

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] → max() → 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
```

```
round(...)  
round(number[, ndigits]) -> number
```

Round a number to a given precision in decimal digits (default 0 digits).

This returns an int when called with one argument,
otherwise the same type as the number.
ndigits may be negative.

round()

```
help(round)
```

```
round(...)  
round(number[, ndigits]) -> number
```

Round a number to a given precision in decimal digits (default 0 digits).
This returns an int when called with one argument,
otherwise the same type as the number.
ndigits may be negative.

round()



round()

```
help(round)
```

```
round(...)  
round(number[, ndigits]) -> number
```

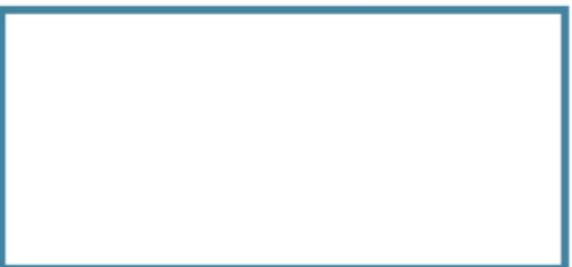
Round a number to a given precision in decimal digits (default 0 digits).

This returns an int when called with one argument,
otherwise the same type as the number.

ndigits may be negative.

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

```
round()
```



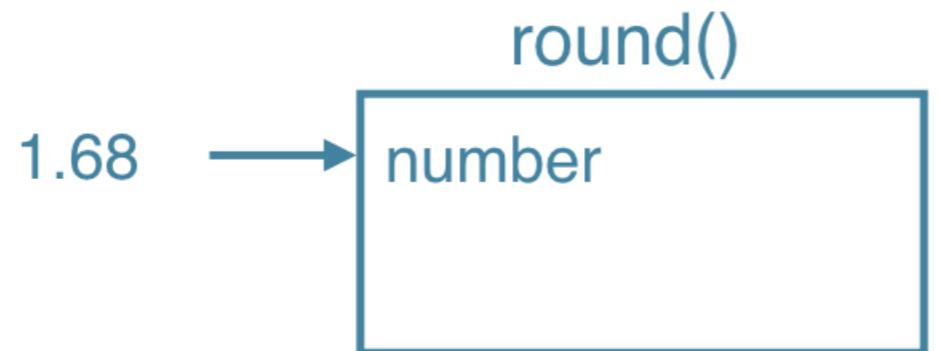
round()

```
help(round)
```

```
round(...)  
round(number[, ndigits]) -> number
```

Round a number to a given precision in decimal digits (default 0 digits).
This returns an int when called with one argument,
otherwise the same type as the number.
ndigits may be negative.

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



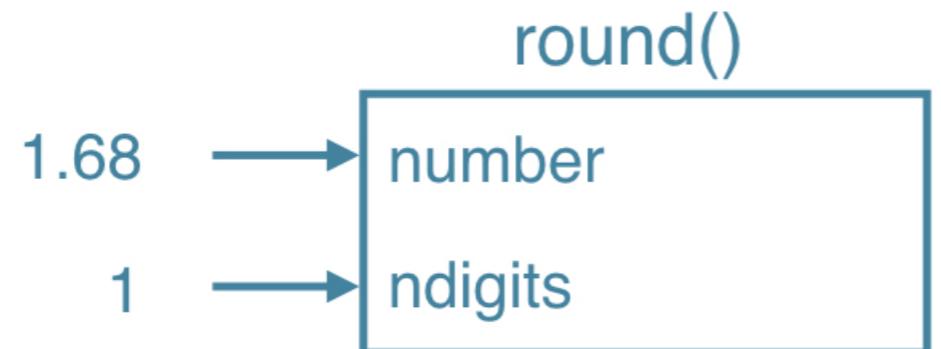
round()

