galvanize

gU

PYTHON FOR DATA ANALYSIS

Lesson 1.1

galvanize

Schedule

Start day: Today! (24th March, 2015)

2 Week course, 3 meetings per week (Tuesday, Thursday, and Sunday/Saturday)

Tuesday/Thursday Session (6-9pm)

Saturday/Sunday (10am-3pm, including a break)

Schedule:

- Tuesday March 24th (6-9pm)
- Thursday March 26th (6-9pm)
- Sunday March 29th (10am-3pm)
- Tuesday March 31st (6-9pm)
- Thursday April 2nd (6-9pm)
- Saturday April 4th (10am-3pm)

Syllabus

Week	Tuesday	Thursday	Weekend

1	• Environment Setup
	  ○ Terminal
	  ○ Git Basics
	• `Control Flow (if, else, while, for)`
	• File Input and Output
	• Basic Data Types (Strings, Integers, Floats)
	• Basic Data Structures (Lists)
	• Slicing & Indexing
	• Variables
		• Functions
		• Scope
		• Intermediate Data Structures (Dictionaries, Sets, multidimensional lists)
		• Reference vs. Value
		• Importing modules & libraries
		• Basics of Data Manipulation (text files)
			• Classes
			• Object Oriented Programming
			• Project Introduction

2	• `Advanced Python (enumerate, zip, with, defaultdict, Counter)`
	• Transforming data
	• Functional Programming
	• O(n) / Running time
		• numpy
		• Matrix operations
		• Vectorized functions
		• Basic Linear Algebra
			• Data Science workflow
			• Data Manipulations (working with data) and Statistics
			  ○ Pandas (and plotting)
			  ○ Scipy

AGENDA

- Why python?
- Getting start with Python
- Basics of Data Types
- Basic Control flow
- Tricks and Workflow (and debuggers!)
- IPython-notebook project

About Galvanize

- **Accredited Side: GalvanizeU**
  - Fully accredited Master Degree in Data Science
    - 12 Months program
    - (Internship included)
- **Un-accredited Side: gSchool**
  - gS-DS (Zipfian Academy)
  - Full Stack Web Development (6 months)

About Me

- **Jonathan Dinu**
- *VP of Academic Excellence @ Galvanize*
  - Studied CS and Physics at Berkeley
  - Formerly built algos for Alpine Data Labs
  - Then founded Zipfian Academy

- Email: jonathan@galvanize.com
- @clearspandex

galvanize

WHY PYTHON?

- General Purpose/Flexible Programming Language (it's the glue!)
- Growing [Data] Analytics Libraries & Community:
  - SciPy.org (for mathematics, science, and engineering)
  - StatsModels (Statistics)
  - Pandas (Frameworks)
  - SciKit-Learn (Machine Learning)
- Graphics
  - Libraries (ggplot, matplotlib)
  - APIs – Plot.ly
- Easy to Learn and Code

Q: What is Python?

Q: What is Python?

A: An **open source**, high-level, dynamic scripting language.

- **open source:** free! (both binaries and source files)

Q: What is Python?

A: An open source, **high-level**, dynamic scripting language.

- open source: free! (both binaries and source files)
- **high-level:** interpreted (add(a,b)) ; (not compiled '+')

Q: What is Python?

A: An open source, high-level, **dynamic** scripting language.

- open source: free! (both binaries and source files)
- high-level: interpreted (not compiled)
- **dynamic:** things that would typically happen at compile time happen at runtime instead (e.g., *dynamic typing*)

DYNAMIC TYPING

```
>>> x = 1
>>> x
1
>>> x = 'horse'
'horse'
>>>
```

Q: What is Python?

A: An open source, high-level, dynamic **scripting language**.

- open source: free! (both binaries and source files)
- high-level: interpreted (not compiled)
- dynamic: things that would typically happen at compile time happen at runtime instead (eg, *dynamic typing*)
- **scripting language: it's the glue!**

BASIC DATA STRUCTURES

The most basic data structure is the None type.
This is the equivalent of NULL in other languages.

