FOR INSTRUCTOR PURPOSES ONLY

INSTRUCTOR NOTES

- Instructor Prep:
  - Review & modify lesson plan & slide deck
  - Write learning objectives & relevant information on board
  - Review Student Roster and Instructor Checklist
  - Prepare handout to distribute to students.

FOR INSTRUCTOR PURPOSES ONLY

MATERIALS

FOR INSTRUCTOR PURPOSES ONLY

PRE-WORK

- Bring a laptop with Anaconda using Python 2.7 installed. Note: Make sure to choose the *Python 2.7* version for your operating system.

- If you are using a PC, also install git-bash terminal.

- *Optional*: Install a text editor like Sublime Text 3 or Atom on your computer. If you are using Anaconda, **Spyder** is included in the distribution.

GENERAL ASSEMBLY

PYTHON PROGRAMMING 101

Greg Baker
Director & Head of AI
IFOST, Queckt, BigInsights

LEARNING OBJECTIVES

- Discuss the history of Python
- Define how Python compares to other programming languages
- Describe the benefits of a Python workflow when looking at data
- Demonstrate basic Python programming fundamentals to solve a real world problem
- Create a custom learning plan to build your data science skills after this workshop!

DATA SCIENCE 101

PRE-WORK

PRE-WORK REVIEW

- Bring a laptop with Anaconda installed. Scroll to your operating system version and click on the install button for Anaconda with Python 2.7.

- We will be using Jupyter Notebooks as the main IDE for the workshop. If you have installed Anaconda, then you are ready to go!

OPENING

ABOUT ME

- Welcome to Python Programming 101!

- Here's a bit about me:

- *Name: Greg Baker*
- *Background: Freelance consultant except for when I worked for google*
- *At GA: I teach data science classes*
- *Fun Fact: I was on the team at CSIRO who developed WiFi. I quit because I thought what they were doing was a bad idea.*

ABOUT YOU

- So that Greg knows what to say… let's talk a bit about you!
- Name
- What brings you to GA
  - Current activities
  - Goals
  - Have you programmed before?
- (Fun fact)

OUR EXPECTATIONS

- You're ready to take charge of your learning experience.
- You're curious and excited about Python!
- You've installed Anaconda with Python 2.7 (or 3.x)

THE BIG PICTURE

- What we'll cover:

  - What is Python and what can Python do for me?
  - Implementing Python into your workflow
  - Lists, dictionaries and other types — duck typing
  - Python Libraries
  - How to complain about the weather in pictures (graphs in Python)
  - Python control structures
  - What sort of code to expect if you are coming along for a data science course

