

Hello Python!

INTRODUCTION TO PYTHON



Hugo Bowne-Anderson
Data Scientist at DataCamp

How you will learn

The screenshot shows a Python exercise interface. At the top, there's a navigation bar with a logo, 'Learn / Courses / Introduction to Python', and course outline buttons. Below the navigation is a toolbar with a back arrow, forward arrow, and light mode switch.

Exercise: Python as a calculator

Instructions: 500 XP

script.py

```
1 # Addition
2 print(5 + 5)
3
4 # Subtraction
5 print(5 - 5)
6
7 # Multiplication
8 print(5 * 5)
9
10 # Division
11
```

IPython Shell: In [1]:

Below the code editor, there are buttons for 'Run Code' and 'Submit Answer'. A 'Take Hint (-30 XP)' button is located at the bottom left of the exercise area.

Python



- General purpose: build anything
- Open source! Free!
- Python packages, also for data science
 - Many applications and fields

IPython Shell

Execute Python commands

The screenshot shows a Python exercise interface. On the left, there's a sidebar with navigation links: 'Learn / Courses / Introduction to Python' and a 'Course Outline' button. Below that is a section titled 'Exercise' with a sub-section 'Python as a calculator'. It contains text about Python being suited for calculations and some examples. A 'Light Mode' toggle is also present. On the right, the main area has a dark theme. It shows a code editor with a file named 'script.py' containing the following code:

```
script.py
1 # Addition
2
3
4 # Subtraction
5
6
7 # Multiplication
8
9
10 # Division
11
```

Below the code editor are three buttons: a circular arrow icon, 'Run Code', and 'Submit Answer'. At the bottom, there's an 'IPython Shell' tab and an input field labeled 'In [1]:'. To the left of the main area, there's a sidebar with 'Instructions' (100 XP) and a list of tasks:

- Print the sum of `5 + 5`.
- Print the result of subtracting `5` from `5`.
- Multiply `3` by `5`.
- Divide `10` by `2`.

At the bottom left of the sidebar is a 'Take Hint (-30 XP)' button.

IPython Shell

Execute Python commands

The screenshot shows a Python exercise interface. On the left, there's a sidebar with navigation links: 'Learn / Courses / Introduction to Python' and a 'Course Outline' button. Below that is a section titled 'Exercise' with a sub-section 'Python as a calculator'. It contains text about Python being suited for calculations and some examples. A 'Instructions' section offers 100 XP and lists four tasks: 'Print the sum of 5 + 5.', 'Print the result of subtracting 5 from 5.', 'Multiply 3 by 5.', and 'Divide 10 by 2.'. A 'Take Hint (-30 XP)' button is also present. On the right, the main area is titled 'script.py' and contains the following code:

```
script.py
1 # Addition
2
3
4 # Subtraction
5
6
7 # Multiplication
8
9
10 # Division
11
```

Below the code editor are three buttons: a refresh icon, 'Run Code', and 'Submit Answer'. At the bottom, there's an 'IPython Shell' window with the prompt 'In [1]:' followed by a blank input field.

IPython Shell

The screenshot shows a Python exercise interface. At the top left, there's a navigation bar with 'Learn / Courses / Introduction to Python'. Below it, a sidebar on the left contains sections for 'Exercise' (selected), 'Python as a calculator' (with a brief description), 'Instructions' (containing four tasks with placeholder values like 5, 3, etc.), and a 'Take Hint (-50 XP)' button. The main area features an 'IPython Shell' interface with a dark theme. It has a code editor window titled 'script.py' containing the number '1'. Below the editor are three buttons: a blue 'Run Code' button and a green 'Submit Answer' button. At the bottom of the shell window, it says 'In [1]:'. The overall interface is clean and modern, designed for learning Python basics.

