

# PYTHON PROGRAMMING CRASH COURSE

**Day 1**

August 2018



# Who am i?

- Founder of [The Cognitive Company](#)
- Engineer at Hut34
- Data Science at General Assembly
- Mining Engineer
- Foodie



@mattszwec



Matthew Szwec

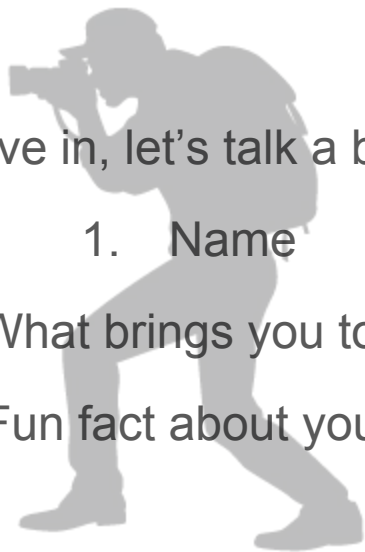


matt@thecognitiveco.com

# About you?

Before we dive in, let's talk a bit about you!

1. Name
2. What brings you to GA?
3. Fun fact about yourself!



# Expectations

While I am presenting don't be afraid to ask questions

If a class member is speaking please respect them

During the coding activities ask the person to your left and right for the answer  
before asking Stefano or I

# Agenda Day 1

- Overview
- Install and Setup
- Programming Workflow
- Python Fundamentals
- Handling Data
- Assignment

# Learning Objectives



# Learning Objectives

- Understand how to setup Python programming environment
- Discuss the history of Python & how it's used in different industries
- 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!

# Overview

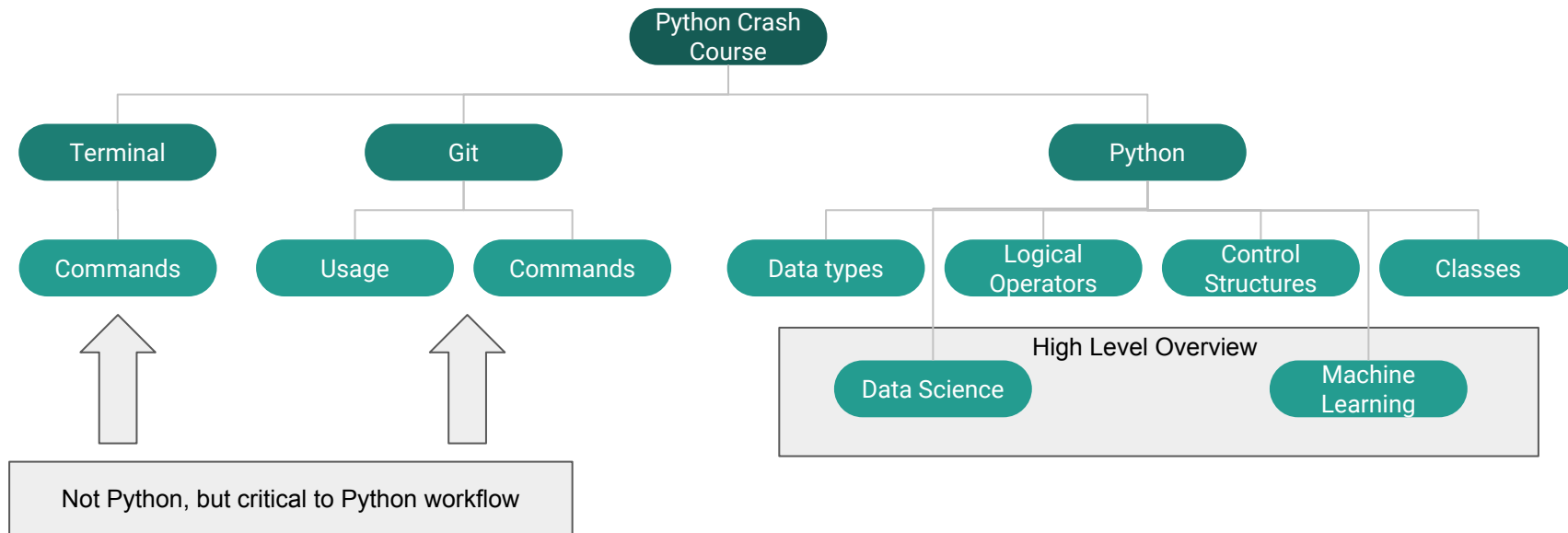




# Why are we here?

- Why are we doing this course: Introduction to Python in preparation for GA Data Science Course
- What will we learn: Everything you need to get started with python on day 1 of your course
- Course is best done on Mac or Linux (Ubuntu) => Windows is more difficult to setup
- Questions: During presentations don't be afraid to ask questions
- During the activities ask the person to your left and right if they know the answer before asking me (there is only one of me)

# Overview



# Install Fest



# Slides

Download slides

from

<https://goo.gl/KtHbZD>

WiFi

GA-Guest  
yellowpencil

# Install Fest

Install the following software for your OS:

- Install Anaconda 3.6 (<https://www.anaconda.com/download/> )
- Install (WinDoz) Git Bash <https://gitforwindows.org/>
- Install (MacOS) Git (<https://git-scm.com/downloads>)
- Install sublime text 3 (<https://www.sublimetext.com/3> )
- Make a github account (<https://github.com> )
- Make a Slack account (<https://slack.com/>, @pycrashcourse )

WiFi  
GA-Guest  
yellowpencil

# Anaconda 101



# Install Fest

15<sub>min</sub>

Install the following software for your OS:

- Install Ananconda 3.6 (<https://www.anaconda.com/download/> )
- Install Git (<https://git-scm.com/downloads>)
- Install sublime text 3 (<https://www.sublimetext.com/3> )
- Make a github account (<https://github.com> )
- Make a Slack account (<https://slack.com/>, @pycrashcourse )

WiFi  
GA-Guest  
yellowpencil



# Development Env Check

## Part 2. Anaconda Installation

In this course, we'll be working closely with tools that utilize the Python programming language. Anaconda is a popular cross-platform tool that helps install and manage Python-related data science libraries.

1. [Download Anaconda](#) and follow the installation instructions package for your operating system. Please make sure that you're downloading the latest stable version for Python 3!
2. Agree to the terms and let Anaconda complete its default installation.
3. Once installed, navigate to your command line (on OS X, this is the terminal application; on Windows, use your new `Git Bash` shell) and confirm that it's installed by typing in the `which conda` command.





# Activity Make Anaconda Env

```
conda create -n dat2018 python=3.6
```

```
conda info --envs
```

```
source activate dat2018
```

```
pip install numpy sklearn pandas
```

```
source deactivate
```

```
'Hello world'
```

```
Python -v
```



EXERCISE



# Development Env Check

You should see:

```
$ which conda
/Users/USERNAME/anaconda3/bin/conda
```

- If the command line returns a file path (like in the example below), you've successfully installed Anaconda.
  - If the command line returns nothing (and sends you back to the prompt), check in with your instructor.
    - Note: Your file path may look different.
    - Note: You'll often see commands that look like: `$ which conda` above — when you see those, type in everything except the dollar sign. The dollar sign is used to denote a code prompt in your window.
1. Once installed, run the following command to ensure that some frequently used libraries are installed. Anaconda may also update your packages at this time (which is OK!).
  - 2.

```
conda install jupyter notebook python matplotlib nltk numpy pip setuptools scikit-learn scipy statsmodels
```



# Colour your terminal (Windows)

WINDOWS ANACONDA PROMPT

1. Right click on the top bar.
2. Change the background to black, text to suit.
3. Black will be our anaconda terminal

WINDOWS GIT BASH

1. Right click on the top bar.
2. Change the background to a second colour, text to suit.
3. This will be our git terminal

# Install Fest

15<sub>min</sub>

Install the following software for your OS:

- Install Ananconda 3.6 (<https://www.anaconda.com/download/> )
- Install Git (<https://git-scm.com/downloads>)
- Install sublime text 3 (<https://www.sublimetext.com/3> )
- Make a github account (<https://github.com> )
- Make a Slack account (<https://slack.com/>, @pycrashcourse )

