature	measured_at	location
76	2016-01-01 14:00:01	valve
86	2016-01-01 14:00:01	compressor
72	2016-01-01 15:00:01	valve
88	2016-01-01 15:00:01	compressor
68	2016-01-01 16:00:01	valve
78	2016-01-01 16:00:01	compressor

Pandas DataFrame

Tabular data

temperature	measured_at	location
76	2016-01-01 14:00:01	valve
86	2016-01-01 14:00:01	compressor
72	2016-01-01 15:00:01	valve
88	2016-01-01 15:00:01	compressor
68	2016-01-01 16:00:01	valve
78	2016-01-01 16:00:01	compressor

row is a observation
column is a variable

Pandas DataFrame

Tabular data

country	capital	area	population
Brazil	Brasilia	8.516	200.4
Russia	Moscow	17.10	143.5
India	New Delhi	3.286	1252
China	Beijing	9.597	1357
South	Pretoria	1.221	52.98

Pandas DataFrame

Tabular data

country	capital	area	population
Brazil	Brasilia	8.516	200.4
Russia	Moscow	17.10	143.5
India	New Delhi	3.286	1252
China	Beijing	9.597	1357
South	Pretoria	1.221	52.98
str	str	float	float

Pandas DataFrame

Tabular data

- Build on NumPy
- Several ways to create a DataFrame:
 - from dictionary
 - from csv file
 - ...
- Great because it allows high level data manipulation

	country	capital	area	population
BR	Brazil	Brasilia	8.516	200.40
RU	Russia	Moscow	17.100	143.50
IN	India	New Delhi	3.286	1252.00
CH	China	Beijing	9.597	1357.00
SA	South Africa	Pretoria	1.221	52.98

Pandas DataFrame

from dict

```
dict = {  
    "country": ["Brazil", "Russia", "India", "China", "South Africa"],  
    "capital": ["Brasilia", "Moscow", "New Delhi", "Beijing", "Pretoria"],  
    "area": [8.516, 17.10, 3.286, 9.597, 1.221],  
    "population": [200.4, 143.5, 1252, 1357, 52.98] }  
  
import pandas as pd  
brics = pd.DataFrame(dict)
```

- keys are the column labels
- values are the data, column by column

Pandas DataFrame

from csv

```
,country,capital,area,population  
BR,Brazil,Brasilia,8.516,200.4  
RU,Russia,Moscow,17.10,143.5  
IN,India,New Delhi,3.286,1252  
CH,China,Beijing,9.597,1357  
SA,South Africa,Pretoria,1.221,52.98
```

```
import pandas as pd  
brics = pd.read_csv("path/to/brics.csv")
```


Pandas DataFrame

from csv

```
,country,capital,area,population  
BR,Brazil,Brasilia,8.516,200.4  
RU,Russia,Moscow,17.10,143.5  
IN,India,New Delhi,3.286,1252  
CH,China,Beijing,9.597,1357  
SA,South Africa,Pretoria,1.221,52.98
```

```
import pandas as pd  
brics = pd.read_csv("path/to/brics.csv", index_col = 0)
```

- 1- Open VSC
- 2- Open *PYTHON-101/code/02-intermediate/05-pandas.py*
- 3- Play with it

Let's practice

Pandas DataFrame

Select & Access Data

- 2 methods:
 - Square brackets
 - Advanced methods: *loc* & *iloc*

Pandas DataFrame

Column Access []

```
      country  capital  area  population
BR      Brazil  Brasilia  8.516      200.40
RU      Russia   Moscow 17.100      143.50
IN       India New Delhi  3.286     1252.00
CH       China   Beijing  9.597     1357.00
SA  South Africa  Pretoria  1.221       52.98
```

```
brics["country"]
```

```
BR      Brazil
RU      Russia
IN       India
CH       China
SA  South Africa
Name: country, dtype: object
```

Pandas DataFrame

Column Access []

	country	capital	area	population
BR	Brazil	Brasilia	8.516	200.40
RU	Russia	Moscow	17.100	143.50
IN	India	New Delhi	3.286	1252.00
CH	China	Beijing	9.597	1357.00
SA	South Africa	Pretoria	1.221	52.98

```
type(brics["country"])
```

```
pandas.core.series.Series
```

