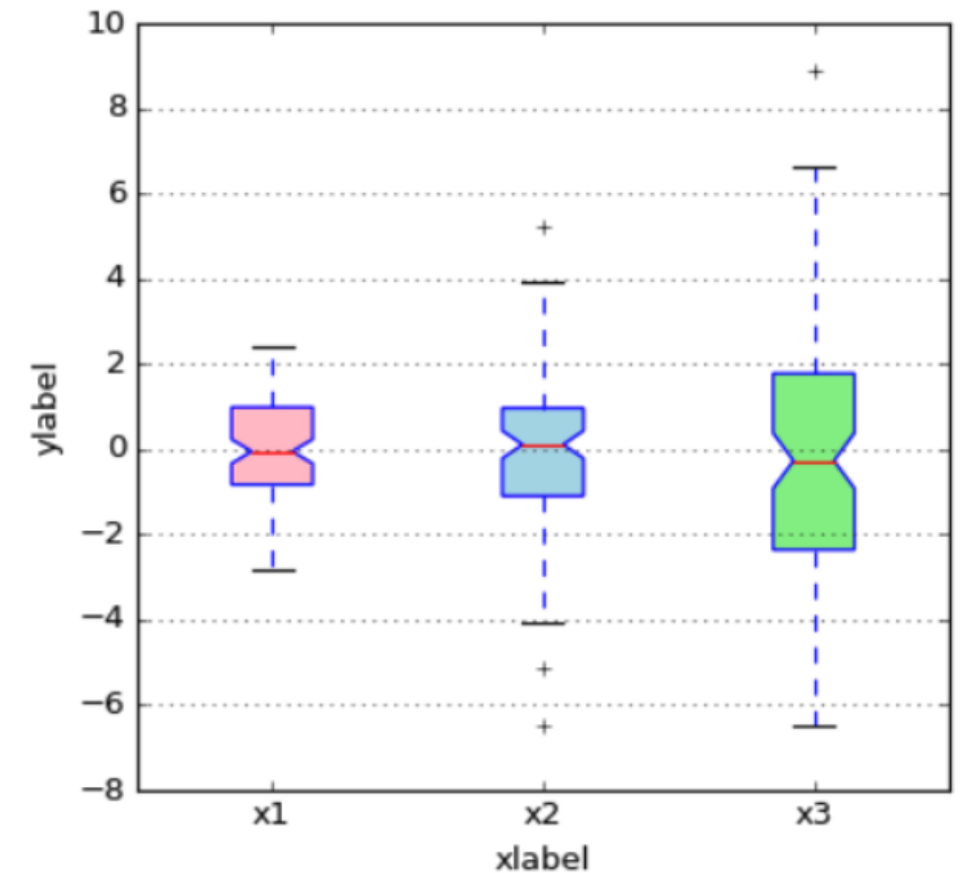
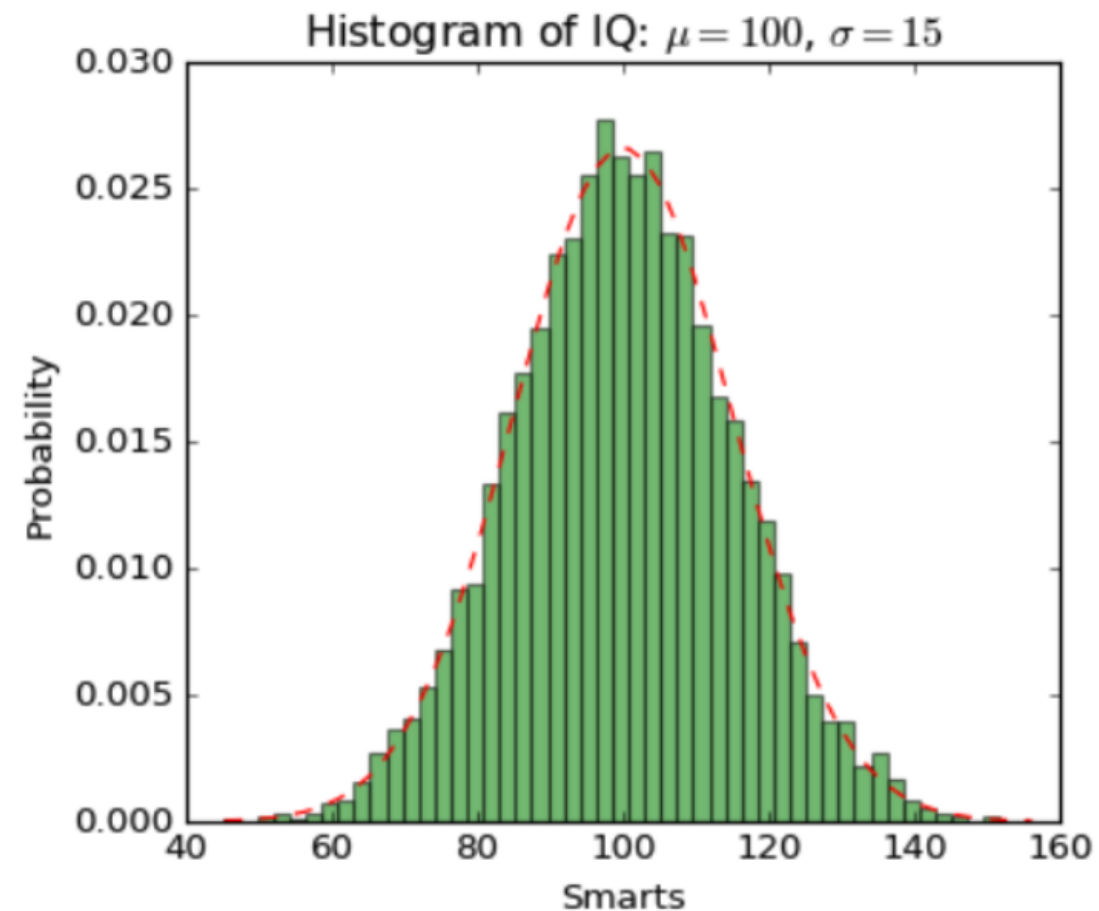