WiFi  
GA-Guest  
yellowpencil

# Intro to Git



**Warning!**

# Warning!

- ***“Git is infuriating”*** - Mandy Brown
- It takes a long time to feel comfortable with it
- Most explanations of it get very technical very quickly
- Focus on the concepts

# History of Git

- Made in 2005 by Linus Torvalds
- Before that, he made the Linux Kernel
- Here is a [Ted talk](#)
- Here is his [GitHub](#)
- Here is the [source code for Git](#)



# Why is it called Git?

“  
*I'm an egotistical bastard, and I name all my projects after  
myself*

- *Linus Torvalds*

# What is Git?



A **version control system** (or **VCS**) It takes snapshots of our projects

Gives us a project-wide undo button! A collaboration tool

It merges differences in our code for us

A local development tool Supports non-linear development

# What is Git?

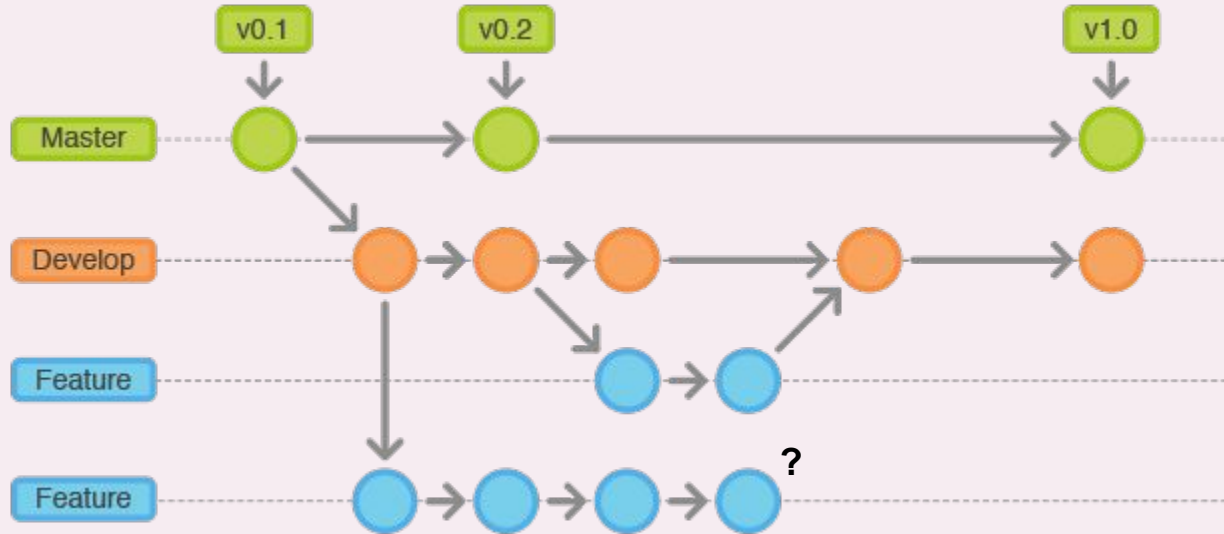


It's a tool for modern-day teamwork - collaboration among people, working asynchronously, on a shared body of work.

It saves us from moving floppy disks around, or saving lots of copies of the one file.

More people == more likely to use it

# Git Workflow





# Why use it?

You make a change and realise it was a horrible mistake? ***Git can undo it***

You want to figure out where everything went wrong? ***Git will show you***

You want to try out a new innovative feature that will probably destroy everything?  
***Git can protect you***

You want to work with a bunch of people? ***Git will make that easier***



# Terminology

**Repository** - A project

**Branch** - A version of your project

**Origin** - A place where your code is stored

**Add** - Tell Git to pay attention to a file(s)

**Commit** - Tell Git to take a snapshot of a file(s)

**Push** - Tell Git to take all of the code that it has locally and put it up on GitHub



# Terminology

**Merge Conflict** - When two pieces of code can't be automatically merged, you get one of these - you need to decide what you want

**Fork** - Your copy of someone else's GitHub repository **Pull Request** - When you request to have a project include your code

**Clone** - When you take code from GitHub and get an exact local copy on your computer

# Development Env Check



## Part 3. Git Configuration

1. To check if your Git installation was successful, open a new terminal window and try to run Git from the command line:

```
$ git --version
```

The output should be something like this:

```
$ git --version  
git version 2.X.X
```



# Development Env Check



## Part 3. Git Configuration

1. Next, you'll need to provide Git with your name and email. Make sure to use the same email address that you registered at <https://git.generalassemb.ly>:

```
$ git config --global user.name "Your Name"
$ git config --global user.email your.name@example.com
$ git config --global core.editor "subl -n -w"
```

These identifiers will be added to your commits and show up when you push your changes to [GitHub](#) from the command line!

\*set up **Sublime** as your default editor <https://goo.gl/rvf7xA>

# Install Fest

15<sub>min</sub>

Install the following software for your OS:

- Install Ananconda 3.6 (<https://www.anaconda.com/download/> )
- Install Git (<https://git-scm.com/downloads>)
- Install sublime text 3 (<https://www.sublimetext.com/3> )
- Make a github account (<https://github.com> )
- Make a Slack account (<https://slack.com/>, @pycrashcourse )

WiFi  
GA-Guest  
yellowpencil

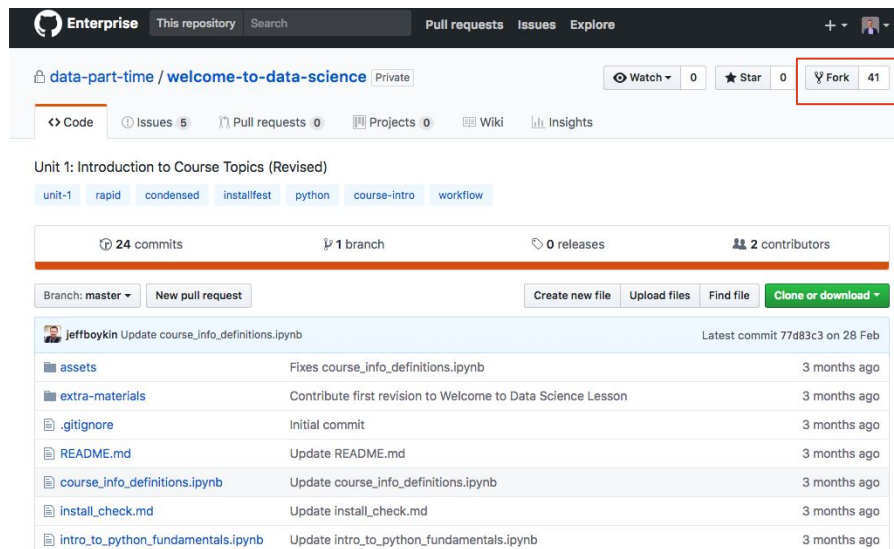
# GitHub



Navigate here:

<https://github.com/kilaorange/pycrashcourse.git>

Fork repo:



The screenshot shows the GitHub interface for the repository 'data-part-time / welcome-to-data-science'. The repository is private and has 0 stars and 41 forks. The 'Fork' button is highlighted with a red box. Below the repository name, there are tabs for 'Code', 'Issues', 'Pull requests', 'Projects', 'Wiki', and 'Insights'. The 'Code' tab is selected. The repository has 24 commits, 1 branch, 0 releases, and 2 contributors. A table of commits is shown, with the latest commit being 'Update course\_info\_definitions.ipynb' by jeffboykin, committed 3 months ago.

Commit	Author	Message	Time
77d83c3	jeffboykin	Update course_info_definitions.ipynb	3 months ago
		Fixes course_info_definitions.ipynb	3 months ago
		Contribute first revision to Welcome to Data Science Lesson	3 months ago
		Initial commit	3 months ago
		Update README.md	3 months ago
		Update course_info_definitions.ipynb	3 months ago
		Update install_check.md	3 months ago
		Update intro_to_python_fundamentals.ipynb	3 months ago

(<https://git.generalassemb.ly/data-part-time/welcome-to-data-science>)

# Github



Use the terminal to clone repo:

```
$git clone https://github.com/<your_name>/pycrashcourse.git
```

```
$cd pycrashcourse
```

Configure your local clone to point to the official course repository

```
$ git remote -v
```

```
$ git remote add upstream https://github.com/kilaorange/pycrashcourse.git
```

```
$ git remote -v
```

# GitHub



Ensure you're in the master branch

```
$ git checkout master
```

Grab the latest changes from the master

```
$ git fetch upstream
```

Merge the master changes with your repo

```
$ git merge upstream/master
```

*WARNING: Be careful not to overwrite files you have already changed in your repo unless you want to replace them with the master versions!*

*(Consider renaming yours or doing a PULL REQUEST.)*

# Git Commands



git init

git add README.md

git add -A

git commit -m "Your commit message"

git status

git log

git reset --hard .....

# Install Fest

15<sub>min</sub>

Install the following software for your OS:

- Install Ananconda 3.6 (<https://www.anaconda.com/download/> )
- Install Git (<https://git-scm.com/downloads>)
- Install sublime text 3 (<https://www.sublimetext.com/3> )
- Make a github account (<https://github.com> )
- Make a Slack account (<https://slack.com/>, @pycrashcourse )

WiFi  
GA-Guest  
yellowpencil





# Terminal 101



# Terminal 101

- What is the terminal => interactive console
- Why do we need it => more powerful than UI
- TL;DR - What are the key commands (all lowercase, case sensitive):
  - cd, mv, cp, mkdir, pwd, rm # Navigating the terminal
  - ls, ls -lsa
  - grep, ps aux, piping |
  - nano, vim, touch
  - wc -l
  - tar czvf, tar xzvf, zip, unzip # Compressing Files
  - ssh, scp
  - whoami, which

# Terminal 101 - What is it?

A way to manipulate and interact with your computer It's entirely text-based

Not the **W.I.M.P** (Windows, Icons, Menus and Pointers) style!

# Terminal 101 - Why use it?

- It's (eventually) very fast
- It's automatable and flexible No interruptions
- It gives us what we expect Sometimes it is the only way
  - Command Line Interaction (C.L.I.)
  - Web servers

# Terminal 101 - The Bash Shell

- Bash is a regular program on your computer It was created to take commands from you
- We talk to it using the **Bash Shell Language**
- When I say "shell", it's just that program we were talking about before
- It's an interface to interact with other programs

# Terminal 101 - What can you do with it?

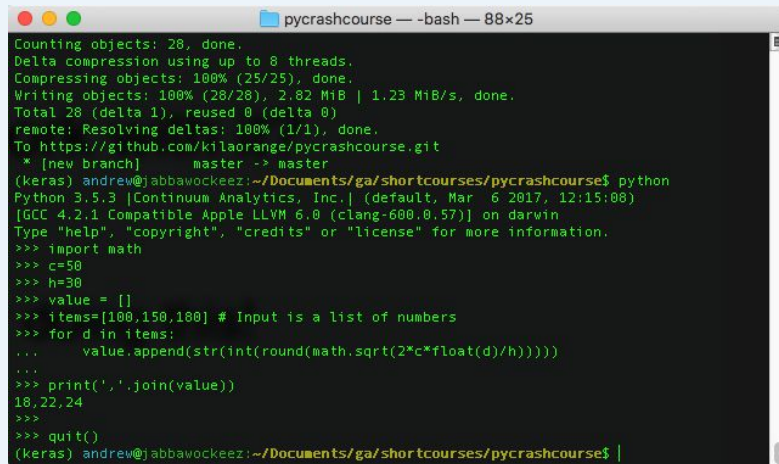
- Most of you will have a lot of experience with the WIMP (Windows, Icons, Menus, Pointer) style of system
- That's not the only way. We are going to be using a text-only "console" or "terminal"
- This is going to seem alien and primitive but you will soon see the power!

# Terminal 101 - What can you do with it?

- Anything! Run programs to make all sorts of changes
  - Editing files and images
  - Converting files between types
- Creating back-ups
- Making and copying files
- Downloading, compiling, and running programs
- We can do a lot more with the Terminal!

# Terminal 101 - How do you work with it?

- Non-interactively
- Running scripts. We are already doing this!
- Interactively
- Opening up a **REPL**



```
pycrashcourse — -bash — 88x25
Counting objects: 28, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (25/25), done.
Writing objects: 100% (28/28), 2.82 MiB | 1.23 MiB/s, done.
Total 28 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/kilaorange/pycrashcourse.git
 * [new branch]      master -> master
(keras) andrew@jabbawockeez:~/Documents/ga/shortcourses/pycrashcourse$ python
Python 3.5.3 [Continuum Analytics, Inc.] (default, Mar  6 2017, 12:15:00)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>> import math
>>> c=50
>>> h=30
>>> value = []
>>> items=[100,150,180] # Input is a list of numbers
>>> for d in items:
...     value.append(str(int(round(math.sqrt(2*c*float(d)/h))))))
...
>>> print(', '.join(value))
18,22,24
>>>
>>> quit()
(keras) andrew@jabbawockeez:~/Documents/ga/shortcourses/pycrashcourse$ |
```



# Terminal 101 - Commands

[tab]	# Autocomplete command
pwd	# Where am I? The programmer's "um"
ls	# List all files in the current directory
cd	# Change Directories
mkdir	# Make a Directory
rmdir	# Remove an empty directory
rm	# Remove a file or a directory [There is no undo]
touch	# Create a file
open	# Open a file in the default application
code	# Open the VSCode Editor (atom will open in Atom)
say	# Make your computer talk

# Advanced Commands

ls -lsa                                   # long format, system blocks, view hidden files (.)

Ls -lt                                   # long format, sort by time modified (most recently modified first)

ps aux | grep <keyword>   # process status, all users, usernames, even those without controlling terminal, search for keyword e.g. "python"

nano, vim                               # editors

# Advanced Commands

`wc -l` # count number of lines in file e.g. a csv

`tar czvf, gzip, zip` # compress into archive

`tar xzvf, unzip` # decompress file

`ssh, scp` # Secure shell, secure copy

`whoami, which <keyword>` # list username, which python

# Activity - Terminal 101

1. Navigate to your home directory with **cd ~**
2. Use **pwd** to discover its name
3. Use **ls** to see what is in your home directory
4. Use **cd ~** to navigate back down to your home directory
5. Create a new directory with **mkdir** called **sandbox**
6. Navigate to your downloads with **cd ..** or **cd ~/Downloads**
7. Create a file in **Downloads** with **touch** called file.txt
8. Copy file.txt to your sandbox with **cp file.txt ~/sandbox/**
9. Rename **file.txt** to **hello.py** with **mv file.txt hello.py**
10. Change directory to **cd ~/sandbox**
11. Make a new file called **fake.py** using **nano fake.py**
12. Inside the file type `print("hello world!")`, then push Ctrl+o, Enter, Ctrl+x to save and exit
13. Make a directory called 'crash\_course' using **mkdir crash\_course**
14. Remove **fake.py** and **crash\_course** with **rm**, you will need **-f** for one of the removals
15. Well done you've finished!



EXERCISE

# Additional Resources

Read these for more info:

- [Quick Left's Tutorials](#) - start from the bottom!
- [Learn CLI the Hard Way](#)
- Track down the [Terminal City Murderer](#)
- [40 Terminal Tricks and Tips](#)

# Getting started with Python



# Getting started with Python

What can we expect?