Pandas DataFrame

Column Access []

```
      country  capital  area  population
BR      Brazil  Brasilia  8.516      200.40
RU      Russia   Moscow 17.100      143.50
IN       India New Delhi  3.286     1252.00
CH       China   Beijing  9.597     1357.00
SA South Africa Pretoria  1.221       52.98
```

```
brics[["country"]]
```

```
      country
BR      Brazil
RU      Russia
IN       India
CH       China
SA South Africa
```

Pandas DataFrame

Column Access []

	country	capital	area	population
BR	Brazil	Brasilia	8.516	200.40
RU	Russia	Moscow	17.100	143.50
IN	India	New Delhi	3.286	1252.00
CH	China	Beijing	9.597	1357.00
SA	South Africa	Pretoria	1.221	52.98

```
type(brics[["country"]])
```

```
pandas.core.frame.DataFrame
```

Pandas DataFrame

Column Access []

	country	capital	area	population
BR	Brazil	Brasilia	8.516	200.40
RU	Russia	Moscow	17.100	143.50
IN	India	New Delhi	3.286	1252.00
CH	China	Beijing	9.597	1357.00
SA	South Africa	Pretoria	1.221	52.98

```
brics[["country", "capital"]]
```

	country	capital
BR	Brazil	Brasilia
RU	Russia	Moscow
IN	India	New Delhi
CH	China	Beijing
SA	South Africa	Pretoria

Pandas DataFrame

Row Access []

	country	capital	area	population
BR	Brazil	Brasilia	8.516	200.40
RU	Russia	Moscow	17.100	143.50
IN	India	New Delhi	3.286	1252.00
CH	China	Beijing	9.597	1357.00
SA	South Africa	Pretoria	1.221	52.98

```
brics[1:4]
```

	country	capital	area	population
RU	Russia	Moscow	17.100	143.5
IN	India	New Delhi	3.286	1252.0
CH	China	Beijing	9.597	1357.0

Pandas DataFrame

Row Access loc

	country	capital	area	population
BR	Brazil	Brasilia	8.516	200.40
RU	Russia	Moscow	17.100	143.50
IN	India	New Delhi	3.286	1252.00
CH	China	Beijing	9.597	1357.00
SA	South Africa	Pretoria	1.221	52.98

```
brics.loc["RU"]
```

```
country      Russia
capital      Moscow
area         17.1
population   143.5
Name: RU, dtype: object
```

- Row as pandas Series

Pandas DataFrame

Row Access loc

	country	capital	area	population
BR	Brazil	Brasilia	8.516	200.40
RU	Russia	Moscow	17.100	143.50
IN	India	New Delhi	3.286	1252.00
CH	China	Beijing	9.597	1357.00
SA	South Africa	Pretoria	1.221	52.98

```
brics.loc[["RU"]]
```

	country	capital	area	population
RU	Russia	Moscow	17.1	143.5

- DataFrame

Pandas DataFrame

Row Access loc

	country	capital	area	population
BR	Brazil	Brasilia	8.516	200.40
RU	Russia	Moscow	17.100	143.50
IN	India	New Delhi	3.286	1252.00
CH	China	Beijing	9.597	1357.00
SA	South Africa	Pretoria	1.221	52.98

```
brics.loc[["RU", "IN", "CH"]]
```

	country	capital	area	population
RU	Russia	Moscow	17.100	143.5
IN	India	New Delhi	3.286	1252.0
CH	China	Beijing	9.597	1357.0