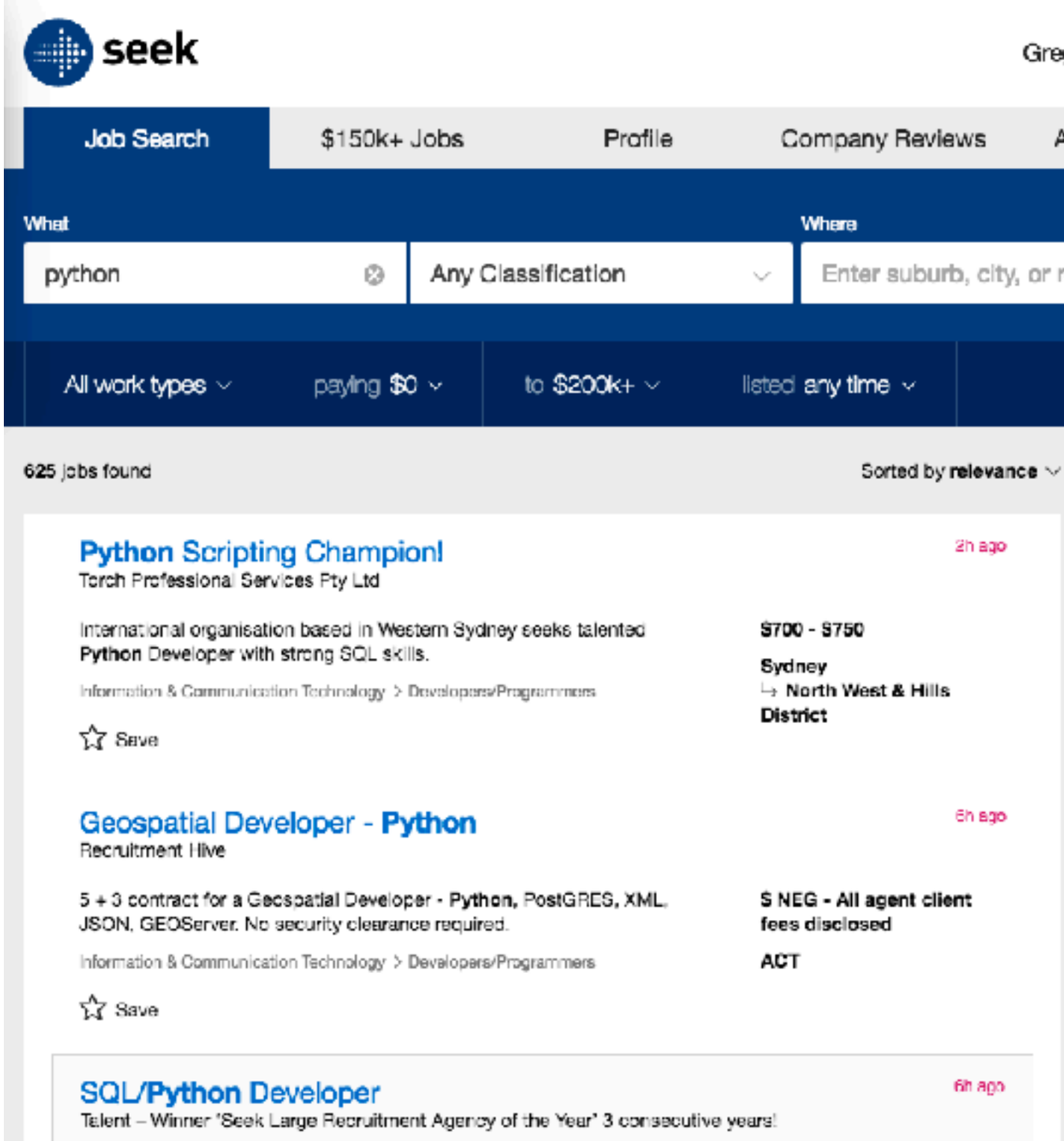
THE BIG PICTURE

- Why this topic matters:
  - Programming is a sought-after skill
  - Python has been gaining popularity (why will see why!)

- Why this topic rocks:
  - Python opens up a door to a variety of opportunities, from data science to research

WHAT IS PYTHON AND WHAT CAN IT DO FOR ME?

This might be why you want to learn Python



WHAT IS PYTHON?

- Created by Guido Van Rossum in 1991
- Emphasizes **productivity** and code **readability**
  - The language is **easy** to pick up and learn
  - This gentle learning curve brings makes it easier for many to contribute to **production** level code
  - **Readable** code means that almost anyone can pick up a piece of code and understand what it is doing

SOME CHARACTERISTICS OF PYTHON

- **Byte-code Interpreted language**:
- There isn’t a step where you “run the compiler”.
- Just write your code and run it.
- Modules get automatically compiled to “.pyc” files when they are first used.

SOME CHARACTERISTICS OF PYTHON

- **Layout for scope**:
- There isn't a BEGIN / END or { } to group statements
- It's just how much of whitespace indent there is…

```
if progress == 100:
   completed=True
else:
   completed=False
```

SOME CHARACTERISTICS OF PYTHON

- **Not always developed in a normal IDE**:
- PyCharm is a popular environment for developers
- Many data scientists use **Jupyter**, though

SOME CHARACTERISTICS OF PYTHON

- **High-level programming**:
- You get a lot done without many lines of code

```
import requests
api = 'http://pokeapi.co/api/v2/pokemon/'
r = requests.get(api + 'pikachu')
print [p['ability']['name']
          for p in r.json()['abilities']]
```

WHY PYTHON?

Python is:

- Great for rapid prototyping and full-stack commercial applications.
- A **modern**, elegant, object-oriented language.
- Highly **expressive**, i.e., you can be more **productive**.
- Well documented and has an established and **growing community.**
- Comes with **"batteries included**" **-** in other words, Python has libraries that will help you do a ton of different tasks!

DEMO

IMPLEMENTING A PYTHON WORKFLOW

PYTHON PROGRAMMING

- Let us what a Python program looks like.
- Starting with the typical `"Hello World!"` program:
  - In essence, we are writing code to print the message "Hello World!" in the screen.

Python

```
1 | print("hello world")
```

A very very simple program: one line of code that will `print` the string `'Hello World!`.

It is easy to read and understand.

PYTHON PROGRAMMING

What about Java?

Java

```java
public class HelloWorld
{
    public static void main (String[] args)
    {
        System.out.println("Hello, world!");
    }
}
```