- What is python
- Why do we use it
- What is it good for
- Start the REPL
- Make a hello world script
- Start jupyter
- Jupyter tips and tricks + keyboard shortcuts

---

# WHERE IS PYTHON USED?

---

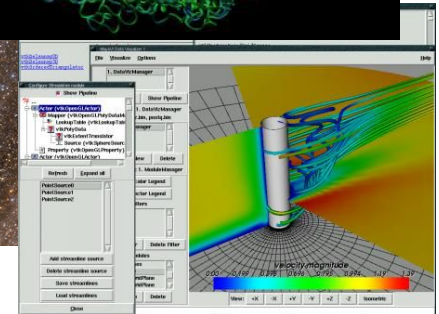
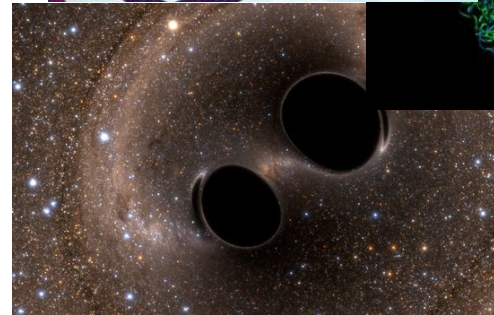
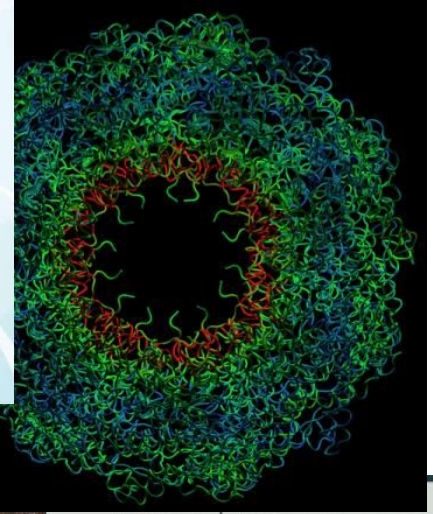
- › Everywhere! Industry & Academia
- › Art Middleware – Tools and Plugins
- › Research
- › Web Development and Web Applications
- › Game Development
- › Windows Applications
- › You name it!





# EXAMPLES

- Industry
  - Drug discovery
  - Financial services
  - Films and special effects
- Academia
  - Gravitational waves
  - Scientific visualisation
  - Biomolecule simulation



---

# DISCUSSION

Other places  
where is  
python used?

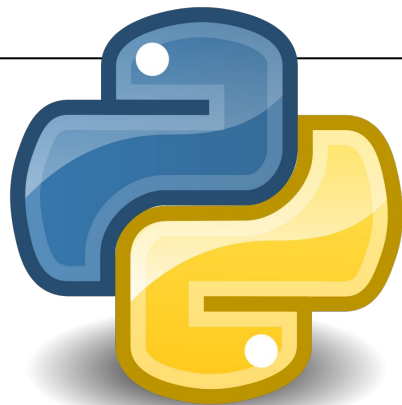


---

# 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.



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

Python

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

It is easy to read and understand.

---

# PYTHON: INTERACTIVE SHELLS V SCRIPTS

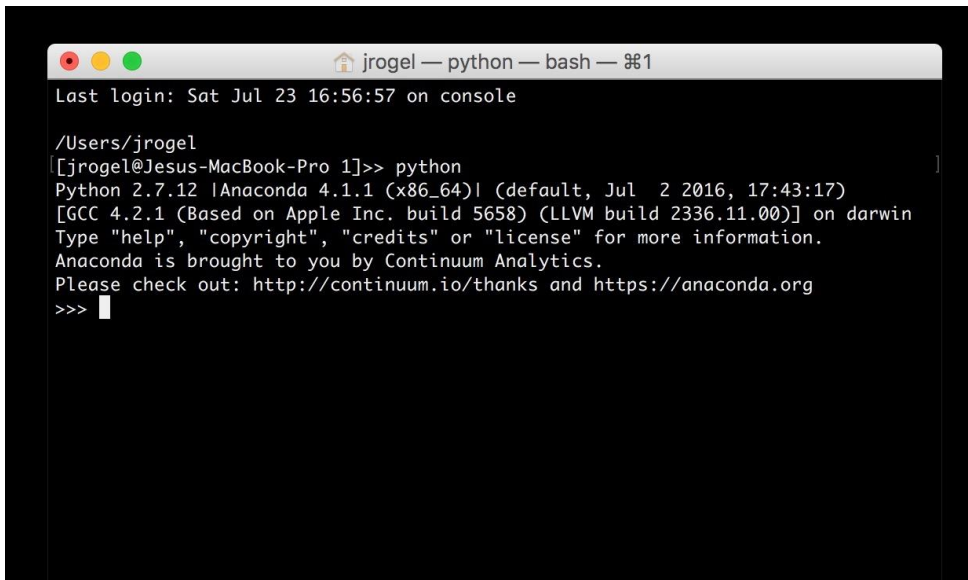
---

- › 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”**



```
jrogel — python — bash — ㉿1
Last login: Sat Jul 23 16:56:57 on console

/Users/jrogel
[jrogel@Jesus-MacBook-Pro 1]>> python
Python 2.7.12 |Anaconda 4.1.1 (x86_64)| (default, Jul  2 2016, 17:43:17)
[GCC 4.2.1 (Based on Apple Inc. build 5658) (LLVM build 2336.11.00)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
Anaconda is brought to you by Continuum Analytics.
Please check out: http://continuum.io/thanks and https://anaconda.org
>>> 
```



# 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”**

```
jrogel — python • python.app /Applications/anaconda/bin/ipython — bash — %1

/Users/jrogel
[jrogel@Jesus-MacBook-Pro 5]>> ipython
Python 2.7.12 |Anaconda 4.1.1 (x86_64)| (default, Jul 2 2016, 17:43:17)
Type "copyright", "credits" or "license" for more information.

IPython 4.2.0 -- An enhanced Interactive Python.
?      -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help    -> Python's own help system.
object? -> Details about 'object', use 'object??' for extra details.

[In [1]: 1+1
Out[1]: 2

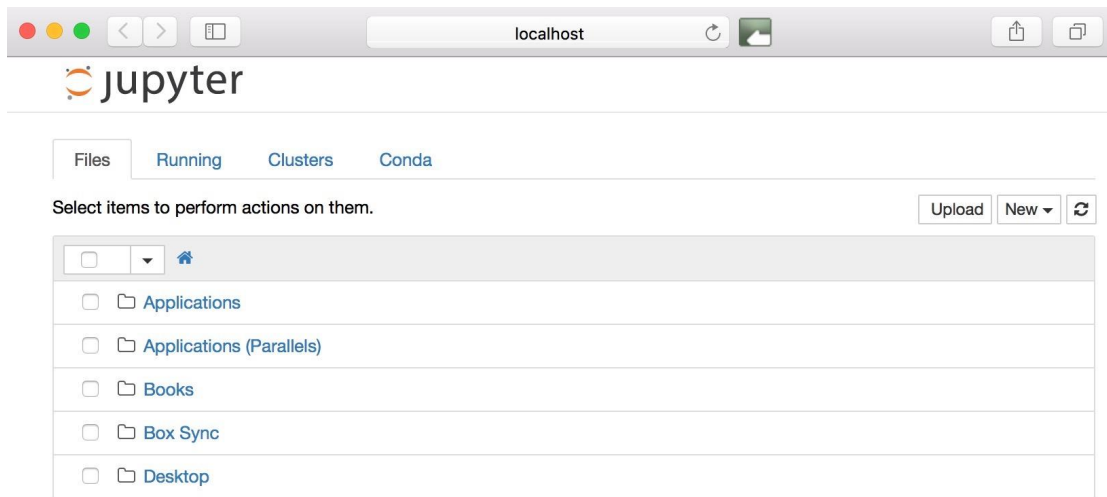
[In [2]: print("Hello World!")
Hello World!

[In [3]: ]
```



# JUPYTER NOTEBOOK

- › A Jupyter notebook is a web interface that lets us use formatting alongside our code. It is the extremely common and very useful! You can launch it by typing:  
› **“jupyter notebook”**



---

# PYTHON: INTERACTIVE SHELLS V SCRIPTS

---

- › 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:

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

Python

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

```
> python hi.py
```

---

# 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:

```
1      def main():  
2          print("hello world")  
3  
4      if __name__ == '__main__':  
5          main()
```