Pandas DataFrame

Row & Column Access loc

	country	capital	area	population
BR	Brazil	Brasilia	8.516	200.40
RU	Russia	Moscow	17.100	143.50
IN	India	New Delhi	3.286	1252.00
CH	China	Beijing	9.597	1357.00
SA	South Africa	Pretoria	1.221	52.98

```
brics.loc[["RU", "IN", "CH"], ["country", "capital"]]
```

	country	capital
RU	Russia	Moscow
IN	India	New Delhi
CH	China	Beijing

Pandas DataFrame

Row & Column Access loc

	country	capital	area	population
BR	Brazil	Brasilia	8.516	200.40
RU	Russia	Moscow	17.100	143.50
IN	India	New Delhi	3.286	1252.00
CH	China	Beijing	9.597	1357.00
SA	South Africa	Pretoria	1.221	52.98

```
brics.loc[:, ["country", "capital"]]
```

	country	capital
BR	Brazil	Brasilia
RU	Russia	Moscow
IN	India	New Delhi
CH	China	Beijing
SA	South Africa	Pretoria

- 1- Open VSC
- 2- Open *PYTHON-101/code/02-intermediate/05-pandas.py*
- 3- Play with it

Let's practice

Comparison Operators

Numeric comparisons

```
2 < 3
```

```
True
```

```
2 == 3
```

```
False
```

```
2 <= 3
```

```
True
```

```
3 <= 3
```

```
True
```

```
x = 2
```

```
y = 3
```

```
x < y
```

```
True
```


Comparison Operators

Other comparisons

```
"carl" < "chris"
```

```
True
```

```
3 < "chris"
```

```
TypeError: unorderable types: int() < str()
```

```
3 < 4.1
```

```
True
```

```
bmi
```

```
array([21.852, 20.975, 21.75 , 24.747, 21.441])
```

```
bmi > 23
```

```
array([False, False, False, True, False], dtype=bool)
```

Comparison Operators

Comparator	Meaning
<	Strictly less than
<=	Less than or equal
>	Strictly greater than
>=	Greater than or equal
==	Equal
!=	Not equal

- 1- Open VSC
- 2- Open *PYTHON-101/code/02-intermediate/06-comparison_operators.py*
- 3- Play with it

Let's practice

Boolean Operators

and

True and True

True

```
x = 12
x > 5 and x < 15
# True     True
```

True

False and True

False

True and False

False

False and False

False

Boolean Operators

or

```
True or True
```

```
True
```

```
False or False
```

```
False
```

```
False or True
```

```
True
```

```
y = 5  
y < 7 or y > 13
```

```
True
```

```
True or False
```

```
True
```

Boolean Operators

not

```
not True
```

```
False
```

```
not False
```

```
True
```

Boolean Operators

NumPy

```
bmi      # calculation of bmi left out
```

```
array([21.852, 20.975, 21.75 , 24.747, 21.441])
```

```
bmi > 21
```

```
array([True, False, True, True, True], dtype=bool)
```

```
bmi < 22
```

```
array([True, True, True, False, True], dtype=bool)
```

```
bmi > 21 and bmi < 22
```

```
ValueError: The truth value of an array with more than one element is  
ambiguous. Use a.any() or a.all()
```

Boolean Operators

NumPy

- `logical_and()`
- `logical_or()`
- `logical_not()`

```
np.logical_and(bmi > 21, bmi < 22)
```

```
array([True, False, True, False, True], dtype=bool)
```

```
bmi[np.logical_and(bmi > 21, bmi < 22)]
```

```
array([21.852, 21.75, 21.441])
```


- 1- Open VSC
- 2- Open *PYTHON-101/code/02-intermediate/07-boolean_operators.py*
- 3- Play with it

Let's practice

if, elif, else Operators

if

```
if condition :  
    expression
```

control.py

```
z = 4  
if z % 2 == 0 :    # True  
    print("z is even")
```

```
z is even
```

if, elif, else Operators

else

```
if condition :  
    expression  
else :  
    expression
```

control.py

```
z = 5  
if z % 2 == 0 :    # False  
    print("z is even")  
else :  
    print("z is odd")
```

z is odd

if, elif, else Operators

elif

```
if condition :  
    expression  
elif condition :  
    expression  
else :  
    expression
```

control.py

```
z = 3  
if z % 2 == 0 :  
    print("z is divisible by 2")    # False  
elif z % 3 == 0 :  
    print("z is divisible by 3")    # True  
else :  
    print("z is neither divisible by 2 nor by 3")
```

z is divisible by 3

if, elif, else Operators

elif

```
if condition :  
    expression  
elif condition :  
    expression  
else :  
    expression
```

control.py

```
z = 6  
if z % 2 == 0 :  
    print("z is divisible by 2")    # True  
elif z % 3 == 0 :  
    print("z is divisible by 3")    # Never reached  
else :  
    print("z is neither divisible by 2 nor by 3")
```

z is divisible by 2

- 1- Open VSC
- 2- Open *PYTHON-101/code/02-intermediate/08-condition_operators.py*
- 3- Play with it

