INTRO TO PYTHON FOR DATA SCIENCE

Hello Python!

What you will learn

- Python
- Specifically for Data Science
- Store data
- Manipulate data
- Tools for data analysis

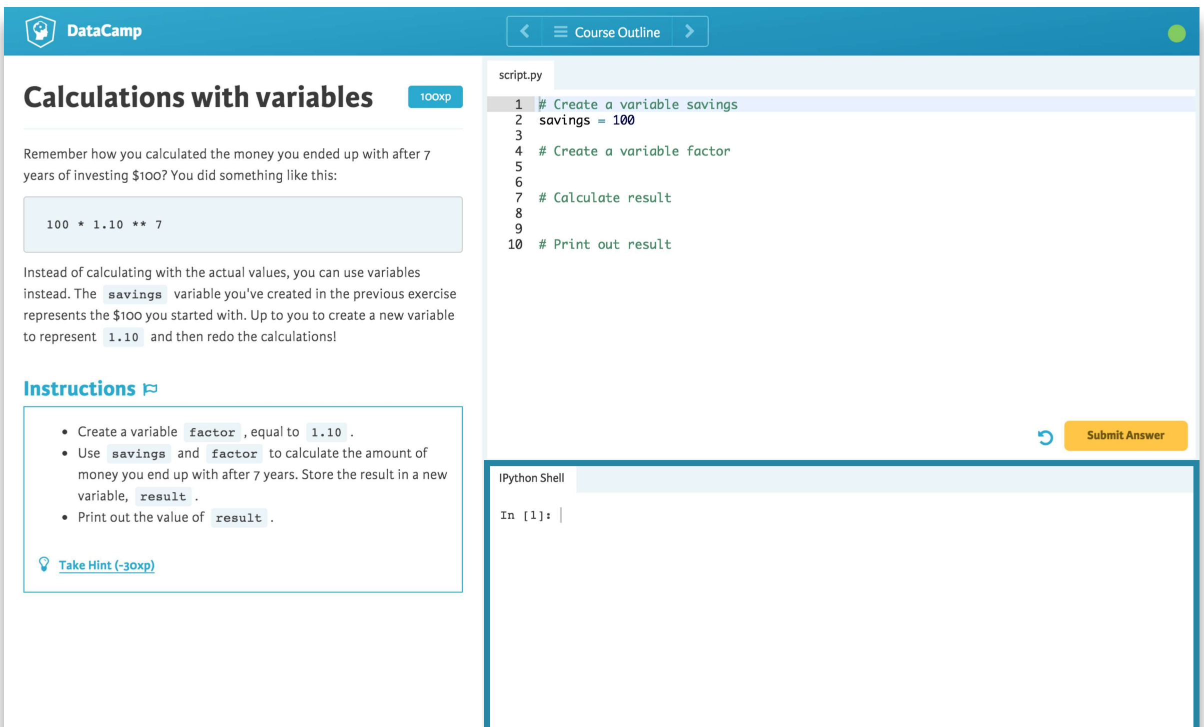
How you will learn

Python

- Guido Van Rossum
- 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



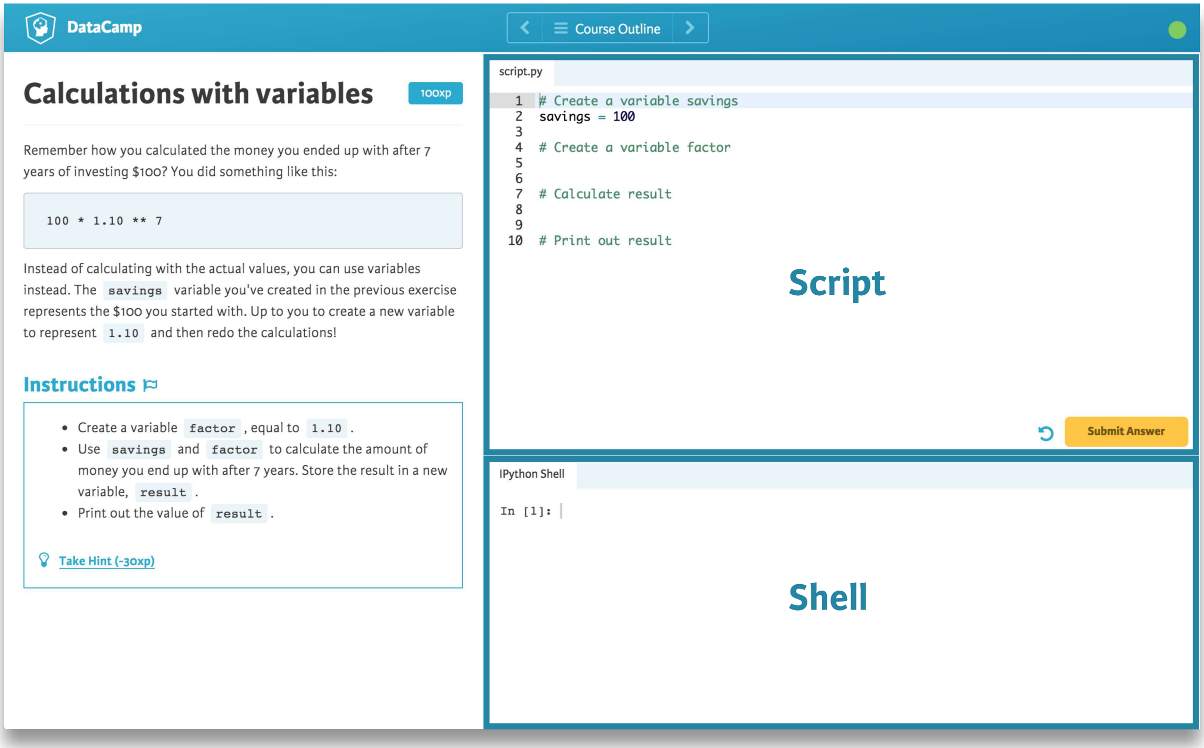
IPython Shell

Python Script

- Text Files - .py
- List of Python Commands
- Similar to typing in IPython Shell

Python Script

DataCamp Interface



Let’s practice!

Variables and Types

Variable

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

```
In [1]: height = 1.79

In [2]: weight = 68.7

In [3]: height
Out[3]: 1.79
```

Calculate BMI

```
In [1]: height = 1.79

In [2]: weight = 68.7

In [3]: height
Out[3]: 1.79

In [4]: 68.7 / 1.79 ** 2
Out[4]: 21.4413

In [5]: weight / height ** 2
Out[5]: 21.4413

In [6]: bmi = weight / height ** 2

In [7]: bmi
Out[7]: 21.4413
```

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

Reproducibility

my_script.py

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

```
Output:
21.4413
```

Reproducibility

my_script.py

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

```
Output:
23.1578
```

Python Types

```
In [8]: type(bmi)
Out[8]: float

In [9]: day_of_week = 5

In [10]: type(day_of_week)
Out[10]: int
```

Python Types (2)

```
In [11]: x = "body mass index"

In [12]: y = 'this works too'

In [13]: type(y)
Out[13]: str

In [14]: z = True

In [15]: type(z)
Out[15]: bool
```

Python Types (3)

```
In [16]: 2 + 3
Out[16]: 5

In [17]: 'ab' + 'cd'
Out[17]: 'abcd'
```

Different type = different behavior!

Let's practice!

INTRO TO PYTHON FOR DATA SCIENCE

Python Lists

Python Data Types

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

```
In [1]: height = 1.73

In [2]: tall = True
```

- Each variable represents <u>single</u> value

