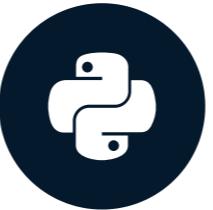


# Hello Python!

## INTRODUCTION TO PYTHON



**Hugo Bowne-Anderson**  
Data Scientist at DataCamp

# How you will learn

The screenshot shows a DataCamp exercise interface for "Calculations with variables".

**Instructions:**

- Create a variable `growth_multiplier`, equal to `1.1`.
- Create a variable `result`, equal to the amount of money you saved after `7` years.
- Print out the value of `result`.

**Code Snippet:**

```
savings = 100
growth_multiplier = 1.1
result = savings *
# Print out result
```

**Python Shell:**

```
In [1]:
```

**Buttons:**

- Course Outline
- Daily XP 0
- Run Code
- Submit Answer

# Python



- General purpose: build anything
- Open source! Free!
- Python packages, also for data science
  - Many applications and fields
- Version 3.x - <https://www.python.org/downloads/>

# IPython Shell

## Execute Python commands

The screenshot shows a DataCamp exercise interface for "Calculations with variables".

**Exercise Overview:** Daily XP 100

**Instructions (100 XP):**

- Create a variable `growth_multiplier`, equal to `1.1`.
- Create a variable, `result`, equal to the amount of money you saved after `7` years.
- Print out the value of `result`.

**Code Editor:** script.py

```
1 # Create a variable savings
2 savings = 100
3
4 # Create a variable growth_multiplier
5 growth_multiplier = 1.1
6
7 # Calculate result
8 result = savings * growth_multiplier ** 7
9
10 # Print out result
11 print(result)
12
13
```

**Run Code** button

**IPython Shell:** In [1]:

# IPython Shell

## Execute Python commands

The screenshot shows a DataCamp exercise interface for "Calculations with variables". The main area displays a script named "script.py" with the following code:

```
script.py
1 # Create a variable savings
2 savings = 100
3
4 # Create a variable growth_multiplier
5 growth_multiplier = 1.1
6
7 # Calculate result
8 result = savings * growth_multiplier ** 7
9
10 # Print out result
11 print(result)
12
13
```

To the left, there's an "Instructions" section with a "100 XP" badge, containing the following tasks:

- Create a variable `growth_multiplier`, equal to `1.1`.
- Create a variable, `result`, equal to the amount of money you saved after `7` years.
- Print out the value of `result`.

At the bottom, there are "Run Code" and "Submit Answer" buttons, along with an "IPython Shell" tab which shows the input "In [1]:".

# IPython Shell

The screenshot shows a Python exercise interface from DataCamp. On the left, there's an 'Exercise' sidebar with a title 'Calculations with variables'. It contains instructions about calculating money after 7 years of investing \$100, followed by a code snippet: `100 * 1.1 ** 7`. Below this is a list of tasks and a 'Take Hint (-30 XP)' button. The main area has a 'script.py' tab open, showing the following code:

```
script.py
1 # Create a variable savings
2 savings = 100
3
4 # Create a variable growth_multiplier
5
6
7 # Calculate result
8
9
10 # Print out result
11
```

At the bottom of this tab are 'Run Code' and 'Submit Answer' buttons. Below the code editor is a 'Python Shell' window with the prompt 'In [1]: |'. The status bar at the bottom shows '100 XP'.

# Python Script

- Text files - `.py`
- List of Python commands
- Similar to typing in IPython Shell

The screenshot shows a DataCamp exercise interface for a Python script named `script.py`. The code in the editor is:

```
1 # Create a variable savings
2 savings = 100
3
4 # Create a variable growth_multiplier
5 growth_multiplier = 1.1
6
7 # Calculate result
8 result = savings * growth_multiplier ** 7
9
10 # Print out result
11 print(result)
12
13
```

The exercise title is "Calculations with variables". The instructions state: "Remember how you calculated the money you ended up with after 7 years of investing \$100? You did something like this: `100 * 1.1 ** 7`". It then explains: "Instead of calculating with the actual values, you can use variables instead. The `savings` variable you've created in the previous exercise represents the \$100 you started with. It's up to you to create a new variable to represent `1.1` and then redo the calculations!"

The instructions list three tasks:

- Create a variable `growth_multiplier`, equal to `1.1`.
- Create a variable, `result`, equal to the amount of money you saved after 7 years.
- Print out the value of `result`.

Buttons at the bottom include "Run Code" and "Submit Answer". Below the code editor is an IPython Shell tab showing "In [1]:".

# Python Script

The screenshot shows a Python script exercise interface. At the top, there's a navigation bar with icons for back, forward, course outline, and daily XP (100). Below the navigation is a sidebar titled "Exercise" with the section "Calculations with variables". The main content area has a heading "Remember how you calculated the money you ended up with after 7 years of investing \$100? You did something like this:" followed by a code snippet: `100 * 1.1 ** 7`. A note below explains: "Instead of calculating with the actual values, you can use variables instead. The `savings` variable you've created in the previous exercise represents the \$100 you started with. It's up to you to create a new variable to represent `1.1` and then redo the calculation!" On the left, under "Instructions", there are three bullet points: "Create a variable `growth_multiplier`, equal to `1.1`", "Create a variable, `result`, equal to the amount of money you saved after 7 years.", and "Print out the value of `result`". A "Take Hint (-30 XP)" button is also present. On the right, there's a "script.py" code editor window showing the code `1`, and a "Run Code" button. Below the code editor is a "Python Shell" window with the prompt `In [1]:`.

# Python Script

The screenshot shows a DataCamp Python script exercise interface. At the top, there's a navigation bar with icons for back, forward, course outline, and other course stats like Daily XP 100. Below the navigation is a sidebar titled "Exercises" with a section for "Calculations with variables". The main area contains a code editor window titled "scr1.py" with the following code:

```
1
```

Below the code editor, there's a text block explaining how to calculate compound interest using variables instead of hard-coded values. It mentions a previous exercise where a variable `savings` was created to represent the initial amount of \$100.

The exercise instructions are listed as follows:

- Create a variable `growth_multiplier`, equal to `1.1`.
- Create a variable `result`, equal to the amount of money you saved after 7 years.
- Print out the value of `result`.

At the bottom of the interface, there are "Run Code" and "Submit Answer" buttons, along with a Python Shell tab showing the command `In [1]:`.

- Use `print()` to generate output from script

# DataCamp Interface

The screenshot shows the DataCamp Python exercise interface. On the left, there's an 'Exercise' sidebar with a title 'Calculations with variables'. It contains a code snippet: `100 * 1.1 ** 7`. Below it, instructions mention using variables instead of hard-coded values. A list of tasks includes creating a variable `growth_multiplier`, calculating a result, and printing it. A 'Take Hint (-30 XP)' button is present. On the right, the main workspace shows a code editor with `script.py` containing the following code:

```
1 # Create a variable savings
2 savings = 100
3
4 # Create a variable growth_multiplier
5
6
7 # Calculate result
8
9
10 # Print out result
11
```

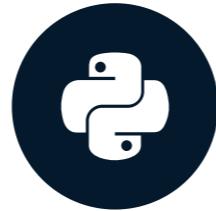
The code editor has tabs for 'Course Outline', 'Daily XP 100', and other icons. Below the code editor is an 'IPython Shell' tab with the prompt 'In [1]:'. At the bottom are buttons for 'Run Code' and 'Submit Answer'.

# **Let's practice!**

**INTRODUCTION TO PYTHON**

# Variables and Types

INTRODUCTION TO PYTHON



**Hugo Bowne-Anderson**  
Data Scientist at DataCamp

# Variable

- Specific, case-sensitive name
- Call up value through variable name
- 1.79 m - 68.7 kg

```
height = 1.79
```

```
weight = 68.7
```

```
height
```

```
1.79
```

# Calculate BMI

```
height = 1.79
```

```
weight = 68.7
```

```
height
```

```
1.79
```

$$\text{BMI} = \frac{\text{weight}}{\text{height}^2}$$

```
68.7 / 1.79 ** 2
```

```
21.4413
```

```
weight / height ** 2
```

```
21.4413
```

```
bmi = weight / height ** 2
```

```
bmi
```

```
21.4413
```

# Reproducibility

```
height = 1.79  
weight = 68.7  
bmi = weight / height ** 2  
print(bmi)
```

```
21.4413
```

# Reproducibility

```
height = 1.79  
weight = 74.2 # <-  
bmi = weight / height ** 2  
print(bmi)
```

23.1578

# Python Types

```
type(bmi)
```

```
float
```

```
day_of_week = 5  
type(day_of_week)
```

```
int
```

# Python Types (2)

```
x = "body mass index"  
y = 'this works too'  
type(y)
```

```
str
```

```
z = True  
type(z)
```

```
bool
```

# Python Types (3)

```
2 + 3
```

```
5
```

```
'ab' + 'cd'
```

```
'abcd'
```

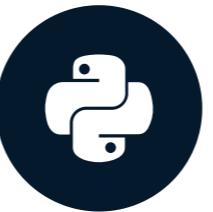
- Different type = different behavior!

# **Let's practice!**

**INTRODUCTION TO PYTHON**

# Python Lists

INTRODUCTION TO PYTHON



**Hugo Bowne-Anderson**  
Data Scientist at DataCamp

# Python Data Types

- float - real numbers
- int - integer numbers
- str - string, text
- bool - True, False

```
height = 1.73  
tall = True
```

- Each variable represents single value

# Problem

- Data Science: many data points
- Height of entire family

```
height1 = 1.73  
height2 = 1.68  
height3 = 1.71  
height4 = 1.89
```

- Inconvenient

# Python List

- [a, b, c]

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

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

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

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

- Name a collection of values
- Contain any type
- Contain different types

# Python List

- [a, b, c]

```
fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]
```

```
fam
```

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

```
fam2 = [[{"name": "liz", "height": 1.73},  
         {"name": "emma", "height": 1.68},  
         {"name": "mom", "height": 1.71},  
         {"name": "dad", "height": 1.89}]]
```

```
fam2
```

```
[["liz", 1.73], ["emma", 1.68], ["mom", 1.71], ["dad", 1.89]]
```

# List type

```
type(fam)
```

```
list
```

```
type(fam2)
```

```
list
```

- Specific functionality
- Specific behavior

# **Let's practice!**

**INTRODUCTION TO PYTHON**

# Subsetting Lists

INTRODUCTION TO PYTHON



**Hugo Bowne-Anderson**  
Data Scientist at DataCamp

# Subsetting lists

```
fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]  
fam
```

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

```
fam[3]
```

```
1.68
```

# Subsetting lists

```
[liz', 1.73, emma', 1.68, mom', 1.71, dad', 1.89]
```

```
fam[6]
```

```
'dad'
```

```
fam[-1]
```

```
1.89
```

```
fam[7]
```

```
1.89
```

# Subsetting lists

```
[liz', 1.73, emma', 1.68, mom', 1.71, dad', 1.89]
```

```
fam[6]
```

```
'dad'
```

```
fam[-1] # <-
```

```
1.89
```

```
fam[7] # <-
```

```
1.89
```

# List slicing

```
fam
```

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

```
fam[3:5]
```

```
[1.68, 'mom']
```

```
fam[1:4]
```

```
[1.73, 'emma', 1.68]
```

[ start : end ]

inclusive

exclusive

# List slicing

```
fam
```

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

```
fam[:4]
```

```
['liz', 1.73, 'emma', 1.68]
```

```
fam[5:]
```

```
[1.71, 'dad', 1.89]
```

# **Let's practice!**

**INTRODUCTION TO PYTHON**

# Manipulating Lists

INTRODUCTION TO PYTHON



**Hugo Bowne-Anderson**  
Data Scientist at DataCamp

# List Manipulation

- Change list elements
- Add list elements
- Remove list elements

# Changing list elements

```
fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]  
fam
```

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

```
fam[7] = 1.86  
fam
```

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.86]
```

```
fam[0:2] = ["lisa", 1.74]  
fam
```

```
['lisa', 1.74, 'emma', 1.68, 'mom', 1.71, 'dad', 1.86]
```

# Adding and removing elements

```
fam + ["me", 1.79]
```

```
['lisa', 1.74, 'emma', 1.68, 'mom', 1.71, 'dad', 1.86, 'me', 1.79]
```

```
fam_ext = fam + ["me", 1.79]
```

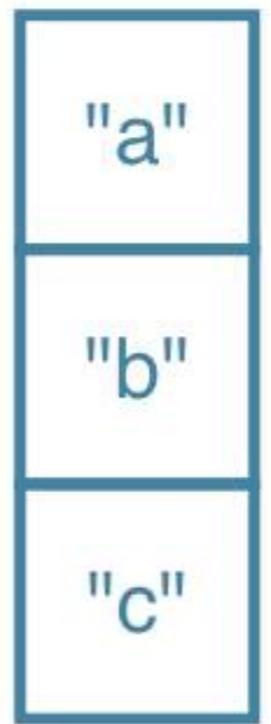
```
del(fam[2])
```

```
fam
```

```
['lisa', 1.74, 1.68, 'mom', 1.71, 'dad', 1.86]
```

# Behind the scenes (1)

```
x = ["a", "b", "c"]
```



# Behind the scenes (1)

```
x = ["a", "b", "c"]
```

```
y = x
```

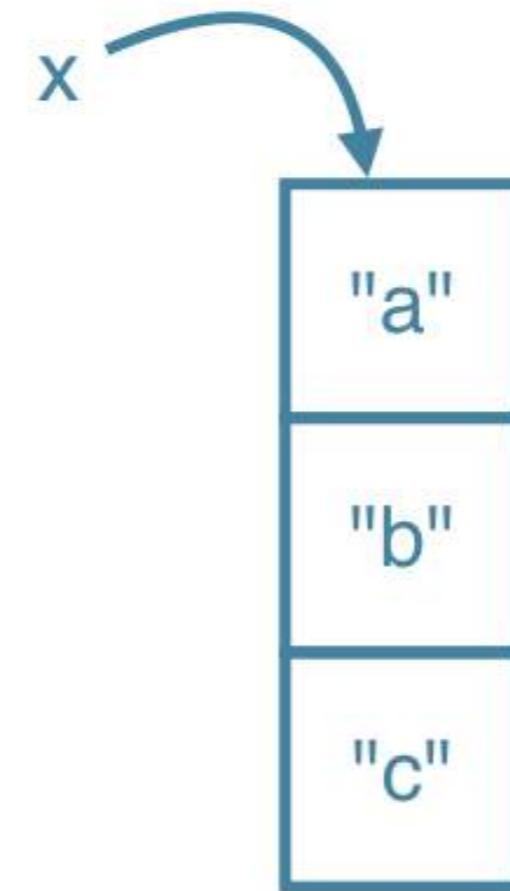
```
y[1] = "z"
```

```
y
```

```
['a', 'z', 'c']
```

```
x
```

```
['a', 'z', 'c']
```



# Behind the scenes (1)

```
x = ["a", "b", "c"]
```

```
y = x
```

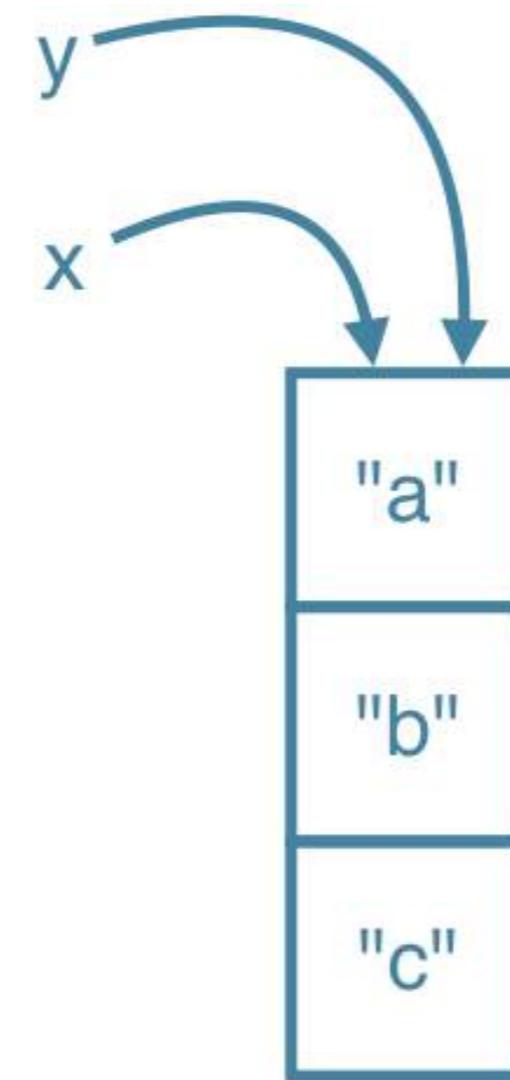
```
y[1] = "z"
```

```
y
```

```
['a', 'z', 'c']
```

```
x
```

```
['a', 'z', 'c']
```



# Behind the scenes (1)

```
x = ["a", "b", "c"]
```

```
y = x
```

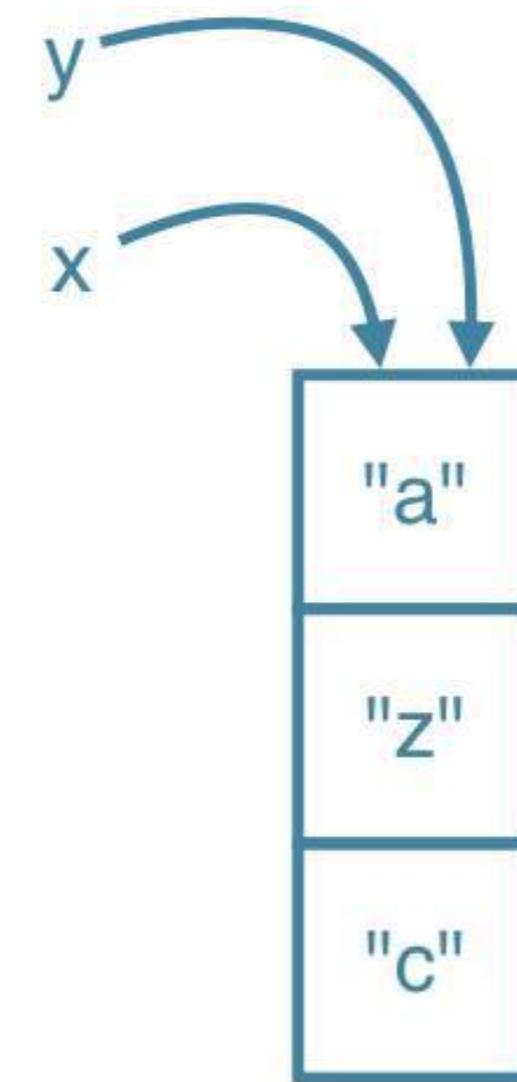
```
y[1] = "z"
```

```
y
```

```
['a', 'z', 'c']
```

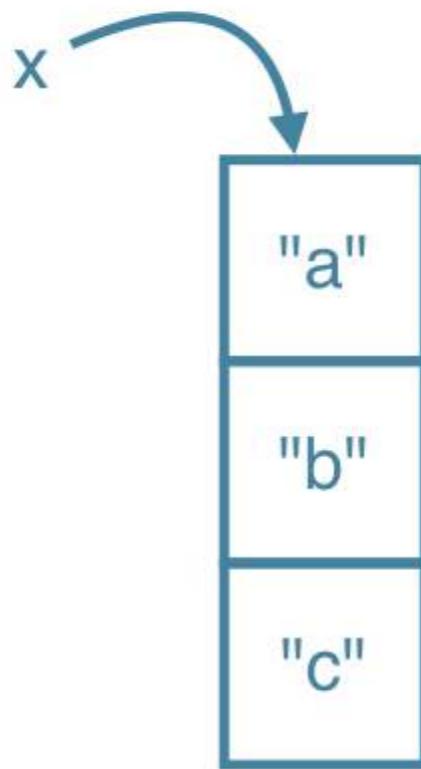
```
x
```

```
['a', 'z', 'c']
```



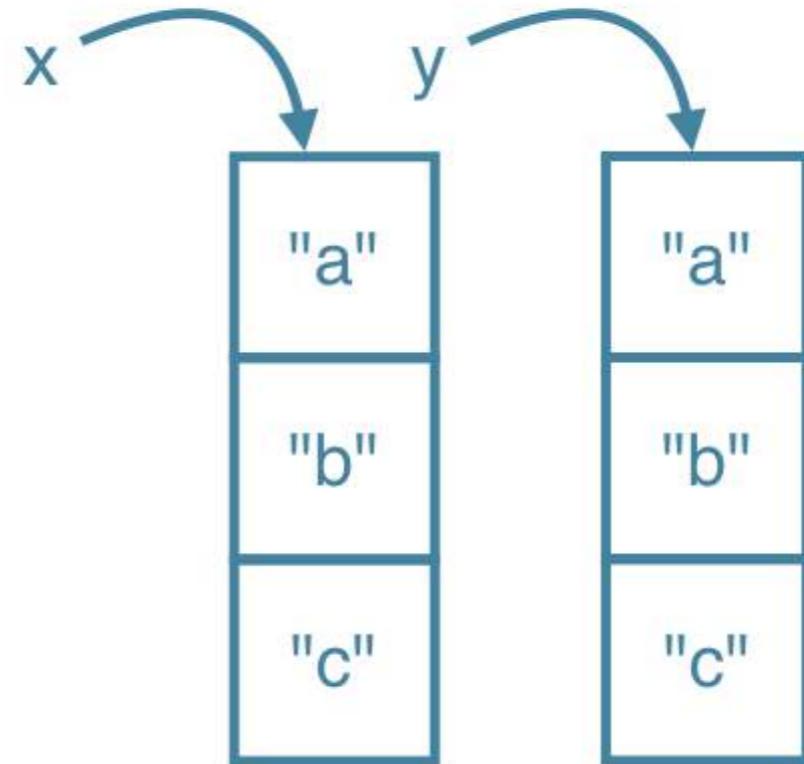
# Behind the scenes (2)