Intermediate Python

Matplotlib

Data Visualization

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

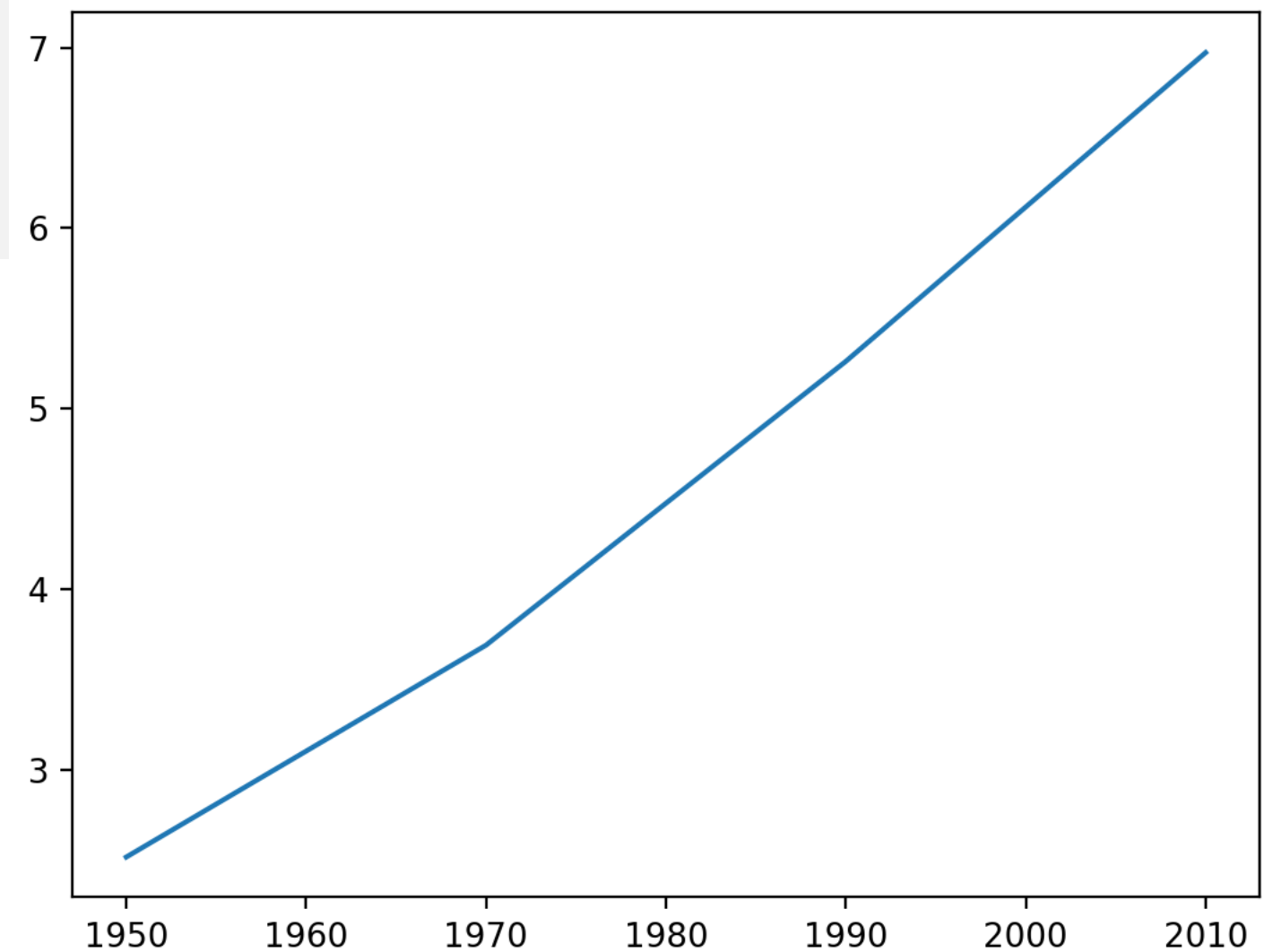


Matplotlib

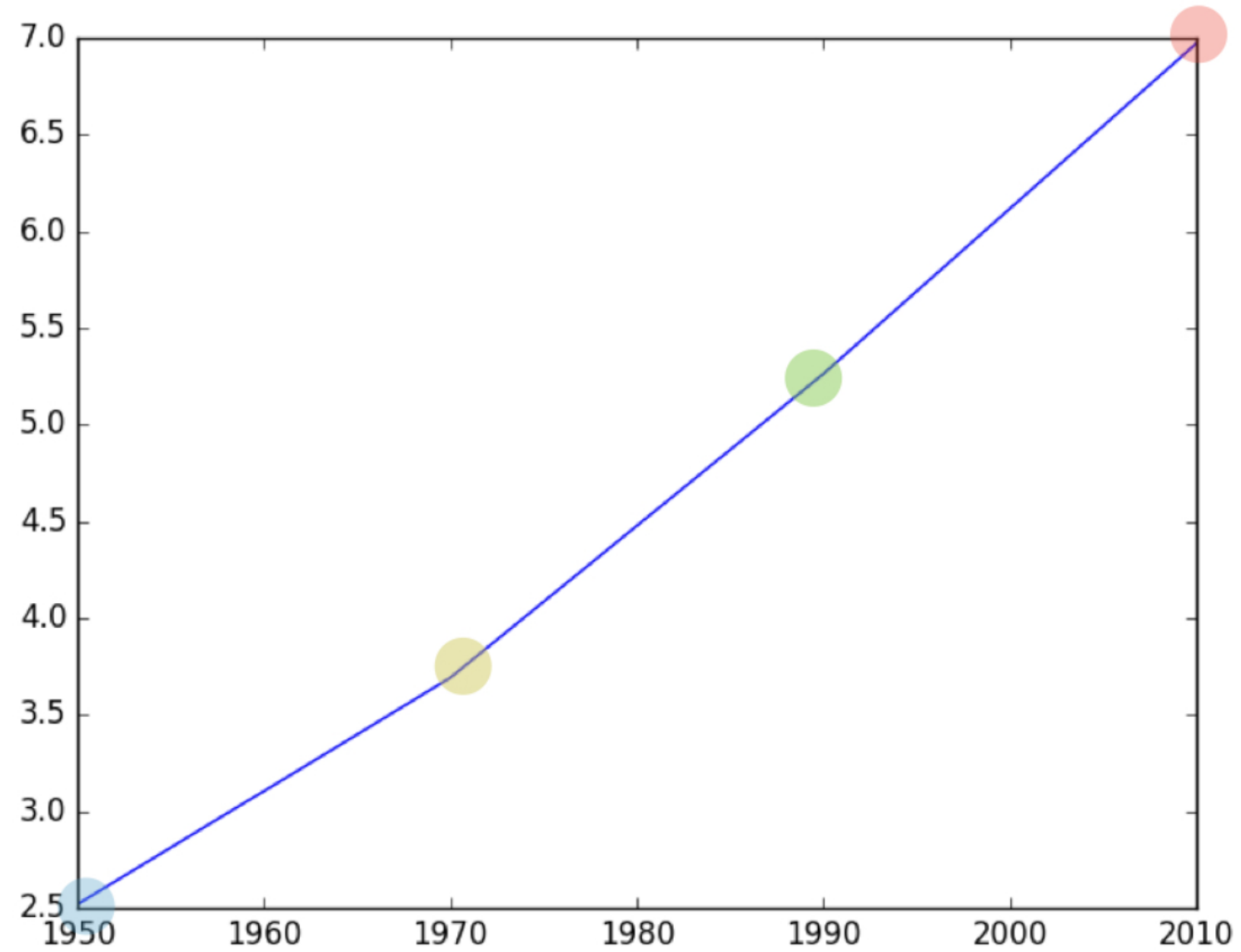
```
import matplotlib.pyplot as plt
year = [1950, 1970, 1990, 2010]
pop = [2.519, 3.692, 5.263, 6.972]
plt.plot(year, pop)
plt.show()
```

