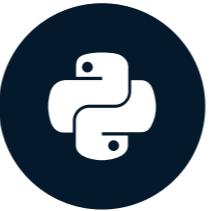


Hello Python!

INTRODUCTION TO PYTHON



Hugo Bowne-Anderson
Data Scientist at DataCamp

How you will learn

The screenshot shows a learning interface for an "Introduction to Python" course. The top navigation bar includes a logo, the path "Learn / Courses / Introduction to Python", and course outline navigation buttons. A "Light Mode" toggle is also present.

Exercise: Python as a calculator

Python is perfectly suited to do basic calculations. It can do addition, subtraction, multiplication and division.

The code in the script gives some examples.

Now it's your turn to practice!

Instructions:

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

Submit Answer

Python Shell: In [1]:

A large, semi-transparent watermark of the word "DATA" is overlaid on the left side of the interface.

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 learning platform interface for an "Introduction to Python" course. The main area is titled "Exercise" and contains the following content:

Python as a calculator

Python is perfectly suited to do basic calculations. It can do addition, subtraction, multiplication and division.

The code in the script gives some examples:

```
script.py
1 # Addition
2 # Subtraction
3 # Multiplication
4 # Division
```

Now it's your turn to practice!

Instructions

- Print the sum of 5 + 3.
- Print the result of subtracting 7 from 12.
- Multiply 3 by 5.
- Divide 10 by 2.

Take Hint (-30 XP)

At the bottom, there is an "IPython Shell" section with the prompt "In [1]:".

The IPython shell section has a large red arrow pointing towards the "Run Code" button, which is highlighted in green. The "Submit Answer" button is also green. The "Run Code" button has a small icon of a play button.

IPython Shell

Execute Python commands

The screenshot shows a Python exercise interface. At the top, there's a navigation bar with 'Learn / Courses / Introduction to Python' and a 'Course Outline' button. Below that is a toolbar with a 'Light Mode' switch. The main area is titled 'Exercise' and contains the following content:

Python as a calculator

Python is perfectly suited to do basic calculations. It can do addition, subtraction, multiplication and division.

The code in the script gives some examples:

```
1 # Addition
2 # Subtraction
3 # Multiplication
4 # Division
```

Now it's your turn to practice!

Instructions

- Print the sum of 5 + 3.
- Print the result of subtracting 7 from 12.
- Multiply 3 by 5.
- Divide 10 by 2.

Take Hint (-30 XP)

On the right side of the interface, there's an IPython Shell window with a green border. It has a title bar 'IPython Shell' and a command input field 'In [1]:'. The shell is currently empty.

IPython Shell

The screenshot shows a Python exercise interface. At the top, there's a navigation bar with 'Learn / Courses / Introduction to Python' and a 'Course Outline' button. A 'Light Mode' switch is also present. The main area is titled 'Exercise' and contains the following content:

Python as a calculator

Python is perfectly suited to do basic calculations. It can add, subtract, multiply and division.

The code in the script gives some examples.

Now it's your turn to practice!

Instructions

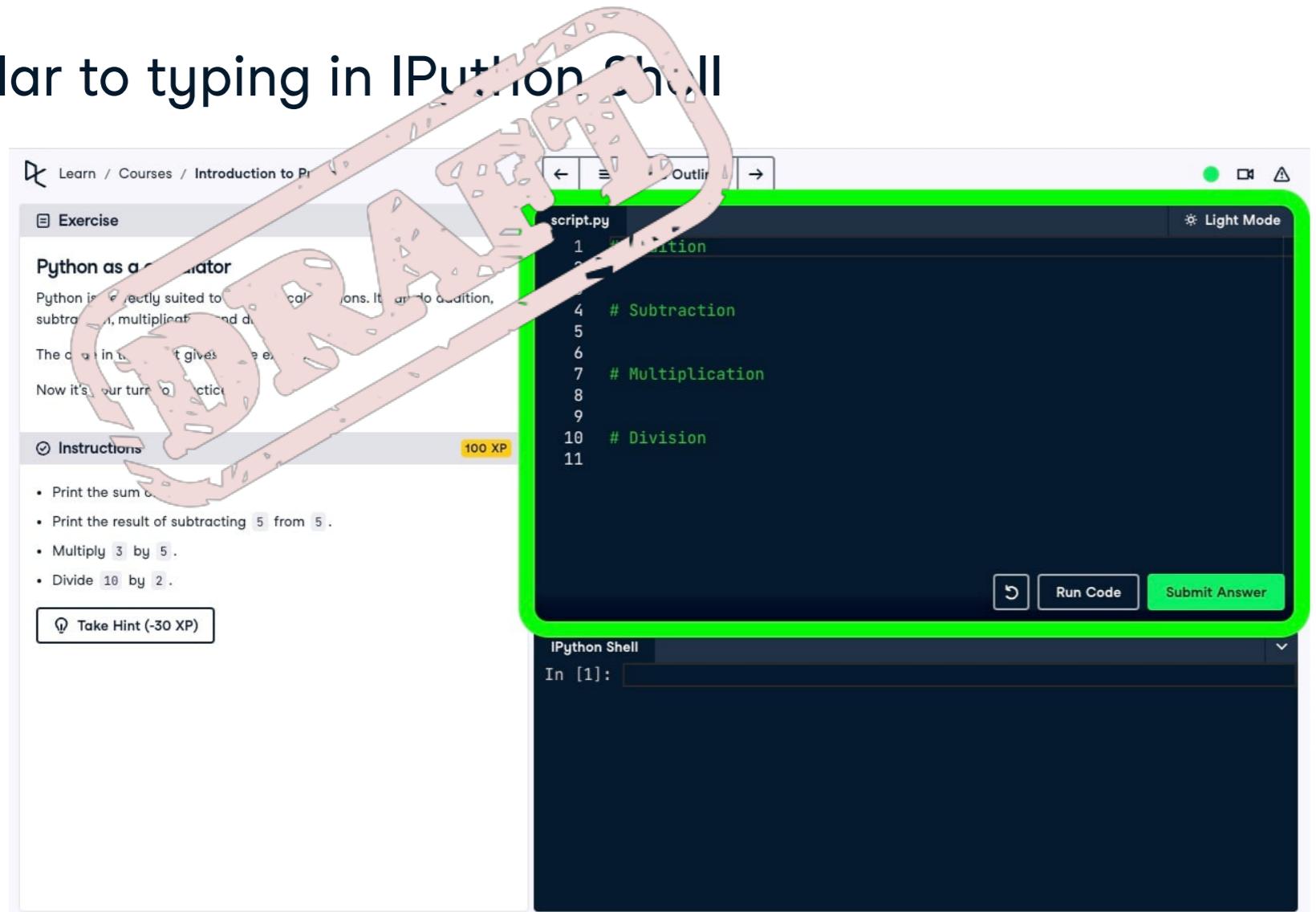
- Print the sum of 5 + 5.
- Print the result of subtracting 5 from 10.
- Multiply 3 by 5.
- Divide 10 by 2.

Take Hint (-50 XP)

Below the instructions is an IPython Shell window with the title 'IPython Shell'. It shows the command 'In [1]:' followed by a blank input field. There are three buttons at the bottom of the shell: a blue question mark icon, a grey 'Run Code' button, and a green 'Submit Answer' button.

Python Script

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



A screenshot of a Python script editor interface. The main window shows a code editor with a file named `script.py` containing the following code:

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

The code editor has a green border. To the left of the code editor, there is a sidebar with the following content:

- Exercise**:
 - Python as a calculator**: A brief introduction to Python's arithmetic operators.
 - Instructions**:
 - Print the sum of 5 and 5.
 - Print the result of subtracting 5 from 5.
 - Multiply 3 by 5.
 - Divide 10 by 2.

Below the instructions is a button labeled "Take Hint (-30 XP)".

At the bottom of the code editor window, there are three buttons: "Run Code", "Submit Answer", and a refresh icon.

Below the code editor is an "IPython Shell" section with the text "In [1]:".

Python Script

The screenshot shows a Python script editor interface. At the top, there's a navigation bar with 'Learn / Courses / Introduction to Python' and a 'Course Outline' button. Below that is a toolbar with a 'Light Mode' switch. The main area is divided into sections: 'Exercise' (containing 'Python as a calculator' instructions), 'Instructions' (with a list of tasks), and an 'IPython Shell' (where code can be run). A large, semi-transparent watermark with the word 'DRAFT' is overlaid across the entire interface.

Learn / Courses / Introduction to Python

Course Outline

Light Mode

Exercise

Python as a calculator

Python is perfectly suited to do basic calculations. It can do addition, subtraction, multiplication and division.

The code in the script gives some examples.

Now it's your turn to practice!

Instructions

- Print the sum of `4 + 5`.
- Print the result of subtracting `5` from `10`.
- Multiply `3` by `5`.
- Divide `10` by `2`.

Take Hint (-30 XP)

script.py

```
1 4
```

Run Code Submit Answer

IPython Shell

In [1]:

Python Script

The screenshot shows a Python script exercise interface. At the top, there's a navigation bar with 'Learn / Courses / Introduction to Python' and a 'Course Outline' button. Below that is a toolbar with a 'Light Mode' switch. The main area is divided into sections: 'Exercise' (containing 'Python as a calculator' text and examples), 'Instructions' (with a list of tasks like 'Print the sum of 4 + 5'), and an 'IPython Shell' (where code can be run). A large, semi-transparent watermark reading 'DRAFT' is overlaid across the entire interface.

- 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 the DataCamp logo, 'Learn / Courses / Introduction to Python', and course outline and mode buttons. The main area is titled 'Exercise' and has a sub-section 'Python as a calculator'. It contains text about Python's suitability for calculations and examples from a script named 'script.py'. The script code is as follows:

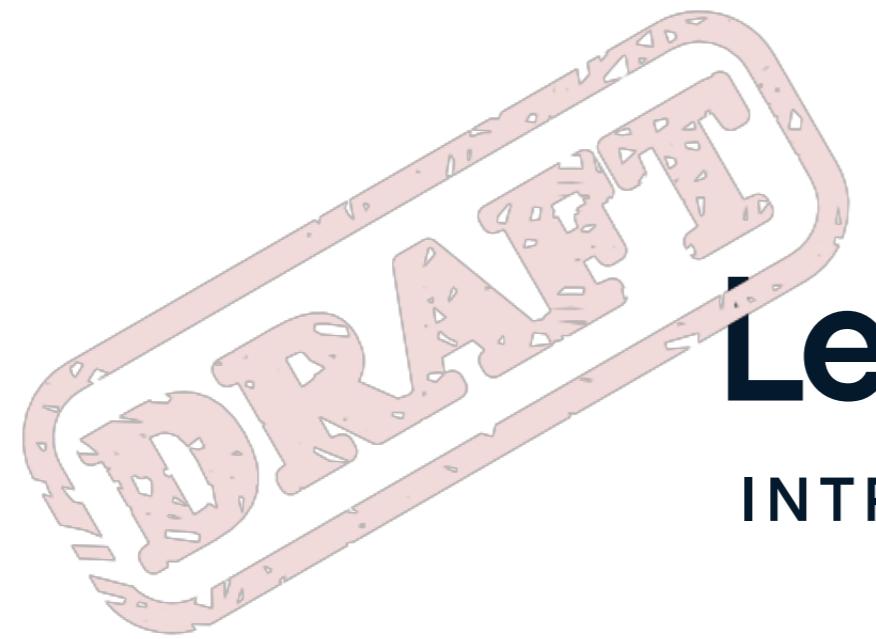
```
script.py
1 # Addition
2
3
4 # Subtraction
5
6 # Multiplication
7
8 # Division
9
10
11
```

The code editor includes a 'Run Code' and 'Submit Answer' button. Below the code editor is an 'IPython Shell' section with the prompt 'In [1]:'. On the left side of the exercise area, there's a decorative banner with the word 'DATA' and a progress bar indicating '100 XP'.

