Introduction to Python for DataScience - Jupyter notebook edition

August 5, 2018

1 Introduction to Python for DataScience: Jupyter notebook

2 Introduction

The following documentoutlines the written portion of the lessons from DataCamp's "Intro to Python for Data Science" course. This is a beginner course that requires little knowledge in Python programming.

As a note: All text is completely copied and pasted from the course. There are insances where the document refers to the "editor on the right", please note, that in this notebook document all of the instances are noted in the "Python-chunks" (areas containing working python-code), which occurs below the text, rather than to the right.

If you have this document open on "R-Notebook", simply click "run" -> "Run all" (Or just press 'ctrl + alt + r'), let the "r-chunks" run (This might take a bit of time) then click "Preview". All necssary data is embedded within the code, no need to set a working directory or open an R-project. Additionally, will need to ensure that python is set as a usable path in your windows environment because this will be executed through R-studio. Here is a helpful video. If you have utilized Jupyter notebooks, then it is the same process for getting those connected.

This document was created by Neil Yetz on 03/15/2018. Please send any questions or concerns in this document to Neil at ndyetz@gmail.com

3 Chapter 1: Python Basics An introduction to the basic concepts of Python. Learn how to use Python both interactively and through a script. Create your first variables and acquaint yourself with Python's basic data types.

3.1 The Python Interface

In the Python script on the right, you can type Python code to solve the exercises. If you hit Submit Answer, your python script (`script.py`) is executed and the output is shown in the IPython Shell. DataCamp checks whether your submission is correct and gives you feedback.

You can hit Submit Answer as often as you want. If you're stuck, you can click Get Hint, and ultimately Get Solution.

You can also use the IPython Shell interactively by simply typing commands and hitting Enter. When you work in the shell directly, your code will not be checked for correctness so it is a great way to experiment

Instructions

Experiment in the IPython Shell; type `5 / 8`, for example. Add another line of code to the Python script: `print(7 + 10)`. Hit Submit Answer to execute the Python script and receive feedback.

```
In [49]: # Example, do not modify!
         print(5 / 8)

         # Put code below here
         print(7 + 10)
```

```
0.625
17
```

3.2 Any comments?

Something that Filip didn't mention in his videos is that you can add comments to your Python scripts. Comments are important to make sure that you and others can understand what your code is about.

To add comments to your Python script, you can use the `#` tag. These comments are not run as Python code, so they will not influence your result. As an example, take the comment on the right, `# Just testing division`; it is completely ignored during execution.

Instructions

Above the `print(7 + 10)`, add the comment `# Addition works too`.

```
In [50]: # Just testing division
         print(5 / 8)

         # Addition works too
         print(7 + 10)
```

```
0.625
17
```

3.3 Python as a calculator

Python is perfectly suited to do basic calculations. Apart from addition, subtraction, multiplication and division, there is also support for more advanced operations such as:

Exponentiation: `**`. This operator raises the number to its left to the power of the number to its right. For example `4**2` will give `16`.

Modulo: `%`. This operator returns the remainder of the division of the number to the left by the number on its right. For example `18 % 7` equals `4`.

The code in the script on the right gives some examples.

Instructions

Suppose you have $100, which you can invest with a 10% return each year. After one year, it's 100?1.1=110 dollars, and after two years it's 100?1.1?1.1=121. Add code on the right to calculate how much money you end up with after 7 years.

```
In [51]: # Addition and subtraction
         print(5 + 5)
         print(5 - 5)

         # Multiplication and division
         print(3 * 5)
         print(10 / 2)

         # Exponentiation
         print(4 ** 2)

         # Modulo
         print(18 % 7)

         # How much is your $100 worth after 7 years?
         print(100*1.1**7)
```

```
10
0
15
5.0
16
4
194.87171000000012
```

3.4 Variable Assignment

In Python, a variable allows you to refer to a value with a name. To create a variable use `=`, like this example:

```
x = 5
```

You can now use the name of this variable, `x`, instead of the actual value, `5`.
Remember, = in Python means assignment, it doesn't test equality!
INSTRUCTIONS
Create a variable `savings` with the value `100`.
Check out this variable by typing `print(savings)` in the script.

```
In [52]: # Create a variable savings
         savings = 100

         # Print out savings
         print(savings)
```

```
100
```

3.5 Calculations with variables

Remember how you calculated the money you ended up with after 7 years of investing $100? You did something like this:

```
100 * 1.10 ** 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 `1.10` and then redo the calculations!

INSTRUCTIONS

Create a variable `factor`, equal to `1.10`.

Use `savings` and `factor` to calculate the amount of money you end up with after 7 years. Store the result in a new variable, `result`.

Print out the value of `result`.

```
In [53]: # Create a variable savings
         savings = 100

         # Create a variable factor
         factor = 1.10

         # Calculate result
         result = (savings * factor**7)

         # Print out result
         print(result)
```

```
194.87171000000012
```

3.6 Other variable types

In the previous exercise, you worked with two Python data types:

`int`, or integer: a number without a fractional part. `savings`, with the value `100`, is an example of an integer.

`float`, or floating point: a number that has both an integer and fractional part, separated by a point. `factor`, with the value `1.10`, is an example of a float.

Next to numerical data types, there are two other very common data types:

`str`, or string: a type to represent text. You can use single or double quotes to build a string.

`bool`, or boolean: a type to represent logical values. Can only be `True` or `False` (the capitalization is important!).

INSTRUCTIONS

Create a new string, `desc`, with the value `"compound interest"`.

Create a new boolean, `profitable`, with the value `True`.

```
In [56]: # Create a variable desc
         desc = "compound interest"

         # Create a variable profitable
         profitable = True
```

In [ ]:

In [54]:
```
import os as os
import pandas as pd
os.chdir("C:/Users/Neil/Desktop/Git/Data_Camp/Introduction to Python for Data Science")

print(os.getcwd())
```

```
C:\Users\Neil\Desktop\Git\Data_Camp\Introduction to Python for Data Science
```

In [55]:
```
baseball = pd.read_csv("baseball.csv")
fifa =  pd.read_csv("fifa.csv", skipinitialspace=True, usecols=[0,2,3,4,5,6,7,8,9,10,
positions = list(fifa.position)
heights = list(fifa.height)
```

In [ ]:

In [ ]: