Python Workshop Series Session 1: *Hello World!*

Mea Trahan

Research Computing

Slides: https://github.com/ResearchComputing/Python_Fall_2021

Installing Python

Check the file: *software_installation.pdf*
in the github repository!

Nuts and Bolts Overview of Python Programming

Who are we?

Why are we here?

Should You Be Here?

Target Audience:

(minimally) experienced programmers

Preparation:

Is Intel's distribution for Python 3.x installed?

If not: see installation instructions!

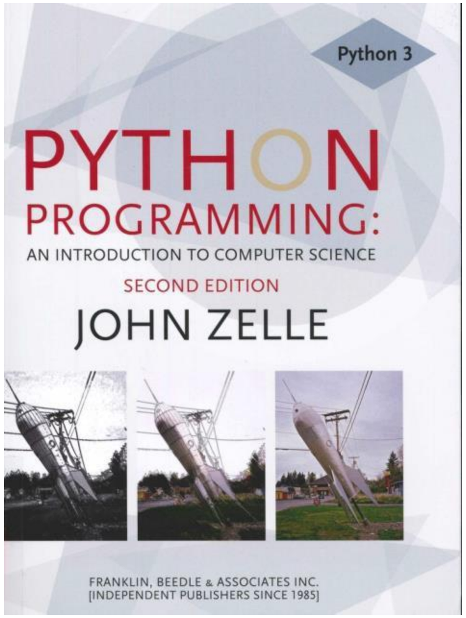
Workshop Series Outline

Oct 13	overview, variables, I/O	*Python Programming Fundamentals*
Oct 20	conditionals, functions	
Oct 27	loops, lists etc.	
Nov 3	objects, methods, modules	
Nov 10	Package management	*Python for Research*
Nov 17	NumPy (efficiency tips)	
Dec 1	Matplotlib (creating plots)	
Dec 8	H5Py (portable file format)	

Useful References

- Free Online Text
  - How to Think Like a Computer Scientist (Wentworth et al.)
  - http://openbookproject.net/thinkcs/python/english3e/index.html
  - Highly recommended

- Textbook
  - Python Programming:
    An Introduction to Computer Science (Zelle)


Python 3
PYTHON
PROGRAMMING:
AN INTRODUCTION TO COMPUTER SCIENCE
SECOND EDITION
JOHN ZELLE
FRANKLIN, BEEDLE & ASSOCIATES INC.
[INDEPENDENT PUBLISHERS SINCE 1985]

Today’s Session: Getting Around in Python

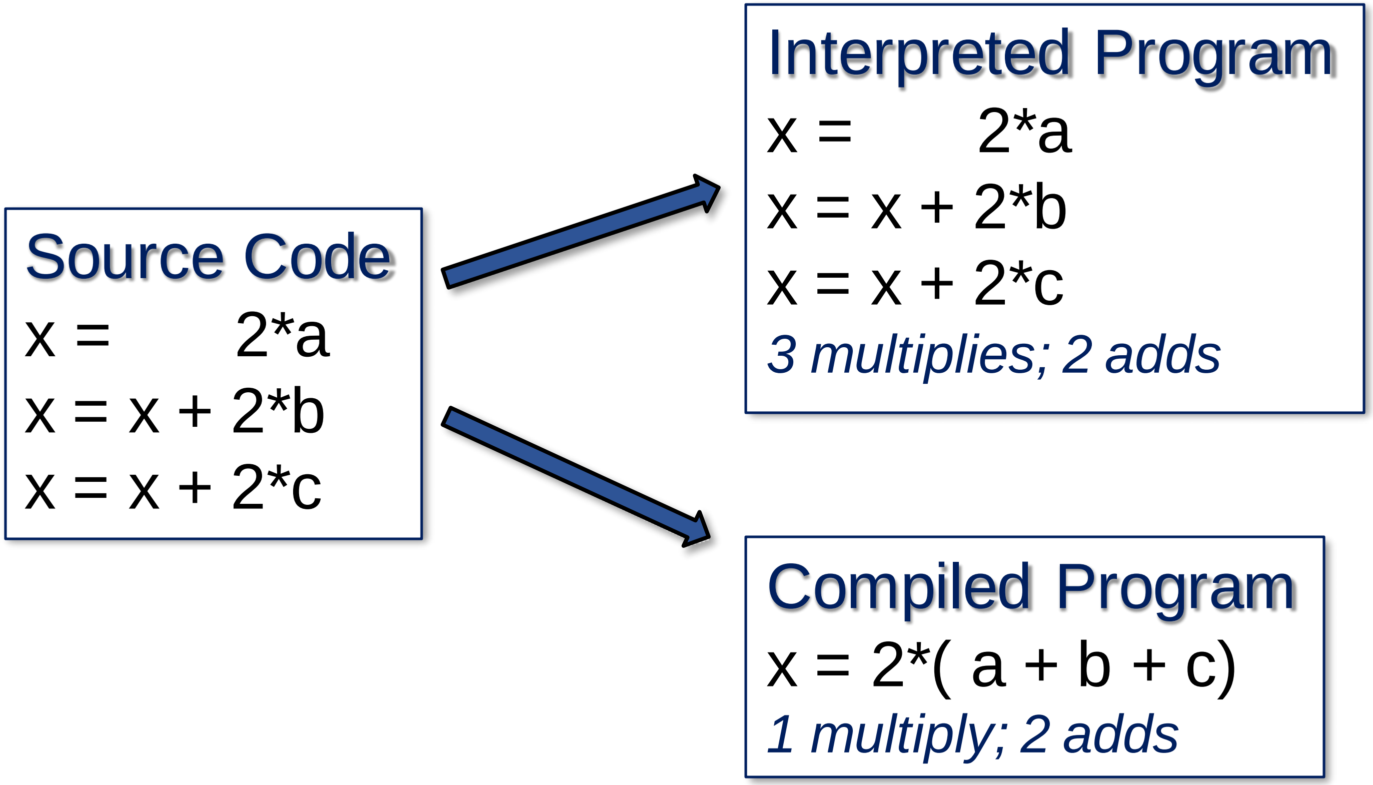
- Overview
- Running Python programs
- Variables and Arithmetic
- Basic I/O

- Recommended Reading:
  - Online Text Chapters: 1, 2, 13 (files)

Python, an Interpreted Language

- Python is an *interpreted* language
- Separate program (the interpreter) runs Python code.
- Interpreters execute code "naively." (line by line)
- Compilers take holistic approach. Interpreters do not.
- Efficiency losses when compared to compiled code.

Compilation vs. Interpretation



Source Code

```
x =       2*a
x = x + 2*b
x = x + 2*c
```

Interpreted Program

```
x =       2*a
x = x + 2*b
x = x + 2*c
```

3 multiplies; 2 adds

Compiled Program

```
x = 2*( a + b + c)
```

1 multiply; 2 adds

- The NumPy, Cython & F2Py packages help to overcome this limitation.

First Program

- Open a text editor and type:

```
print(“hello world”)
```

- Save the file as hello.py

- This is a complete Python program
  - … no semicolons, no brackets
  - … no “begin program,” no “end program,” etc.
  - .py extension customary (not required)

Running a Python Program

There are various ways to invoke the interpreter

- Command line (1): “*python hello.py*”
- Command line (2): ./hello.py (similar to bash script)
- Interactive sessions
- Jupyter Notebook (or other IDE)

…follow along as we try a few…

Command Line (1)

- Typical method for running Python programs.

- To use this method:

  1. Open a shell ("anaconda prompt" in Windows)
     - Activate your conda environment:

       ```
       conda activate idp
       ```

  1. Navigate to the folder containing hello.py

  2. Type: python hello.py

Command Line (2)

- Can execute code in fashion similar to a bash script
- Must add “shebang” sign #! and path to python interpreter:
- Try it (hello2.py):

```
#! path-to-python
print(“hello”)
```

1. which python
2. chmod +x hello2.py
3. ./hello2.py

Running the Interpreter Directly

- Similar to IDL and R interpreters
- Type python and enter statements one at a time
- Type exit() when finished (exit is a function)
- Let’s try it out…

- To run existing program within interactive session:
  - exec(open(“hello.py”).read( ))
  - This is clunky and *nonstandard*

Checking the Python Version

- We can access the python version within a program

```
#! /usr/bin/python
import sys
print(sys.version)
```

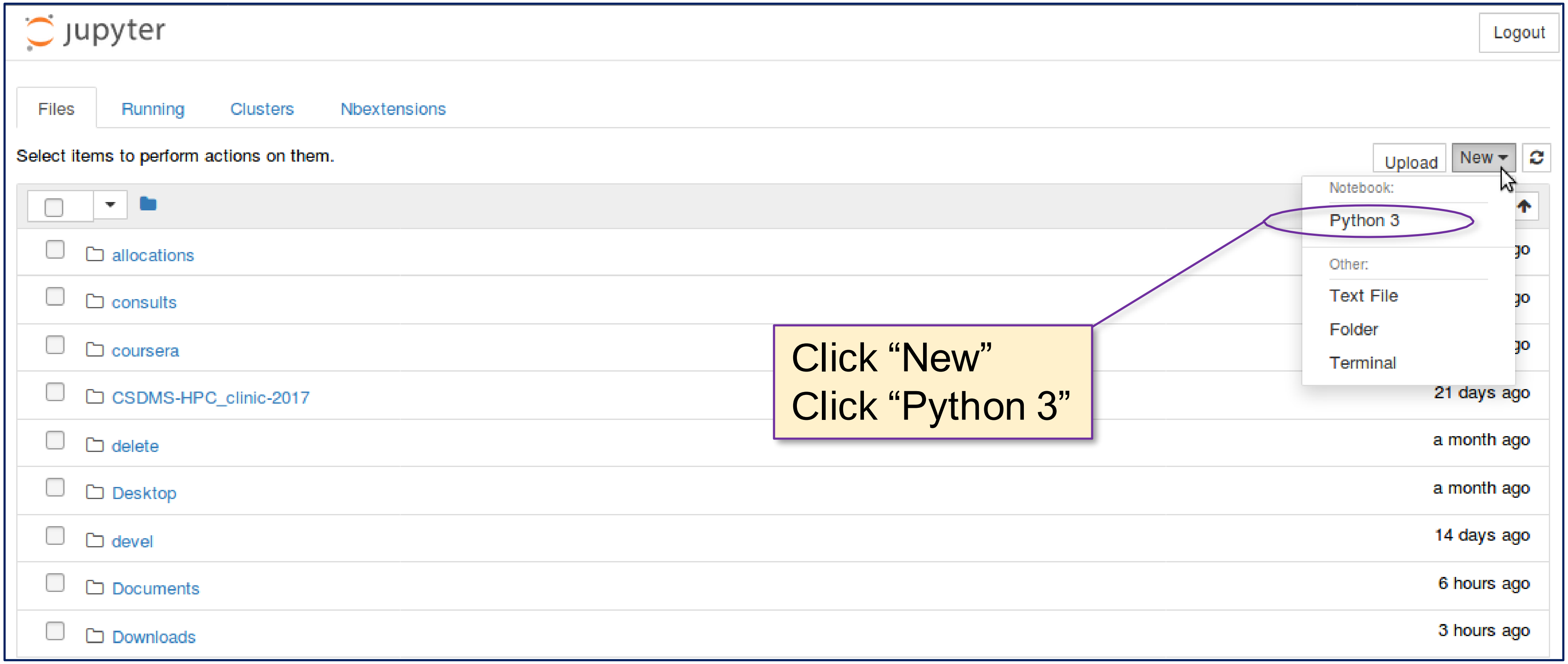
- Save this as ./hello3.py
- Type: chmod +x hello3.py
- ./hello3.py
- *sys* is a *module* (collection of functions & variables)
- *version* is a variable defined within the sys module

Jupyter Notebook

- Browser-integrated IDE
- Popular for interactive data-analysis
- I will use this throughout the workshop

- Let’s try out the notebook
  - Access your shell (“anaconda prompt” in Windows)
  - Type: conda activate pyclass21
  - Type: jupyter lab ← note the “Y”
  - Follow along…