Instructions

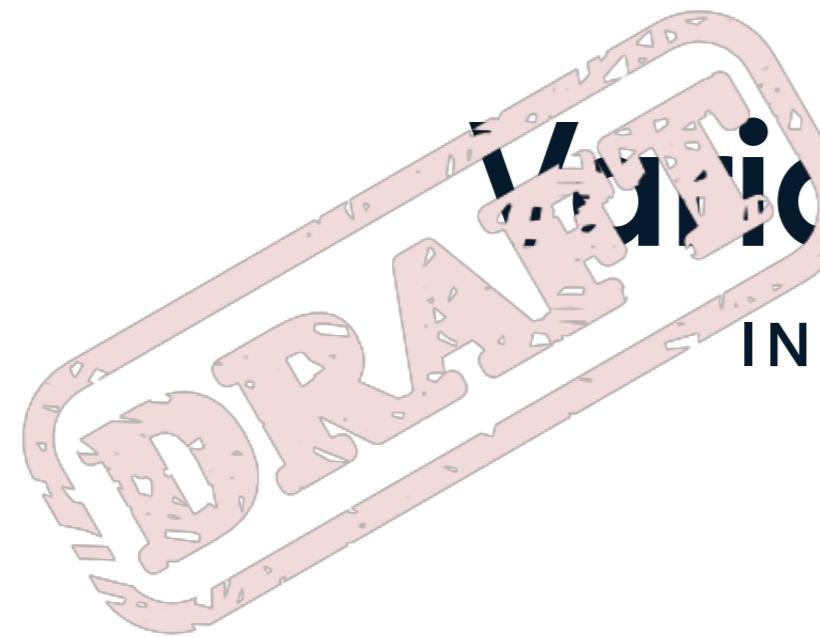
- Print the sum of `5 + 5`.
- Print the result of subtracting `5` from `10`.
- Multiply `3` by `5`.
- Divide `10` by `2`.

Take Hint (-30 XP)



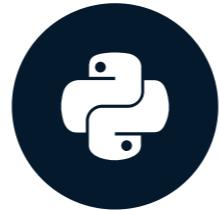
Let's practice!

INTRODUCTION TO PYTHON



Variables and Types

INTRODUCTION TO PYTHON

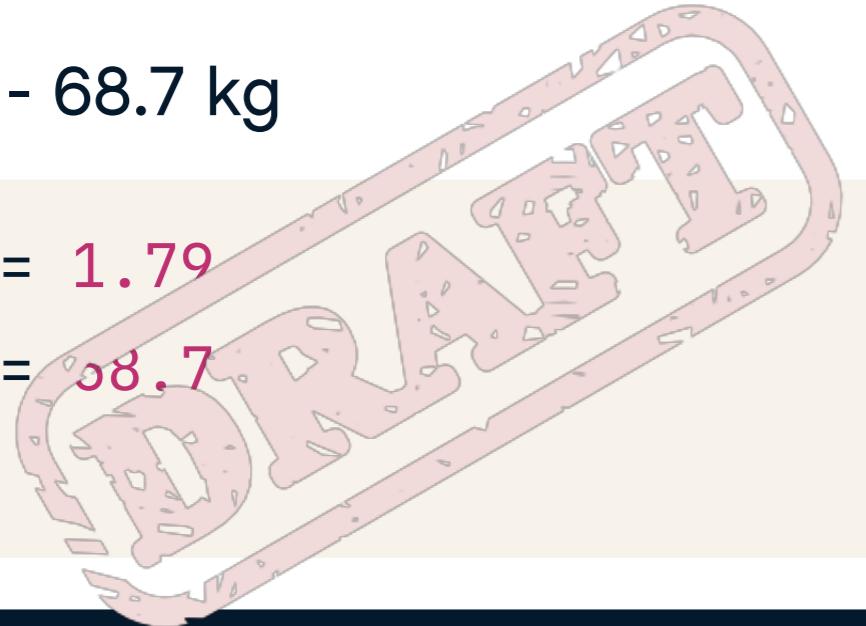


Hugo Bowne-Anderson
Data Scientist at DataCamp

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

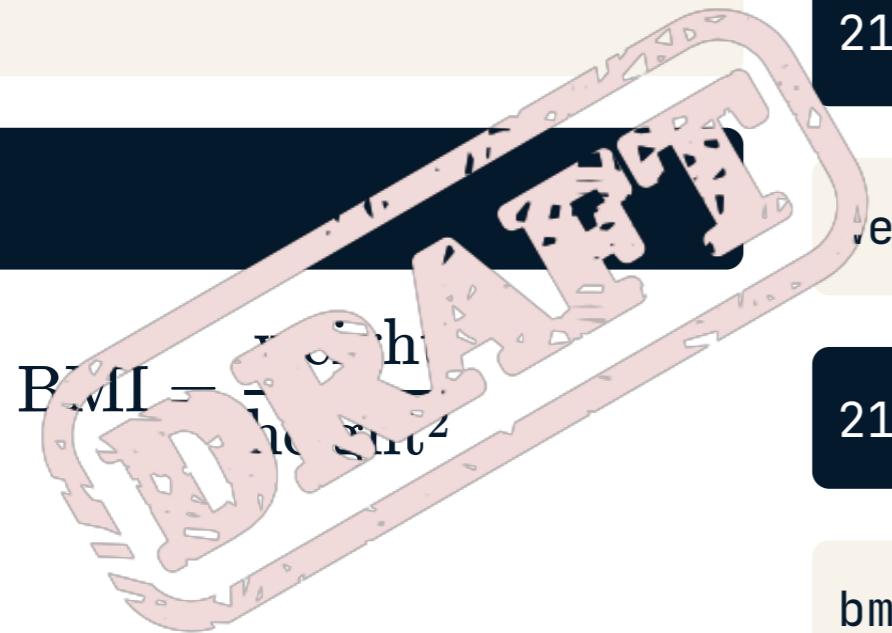
Calculate BMI

```
height = 1.79
```

```
weight = 68.7
```

```
height
```

```
1.79
```



```
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)
```

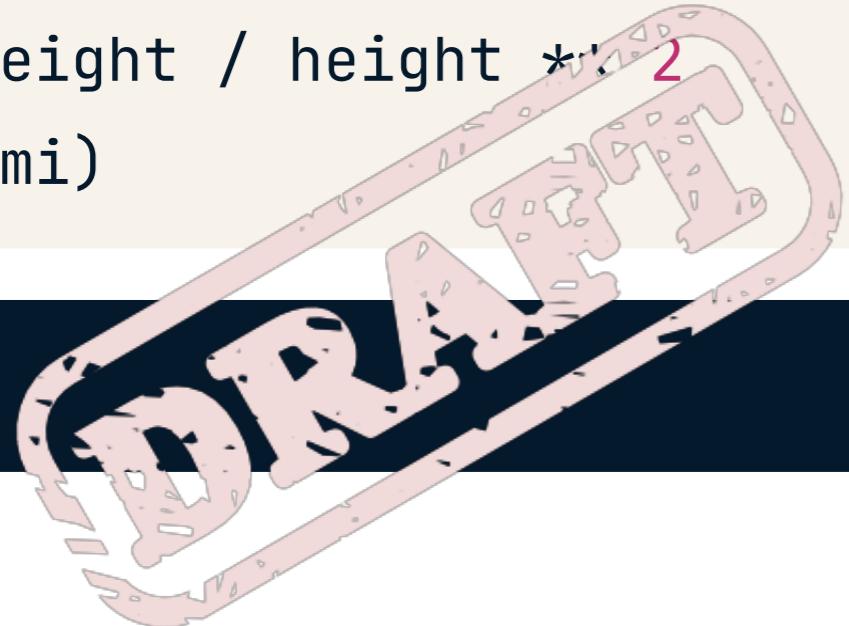
21.4413



Reproducibility

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

23.1578



Python Types

```
type(bmi)
```

```
float
```

```
day_of_week
```

```
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

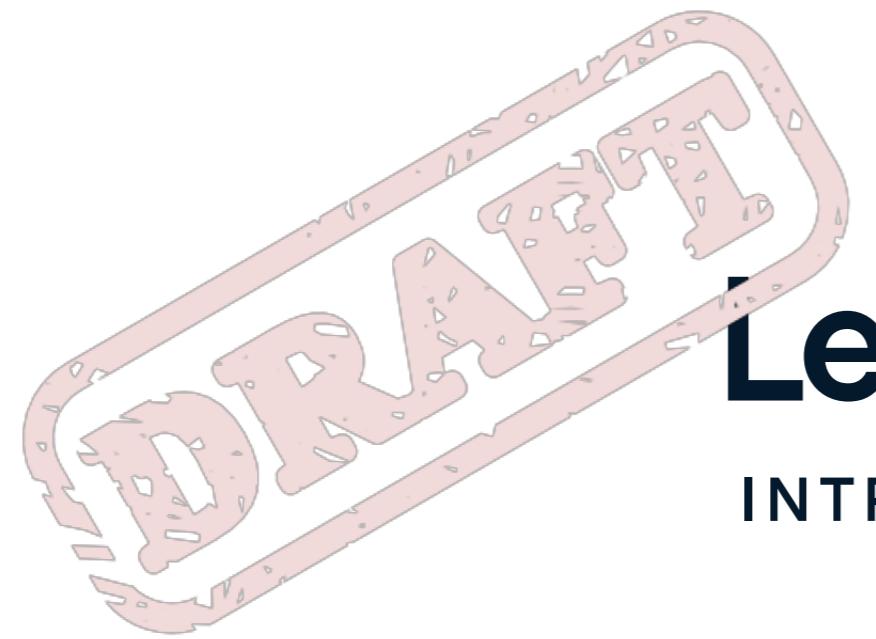
5

'ab' + 'cd'

'abcd'

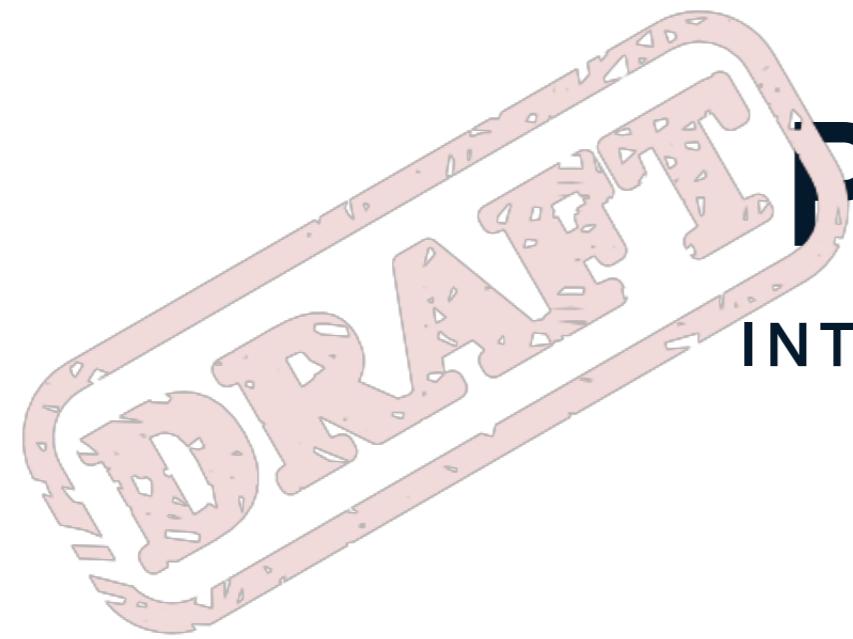


- Different type = different behavior!



Let's practice!

INTRODUCTION TO PYTHON



Python Lists

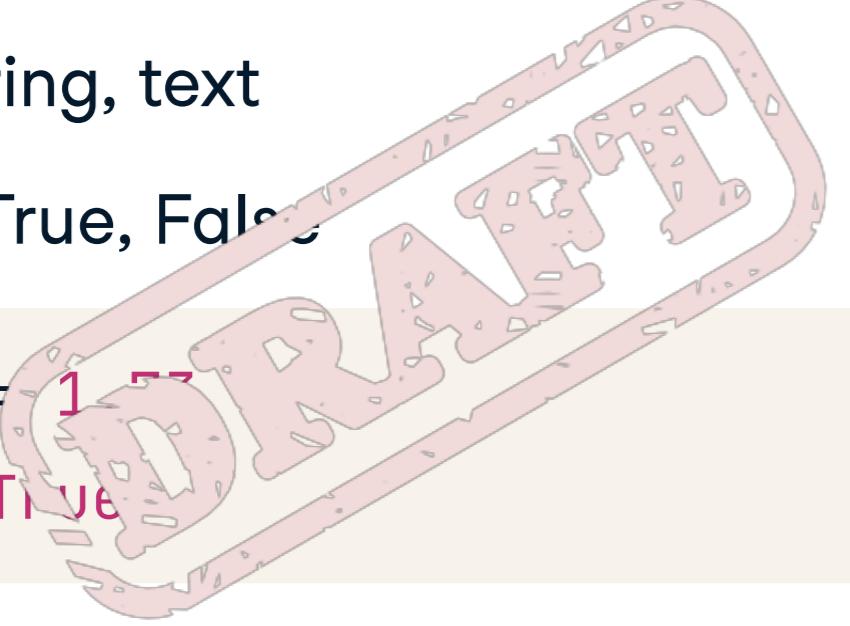
INTRODUCTION TO PYTHON



Hugo Bowne-Anderson
Data Scientist at DataCamp

Python Data Types

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