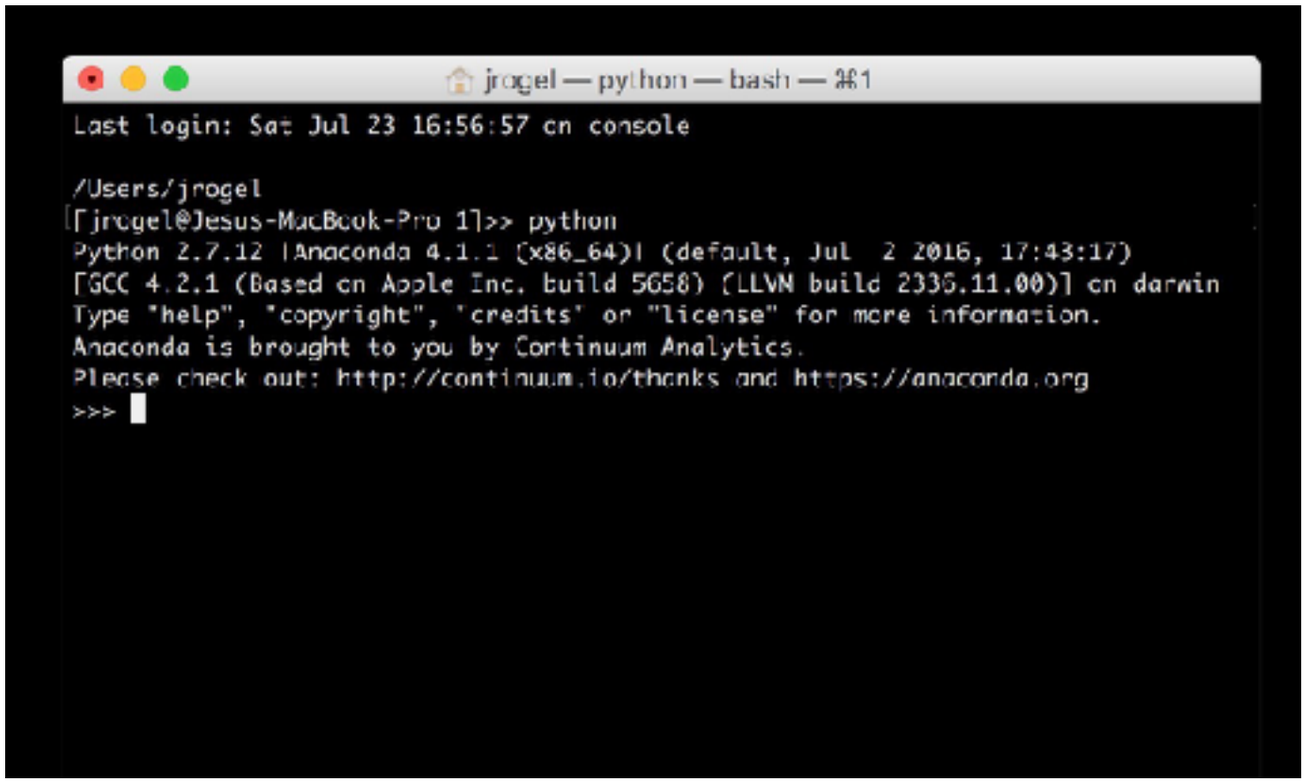
- Once again, it does exactly the same, but it takes a lot more code!

PYTHON: INTERACTIVE SHELLS V

- In our “Hello World!” python program, we are assuming that we are using an **interactive shell**,
- In other words, we are writing code that is executed immediately by the Python interpreter.
- We are able to "*interact*" with the results of the commands we pass. We can do this using a:
  - Python shell
  - iPython shell
  - Jupyter notebook

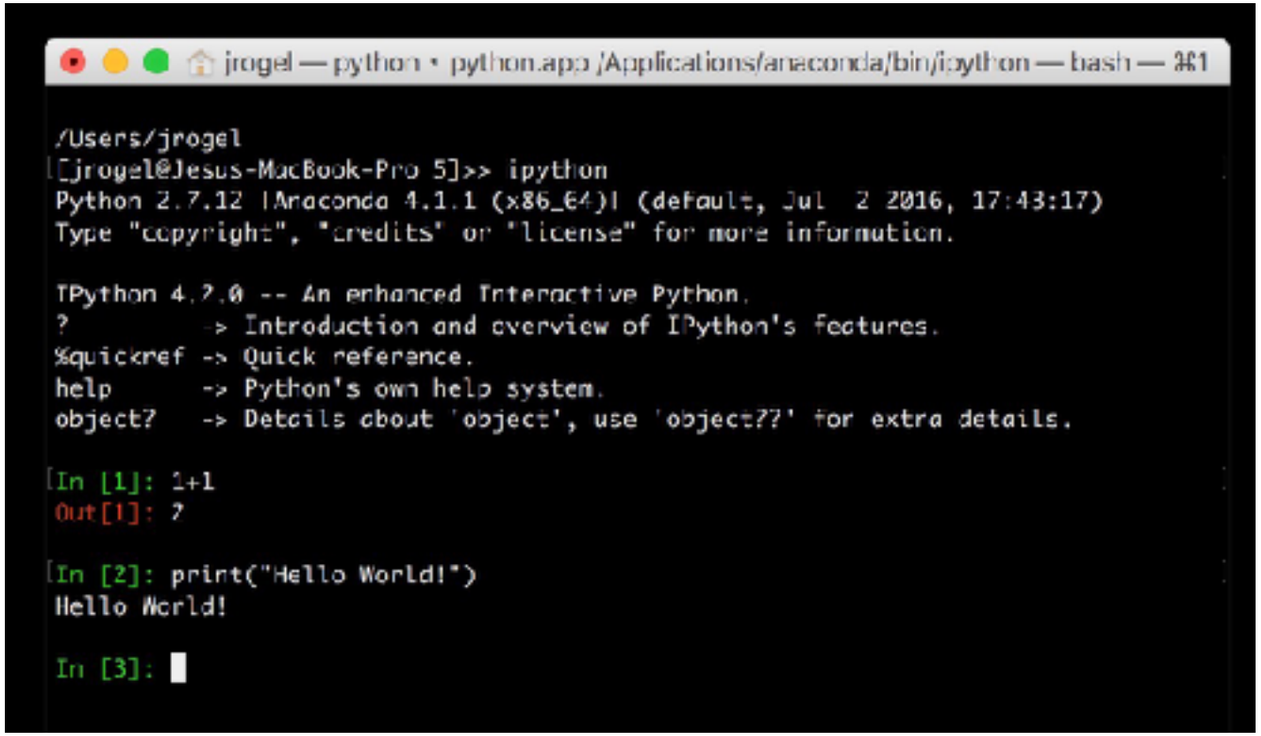
PYTHON SHELL

- A python shell is similar to a Command Line Terminal and it can be launched by typing:
- **“python”**