Python Script

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

The screenshot shows a Python script editor interface. The main area displays a code file named `script.py` with the following content:

```
1 # Addition
2
3 # Subtraction
4
5 # Multiplication
6
7 # Division
8
9
10
11
```

To the left of the code editor, there is a sidebar with the following sections:

- Exercise**:
 - Python as a calculator**: A brief introduction stating "Python is perfectly suited to do basic calculations. It can do addition, subtraction, multiplication and division."
 - Instructions**:
 - Print the sum of `5 + 5`.
 - Print the result of subtracting `5` from `5`.
 - Multiply `3` by `5`.
 - Divide `10` by `2`.
 - Take Hint (-30 XP)**

At the bottom of the code editor, there are three buttons: `Run Code` and `Submit Answer`. Below the code editor, there is a small `IPython Shell` window showing the prompt `In [1]:`.

Python Script

The screenshot shows a Python script editor interface. On the left, there's a sidebar with navigation links like 'Learn / Courses / Introduction to Python' and a 'Course Outline'. Below that is an 'Exercise' section titled 'Python as a calculator' with instructions about basic calculations. A 'Instructions' section lists four tasks with placeholder values (4, 5, 3, 2) for practice. A 'Take Hint (-30 XP)' button is also present. The main area has a dark theme with a 'script.py' file open. The code '1 4' is typed in the editor, with '4' highlighted in red, indicating a syntax error or a placeholder. Below the editor are three buttons: 'Run Code', 'Submit Answer', and a refresh icon. At the bottom is an 'IPython Shell' window with the prompt 'In [1]:'.

Python Script

The screenshot shows a Python script editor interface. On the left, there's a sidebar with navigation links like 'Learn / Courses / Introduction to Python' and a 'Course Outline'. Below that is the 'Exercise' section, which has a title 'Python as a calculator' and a brief description: 'Python is perfectly suited to do basic calculations. It can do addition, subtraction, multiplication and division.' It also includes some sample code and a call to action: 'Now it's your turn to practice!'. The main area is a code editor titled 'script.py' containing the number '1'. At the bottom of the editor are three buttons: 'Run Code', 'Submit Answer', and a 'Take Hint (-30 XP)' button. Below the editor is an 'IPython Shell' window with the prompt 'In [1]:'. The entire interface is set against a dark background.

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

DataCamp Interface

The screenshot shows the DataCamp Python exercise interface. At the top, there's a navigation bar with a logo, 'Learn / Courses / Introduction to Python', and course outline/back/forward buttons. To the right is a light/dark mode switch.

The main area is divided into sections:

- Exercise:** A title 'Python as a calculator' is followed by text explaining Python's suitability for basic calculations. It mentions addition, subtraction, multiplication, and division, and notes that the code script provides examples.
- Instructions:** A section titled 'Instructions' offers 100 XP. It lists four tasks:
 - Print the sum of `5 + 5`.
 - Print the result of subtracting `5` from `5`.
 - Multiply `3` by `5`.
 - Divide `10` by `2`.
- script.py:** A code editor window containing the following Python code:

```
1 # Addition
2
3
4 # Subtraction
5
6
7 # Multiplication
8
9
10 # Division
11
```
- IPython Shell:** A terminal-like interface labeled 'IPython Shell' with the prompt 'In [1]:'.
- Buttons:** At the bottom are 'Run Code' and 'Submit Answer' buttons.

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]
```

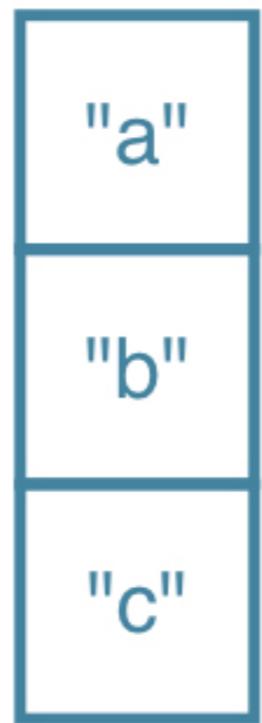
```
del fam[2]
```