```
height = 1.77  
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 = 1.80
```



- 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]
```

```
[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

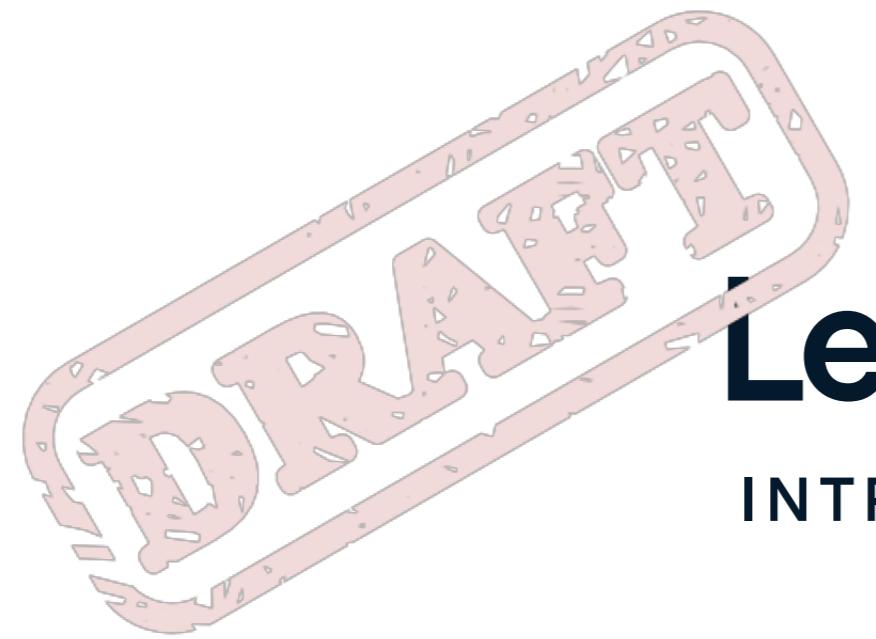
```
type(fam)
```

```
list
```

```
type(fam2)
```

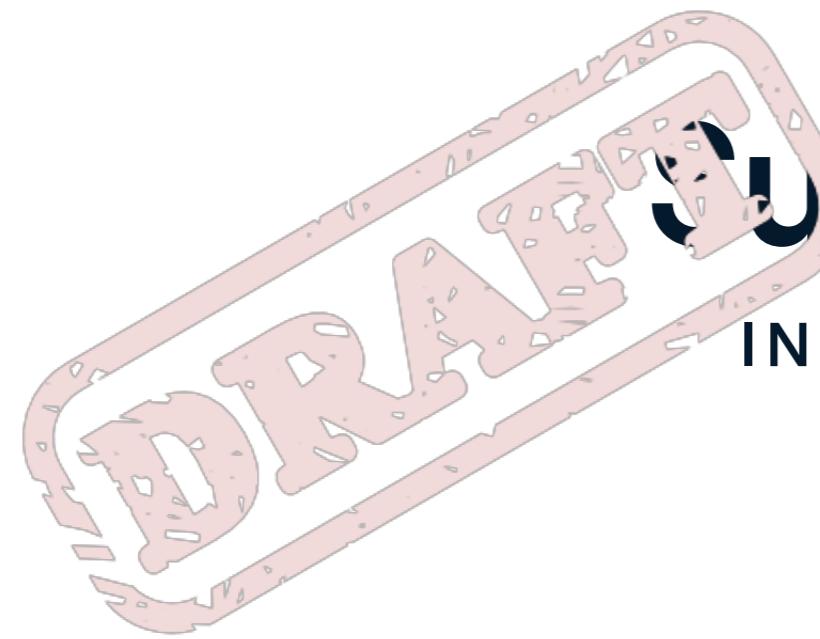
```
list
```

- 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
```

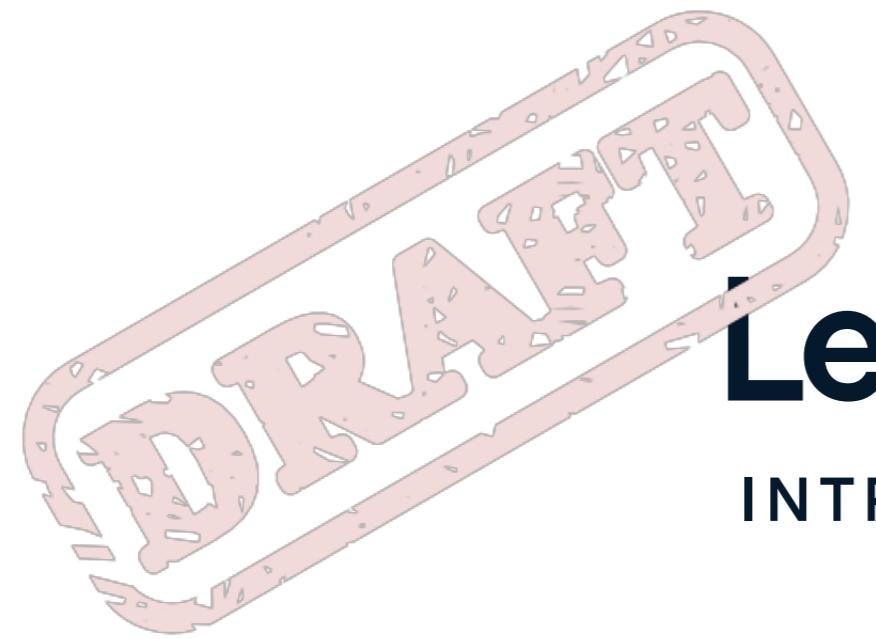
```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

```
fam[:4]
```

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

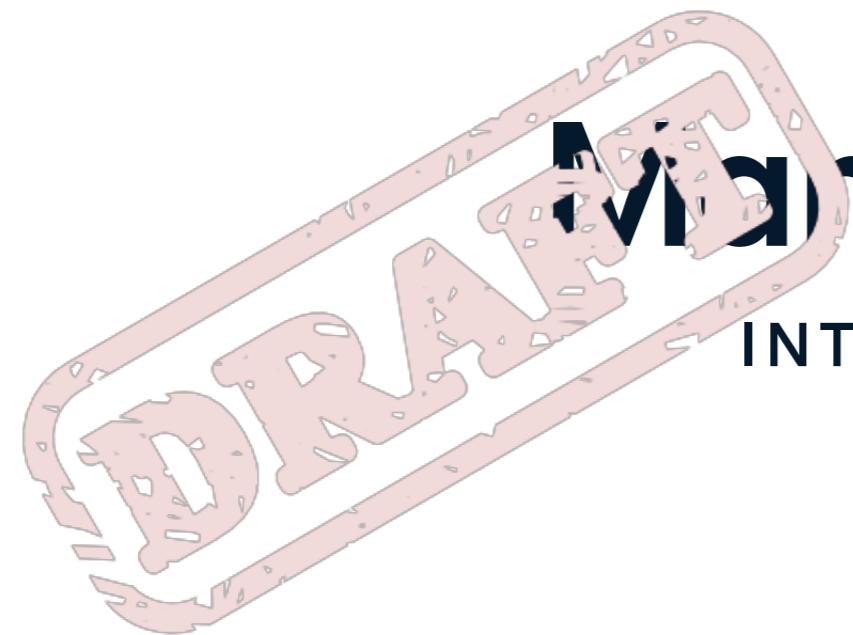
```
fam[5:]
```

```
[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, 'lisa', 1.74, '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]
```

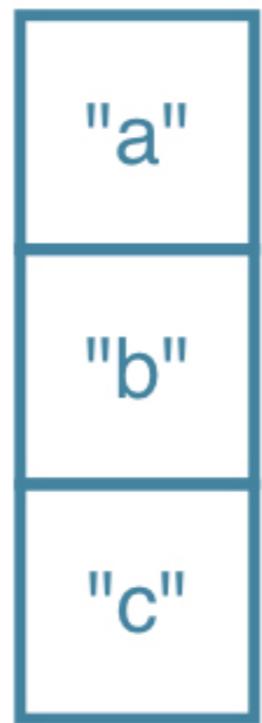
```
['lisa', 1.74, 'emma', 1.68, 'mom', 1.71, 'dad', 1.86, 'me', 1.79]
```

```
fam_ext = fam + ["me", 1.79]  
del fam[2]  
fam
```

```
['lisa', 1.74, 'emma', 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
```

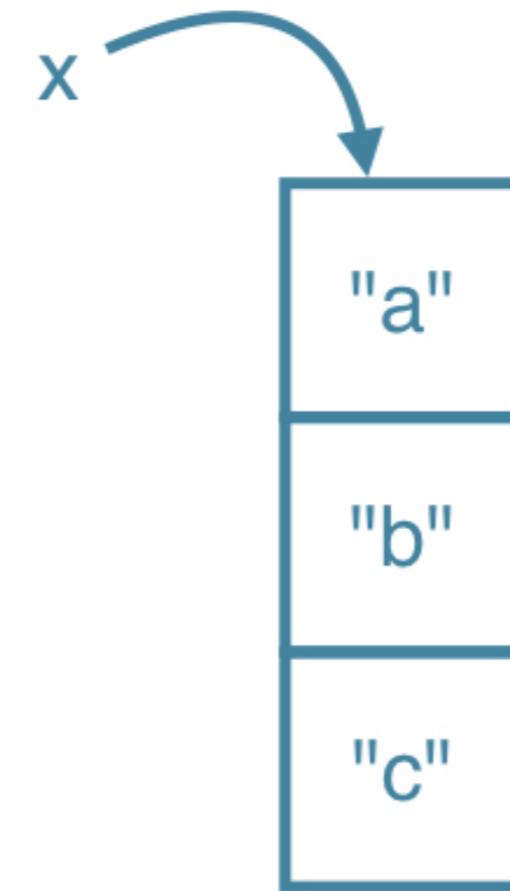
```
y[1] = "z"
```