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

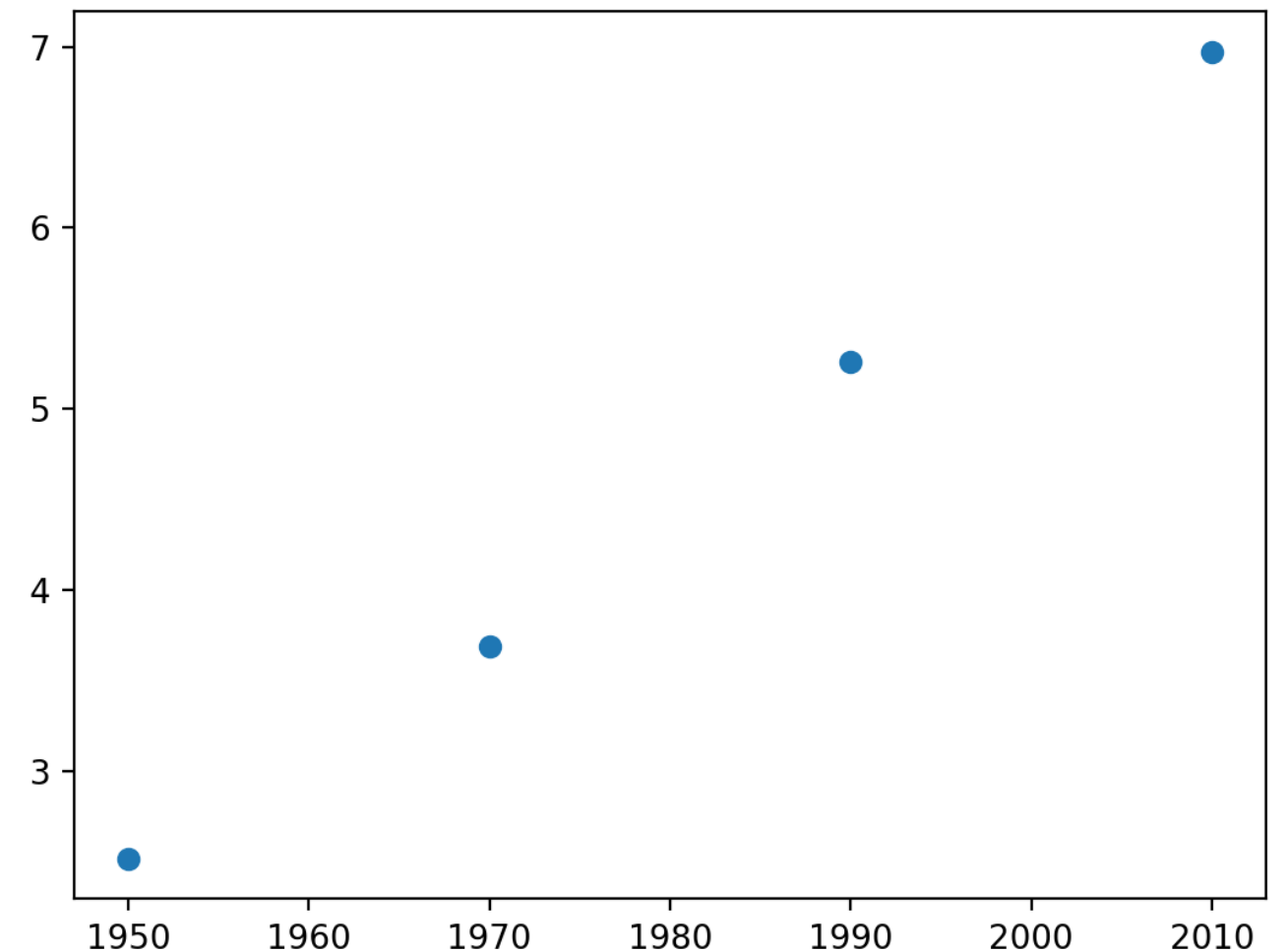
Scatter plot

```
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()
```



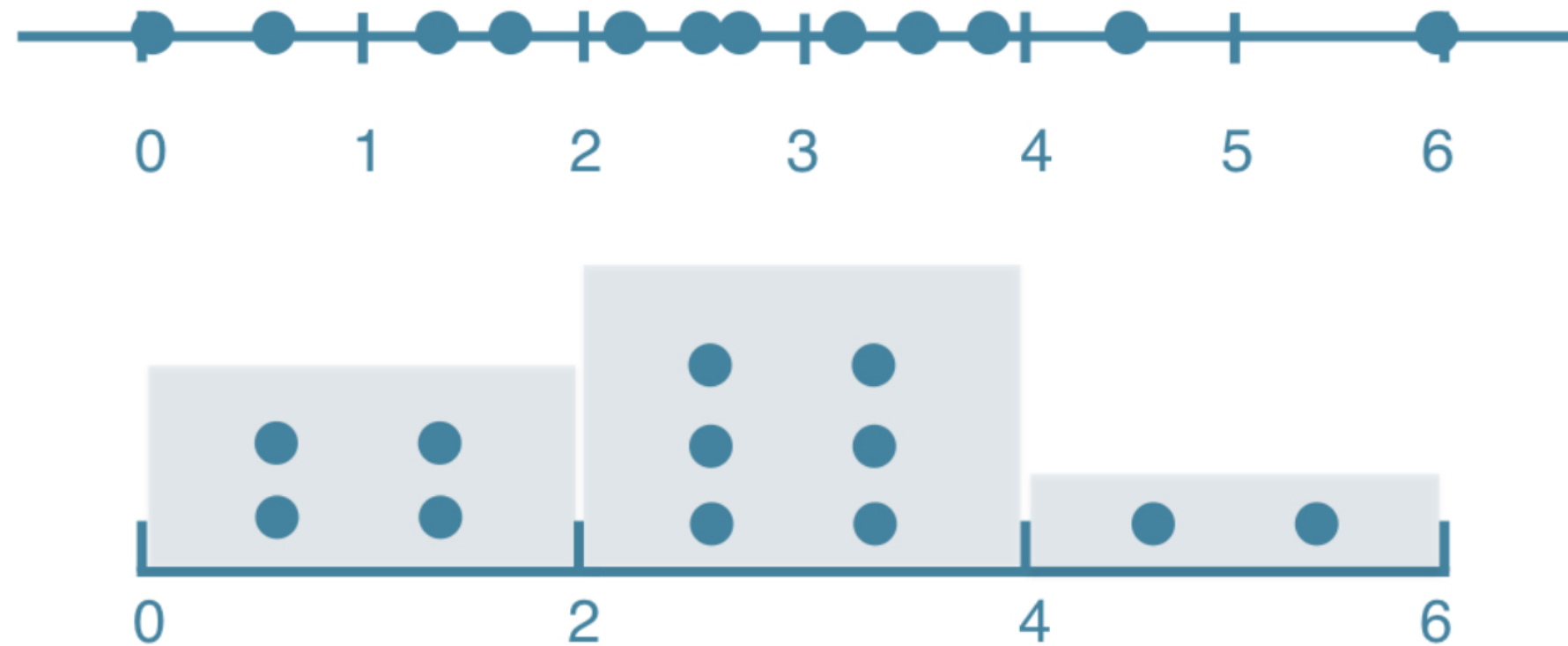
- 1- Open VSC
- 2- Open *PYTHON-101/code/02-intermediate/01-basic_plots.py*
- 3- Play with it