Problem

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

```
In [3]: height1 = 1.73

In [4]: height2 = 1.68

In [5]: height3 = 1.71

In [6]: height4 = 1.89
```

- Inconvenient

Python List

[a, b, c]

```
In [7]: [1.73, 1.68, 1.71, 1.89]
Out[7]: [1.73, 1.68, 1.71, 1.89]

In [8]: fam = [1.73, 1.68, 1.71, 1.89]

In [9]: fam
Out[9]: [1.73, 1.68, 1.71, 1.89]
```

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

Python List

[a, b, c]

```
In [10]: fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]

In [11]: fam
Out[11]: ['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]


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

Python List

[a, b, c]

```
In [10]: fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]

In [11]: fam
Out[11]: ['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]


In [11]: fam2 = [["liz", 1.73],
                 ["emma", 1.68],
                 ["mom", 1.71],
                 ["dad", 1.89]]

In [12]: fam2
Out[12]: [['liz', 1.73], ['emma', 1.68],
          ['mom', 1.71], ['dad', 1.89]]
```

List type

```
In [13]: type(fam)
Out[13]: list

In [14]: type(fam2)
Out[14]: list
```

- Specific functionality
- Specific behavior

Let's practice!

Subsetting Lists

Subsetting lists

```
In [1]: fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]

In [2]: fam
Out[2]: ['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
    index:   0      1       2       3      4      5      6      7
```

"zero-based indexing"

Subsetting lists

```
In [1]: fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]

In [2]: fam
Out[2]: ['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
    index:  0      1       2       3     4      5      6      7


In [3]: fam[3]
Out[3]: 1.68
```

Subsetting lists

```
In [1]: fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]