```
y
```

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

```
x
```

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



Behind the scenes (1)

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

```
y = x
```

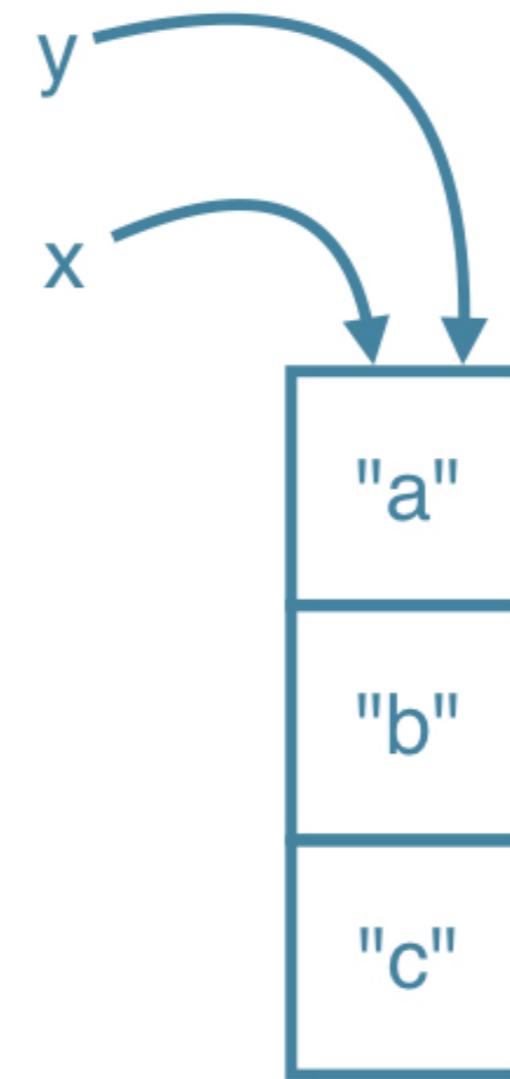
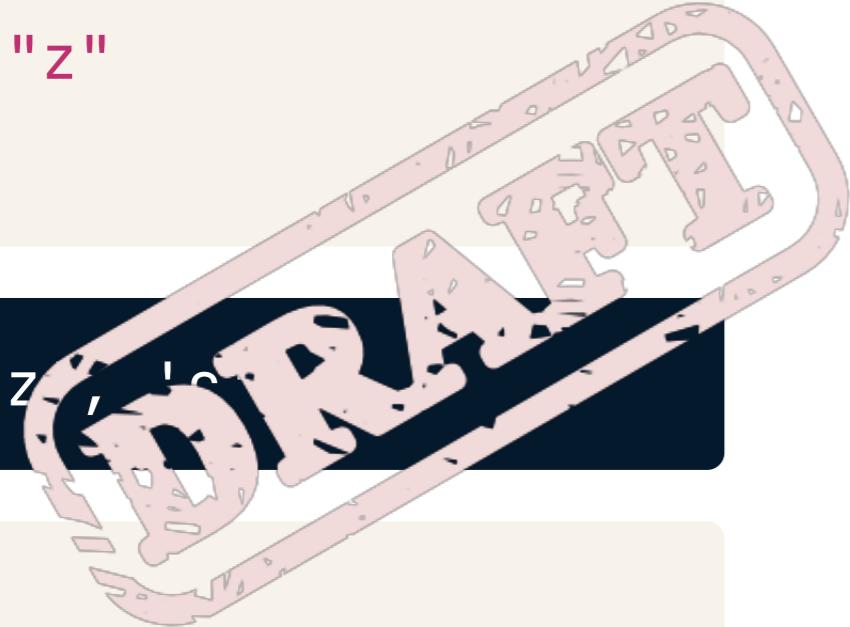
```
y[1] = "z"
```

```
y
```

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

```
x
```

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



Behind the scenes (1)

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

```
y = x
```

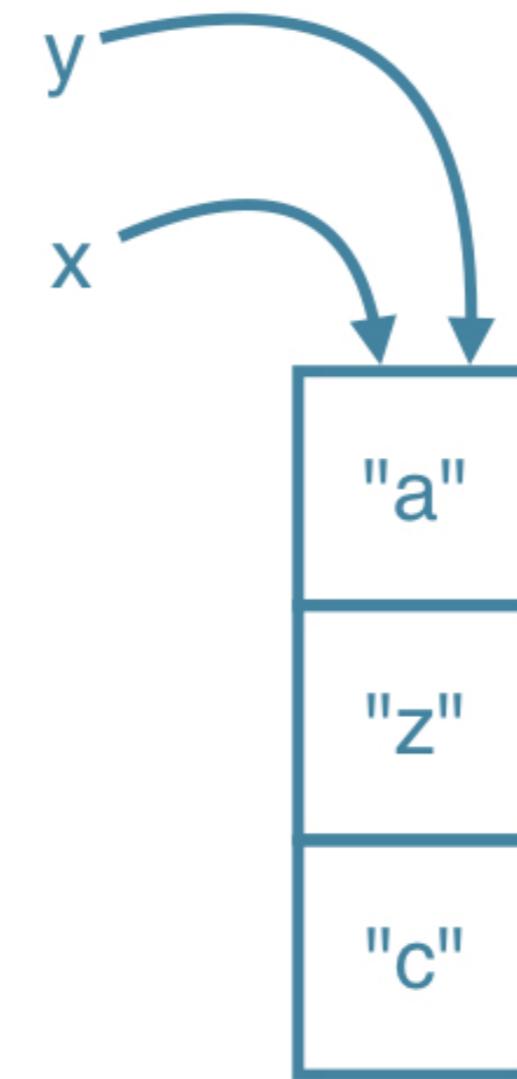
```
y[1] = "z"
```

```
y
```

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

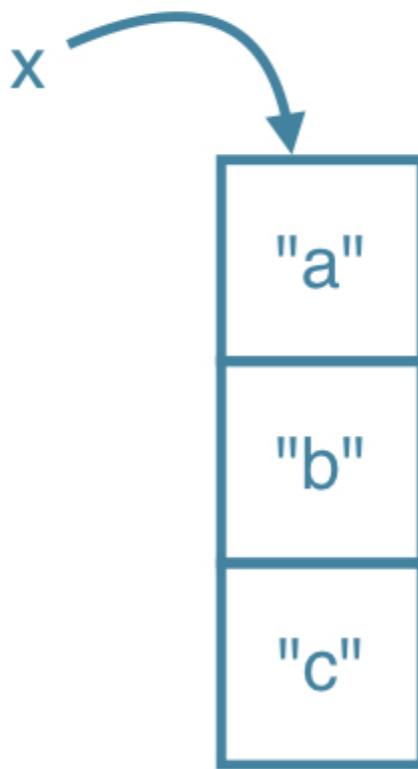
```
x
```

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



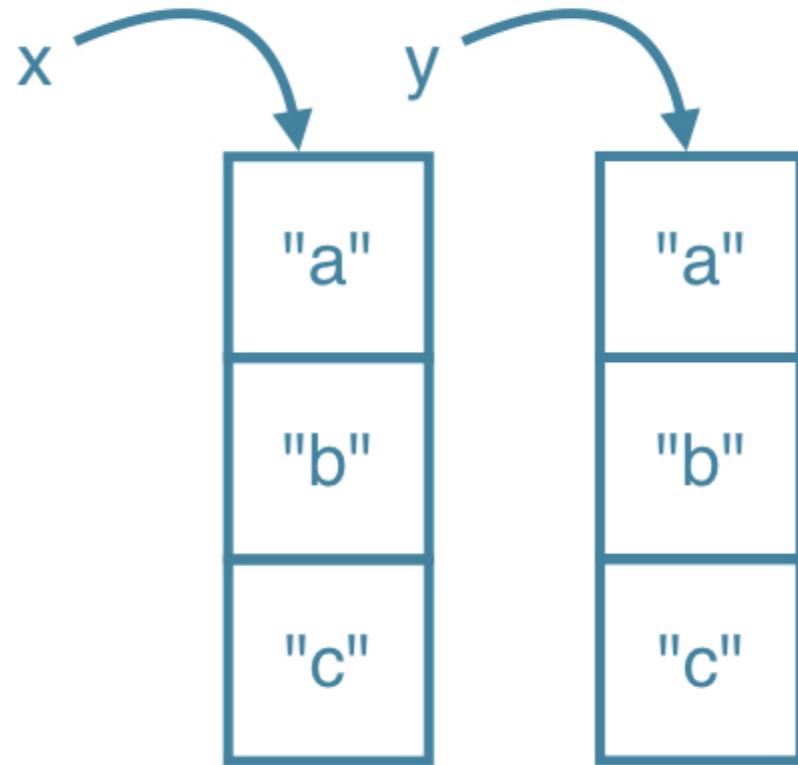
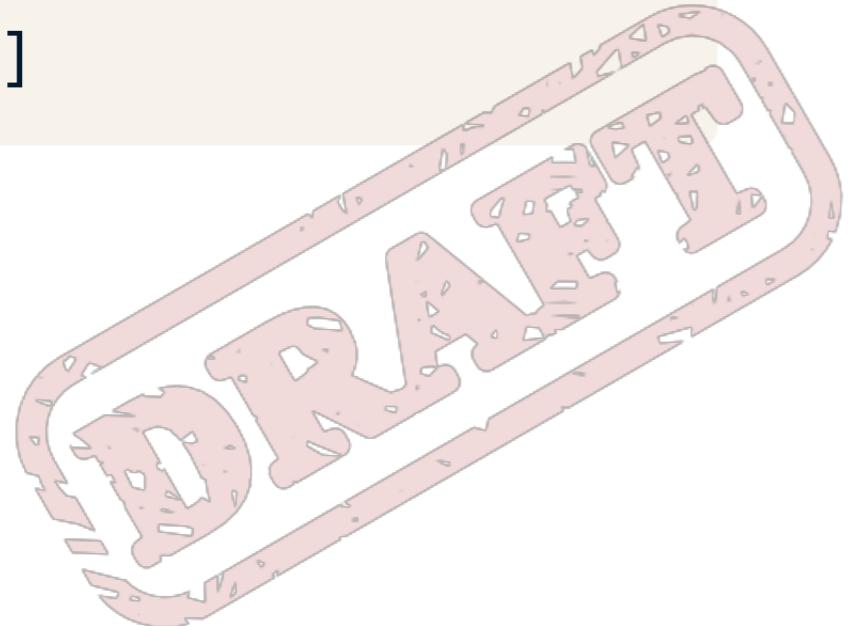
Behind the scenes (2)

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



Behind the scenes (2)

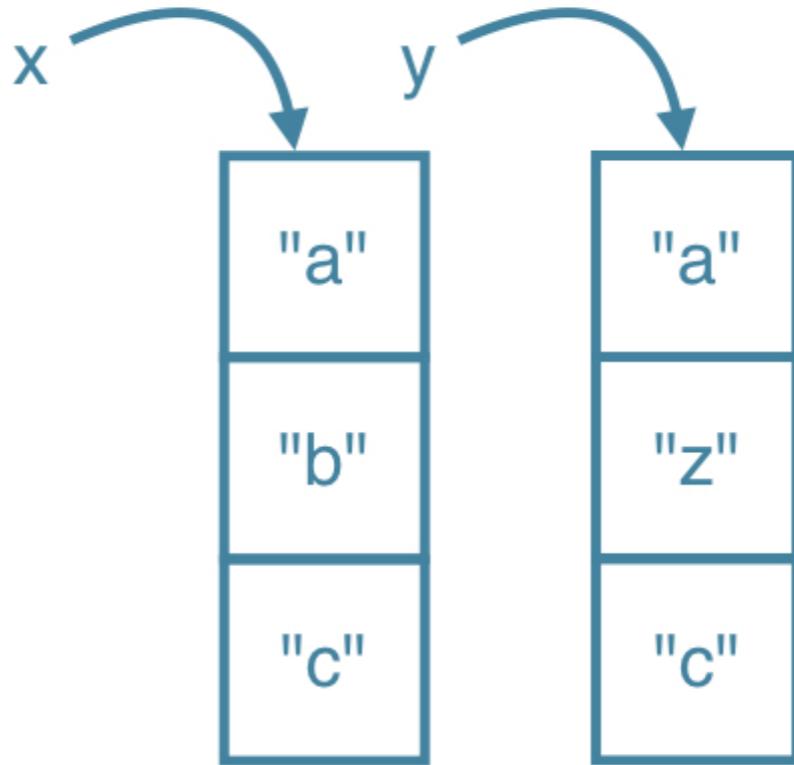
```
x = ["a", "b", "c"]  
y = list(x)  
y = x[:]
```



Behind the scenes (2)

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

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





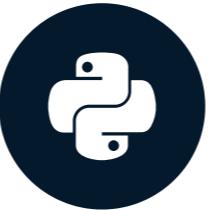
Let's practice!

INTRODUCTION TO PYTHON



Functions

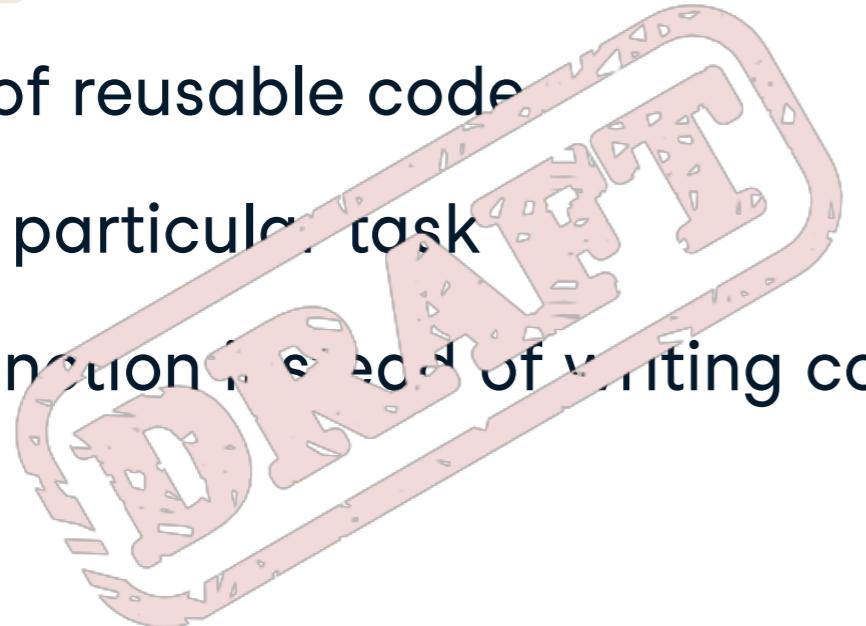
INTRODUCTION TO PYTHON



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] →
```

```
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
```