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

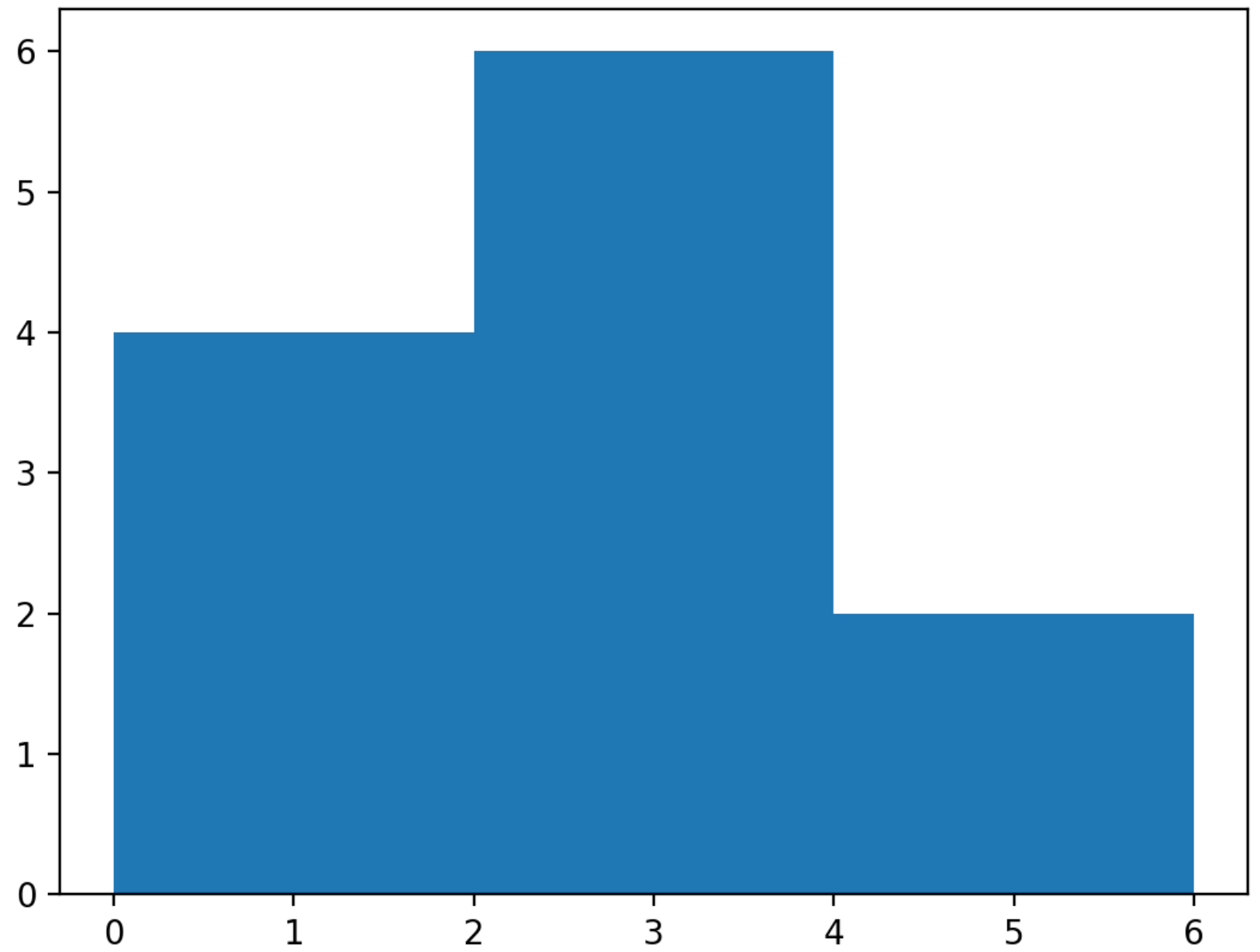
Histogram

```
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