```
x = ["a", "b", "c"]
```



# Behind the scenes (2)

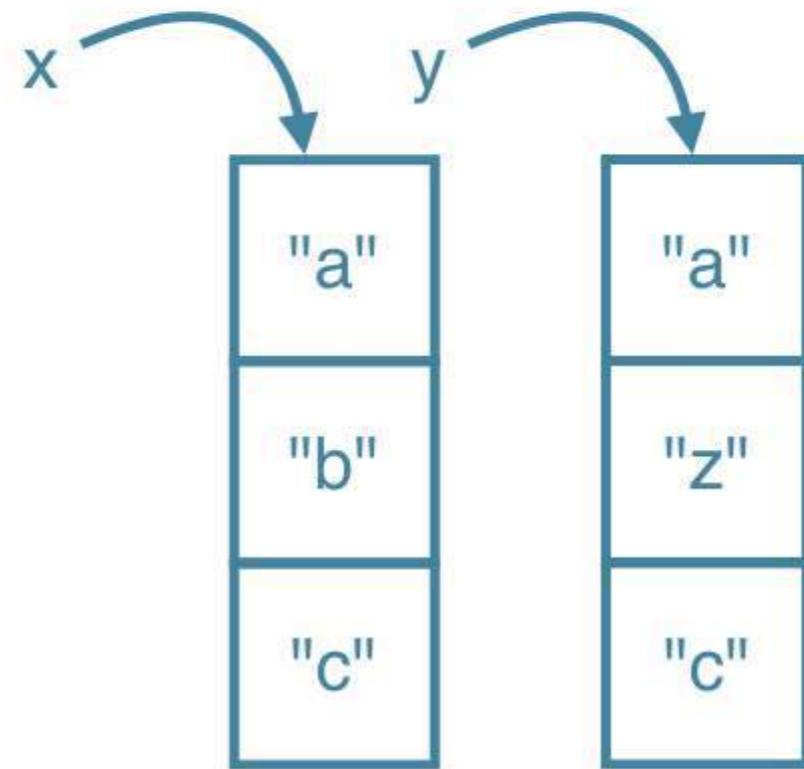
```
x = ["a", "b", "c"]  
y = list(x)  
y = x[:]
```



# Behind the scenes (2)

```
x = ["a", "b", "c"]
y = list(x)
y = x[:]
y[1] = "z"
x
```

```
['a', 'b', 'c']
```



# **Let's practice!**

**INTRODUCTION TO PYTHON**

# Functions

INTRODUCTION TO PYTHON



**Hugo Bowne-Anderson**  
Data Scientist at DataCamp

# Functions

- Nothing new!
- `type()`
- Piece of reusable code
- Solves particular task
- Call function instead of writing code yourself

# Example

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

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

```
max(fam)
```

```
1.89
```

```
max()
```

# Example

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

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

```
max(fam)
```

```
1.89
```

[1.73, 1.68, 1.71, 1.89] →

max()

# Example

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

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

```
max(fam)
```

```
1.89
```

[1.73, 1.68, 1.71, 1.89] →  → 1.89

# Example

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

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

```
max(fam)
```

```
1.89
```

```
tallest = max(fam)  
tallest
```

```
1.89
```

# round()

```
round(1.68, 1)
```

```
1.7
```

```
round(1.68)
```

```
2
```

```
help(round) # Open up documentation
```

```
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.
```

# round()

```
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.

round()

# round()

```
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.

```
round(1.68, 1)
```

round()



# round()

```
help(round)
```

Help on built-in function round in module builtins:

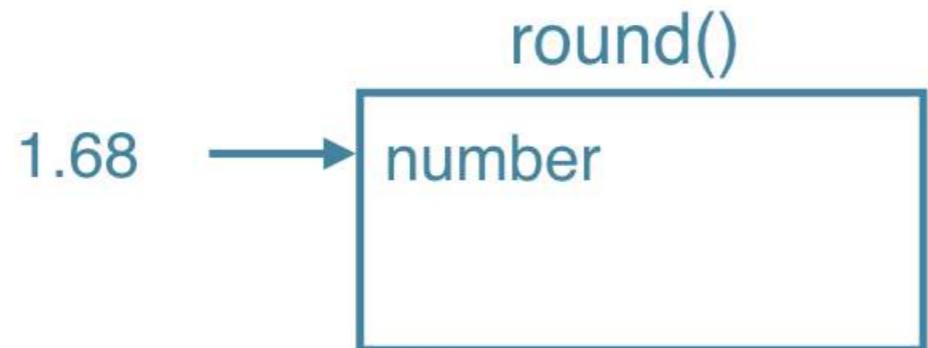
```
round(number, ndigits=None)
```

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.

```
round(1.68, 1)
```



# round()

```
help(round)
```

Help on built-in function round in module builtins:

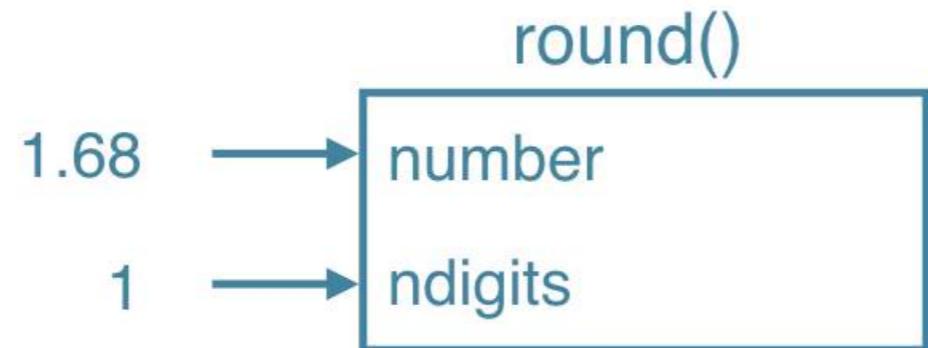
```
round(number, ndigits=None)
```

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.

```
round(1.68, 1)
```



# round()

```
help(round)
```

Help on built-in function round in module builtins:

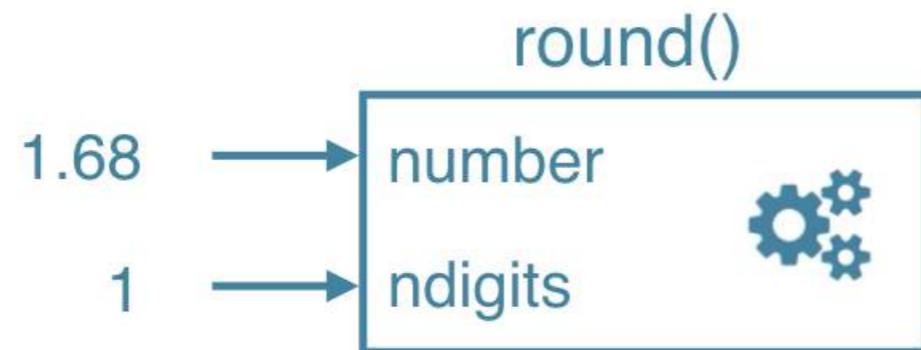
```
round(number, ndigits=None)
```

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.

```
round(1.68, 1)
```



# round()

```
help(round)
```

Help on built-in function round in module builtins:

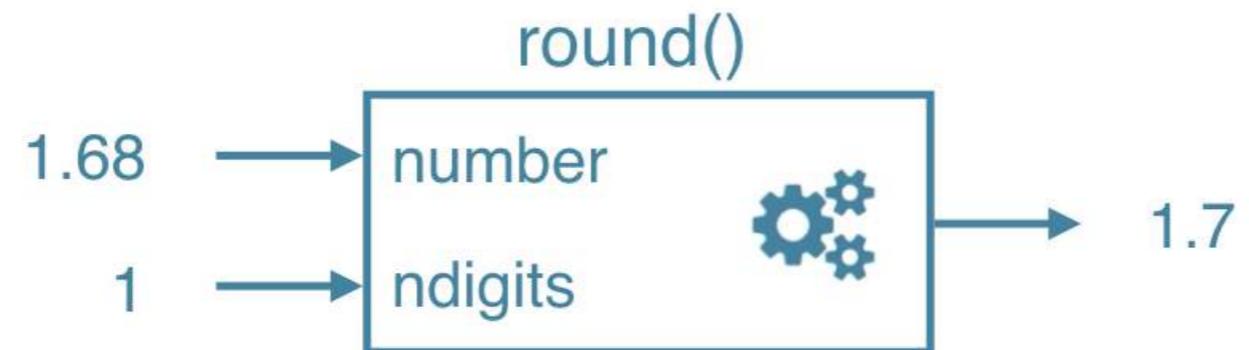
```
round(number, ndigits=None)
```

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.

```
round(1.68, 1)
```



# round()

```
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.

round()

# round()

```
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.

```
round(1.68)
```

round()

# round()

```
help(round)
```

Help on built-in function round in module builtins:

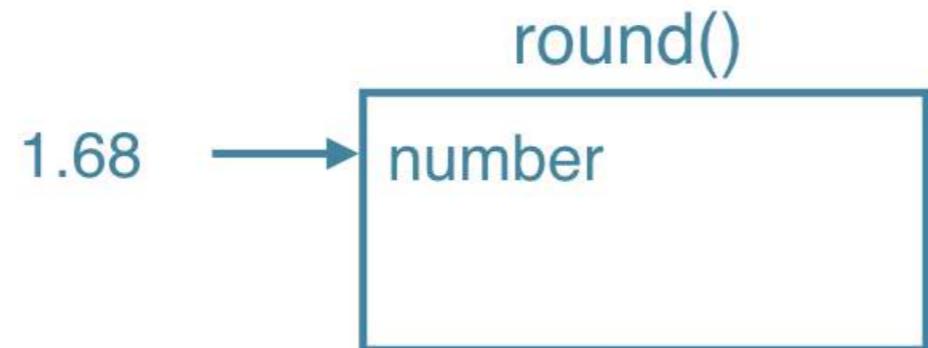
```
round(number, ndigits=None)
```

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.

```
round(1.68)
```



# round()

```
help(round)
```

Help on built-in function round in module builtins:

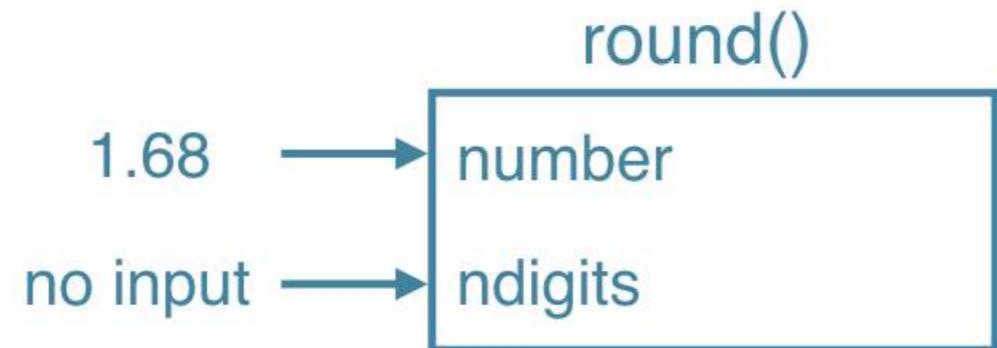
```
round(number, ndigits=None)
```

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.

```
round(1.68)
```



# round()

```
help(round)
```

Help on built-in function round in module builtins:

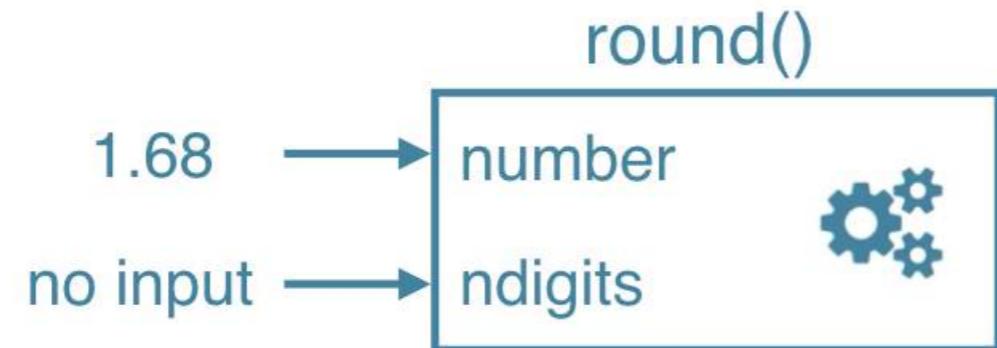
```
round(number, ndigits=None)
```

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.

```
round(1.68)
```



# round()

```
help(round)
```

Help on built-in function round in module builtins:

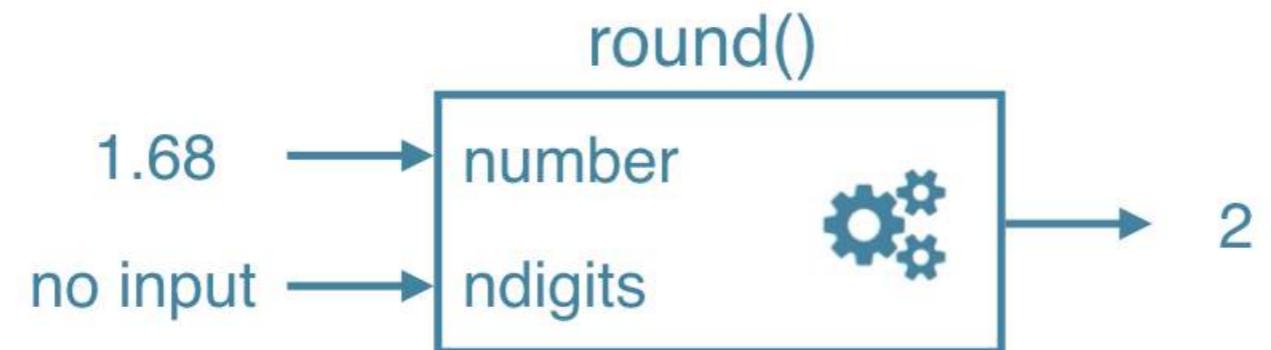
```
round(number, ndigits=None)
```

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.

```
round(1.68)
```



# round()

```
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.

- `round(number)`
- `round(number, ndigits)`

# Find functions

- How to know?
- Standard task -> probably function exists!
- The internet is your friend

# **Let's practice!**

**INTRODUCTION TO PYTHON**

# Methods

INTRODUCTION TO PYTHON



**Hugo Bowne-Anderson**  
Data Scientist at DataCamp

# Built-in Functions

- Maximum of list: `max()`
- Length of list or string: `len()`
- Get index in list: ?
- Reversing a list: ?

# Back 2 Basics

```
sister = "liz"
```

Object

```
height = 1.73
```

Object

```
fam = ["liz", 1.73, "emma", 1.68,  
       "mom", 1.71, "dad", 1.89]
```

Object

# Back 2 Basics

```
sister = "liz"
```

```
height = 1.73
```

```
fam = ["liz", 1.73, "emma", 1.68,  
       "mom", 1.71, "dad", 1.89]
```

type

Object str

Object float

Object list

- Methods: Functions that belong to objects

# Back 2 Basics

```
sister = "liz"
```

```
height = 1.73
```

```
fam = ["liz", 1.73, "emma", 1.68,  
       "mom", 1.71, "dad", 1.89]
```

	type	examples of methods
Object	str	capitalize() replace()

Object	float	bit_length() conjugate()
--------	-------	-----------------------------

Object	list	index() count()
--------	------	--------------------

- Methods: Functions that belong to objects

# list methods

```
fam
```

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

```
fam.index("mom") # "Call method index() on fam"
```

```
4
```

```
fam.count(1.73)
```

```
1
```

# str methods

```
sister
```

```
'liz'
```

```
sister.capitalize()
```

```
'Liz'
```

```
sister.replace("z", "sa")
```

```
'lisa'
```

# Methods

- Everything = object
- Object have methods associated, depending on type

```
sister.replace("z", "sa")
```

```
'lisa'
```

```
fam.replace("mom", "mommy")
```

```
AttributeError: 'list' object has no attribute 'replace'
```

# Methods

```
sister.index("z")
```

2

```
fam.index("mom")
```

4

# Methods (2)

```
fam
```

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

```
fam.append("me")
```

```
fam
```

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89, 'me']
```

```
fam.append(1.79)
```

```
fam
```

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89, 'me', 1.79]
```

# Summary

## Functions

```
type(fam)
```

```
list
```

## Methods: call functions on objects

```
fam.index("dad")
```

```
6
```

# **Let's practice!**

**INTRODUCTION TO PYTHON**

# Packages

INTRODUCTION TO PYTHON



**Hugo Bowne-Anderson**  
Data Scientist at DataCamp

# Motivation

- Functions and methods are powerful
- All code in Python distribution?
  - Huge code base: messy
  - Lots of code you won't use
  - Maintenance problem

# Packages

- Directory of Python Scripts
- Each script = module
- Specify functions, methods, types
- Thousands of packages available
  - NumPy
  - Matplotlib
  - scikit-learn

```
pkg/  
    mod1.py  
    mod2.py  
    ...
```

# Install package

- <http://pip.readthedocs.org/en/stable/installing/>
- Download `get-pip.py`
- Terminal:
  - `python3 get-pip.py`
  - `pip3 install numpy`

# Import package

```
import numpy  
array([1, 2, 3])
```

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

```
numpy.array([1, 2, 3])
```

```
array([1, 2, 3])
```

```
import numpy as np  
np.array([1, 2, 3])
```

```
array([1, 2, 3])
```

```
from numpy import array  
array([1, 2, 3])
```

```
array([1, 2, 3])
```

# from numpy import array

- my\_script.py

```
from numpy import array

fam = ["liz", 1.73, "emma", 1.68,
       "mom", 1.71, "dad", 1.89]

...
fam_ext = fam + ["me", 1.79]

...
print(str(len(fam_ext)) + " elements in fam_ext")

...
np_fam = array(fam_ext)
```

- Using NumPy, but not very clear

# import numpy

```
import numpy as np

fam = ["liz", 1.73, "emma", 1.68,
       "mom", 1.71, "dad", 1.89]

...
fam_ext = fam + ["me", 1.79]

...
print(str(len(fam_ext)) + " elements in fam_ext")

...
np_fam = np.array(fam_ext) # Clearly using NumPy
```

# **Let's practice!**

**INTRODUCTION TO PYTHON**

# NumPy

## INTRODUCTION TO PYTHON



**Hugo Bowne-Anderson**  
Data Scientist at DataCamp

# Lists Recap

- Powerful
- Collection of values
- Hold different types
- Change, add, remove
- Need for Data Science
  - Mathematical operations over collections
  - Speed

# Illustration

```
height = [1.73, 1.68, 1.71, 1.89, 1.79]  
height
```

```
[1.73, 1.68, 1.71, 1.89, 1.79]
```

```
weight = [65.4, 59.2, 63.6, 88.4, 68.7]  
weight
```

```
[65.4, 59.2, 63.6, 88.4, 68.7]
```

```
weight / height ** 2
```

```
TypeError: unsupported operand type(s) for ** or pow(): 'list' and 'int'
```

# Solution: NumPy

- Numeric Python
- Alternative to Python List: NumPy Array
- Calculations over entire arrays
- Easy and Fast
- Installation
  - In the terminal: `pip3 install numpy`

# NumPy

```
import numpy as np  
np_height = np.array(height)  
np_height
```

```
array([1.73, 1.68, 1.71, 1.89, 1.79])
```

```
np_weight = np.array(weight)  
np_weight
```

```
array([65.4, 59.2, 63.6, 88.4, 68.7])
```

```
bmi = np_weight / np_height ** 2  
bmi
```

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

# Comparison

```
height = [1.73, 1.68, 1.71, 1.89, 1.79]  
weight = [65.4, 59.2, 63.6, 88.4, 68.7]  
weight / height ** 2
```

```
TypeError: unsupported operand type(s) for ** or pow(): 'list' and 'int'
```

```
np_height = np.array(height)  
np_weight = np.array(weight)  
np_weight / np_height ** 2
```

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

# NumPy: remarks

```
np.array([1.0, "is", True])
```

```
array(['1.0', 'is', 'True'], dtype='<U32')
```

- NumPy arrays: contain only one type

# NumPy: remarks

```
python_list = [1, 2, 3]  
numpy_array = np.array([1, 2, 3])
```

```
python_list + python_list
```

```
[1, 2, 3, 1, 2, 3]
```

```
numpy_array + numpy_array
```

```
array([2, 4, 6])
```

- Different types: different behavior!

