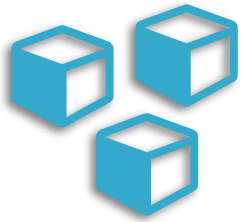
INTERMEDIATE PYTHON FOR DATA SCIENCE

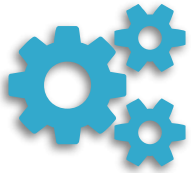
Basic Plots with Matplotlib

- Visualization

- Data Structures



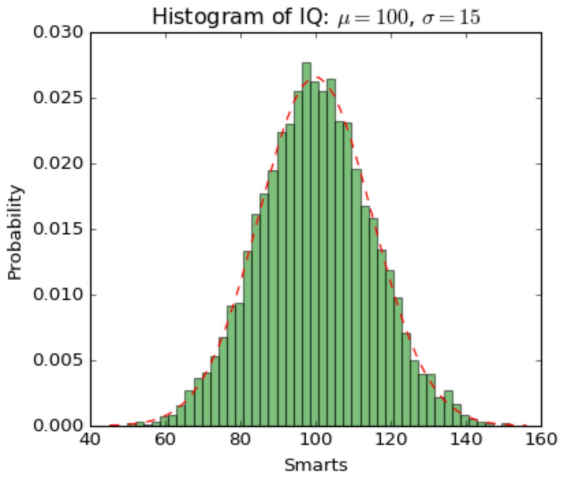
- Control Structures

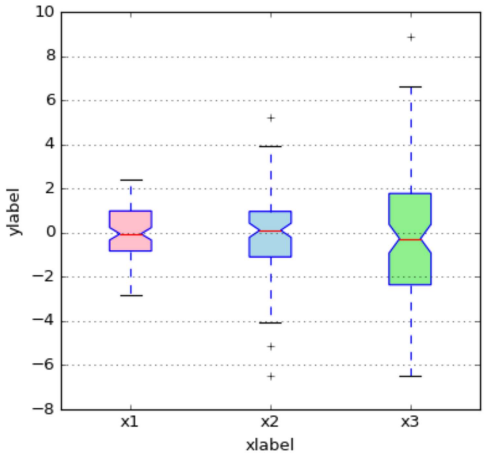


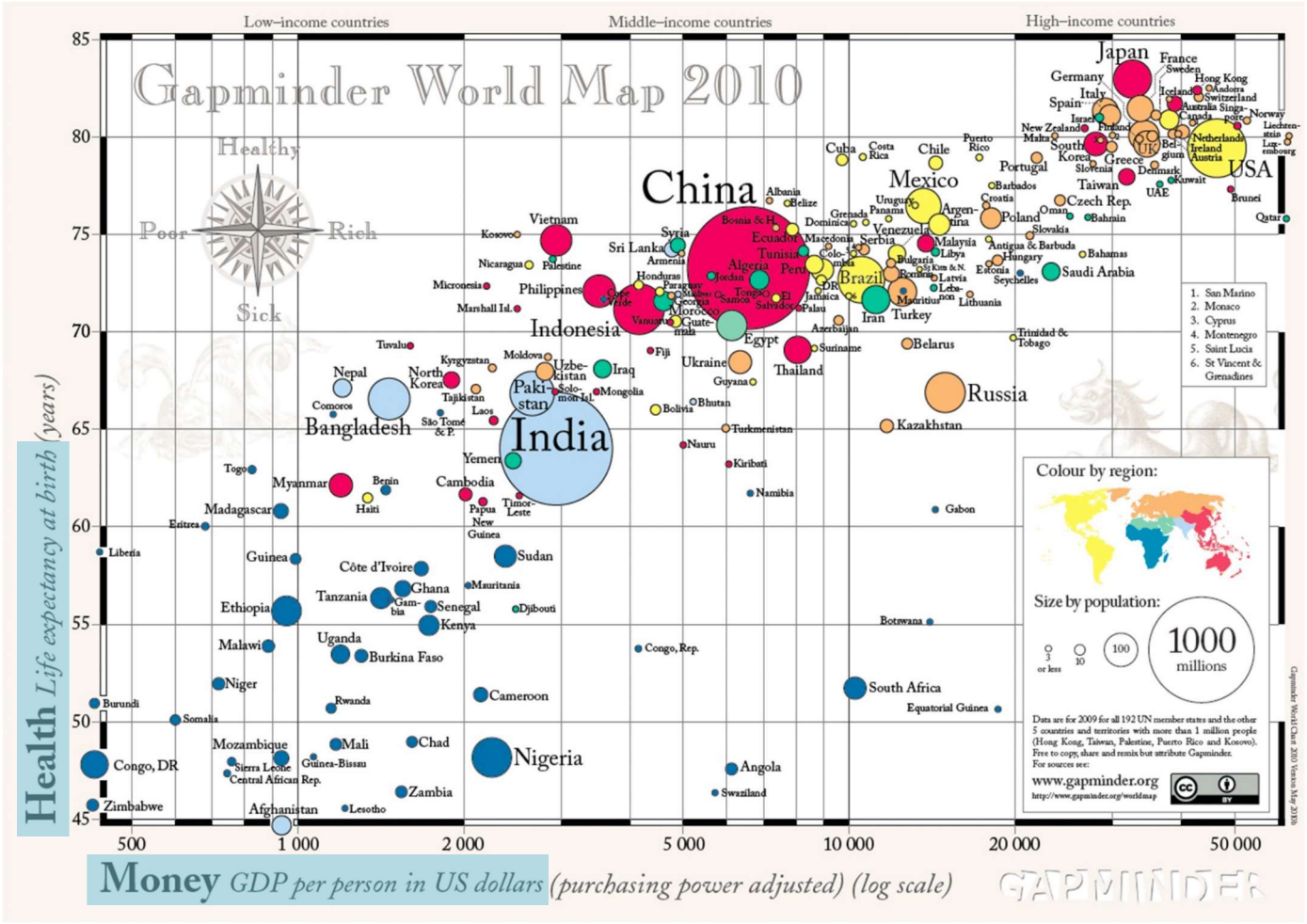
- Case Study

Data Visualization

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



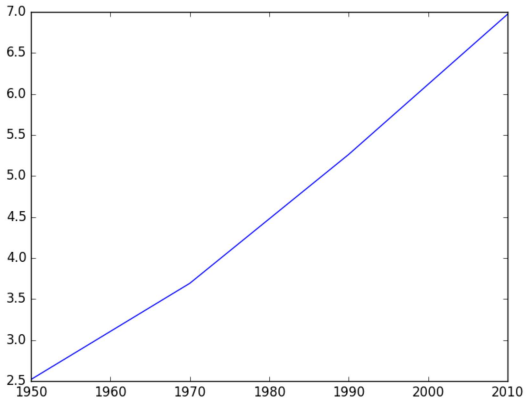




Source: GapMinder, Wealth and Health of Nations

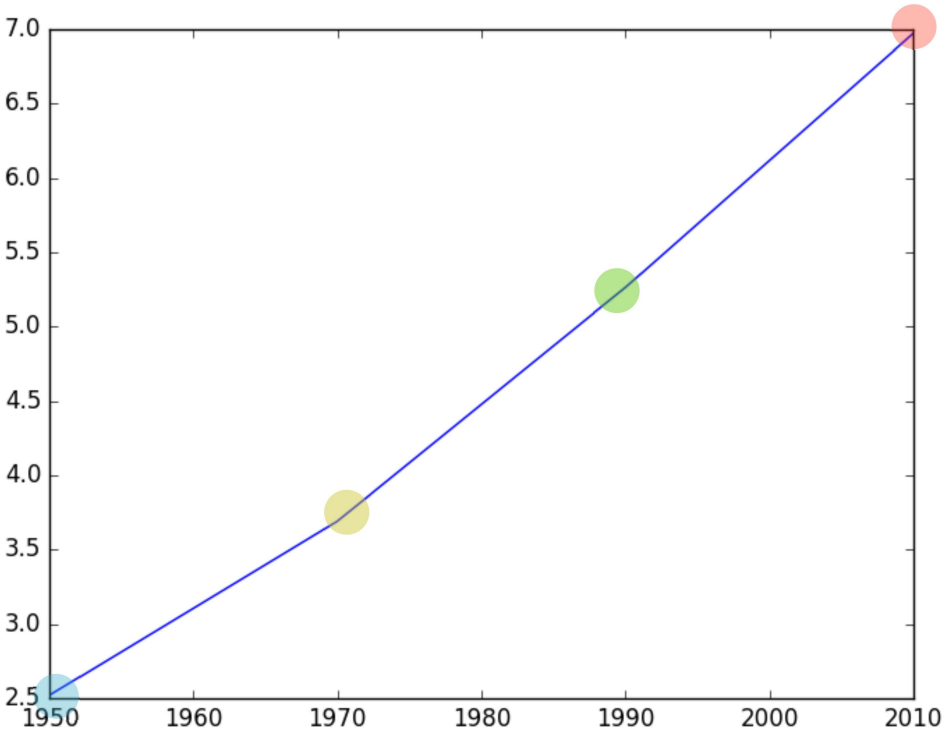
Matplotlib

```
In [1]: import matplotlib.pyplot as plt

In [2]: year = [1950, 1970, 1990, 2010]

In [3]: pop = [2.519, 3.692, 5.263, 6.972]

In [4]: plt.plot(year, pop)
                 x     y
In [5]: plt.show()
```