PYTHON SHELL

- A python shell is more interesting than a plain terminal, providing syntax coloring and shortcuts to interact with our code. It can be launched with:
- **“`ipython`"**



PYTHON: INTERACTIVE SHELLS V

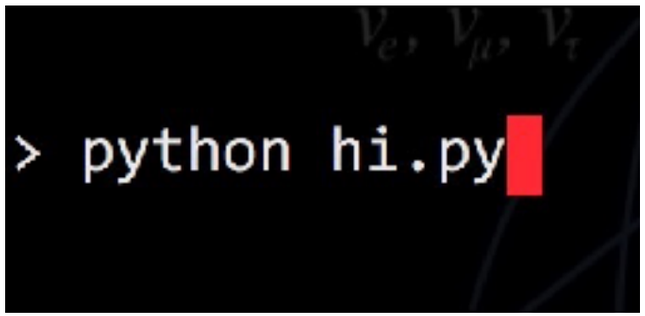
- Sometimes we do not need to interact with our Python code.
- Instead, we may want to execute a program and simply get results.
- In those cases, we need to create a Python script.
- To do so, we can use a text editor of our choice and save the code in a file with extension “`.py`”.

PYTHON SCRIPT

- A barebones script for the "Hello World!" program (saved to a file called `hi.py`) looks like this:

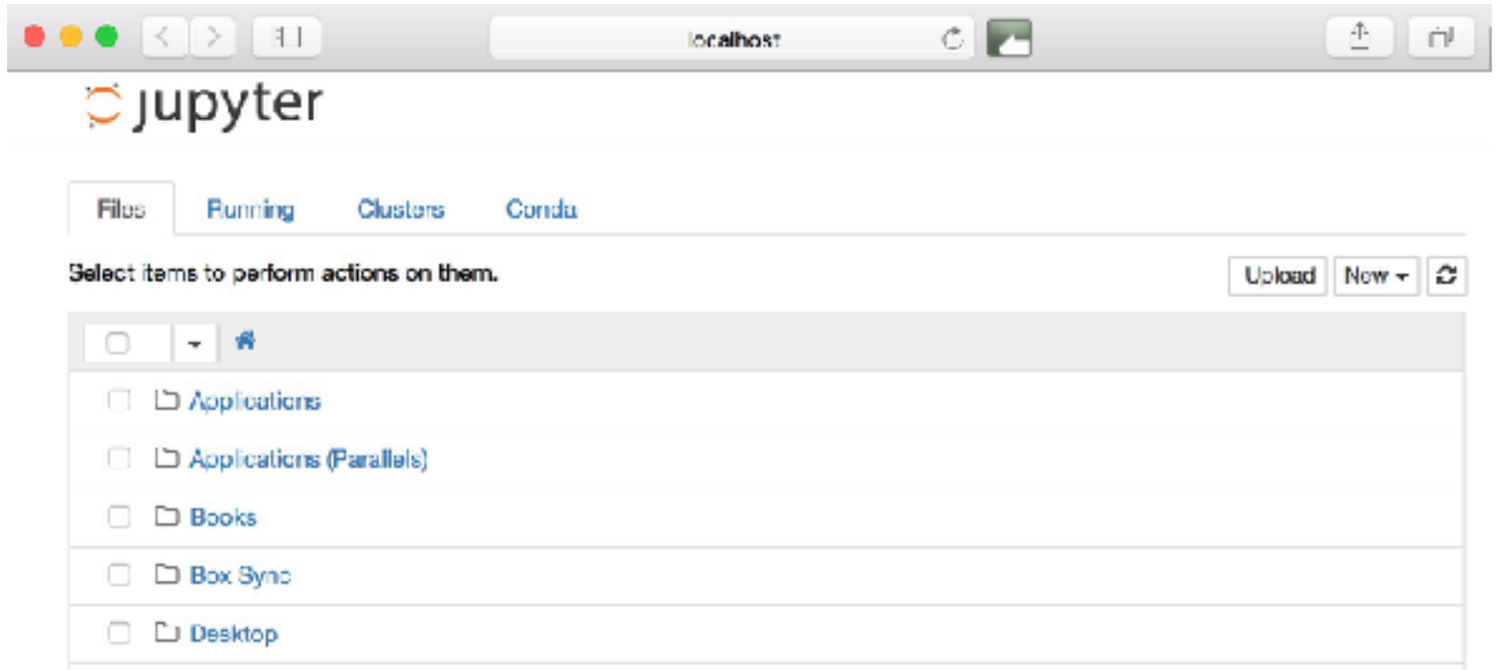
```python
print("hello world")
```

- To run the script by passing it as a command to the Python interpreter we need to write:



JUPYTER NOTEBOOK

- A Jupyter notebook is a web interface that let's us use formatting along side our code. It is the extremely common and very useful! You can launch it by typing:
- **"`jupyter notebook`"**



SCRIPT EDITORS AND IDE

- Integrated development environments (IDEs) provide comprehensive facilities for computer programmers involved in software development.
  - PyCharm
  - Eclipse with PyDev
  - Atom
  - Spyder (included in Anaconda)

INSTRUCTIONS

- We recommend using a Jupyter notebook for this practice.