```
fam
```

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

Behind the scenes (1)

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



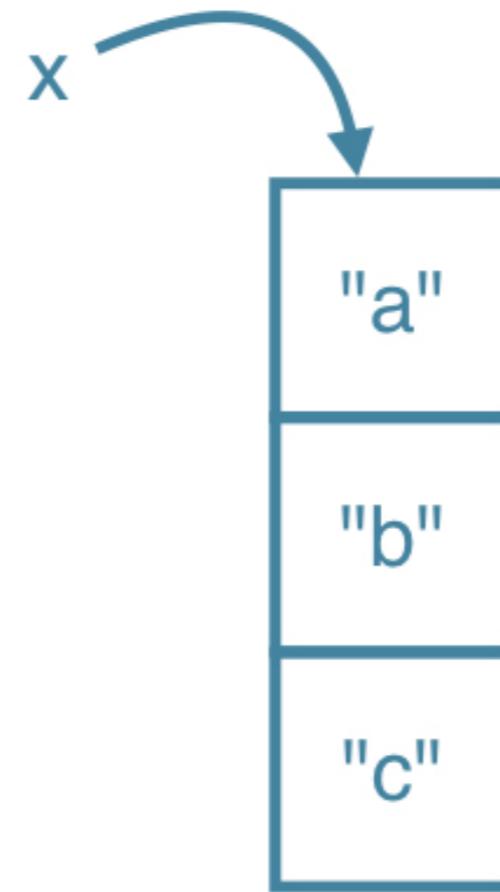
Behind the scenes (1)

```
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
```

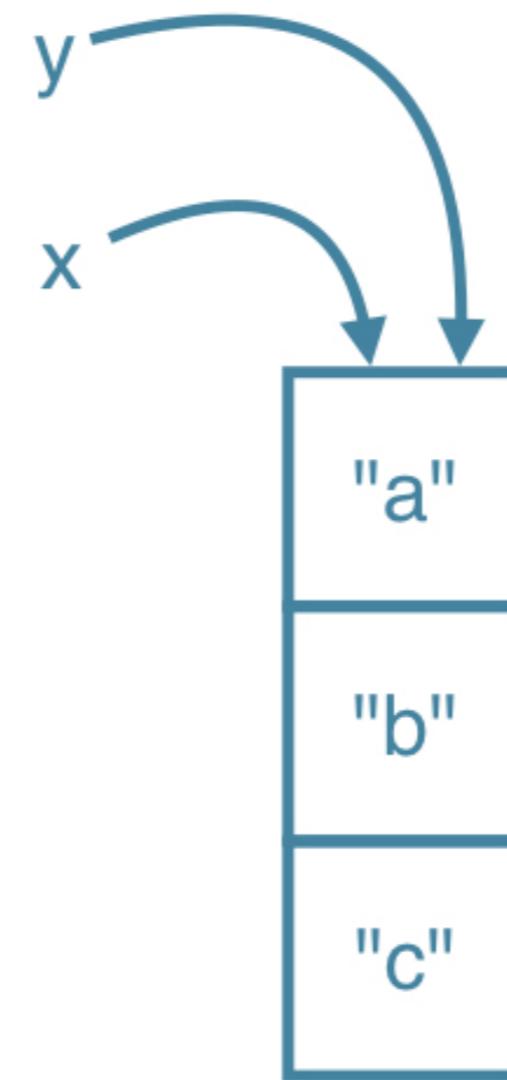
```
y[1] = "z"
```

```
y
```

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

```
x
```

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



Behind the scenes (1)

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