There are four basic numeric types:

1. int ($< 2^{63}$) / long ($\geq 2^{63}$)*
   * on 64-bit OS X/Linux, sys.maxint = 2**63-1
2. float (a “decimal”)
3. bool (True/False) or (1/0)
4. complex (“imaginary”)

```
>>> type(None)
<type 'NoneType'>
>>> type(1)
<type 'int'>
>>> type(2.5)
<type 'float'>
>>> type(True)
<type 'bool'>
>>> type(2+3j)
<type 'complex'>
```

http://docs.python.org/2/library/stdtypes.html#numeric-types-int-float-long-complex

BASIC DATA STRUCTURES

The next basic data type is the array, implemented in Python as a **list**.

A list is a (zero-based numbered), *ordered* collection of elements, and these elements can be of arbitrary type.

Lists are mutable, meaning they can be changed in-place.

```
>>> a = [1,'b',True]
>>> a[2]
True
>>> a[1]='aa'
>>> a
[1,'aa',True]
```

http://docs.python.org/2/library/stdtypes.html#numeric-types-int-float-long-complex

BASIC DATA STRUCTURES

Tuples: immutable arrays of arbitrary elements.

```
>>> x = (1,'a',2.5)
>>> x[0]
1
>>> x[0]='b' #trying to change a value
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> a,b = (1,2)
>>> a
1
```

Tuples are frequently used behind the scenes in a special type of variable assignment called tuple packing/unpacking.

http://docs.python.org/2/library/stdtypes.html#numeric-types-int-float-long-complex

BASIC DATA STRUCTURES

The **string** type in Python represents an *immutable ordered* array of characters (note there is no char type).

Strings support *slicing* and *indexing* operations like arrays, and have many other string-specific functions as well.

String processing is one area where Python excels.

http://docs.python.org/2/library/stdtypes.html

BASIC DATA STRUCTURES

Our final example of a *"data type" is the Python file object.* This example represents an open connection to a file on your laptop.

```
>>> with open('output_file.txt','w') as f:
...     f.write('test')
```

These are particularly easy to use in Python, especially using the with statement *context manager*, which automatically closes the file handle when it goes out of scope.

http://docs.python.org/2/library/stdtypes.html#file-objects

PYTHON CONTROL FLOW

CONTROL FLOW

Python has a number of control flow tools that will be familiar from other languages. The first is the **if-else** statement, whose compound syntax looks like this:

```
>>> x, y = False, False
>>> if x :
...     print 'apple'
... elif y :
...     print 'orange'
... else :
...     print 'sandwich'
...
sandwich
```

A **while loop** executes while a given condition evaluates to True.

```
>>> x = 0
>>> while True :
...     print 'HELLO!'
...     x += 1
...     if x >= 3 :
...         break
...
HELLO!
HELLO!
HELLO!
```

The familiar (& useful) **for loop** construct executes a block of code for a range of values.

```
>>> for k in range(4) :
...     print k**2
...
0
1
4
9
```

The object that a for loop iterates over is called (appropriately) an *iterable*.

A useful but possibly unfamiliar construct is the **try-except block**:

```
>>> try:
...     print undefined_variable
... except :
...     print 'An Exception has been caught'
...
An Exception has been caught
```

This is useful for catching and dealing with errors, also called exception handling.

Question (5 minutes):

Now that we know how to work with numbers and strings, let's write a program that might actually be useful!

Let's say you want to find out how much you weigh in stone.
A concise program can make short work of this task. Since a stone is 14 pounds, and there are about 2.2 pounds in a kilogram, the following formula should do the trick:

$$m_{stone} = \frac{m_{kg} \times 2.2}{14}$$

So, let's turn this formula into a program

Walkthrough

First step:

Ask the user: "What is your mass in kilograms?"

Hint: Use the function “input” to save the value.

Second Step: Make the calculation

Third Step: Print the results.

Question (10 minutes) :

Write a program which will find all such numbers which are divisible by 7 but are not a multiple of 5, between 12 and 5000 (both included).

Extra: The numbers obtained should be printed in a comma-separated sequence on a single line.

Hints:

Consider use range(#begin, #end) method

Question (15 minutes) :

Write a program which can compute the factorial of a given numbers.

The results should be printed in a comma-separated sequence on a single line.

Suppose the following input is supplied to the program:

8

Then, the output should be:

40320

Hints:

In case of input data being supplied to the question, it should be assumed to be a console input.