1. Open Jupyter: in a terminal type: `` `jupyter notebook` ``
2. Navigate to an appropriate folder where your work will be saved
3. On the top-right-hand-side click in the button called "New" and select "`Python 2`" or "`Root`" (depending on your installation of Python)
4. Voilà, you are ready to type the commands we will cover


LET’S DO IT
Start Jupyter
Write a snippet that says “Hello World”

PYTHON SCRIPT

- Unlike other languages, there's no `main()` function that gets run automatically
- the `main()` function is implicitly all the code at the top level.
- A more sophisticated version of the “Hello World!” program is as follows:

Python

```python
def main():
    print("hello world")

if __name__ == '__main__':
    main()
```

PYTHON BASICS

- Variables, types, assignment
- Using modules

TYPES, VARIABLES, ASSIGNMENT

- Like any other programming language, we need to use **types** and variables and be able to assign values to them

Python

```python
# variable assignments
x = 1.0
my_variable = 12.2
type(x)

y = 1
type(y)

b1 = True
type(b1)

s = "String"
type(s)
```

YOUR TURN

- Try the following in your Jupyter Notebook:

Python

```
import types
print(dir(types))

1+2, 1-2, 1*2, 1/2

1.0+2.0, 1.0-2.0, 1.0*2.0, 1.0/2.0

# Comment

# Comparison: >, <, <=, <=, ==
2 > 1

# Testing for equality
2 == 2
```

LISTS

- Lists are collections of objects
- They can be changed

Python

```python
l = [1,2,3,4]
print(type(l))
print(l)
print(l)
print(l[1:3])
print(l[::2])

# Python starts counting from 0
print(l[0])
```

TUPLES

- Tuples are very similar to lists, but
- They cannot be changed

Python

```python
point = (10, 20)
print(point, type(point))

x, y = point
print("x =", x)
print("y =", y)
```

DICTIONARIES

- Dictionaries combine keys with values in pairs
- Like in a dictionary, you can search the keys to obtain their corresponding value

```
params = {"parameter1" : 1.0, "parameter2" : 2.0,
                "parameter3": 3.0
             }
print (type(params))
print (params)
print params['parameter1']
```

DUCK TYPING

- If it quacks, it’s a duck
  - Therefore my son’s phone is a duck
- If you can iterate over it, and you can select elements from it, you can use it as a list
  - It might not actually be a list though, but that’s OK.

FIVE HANDY FUNCTIONS

```
In [1]: sum([1,2,3])
```

```
Out[1]: 6
```

```
In [2]: len([1,2,3,4,5])
```

```
Out[2]: 5
```

```
In [3]: len("hello world")
```

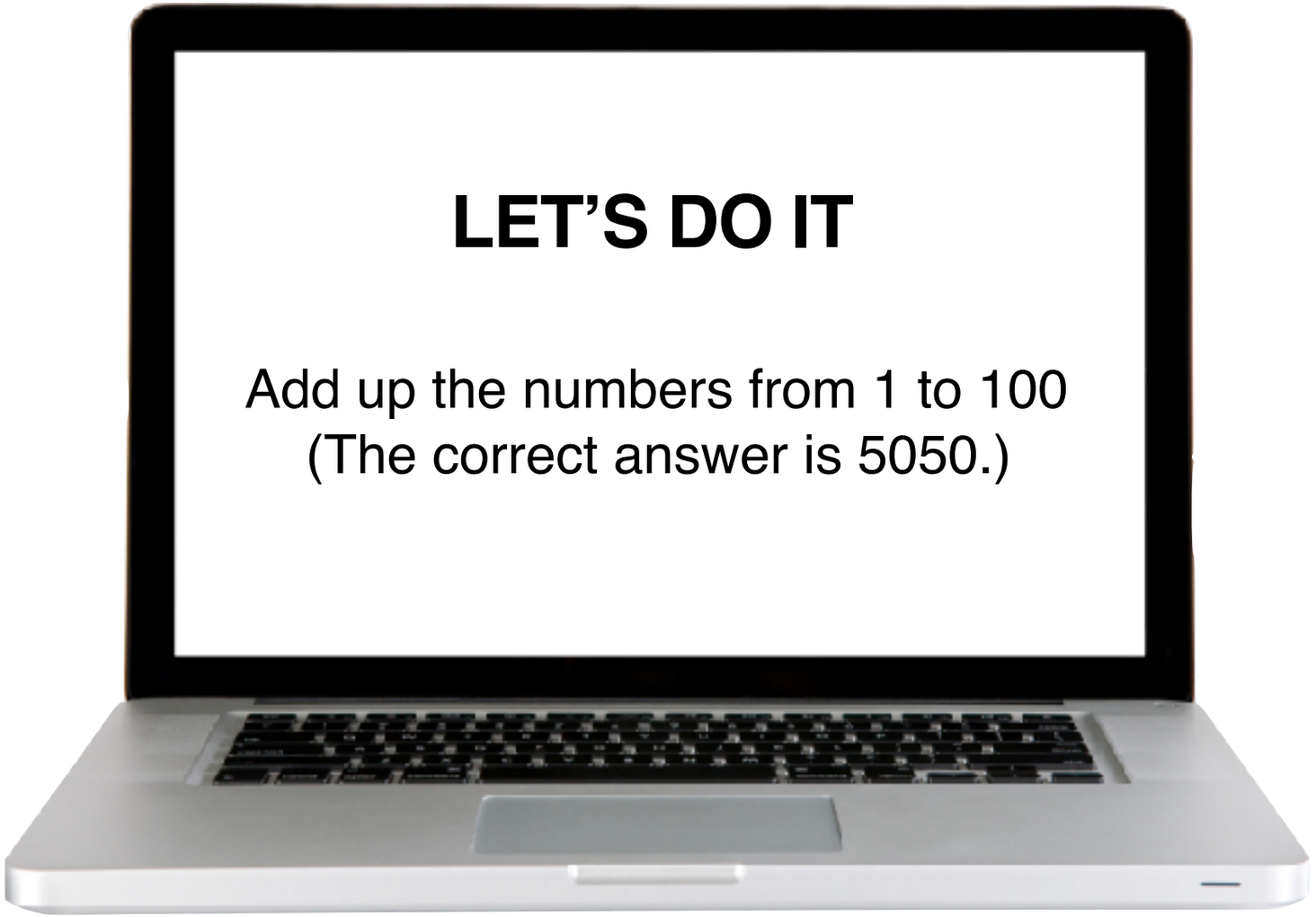
```
Out[3]: 11
```

```
In [4]: range(10)
```

```
Out[4]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
In [5]: params = {"parameter1" : 1.0, "parameter2" : 2.0,
                  "parameter3": 3.0
                 }
        params.items()
```

```
Out[5]: [('parameter1', 1.0), ('parameter3', 3.0), ('parameter2', 2.0)]
```