# NumPy Subsetting

```
bmi
```

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

```
bmi[1]
```

```
20.975
```

```
bmi > 23
```

```
array([False, False, False, True, False])
```

```
bmi[bmi > 23]
```

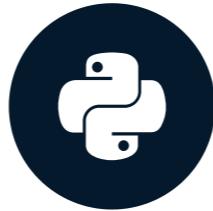
```
array([24.7473475])
```

# **Let's practice!**

**INTRODUCTION TO PYTHON**

# 2D NumPy Arrays

INTRODUCTION TO PYTHON



**Hugo Bowne-Anderson**  
Data Scientist at DataCamp

# Type of NumPy Arrays

```
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 Arrays

```
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
```

```
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
```

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

```
array([['1.73', '1.68', '1.71', '1.89', '1.79'],  
      ['65.4', '59.2', '63.6', '88.4', '68.7']], dtype='|<U32')
```

# Subsetting

```
0      1      2      3      4
```

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

```
np_2d[0]
```

```
array([1.73, 1.68, 1.71, 1.89, 1.79])
```

# Subsetting

```
0      1      2      3      4
```

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

```
np_2d[0][2]
```

```
1.71
```

```
np_2d[0, 2]
```

```
1.71
```

# Subsetting

```
0      1      2      3      4
```

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

```
np_2d[:, 1:3]
```

```
array([[ 1.68,   1.71],  
       [59.2 ,  63.6 ]])
```

```
np_2d[1, :]
```

```
array([65.4, 59.2, 63.6, 88.4, 68.7])
```

# **Let's practice!**

**INTRODUCTION TO PYTHON**

# NumPy: Basic Statistics

INTRODUCTION TO PYTHON



**Hugo Bowne-Anderson**  
Data Scientist at DataCamp

# Data analysis

- Get to know your data
- Little data -> simply look at it
- Big data -> ?

# City-wide survey

```
import numpy as np  
np_city = ... # Implementation left out  
np_city
```

```
array([[1.64, 71.78],  
       [1.37, 63.35],  
       [1.6 , 55.09],  
       ...,  
       [2.04, 74.85],  
       [2.04, 68.72],  
       [2.01, 73.57]])
```

# NumPy

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

```
1.7472
```

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

```
1.75
```

# NumPy

```
np.corrcoef(np_city[:, 0], np_city[:, 1])
```

```
array([[ 1.        , -0.01802],
       [-0.01803,  1.        ]])
```

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

```
0.1992
```

- sum(), sort(), ...
- Enforce single data type: speed!

# Generate data

- Arguments for `np.random.normal()`
  - distribution mean
  - distribution standard deviation
  - number 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))
```

# **Let's practice!**

**INTRODUCTION TO PYTHON**

# Basic plots with Matplotlib

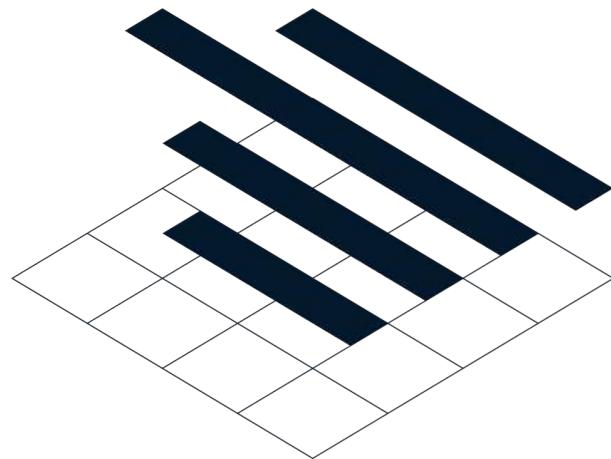
INTERMEDIATE PYTHON



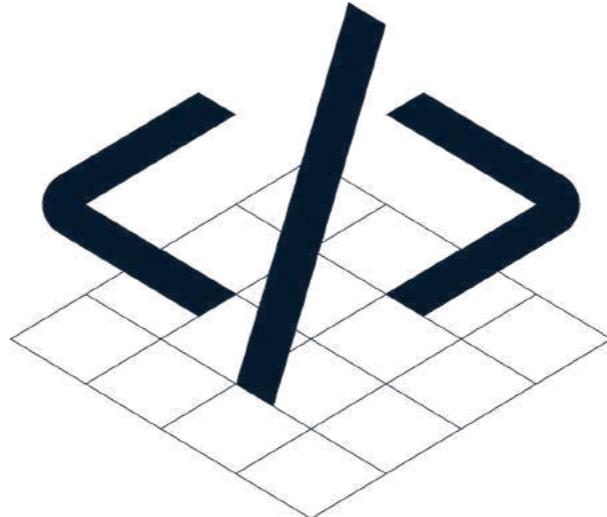
**Hugo Bowne-Anderson**  
Data Scientist at DataCamp

# Basic plots with Matplotlib

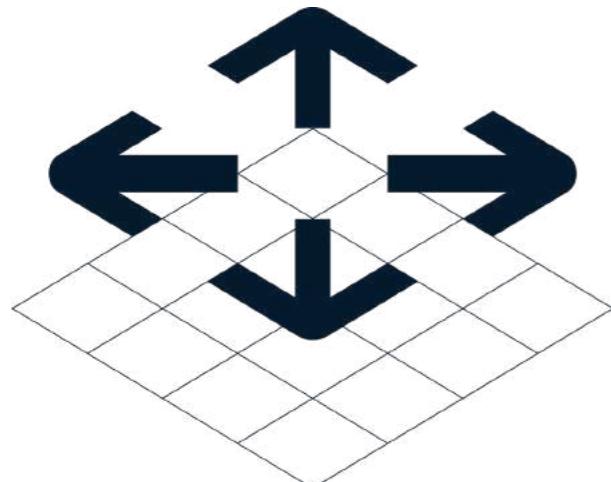
- Visualization



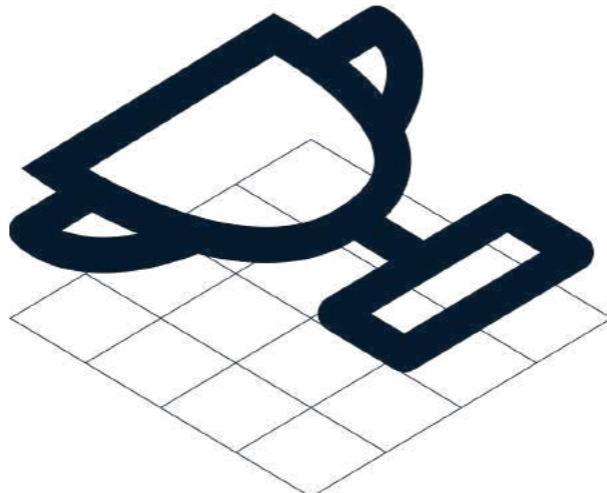
- Data Structure



- Control Structures

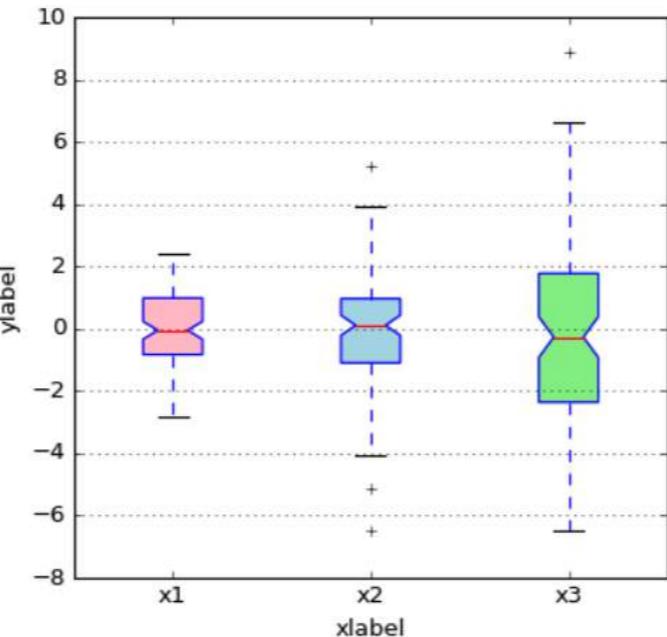
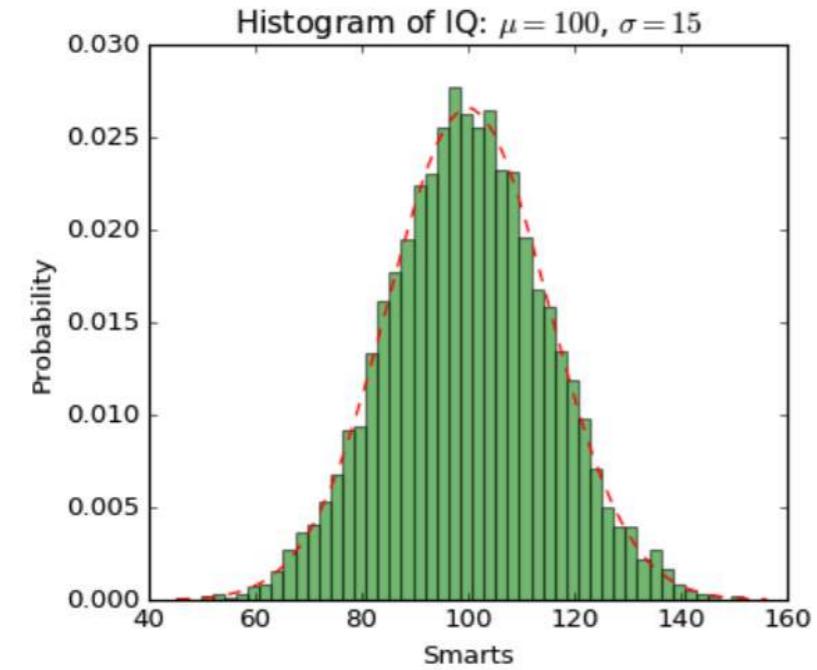


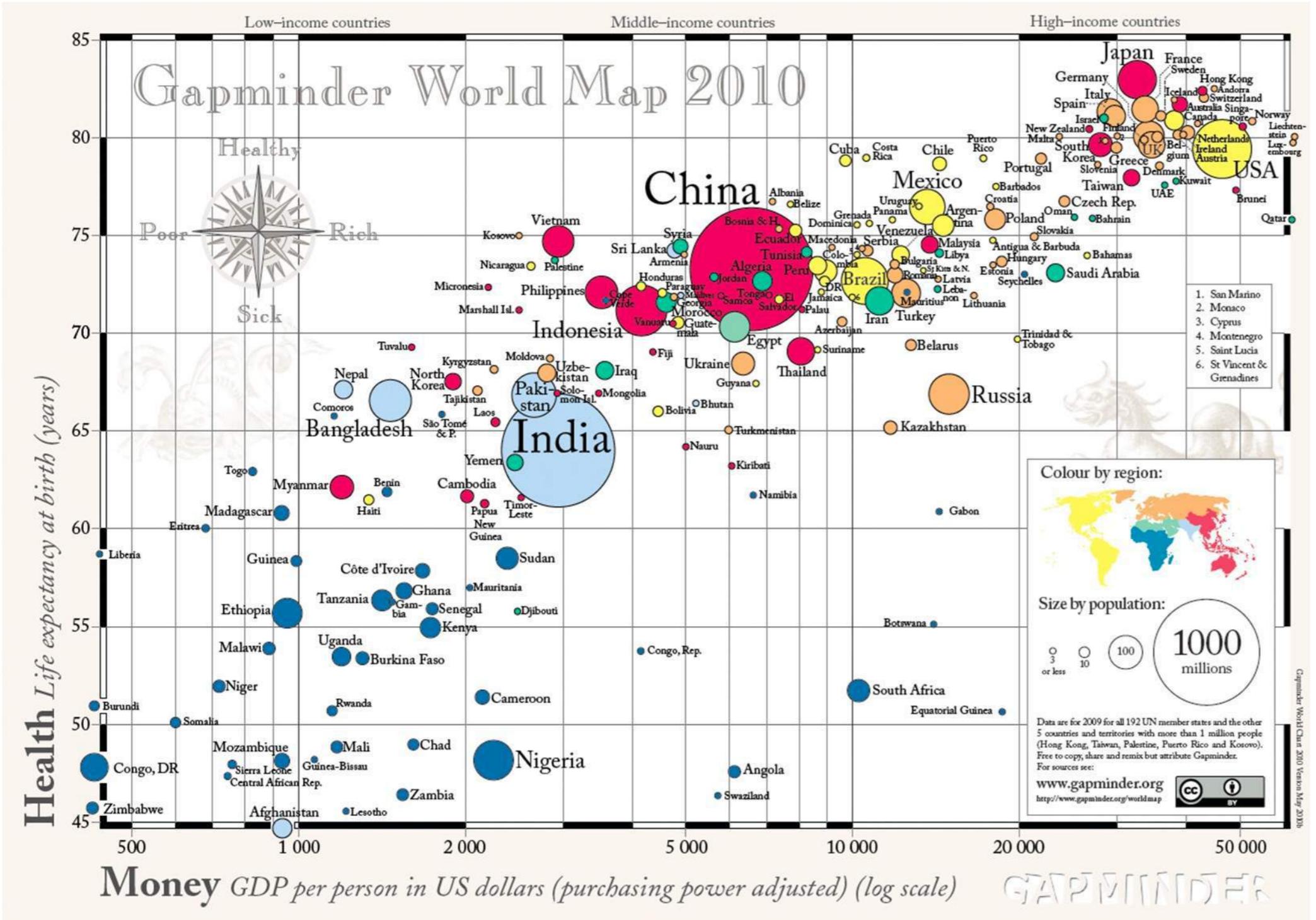
- Case Study



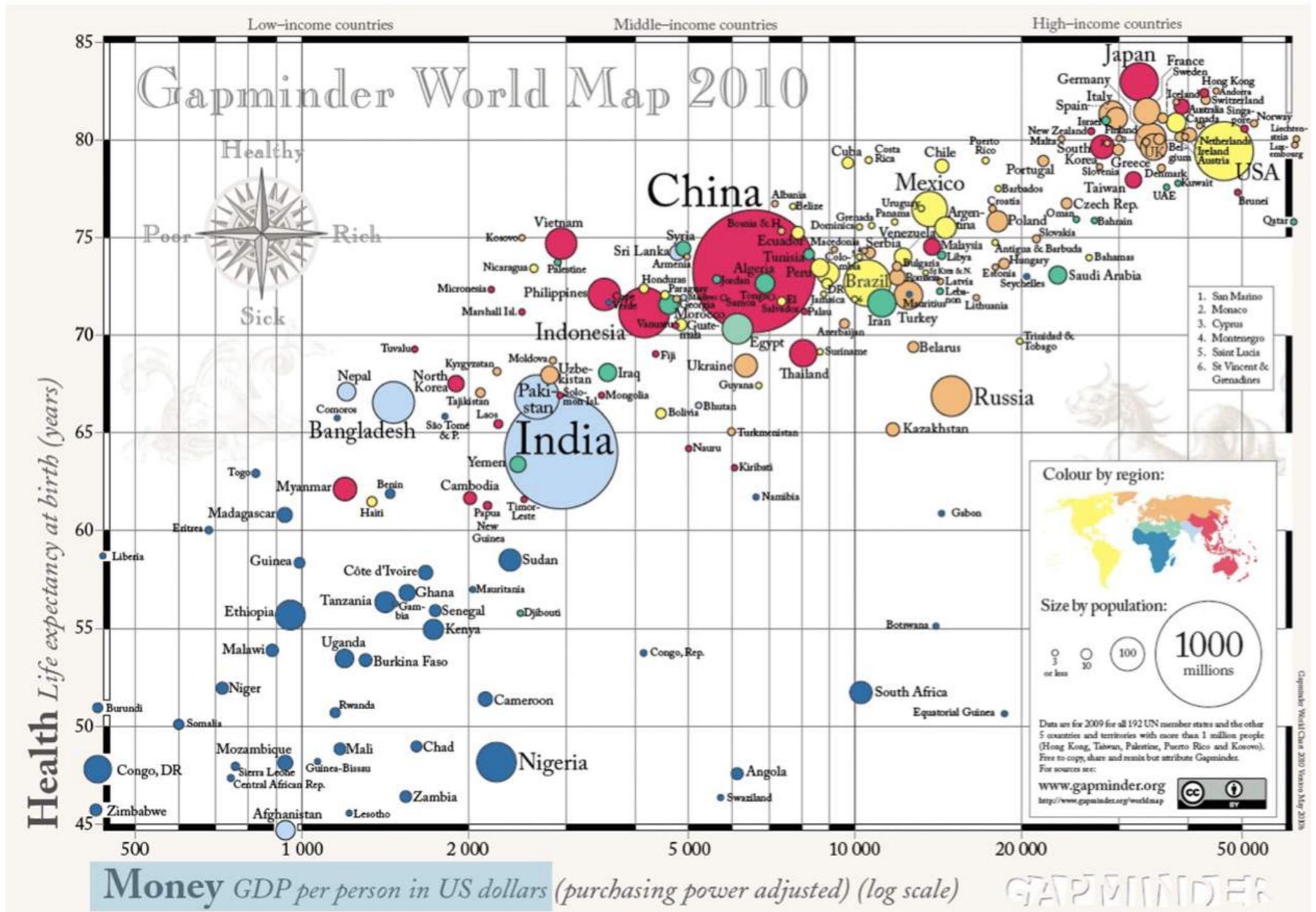
# Data visualization

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

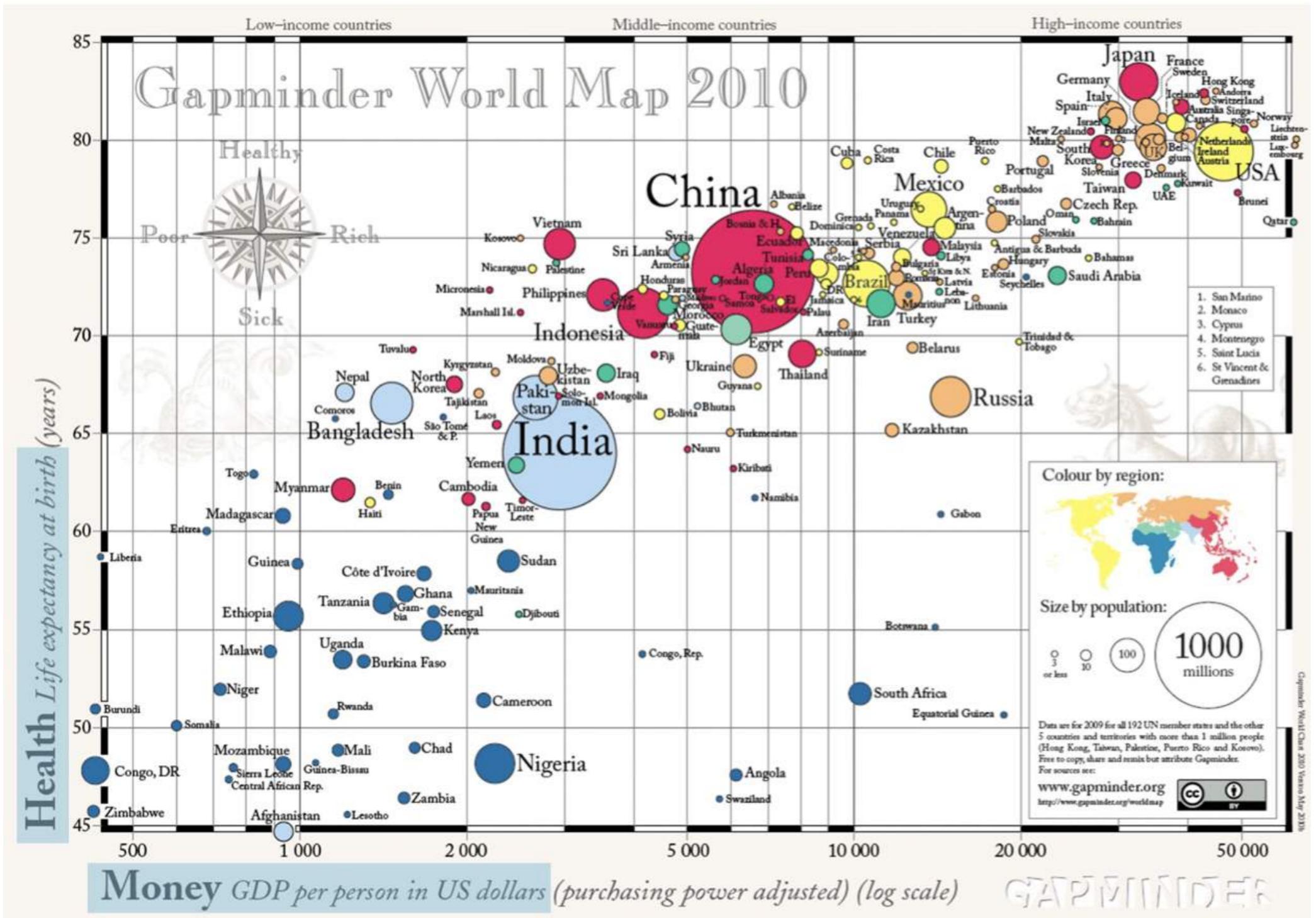




<sup>1</sup> Source: GapMinder, Wealth and Health of Nations



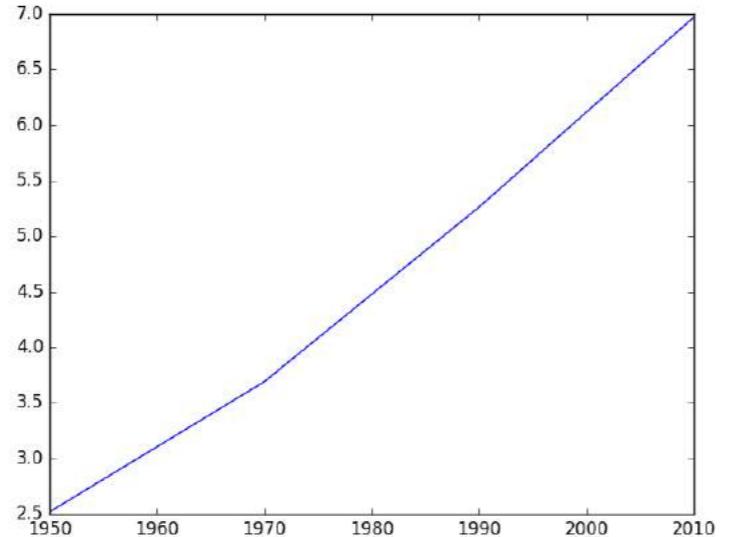
<sup>1</sup> Source: GapMinder, Wealth and Health of Nations