```
help(round)
```

```
round(...)  
round(number[, ndigits]) -> number
```

Round a number to a given precision in decimal digits (default 0 digits).
This returns an int when called with one argument,
otherwise the same type as the number.
ndigits may be negative.

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



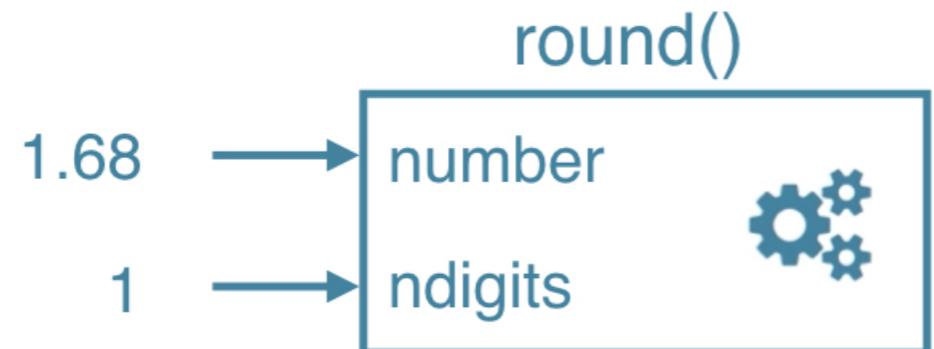
round()

```
help(round)
```

```
round(...)  
round(number[, ndigits]) -> number
```

Round a number to a given precision in decimal digits (default 0 digits).
This returns an int when called with one argument,
otherwise the same type as the number.
ndigits may be negative.

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



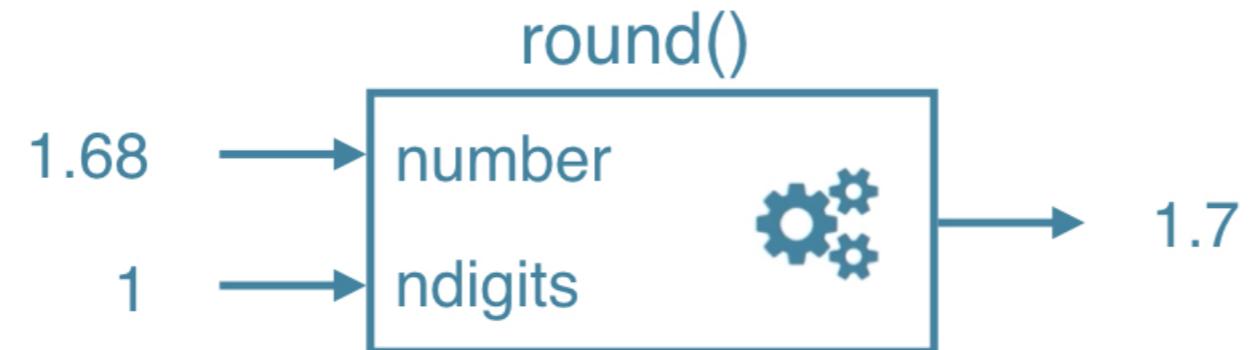
round()

```
help(round)
```

```
round(...)  
round(number[, ndigits]) -> number
```

Round a number to a given precision in decimal digits (default 0 digits).
This returns an int when called with one argument,
otherwise the same type as the number.
ndigits may be negative.

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



round()

```
help(round)
```

```
round(...)  
round(number[, ndigits]) -> number
```

Round a number to a given precision in decimal digits (default 0 digits).
This returns an int when called with one argument,
otherwise the same type as the number.
ndigits may be negative.

round()



round()

```
help(round)
```

```
round(...)  
round(number[, ndigits]) -> number
```

Round a number to a given precision in decimal digits (default 0 digits).
This returns an int when called with one argument,
otherwise the same type as the number.
ndigits may be negative.

```
round(1.68)
```

```
round()
```



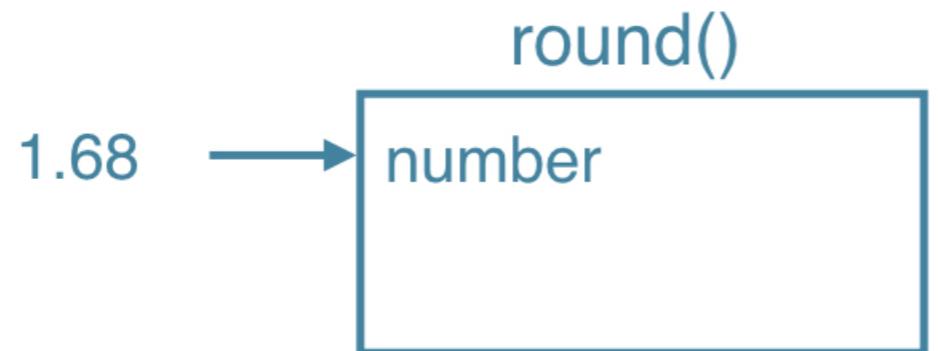
round()