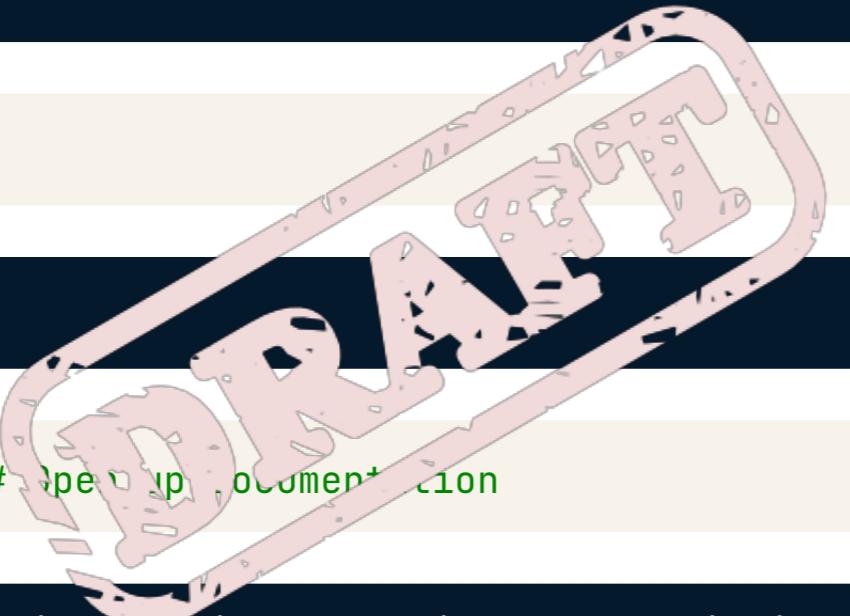
```
tallest = max(fam)  
tallest
```

```
1.89
```

round()

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

```
1.7
```



```
round(1.68)
```

```
2
```

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

```
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)
```

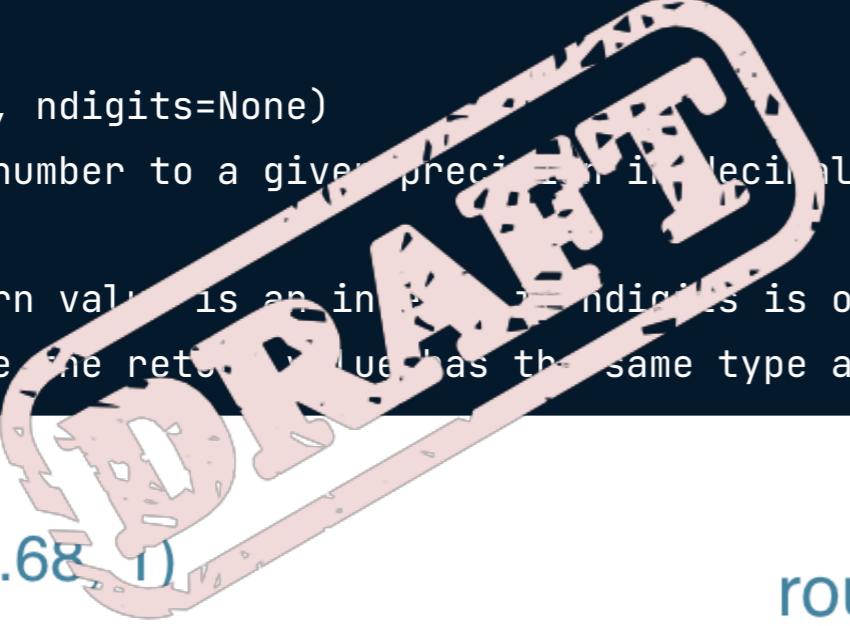
```
Help on built-in function round in module builtins:
```

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

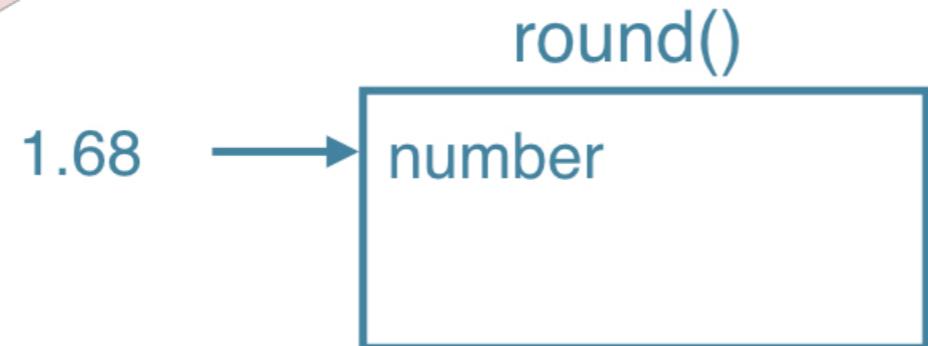
Round a number to a given precision in decimal digits.

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:
```

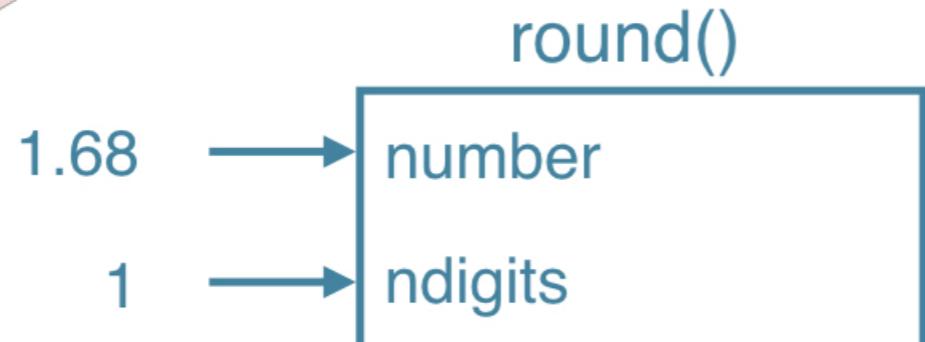
```
round(number, ndigits=None)
```

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:
```

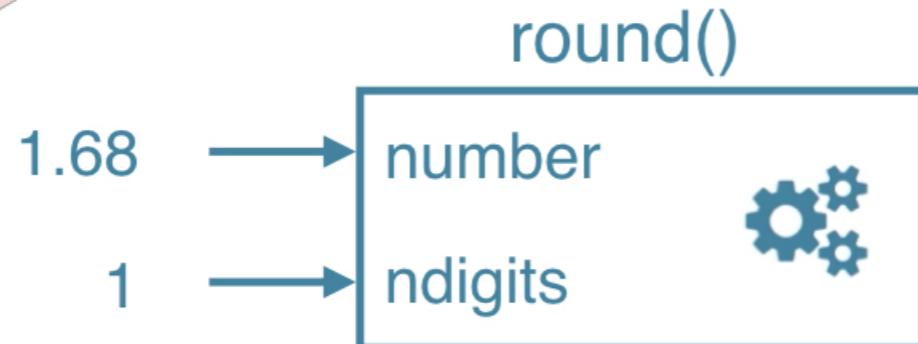
```
round(number, ndigits=None)
```

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:
```

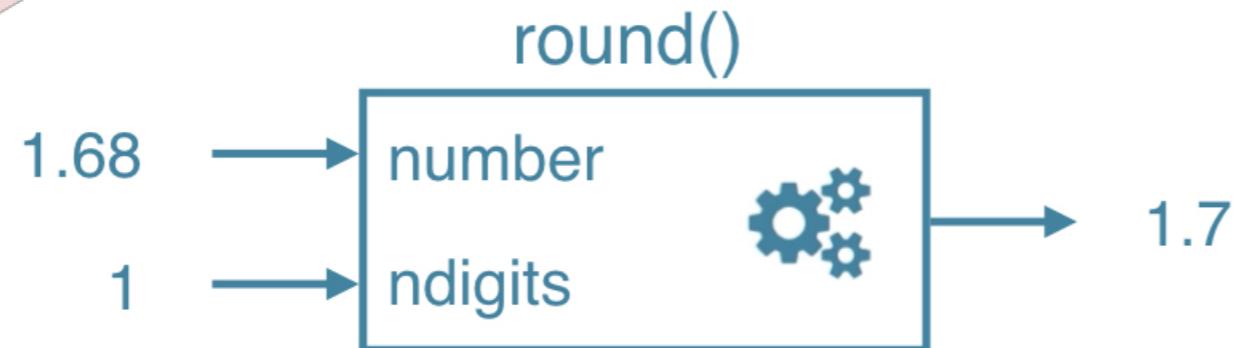
```
round(number, ndigits=None)
```

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)
```