Matplotlib

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

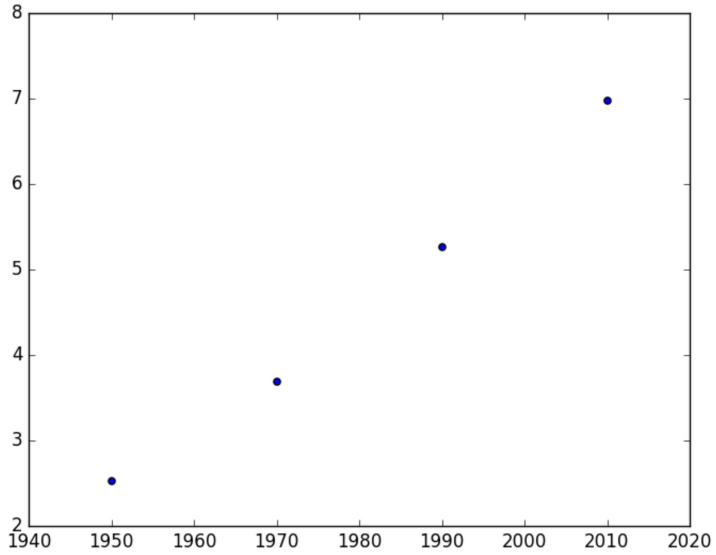


Scatter plot

```
In [1]: import matplotlib.pyplot as plt
In [2]: year = [1950, 1970, 1990, 2010]
In [3]: pop = [2.519, 3.692, 5.263, 6.972]
In [4]: plt.plot(year, pop)
In [5]: plt.show()
```

Scatter plot

```
In [1]: import matplotlib.pyplot as plt
In [2]: year = [1950, 1970, 1990, 2010]
In [3]: pop = [2.519, 3.692, 5.263, 6.972]
In [4]: plt.scatter(year, pop)
In [5]: plt.show()
```



INTERMEDIATE PYTHON FOR DATA SCIENCE

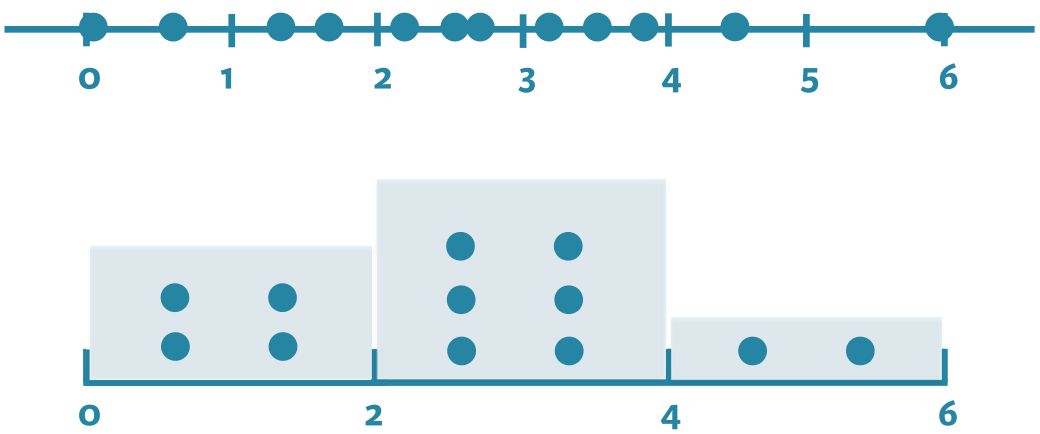
Let's practice!

INTERMEDIATE PYTHON FOR DATA SCIENCE

Histogram

Histogram

- Explore dataset
- Get idea about distribution

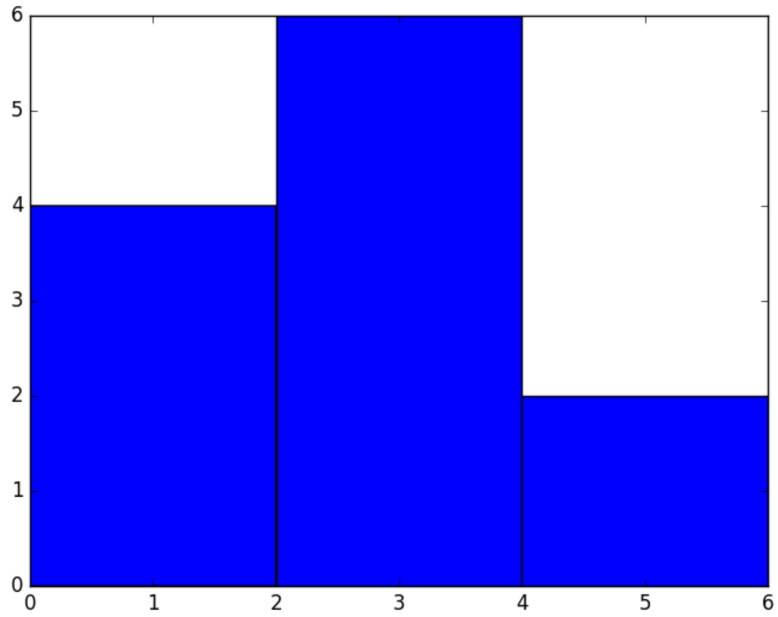


Matplotlib

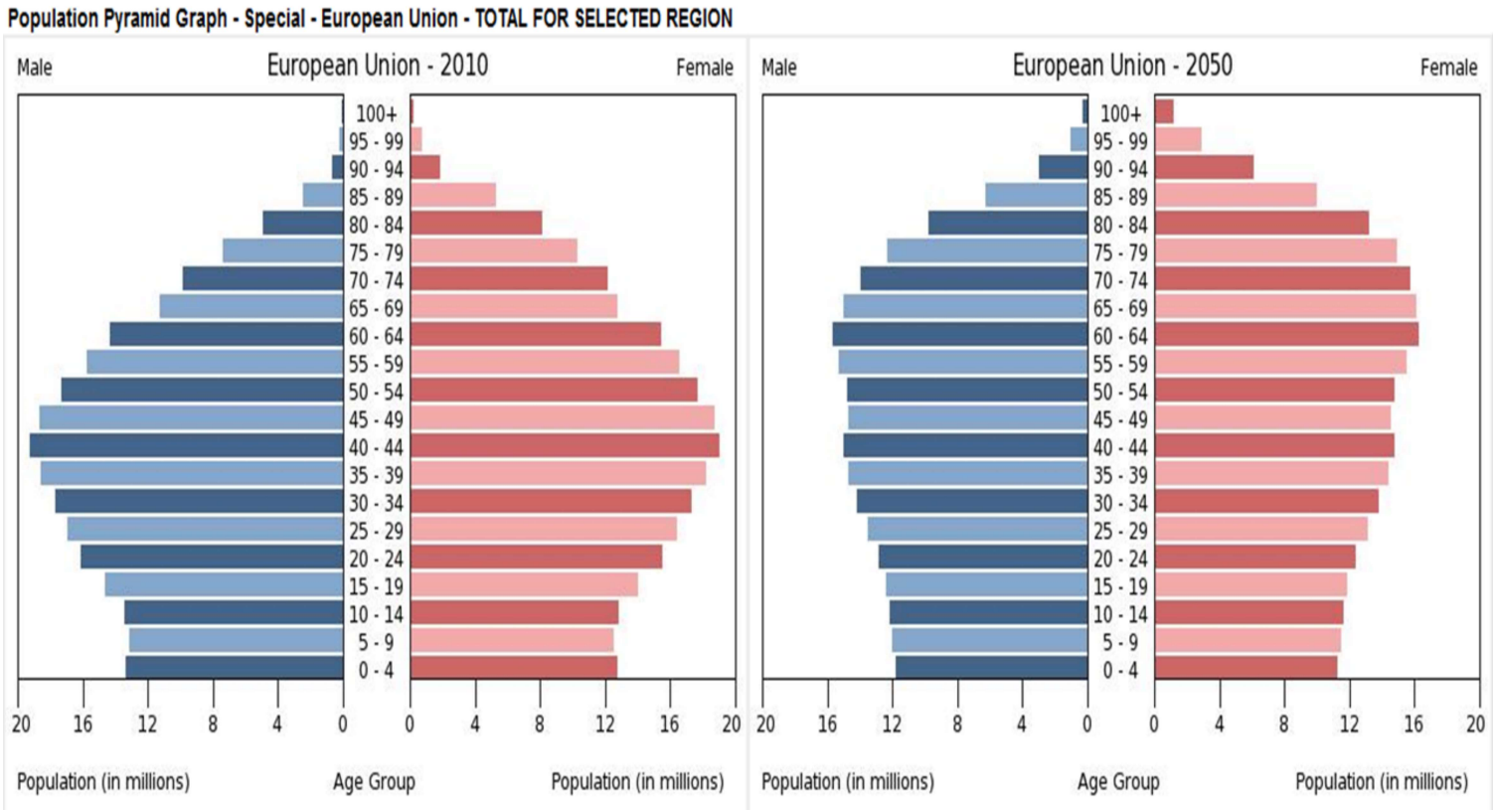
```
In [1]: import matplotlib.pyplot as plt

In [2]: help(plt.hist)

  Help on function hist in module matplotlib.pyplot:

  hist(x, bins=10, range=None, normed=False, weights=None,
  cumulative=False, bottom=None, histtype='bar', align='mid',
  orientation='vertical', rwidth=None, log=False, color=None,
  label=None, stacked=False, hold=None, data=None, **kwargs)
      Plot a histogram.

      Compute and draw the histogram of *x*. The return value is a
      tuple (*n*, *bins*, *patches*) or ([*n0*, *n1*, ...],
      *bins*, [*patches0*, *patches1*,...]) if the input contains
      multiple data.

  ...
```

Matplotlib example

```
In [3]: values = [0,0.6,1.4,1.6,2.2,2.5,2.6,3.2,3.5,3.9,4.2,6]
In [4]: plt.hist(values, bins = 3)
In [5]: plt.show()
```



Population Pyramid



INTERMEDIATE PYTHON FOR DATA SCIENCE

Let's practice!