Python

# More Python



# Up next:

- Variables, constants
- Data types
- Logical operators
- Control structures (For, While, If, List comprehension)
- Importing new libraries
- Programming paradigms (OO) and pseudo code
- Functions 1a (basic function definition and usage, e.g. numpy, scipy)
- Globals variables
- Homework assignment

# Installing libraries



# 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 3"** or **"Root"** (depending on your installation of Python)
4. Voilà, you are ready to type the commands we will cover

---

# PACKAGES

---

- Libraries of code written to solve particular set of problems
- Can be installed @ terminal with: **pip install <package name>** or **conda install <package name>**
- You can install packages in Jupyter Notebook using
- **!** pip --user install <package name>
- Ever used Excel? How do yo 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**



# PACKAGES

## Popular Data Science Packages



---

# 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](#)



# 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:

TRY IT  
YOURSELF

```
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)
```

Python

EXERCISE



# Variables and Data Types



# TYPES, VARIABLES, ASSIGNMENT

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



```
1  # variable assignments
2  x = 1.0
3  my_variable = 12.2
4  type(x)
5
6  y = 1
7  type(y)
8
9  b1 = True
10 type(b1)
11
12 s = "String"
13 type(s)
```

Python

# YOUR TURN

› Try the following in your Jupyter Notebook:

```
1 import types
2 print(dir(types))
3
4 1+2, 1-2, 1*2, 1/2
5
6 1.0+2.0, 1.0-2.0, 1.0*2.0, 1.0/2.0
7
8 # Comment
9
10 # Comparison: >, <, <=, <=, ==
11 2 > 1
12
13 # Testing for equality
14 2 == 2
```

Python



EXERCISE



# LISTS

- › Lists are collections of objects
- › They can be changed



```
1 l = [1,2,3,4]
2 print(type(l))
3 print(l)
4 print(l)
5 print(l[1:3])
6 print(l[:2])
7
8 # Python starts counting from 0
9 print(l[0])
```

Python

EXERCISE

# TUPLES

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



```
1 | point = (10, 20)
2 | print(point, type(point))
3 |
4 | x, y = point
5 | print("x =", x)
6 | print("y =", y)
```

Python

EXERCISE

# DICTIONARIES

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



```
1 | params = {"parameter1" : 1.0, "parameter2" : 2.0,  
2 | "parameter3" : 3.0,}  
3 | print(type(params))  
4 | print(params)
```

Python

EXERCISE

# Pseudocode



---

# PSEUDOCODE

---

- › Pseudocode allows us to represent a program concisely.
- › The only thing you need is a statement to show where you are starting and where you are ending a program.
- › Calculate and print the average of three numbers: 5, 10, and 15.

```
1  Start
2      num1 = 5
3      num2 = 10
4      num3 = 15
5      sum = num1 + num2 + num3
6      average = sum/3.0
7      print average
8  End
```

---

# ACTIVITY: WRITE YOUR OWN (PSEUDO)CODE

---



## DIRECTIONS

1. Create some pseudocode for the following tasks
  1. Create a short script that will calculate the area circle with radius  $r$ .
  2. Calculate and print the square of a number. If the number is larger than 10 also calculate the cube.
  3. List the letters in the sentence "Python is awesome"

10<sub>min</sub>

## DELIVERABLE

Pseudocode explaining the necessary steps to achieve one of the tasks above. Present back to class.

# Programming Fundamentals



# 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 3"** or **"Root"** (depending on your installation of Python)
4. Voilà, you are ready to type the commands we will cover



---

# PYTHON PROGRAMMING FUNDAMENTALS

---

- › Understanding core programming concepts and why they are used is just as important as knowing how to write code.
- › Before we start, we'll review some basic programming concepts:
- › **Variables:** A symbolic name that stores a **value (some specific piece of information)**. Variables have different **types**.
- › For example: **`r = 3`**



---

# PYTHON PROGRAMMING FUNDAMENTALS

---

- › **Data Structures:** Data structure is a particular way of storing and organizing data in a computer so that it can be used efficiently. Some examples include:
  - › Lists
  - › Tuples
  - › Arrays
  - › Matrices
  - › Dataframes



---

# PYTHON PROGRAMMING FUNDAMENTALS

---

- › **Syntax:** The syntax of a programming language is the set of rules that define the combinations of symbols that are considered to be correctly structured programs in that language
- › The interrelationship of all of these elements make it possible for us to write programs to implement algorithms and solve problems



# Control Structures



# If Statements



---

# PYTHON PROGRAMMING FUNDAMENTALS

---

- › **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 pseudocode:

```
1 | IF a person is older than 18
2 | THEN they can drive
3 | ELSE they cannot drive
```

In Python we can write:

```
1 | if age_person > 18:
2 |     return "They can drive"
3 | else:
4 |     return "They cannot drive"
```

Python

# IF

---

Another example:

```
1 | A = 10
2 | B = 100
3 | if A>B:
4 |     print("A is larger than B")
5 | elif A==B:
6 |     print("A is equal to B")
7 | else:
8 |     print("A is smaller than B")
```

Python



# Logical Operators



# Logical Operators

## And (&)

```
If a == 1 and b == 1:  
    print('both a and b equal 1')
```

```
If a == 1 & b == 1:  
    print('both a and b equal 1')
```

## Or (|)

```
If a == 1 or b == 1:  
    print('One of a or b equal 1')
```

```
If a == 1 | b == 1:  
    print('One of a or b equal 1')
```

## Not (!)

```
If not a == 1:  
    print('a does not equal 1')
```

```
If a != 1:  
    print('a does not equal 1')
```

# For Statements



---

# FOR LOOP

---

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

Think of this as a repetitive action that has to be performed until further notice.

In pseudocode:

```
1 | FOR each user of a service in a list
2 |     PRINT greet them
```

In Python we can write:

```
1 | users = ["Jeff", "Jay", "Theresa"]
2 |
3 | for user in users:
4 |     print("Hello %s" % user)
```

Python

---

# FOR LOOP

---

**Tip:** When creating a for loop, make sure it's condition will always be met to help prevent an endless loop!

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

```
1  for x in [1,2,3]:  
2      print(x)  
3  
4  for key, value in params.items():  
5      print(key + " = " + str(value))
```

Python

# While Statements



# While Statements

A while loop statement in Python programming language repeatedly executes a target statement as long as a given condition is true.

```
count = 0
while (count < 9):
    print 'The count is:', count
    count = count + 1

print "Good bye!"
```

When the above code is executed, it produces the following result –

```
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
The count is: 5
The count is: 6
The count is: 7
The count is: 8
Good bye!
```

# List Comprehension





---

# LIST COMPREHENSION

---

List comprehension is 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:

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

Python

# Two ways of writing the same thing

```
# For Loop
msg = []
for x in range(0,10):
    msg.append(x**2)

print(msg)

>>> [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
# List Comprehension
msg = [x**2 for x in range(0,10) ]
print(msg)

>>> [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

# Functions



---

# 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.



---

# FUNCTIONS

---

Here is how you define a function in Python:

```
1 | def function_name(input1, input2...):  
2 |     1st block of instructions  
3 |     2nd block of instructions  
4 |     ...
```

Python

Let's define a function that returns the square of the input value:

```
1 | def square(x):  
2 |     """  
3 |     Return the square of x.  
4 |     """  
5 |     return x ** 2
```

Python

---

# FUNCTIONS

---

We can call this function as follows:

```
1  var1 = 7
2
3  var2 = square(var1)
4
5  print(var2)
```

Python



# Python functions

The components:

- Definition
- Name
- Arguments
  - Body
  - Return

```
def function_name(arguments, more_arguments):  
    function_body = "do some things"  
    return "something"
```

# More complicated functions

```
h = 10
r = 3
pi = 3.1415

volume_of_cyclinder = pi * (r*0.254)^2 * h*0.254

def in_to_m(inches):
    meters = inches * 0.0254
    return meters

volume_of_cyclinder = pi * in_to_m(r)^2 * in_to_m(h)
```



# Writing Programs



---

# WRITING PROGRAMS IN PYTHON

---

- Python is an *interpreted* language, which means you can run the program as soon as you make changes to the file.
- This makes iterating, revising, and troubleshooting programs is much quicker than many other languages.
- Let us create a complete program that will calculate the area circle with radius  $r$ .