LET’S DO IT
Add up the numbers from 1 to 100
(The correct answer is 5050.)

GUIDED PRACTICE

INSTALLING & CONFIGURING COMMON PYTHON LIBRARIES

IMPORTING A MODULE

- We need to import the functionality of packages and modules before we can use them
- Here we import the “math” module to use mathematical functions:

Python

```
1  import math
2  x = math.cos(2 * math.pi)
3  print(x)
4
5  from math import *
6
7  log(10)
8
9  log(10,2)
```

PACKAGES

- Libraries of code written to solve particular set of problems
- Can be installed with: **`pip install <package name>`**
  or **`conda install <package name>`**

- Ever used Excel? How do you fancy working with data structured in a similar way, but better graphics and less hassle? Try **pandas**
- Does your application require the use of advanced mathematical or numerical operations using arrays, vectors or matrices? Try **SciPy** (scientific python) and **NumPy** (numerical python)

PACKAGES

- Libraries of code written to solve particular set of problems
- Can be installed with: **`pip install <package name>`**
  or **`conda install <package name>`**
- Does your application require the use of advanced mathematical or numerical operations using arrays, vectors or matrices? Try **SciPy** (scientific python) and **NumPy** (numerical python)

PACKAGES

- Are you interested in using python in a data science workflow to exploit machine learning in your applications? Look no further than Scikit-learn

- Are you tired of boring-looking charts? Are you frustrated looking for the right menu to move a label in your plot? Take a look at the visuals offered by matplotlib or seaborn

PACKAGES

- Is your boss asking about significance testing and confidence intervals? Are you interested in descriptive statistics, statistical tests, or plotting functions? Well statsmodels offers you that and more.

- All the data you require is available freely on the web but there is no download button and you need to scrape the website? You can extract data from HTML using Beautiful Soup

PACKAGES

- Do you want to write the next awesome artificial intelligence program that kinda sorta understands written texts? Try nltk or spaCey


LET’S DO IT
!pip install seaborn
import seaborn

BREAK

DEMO

DRAWING GRAPHS IN PYTHON

SAMPLE DATA - JUPYTER NOTEBOOK CODE

In [1]:
```
%matplotlib inline
```

In [2]:
```
import pandas
import matplotlib
```

In [3]:
```
climate = pandas.read_csv('sydney-yearly-data.csv', index_col=0)
```

In [4]:
```
climate.plot()
```

SAMPLE DATA - JUPYTER NOTEBOOK CODE


```
In [1]: %matplotlib inline
```

Jupyter command

- Display images inline in the notebook

SAMPLE DATA - JUPYTER NOTEBOOK CODE


```
In [1]: %matplotlib inline

In [2]: import pandas
        import matplotlib
```

Python code

- Import two modules
- Most of the time you'll see **`import pandas as pd`** because programmers are lazy
- Both of these modules (pandas and matplotlib) are not part of the standard Python install
    - pip install pandas ; pip install matplotlib

SAMPLE DATA - JUPYTER NOTEBOOK CODE


```
In [1]: %matplotlib inline

In [2]: import pandas
        import matplotlib

In [3]: climate = pandas.read_csv('sydney-yearly-data.csv', index_col=0)
```

Python code

- Call the function “read_csv” in the pandas module
- Filename to read = ‘sydney-yearly-data.csv’
- Make the first column of the CSV file be an index
- Store the result in the name “climate”

SAMPLE DATA - JUPYTER NOTEBOOK CODE


```
In [1]: %matplotlib inline

In [2]: import pandas
        import matplotlib

In [3]: climate = pandas.read_csv('sydney-yearly-data.csv', index_col=0)

In [4]: climate.plot()
```

- Draw a pretty graph of all the columns of data in “climate”

INTRODUCTION

PYTHON CONTROL STRUCTURES

PYTHON PROGRAMMING

- **Control Structures**: A block of programming that analyses variables and chooses a direction in which to go based on given parameters.

