Data Foundations: Python 101 Part I

Instructor: Anthony Rios

Outline

Covered Readings

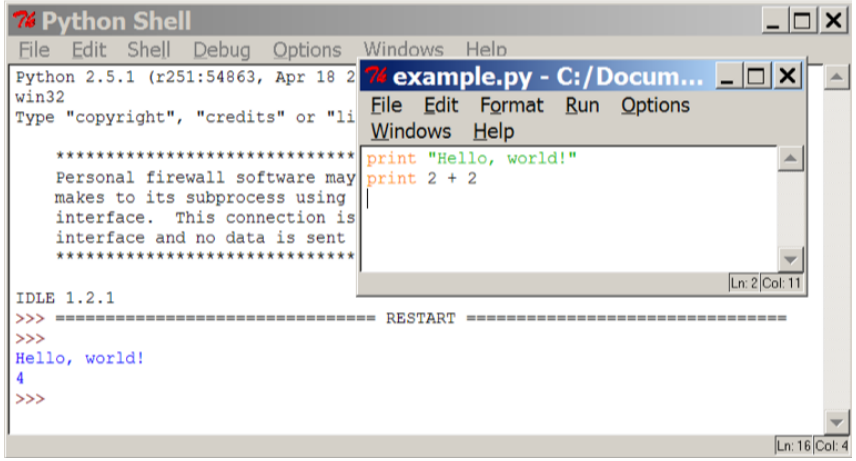
Python for Everybody: Exploring Data In Python 3
by Charles Russell Severance
Free URL: **https://www.py4e.com/book**

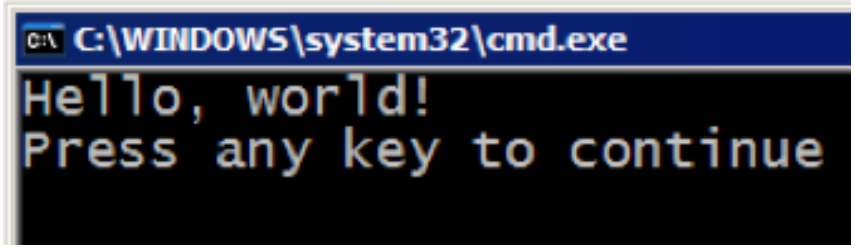
Chapters covered: **1**, **2**, **3**, and **6**

Do not panic! Ask questions on Blackboard! If you are not comfortable, please email me!

Programming Basics

- **code** or **source code**: The sequence of instructions in a program.
- **syntax**: The set of legal structures and commands that can be used in a particular programming language.
- **output**: The messages printed to the user by a program.
- **console** or **terminal**: The text box onto which output is printed.
  - Some source code editors pop up the console as an external window, and others contain their own console window.





Python

- Open source gener-purpose language
- Object Oriented, Procedural, Functional
- Easy to interface with C/Objective C/Java/Fortran
- Great interactive environment. Can also be ran in "batch mode".

- Download: `http://www.python.org`
- Documentation: `http://www.python.org/doc/`

The Python Interpreter

- **Interactive interface to Python**
  anthonyrios@Macbook:∼$ python
  Python 3.6.4 —Anaconda, Inc.— (default, Jan 16 2018, 12:04:33)
  GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)on darwin
  Type "help", "copyright", "credits" or "license" for more information.
  >>>

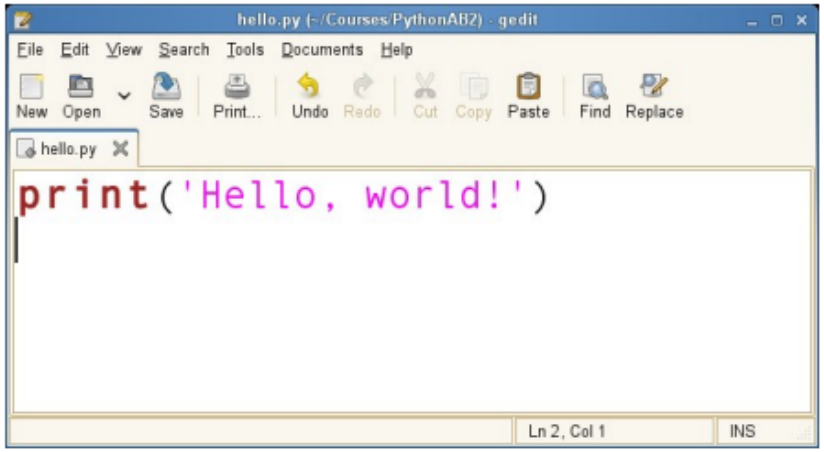
- **Python interpreter evaluates inputs**
  >>>$3 * (7 + 2)$
  27