```
help(round)
```

```
round(...)  
round(number[, ndigits]) -> number
```

Round a number to a given precision in decimal digits (default 0 digits).
This returns an int when called with one argument,
otherwise the same type as the number.
ndigits may be negative.

```
round(1.68)
```



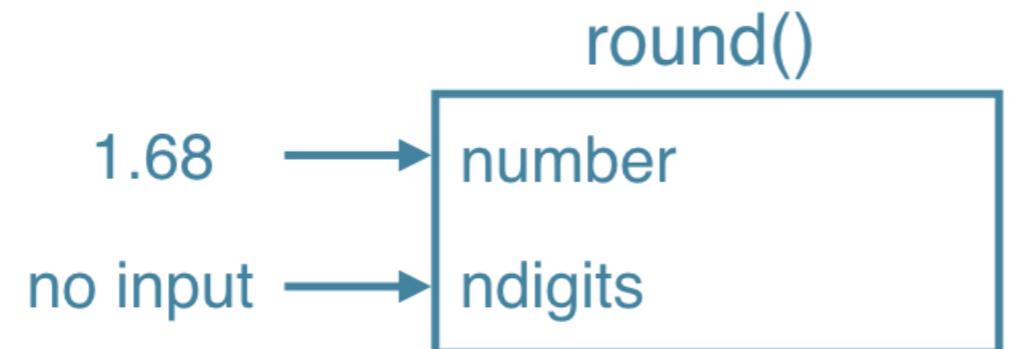
round()

```
help(round)
```

```
round(...)  
round(number[, ndigits]) -> number
```

Round a number to a given precision in decimal digits (default 0 digits).
This returns an int when called with one argument,
otherwise the same type as the number.
ndigits may be negative.

```
round(1.68)
```



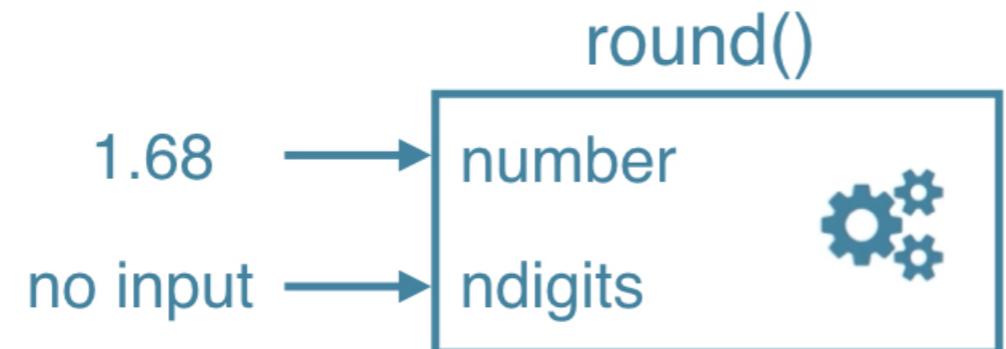
round()

```
help(round)
```

```
round(...)  
round(number[, ndigits]) -> number
```

Round a number to a given precision in decimal digits (default 0 digits).
This returns an int when called with one argument,
otherwise the same type as the number.
ndigits may be negative.