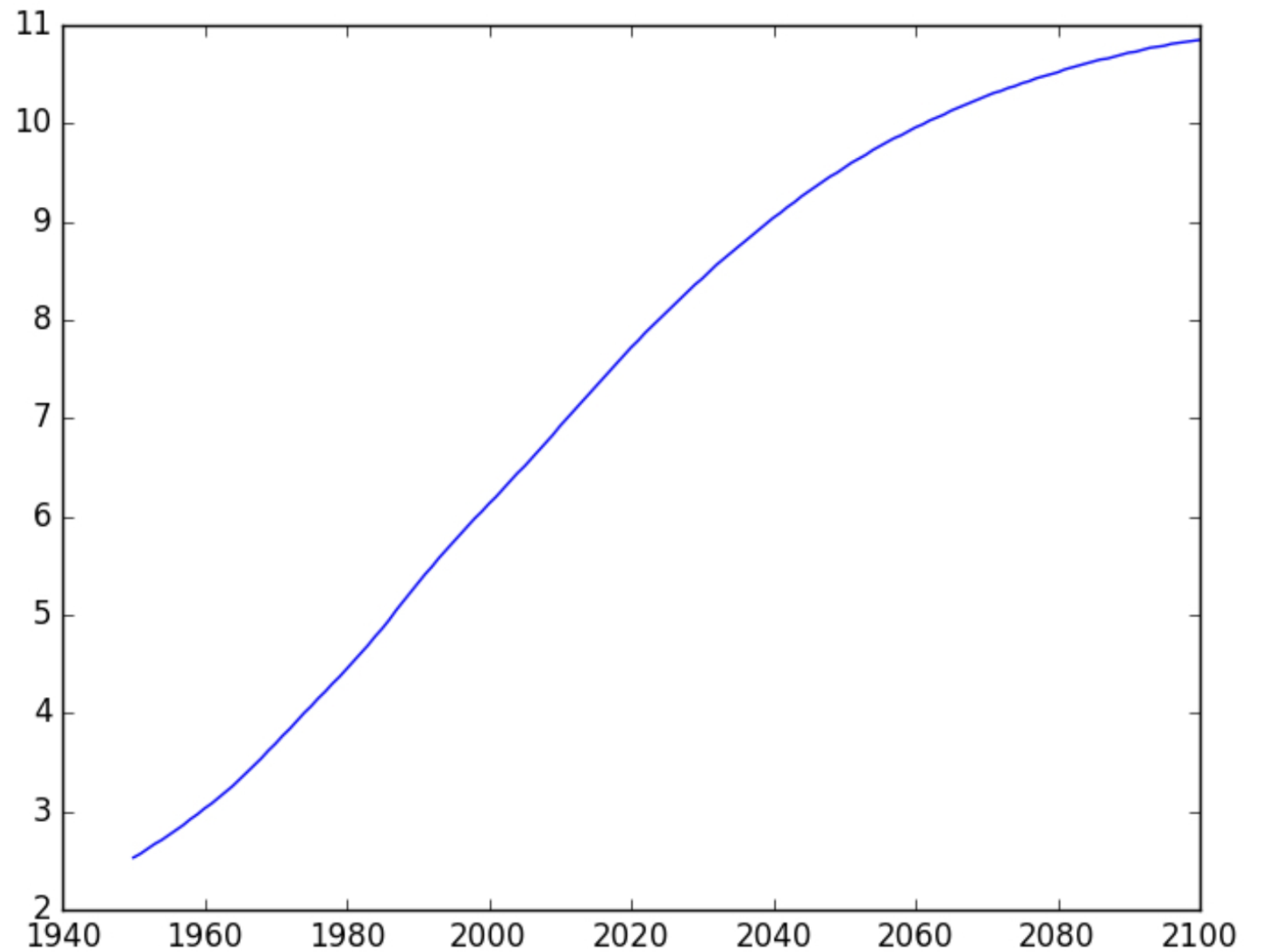
Customization

- 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

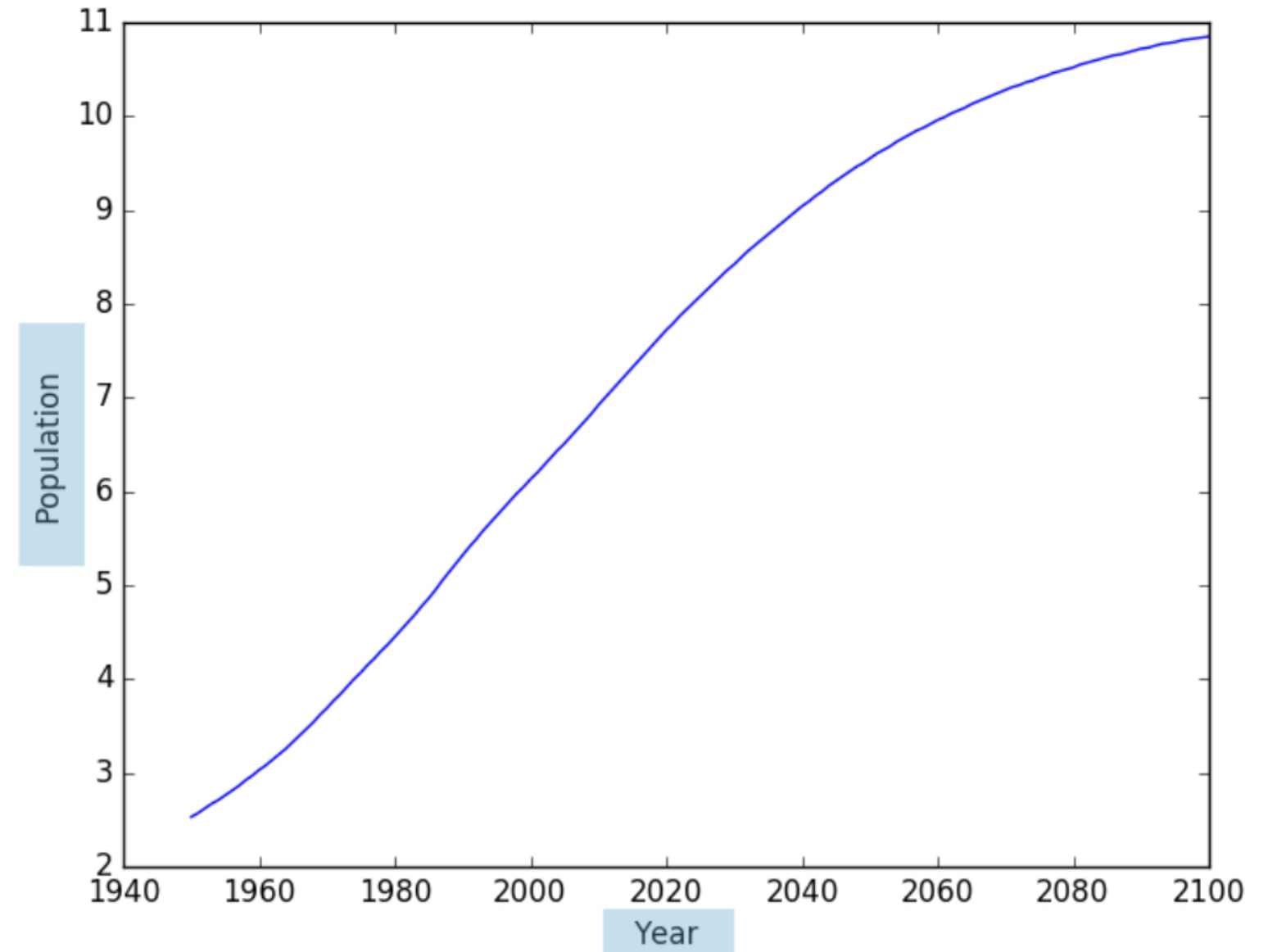
Customization