- The term flow control details the direction the program takes (which way program control “flows”). It determines how a computer will respond when given certain conditions and parameters. Some typical structures include:

  - **if** statement
  - **for** loop
  - Functions

IF

An `if` statement is a conditional structure that, if proved true, performs a function or displays information.

Think of this as a decision that moves the flow of your program depending on the answer to a TRUE-FALSE question.

In Python we can write:

Python

```python
if age_person > 18:
    return "They can drive"
else:
    return "They cannot drive"
```

IF

Another example:

Python

```python
A = 10
B = 100
if A>B:
    print("A is larger than B")
elif A==B:
    print("A is equal to B")
else:
    print("A is smaller than B")
```

FOR LOOP

A loop statement in programming performs a predefined set of instructions or tasks while or until a predetermined condition is met.

In Python we can write:

```python
users = ["Jeff", "Jay", "Theresa"]

for user in users:
    print("Hello %s" % user)
```

Which is the same as writing:

```
user = "Jeff"
print("Hello %s" % user)
user = "Jay"
print("Hello %s" % user)
user = "Theresa"
print("Hello %s" % user)
```

FOR LOOP

Let us see other examples. Can you explain what the program is doing?

Python

```python
for x in [1,2,3]:
    print(x)

for key, value in params.items():
    print(key + " = " + str(value))
```

WEIRD THINGS ABOUT FOR LOOPS

- There isn't an initialiser-condition-iterator (as in Java, Perl, C or C++)
  - You can break out or carry on with **break** or **continue**
- You can have a for loop over anything that supports iteration
  - Lists, Strings, Dictionaries, Generators, Files
- You can have an else clause (if we made it to the end without interruption)

```
for line in open('file.txt'):
    if 'Hello' in line: continue
    if 'Goodbye' in line: break
    print(line)
else:
    print("Goodbye not found")
```

LIST COMPREHENSIONS

List comprehensions are an elegant way to define and create list in Python. It uses a for loop inside the definition of the list itself.

Let’s take a look at one, and see if you can figure out what is happening:

Python

```python
l1 = [x**2 for x in range(0,5)]
```

INCOMPREHENSIBLE STUFF I WRITE ALL THE TIME

- List comprehensions can have if statements embedded in them.
- % operator in Python means “remainder when divided by”

```
sum([x for x in range(100) if x % 2 == 1])
```

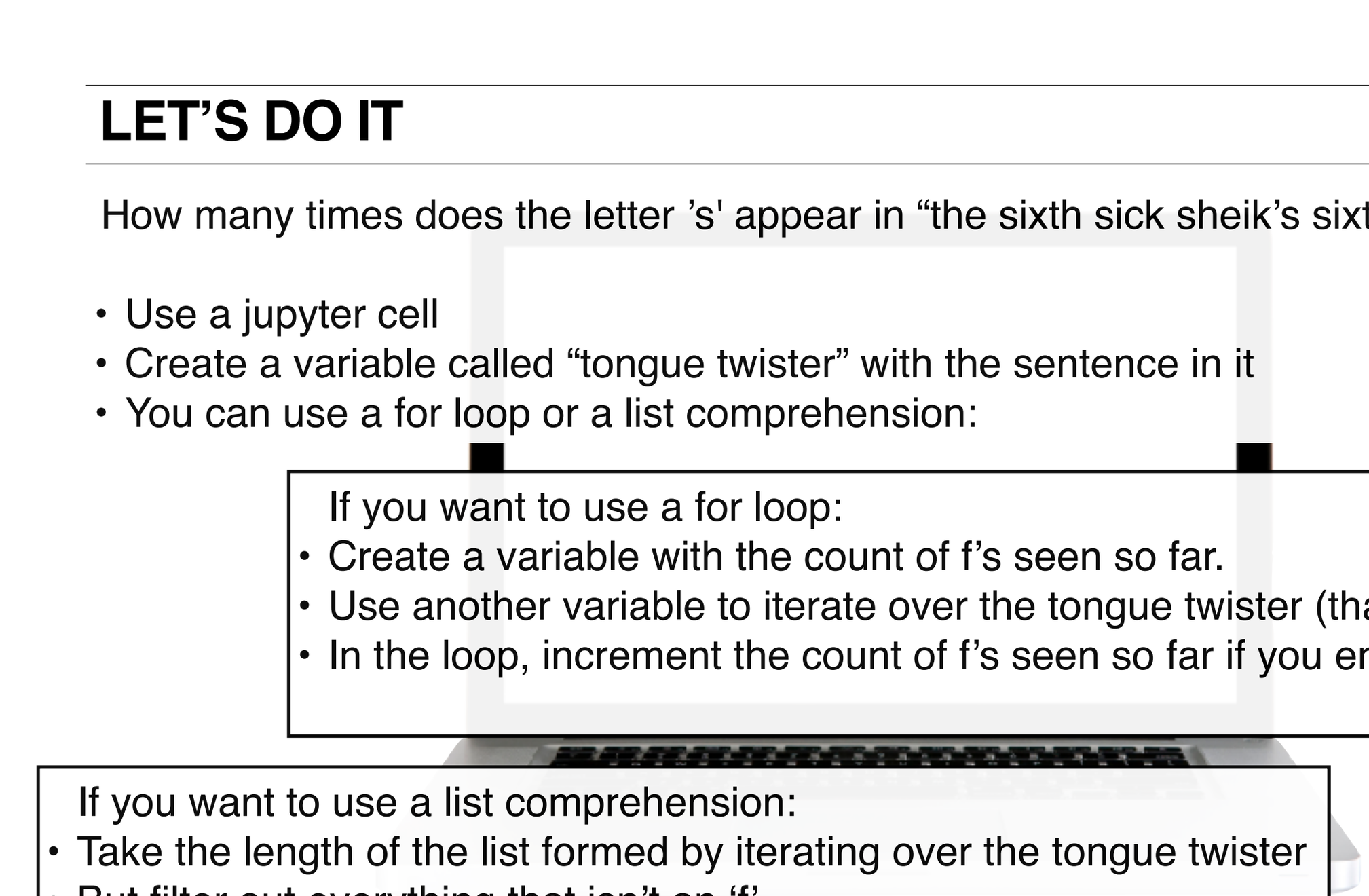
Means the same as

```
odds = []
for x in range(100):
    if x % 2 == 1:
        odds.append(x)
sum(odds)
```