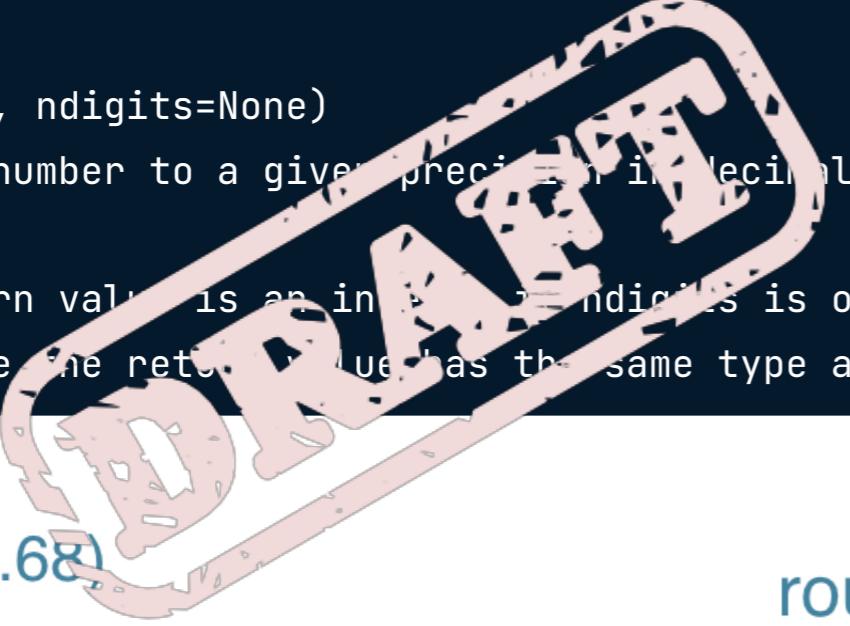
```
Help on built-in function round in module builtins:
```

```
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.

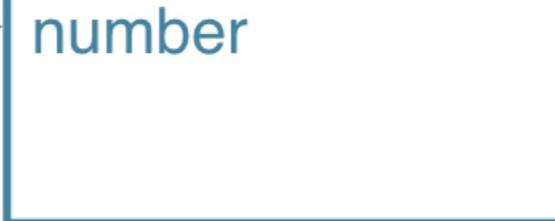
Otherwise the return value has the same type as the number. ndigits may be negative.



```
round(1.68)
```

round()

1.68



round()

```
help(round)
```

```
Help on built-in function round in module builtins:
```

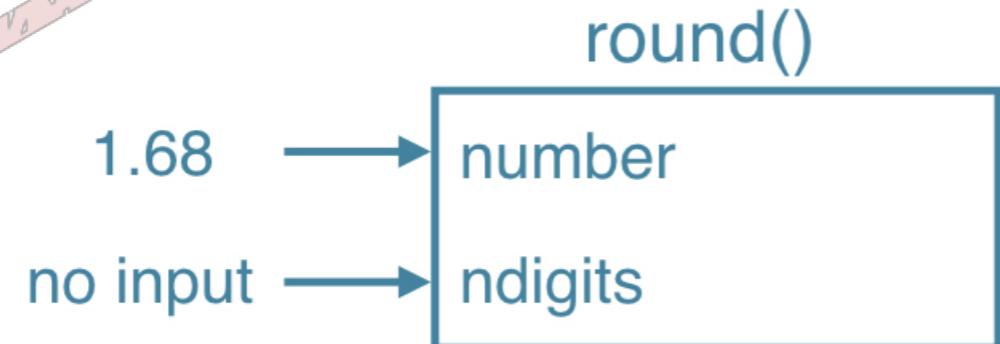
```
round(number, ndigits=None)
```

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:
```

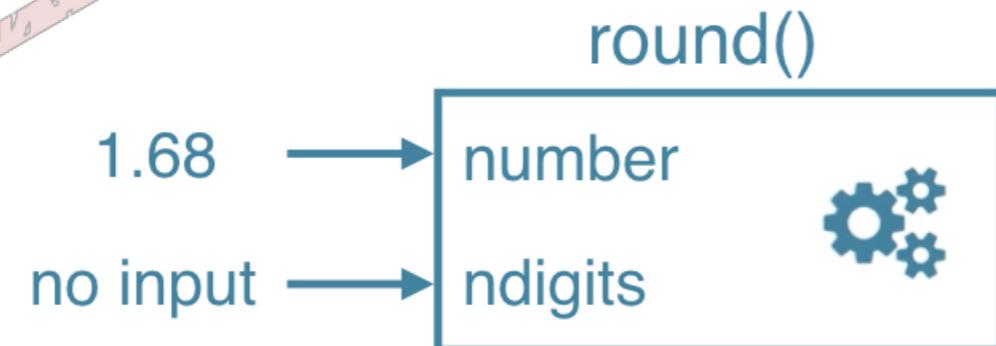
```
round(number, ndigits=None)
```

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:
```

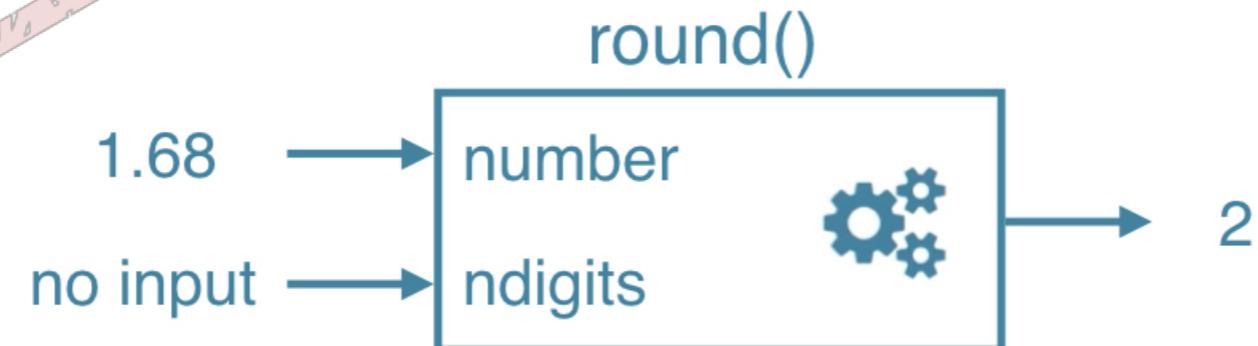
```
round(number, ndigits=None)
```

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.
```

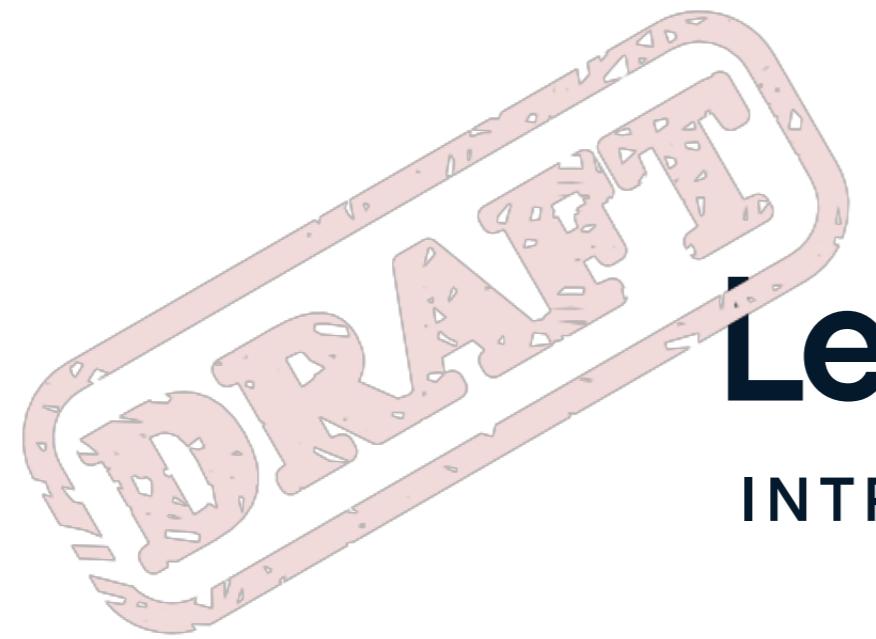
```
Otherwise the return value has the same type as the number. ndigits may be negative.
```

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

Find functions

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





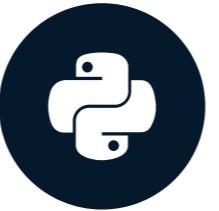
Let's practice!

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: `.`

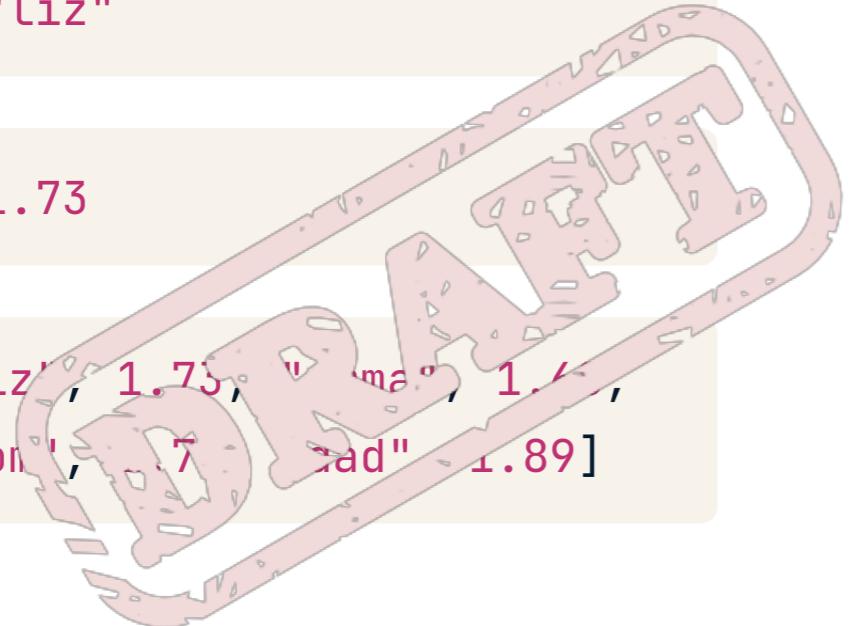


Back 2 Basics

```
sister = "liz"
```

```
height = 1.73
```

```
fam = ["liz", 1.73, "ma", 1.67,  
       "mon", 1.7, "dad", 1.89]
```



Object

Object

Object

Back 2 Basics

```
sister = "liz"
```

```
height = 1.73
```

```
fam = ["liz", 1.73, "ma", 1.67,  
       "mon", 1.7, "dad", 1.89]
```

- Methods: Functions that belong to objects

Object type
str

Object float

Object list

Back 2 Basics

```
sister = "liz"
```

```
height = 1.73
```

```
fam = ["liz", 1.73, "ma", 1.67,  
       "mon", 1.7, "dad", 1.89]
```

- Methods: Functions that belong to objects

	type	examples of methods
Object	str	capitalize() replace()
Object	float	bit_length() conjugate()
Object	list	index() count()

list methods

fam

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

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

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("lisa", "milly")
```

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

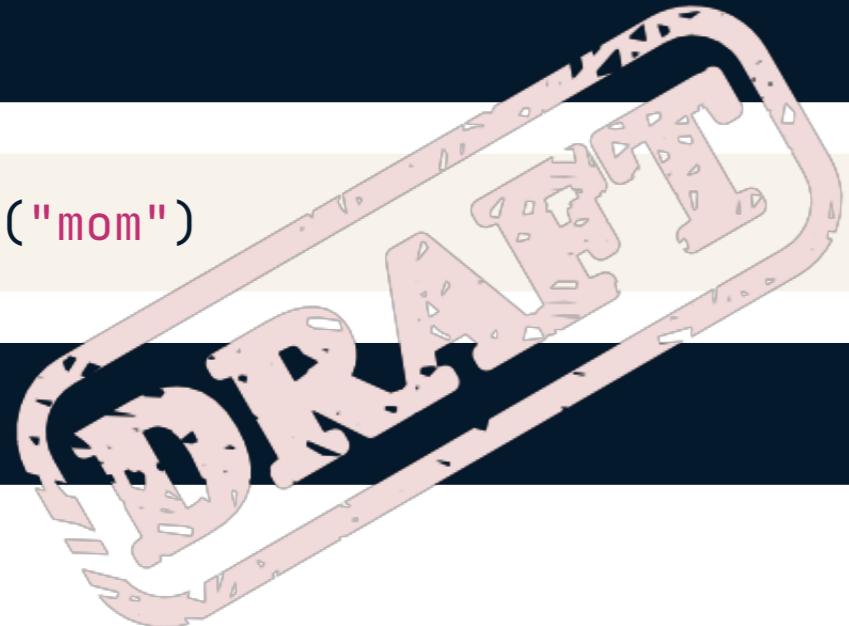
Methods

```
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



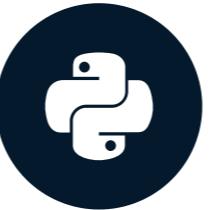
Let's practice!