```
y = x
```

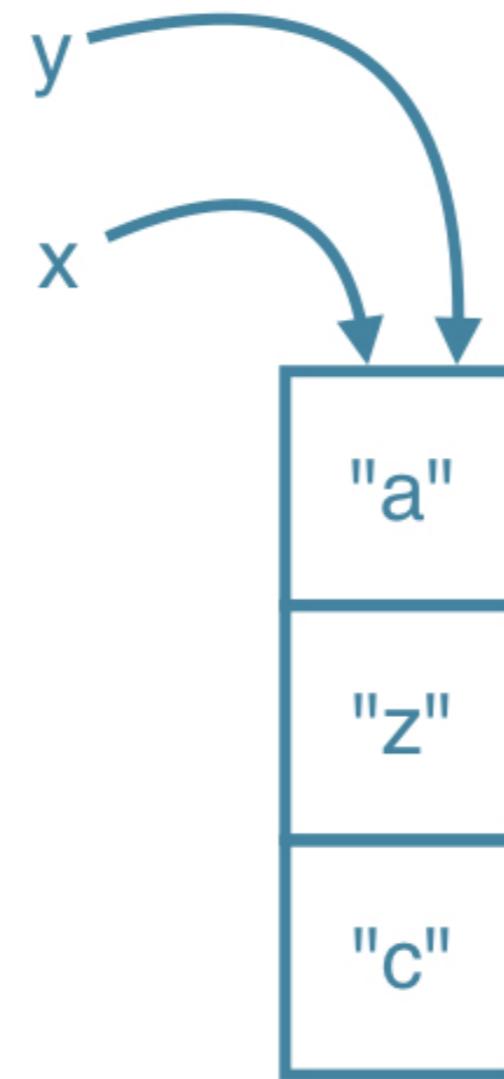
```
y[1] = "z"
```

```
y
```

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

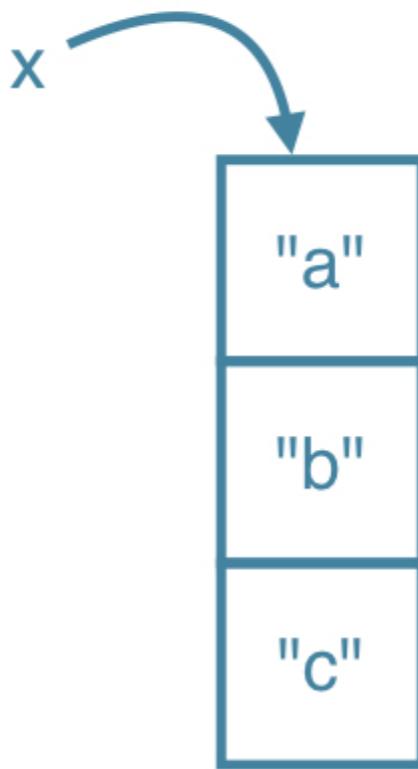
```
x
```

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



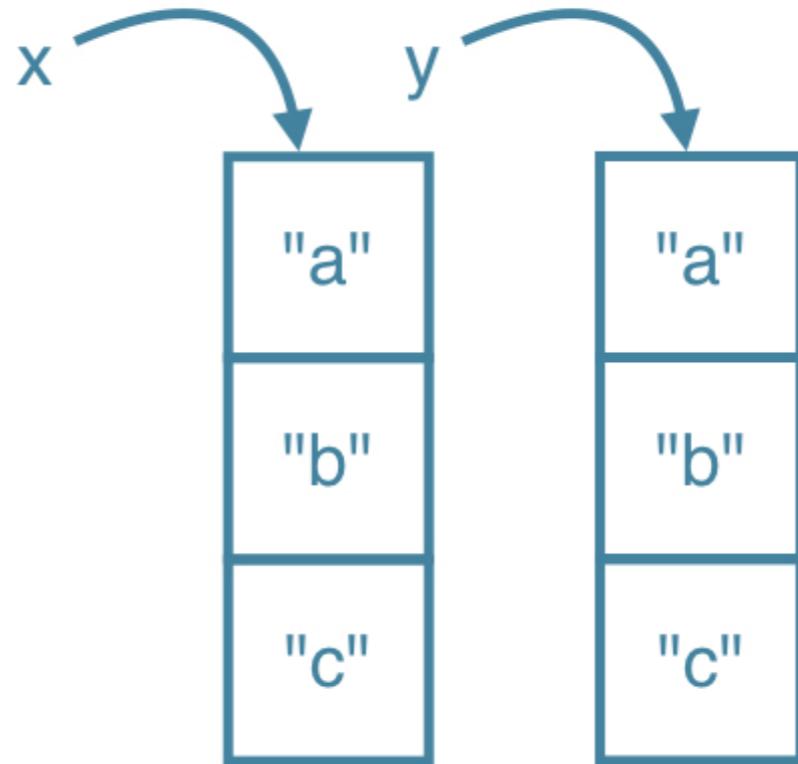
Behind the scenes (2)

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



Behind the scenes (2)

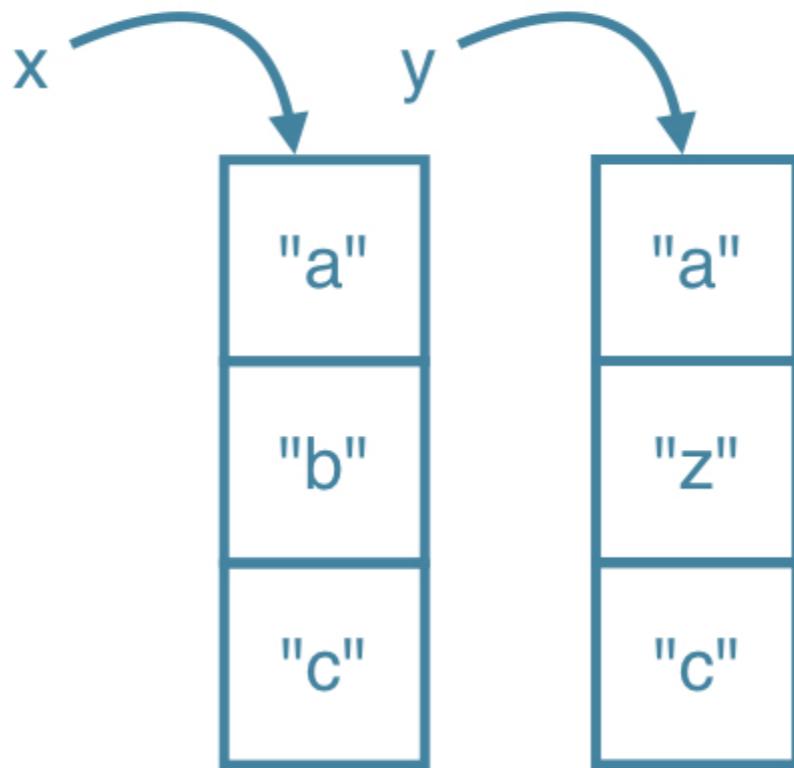
```
x = ["a", "b", "c"]  