INTERMEDIATE PYTHON FOR DATA SCIENCE

Customization

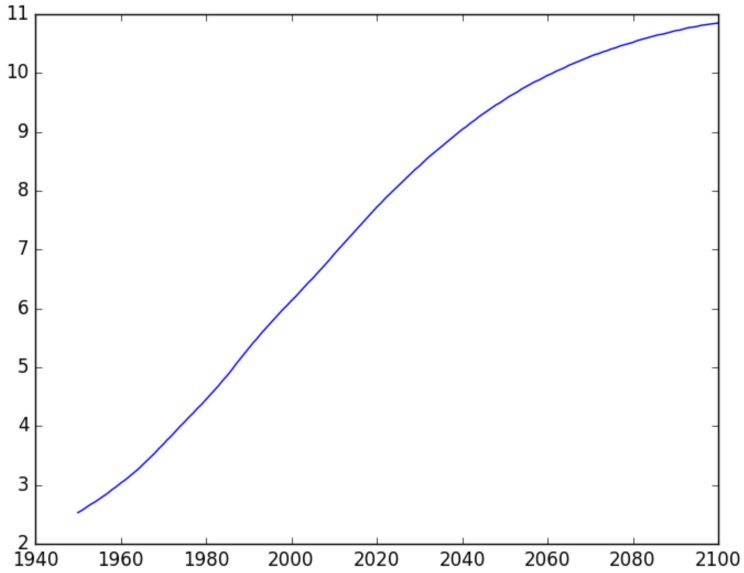
Data Visualization

- Many options
  - Different plot types
  - Many customizations
- Choice depends on
  - Data
  - Story you want to tell

Basic Plot

population.py

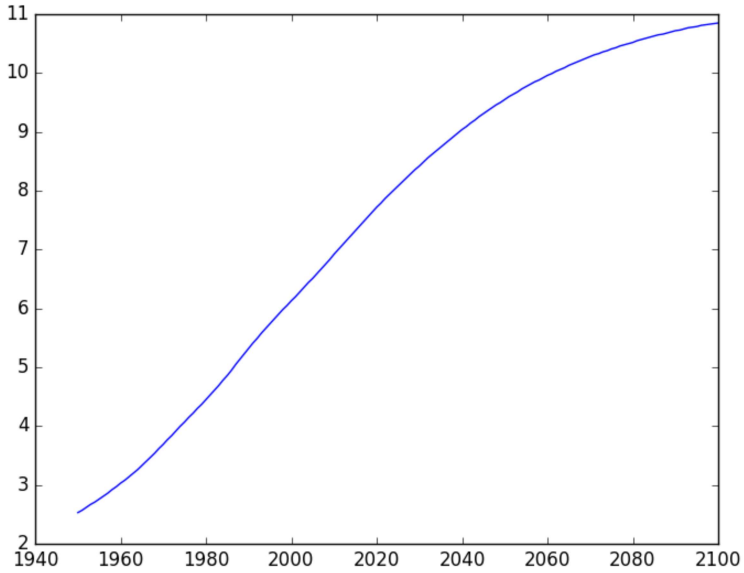
```
import matplotlib.pyplot as plt
year = [1950, 1951, 1952, ..., 2100]
pop = [2.538, 2.57, 2.62, ..., 10.85]

plt.plot(year, pop)

plt.show()
```



Axis labels

population.py

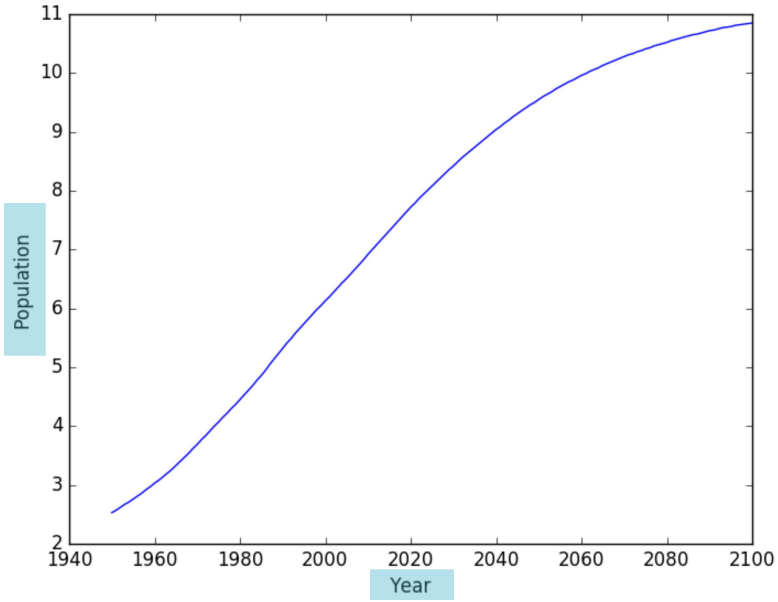
```
import matplotlib.pyplot as plt
year = [1950, 1951, 1952, ..., 2100]
pop = [2.538, 2.57, 2.62, ..., 10.85]

plt.plot(year, pop)

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

plt.show()
```



Axis labels

population.py

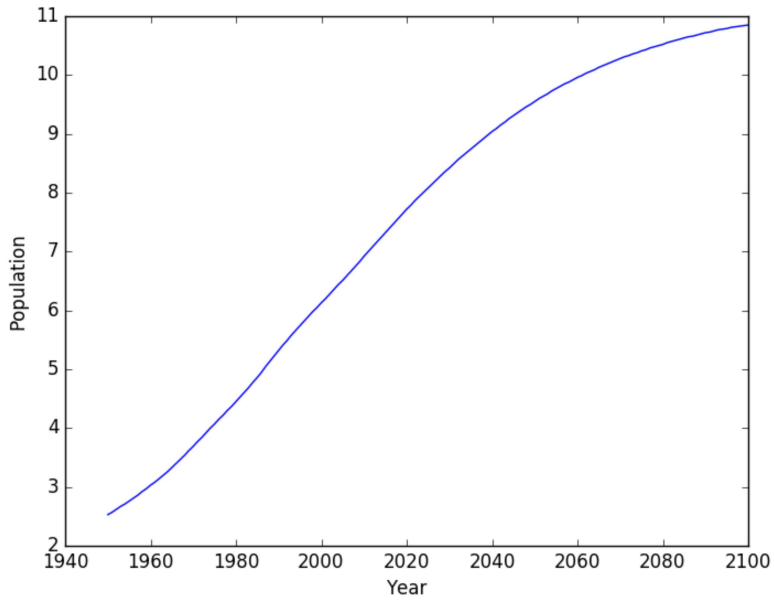
```
import matplotlib.pyplot as plt
year = [1950, 1951, 1952, ..., 2100]
pop = [2.538, 2.57, 2.62, ..., 10.85]

plt.plot(year, pop)

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

plt.show()
```



Title

population.py

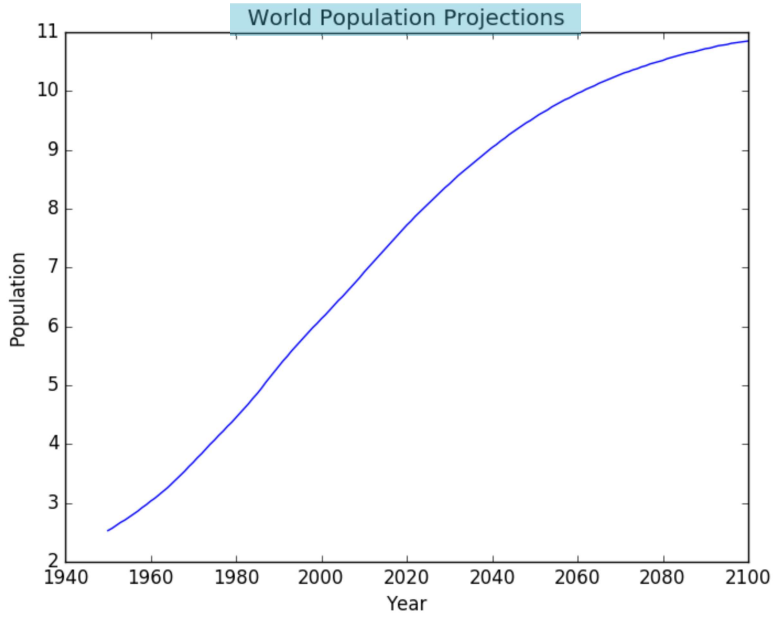
```
import matplotlib.pyplot as plt
year = [1950, 1951, 1952, ..., 2100]
pop = [2.538, 2.57, 2.62, ..., 10.85]

plt.plot(year, pop)

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

plt.show()
```