The Jupyter Interface



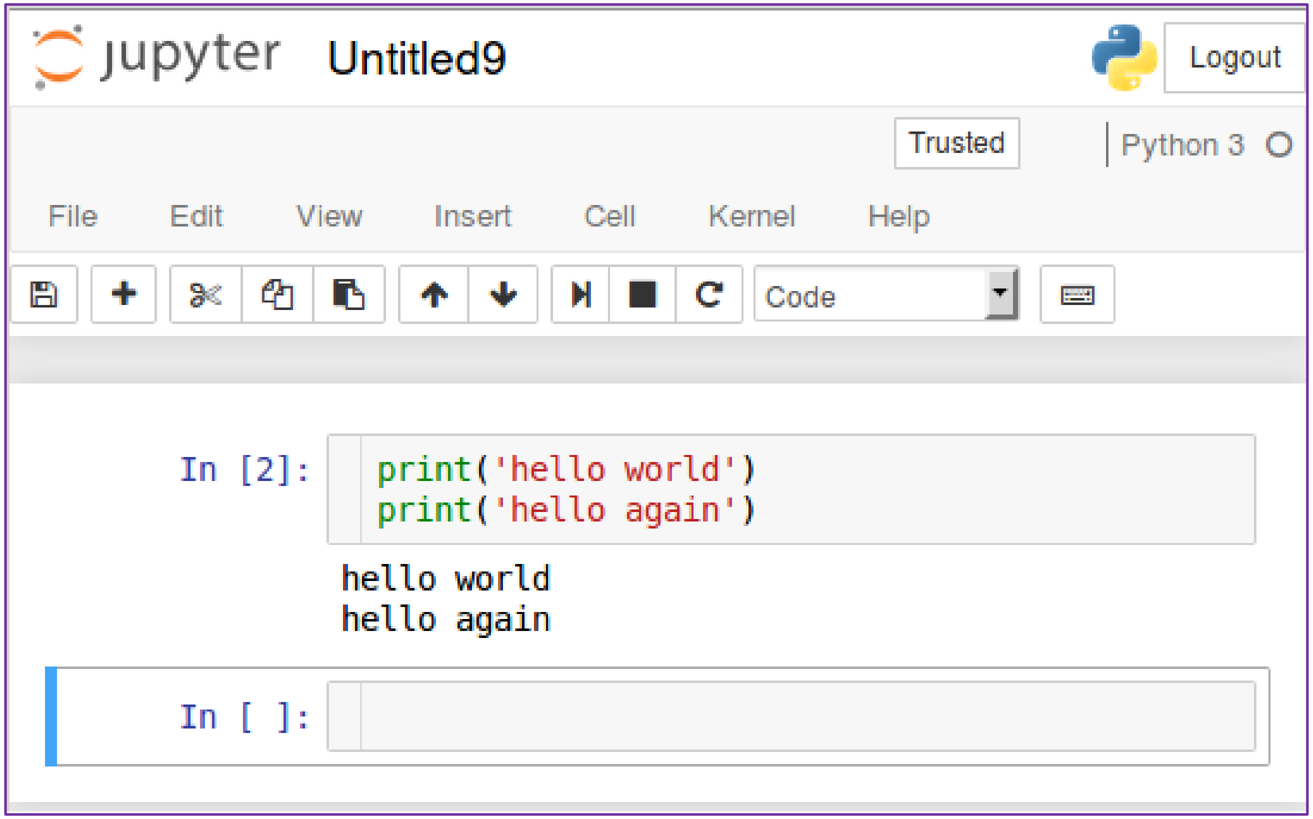
Select items to perform actions on them.

□ allocations	
□ consults	
□ coursera	
□ CSDMS-HPC_clinic-2017	21 days ago
□ delete	a month ago
□ Desktop	a month ago
□ devel	14 days ago
□ Documents	6 hours ago
□ Downloads	3 hours ago

Click “New”
Click “Python 3”

- Jupyter supports different interactive notebook types (e.g., R, Python 2.x etc.)
- Start a Python 3 notebook

The Jupyter Interface



```
In [2]: print('hello world')
        print('hello again')

hello world
hello again

In [ ]:
```

- Pressing ‘enter’ starts a new line
- Pressing ‘shift’ + ‘enter’ executes all lines of code within a cell

NOTE: Typical Program Structure

- Customary to include main program inside function
- Very helpful for complex and/or production codes

```
def main( ):
    print(“hello world”)

if __name__== “__main__”:
    main( )
```

- Program is a function definition + function call
- Unnecessary for our short exercises

Variables in Python

- Variables are not declared (implicitly typed)
- Variables are created at assignment time
- Variable type determined implicitly via assignment

• x = 2	int	
• y = 3.0	float	
• Z = “hello”	str	double or single quotes
• z = True	Bool	note capital “T” , “F” in False

- Beware: Python is CASE SENSITIVE (z is not Z)