```
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

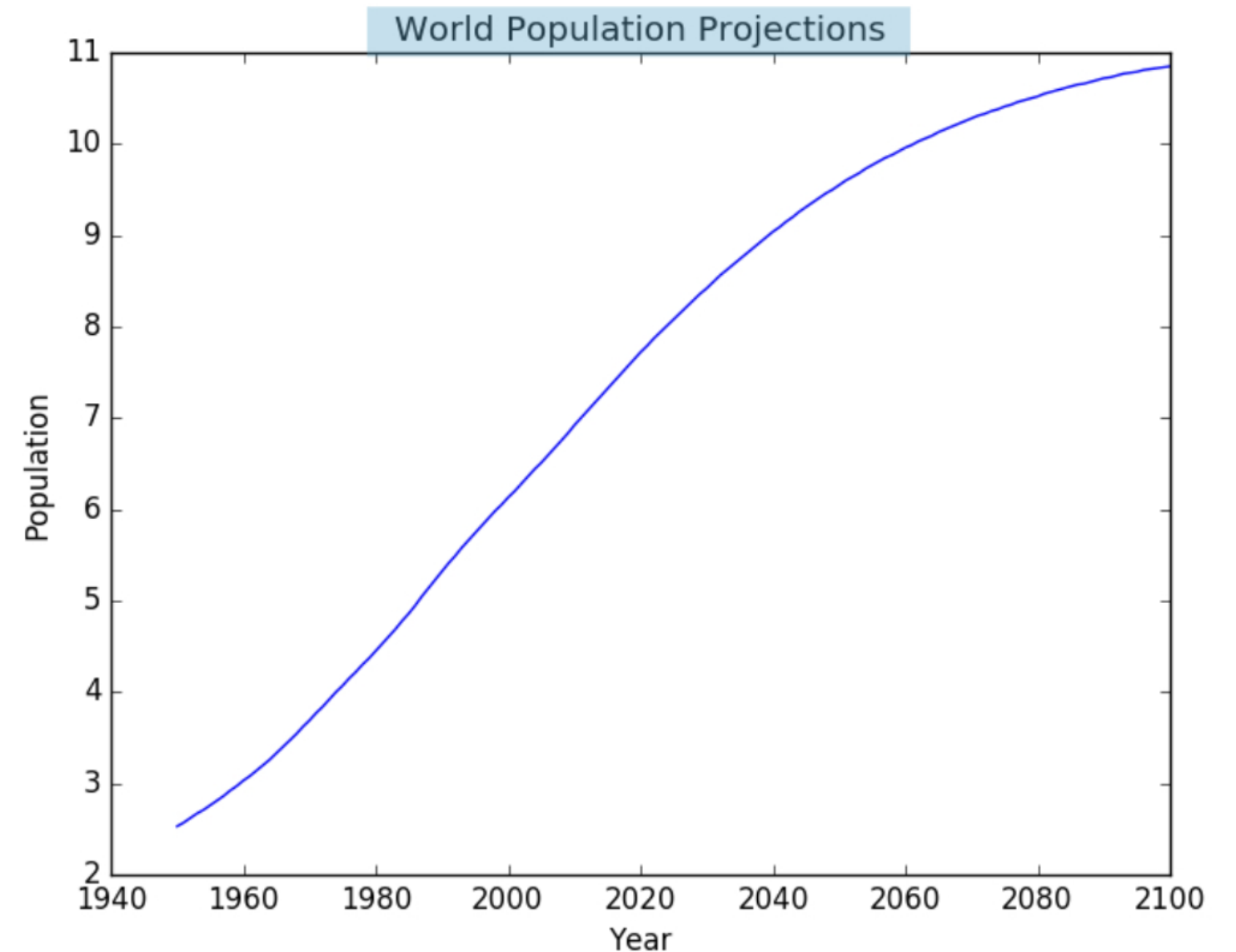
Customization

```
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

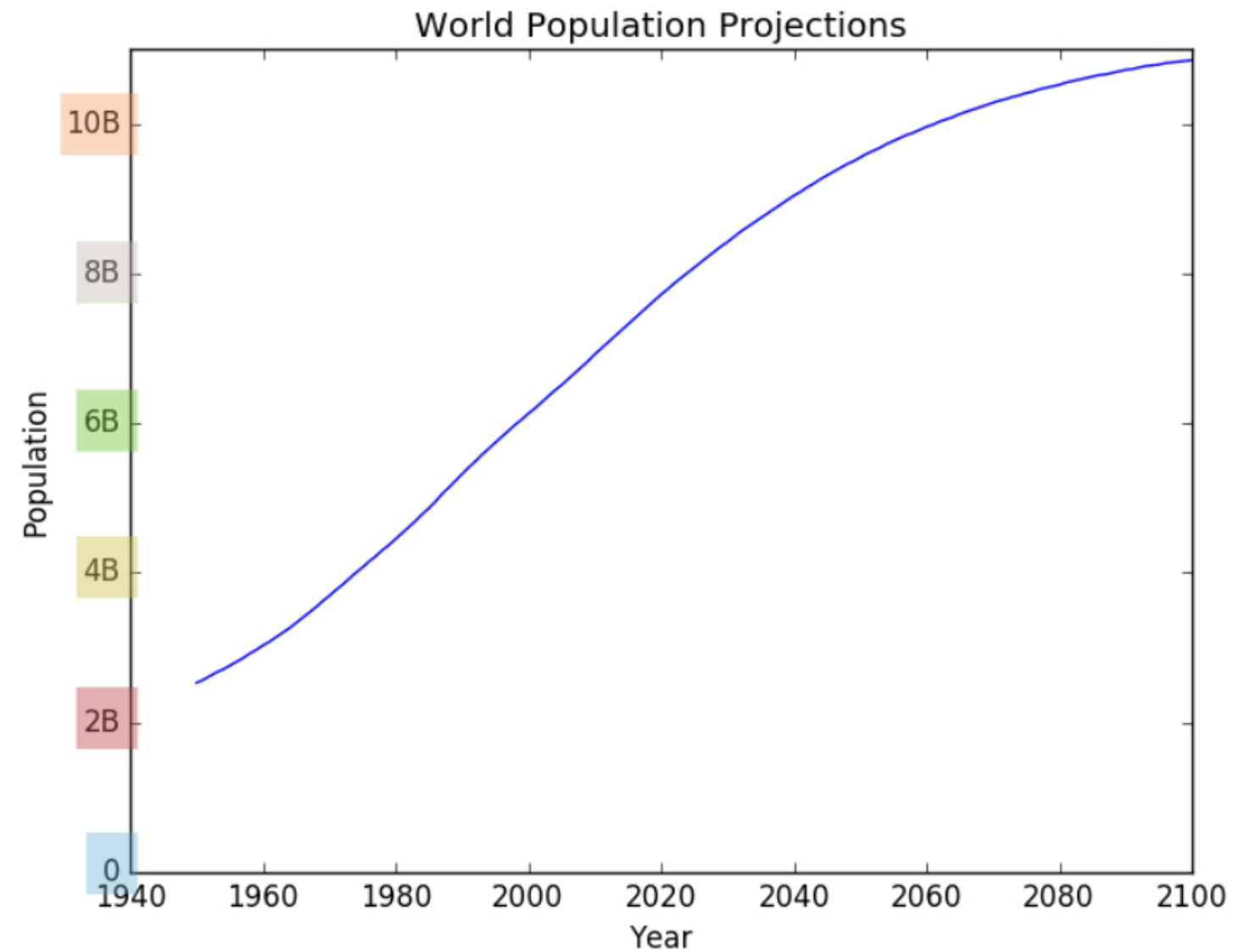
```
import matplotlib.pyplot as plt
year = [1950, 1951, 1952, ..., 2100]
pop = [2.538, 2.57, 2.62, ..., 10.85]