Title

population.py

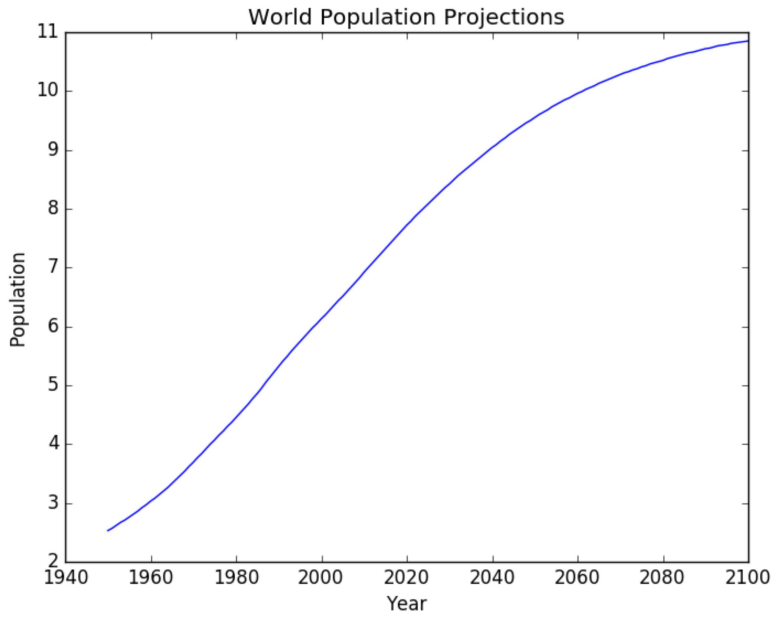
```
import matplotlib.pyplot as plt
year = [1950, 1951, 1952, ..., 2100]
pop = [2.538, 2.57, 2.62, ..., 10.85]

plt.plot(year, pop)

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

plt.show()
```



Ticks

population.py

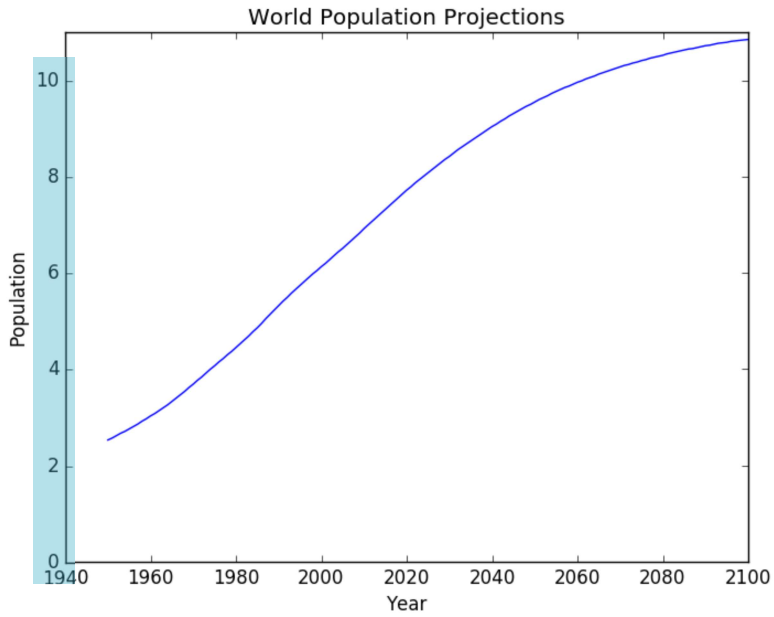
```
import matplotlib.pyplot as plt
year = [1950, 1951, 1952, ..., 2100]
pop = [2.538, 2.57, 2.62, ..., 10.85]

plt.plot(year, pop)

plt.xlabel('Year')
plt.ylabel('Population')
plt.title('World Population Projections')
plt.yticks([0, 2, 4, 6, 8, 10])

plt.show()
```



Ticks

population.py

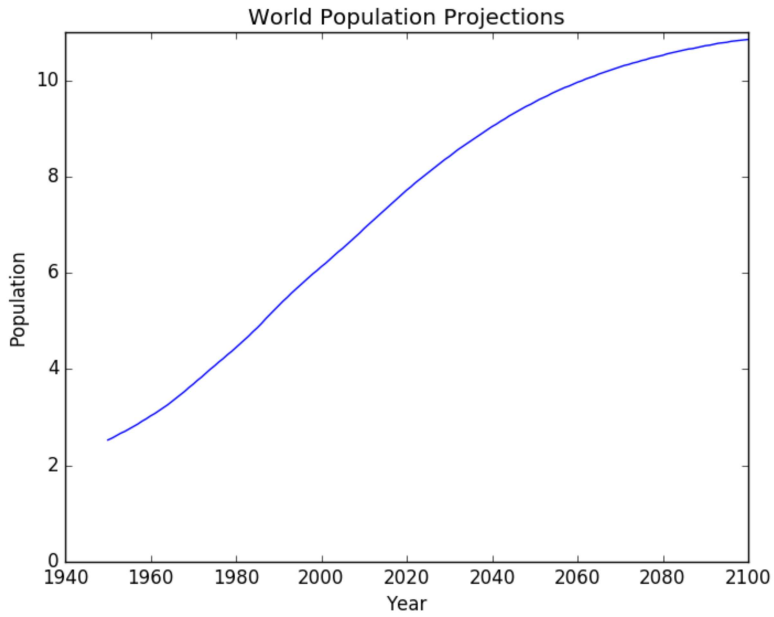
```
import matplotlib.pyplot as plt
year = [1950, 1951, 1952, ..., 2100]
pop = [2.538, 2.57, 2.62, ..., 10.85]

plt.plot(year, pop)

plt.xlabel('Year')
plt.ylabel('Population')
plt.title('World Population Projections')
plt.yticks([0, 2, 4, 6, 8, 10])

plt.show()
```



Ticks (2)

population.py

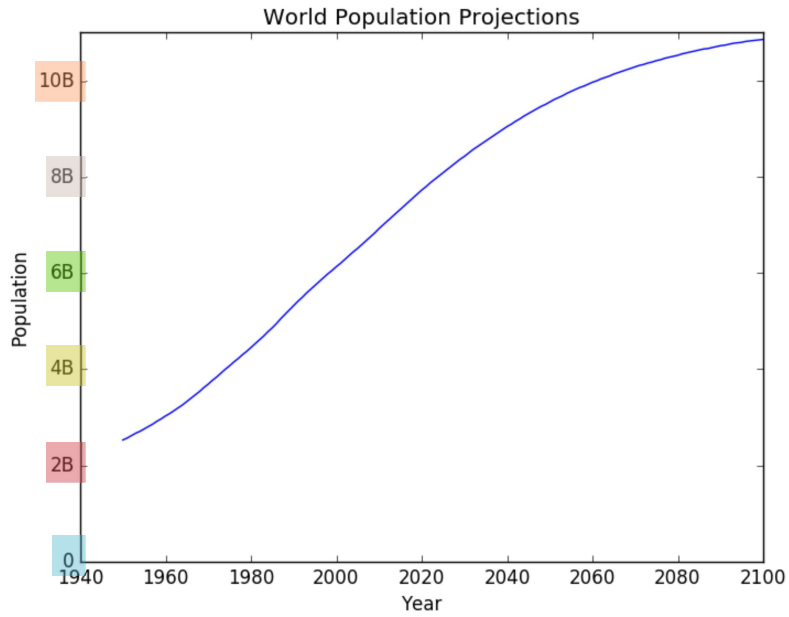
```
import matplotlib.pyplot as plt
year = [1950, 1951, 1952, ..., 2100]
pop = [2.538, 2.57, 2.62, ..., 10.85]

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()
```



Ticks (2)

population.py

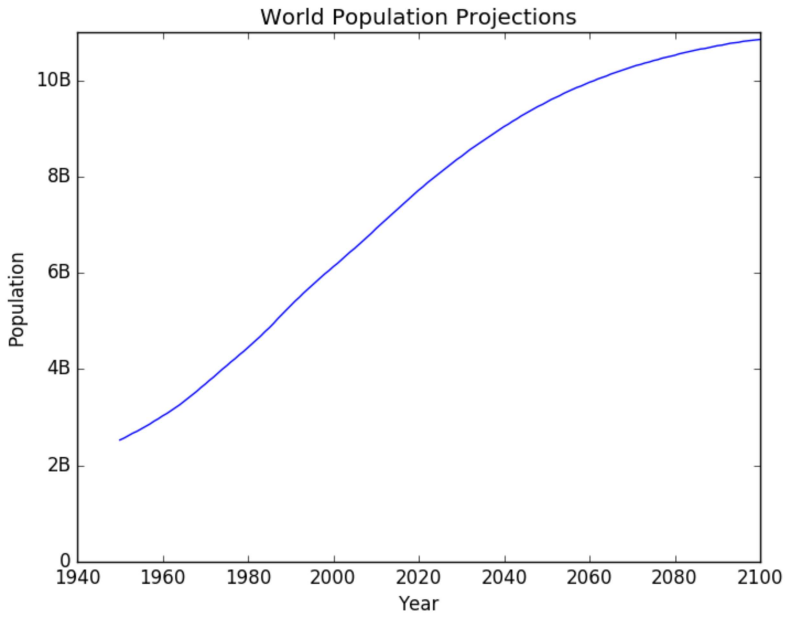
```
import matplotlib.pyplot as plt
year = [1950, 1951, 1952, ..., 2100]
pop = [2.538, 2.57, 2.62, ..., 10.85]

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()
```