- Check variable type using type function:
  - print(‘z is: ‘, type(z))

Arithmetic in Python

- Arithmetic in Python respects order of operations
- Addition: +
- Subtraction: -
- Multiplication: *
- Division: / (beware: returns float result)
- Floor Division : // (returns int or float; rounds down)
- Mod Division : % 3%2 → 1
- Exponentiation: ** 2**4 → 16
- Can concatenate strings using “+”
  - x = ‘hello’ + ‘ there’
  - print (x) → displays ‘hello there’

Print Function: Call Syntax

```
print( item1, item2, item3, …, sep = ‘ ‘, end= ‘\n’)
```

- item1, item2, item3
  - Comma-separated list of variables whose values you wish to display
- sep:
  - optional keyword parameter
  - separation string inserted between displayed values (defaults to whitespace)
- end:
  - optional keyword parameter
  - string appended to end of printed values (defaults to newline)

Calling Print

- Start with this:

```
name = ‘John’
age = 30
name2 = ‘Mary’
age2 = 31
```

- Then try these different print combinations:

```
print(name, ‘is’, age, ‘years old.’)
print(name2, ‘is’, age2, ‘years old.’)
```

```
print(name, ‘is’, age, ‘years old.’ , end = ‘ ; ‘ )
print(name2, ‘is’, age2, ‘years old.’)
```

```
print(name, age, sep= ‘ : ‘)
print(name2, age2, sep = ‘ : ‘)
```

Type Conversion

- Variables can be recast using type conversion functions
- x = int (43.4) → x = 43
- y = float (x) → y = 43.0
- z = str (x) → z = “43”
- n = bool (0) → n = False
- m = bool (x) → m = True

Basic User Input

- The input function can be used to grab user input:

```
   num_str = input(  “Enter a number:  “  )
cat_name = input (  “What is your cat’s name?” )
```

- Accepts one string argument that contains the prompt seen by the user.

- Note that it ALWAYS returns a string.
- Recast as int or float to do math…

Exercise

Write a short program that asks the user their age.

Have the program print a message indicating how old the user will be in 10 years.

Variables and Memory

- Memory in python is a bit non-intuitive (to me at least)
- Characters and integers exist in one place in memory
- Can explore this using the “is” operator
  - True if variables point to *same memory location*
  - False otherwise
  - DOES NOT compare VALUES
- Try these:

```
a = 1
b = 1
print (a is b)
```

```
a = 1.0
b = 1.0
print (a is b)
```

```
a = ‘T’
b = ‘T’
print (a is b)
```

Variables and Memory

- Intrinsic variables, like ‘int’ don’t occupy a set amount of RAM
- e.g., all ‘ints’ are not 4 bytes…
- Can explore this using the getsizeof function
  - part of the sys module
  - returns size of an object in bytes
- Try these:

```
import sys
print( sys.getsizeof  ( 2**30))
```

```
import sys
print( sys.getsizeof  ( 2**60))
```

- Standard X-byte datatypes available via NumPy package (week 5)

Lists in Python

- Multiple values can be grouped into a list
  - mylist = [1, 2, 10]
- List elements accessed with [ ] notation
- Element numbering starts at 0
- print (mylist [1]) → displays 2
- Lists can contain different variable types
  - mylist = [1, ‘two’ , 10.0]
- Strings can be accessed element-wise like a list
  - mystring = ‘John’
  - print (mystring[1]) → displays ‘o’
- More on lists in two weeks…

I/O: Writing to a File

```
# generate some data
line1 = “This is the first line”
line2 = “This is the second line”

#  write data to a file
filename = ‘myfile.txt’
filemode = ‘w’     use ‘w’ when writing; ‘r’ when reading
file = open ( filename, filemode)
file.write(line1)
file.write(line2)
file.close( )
```

I/O: Reading From a File

```
#  read data from a file   (use readline)
filename = ‘myfile.txt’
filemode = ‘r’    use ‘w’ when writing; ‘r’ when reading
file = open ( filename , filemode)
line1 = file.readline( )
line2 = file.readline( )
file.close( )
print( line1)
print( line2)
```

NOTE: file.read() will read entire file into single string

Next Week: Conditionals and Functions

Slides: https://github.com/ResearchComputing/Python_Fall_2021/
Survey: http://tinyurl.com/curc-survey18

Thank you!