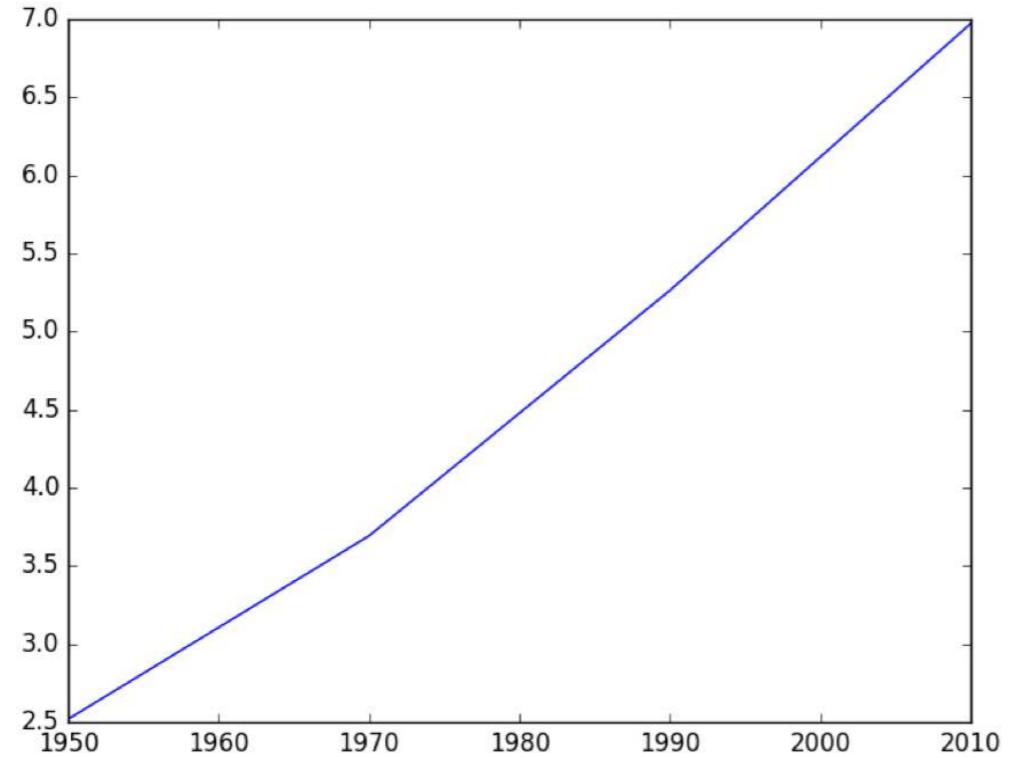
<sup>1</sup> Source: GapMinder, Wealth and Health of Nations

# Matplotlib

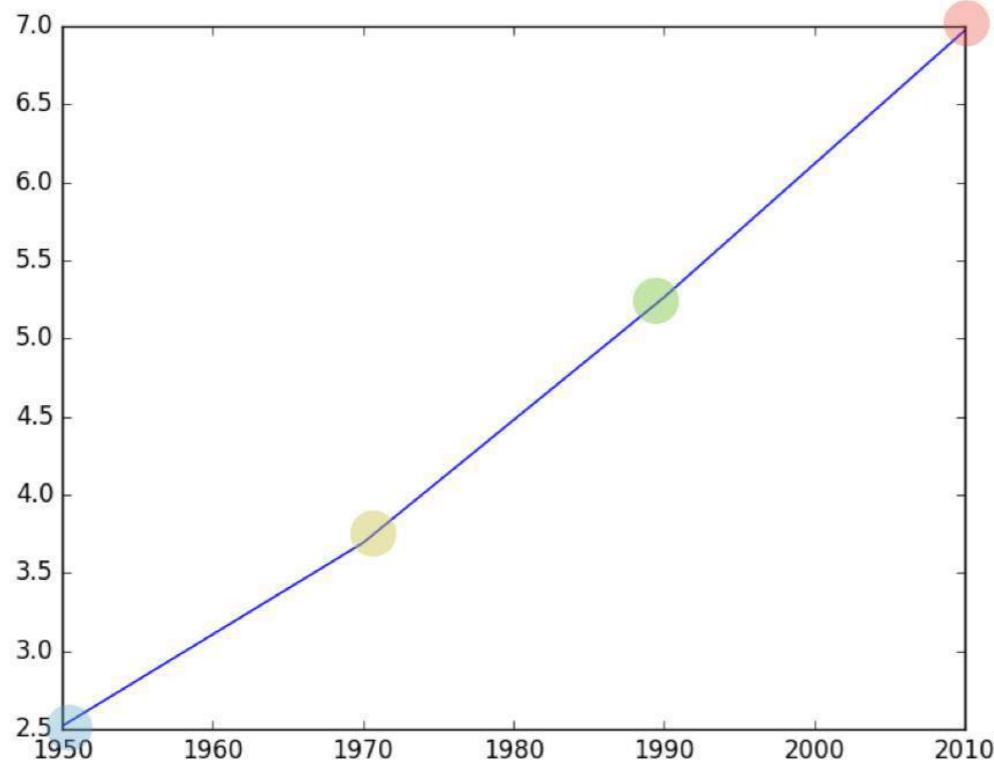
```
import matplotlib.pyplot as plt  
year = [1950, 1970, 1990, 2010]  
pop = [2.519, 3.692, 5.263, 6.972]  
plt.plot(year, pop)  
plt.show()
```



# Matplotlib



# Matplotlib



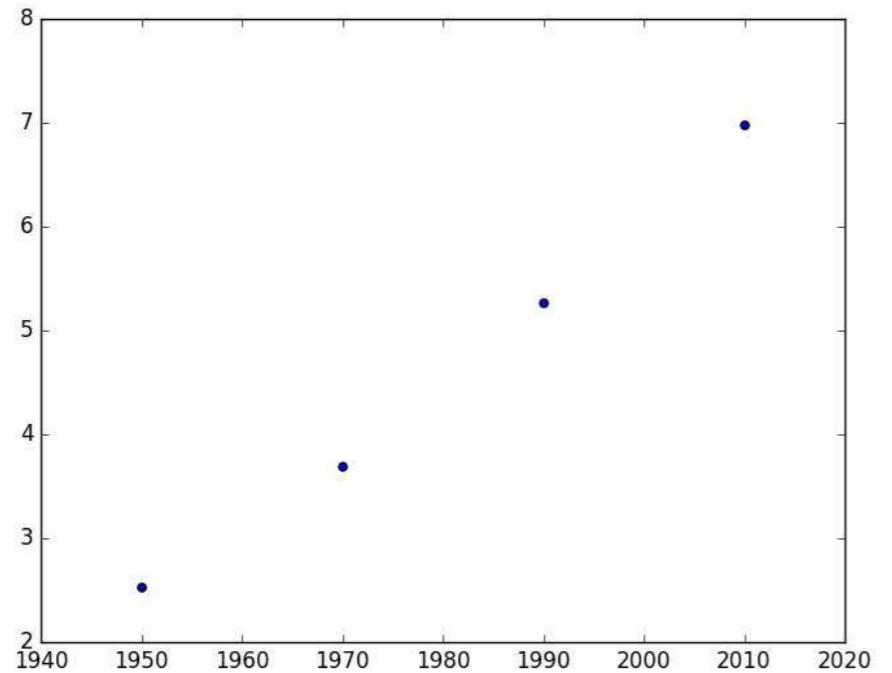
```
year = [1950 , 1970 , 1990 , 2010]  
pop = [2.519, 3.692, 5.263, 6.972]
```

# 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()
```

# 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()
```

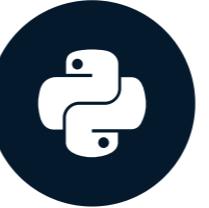


# **Let's practice!**

## **INTERMEDIATE PYTHON**

# Histogram

## INTERMEDIATE PYTHON



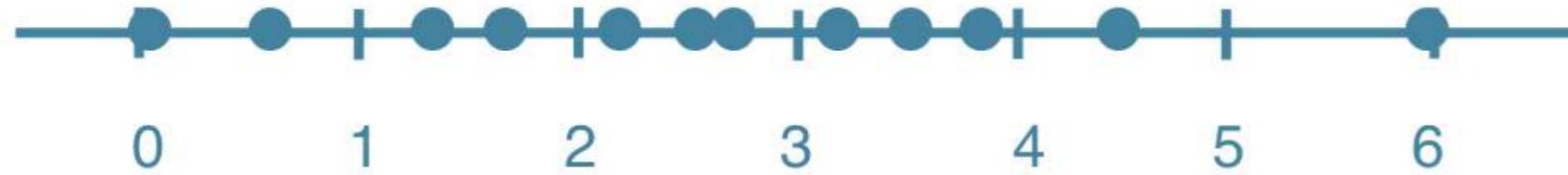
**Hugo Bowne-Anderson**  
Data Scientist at DataCamp