Add historical data

population.py

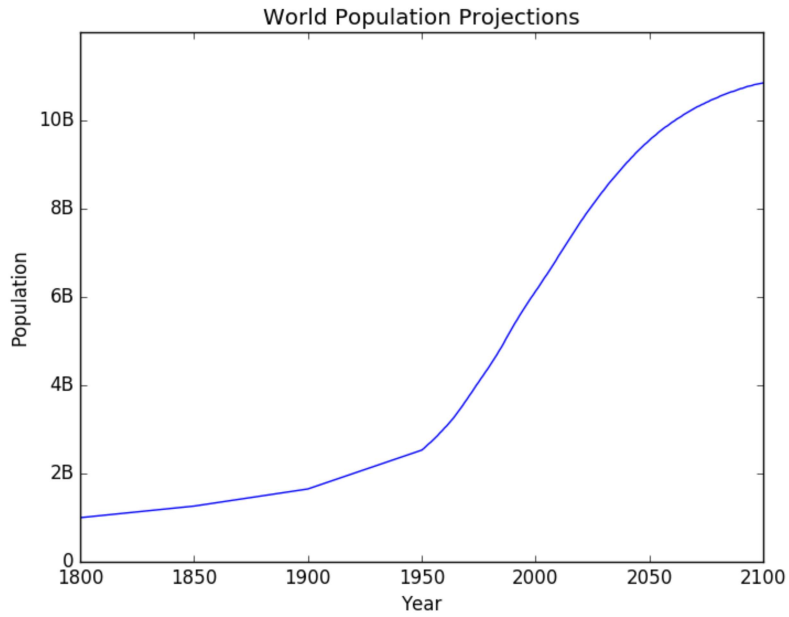
```
import matplotlib.pyplot as plt
year = [1950, 1951, 1952, ..., 2100]
pop = [2.538, 2.57, 2.62, ..., 10.85]

# 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()
```



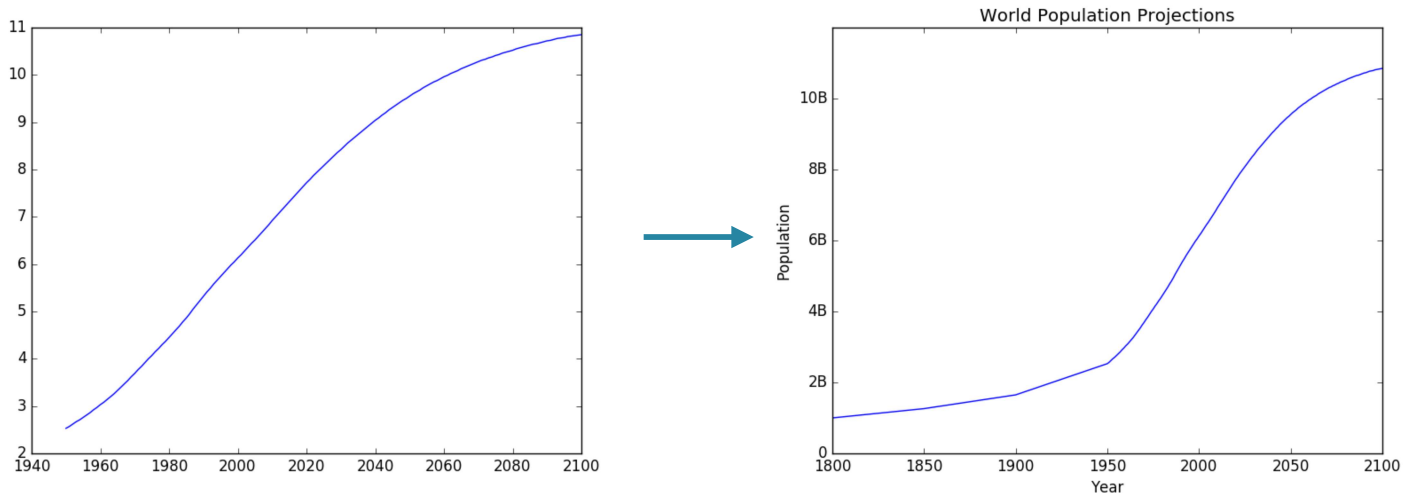
Add historical data

population.py

```
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()
```



Before vs After



INTERMEDIATE PYTHON FOR DATA SCIENCE

Let's practice!

INTERMEDIATE PYTHON FOR DATA SCIENCE

Dictionaries, Part 1

List

```
In [1]: pop = [30.55, 2.77, 39.21]

In [2]: countries = ["afghanistan", "albania", "algeria"]

In [3]: ind_alb = countries.index("albania")

In [4]: ind_alb
Out[4]: 1

In [5]: pop[ind_alb]
Out[5]: 2.77
```

Not convenient
Not intuitive

Dictionary

```
In [1]: pop = [30.55, 2.77, 39.21]

In [2]: countries = ["afghanistan", "albania", "algeria"]

...

In [6]: world = {"afghanistan":30.55, "albania":2.77, "algeria":39.21}

In [7]: world["albania"]
Out[7]: 2.77
```

dict_name[key]

result: **value**

INTERMEDIATE PYTHON FOR DATA SCIENCE

Let's practice!

INTERMEDIATE PYTHON FOR DATA SCIENCE

Dictionaries, Part 2

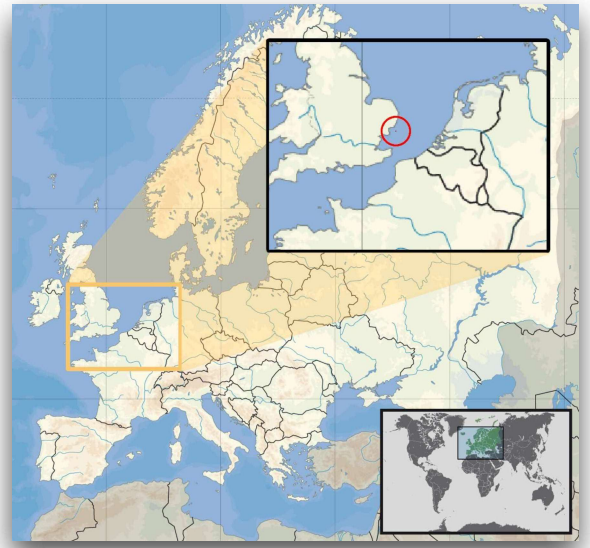
Recap

```
In [1]: world = {"afghanistan":30.55, "albania":2.77, "algeria":39.21}

In [2]: world["albania"]
Out[2]: 2.77

In [3]: world = {"afghanistan":30.55, "albania":2.77,
                 "algeria":39.21, "albania":2.81}

In [4]: world
Out[4]: {'afghanistan': 30.55, 'albania': 2.81, 'algeria': 39.21}
```

keys have to be "immutable" objects

```
In [5]: {0:"hello", True:"dear", "two":"world"}
Out[5]: {0: 'hello', True: 'dear', 'two': 'world'}

In [6]: {["just", "to", "test"]: "value"}
TypeError: unhashable type: 'list'
```

Principality of Sealand



Source: Wikipedia

Dictionary

```
In [8]: world["sealand"] = 0.000027

In [9]: world
Out[9]: {'afghanistan': 30.55, 'albania': 2.81,
                'algeria': 39.21, 'sealand': 2.7e-05}

In [10]: "sealand" in world
Out[10]: True

In [11]: world["sealand"] = 0.000028

In [12]: world
Out[12]: {'afghanistan': 30.55, 'albania': 2.81,
          'algeria': 39.21, 'sealand': 2.8e-05}

In [13]: del(world["sealand"])

In [14]: world
Out[14]: {'afghanistan': 30.55, 'albania': 2.81, 'algeria': 39.21}
```

List vs Dictionary

List	Dictionary
Select, update and remove: [ ]	Select, update and remove: [ ]
Indexed by range of numbers	Indexed by unique keys
Collection of values order matters select entire subsets	Lookup table with unique keys

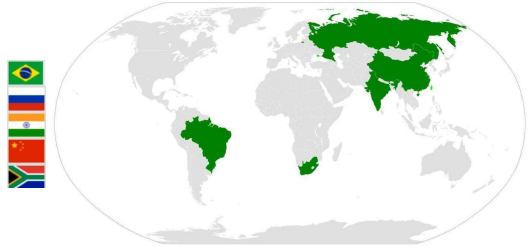
INTERMEDIATE PYTHON FOR DATA SCIENCE

Let's practice!