y = list(x)  
y = x[:]
```



Behind the scenes (2)

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

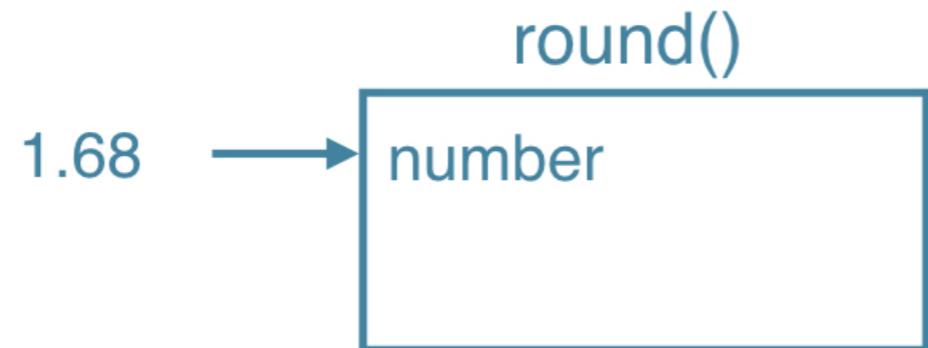
```
round(number, ndigits=None)
```

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:

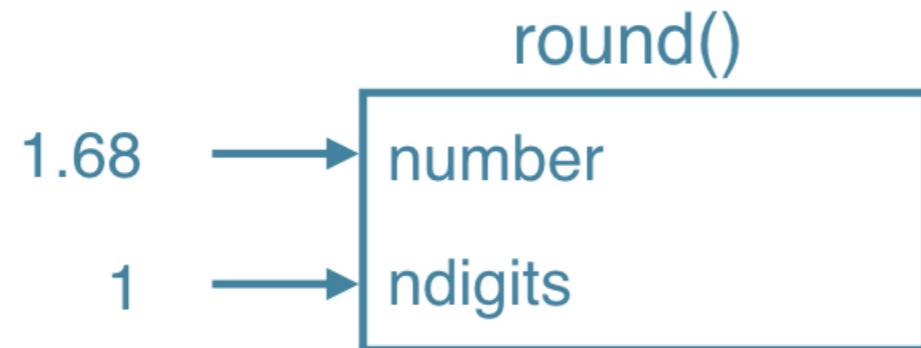
```
round(number, ndigits=None)
```

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:

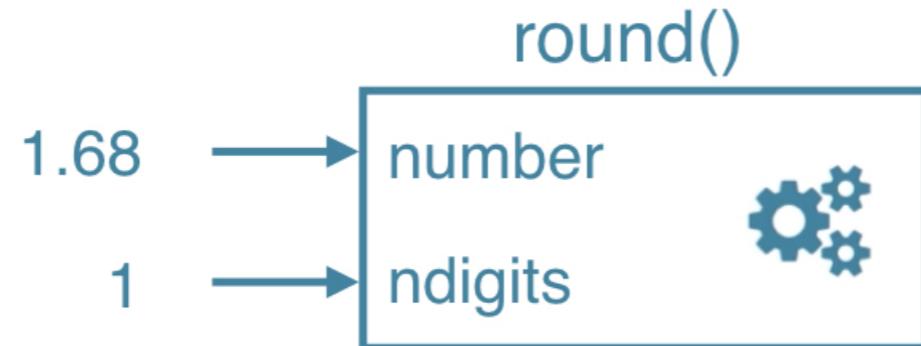
```
round(number, ndigits=None)