In [2]: fam
Out[2]: ['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
    index:  0      1      2       3      4      5     6      7


In [3]: fam[3]
Out[3]: 1.68

In [4]: fam[6]
Out[4]: 'dad'
```

Subsetting lists

```
In [1]: fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]

In [2]: fam
Out[2]: ['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
    index:  0      1      2       3      4     5      6      7
           -8     -7     -6      -5     -4    -3     -2     -1

In [3]: fam[3]
Out[3]: 1.68

In [4]: fam[6]
Out[4]: 'dad'

In [5]: fam[-1]
Out[5]: 1.89
```

Subsetting lists

```
In [1]: fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]

In [2]: fam
Out[2]: ['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
   index:   0       1      2       3      4     5      6      7
           -8      -7     -6      -5     -4    -3     -2     -1

In [3]: fam[3]
Out[3]: 1.68

In [4]: fam[6]
Out[4]: 'dad'

In [5]: fam[-1]
Out[5]: 1.89

In [6]: fam[-2]
Out[6]: 'dad'
```

List slicing

```
In [7]: fam
Out[7]: ['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
              0       1       2       3      4     5      6      7

In [8]: fam[3:5]
Out[8]: [1.68, 'mom']
```

[start : end]

inclusive **exclusive**

List slicing

```
In [7]: fam
Out[7]: ['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
                0     1       2      3      4      5      6      7

In [8]: fam[3:5]
Out[8]: [1.68, 'mom']

In [9]: fam[1:4]
Out[9]: [1.73, 'emma', 1.68]
```

[start : end]

inclusive **exclusive**

List slicing

```
In [7]: fam
Out[7]: ['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
           0       1       2       3      4      5      6     7

In [8]: fam[3:5]
Out[8]: [1.68, 'mom']

In [9]: fam[1:4]
Out[9]: [1.73, 'emma', 1.68]

In [10]: fam[:4]
Out[10]: ['liz', 1.73, 'emma', 1.68]
```

List slicing

```
In [7]: fam
Out[7]: ['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
           0      1      2       3      4     5      6      7

In [8]: fam[3:5]
Out[8]: [1.68, 'mom']

In [9]: fam[1:4]
Out[9]: [1.73, 'emma', 1.68]

In [10]: fam[:4]
Out[10]: ['liz', 1.73, 'emma', 1.68]

In [11]: fam[5:]
Out[11]: [1.71, 'dad', 1.89]
```

Let’s practice!

INTRO TO PYTHON FOR DATA SCIENCE

Manipulating Lists

List Manipulation

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

Changing list elements

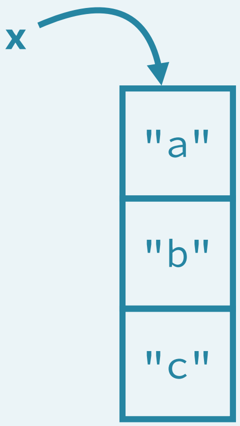
```
In [1]: fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]

In [2]: fam
Out[2]: ['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]

In [3]: fam[7] = 1.86

In [4]: fam
Out[4]: ['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.86]

In [5]: fam[0:2] = ["lisa", 1.74]

In [6]: fam
Out[6]: ['lisa', 1.74, 'emma', 1.68, 'mom', 1.71, 'dad', 1.86]
```

Adding and removing elements

```
In [7]: fam + ["me", 1.79]
Out[7]: ['lisa', 1.74,'emma', 1.68,
                         'mom', 1.71, 'dad', 1.86, 'me', 1.79]

In [8]: fam_ext = fam + ["me", 1.79]



In [9]: del(fam[2])

In [10]: fam
Out[10]: ['lisa', 1.74, 1.68, 'mom', 1.71, 'dad', 1.86]

In [11]: del(fam[2])

In [12]: fam
Out[12]: ['lisa', 1.74, 'mom', 1.71, 'dad', 1.86]
```

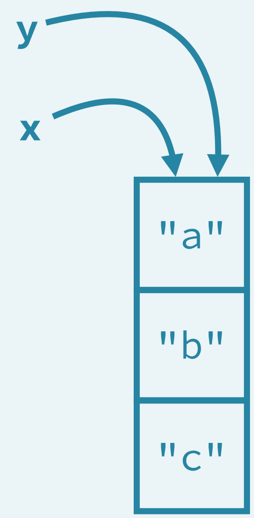
Behind the scenes (1)

```
In [13]: x = ["a", "b", "c"]