Let's practice

Filtering DataFrames

```
import pandas as pd  
brics = pd.read_csv("path/to/brics.csv", index_col = 0)  
brics
```

	country	capital	area	population
BR	Brazil	Brasilia	8.516	200.40
RU	Russia	Moscow	17.100	143.50
IN	India	New Delhi	3.286	1252.00
CH	China	Beijing	9.597	1357.00
SA	South Africa	Pretoria	1.221	52.98

Filtering DataFrames

	country	capital	area	population
BR	Brazil	Brasilia	8.516	200.40
RU	Russia	Moscow	17.100	143.50
IN	India	New Delhi	3.286	1252.00
CH	China	Beijing	9.597	1357.00
SA	South Africa	Pretoria	1.221	52.98

- Select countries with area over 8 million km2
- 3 steps
 - Select the area column
 - Do comparison on area column
 - Use result to select countries

Filtering DataFrames

Step 1: Get Column

	country	capital	area	population
BR	Brazil	Brasilia	8.516	200.40
RU	Russia	Moscow	17.100	143.50
IN	India	New Delhi	3.286	1252.00
CH	China	Beijing	9.597	1357.00
SA	South Africa	Pretoria	1.221	52.98

```
brics["area"]
```

```
BR    8.516
RU   17.100
IN    3.286
CH    9.597
SA    1.221
Name: area, dtype: float64    # - Need Pandas Series
```

- Alternatives:

```
brics.loc[:, "area"]
brics.iloc[:, 2]
```

Filtering DataFrames

Step 2: Compare

```
brics["area"]
```

```
BR      8.516  
RU     17.100  
IN      3.286  
CH      9.597  
SA      1.221  
Name: area, dtype: float64
```

```
brics["area"] > 8
```

```
BR      True  
RU      True  
IN     False  
CH      True  
SA     False  
Name: area, dtype: bool
```

```
is_huge = brics["area"] > 8
```

Filtering DataFrames

Step 3: Subset the DataFrame

```
is_huge
```

```
BR    True  
RU    True  
IN    False  
CH    True  
SA    False  
Name: area, dtype: bool
```

```
brics[is_huge]
```

	country	capital	area	population
BR	Brazil	Brasilia	8.516	200.4
RU	Russia	Moscow	17.100	143.5
CH	China	Beijing	9.597	1357.0

Filtering DataFrames

Summary

```
country capital area population
BR Brazil Brasilia 8.516 200.40
RU Russia Moscow 17.100 143.50
IN India New Delhi 3.286 1252.00
CH China Beijing 9.597 1357.00
SA South Africa Pretoria 1.221 52.988
```

```
is_huge = brics["area"] > 8
brics[is_huge]
```

```
country capital area population
BR Brazil Brasilia 8.516 200.4
RU Russia Moscow 17.100 143.5
CH China Beijing 9.597 1357.0
```

```
brics[brics["area"] > 8]
```

```
country capital area population
BR Brazil Brasilia 8.516 200.4
RU Russia Moscow 17.100 143.5
CH China Beijing 9.597 1357.0
```

Filtering DataFrames

Boolean operators

```
      country  capital  area  population
BR      Brazil  Brasilia  8.516      200.40
RU      Russia   Moscow 17.100      143.50
IN       India  New Delhi  3.286     1252.00
CH       China   Beijing  9.597     1357.00
SA  South Africa  Pretoria  1.221       52.98
```

```
import numpy as np
np.logical_and(brics["area"] > 8, brics["area"] < 10)
```

```
BR      True
RU     False
IN     False
CH      True
SA     False
Name: area, dtype: bool
```

```
brics[np.logical_and(brics["area"] > 8, brics["area"] < 10)]
```

```
      country  capital  area  population
BR  Brazil  Brasilia  8.516      200.4
CH   China   Beijing  9.597     1357.0
```