# Histogram

- Explore dataset
- Get idea about distribution

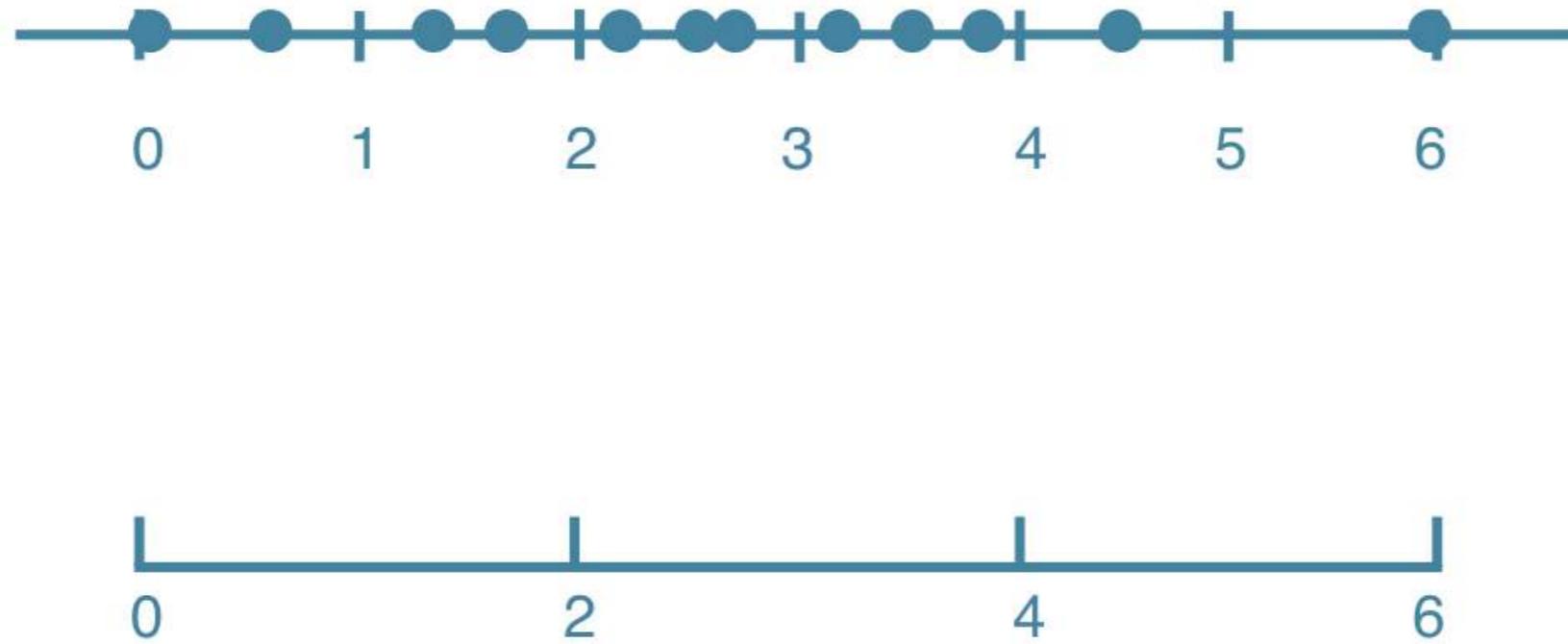
# Histogram

- Explore dataset
- Get idea about distribution



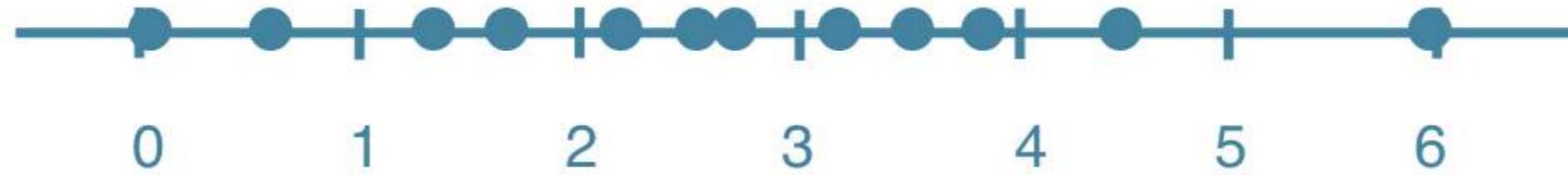
# Histogram

- Explore dataset
- Get idea about distribution



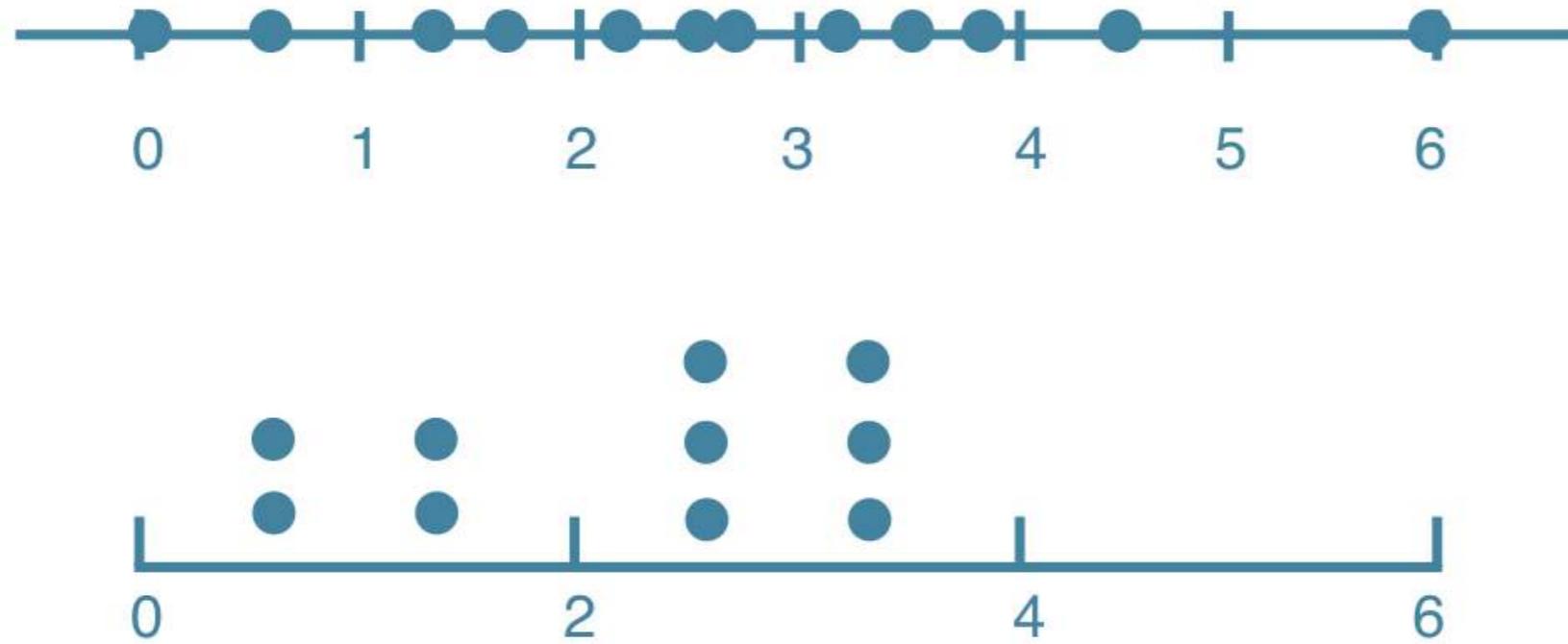
# Histogram

- Explore dataset
- Get idea about distribution



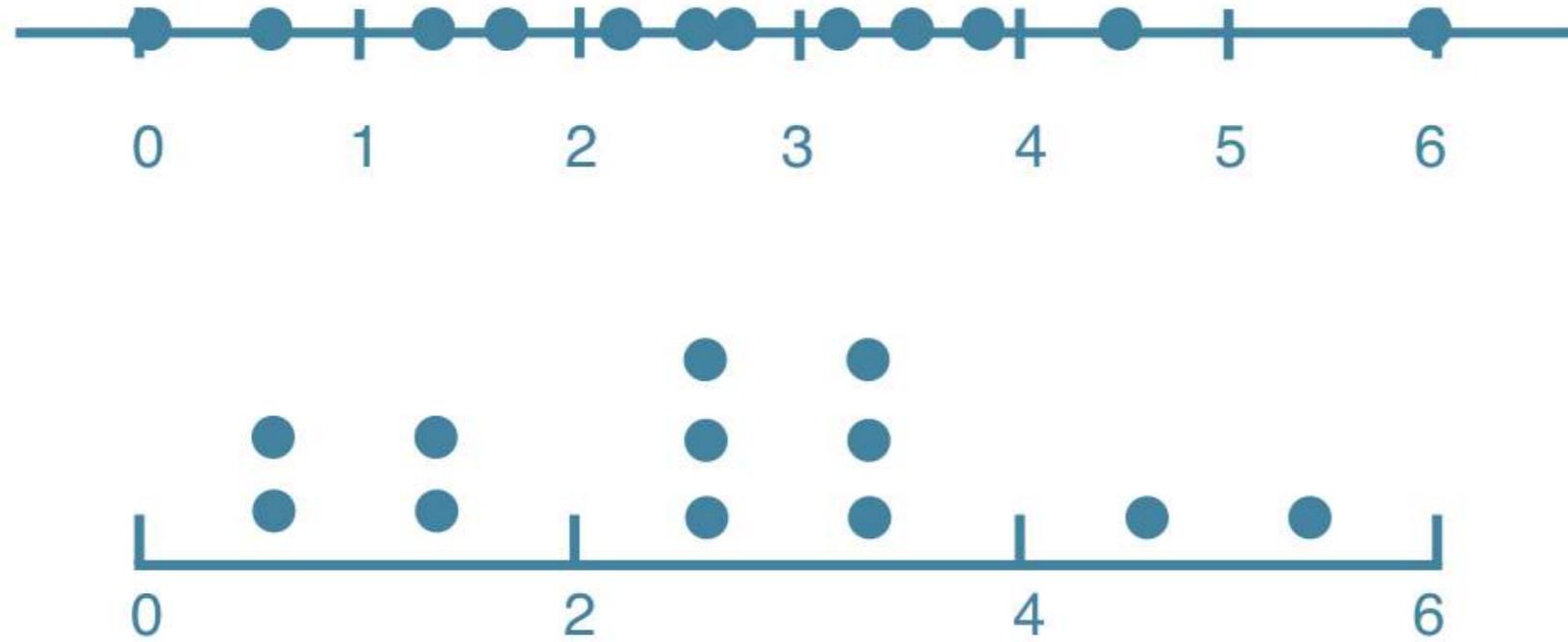
# Histogram

- Explore dataset
- Get idea about distribution



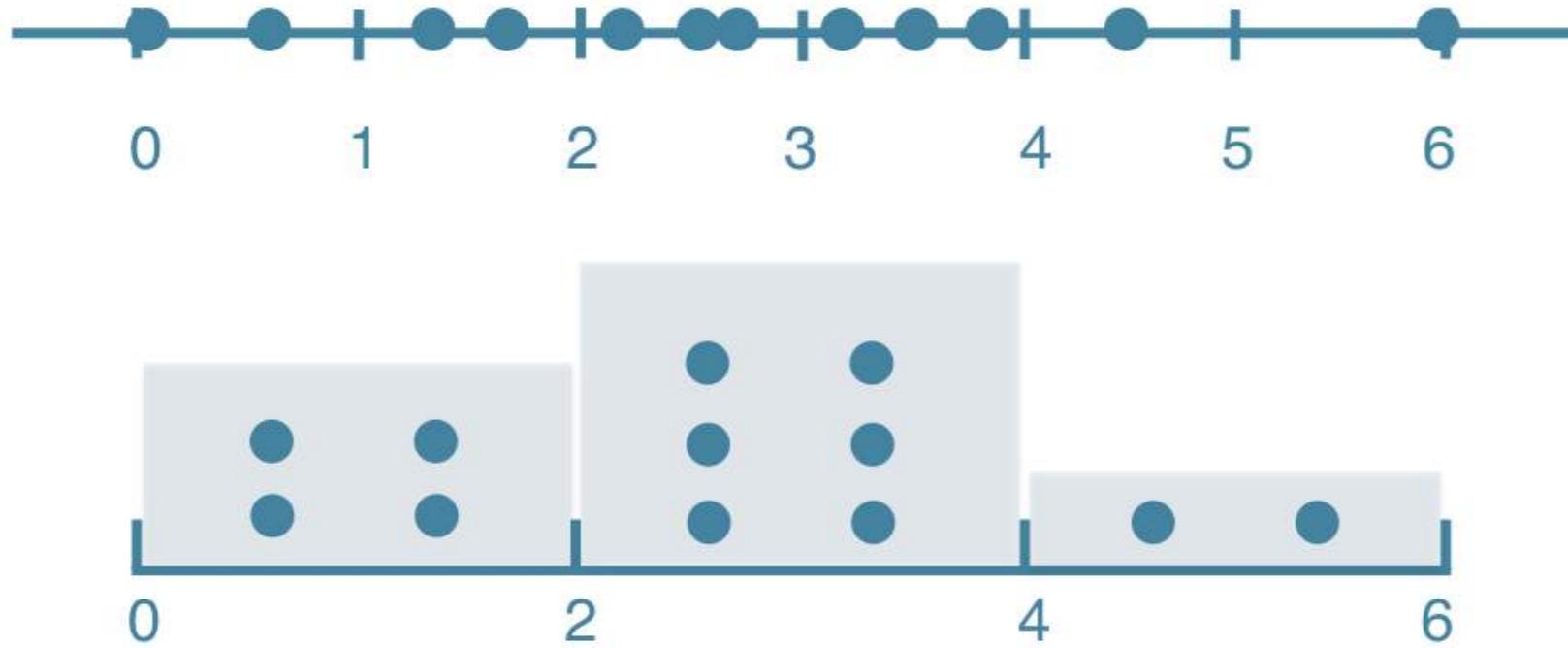
# Histogram

- Explore dataset
- Get idea about distribution



# Histogram

- Explore dataset
- Get idea about distribution



# Matplotlib

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

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

Help on function hist in module matplotlib.pyplot:

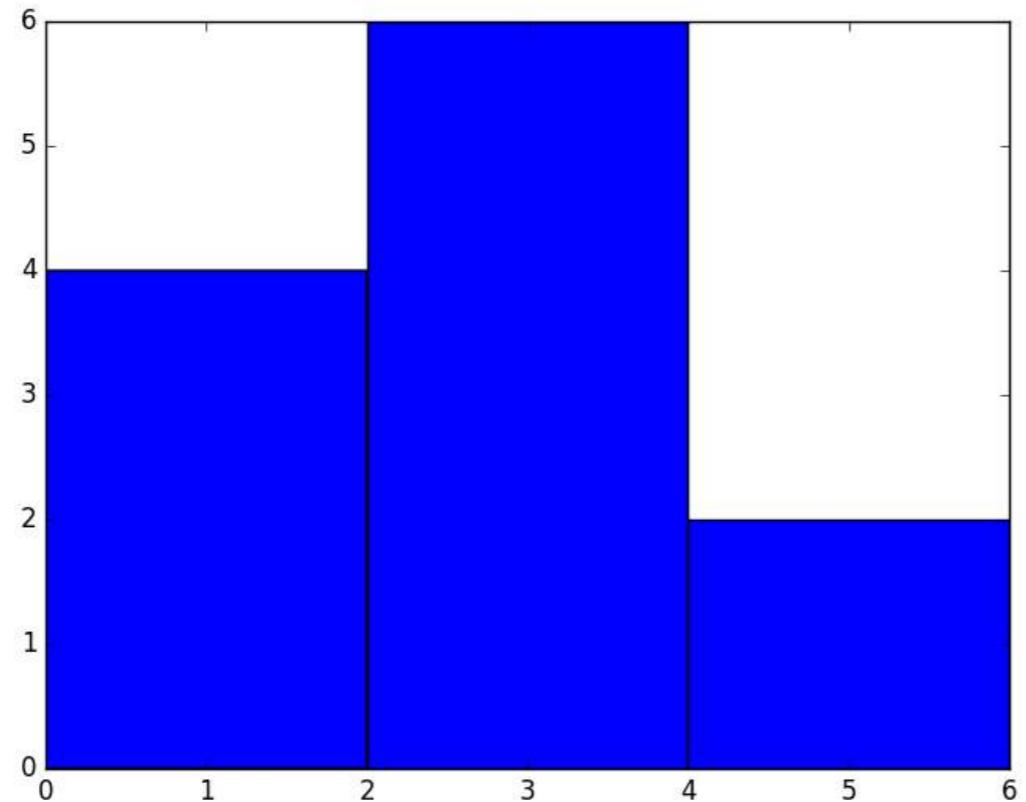
```
hist(x, bins=None, range=None, density=False, weights=None,  
cumulative=False, bottom=None, histtype='bar', align='mid',  
orientation='vertical', rwidth=None, log=False, color=None,  
label=None, stacked=False, *, data=None, **kwargs)
```

Plot a histogram.

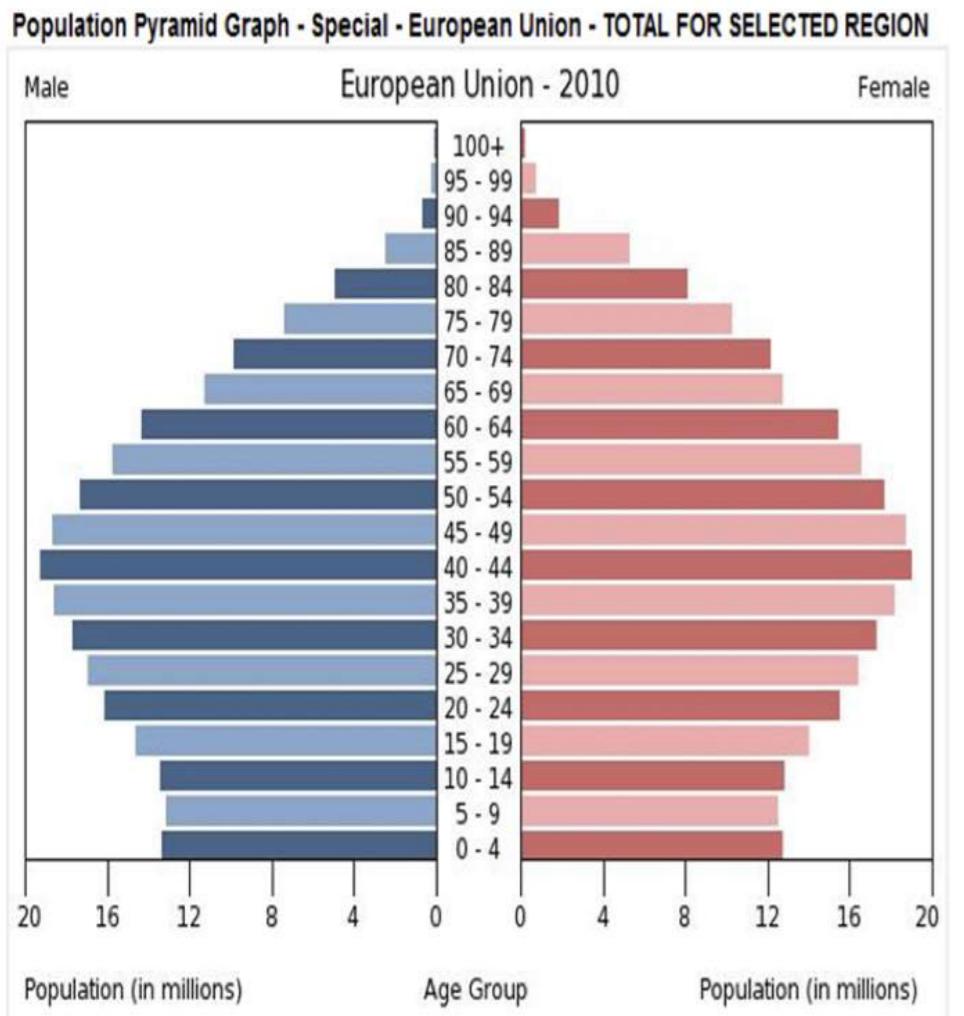
Compute and draw the histogram of `*x*`. The return value is a tuple `(*n*, *bins*, *patches*)` or `([*n0*, *n1*, ...], *bins*, [*patches0*, *patches1*, ...])` if the input contains multiple data.

# Matplotlib example

```
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()
```

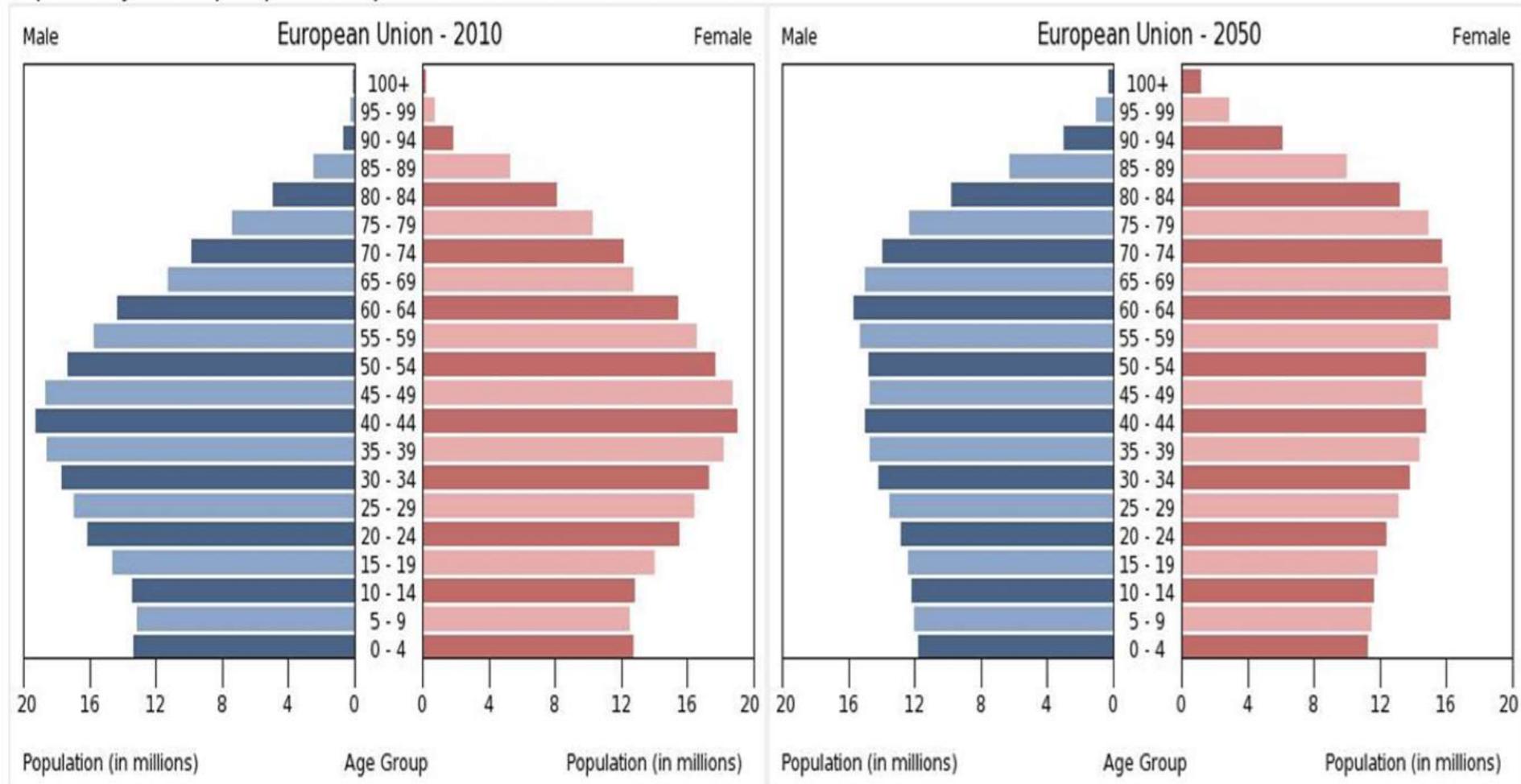


# Population pyramid



# Population pyramid

Population Pyramid Graph - Special - European Union - TOTAL FOR SELECTED REGION



# **Let's practice!**

## **INTERMEDIATE PYTHON**

# Customization

INTERMEDIATE PYTHON



**Hugo Bowne-Anderson**  
Data Scientist at DataCamp

# Data visualization

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

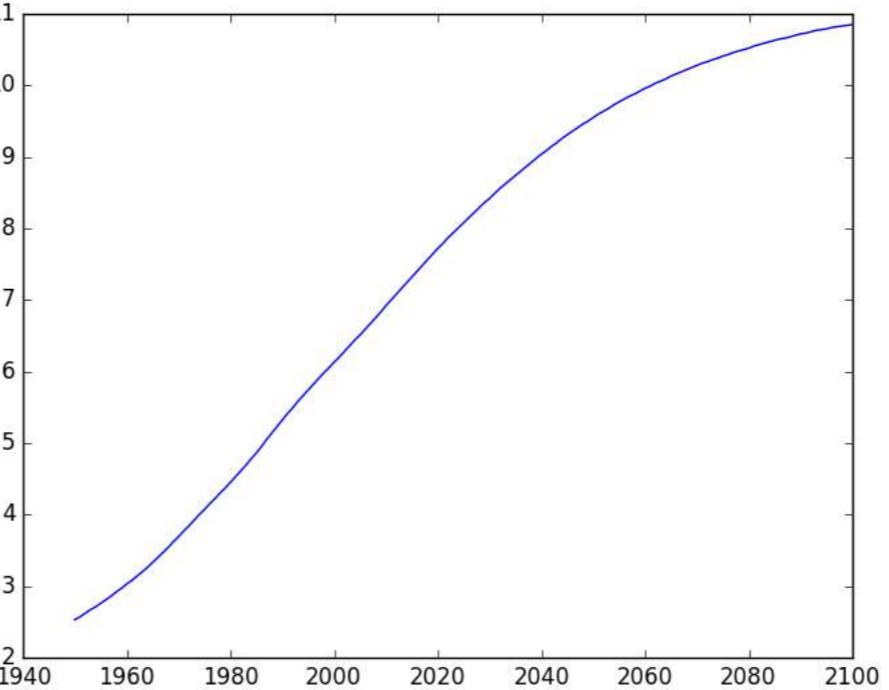
# Basic plot

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

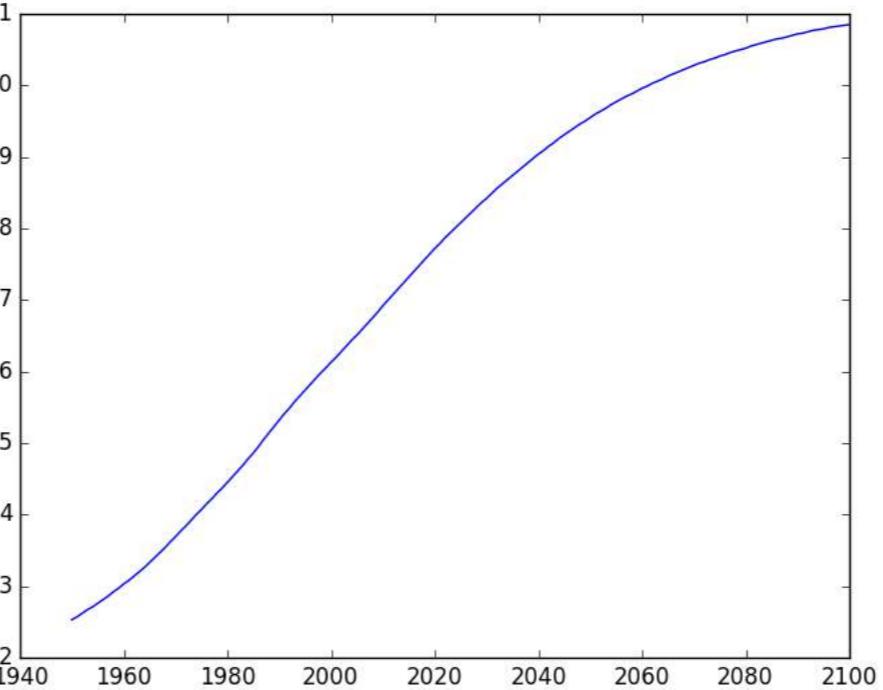
population.py

```
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

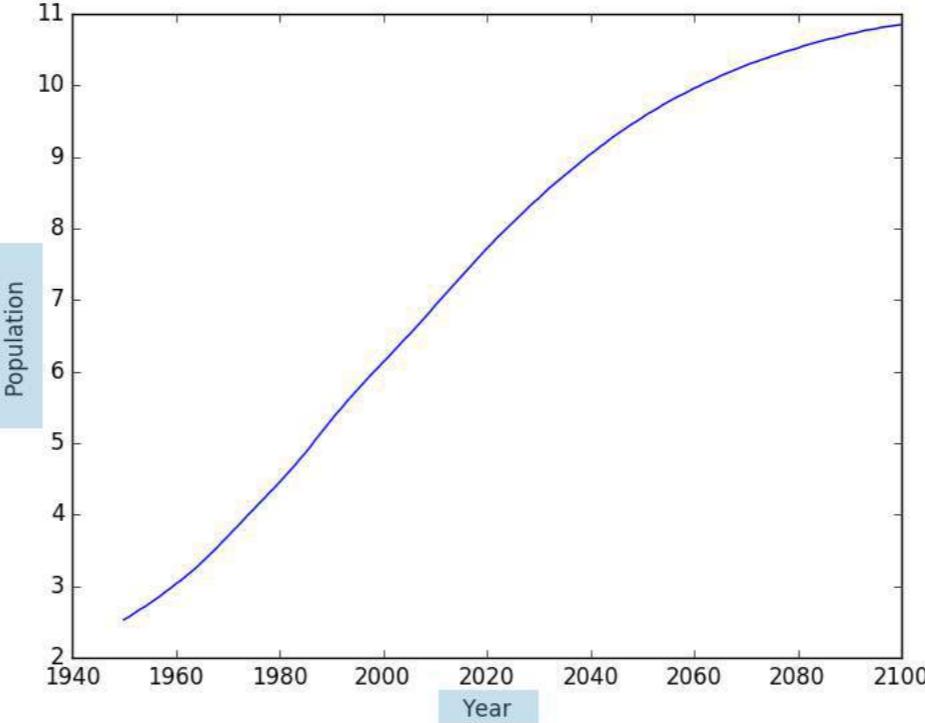
population.py