```
round(1.68)
```



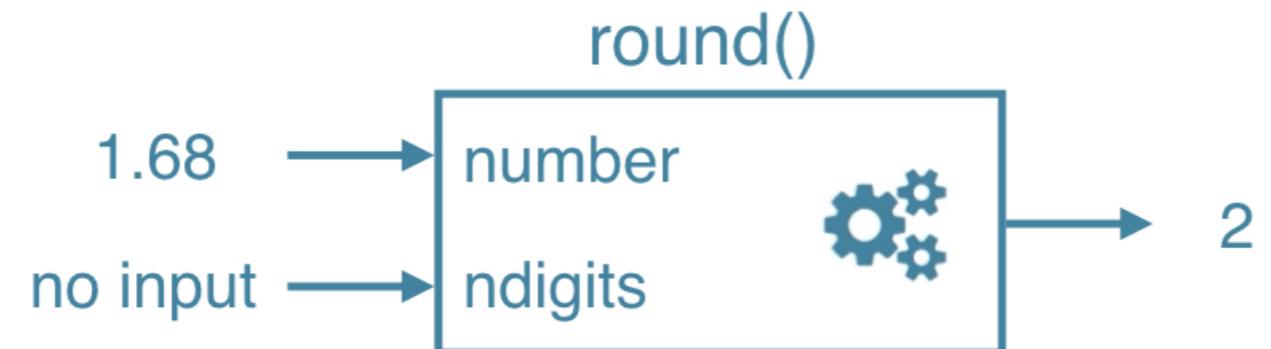
round()

```
help(round)
```

```
round(...)  
round(number[, ndigits]) -> number
```

Round a number to a given precision in decimal digits (default 0 digits).
This returns an int when called with one argument,
otherwise the same type as the number.
ndigits may be negative.

```
round(1.68)
```



round()

```
help(round)
```

```
round(...)  
round(number[, ndigits]) -> number
```

Round a number to a given precision in decimal digits (default 0 digits).
This returns an int when called with one argument,
otherwise the same type as the number.
ndigits may be negative.

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

Find functions

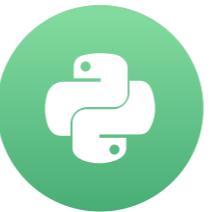
- How to know?
- Standard task -> probably function exists!
- The internet is your friend

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"
```

```
height = 1.73
```

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

Object

Object

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"
```

	type		examples of methods
sister = "liz"	Object	str	capitalize() replace()

```
height = 1.73
```

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

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

	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

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

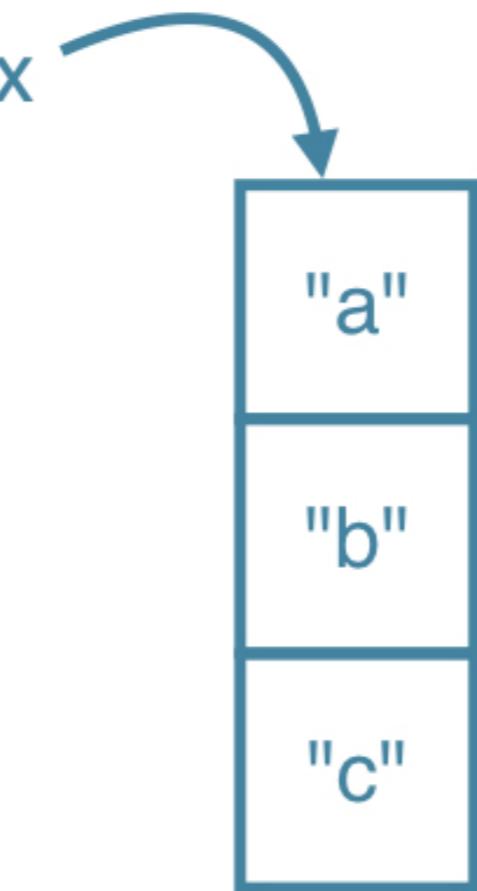
```
y[1] = "z"
```

```
y
```

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

```
x
```

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



Behind the scenes (1)

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