LET’S DO IT

How many times does the letter ’s' appear in “the sixth sick sheik’s sixth sick sheep”?

- Use a jupyter cell
- Create a variable called “tongue twister” with the sentence in it
- You can use a for loop or a list comprehension:



If you want to use a for loop:

- Create a variable with the count of f’s seen so far.
- Use another variable to iterate over the tongue twister (that’s your for loop)
- In the loop, increment the count of f’s seen so far if you encounter a letter ‘f’

If you want to use a list comprehension:

- Take the length of the list formed by iterating over the tongue twister
- But filter out everything that isn’t an ‘f’

FUNCTIONS

A function is a group of instructions used by programming languages to return a single result or a set of results.

Functions are a convenient way to divide our code into useful blocks, providing us with order, making the code more readable and reusable.

Here is how you define a function in Python:

```python
def function_name(input1, input2...):
    1st block of instructions
    2nd block of instructions
    ...
```

FUNCTIONS

Let’s define a function that returns the square of the input value. It even has a “““documentation string (docstring)””” that describe what it does

Python

```python
def square(x):
    """
    Return the square of x.
    """
    return x ** 2
```

```
In [2]: square?

In [ ]:
```

```
Signature: square(x)
Docstring: Return the square of x
File:      ~/Documents/Development/python-workshop
Type:      function
```

FUNCTIONS

We can call this function as follows:

Python

```python
var1 = 7

var2 = square(var1)

print(var2)
```

GUIDED PRACTICE

REAL DATA SCIENCE CODE

Pinky swear: nothing in the data science courses is all that hard. Here's some real code to play with.

INSTRUCTIONS

- Let's open a new Jupyter notebook for this practice.
- Work in pairs if you aren't feeling confident.

1. Save the file called **`Python101_Part2_GuidedPractice.ipynb`** in a known location in your file system
2. Start Jupyter (jupyter notebook) and navigate to the location where you saved the file
3. Open the file by clicking on the name
4. Voilà, you can start the Guided Practice

CONCLUSION

REVIEW & RECAP

- In this workshop, we've covered the following topics:
  - Python as a popular, flexible programming language
  - Python has applications in many different areas
  - Python is particularly great for data manipulation
  - Python programming basics include: types, variables, functions, and more!

WHAT SHOULD YOU DO NEXT?

For beginner programmers:

- Go through Learn Python the hard way
- Familiarise yourself with the language by going through A Beginner's Python Tutorial

WHAT SHOULD YOU DO NEXT?

For existing programmers who are new to Python, try these:

- Read the information in Moving to Python From Other Languages
- Python for java developers
- Python for MATLAB users

WHAT SHOULD YOU DO NEXT?

For anyone looking for a challenge :)

- Challenge yourself by tackling the Python Challenge

WANT MORE?

General Assembly offers courses that use Python!

Check out our:

- Part-time Data Science Course
- Data Science Immersive Course

TEST YOURSELF (PERHAPS LATER)

- Let's open a Jupyter notebook for this practice.

1. Save the file called **`Python101_Part2_IndPractice.ipynb`** in a known location in your file system
2. Start Jupyter (you know how now) and navigate to the location where you saved the file
3. Open the file by clicking on the name
4. Voilà, you can start the Independent Practice

Q&A

PYTHON PROGRAMMING101

EXIT TICKETS

DON’T FORGET TO FILL OUT YOUR EXIT TICKET