```
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')

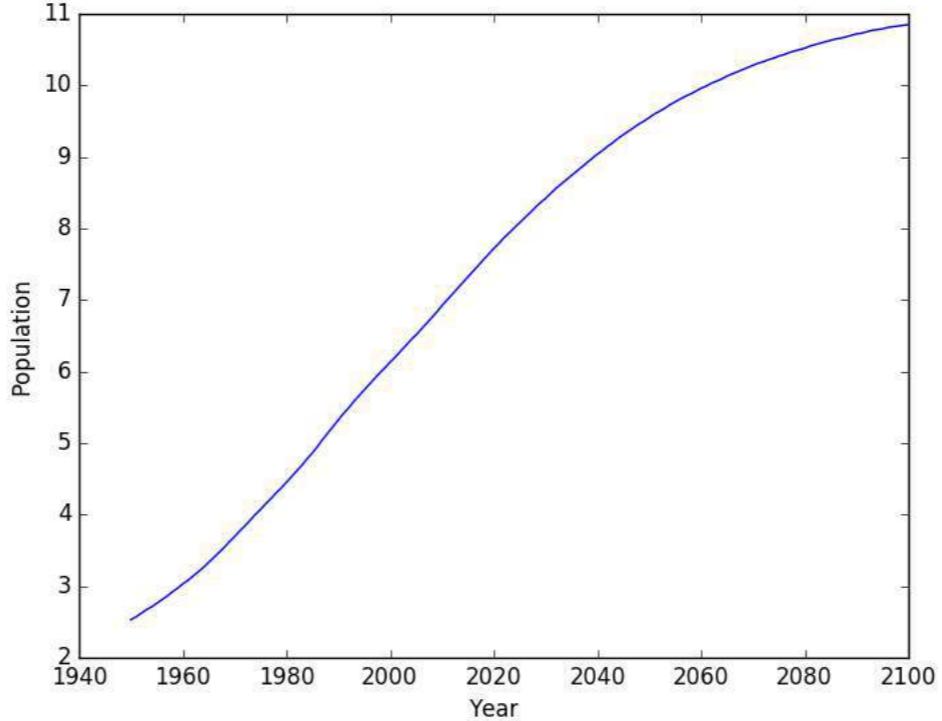
plt.show()
```



# Title

population.py

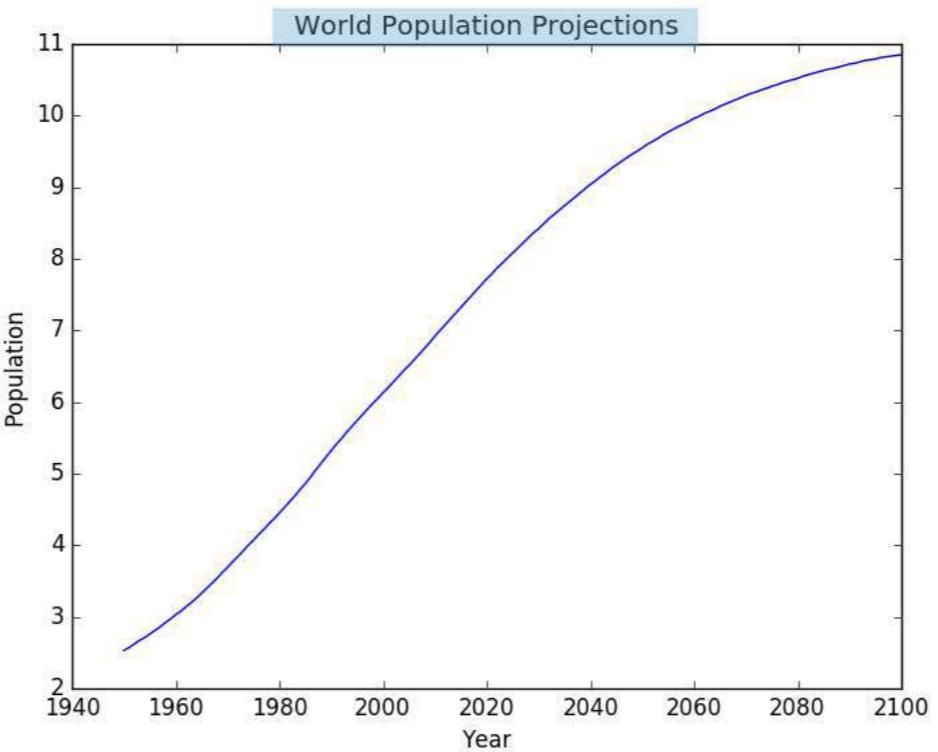
```
import matplotlib.pyplot as plt  
year = [1950, 1951, 1952, ..., 2100]  
pop = [2.538, 2.57, 2.62, ..., 10.85]  
  
plt.plot(year, pop)  
  
plt.xlabel('Year')  
plt.ylabel('Population')  
plt.title('World Population Projections')  
  
plt.show()
```



# 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

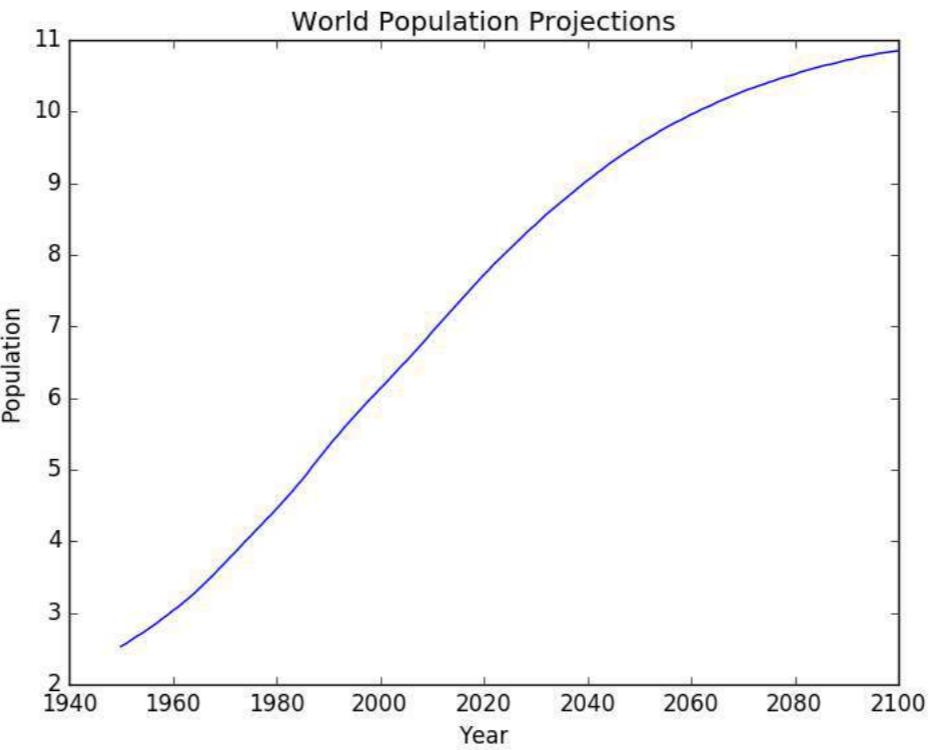
population.py

```
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

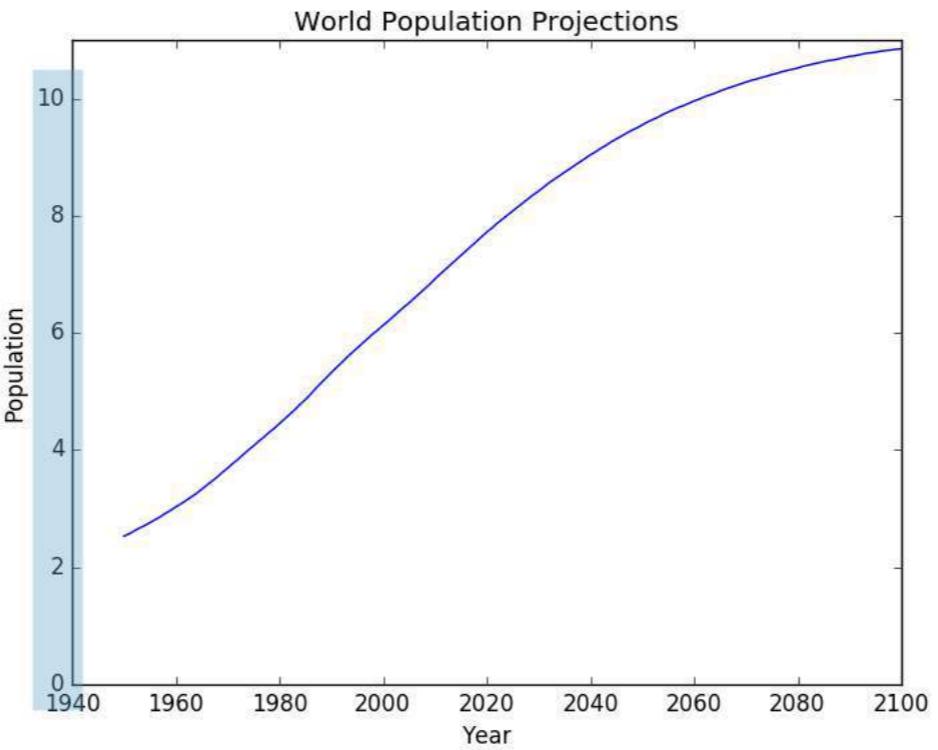
population.py

```
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)

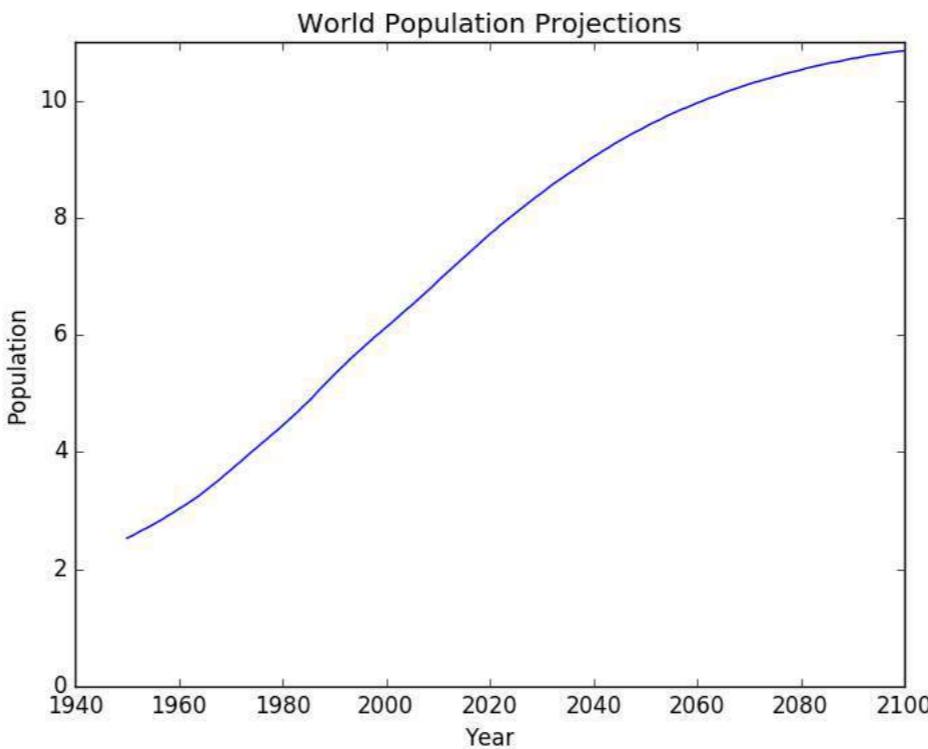
## population.py

```
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)

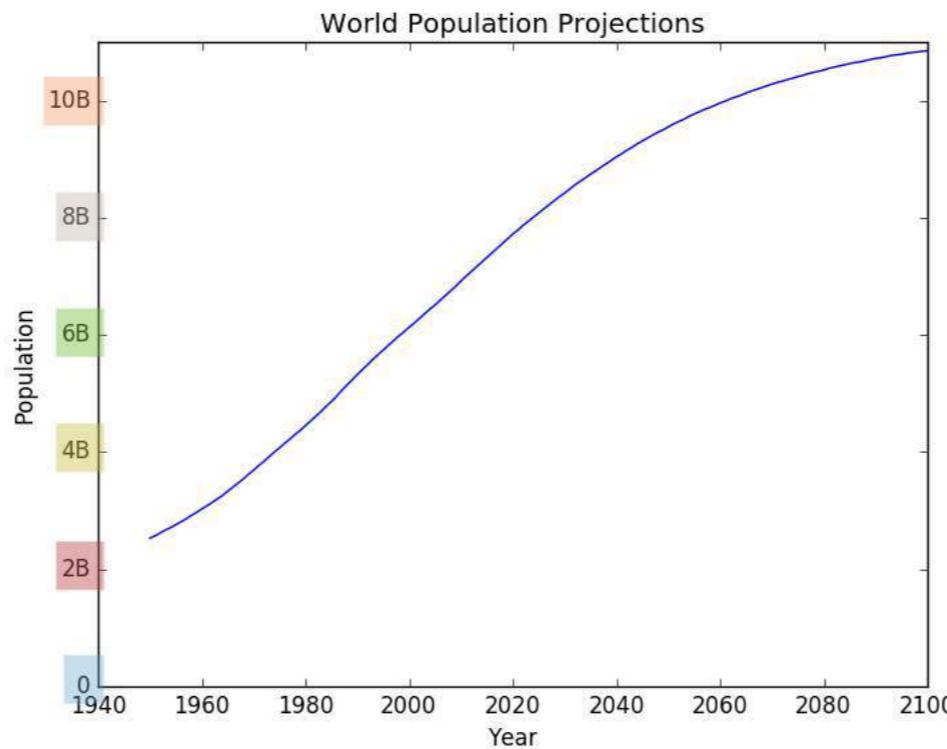
## population.py

```
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

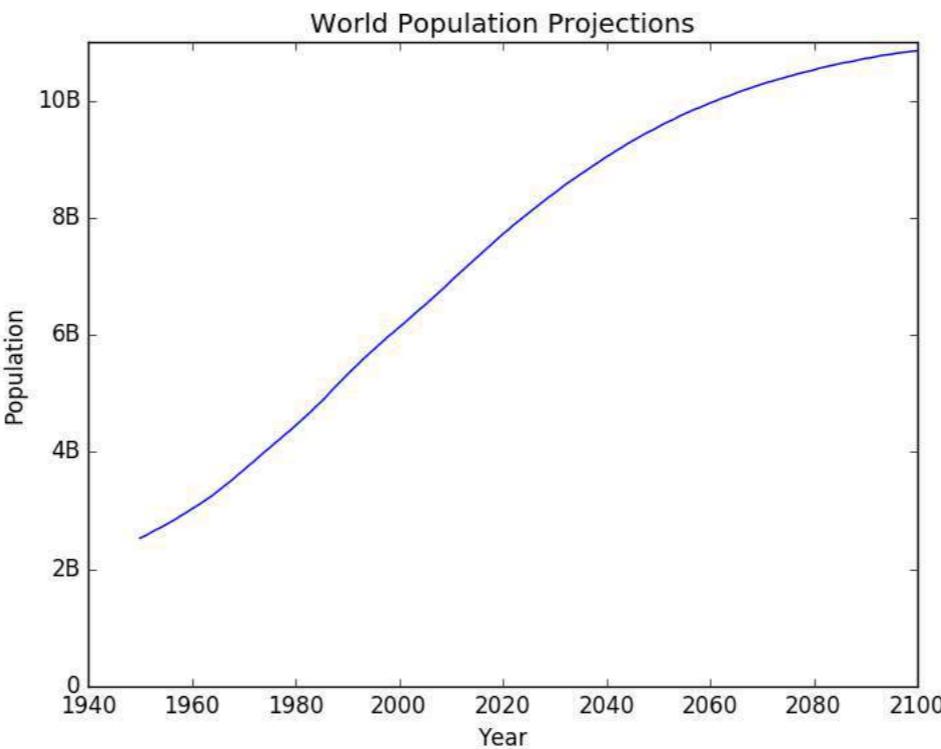
```
import matplotlib.pyplot as plt
year = [1950, 1951, 1952, ..., 2100]
pop = [2.538, 2.57, 2.62, ..., 10.85]

# 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

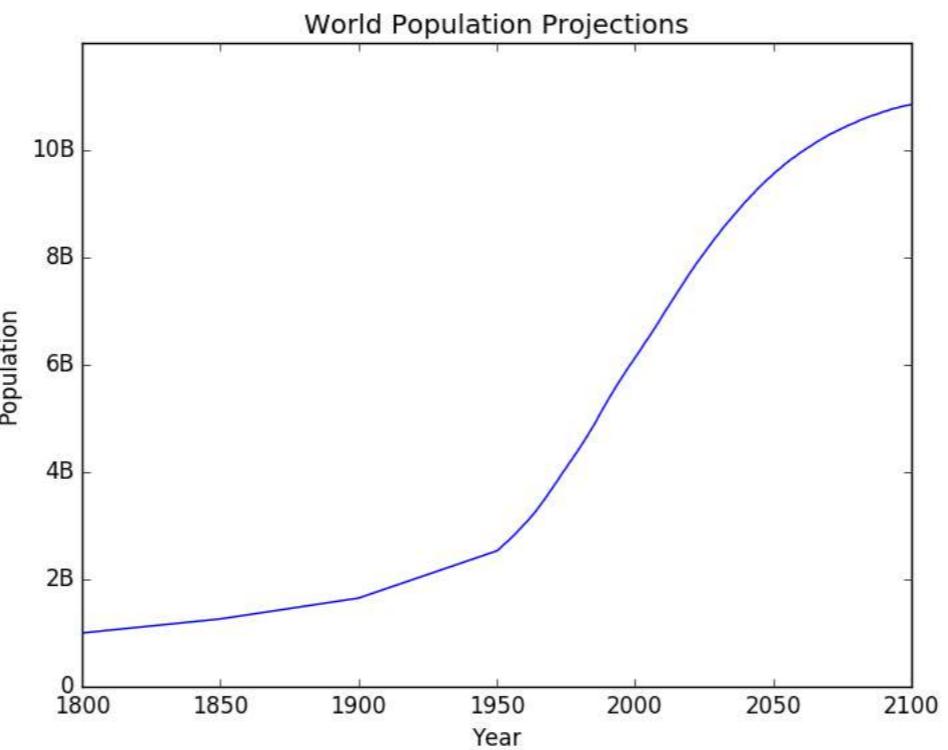
```
import matplotlib.pyplot as plt
year = [1950, 1951, 1952, ..., 2100]
pop = [2.538, 2.57, 2.62, ..., 10.85]

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

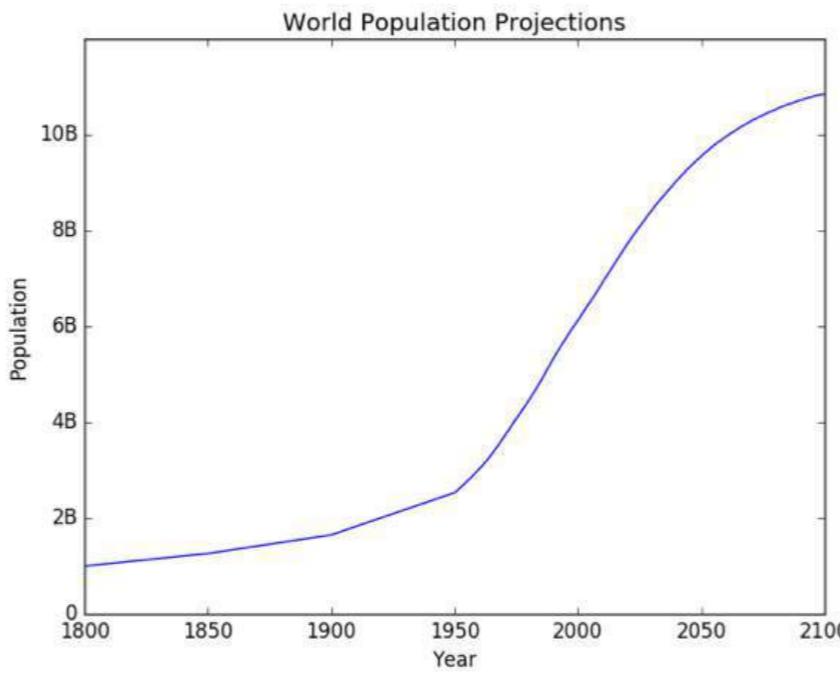
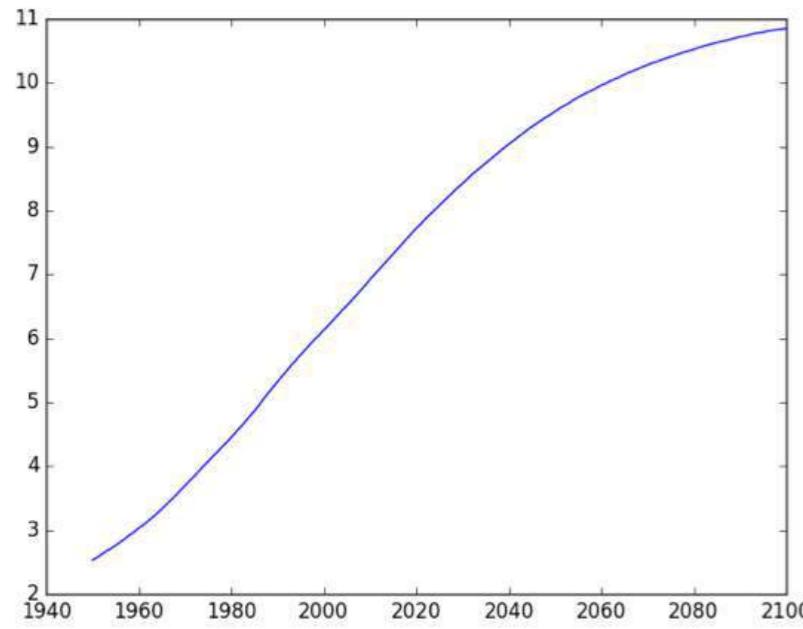
plt.plot(year, pop)

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

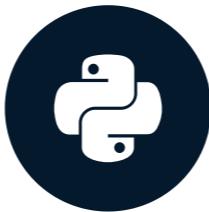


# **Let's practice!**

## **INTERMEDIATE PYTHON**

# Dictionaries, Part 1

INTERMEDIATE PYTHON



**Hugo Bowne-Anderson**  
Data Scientist at DataCamp

# List

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

1