- **Python prompts start with** ">>>"

- **To exit python**: CTRL-D

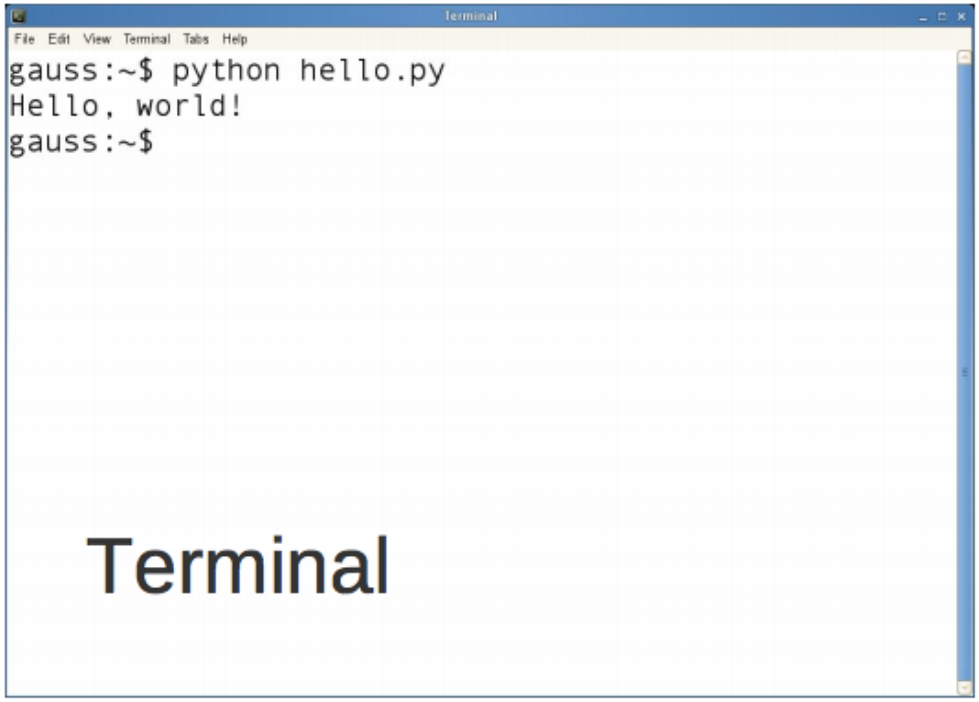
Running Python Scripts

Read / edit the script



`gedit`

Run the script



Terminal

Exercise 1

1. Open Jupyter Lab
2. Download today's exercises from Blackboard.
3. Load the notebook using in Google Colab
4. Complete Exercise 1


5 minutes

Demo

Demo

Types of Values

- Numbers
  - Integers (Whole numbers)
  - Floating point numbers (Floats)
- Text
- Boolean Values
  - True
  - False

Integers

Integers $\in \mathbb{Z}$ where $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, 3, \ldots\}$

```
>>> 4+2
6

>>> 3 + 5          # spaces around the "+" are ignored
8

>>> 4 - 2
2

>>> 3 - 5
-2
```

Integers

```
>>> 4*2
8
>>> 3 * 5          # multiplication uses a "*" instead of a "×"
15
>>> 4/2          # Division uses a "/" instead of a "÷" (python 2.7)
2
>>> 5/3          # For integers, division rounds down (python 2.7)
1
>>> -5 / 3          # Strictly down (python 2.7)
-2
>>> int(-5/3)          # cast to int (python 3+)
-1
>>> 5/2          # python 3+
2.5
```