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]

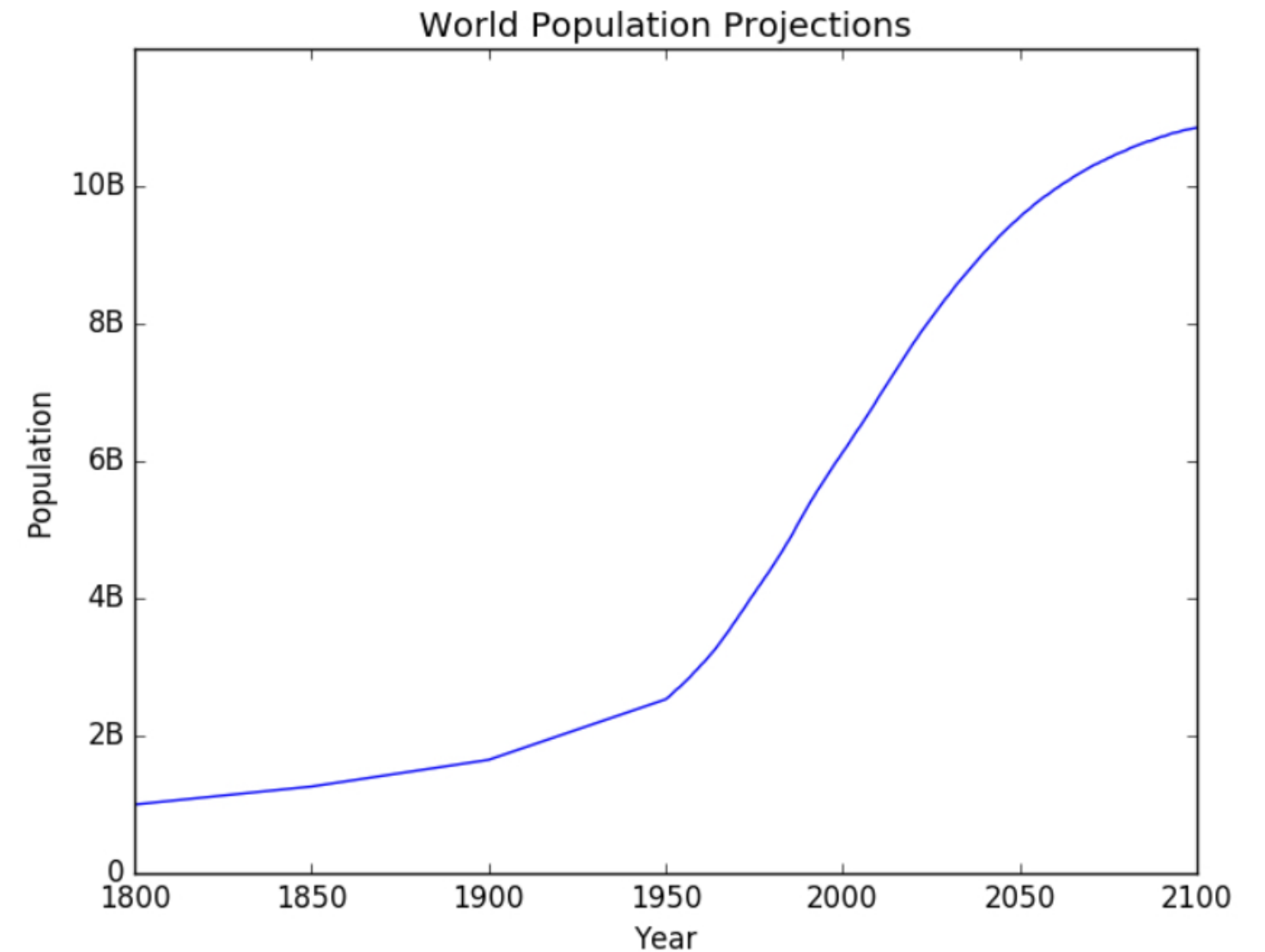
# Add more data
year = [1800, 1850, 1900] + year
pop = [1.0, 1.262, 1.650] + pop

plt.plot(year, pop)