```
pop[ind_alb]
```

2.77

- Not convenient
- Not intuitive

# Dictionary

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

...

{

}

# Dictionary

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

...

```
{"afghanistan":30.55, }
```

# 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
```

# **Let's practice!**

## **INTERMEDIATE PYTHON**

# Dictionaries, Part 2

INTERMEDIATE PYTHON



Hugo Bowne-Anderson  
Data Scientist at DataCamp

# Recap

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

```
2.77
```

```
world = {"afghanistan":30.55, "albania":2.77,  
         "algeria":39.21, "albania":2.81}  
world
```

```
{'afghanistan': 30.55, 'albania': 2.81, 'algeria': 39.21}
```

# Recap

- Keys have to be "immutable" objects

```
{0:"hello", True:"dear", "two":"world"}
```

```
{0: 'hello', True: 'dear', 'two': 'world'}
```

```
{"just", "to", "test": "value"}
```

```
TypeError: unhashable type: 'list'
```

# Principality of Sealand



<sup>1</sup> Source: Wikipedia

# Dictionary

```
world["sealand"] = 0.000027  
world
```

```
{'afghanistan': 30.55, 'albania': 2.81,  
'algeria': 39.21, 'sealand': 2.7e-05}
```

```
"sealand" in world
```

```
True
```

# Dictionary

```
world["sealand"] = 0.000028  
world
```

```
{'afghanistan': 30.55, 'albania': 2.81,  
 'algeria': 39.21, 'sealand': 2.8e-05}
```

```
del(world["sealand"])  
world
```

```
{'afghanistan': 30.55, 'albania': 2.81, 'algeria': 39.21}
```

# List vs. Dictionary

# List vs. Dictionary

# List vs. Dictionary

List	Dictionary
Select, update, and remove with <code>[]</code>	Select, update, and remove with <code>[]</code>

# List vs. Dictionary

List	Dictionary
Select, update, and remove with <code>[]</code>	Select, update, and remove with <code>[]</code>

# List vs. Dictionary

List	Dictionary
Select, update, and remove with <code>[]</code>	Select, update, and remove with <code>[]</code>
Indexed by range of numbers	

# List vs. Dictionary

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

# List vs. Dictionary

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	

# List vs. Dictionary

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

# **Let's practice!**

## **INTERMEDIATE PYTHON**

# Pandas, Part 1

INTERMEDIATE PYTHON



**Hugo Bowne-Anderson**  
Data Scientist at DataCamp

# 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

# 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

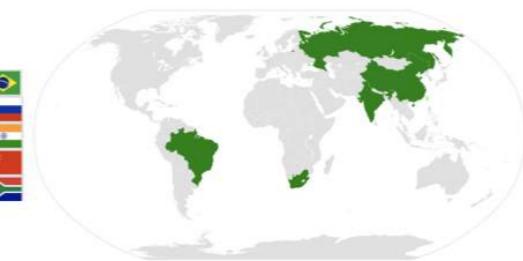
row = observations  
column = variable

# 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

# Datasets in Python

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

float      float

# Datasets in Python

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!
  - High level data manipulation tool
  - Wes McKinney
  - Built on NumPy
  - DataFrame

# DataFrame

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

# DataFrame from Dictionary

```
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)

```
import pandas as pd  
brics = pd.DataFrame(dict)
```

# DataFrame from Dictionary (2)

```
brics
```

```
    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
```

```
brics.index = ["BR", "RU", "IN", "CH", "SA"]
brics
```

```
    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
```

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

```
  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
```

# DataFrame from CSV file

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

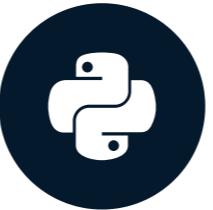
	country	population	area	capital
BR	Brazil	200	8515767	Brasilia
RU	Russia	144	17098242	Moscow
IN	India	1252	3287590	New Delhi
CH	China	1357	9596961	Beijing
SA	South Africa	55	1221037	Pretoria

# **Let's practice!**

## **INTERMEDIATE PYTHON**

# Pandas, Part 2

INTERMEDIATE PYTHON



**Hugo Bowne-Anderson**  
Data Scientist at DataCamp

# brics

```
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

# 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
```

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

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

# 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
```

- 1D labelled array

# 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
```

# 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
```

# 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
```

# 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
```

# Row Access []

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

```
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
```

# 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
```

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

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

- Row as pandas Series

# 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

# 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
```

# 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
```

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

```
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
```

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

```
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

```
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
```

```
brics.iloc[[1]]
```

```
country  capital   area  population
RU      Russia    Moscow  17.1       143.5
```

# Row Access iloc

```
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
```

# Row Access iloc

```
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.iloc[[1,2,3]]
```

```
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 iloc

```
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
```

# Row & Column iloc

```
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.iloc[[1,2,3], [0, 1]]
```

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

# Row & Column iloc

```
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
```

# Row & Column iloc

```
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.iloc[:, [0,1]]
```

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

# **Let's practice!**

## **INTERMEDIATE PYTHON**

# Comparison Operators

INTERMEDIATE PYTHON



**Hugo Bowne-Anderson**  
Data Scientist at DataCamp

# NumPy recap

```
# Code from Intro to Python for Data Science, Chapter 4
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
bmi
```

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

```
bmi > 23
```

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

```
bmi[bmi > 23]
```

```
array([ 24.747])
```

- Comparison operators: how Python values relate

# Numeric comparisons

```
2 < 3
```

```
True
```

```
2 == 3
```

```
False
```

```
2 <= 3
```

```
True
```

```
3 <= 3
```

```
True
```

```
x = 2  
y = 3  
x < y
```

```
True
```

# Other comparisons

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

```
True
```

```
3 < "chris"
```

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

```
3 < 4.1
```

```
True
```

# Other comparisons

```
bmi
```

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

```
bmi > 23
```

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

# Comparators

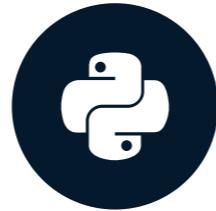
Comparator	Meaning
<	Strictly less than
<=	Less than or equal
>	Strictly greater than
>=	Greater than or equal
==	Equal
!=	Not equal

# **Let's practice!**

## **INTERMEDIATE PYTHON**

# Boolean Operators

INTERMEDIATE PYTHON



**Hugo Bowne-Anderson**  
Data Scientist at DataCamp

# Boolean Operators

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

# 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

# or

True **or** True

True

False **or** True

True

True **or** False

True

False **or** False

False

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

True

# not

**not** True

False

**not** False

True

# 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()
```

# 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)]
```

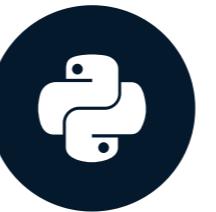
```
array([21.852, 21.75, 21.441])
```

# **Let's practice!**

## **INTERMEDIATE PYTHON**

# **if, elif, else**

**INTERMEDIATE PYTHON**



**Hugo Bowne-Anderson**  
Data Scientist at DataCamp

# 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 :      # True  
    print("z is even")
```

z is even

# if

```
if condition :  
    expression
```

- expression not part of if

control.py

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

z is even

# if

```
if condition :  
    expression
```

## control.py

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

```
checking 4  
z is even
```

# if

```
if condition :  
    expression
```

control.py

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

# 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

# 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
```

# 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
```

# **Let's practice!**

## **INTERMEDIATE PYTHON**

# Filtering pandas DataFrames

INTERMEDIATE PYTHON



**Hugo Bowne-Anderson**  
Data Scientist at DataCamp

# brics

```
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

# Goal

	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 km<sup>2</sup>
- 3 steps
  - Select the area column
  - Do comparison on area column
  - Use result to select countries

# 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]
```

# 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
```

# Step 3: Subset DF

```
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
```

# 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
```

# 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
```

# **Let's practice!**

## **INTERMEDIATE PYTHON**

# while loop

## INTERMEDIATE PYTHON



**Hugo Bowne-Anderson**  
Data Scientist at DataCamp

# if-elif-else

control.py

- Goes through construct only once!

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

... # 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

```
while condition :  
    expression
```

## while\_loop.py

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

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

# While

```
while condition :  
    expression
```

while\_loop.py

```
error = 50.0  
#      50  
while error > 1:      # True  
    error = error / 4  
    print(error)
```

12.5

# While

```
while condition :  
    expression
```

while\_loop.py

```
error = 50.0  
#      12.5  
while error > 1:      # True  
    error = error / 4  
    print(error)
```

```
12.5  
3.125
```

# While

```
while condition :  
    expression
```

while\_loop.py

```
error = 50.0  
#      3.125  
while error > 1:      # True  
    error = error / 4  
    print(error)
```

```
12.5  
3.125  
0.78125
```

# While

```
while condition :  
    expression
```

while\_loop.py

```
error = 50.0  
#      0.78125  
while error > 1:      # False  
    error = error / 4  
    print(error)
```

```
12.5  
3.125  
0.78125
```

# While

```
while condition :  
    expression
```

## while\_loop.py

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

- DataCamp: session disconnected
  - Local system: Control + C

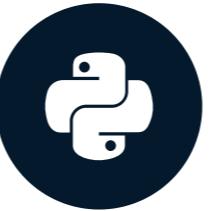
50 50 50 50 50 50 50 50

# **Let's practice!**

## **INTERMEDIATE PYTHON**

# for loop

## INTERMEDIATE PYTHON



**Hugo Bowne-Anderson**  
Data Scientist at DataCamp

# 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)
```

```
[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])
```

```
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
```

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
```

```
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)
```

```
1.73  
1.68  
1.71  
1.89
```

- No access to indexes

# for loop

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

family.py

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

- ???

```
index 0: 1.73  
index 1: 1.68  
index 2: 1.71  
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))
```

```
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())
```

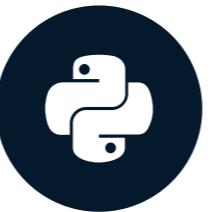
F  
A  
M  
I  
L  
Y

# **Let's practice!**

## **INTERMEDIATE PYTHON**

# Loop Data Structures Part 1

INTERMEDIATE PYTHON



**Hugo Bowne-Anderson**  
Data Scientist at DataCamp

# 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))
```

```
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))
```

```
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 }  
  
for k, v in world.items() :  
    print(k + " -- " + str(v))
```

```
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)
```

```
21.852  
20.975  
21.750  
24.747  
21.441
```

# 2D NumPy Arrays

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])
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]
```

# 2D NumPy Arrays

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])
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
...
```

# Recap

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

# **Let's practice!**

## **INTERMEDIATE PYTHON**

# Loop Data Structures Part 2

INTERMEDIATE PYTHON



**Hugo Bowne-Anderson**  
Data Scientist at DataCamp

# 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

## dfloop.py

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

# for, first try

dfloop.py

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

```
country  
capital  
area  
population
```

# iterrows

## 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)
```

```
BR  
country      Brazil  
capital      Brasilia  
area         8.516  
population   200.4  
Name: BR, dtype: object  
...  
RU  
country      Russia  
capital      Moscow  
area         17.1  
population   143.5  
Name: RU, dtype: object  
IN ...
```

# Selective print

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"])
```

BR: Brasilia

RU: Moscow

IN: New Delhi

CH: Beijing

SA: Pretoria

# Add column

dfloop.py

```
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

# apply

dfloop.py

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

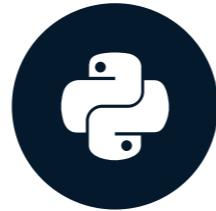
	country	capital	area	population	name_length
BR	Brazil	Brasilia	8.516	200.40	6
RU	Russia	Moscow	17.100	143.50	6
IN	India	New Delhi	3.286	1252.00	5
CH	China	Beijing	9.597	1357.00	5
SA	South Africa	Pretoria	1.221	52.98	12

# **Let's practice!**

## **INTERMEDIATE PYTHON**

# Random Numbers

INTERMEDIATE PYTHON



**Hugo Bowne-Anderson**  
Data Scientist at DataCamp

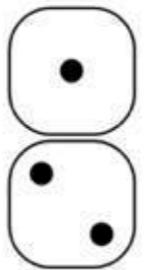


100 x





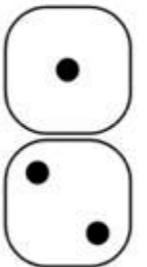
100 x



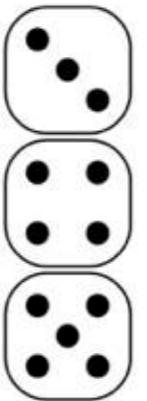
-1



100 x



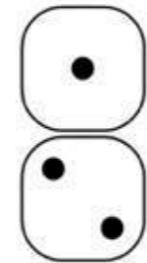
-1



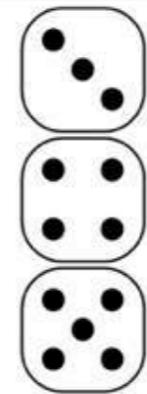
+1



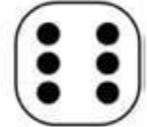
100 x



-1



+1



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

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

```
import numpy as np  
np.random.rand()      # Pseudo-random numbers
```

```
0.9535543896720104    # Mathematical formula
```

```
np.random.seed(123)    # Starting from a seed  
np.random.rand()
```

```
0.6964691855978616
```

```
np.random.rand()
```

```
0.28613933495037946
```

# Random generators

```
np.random.seed(123)  
np.random.rand()
```

```
0.696469185597861 # Same seed: same random numbers!
```

```
np.random.rand() # Ensures "reproducibility"
```

```
0.28613933495037946
```

# Coin toss

game.py

```
import numpy as np
np.random.seed(123)
coin = np.random.randint(0,2) # Randomly generate 0 or 1
print(coin)
```

0

# Coin toss

game.py

```
import numpy as np
np.random.seed(123)
coin = np.random.randint(0,2) # Randomly generate 0 or 1
print(coin)
if coin == 0:
    print("heads")
else:
    print("tails")
```

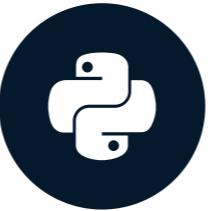
```
0
heads
```

# **Let's practice!**

## **INTERMEDIATE PYTHON**

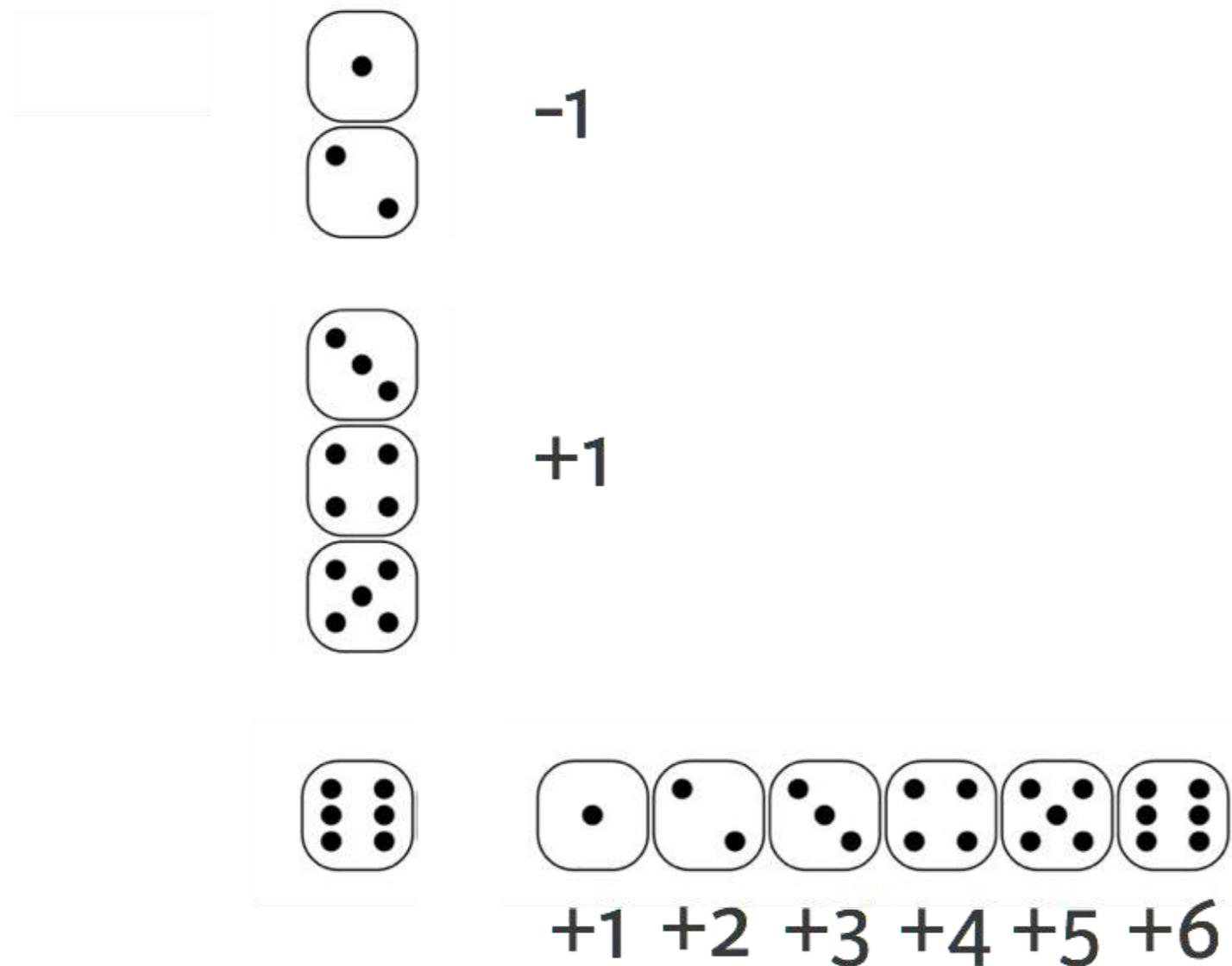
# Random Walk

INTERMEDIATE PYTHON

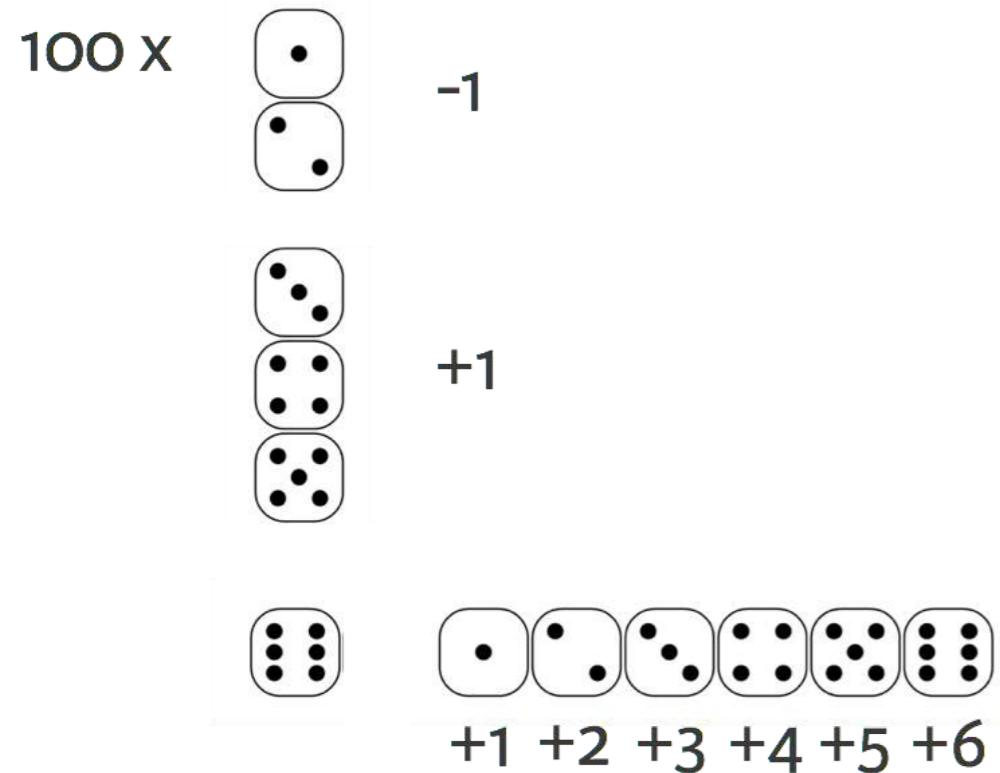


**Hugo Bowne-Anderson**  
Data Scientist at DataCamp

# Random Step



# Random Walk



## 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)
```

```
['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)
```

```
[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

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

# **Let's practice!**

## **INTERMEDIATE PYTHON**

# Distribution

## INTERMEDIATE PYTHON

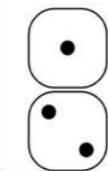


**Hugo Bowne-Anderson**  
Data Scientist at DataCamp

# Distribution

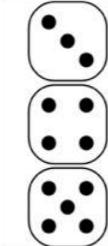


100 x



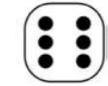
-1

Each random walk has an end point



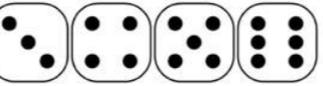
+1

Distribution!

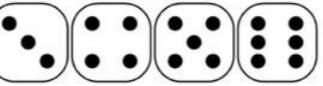


+1 +2 +3 +4 +5 +6

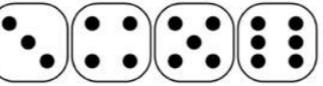
Calculate chances!



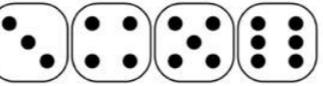
+1 +2 +3 +4 +5 +6



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



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



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

# 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[-1] + coin)
    final_tails.append(tails[-1])
print(final_tails)
```