- 1- Open VSC
- 2- Open *PYTHON-101/code/02-intermediate/09-filtering_dataframe.py*
- 3- Play with it

Let's practice

While loop

```
while condition :  
    expression
```

- While loop is a if-statement, repeated
- It repeats the action until condition is met
- Examples:
 - Error starts at 50
 - Divide error by 4 on every run
 - Continue until error is no longer > 1

While loop

```
while condition :  
    expression
```

```
error = 50.0  
  
while error > 1:  
    error = error / 4  
    print(error)
```

```
12.5  
3.125  
0.78125
```


While loop

```
while condition :  
    expression
```

```
error = 50.0  
while error > 1 :    # always True  
    # error = error / 4  
    print(error)
```

```
50  
50  
50  
...
```

While loop

```
while condition :  
    expression
```

```
error = 50.0  
while error > 1 :    # always True  
    # error = error / 4  
    print(error)
```

Control + C

```
50  
50  
50  
...
```

- 1- Open VSC
- 2- Open *PYTHON-101/code/02-intermediate/10-while_loop.py*
- 3- Play with it

Let's practice

For loop

```
for var in seq :  
    expression
```

- for *value* in *sequence*, execute *expression*

For loop

```
fam = [1.73, 1.68, 1.71, 1.89]  
print(fam)
```

```
[1.73, 1.68, 1.71, 1.89]
```

For loop

```
fam = [1.73, 1.68, 1.71, 1.89]  
print(fam)
```

```
[1.73, 1.68, 1.71, 1.89]
```

```
fam = [1.73, 1.68, 1.71, 1.89]  
for height in fam :  
    print(height)
```

For loop

```
fam = [1.73, 1.68, 1.71, 1.89]  
print(fam)
```

```
[1.73, 1.68, 1.71, 1.89]
```

```
fam = [1.73, 1.68, 1.71, 1.89]  
for height in fam :  
    print(height)
```

```
1.73
```

For loop

```
fam = [1.73, 1.68, 1.71, 1.89]  
print(fam)
```

```
[1.73, 1.68, 1.71, 1.89]
```

```
fam = [1.73, 1.68, 1.71, 1.89]  
for height in fam :  
    print(height)
```

```
1.73  
1.68
```


For loop

```
fam = [1.73, 1.68, 1.71, 1.89]  
print(fam)
```

```
[1.73, 1.68, 1.71, 1.89]
```

```
fam = [1.73, 1.68, 1.71, 1.89]  
for height in fam :  
    print(height)
```

```
1.73  
1.68  
1.71
```

For loop

```
fam = [1.73, 1.68, 1.71, 1.89]  
print(fam)
```

```
[1.73, 1.68, 1.71, 1.89]
```

```
fam = [1.73, 1.68, 1.71, 1.89]  
for height in fam :  
    print(height)
```

```
1.73  
1.68  
1.71  
1.89
```

For loop

```
index 0: 1.73  
index 1: 1.68  
index 2: 1.71  
index 3: 1.89
```

For loop

```
fam = [1.73, 1.68, 1.71, 1.89]
for index, height in enumerate(fam) :
    print("index " + str(index) + ": " + str(height))
```

```
index 0: 1.73
index 1: 1.68
index 2: 1.71
index 3: 1.89
```

For loop

```
for c in "family" :  
    print(c.capitalize())
```

```
F  
A  
M  
I  
L  
Y
```

- 1- Open VSC
- 2- Open *PYTHON-101/code/02-intermediate/11-for_loop.py*
- 3- Play with it

Let's practice

Loop on data structure

Dictionary

```
for var in seq :  
    expression
```

Loop on data structure

Dictionary

```
for var in seq :  
    expression
```

```
world = { "afghanistan":30.55,  
          "albania":2.77,  
          "algeria":39.21 }  
for key, value in world :  
    print(key + " -- " + str(value))
```