Integers

```
>>> 4**2          # Raising to powers uses "4**" instead of "4^2
16

>>> 5 ** 3          # Spaces are allowed around "**", but not within it
125

>>> 4%2          # Remainder uses "%" (4 = 2 × 2 + 0)
0

>>> 5%3          # 5 = 1 × 3 + 2
2

>>> -5 % 3          # −5 = −2 × 3 + 1
1
```

Mathematical Operations

Python	Math	English
$a + b$	$a + b$	Addition
$a - b$	$a - b$	Subtraction
$a * b$	$a \times b$	Multiplication
a/b	$a \div b$	Division
$a * *b$	a^b	Exponentiation
$a\%b$	$a \; mod \; b$	Modulo/Remainder after division

Exercise 2

In Python, using the notebook loaded from Exercise 1, calculate (cast to int (i.e., int(4/3)) if you use python 3+):

1. (a) $12 + 4$ (b) $12 + 5$
2. (a) 12 - 4 (b) 12 - 5
3. (a) 12×4 (b) 12×5
4. (a) $12 \div 4$ (b) $12 \div 5$
5. (a) 12^4 (b) 12^5

Which answer is "wrong"?

 2 minutes

Floating Point Numbers $\notin \mathbb{R}$

```
>>> 1.0
1.0

>>> 0.5
0.5

>>> 0.25
0.25

>>> 0.1          # 0.1 is not stored accurately
0.1

0.1 + 0.1 + 0.1          # Floats are printed in decimal, but stored in binary
0.30000000000000004
```

Only **17 significant figures**

Floating Point Numbers

```
>>> 5.0 + 2.0
7.0
```

```
>>> 5.0 * 2.0
10.0
```

```
>>> 5.0 - 2.0
3.0
```

```
>>> 5.0 / 2.0
2.5
```

```
>>> 5.0 % 2.0
1.0
```

```
>>> 5.0 ** 2.0
25.0
```

Exercise 3

In Python, calculate:

1. $12.0 + 4.0$
2. $12.0 \div 4.0$
3. $25.0^{0.5}$
4. $5.0^{-1.0}$
5. $5.0 \div 2$

 2 minutes

Strings

```
>>> 'Hello world!'
'Hello world!'
```

```
>>> "Hello world!"
'Hello world!'
```

```
>>> print("Hello world!")
Hello world!
```

Why do we need quotes?

```
>>> 3          # Returns a number
3
```

- hello — is it a command or is it a string?
- print — is it a command or is it a string?
- 'hello' and 'print' are **strings**.

```
>>> hello      # hello is neither a string nor a python command
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'hello' is not defined

>>> print      # print is a command
<built-in function print>
```

Mixed Quotes and Joining Strings

```
>>> print('He said "Hello" to her.')
He said "Hello" to her.

>>> print("He said 'Hello' to her.")
He said 'Hello' to her.

>>> 'He said' + 'something.'
'He saidsomething.'

>>> 'He said ' + 'something.'
'He said something'
```

Repeated Text

```
>>> 'Bang! ' * 3
'Bang! Bang! Bang! '
```

```
>>> 3 * 'Bang! '
'Bang! Bang! Bang! '
```

Exercise 4

Predict what interactive Python will print when you type the following expressions. Then check.

- 'Hello, ' + "world!"
- 'Hello!' * 3
- " * 100000000000 # That is two adjacent single quotes
- '4' + '2'


3 minutes

Comparisons

Are the two values the same? 5+2 and 7.

Is one value bigger than the other? 5+2 and 8.

```
>>> 5 > 4          # Returns a boolean type
True

>>> 5.0 < 4.0
False

>>> 5 == 4          # need to use double equal sign
False

>>> 2 == 2
True
```

True and False are "boolean values", similar to numbers, string, etc.