```
1  # We are importing the value of pi from
2  # that module - Easy to read, right?
3  from math import pi
4
5  def circ_area(r):
6      return pi * r**2
7
8  r = 3
9  area = circ_area(r)
10 print(area)
```

Python

---

# ACTIVITY: WRITE YOUR OWN (PSEUDO)CODE

---



## DIRECTIONS

Turn your pseudocode from the exercise above into Python!

1. Calculate and print the square of a number. If the number is larger than 10 also calculate the cube.
2. List the letters in the sentence "Python is awesome"

## DELIVERABLE

Python code that executes the tasks above in Jupyter notebook.

# Handling Data



# INSTRUCTIONS

---

- › Let's open a new Jupyter notebook for this practice.
- › We will work in pairs.

1. Start Jupyter (you know how now) and navigate to the file called **“labs/Python101\_Part2\_GuidedPractice.ipynb”**
2. Open the file by clicking on the name
3. Voilà, you can start the Guided Practice



**Putting it all together!**



---

## CHOOSE YOUR POISON

---

# Choose your own adventure!

- › Given your interests and knowledge, which are you more interested in learning about:
  - › Practical applications of Python?
  - › Python fundamentals?



# OPTION 1 - Python Practical Applications

---



Independent practice

1. Start Jupyter (you know how now) and navigate to the file called **"labs/Python101\_Part2\_IndPractice.ipynb"**
2. Open the file by clicking on the name
3. Voilà, you can start the Guided Practice



---

# OPTION 2: PYTHON FUNDAMENTALS

---

## DIRECTIONS



### EXERCISE

1. Write Python code to complete the following tasks:
  1. A recipe you are reading states how many grams you need for the ingredient. Unfortunately, your store only sells items in ounces. Create a program to convert grams to ounces. [ounces = (28.3495231)grams]
  2. Read in a Fahrenheit temperature. Calculate and display the equivalent centigrade temperature. The following formula is used for the conversion:

$$C = (5 / 9) * (F - 32)$$

3. Calculate the amount obtained by investing the principal P for N years at the rate of R. The following formula is used for the conversion:

$$A = P * (1 + R) ^ N$$

## DELIVERABLE

Python code that executes the tasks above

# Homework Assignment



# Homework Assignment

Come prepared next  
Saturday with a Jupyter  
notebook containing the  
answers to these 3  
questions.

# Homework Assignment

1

## Question 9

### Level 2

#### Question:

Write a program that accepts sequence of lines as input and prints the lines after making all characters in the sentence capitalized.

Suppose the following input is supplied to the program:

Hello world

Practice makes perfect

Then, the output should be:

HELLO WORLD

PRACTICE MAKES PERFECT

#### Hints:

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

# Homework Assignment

2

## Question 13

### Level 2

#### Question:

Write a program that accepts a sentence and calculate the number of letters and digits.

Suppose the following input is supplied to the program:

hello world! 123

Then, the output should be:

LETTERS 10

DIGITS 3

#### Hints:

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

# Homework Assignment (Challenge)

3

## Question 18

### Level 3

#### Question:

A website requires the users to input username and password to register. Write a program to check the validity of password input by users.

Following are the criteria for checking the password:

1. At least 1 letter between [a-z]
2. At least 1 number between [0-9]
1. At least 1 letter between [A-Z]
3. At least 1 character from [\$#@]
4. Minimum length of transaction password: 6
5. Maximum length of transaction password: 12

Your program should accept a sequence of comma separated passwords and will check them according to the above criteria. Passwords that match the criteria are to be printed, each separated by a comma.

#### Example

If the following passwords are given as input to the program:

ABd1234@1,a F1#,2w3E\*,2We3345

Then, the output of the program should be:

ABd1234@1

# PYTHON CRASH COURSE

---

# Q&A

See you next week!



# **PYTHON PROGRAMMING CRASH COURSE**

**Day 2**

May 2018



# Agenda Day 2

- Review Homework Assignment
- Markdown
- Functions (Advanced)
- Project Setup
- Python Classes
- Unicode Pitfalls
- Connecting to Databases
- Datascience
- Machine Learning

# **Review Homework Assignment**



# Homework Assignment

1

## Question 9

### Level 2

#### Question:

Write a program that accepts sequence of lines as input and prints the lines after making all characters in the sentence capitalized.

Suppose the following input is supplied to the program:

Hello world

Practice makes perfect

Then, the output should be:

HELLO WORLD

PRACTICE MAKES PERFECT

#### Hints:

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

# Homework Assignment

1

## Question 9

### Level 2

#### Solution:

```
lines = []
while True:
    s = raw_input()
    if s:
        lines.append(s.upper())
    else:
        break;

for sentence in lines:
    print sentence
```

# Homework Assignment

2

Question 13

Level 2

Question:

Write a program that accepts a sentence and calculate the number of letters and digits.

Suppose the following input is supplied to the program:

hello world! 123

Then, the output should be:

LETTERS 10

DIGITS 3

Hints:

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

# Homework Assignment

2

## Question 13

### Level 2

#### Solution:

```
s = raw_input()
d={"DIGITS":0, "LETTERS":0}
for c in s:
    if c.isdigit():
        d["DIGITS"]+=1
    elif c.isalpha():
        d["LETTERS"]+=1
    else:
        pass
print "LETTERS", d["LETTERS"]
print "DIGITS", d["DIGITS"]
```

# Homework Assignment (Challenge)

3

Question 18

Level 3

Question:

A website requires the users to input username and password to register. Write a program to check the validity of password input by users.

Following are the criteria for checking the password:

1. At least 1 letter between [a-z]
2. At least 1 number between [0-9]
1. At least 1 letter between [A-Z]
3. At least 1 character from [\$#@]
4. Minimum length of transaction password: 6
5. Maximum length of transaction password: 12

Your program should accept a sequence of comma separated passwords and will check them according to the above criteria. Passwords that match the criteria are to be printed, each separated by a comma.

Example

If the following passwords are given as input to the program:

ABd1234@1,a F1#,2w3E\*,2We3345

Then, the output of the program should be:

ABd1234@1



# Homework Assignment (Challenge)

3

## Question 18

### Level 3

#### Solutions:

```
import re
value = []
items=[x for x in raw_input().split(',')]
for p in items:
    if len(p)<6 or len(p)>12:
        continue
    else:
        pass
    if not re.search("[a-z]",p):
        continue
    elif not re.search("[0-9]",p):
        continue
    elif not re.search("[A-Z]",p):
        continue
```

#### # cont...

```
elif not re.search("[$#@]",p):
    continue
elif re.search("\s",p):
    continue
else:
    pass
value.append(p)
print ",".join(value)
```

# Markdown



# Jupyter Markdown basics

- A plain text format
- An easy way to generate HTML
- Most commonly ends in the file extension **.md** GitHub and Slack both use it
- See [here](#) and [here](#) for an introduction to it Why am I showing this...

# Jupyter Markdown basics



Markdown is a way to style text on the web. You control the display of the document; formatting words as bold or italic, adding images, and creating lists are just a few of the things we can do with Markdown. Mostly, Markdown is just regular text with a few non-alphabetic characters thrown in, like `#` or `*`

## HEADERS

```
# This is an <h1> tag
## This is an <h2> tag
##### This is an <h6> tag
```

## EMPHASIS

```
*This text will be italic*
_This will also be italic_

**This text will be bold**
__This will also be bold__

*You can combine them*
```

## BLOCKQUOTES

```
As Grace Hopper said:

> I've always been more interested
> in the future than in the past.

As Grace Hopper said:

| I've always been more interested
| in the future than in the past.
```

## LISTS

### Unordered

```
* Item 1
* Item 2
  * Item 2a
  * Item 2b
```

### Ordered

```
1. Item 1
2. Item 2
3. Item 3
  * Item 3a
  * Item 3b
```

## IMAGES

```
![GitHub Logo](/images/logo.png)

Format: ![Alt Text](url)
```

## LINKS

```
http://github.com - automatic!