INTRODUCTION TO PYTHON



Packages

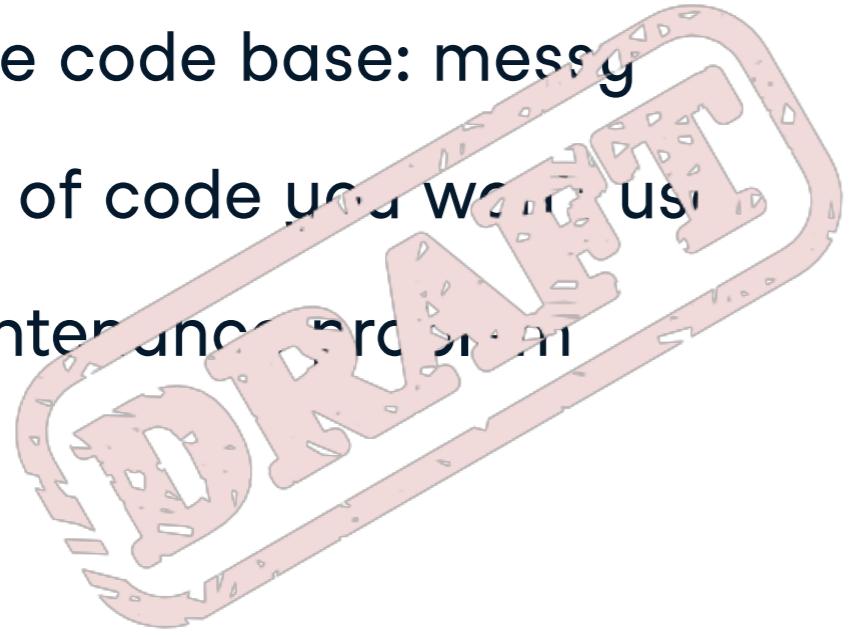
INTRODUCTION TO PYTHON



Hugo Bowne-Anderson
Data Scientist at DataCamp

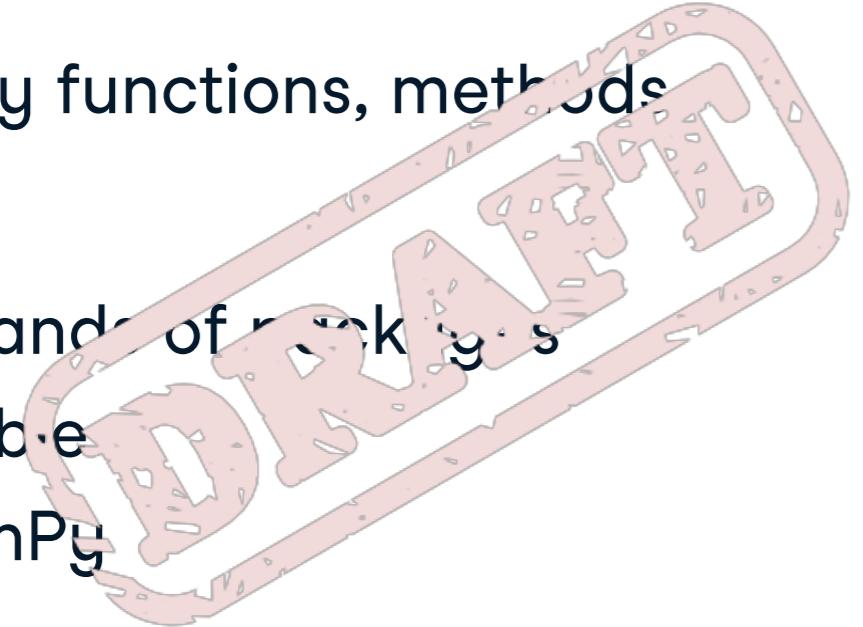
Motivation

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



Packages

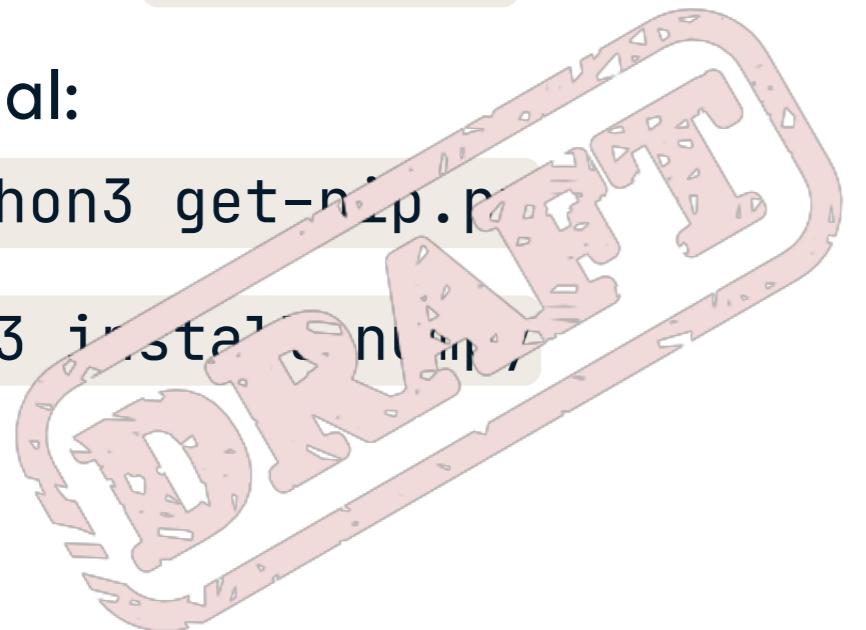
- Directory of Python Scripts
- Each script = module
- Specify functions, methods, types
- Thousands of packages available
 - NumPy
 - Matplotlib
 - scikit-learn



```
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])
```

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

NameError: name 'array' is not defined

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

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

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

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

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

from numpy import array

- my_script.py

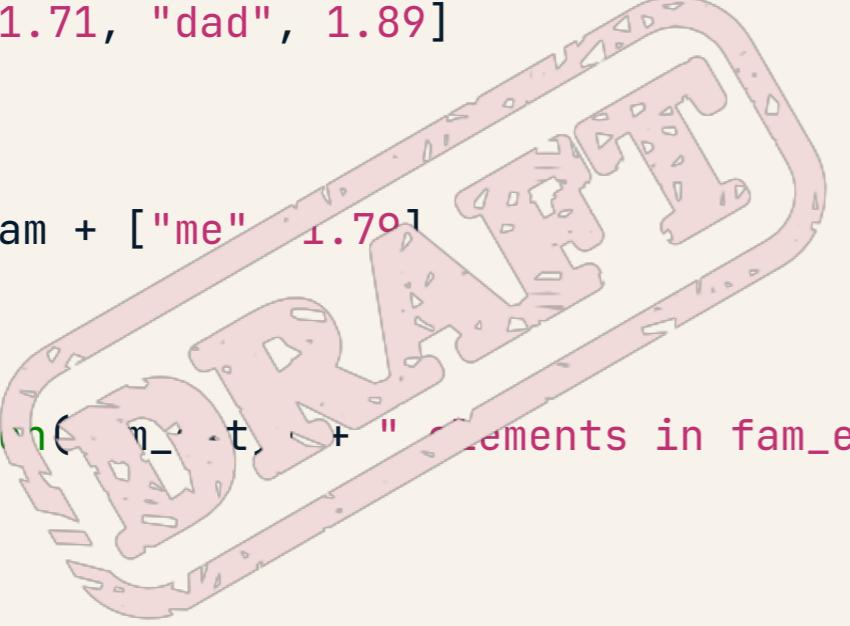
```
from numpy import array

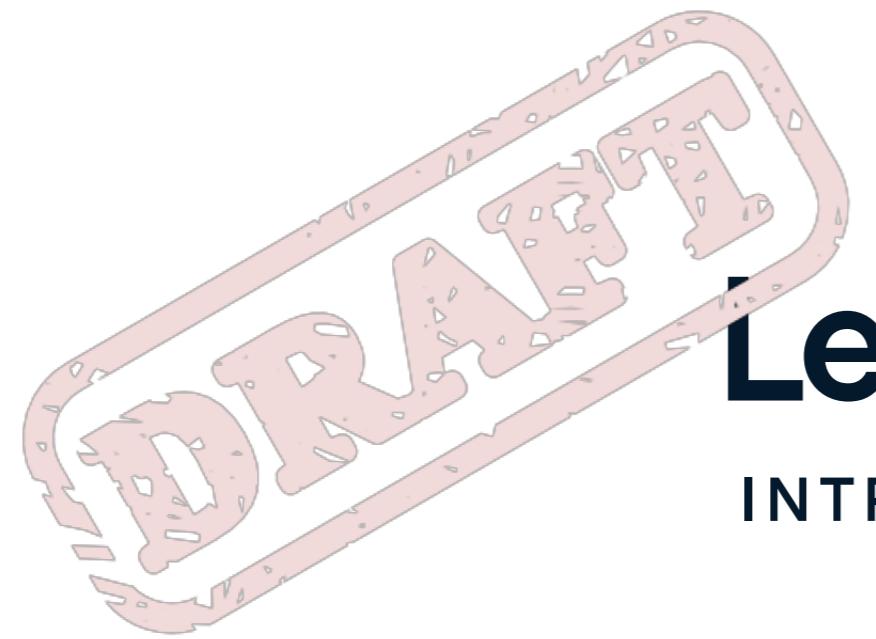
fam = ["liz", 1.73, "emma", 1.7,
       "mom", 1.71, "dad", 1.7]
...
fam_ext = fam + ["me", 70]
...
print(str(len(fam_ext)) + " elements in fam_ext")
...
np_fam = array(fam_ext)
```

- Using NumPy, but not very clear

import numpy

```
import numpy as np\n\nfam = ["liz", 1.73, "emma", 1.68,\n       "mom", 1.71, "dad", 1.89]\n\n...\n\nfam_ext = fam + ["me", 1.70]\n\n...\n\nprint(str(len(fam_ext)) + " elements in fam_ext")\n\n...\n\nnp_fam = np.array(fam_ext) # Clearly using NumPy
```





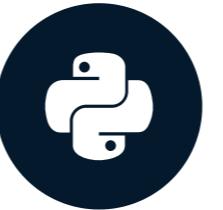
Let's practice!