Loop on data structure

Dictionary

```
for var in seq :  
    expression
```

```
world = { "afghanistan":30.55,  
          "albania":2.77,  
          "algeria":39.21 }  
for key, value in world :  
    print(key + " -- " + str(value))
```

```
ValueError: too many values to  
        unpack (expected 2)
```

Loop on data structure

Dictionary

```
for var in seq :  
    expression
```

```
world = { "afghanistan":30.55,  
          "albania":2.77,  
          "algeria":39.21 }  
for key, value in world.items() :  
    print(key + " -- " + str(value))
```

```
algeria -- 39.21  
afghanistan -- 30.55  
albania -- 2.77
```

Loop on data structure

NumPy array

```
for var in seq :  
    expression
```

```
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])  
bmi = np_weight / np_height ** 2  
for val in bmi :  
    print(val)
```

```
21.852  
20.975  
21.750  
24.747  
21.441
```

Loop on data structure

2D-NumPy array

```
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])
meas = np.array([np_height, np_weight])
for val in meas :
    print(val)
```

```
[ 1.73  1.68  1.71  1.89  1.79]
[ 65.4  59.2  63.6  88.4  68.7]
```

Loop on data structure

2D-NumPy array

```
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])
meas = np.array([np_height, np_weight])
for val in np.nditer(meas) :
    print(val)
```

```
1.73
1.68
1.71
1.89
1.79
65.4
...
```

- 1- Open VSC
- 2- Open *PYTHON-101/code/02-intermediate/12-loop_datastructure.py*
- 3- Play with it

Let's practice

Loop on data structure

Pandas DataFrame

	country	capital	area	population
BR	Brazil	Brasilia	8.516	200.40
RU	Russia	Moscow	17.100	143.50
IN	India	New Delhi	3.286	1252.00
CH	China	Beijing	9.597	1357.00
SA	South Africa	Pretoria	1.221	52.98

Loop on data structure

Pandas DataFrame

	country	capital	area	population
BR	Brazil	Brasilia	8.516	200.40
RU	Russia	Moscow	17.100	143.50
IN	India	New Delhi	3.286	1252.00
CH	China	Beijing	9.597	1357.00
SA	South Africa	Pretoria	1.221	52.98

```
import pandas as pd
brics = pd.read_csv("brics.csv", index_col = 0)
for val in brics :
    print(val)
```


Loop on data structure

	country	capital	area	population
BR	Brazil	Brasilia	8.516	200.40
RU	Russia	Moscow	17.100	143.50
IN	India	New Delhi	3.286	1252.00
CH	China	Beijing	9.597	1357.00
SA	South Africa	Pretoria	1.221	52.98

Pandas DataFrame

```
import pandas as pd
brics = pd.read_csv("brics.csv", index_col = 0)
for val in brics :
    print(val)
```

```
country
capital
area
population
```

Loop on data structure

	country	capital	area	population
BR	Brazil	Brasilia	8.516	200.40
RU	Russia	Moscow	17.100	143.50
IN	India	New Delhi	3.286	1252.00
CH	China	Beijing	9.597	1357.00
SA	South Africa	Pretoria	1.221	52.98

Pandas DataFrame

```
import pandas as pd
brics = pd.read_csv("brics.csv", index_col = 0)
for lab, row in brics.iterrows():
    print(lab)
    print(row)
```

Loop on data structure

	country	capital	area	population
BR	Brazil	Brasilia	8.516	200.40
RU	Russia	Moscow	17.100	143.50
IN	India	New Delhi	3.286	1252.00
CH	China	Beijing	9.597	1357.00
SA	South Africa	Pretoria	1.221	52.98

Pandas DataFrame

```
import pandas as pd
brics = pd.read_csv("brics.csv", index_col = 0)
for lab, row in brics.iterrows():
    print(lab)
    print(row)
```

```
BR
country      Brazil
capital      Brasilia
area         8.516
population   200.4
Name: BR, dtype: object
...
RU
country      Russia
capital      Moscow
area         17.1
```