INTERMEDIATE PYTHON FOR DATA SCIENCE

Pandas, Part 1

Tabular dataset examples



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 = observations
column = variable

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 Africa	Pretoria	1.221	52.98

Datasets in Python

- 2D Numpy array?
  - One data type

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 Africa	Pretoria	1.221	52.98
`str`	`str`	`float`	`float`

- Pandas!
  - High level data manipulation tool
  - Wes McKinney
  - Built on Numpy
  - `DataFrame`

DataFrame

```
In [1]: brics
Out[1]:
         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
```

DataFrame

```
In [1]: brics
Out[1]:
          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
```

observations

DataFrame

```
In [1]: brics
Out[1]:
```

variables

	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

DataFrame

```
In [1]: brics
Out[1]:
```

	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

column labels

row labels

columns with different types

DataFrame from Dictionary

```
In [2]:  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] }
```

keys (column labels) **values (data, column by column)**

```
In [3]: import pandas as pd

In [4]: brics = pd.DataFrame(dict)
```

DataFrame from Dictionary (2)

```
In [5]: brics
Out[5]:
      area    capital       country  population
0    8.516   Brasilia        Brazil      200.40
1   17.100     Moscow        Russia      143.50
2    3.286  New Delhi         India     1252.00
3    9.597    Beijing         China     1357.00
4    1.221   Pretoria  South Africa       52.98

In [6]: brics.index = ["BR", "RU", "IN", "CH", "SA"]
In [7]: brics
Out[7]:
      area    capital       country  population
BR   8.516   Brasilia        Brazil      200.40
RU  17.100     Moscow        Russia      143.50
IN   3.286  New Delhi         India     1252.00
CH   9.597    Beijing         China     1357.00
SA   1.221   Pretoria  South Africa       52.98
```

DataFrame from CSV file

brics.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
```

CSV = comma-separated values

DataFrame from CSV file

brics.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
```

```
In [8]: brics = pd.read_csv("path/to/brics.csv")
In [9]: brics
Out[9]:
  Unnamed: 0       country    capital    area  population
0         BR        Brazil   Brasilia   8.516      200.40
1         RU        Russia     Moscow  17.100      143.50
2         IN         India  New Delhi   3.286     1252.00
3         CH         China    Beijing   9.597     1357.00
4         SA  South Africa   Pretoria   1.221       52.98

In [6]: brics = pd.read_csv("path/to/brics.csv", index_col = 0)
In [7]: brics
Out[7]:
        country  population      area    capital
BR       Brazil         200   8515767   Brasilia
RU       Russia         144  17098242     Moscow
IN        India        1252   3287590  New Delhi
CH        China        1357   9596961    Beijing
SA  South Africa          55   1221037   Pretoria
```

INTERMEDIATE PYTHON FOR DATA SCIENCE

Let's practice!

INTERMEDIATE PYTHON FOR DATA SCIENCE

Pandas, Part 2

brics

```
In [1]: import pandas as pd

In [2]: brics = pd.read_csv("path/to/brics.csv", index_col = 0)

In [3]: brics
Out[3]:
         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
```

Index and Select Data

- Square brackets
- Advanced methods
  - `loc`
  - `iloc`

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

```
In [4]: brics["country"]
Out[4]:
BR           Brazil
RU           Russia
IN            India
CH            China
SA     South Africa
Name: country, dtype: object

In [5]: type(brics["country"])
Out[5]: pandas.core.series.Series
```

1D labelled array

Column Access [ ]

```
In [6]: brics[["country"]]
Out[6]:
         country
BR        Brazil
RU        Russia
IN         India
CH         China
SA  South Africa

In [7]: type(brics[["country"]])
Out[7]: pandas.core.frame.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

Column Access [ ]

```
In [8]: brics[["country", "capital"]]
Out[8]:
         country    capital
BR        Brazil   Brasilia
RU        Russia     Moscow
IN         India  New Delhi
CH         China    Beijing
SA  South Africa   Pretoria
```

	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

Row Access [ ]

```
In [9]: brics[1:4]
Out[9]:
   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
```

indexes

	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

0
1
2
3
4

Discussion `[ ]`

- Square brackets: limited functionality
- Ideally
  - 2D Numpy arrays
  - `my_array[ rows , columns ]`
- Pandas
  - `loc` (label-based)
  - `iloc` (integer position-based)

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

```
In [10]: brics.loc["RU"]
Out[10]:
country       Russia
capital       Moscow
area            17.1
population     143.5
Name: RU, dtype: object
```

Row as Pandas Series

```
In [11]: brics.loc[["RU"]]
Out[11]:
   country capital  area  population
RU  Russia  Moscow  17.1       143.5
```

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

```
In [10]: brics.loc["RU"]
Out[10]:
country       Russia
capital       Moscow
area            17.1
population     143.5
Name: RU, dtype: object

In [11]: brics.loc[["RU"]]
Out[11]:
   country capital  area  population
RU  Russia  Moscow  17.1       143.5

In [12]: brics.loc[["RU", "IN", "CH"]]
Out[12]:
   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
```

Row & Column `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

```
In [13]: brics.loc[["RU", "IN", "CH"], ["country", "capital"]]
Out[13]:
   country    capital
RU  Russia     Moscow
IN   India  New Delhi
CH   China    Beijing
```

Row & Column `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

```
In [13]: brics.loc[["RU", "IN", "CH"], ["country", "capital"]]
Out[13]:
   country    capital
RU  Russia     Moscow
IN   India  New Delhi
CH   China    Beijing

In [14]: brics.loc[:, ["country", "capital"]]
Out[14]:
         country    capital
BR        Brazil   Brasilia
RU        Russia     Moscow
IN         India  New Delhi
CH         China    Beijing
SA  South Africa   Pretoria
```

Recap

- Square brackets
  - Column access `brics[["country", "capital"]]`
  - Row access: only through slicing `brics[1:4]`
- `loc` (label-based)
  - Row access `brics.loc[["RU", "IN", "CH"]]`
  - Column access `brics.loc[:, ["country", "capital"]]`
  - Row & Column access `brics.loc[["RU", "IN", "CH"], ["country", "capital"]]`

Row Access iloc

```
In [15]: brics.loc[["RU"]]
Out[15]:
   country capital  area  population
RU  Russia  Moscow  17.1       143.5

In [16]: brics.iloc[[1]]
Out[16]:
   country capital  area  population
RU  Russia  Moscow  17.1       143.5
```

	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

Row Access iloc

```
In [17]: brics.loc[["RU", "IN", "CH"]]
Out[17]:
   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

In [18]: brics.iloc[[1,2,3]]
Out[18]:
   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
```

	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

Row & Column iloc

	0	1	2	3
	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

```
In [19]: brics.loc[["RU", "IN", "CH"], ["country", "capital"]]
Out[19]:
   country    capital
RU  Russia     Moscow
IN   India  New Delhi
CH   China    Beijing

In [20]: brics.iloc[[1,2,3], [0, 1]]
Out[20]:
   country    capital
RU  Russia     Moscow
IN   India  New Delhi
CH   China    Beijing
```

Row & Column iloc

	0	1	2	3
	country	capital	area	population
0	BR	Brazil	Brasilia	8.516	200.40
1	RU	Russia	Moscow	17.100	143.50
2	IN	India	New Delhi	3.286	1252.00
3	CH	China	Beijing	9.597	1357.00
4	SA	South Africa	Pretoria	1.221	52.98

```
In [21]: brics.loc[:, ["country", "capital"]]
Out[21]:
         country    capital
BR        Brazil   Brasilia
RU        Russia     Moscow
IN         India  New Delhi
CH         China    Beijing
SA  South Africa   Pretoria

In [22]: brics.iloc[:, [0,1]]
Out[22]:
         country    capital
BR        Brazil   Brasilia
RU        Russia     Moscow
IN         India  New Delhi
CH         China    Beijing
SA  South Africa   Pretoria
```

INTERMEDIATE PYTHON FOR DATA SCIENCE

Let's practice!

INTERMEDIATE PYTHON FOR DATA SCIENCE

Comparison Operators

Numpy Recap

```
In [1]: import numpy as np
In [2]: np_height = np.array([1.73, 1.68, 1.71, 1.89, 1.79])
In [3]: np_weight = np.array([65.4, 59.2, 63.6, 88.4, 68.7])
In [4]: bmi = np_weight / np_height ** 2

In [5]: bmi
Out[5]: array([ 21.852,  20.975,  21.75 ,  24.747,  21.441])

In [6]: bmi > 23
Out[6]: array([False, False, False,  True, False], dtype=bool)

In [7]: bmi[bmi > 23]
Out[7]: array([ 24.747])
```

Code from Intro to Python for Data Science, Chapter 4

Comparison operators: how Python values relate

Numeric Comparisons

```
In [8]: 2 < 3
Out[8]: True

In [9]: 2 == 3
Out[9]: False

In [10]: 2 <= 3
Out[10]: True

In [11]: 3 <= 3
Out[11]: True

In [12]: x = 2

In [13]: y = 3

In [14]: x < y
Out[14]: True
```

Other Comparisons

```
In [15]: "carl" < "chris"
Out[15]: True

In [16]: 3 < "chris"
TypeError: unorderable types: int() < str()

In [17]: 3 < 4.1
Out[17]: True

In [18]: bmi
Out[18]: array([ 21.852,  20.975,  21.75 ,  24.747,  21.441])

In [19]: bmi > 23
Out[19]: array([False, False, False,  True, False], dtype=bool)
```

Comparators

<	strictly less than
<=	less than or equal
>	strictly greater than
>=	greater than or equal
==	equal
! =	not equal

INTERMEDIATE PYTHON FOR DATA SCIENCE

Let's practice!

INTERMEDIATE PYTHON FOR DATA SCIENCE

Boolean Operators

Boolean Operators

- `and`
- `or`
- `not`

and

```
In [1]: True and True
Out[1]: True

In [2]: False and True
Out[2]: False

In [3]: True and False
Out[3]: False

In [4]: False and False
Out[4]: False

In [5]: x = 12
          True        True
In [6]: x > 5 and x < 15
Out[6]: True
```

or

```
In [7]: True or True
Out[7]: True

In [8]: False or True
Out[8]: True

In [9]: True or False
Out[9]: True

In [10]: False or False
Out[10]: False

In [11]: y = 5
           True       False
In [12]: y < 7 or y > 13
Out[12]: True
```

not

```
In [13]: not True
Out[13]: False

In [14]: not False
Out[14]: True
```

Numpy

```
In [19]: bmi     # calculation of bmi left out
Out[19]: array([ 21.852,  20.975,  21.75 ,  24.747,  21.441])