```

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:

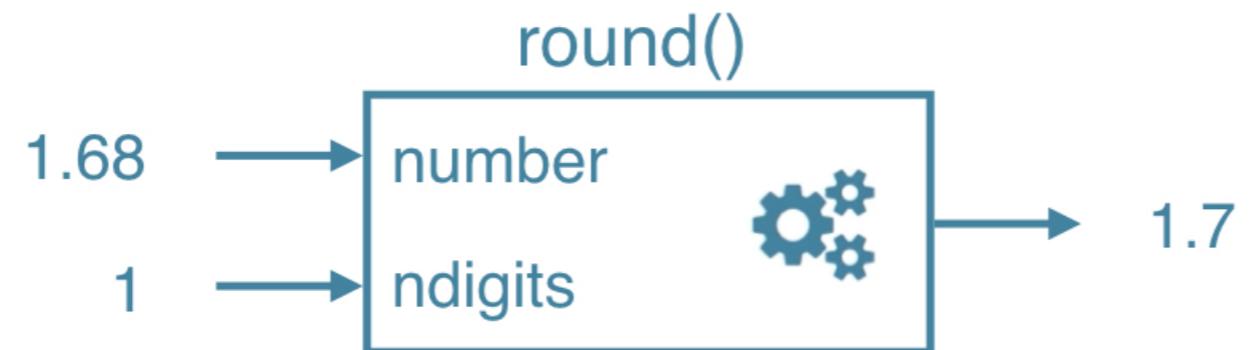
```
round(number, ndigits=None)
```

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:

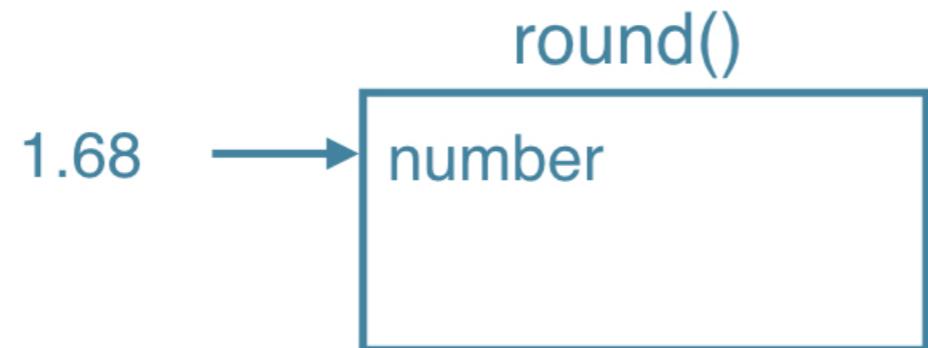
```
round(number, ndigits=None)
```

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:

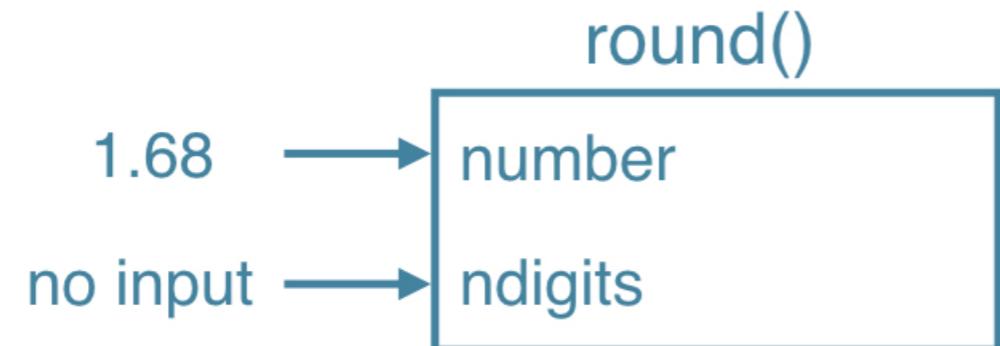
```
round(number, ndigits=None)
```

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:

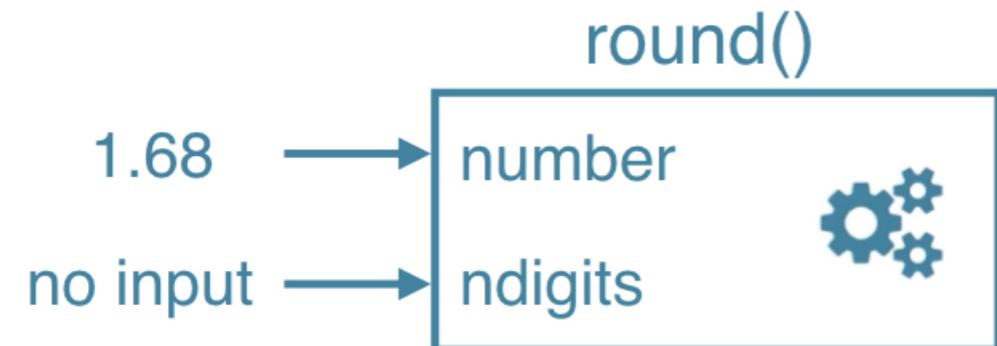
```
round(number, ndigits=None)
```

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:

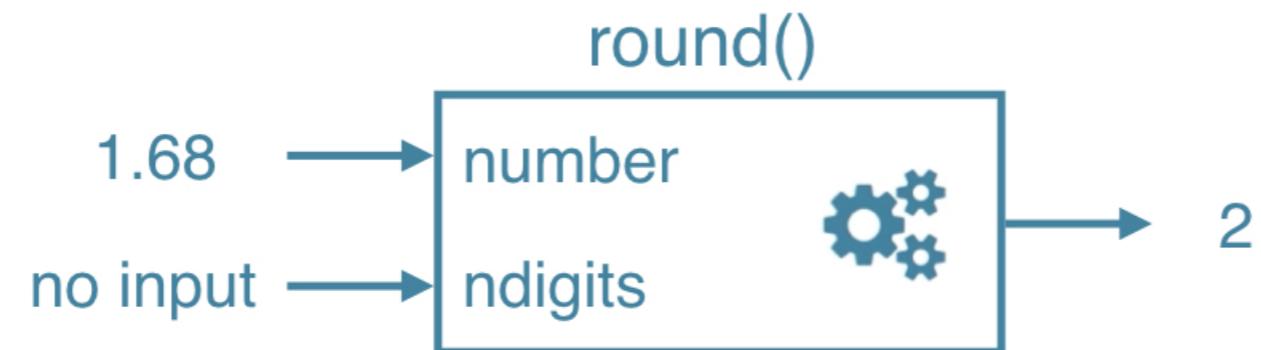
```
round(number, ndigits=None)
```

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

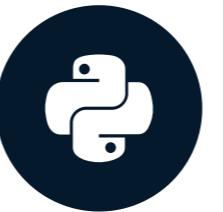
- How to know?
- Standard task -> probably function exists!
- The internet is your friend

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