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, 63.1, 88.4, 63.7]  
weight
```

```
[65.4, 59.2, 63.1, 88.4, 63.7]
```

```
weight / height
```

```
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, 1.7])
```

```
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
```

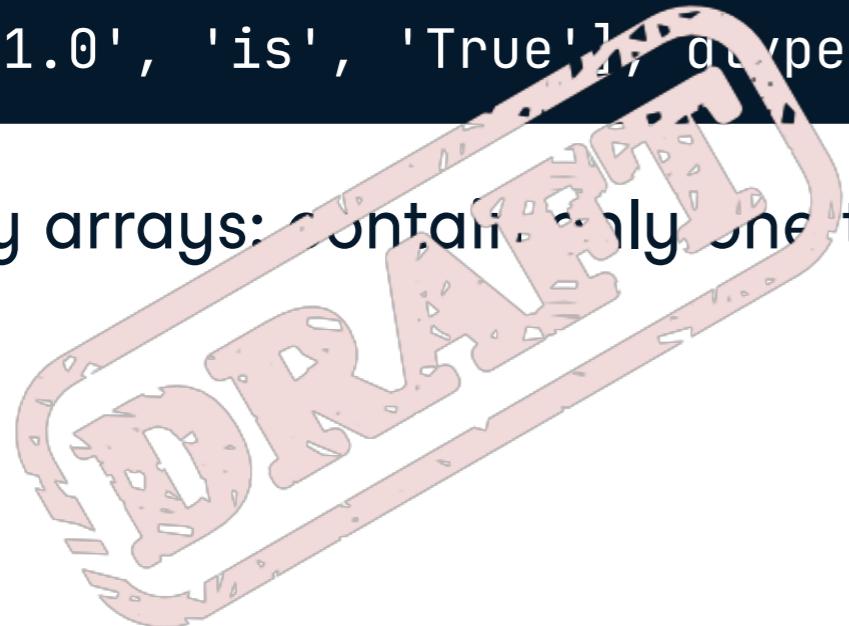
```
array([21.85171573, 20.97505669, 21.75028214, 24.7473475 , 21.44127836])
```

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]
```

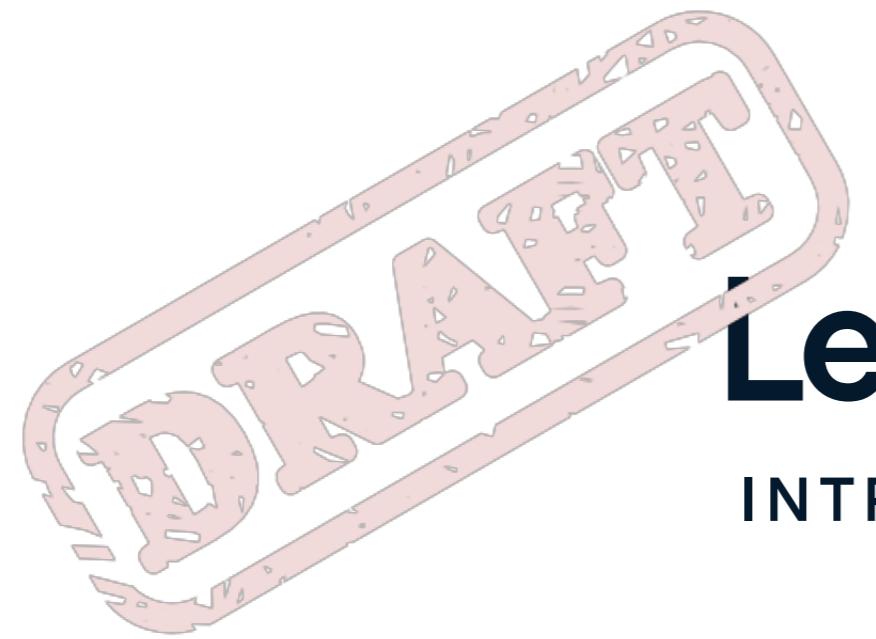
```
20.975
```

```
bmi > 23
```

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

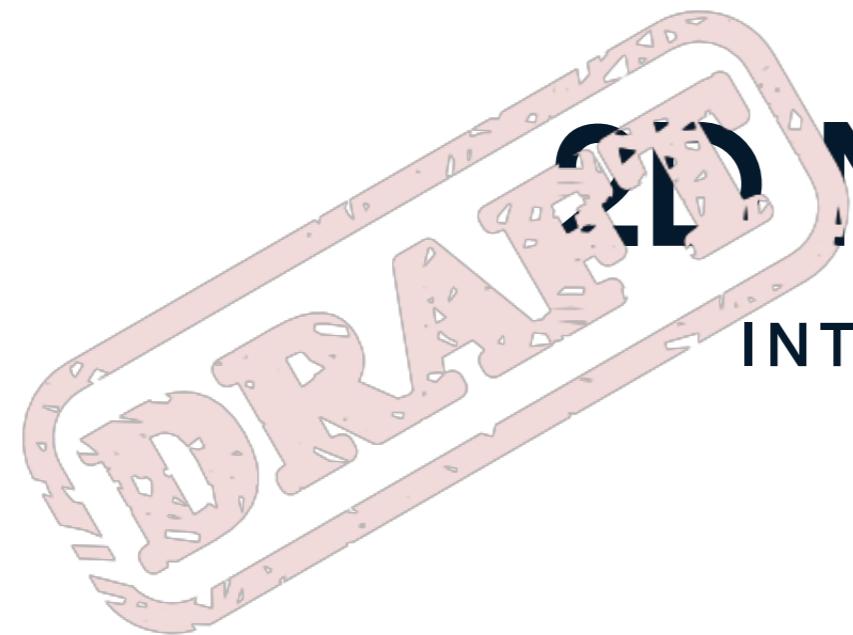
```
bmi[bmi > 23]
```

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



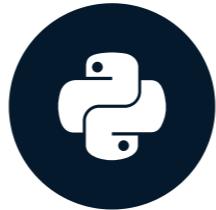
Let's practice!

INTRODUCTION TO PYTHON



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,   63.5,   81.4,   68.7]])  1
```

```
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,   63.5,   81.4,   68.7]]) 1
```

np_2d[0][2]

1.71

np_2d[0, 2]

1.71

Subsetting

```
0      1      2      3      4
```

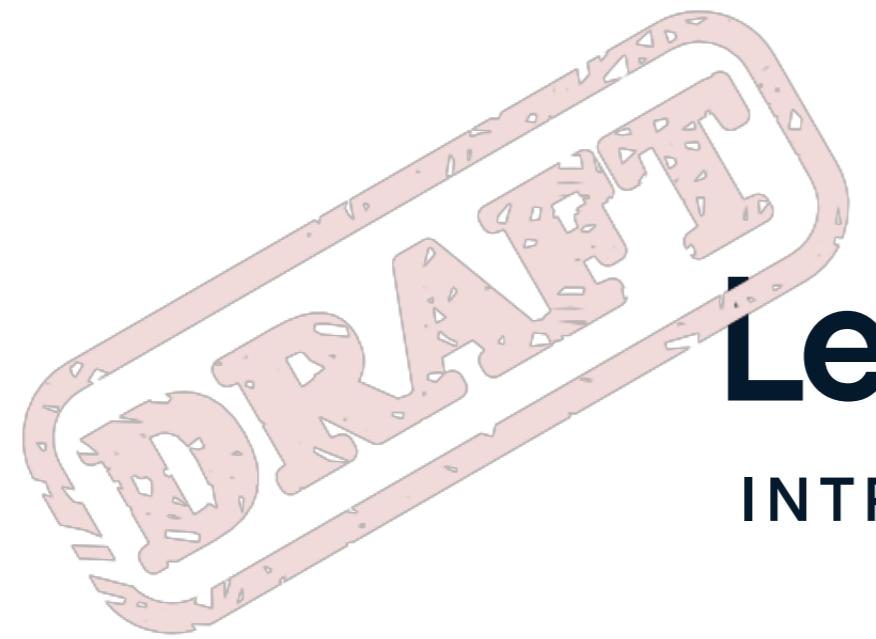
```
array([[ 1.73,   1.68,   1.71,   1.89,   1.79],  
       [ 65.4,   59.2,   63.6,   88.4,   68.7]])
```

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

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

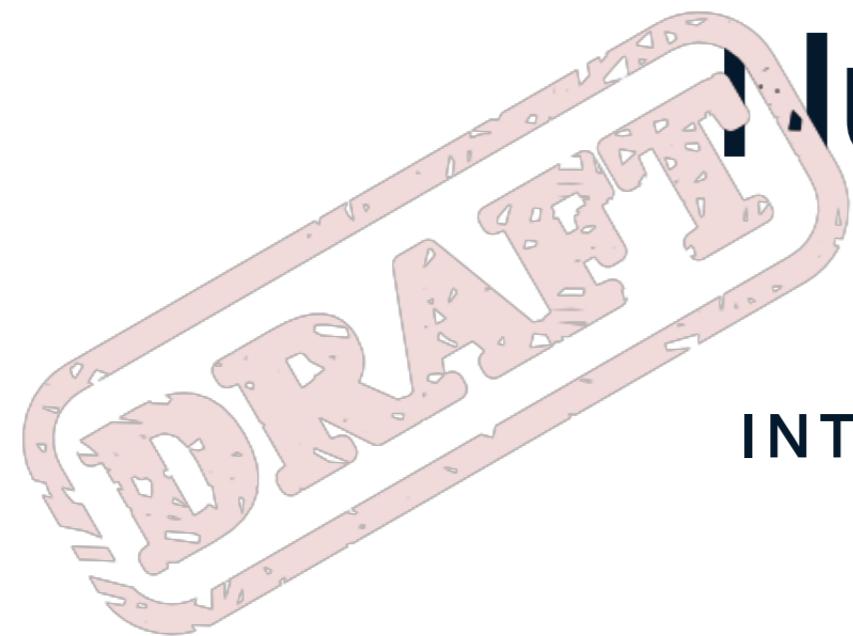
```
np_2d[1, :]
```

```
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, 71.7,  
       [-37.3, -3.],  
       [-3.8, 55.0],  
       ...],  
      [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.        , -0.018021,
       [-0.01803, 1.        ]])
```

```
np.std(np_city[:, 1])
```

```
0.1992
```

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

Generate data

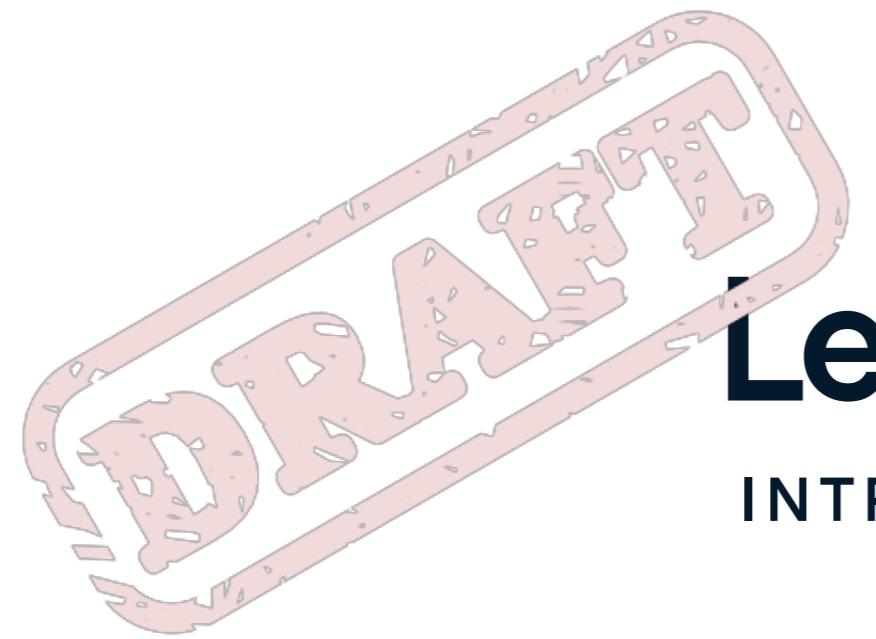
- Arguments for `np.random.normal()`

- 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