```
y = x
```

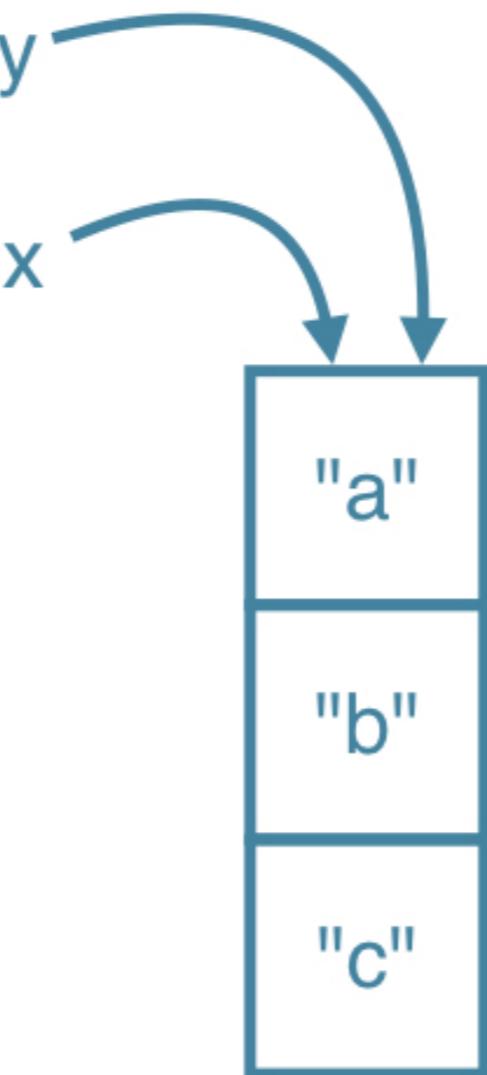
```
y[1] = "z"
```

```
y
```

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

```
x
```

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



Behind the scenes (1)

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