In [20]: bmi > 21
Out[20]: array([ True, False,  True,  True,  True], dtype=bool)

In [21]: bmi < 22
Out[22]: array([ True,  True,  True, False,  True], dtype=bool)

In [23]: 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()
```

Numpy

```
In [19]: bmi     # calculation of bmi left out
Out[19]: array([ 21.852,  20.975,  21.75 ,  24.747,  21.441])

In [20]: bmi > 21
Out[20]: array([ True, False,  True,  True,  True], dtype=bool)

In [21]: bmi < 22
Out[22]: array([ True,  True,  True, False,  True], dtype=bool)

In [23]: 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()

In [24]: np.logical_and(bmi > 21, bmi < 22)
Out[24]: array([ True, False,  True, False,  True], dtype=bool)

In [25]: bmi[np.logical_and(bmi > 21, bmi < 22)]
Out[25]: array([ 21.852,  21.75,  21.441])
```

logical_and()
logical_or()
logical_not()

INTERMEDIATE PYTHON FOR DATA SCIENCE

Let's practice!

INTERMEDIATE PYTHON FOR DATA SCIENCE

if, elif, else

Overview

- Comparison Operators
  - `<, >, >=, <=, ==, !=`
- Boolean Operators
  - `and, or, not`
- Conditional Statements
  - `if, else, elif`

if

```
if condition :
    expression
```

control.py

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

True

Output:

```
z is even
```

if

```
if condition :
    expression
expression # not part of if
```

control.py

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

True

Output:

```
z is even
```

if

```
if condition :
    expression
```

control.py

```
z = 4
if z % 2 == 0 :
    print("checking "  + str(z))
    print("z is even")
```

Output:

```
checking 4
z is even
```

if

```
if condition :
    expression
```

control.py

```
z = 5
if z % 2 == 0 :
    print("checking "  + str(z))
    print("z is even")
```

False

Not executed

```
Output:
```

else

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

control.py

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

False

Output:

```
z is odd
```

elif

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

control.py

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

False

True

Output:

```
z is divisible by 3
```

elif

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

control.py

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

```
Output:
z is divisible by 2
```

INTERMEDIATE PYTHON FOR DATA SCIENCE

Let's practice!

INTERMEDIATE PYTHON FOR DATA SCIENCE

Filtering Pandas DataFrame

brics

```
In [1]: import pandas as pd

In [2]: brics = pd.read_csv("path/to/brics.csv", index_col = 0)

In [3]: brics
Out[3]:
         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
```

Goal

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

```
        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
```

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

```
In [4]: brics["area"]
Out[4]:
BR     8.516
RU    17.100
IN     3.286
CH     9.597
SA     1.221
Name: area, dtype: float64
```

Alternatives:

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

brics.iloc[:,2]
```

Need Pandas Series

Step 2: Compare

```
          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
```

```
In [4]: brics["area"]
Out[4]:
BR     8.516
RU    17.100
IN     3.286
CH     9.597
SA     1.221
Name: area, dtype: float64

In [5]: brics["area"] > 8
Out[5]:
BR     True
RU     True
IN    False
CH     True
SA    False
Name: area, dtype: bool

In [6]: is_huge = brics["area"] > 8
```

Step 3: Subset DF

	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

```
In [7]: is_huge
Out[7]:
BR     True
RU     True
IN    False
CH     True
SA    False
Name: area, dtype: bool

In [8]: brics[is_huge]
Out[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
```

Summary

```
In [9]: is_huge = brics["area"] > 8

In [10]: brics[is_huge]
Out[10]:
   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

In [11]: brics[brics["area"] > 8]
Out[11]:
   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
```

	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

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

```
In [12]: import numpy as np

In [13]: np.logical_and(brics["area"] > 8, brics["area"] < 10)
Out[13]:
BR     True
RU    False
IN    False
CH     True
SA    False
Name: area, dtype: bool

In [14]: brics[np.logical_and(brics["area"] > 8, brics["area"] < 10)]
Out[14]:
   country   capital   area  population
BR  Brazil  Brasilia  8.516       200.4
CH   China   Beijing  9.597      1357.0
```

INTERMEDIATE PYTHON FOR DATA SCIENCE

Let's practice!

INTERMEDIATE PYTHON FOR DATA SCIENCE

while loop

if-elif-else

control.py

Goes through construct only once!

```
z = 6
if z % 2 == 0 :
    print("z is divisible by 2")
elif z % 3 == 0 :
    print("z is divisible by 3")
else :
    print("z is neither divisible by 2 nor by 3")

...
```

True

Executed

Moving on

While loop = repeated if statement

While

```
while condition :
    expression
```

- Numerically calculating model
- "repeating action until condition is met”
- Example
  - Error starts at 50
  - Divide error by 4 on every run
  - Continue until error no longer > 1

While

Error starts at 50
Divide error by 4 on every run
Continue until error no longer > 1

```
while condition :
    expression
```

while_loop.py

```
error = 50.0

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

While

```
while condition :
    expression
```

while_loop.py

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

```
Output:
12.5
```

While

```
while condition :
    expression
```

while_loop.py

```
error = 50.0

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

12.5

True

Output:

```
12.5
3.125
```

While

```
while condition :
    expression
```

while_loop.py

```
error = 50.0

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

3.125

True

Output:

```
12.5
3.125
0.78125
```

While

```
while condition :
    expression
```

while_loop.py

```
error = 50.0

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

0.78125

False

not executed

```
Output:
12.5
3.125
0.78125
```

While

```
while condition :
    expression
```

while_loop.py

```
error = 50.0

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

always True

```
Output:
50
50
50
50
50
50
50
...
```

DataCamp: session disconnected
Local system: Control + C

INTERMEDIATE PYTHON FOR DATA SCIENCE

Let's practice!

INTERMEDIATE PYTHON FOR DATA SCIENCE

for loop

for loop

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

*"for each **var** in **seq**, execute expression"*

fam

family.py

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

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

fam

family.py

```
fam = [1.73, 1.68, 1.71, 1.89]
print(fam[0])
print(fam[1])
print(fam[2])
print(fam[3])
```

```
Output:
1.73
1.68
1.71
1.89
```

for loop

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

family.py

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

for height in fam :
    print(height)
```

for loop

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

family.py

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

for height in fam :
    print(height)
```

first iteration
height = 1.73

Output:

```
1.73
```

for loop

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

family.py

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

for height in fam :
    print(height)
```

second iteration
height = 1.68

```
Output:
1.73
1.68
```

for loop

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

family.py

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

for height in fam :
    print(height)
```

no access to indexes

```
Output:
1.73
1.68
1.71
1.89
```

for loop

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

family.py

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

???
```

Output:

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

enumerate

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

family.py

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

for index, height in enumerate(fam) :
    print("index " + str(index) + ": " + str(height))
```

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

Loop over string

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

strloop.py

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

```
Output:
F
A
M
I
L
Y
```

INTERMEDIATE PYTHON FOR DATA SCIENCE

Let's practice!

INTERMEDIATE PYTHON FOR DATA SCIENCE

Loop Data Structures Part 1

Dictionary

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

dictloop.py

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

for key, value in world :
    print(key + " -- " + str(value))
```

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

Dictionary

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

dictloop.py

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

for key, value in world.items() :
    print(key + " -- " + str(value))
```

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

Dictionary

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

dictloop.py

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

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

Numpy Arrays

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

nploop.py

```
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)
```

```
Output:
21.852
20.975
21.750
24.747
21.441
```

2D Numpy Arrays

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

np2dloop.py

```
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)
```

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

2D Numpy Arrays

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

np2dloop.py

```
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)
```

Output:

```
1.73
1.68
1.71
1.89
1.79
```

Output (cont):

```
65.4
59.2
63.6
88.4
68.7
```

Recap

- Dictionary
  - `for key, val in my_dict.items() :`
- Numpy array
  - `for val in np.nditer(my_array) :`

INTERMEDIATE PYTHON FOR DATA SCIENCE

Let's practice!

INTERMEDIATE PYTHON FOR DATA SCIENCE

Loop Data Structures Part 2

brics

dfloop.py

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

	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

for, first try

	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

dfloop.py

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

for val in brics :
    print(val)
```

```
Output:
country
capital
area
population
```

iterrows

	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

dfloop.py

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

for lab, row in brics.iterrows() :
    print(lab)
    print(row)
```

Output:

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

Output (cont):

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

Selective print

	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

dfloop.py

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

for lab, row in brics.iterrows() :
    print(lab + ": " + row["capital"])
```

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

Add 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

dfloop.py

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

for lab, row in brics.iterrows() :
    brics.loc[lab, "name_length"] = len(row["country"])
print(brics)
```

Creating Series on every iteration

Output:

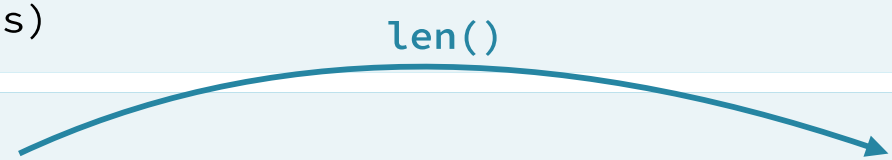
	country	capital	area	population	name_length
BR	Brazil	Brasilia	8.516	200.40	6
RU	Russia	Moscow	17.100	143.50	6
IN	India	New Delhi	3.286	1252.00	5
CH	China	Beijing	9.597	1357.00	5
SA	South Africa	Pretoria	1.221	52.98	12

apply

	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

dfloop.py

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

brics["name_length"] = brics["country"].apply(len)

print(brics)
```



Output:

	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

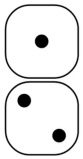
INTERMEDIATE PYTHON FOR DATA SCIENCE

Let's practice!