In [14]: y = x
```



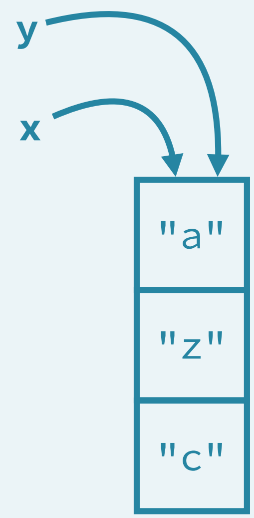
Behind the scenes (1)

```
In [13]: x = ["a", "b", "c"]

In [14]: y = x

In [15]: y[1] = "z"

In [16]: y
Out[16]: ['a', 'z', 'c']

In [17]: x
Out[17]: ['a', 'z', 'c']
```



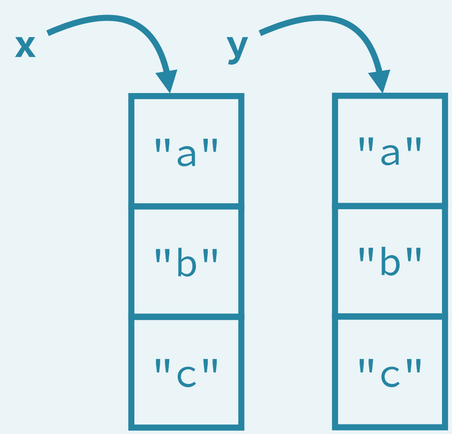
Behind the scenes (1)

```
In [13]: x = ["a", "b", "c"]

In [14]: y = x

In [15]: y[1] = "z"

In [16]: y
Out[16]: ['a', 'z', 'c']

In [17]: x
Out[17]: ['a', 'z', 'c']
```



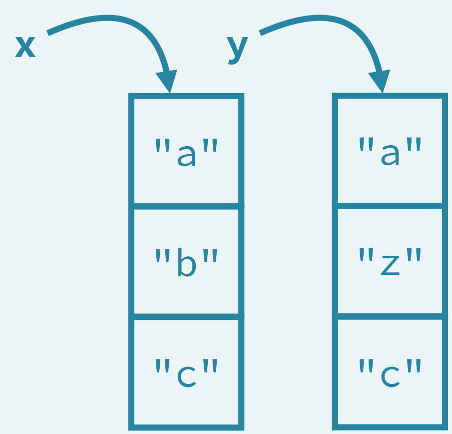
Behind the scenes (2)

```
In [18]: x = ["a", "b", "c"]

In [19]: y = list(x)

In [20]: y = x[:]

In [21]: y[1] = "z"
```



Behind the scenes (2)

```
In [18]: x = ["a", "b", "c"]

In [19]: y = list(x)

In [20]: y = x[:]

In [21]: y[1] = "z"

In [22]: x
Out[22]: ['a', 'b', 'c']
```



Let's practice!

EXERCISE

IPYTHON SHELL

Different ways of importing

There are several ways to import packages and modules into Python. Depending on the import call, you'll have to use different Python code.

Suppose you want to use the function `inv()`, which is in the `linalg` subpackage of the `scipy` package. You want to be able to use this function as follows:

```
my_inv([[1,2], [3,4]])
```

Which `import` statement will you need in order to run the above code without an error?

INSTRUCTIONS 50 XP

Possible Answers

- `import scipy` press 1
- `import scipy.linalg` press 2
- `from scipy.linalg import my_inv` press 3
- `from scipy.linalg import inv as my_inv` press 4

EXERCISE

Blend it all together

In the last few exercises you've learned everything there is to know about heights and weights of baseball players. Now it's time to dive into another sport: soccer.

You've contacted FIFA for some data and they handed you two lists. The lists are the following:

```
positions = ['GK', 'M', 'A', 'D', ...]
heights = [191, 184, 185, 180, ...]
```

Each element in the lists corresponds to a player. The first list, `positions`, contains strings representing each player's position. The possible positions are: `'GK'` (goalkeeper), `'M'` (midfield), `'A'` (attack) and `'D'` (defense). The second list, `heights`, contains integers representing the height of the player in cm. The first player in the lists is a goalkeeper and is pretty tall (191 cm).

You're fairly confident that the median height of goalkeepers is higher than that of other players on the soccer field. Some of your friends don't believe you, so you are determined to show them using the data you received from FIFA and your newly acquired Python skills.

INSTRUCTIONS 100 XP

SCRIPT.PY

```
# heights and positions are av

# Import numpy
import numpy as np

# Convert positions and height
np_heights = np.array(heights)
np_positions = np.array(positi

# Heights of the goalkeepers:
gk_heights = np_heights[np_pos

# Heights of the other players
```

IPYTHON SHELL SLIDES