Loop on data structure

	country	capital	area	population
BR	Brazil	Brasilia	8.516	200.40
RU	Russia	Moscow	17.100	143.50
IN	India	New Delhi	3.286	1252.00
CH	China	Beijing	9.597	1357.00
SA	South Africa	Pretoria	1.221	52.98

Pandas DataFrame

```
import pandas as pd
brics = pd.read_csv("brics.csv", index_col = 0)
for lab, row in brics.iterrows():
    print(lab + ": " + row["capital"])
```

Loop on data structure

	country	capital	area	population
BR	Brazil	Brasilia	8.516	200.40
RU	Russia	Moscow	17.100	143.50
IN	India	New Delhi	3.286	1252.00
CH	China	Beijing	9.597	1357.00
SA	South Africa	Pretoria	1.221	52.98

Pandas DataFrame

```
import pandas as pd
brics = pd.read_csv("brics.csv", index_col = 0)
for lab, row in brics.iterrows():
    print(lab + ": " + row["capital"])
```

```
BR: Brasilia
RU: Moscow
IN: New Delhi
CH: Beijing
SA: Pretoria
```

Loop on data structure

	country	capital	area	population
BR	Brazil	Brasilia	8.516	200.40
RU	Russia	Moscow	17.100	143.50
IN	India	New Delhi	3.286	1252.00
CH	China	Beijing	9.597	1357.00
SA	South Africa	Pretoria	1.221	52.98

Pandas DataFrame

```
import pandas as pd
brics = pd.read_csv("brics.csv", index_col = 0)
for lab, row in brics.iterrows() :
    # - Creating Series on every iteration
    brics.loc[lab, "name_length"] = len(row["country"])
print(brics)
```

Loop on data structure

	country	capital	area	population
BR	Brazil	Brasilia	8.516	200.40
RU	Russia	Moscow	17.100	143.50
IN	India	New Delhi	3.286	1252.00
CH	China	Beijing	9.597	1357.00
SA	South Africa	Pretoria	1.221	52.98

Pandas DataFrame

```
import pandas as pd
brics = pd.read_csv("brics.csv", index_col = 0)
for lab, row in brics.iterrows() :
    # - Creating Series on every iteration
    brics.loc[lab, "name_length"] = len(row["country"])
print(brics)
```

	country	capital	area	population	name_length
BR	Brazil	Brasilia	8.516	200.40	6
RU	Russia	Moscow	17.100	143.50	6
IN	India	New Delhi	3.286	1252.00	5
CH	China	Beijing	9.597	1357.00	5
SA	South Africa	Pretoria	1.221	52.98	12

Loop on data structure

	country	capital	area	population
BR	Brazil	Brasilia	8.516	200.40
RU	Russia	Moscow	17.100	143.50
IN	India	New Delhi	3.286	1252.00
CH	China	Beijing	9.597	1357.00
SA	South Africa	Pretoria	1.221	52.98

Pandas DataFrame

```
import pandas as pd
brics = pd.read_csv("brics.csv", index_col = 0)
brics["name_length"] = brics["country"].apply(len)
print(brics)
```


Loop on data structure

	country	capital	area	population
BR	Brazil	Brasilia	8.516	200.40
RU	Russia	Moscow	17.100	143.50
IN	India	New Delhi	3.286	1252.00
CH	China	Beijing	9.597	1357.00
SA	South Africa	Pretoria	1.221	52.98

Pandas DataFrame

```
import pandas as pd
brics = pd.read_csv("brics.csv", index_col = 0)
brics["name_length"] = brics["country"].apply(len)
print(brics)
```

	country	capital	area	population	name_length
BR	Brazil	Brasilia	8.516	200.40	6
RU	Russia	Moscow	17.100	143.50	6
IN	India	New Delhi	3.286	1252.00	5
CH	China	Beijing	9.597	1357.00	5
SA	South Africa	Pretoria	1.221	52.98	12

- 1- Open VSC
- 2- Open *PYTHON-101/code/02-intermediate/13-loop_dataframe.py*
- 3- Play with it

Let's practice