[GitHub](http://github.com)
```

## BACKSLASH ESCAPES

Markdown allows you to use backslash escapes to generate literal characters which would otherwise have special meaning in Markdown's formatting syntax.

```
\*literal asterisks\*
```

```
*literal asterisks*
```

Markdown provides backslash escapes for the following characters:

\ backslash	() parentheses
` backtick	# hash mark
* asterisk	+ plus sign
_ underscore	- minus sign (hyphen)
{ curly braces	. dot
[] square brackets	! exclamation mark

# Activity - Markdown

1. Open a new terminal and type:

```
$ cd Documents/pycrashcourse # or where ever your repo is stored  
$ source activate dat2018  
$ jupyter notebook
```

1. Create a new Jupyter notebook in python 3
2. Create a new cell by pushing 'a'
3. Push 'Esc' then 'm' to change the cell to markdown
4. Enter some markdown; e.g.

```
# This is the title  
- Here is the first bullet point  
- Here is the second bullet point
```

5. Run the cell or push 'Ctrl+Enter'
6. Save your workbook 'Ctrl+s'

 EXERCISE

# Functions (Advanced)



# Python functions Recap

The components:

- Definition
- Name
- Arguments
  - Body
  - Return

```
def function_name(arguments, more_arguments):  
    function_body = "do some things"  
    return "something"
```

# Python functions Recap

Write a function that accepts a number and prints the line “Like baby, baby, baby, oh,” that many times.

The function should return the line “I thought you'd always be mine”

Use the function to write a hit!





# Functions - namespace and scoping

## Namespace and scoping:

- Variables can exist in the scope of the entire script;
- Or just within a function

```
n = 0

def n_is_two():
    n=2
    print(n)

n_is_two()
print(n)
```

## Global variables:

- Variables can be referenced globally

```
n = 0

def n_is_three():
    global n
    n = 3
    print(n)

n_is_three()
print(n)
```

What happens if we reference a variable within a function without defining it?

```
n = 0

def what_is_n()
    print(n)
```

# Functions - namespace and scoping

## Exercise

Set the status of pluto as “planet”.

Write a function that returns the status of pluto.

Write a second function that updates the status to “dwarf planet”