Order matters: Parenthesis

```
>>> (2**2)**2 == 2**(2**2)
True
```

```
>>> (3**3)**3 == 3**(3**3)
False
```

All Numerical Comparisons

Python	Mathematics	Meaning
$x == y$	$x = y$	Equality
$x \; != y$	$x \neq y$	Not Equal
$x < y$	$x < y$	Less Than
$x <= y$	$x \leq y$	Less Than or Equal To
$x > y$	$x > y$	Greater Than
$x >= y$	$x \geq y$	Greater Than or Equal To

Combining Booleans

```
>>> 1 < 2 and 5 < 6        # True and True
True

>>> 1 < 2 and 5 > 6        # True and False
False

>>> 1 < 2 or 5 < 6         # True or True
True

>>> 1 < 2 or 5 > 6         # True or False
True

>>> 1 > 2 or 5 > 6         # False or False
False
```

Negating Booleans

```
>>> 1 > 2
False

>>> not 1 > 2          # Not False
True

>>> not False
True
```

Not Equal To...

```
>>> 1 == 2
False

>>> 1 != 2
True

>>> not 1 == 2
True
```

Exercise 5

Predict whether Python will print True or False before you type the following expressions.

1. $1 > 2$ or $2 > 1$
2. $1 > 2$ or not $2 > 1$
3. not True
4. $1 > 2$ or True


3 minutes

Variables

Variables implies attaching a name to a value

```
>>> 40 + 2          # An expression
42                  # The expression's value

>>> answer = 42          # Attatch the name "answer" to the value

>>> answer
42
```

Variables

```
>>> answer = 42
```

- answer – the name being attached (no quotes)
- = – A single equals sign
- 42 – The value being named

- == – Comparison: "are these equal?"
- = – Assignment: "attach the name on the left to the value on the right"

Variables

```
>>> answer = 42
>>> answer
42
```

```
>>> answer = 44 - 2
>>> answer
42
```

Variables

```
>>> answer = 42
>>> answer
42
>>> answer = answer - 2
>>> answer
40
```

Variables: Printing

```
>>> print('32')
32

>>> print('I am {} years old'.format(98))
I am 98 years old

>>> print('My name is {} and I am {} feet tall.'.format('Anthony', 7))
My name is Anthony and I am 7 feet tall.

>>> age = 98
>>> age
98
>>> print('I am {} years old'.format(age))
I am 98 years old
```

Variables: Printing

```
>>> pi = 3.14159
>>> pi
3.14159
```

```
>>> # {0} indexes format variable. .xf specifies number of
. . . decimal places.

>>> print('pi: {0:.0f}, pi: {0:.1f}, pi: {0:.3f}'.format(pi, 9))
pi: 3, pi: 3.1, pi: 3.141
```

Conditionals: if ... elif ... else

- So far, all the programs we have written executed **all** the statements they contained

- Suppose we want to write a program which asks the user to enter two numbers and then displays only the larger of the two

- This involves executing certain statements in **some circumstances**, and different statements in **other circumstances**

Conditionals: If ... elif ... else

- By default, the order of statement execution through a method is **linear**
- **one statement after the other** is executed, in textual order (top of page, downwards to end of page)
- Some programming statements **modify that order**, allowing us to:
  - decide whether or not to execute a particular statement
  - perform a statement over and over repetitively (while)
  - The order of statement execution is called the **flow of control**

Conditionals: If ... elif ... else

example.py

```
if True:
        print("if Statement Processed!")
else:
        print("else Statement Processed!")
```

```
anthony@MacBook:~$ python example.py
```

```
if Statement Processed!
```

Conditionals: If ... elif ... else

example.py

```
if False:
        print("if Statement Processed!")
elif True:
        print("elif Statement Processed!")
else:
        print("else Statement Processed!")
```

```
anthony@MacBook:~$ python example.py
elif Statement Processed!
```

Conditionals: If ... elif ... else

example.py

```
age = 65
if age < 45:
        print("You are less than 45 years old!")
elif age >= 45 and age < 65:
        print("You are between 45 and 65 years old!")
else:
        print("You are {} years old!".format(age))
```

anthony@MacBook:~$ python example.py

```
You are 65 years old!
```

Exercise 6

Write the if, elif, else statements to process a score between 0.0 and 1.0. If the score is out of range, print an error message. If the score is between 0.0 and 1.0, print a grade using the following table:

Score	Grade
≥ 0.9	A
≥ 0.8	B
≥ 0.7	C
≥ 0.6	D
< 0.6	B

10 minutes