```
y = x
```

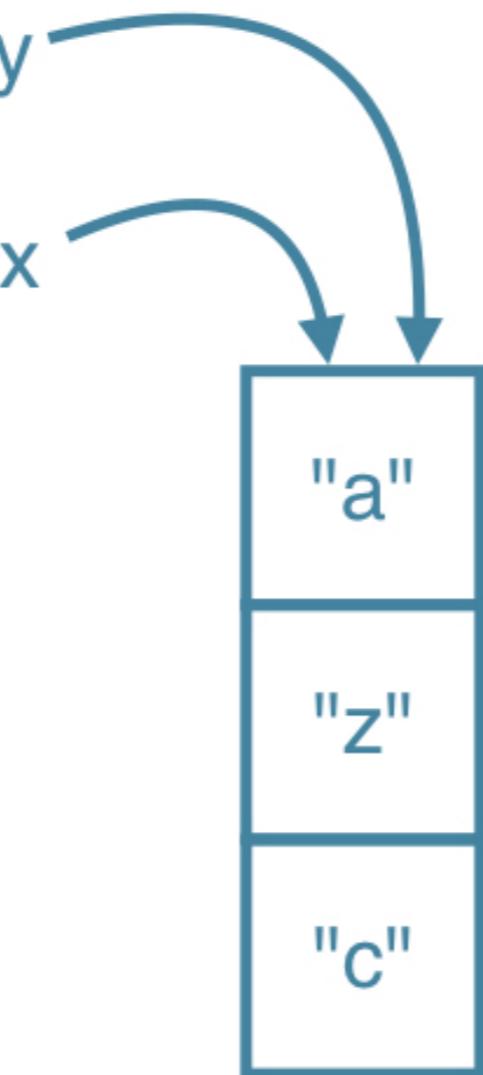
```
y[1] = "z"
```

```
y
```

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

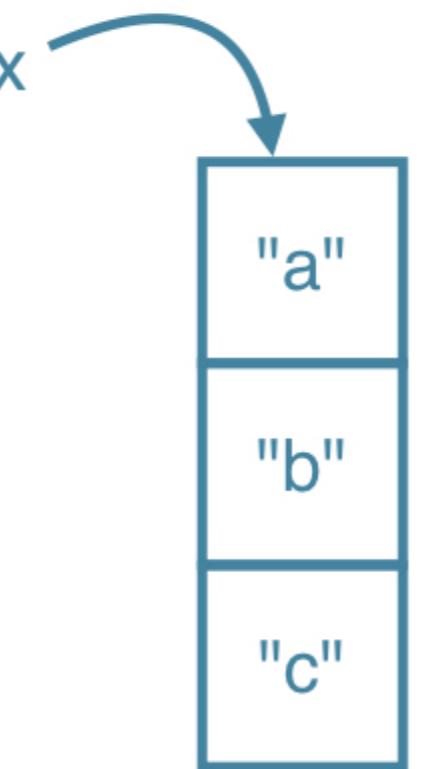
```
x
```

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



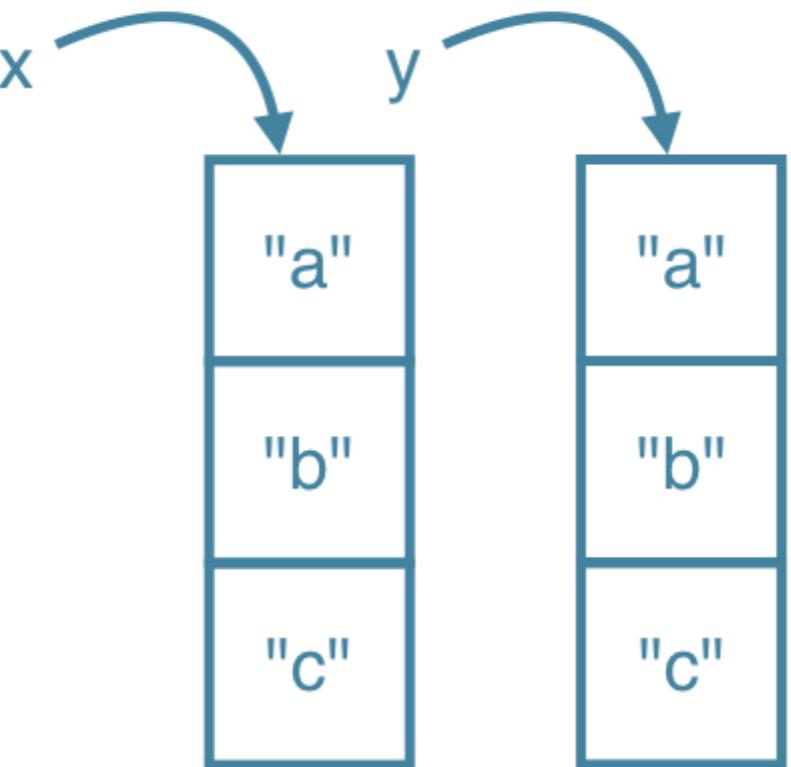
Behind the scenes (2)

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



Behind the scenes (2)

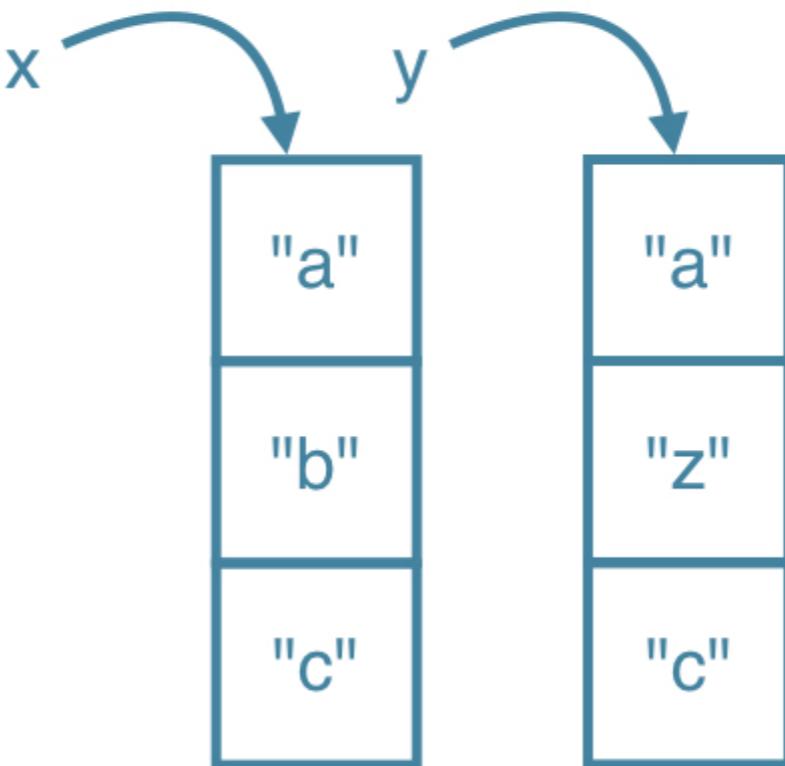
```
x = ["a", "b", "c"]  
y = list(x)  
y = x[:]
```



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

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".

Left Panel (Instructions):

- Exercise:** Calculations with variables
- Remember:** You calculated the money you ended up with after 7 years of investing \$100? You did something like this:
100 * 1.1 ** 7
- 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!
- Instructions:** 100 XP
- 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`.
- Take Hint (-30 XP)**

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 |
9
10 # Print out result
11
```

Buttons: Run Code, Submit Answer

Python Shell:

```
In [1]: |
```

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: Calculations with variables

Description: 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
```

Instructions: 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!

XP: 100 XP

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`.

Take Hint (-30 XP)

Code Editor: 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
```

Run Code | **Submit Answer**

IPython Shell:

```
In [1]:
```

IPython Shell

The screenshot shows a DataCamp exercise interface for an "Introduction to Python" course. The top navigation bar includes the DataCamp logo, a "Course Outline" button, and several status icons. The main area is divided into two tabs: "script.py" and "IPython Shell".

script.py Tab:

- Exercise Title:** Calculations with variables
- Description:** Remember how you calculated the money you ended up with after 7 years of investing \$100? You did something like this:
- Code Example:** `100 * 1.1 ** 7`
- Text:** 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!
- Instructions:** 500 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`.
- Take Hint (-30 XP)**

IPython Shell Tab:

- Text:** In [1]:

Python Script

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

The screenshot shows a DataCamp exercise interface. On the left, there's an 'Exercise' sidebar with a 'Course Outline' button. The main area has a title 'Calculations with variables'. Below it, text says: '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`. To the right, a red box highlights the 'script.py' code editor. It contains the following Python code:

```
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 |
9
10 # Print out result
11
```

Below the code editor are three buttons: a blue 'Run Code' button, a green 'Submit Answer' button, and a small circular icon with a number '5'. At the bottom, there's an 'IPython Shell' section with the text 'In [1]: |'.

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`.

Python Script

The screenshot shows a DataCamp Python script exercise interface. At the top, there's a navigation bar with the DataCamp logo, a 'Course Outline' button, and several status icons. Below the navigation bar, the main area is divided into sections:

- Exercise:** A title 'Calculations with variables' is displayed. Below it, a text block says: 'Remember how you calculated the money you ended up with after 7 years of investing \$100? You did something like this:' followed by the code `100 * 1.1 ** 7`.
- Instructions:** A list of 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`.
- IPython Shell:** A code input field labeled 'In [1]:' is shown, with a cursor at the end. To its right are three buttons: a blue circular arrow icon, a light blue 'Run Code' button, and a green 'Submit Answer' button.

Python Script

The screenshot shows a DataCamp exercise interface for a Python script. The top navigation bar includes the DataCamp logo, a 'Course Outline' button, and several status icons. The main area is titled 'script.py' and contains the following code:

```
1
```

Below the code editor, there's a text block titled 'Calculations with variables' containing instructions about calculating compound interest. It shows the formula `100 * 1.1 ** 7`. A note explains that instead of using actual values, variables can be used.

The 'Instructions' section lists 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`.

A 'Take Hint (-30 XP)' button is available for assistance. At the bottom, there are 'Run Code' and 'Submit Answer' buttons, along with an IPython Shell window showing the command `In [1]:`.

- 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 DataCamp logo, course outline, and other course-related icons. Below it, the main area is divided into several sections:

- Exercise:** A title "Calculations with variables" and a note about calculating money after 7 years of investing \$100.
- Code Editor:** A script named "script.py" containing the following Python 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 |
```
- Instructions:** A list of 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`.
- IPython Shell:** A panel labeled "In [1]:" which is currently empty.

At the bottom right of the main area, there 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

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 **: '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([ 21.852,  20.975,  21.75 ,  24.747,  21.441])
```

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 **: 'list' and 'int'
```

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

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

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

```
bmi[1]
```

```
20.975
```

```
bmi > 23
```

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

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

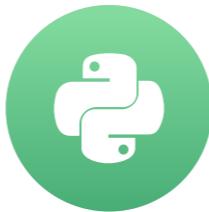
```
array([ 24.747])
```

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