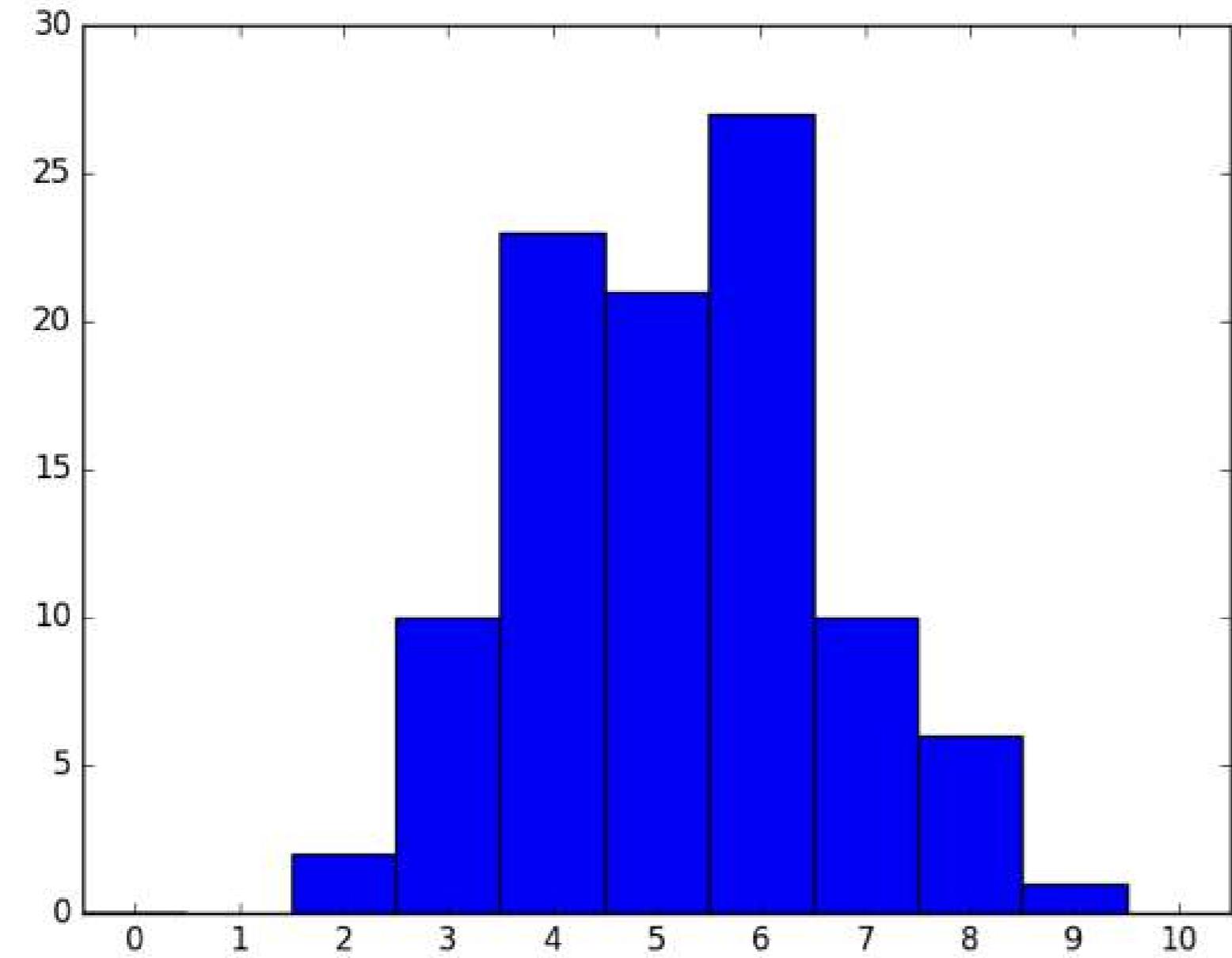
```
[3, 6, 4, 5, 4, 5, 3, 5, 4, 6, 6, 8, 6, 4, 7, 5, 7, 4, 3, 3, ..., 4]
```

# 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[-1] + coin)
    final_tails.append(tails[-1])
plt.hist(final_tails, bins = 10)
plt.show()
```

# Histogram, 100 runs

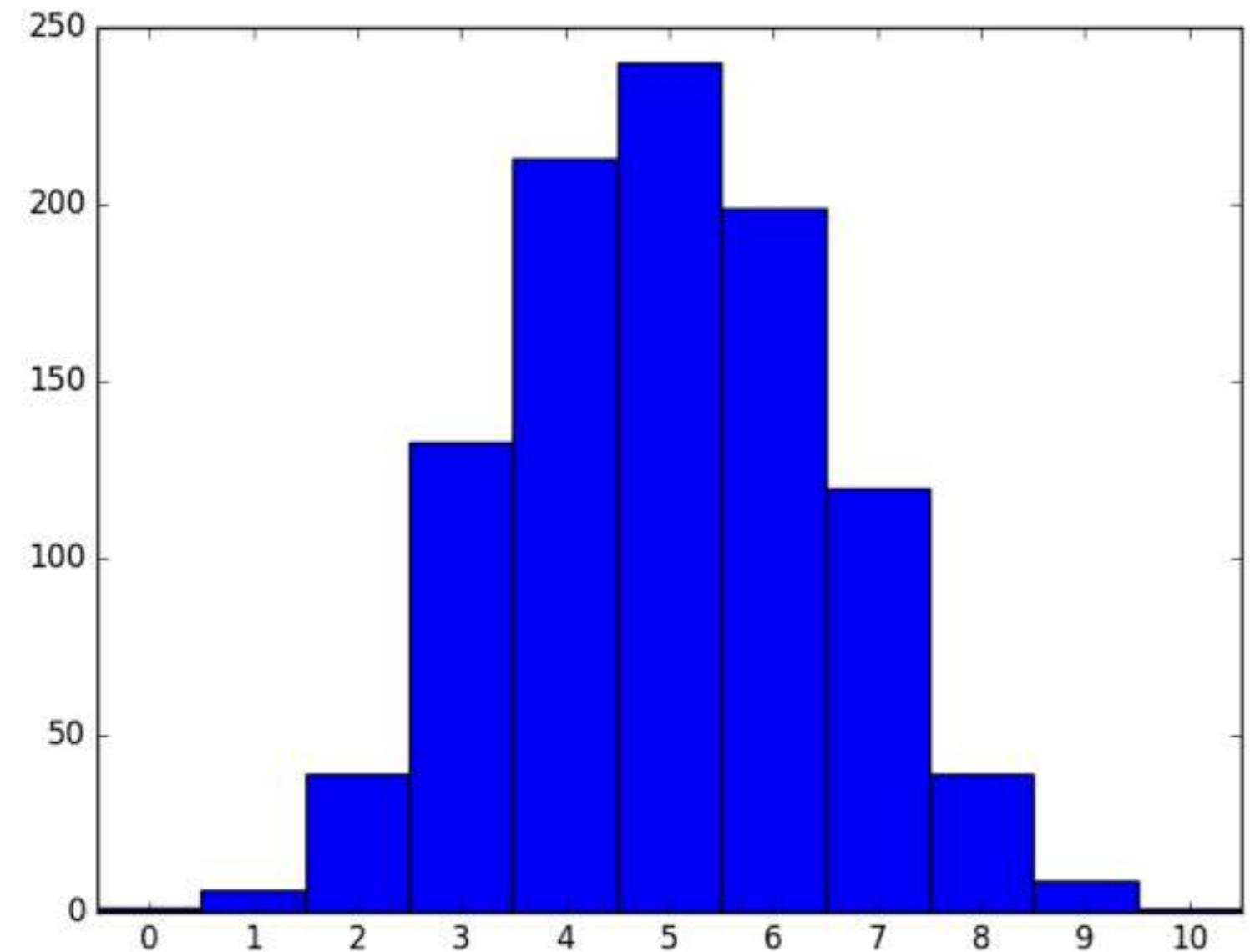


# Histogram, 1,000 runs

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[-1] + coin)
    final_tails.append(tails[-1])
plt.hist(final_tails, bins = 10)
plt.show()
```

# Histogram, 1,000 runs

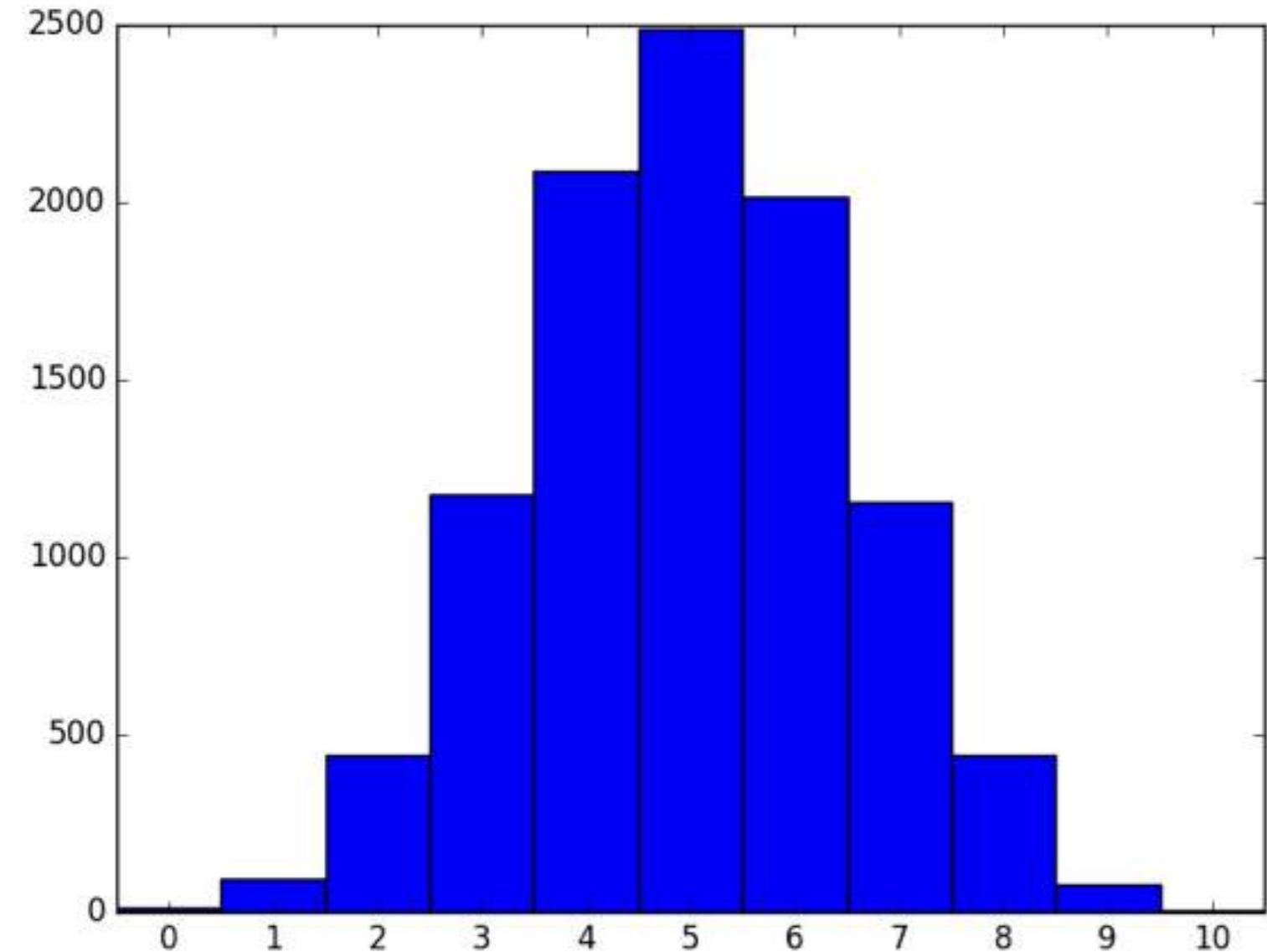


# Histogram, 10,000 runs

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[-1] + coin)
    final_tails.append(tails[-1])
plt.hist(final_tails, bins = 10)
plt.show()
```

# Histogram, 10,000 runs



# **Let's practice!**

## **INTERMEDIATE PYTHON**

# Python For Data Science

## python™ Basics Cheat Sheet

Learn Python Basics online at [www.DataCamp.com](http://www.DataCamp.com)

## > Variables and Data Types

### Variable Assignment

```
>>> x=5
>>> x
5
```

### Calculations With Variables

```
>>> x+2 #Sum of two variables
7
>>> x-2 #Subtraction of two variables
3
>>> x*2 #Multiplication of two variables
10
>>> x**2 #Exponentiation of a variable
25
>>> x%2 #Remainder of a variable
1
>>> x/float(2) #Division of a variable
2.5
```

### Types and Type Conversion

```
str()
'5', '3.45', 'True' #Variables to strings

int()
5, 3, 1 #Variables to integers

float()
5.0, 1.0 #Variables to floats

bool()
True, True, True #Variables to booleans
```

## > Libraries

pandas	NumPy	matplotlib	learn
Data analysis	Scientific computing	2D plotting	Machine learning

### Import Libraries

```
>>> import numpy
>>> import numpy as np
```

### Selective import

```
>>> from math import pi
```

## > Strings

```
>>> my_string = 'thisStringIsAwesome'
>>> my_string
'thisStringIsAwesome'
```

### String Operations

```
>>> my_string * 2
'thisStringIsAwesomethisStringIsAwesome'
>>> my_string + 'Innit'
'thisStringIsAwesomeInnit'
>>> 'm' in my_string
True
```

### String Indexing

Index starts at 0

```
>>> my_string[3]
>>> my_string[4:9]
```

### String Methods

```
>>> my_string.upper() #String to uppercase
>>> my_string.lower() #String to lowercase
>>> my_string.count('w') #Count String elements
>>> my_string.replace('e', 'i') #Replace String elements
>>> my_string.strip() #Strip whitespaces
```

## > NumPy Arrays

Also see Lists

```
>>> my_list = [1, 2, 3, 4]
>>> my_array = np.array(my_list)
>>> my_2darray = np.array([[1,2,3],[4,5,6]])
```

### Selecting Numpy Array Elements

Index starts at 0

```
Subset
>>> my_array[1] #Select item at index 1
2

Slice
>>> my_array[0:2] #Select items at index 0 and 1
array([1, 2])

Subset 2D Numpy arrays
>>> my_2darray[:,0] #my_2darray[rows, columns]
array([1, 4])
```

### Numpy Array Operations

```
>>> my_array > 3
array([False, False, False, True], dtype=bool)
>>> my_array * 2
array([2, 4, 6, 8])
>>> my_array + np.array([5, 6, 7, 8])
array([6, 8, 10, 12])
```

### Numpy Array Functions

```
>>> my_array.shape #Get the dimensions of the array
>>> np.append(other_array) #Append items to an array
>>> np.insert(my_array, 1, 5) #Insert items in an array
>>> np.delete(my_array,[1]) #Delete items in an array
>>> np.mean(my_array) #Mean of the array
>>> np.median(my_array) #Median of the array
>>> my_array.correlcoef() #Correlation coefficient
>>> np.std(my_array) #Standard deviation
```

## > Lists

Also see NumPy Arrays

```
>>> a = 'is'
>>> b = 'nice'
>>> my_list = ['my', 'list', a, b]
>>> my_list2 = [[4,5,6,7], [3,4,5,6]]
```

### Selecting List Elements

Index starts at 0

Subset

```
>>> my_list[1] #Select item at index 1
>>> my_list[-3] #Select 3rd last item
```

Slice

```
>>> my_list[1:3] #Select items at index 1 and 2
>>> my_list[1:] #Select items after index 0
>>> my_list[:3] #Select items before index 3
>>> my_list[:] #Copy my_list
```

Subset Lists of Lists

```
>>> my_list2[1][0] #my_list[list][itemOfList]
>>> my_list2[1][:2]
```

### List Operations

```
>>> my_list + my_list
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list * 2
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list2 > 4
True
```

### List Methods

```
>>> my_list.index(a) #Get the index of an item
>>> my_list.count(a) #Count an item
>>> my_list.append('!) #Append an item at a time
>>> my_list.remove('!) #Remove an item
>>> del(my_list[0:1]) #Remove an item
>>> my_list.reverse() #Reverse the list
>>> my_list.extend('!) #Append an item
>>> my_list.pop(-1) #Remove an item
>>> my_list.insert(0,'!) #Insert an item
>>> my_list.sort() #Sort the list
```

## > Python IDEs (Integrated Development Environment)

ANACONDA.

Leading open data science platform powered by Python

SPYDER

Free IDE that is included with Anaconda

jupyter

Create and share documents with live code

## > Asking For Help

```
>>> help(str)
```

Learn Data Skills Online at  
[www.DataCamp.com](http://www.DataCamp.com)