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

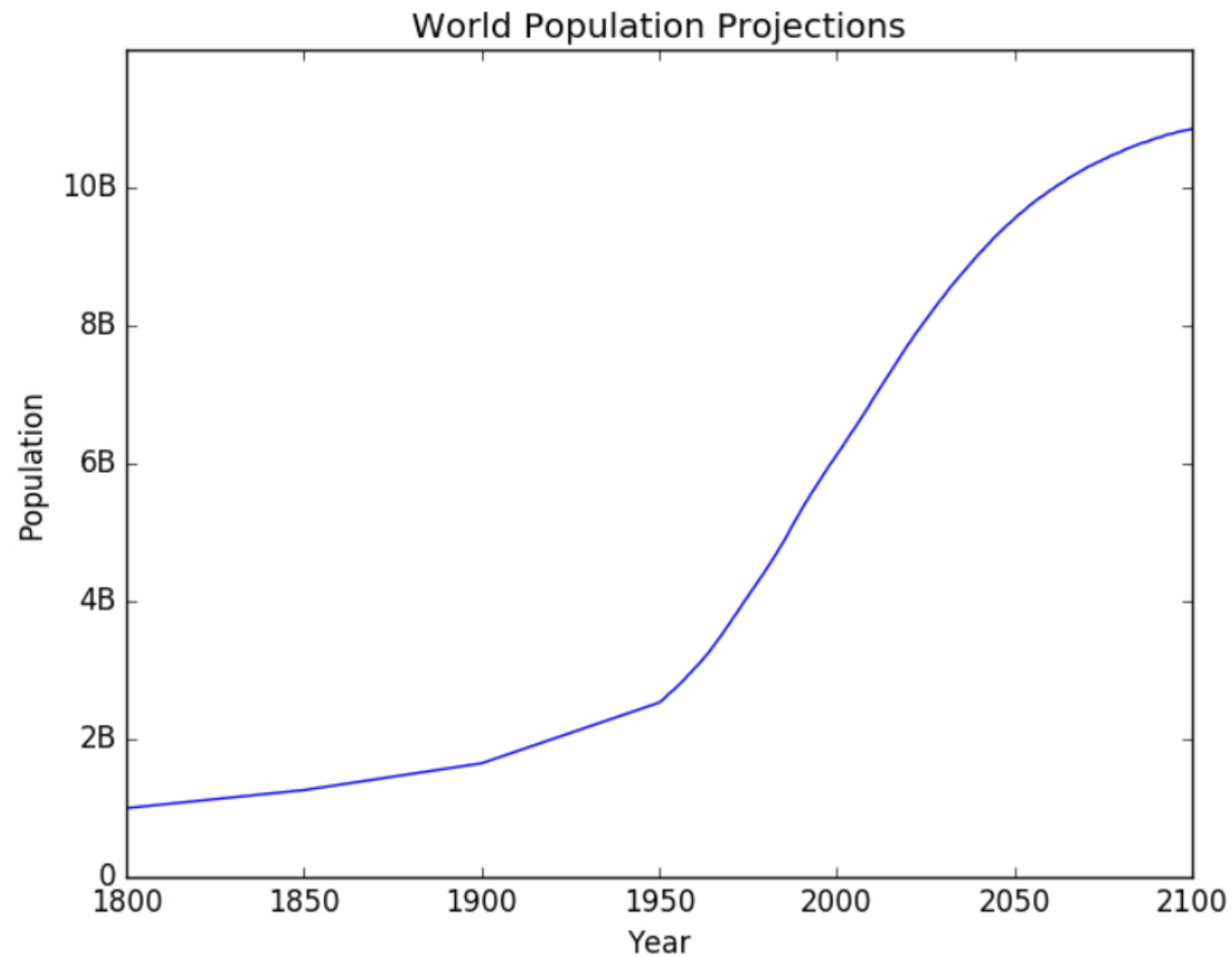
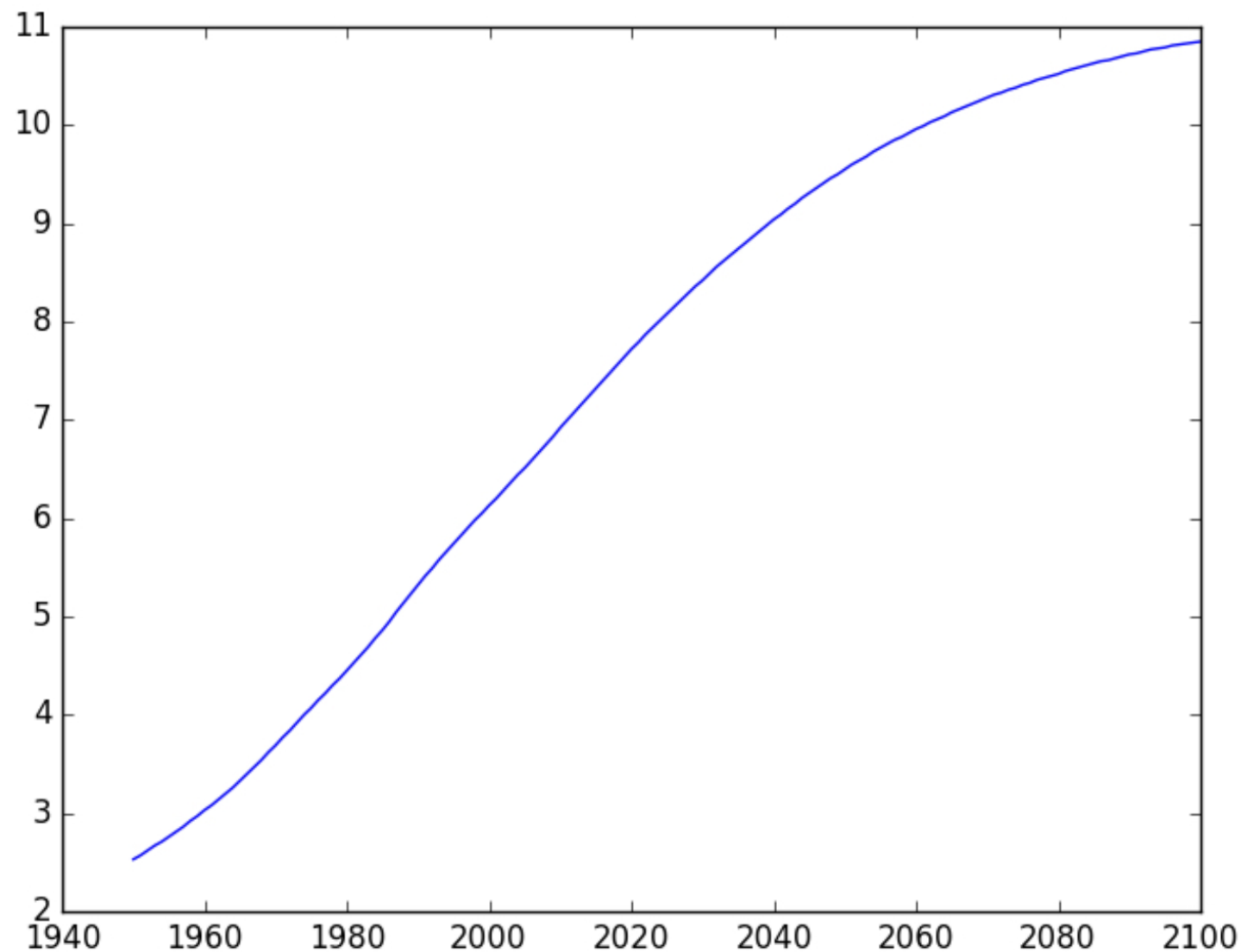
plt.yticks([0, 2, 4, 6, 8, 10],
           ['0', '2B', '4B', '6B', '8B', '10B'])

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'] = 0.0000027  
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

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 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