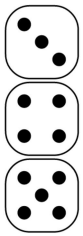
INTERMEDIATE PYTHON FOR DATA SCIENCE

Random Numbers

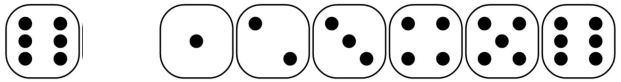
100 x



-1



+1



+1 +2 +3 +4 +5 +6

Can't go below step 0

0.1 % chance of falling down the stairs

Bet: you'll reach step 60

How to solve?

- Analytical
- Simulate the process
  - Hacker statistics!

Random Generators

```
In [1]: import numpy as np

In [2]: np.random.rand()
Out[2]: 0.9535543896720104

In [3]: np.random.seed(123)

In [4]: np.random.rand()
Out[4]: 0.6964691855978616
In [5]: np.random.rand()
Out[5]: 0.28613933495037946

In [6]: np.random.seed(123)

In [7]: np.random.rand()
Out[7]: 0.696469185597861
In [8]: np.random.rand()
Out[8]: 0.28613933495037946
```

Pseudo-random numbers
Mathematical formula
Starting from a seed

Same seed: same random numbers!
Ensures "reproducibility"

Coin Toss

game.py

```
import numpy as np
np.random.seed(123)
coin = np.random.randint(0,2)
print(coin)
```

Randomly generate 0 or 1

```
Output:
0
```

Coin Toss

game.py

```
import numpy as np
np.random.seed(123)
coin = np.random.randint(0,2)
print(coin)
if coin == 0:
    print("heads")
else:
    print("tails")
```

```
Output:
0
heads
```

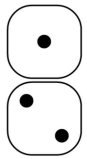
INTERMEDIATE PYTHON FOR DATA SCIENCE

Let's practice!

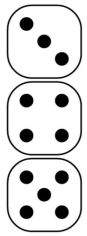
INTERMEDIATE PYTHON FOR DATA SCIENCE

Random Walk

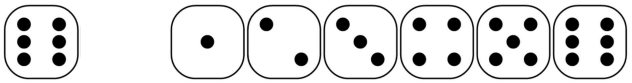
Random Step



-1



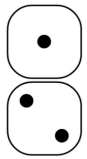
+1



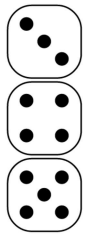
+1 +2 +3 +4 +5 +6

Random Walk

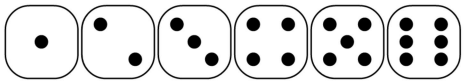
100 x



-1



+1



+1 +2 +3 +4 +5 +6

Known in Science

- Path of molecules
- Gambler's financial status

Heads or Tails

headtails.py

```
import numpy as np
np.random.seed(123)
outcomes = []
for x in range(10) :
    coin = np.random.randint(0, 2)
    if coin == 0 :
        outcomes.append("heads")
    else :
        outcomes.append("tails")
print(outcomes)
```

0 : heads
1 : tails

```
Output:
['heads', 'tails', 'heads', 'heads', 'heads',
'heads', 'heads', 'tails', 'tails', 'heads']
```

Heads or Tails: Random Walk

headtailsrw.py

```
import numpy as np
np.random.seed(123)
tails = [0]
for x in range(10) :
    coin = np.random.randint(0, 2)
    tails.append(tails[x] + coin)

print(tails)
```

```
Output:
[0, 0, 1, 1, 1, 1, 1, 1, 2, 3, 3]
```

Step to Walk

outcomes

```
['heads', 'tails', 'heads', 'heads', 'heads',
'heads', 'heads', 'tails', 'tails', 'heads']
```

tails

```
Output:
[0, 0, 1, 1, 1, 1, 1, 1, 2, 3, 3]
```

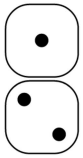
INTERMEDIATE PYTHON FOR DATA SCIENCE

Let's practice!

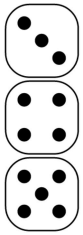
INTERMEDIATE PYTHON FOR DATA SCIENCE

Distribution

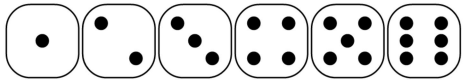
100 x



-1



+1



+1 +2 +3 +4 +5 +6

Each random walk has an end point

Simulate 10,000 times: 10,000 end points

Distribution!

Calculate chances!

Random Walk

headtailsrw.py

```
import numpy as np
np.random.seed(123)
tails = [0]
for x in range(10) :
    coin = np.random.randint(0,2)
    tails.append(tails[x] + coin)
```

100 runs

distribution.py

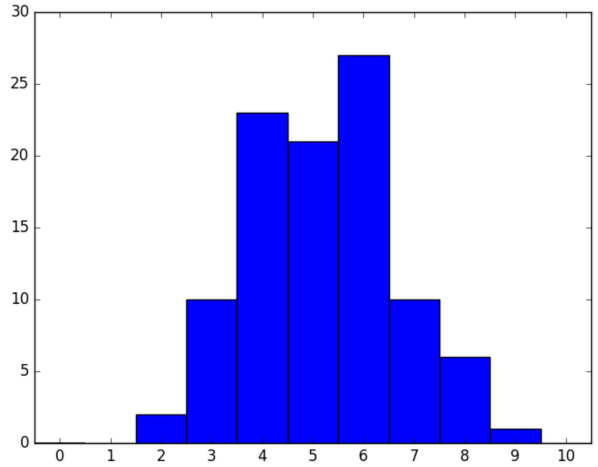
```
import numpy as np
np.random.seed(123)
final_tails = []
for x in range(100) :
    tails = [0]
    for x in range(10) :
        coin = np.random.randint(0,2)
        tails.append(tails[x] + coin)
    final_tails.append(tails[-1])
print(final_tails)
```

```
Output:
[3, 6, 4, 5, 4, 5, 3, 5, 4, 6, 6, 8, 6, 4, 7, 5, 7,
4, 3, 3, 4, 5, 8, 5, 6, 5, 7, 6, 4, 5, 8, 5, 8, 4,
6, 6, 3, 4, 5, 4, 7, 8, 9, 4, 3, 4, 5, 6, 4, 2, 6,
6, 5, 7, 5, 4, 5, 5, 6, 7, 6, 6, 6, 3, ..., 7]
```

Histogram, 100 runs

distribution.py

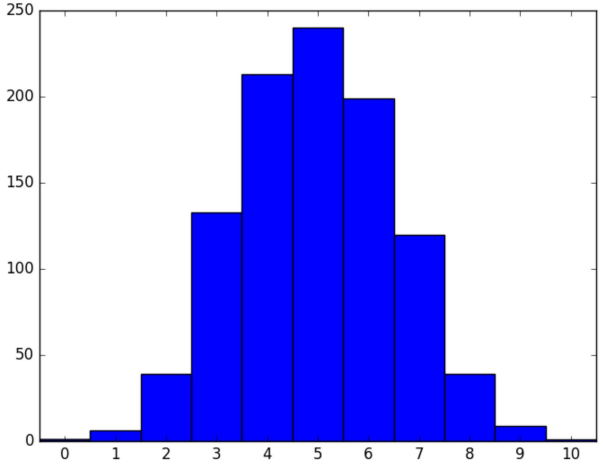
```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(123)
final_tails = []
for x in range(100) :
    tails = [0]
    for x in range(10) :
        coin = np.random.randint(0,2)
        tails.append(tails[x] + coin)
    final_tails.append(tails[-1])
plt.hist(final_tails, bins = 10)
plt.show()
```



Histogram, 1.000 runs

distribution.py

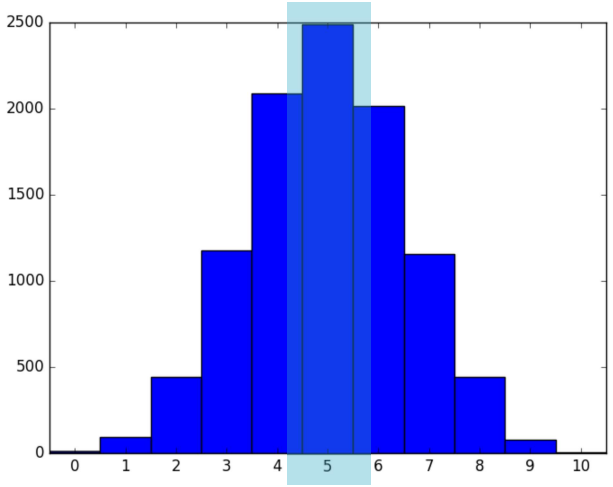
```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(123)
final_tails = []
for x in range(1000) :
    tails = [0]
    for x in range(10) :
        coin = np.random.randint(0,2)
        tails.append(tails[x] + coin)
    final_tails.append(tails[-1])
plt.hist(final_tails, bins = 10)
plt.show()
```



Histogram, 10.000 runs

distribution.py

```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(123)
final_tails = []
for x in range(10000) :
    tails = [0]
    for x in range(10) :
        coin = np.random.randint(0,2)
        tails.append(tails[x] + coin)
    final_tails.append(tails[-1])
plt.hist(final_tails, bins = 10)
plt.show()
```



INTERMEDIATE PYTHON FOR DATA SCIENCE

Let's practice!