```
>>> pluto_status()  
planet  
>>> change_status()  
>>> pluto_status()  
dwarf planet
```

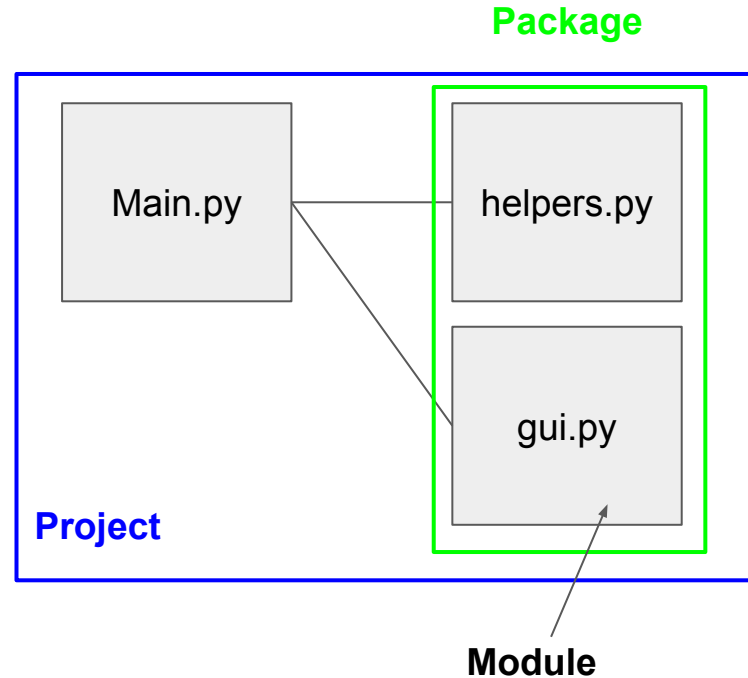


# Project Setup



# Project Setup and Modules

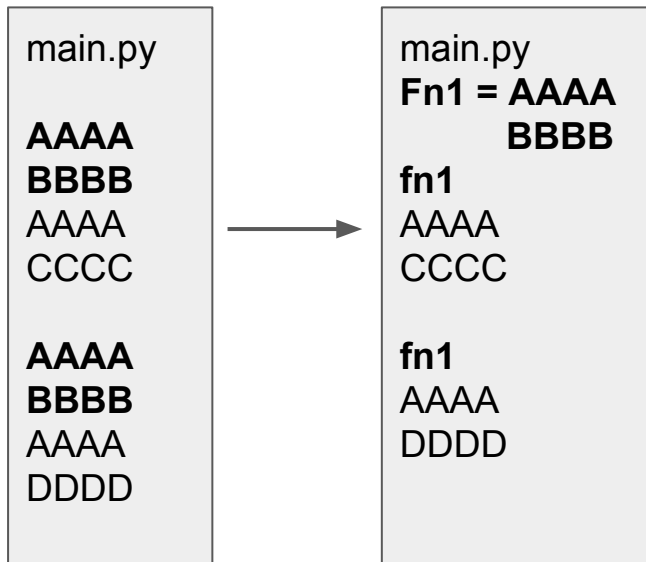
- What is a module - functions that will be reused;
- What is a project - a set of files that work together;
- What is a package - a set of modules;



# Single lines to functions

We start writing scripts rather than inline code at the terminal once the structure is complex: If, for, while.

When you want to use a block of code more than once or you want to constrain how a block is used. Convert it to a function. It also helps to make the code more readable.

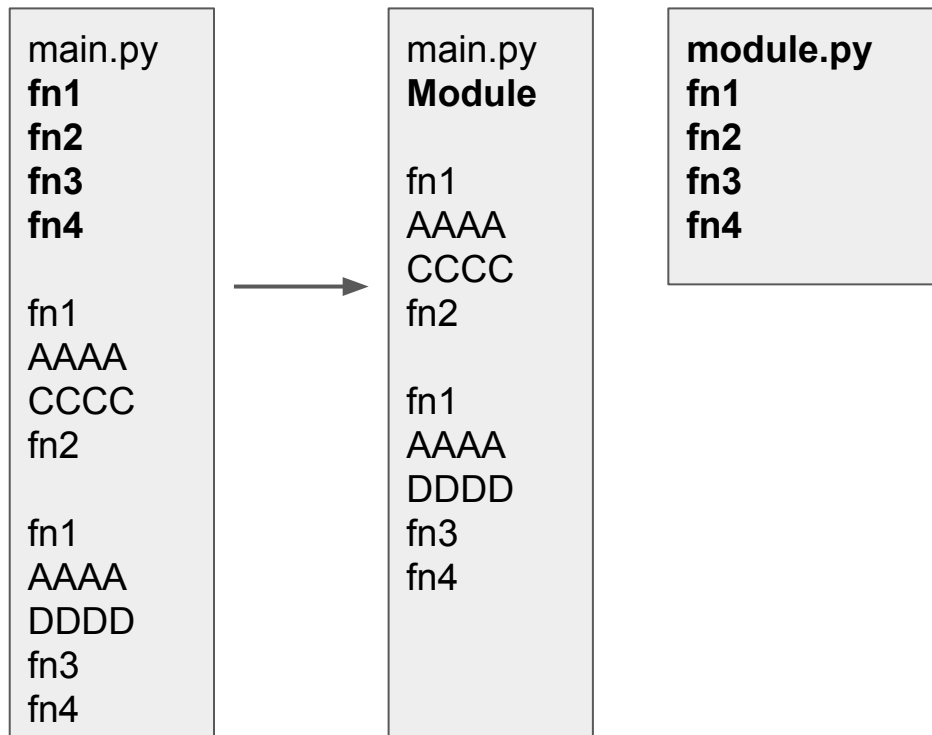


# Functions to modules

When you have lots of functions in your script, and you want to:

1. Use them in other code;
2. Tidy your main procedure;

You may group them into modules that contain functions with similar purpose.

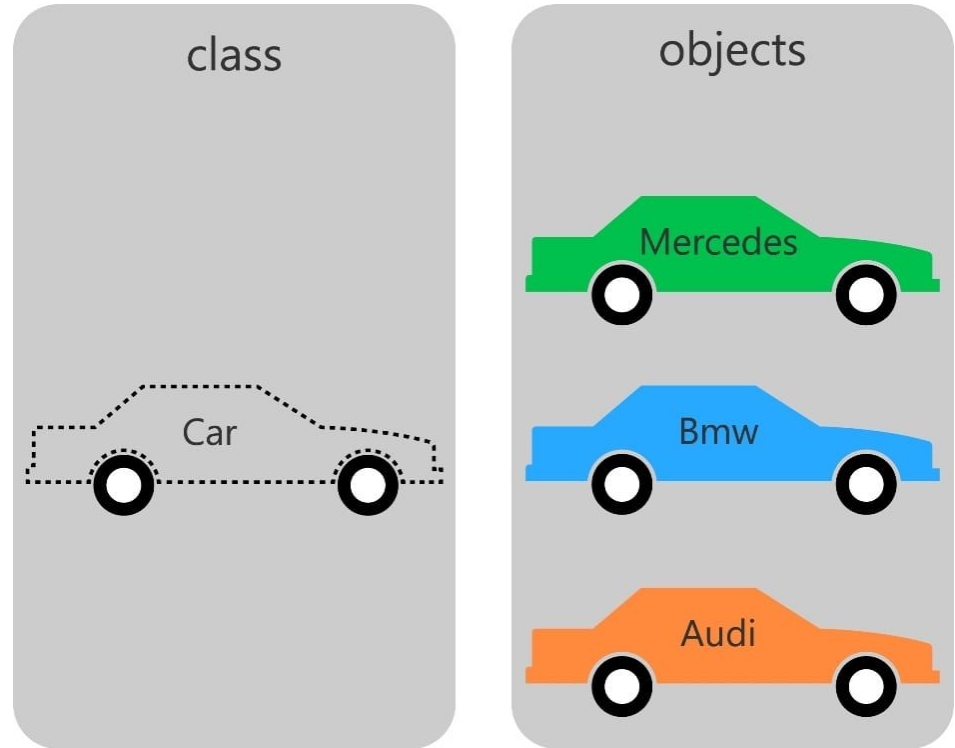


# Python Classes



# CLASS

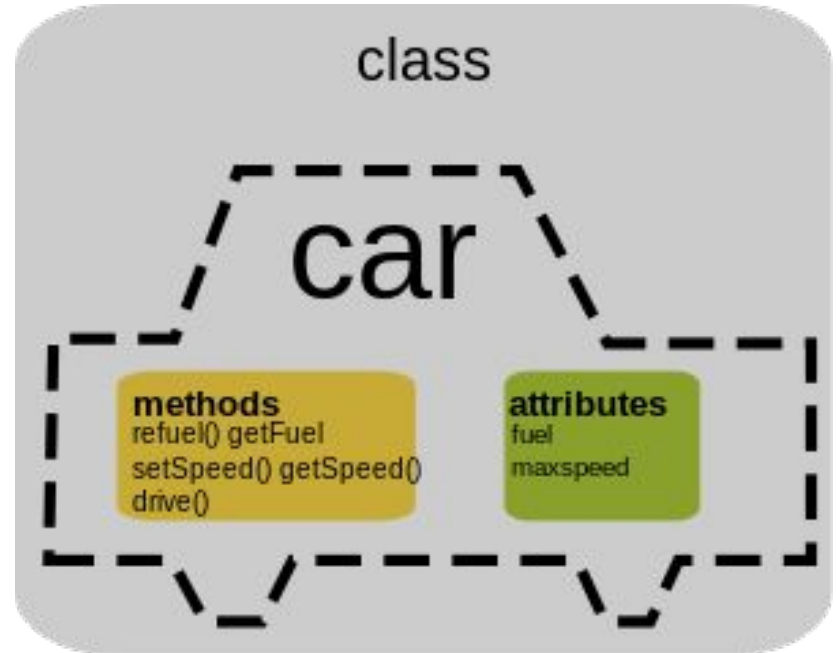
- › In Object Oriented Programming, a **class** is a template definition of the methods and variables in a particular kind of object.
- › In other words, an **object** is a specific *instance* of a **class**
- › The **object** contains actual values instead of variables.





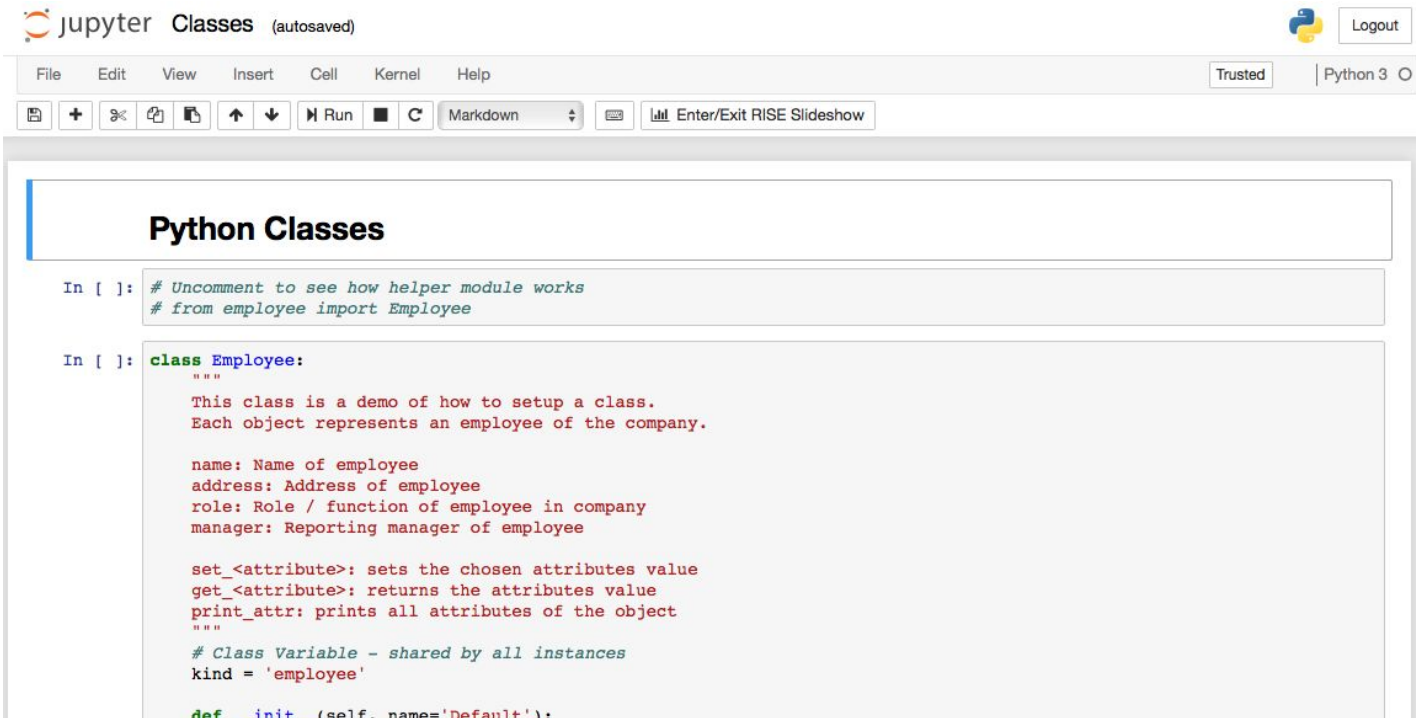
# Classes

- Classes are objects that have methods and attributes
- Methods are object functions
- And attributes are object variables



# Python Classes

See [pycrashcourse/classes/Classes.ipynb](https://pycrashcourse.com/classes/Classes.ipynb)



The screenshot shows a Jupyter Notebook interface. At the top, the title bar says "Jupyter Classes (autosaved)". On the right, there is a Python logo and a "Logout" button. Below the title bar is a menu bar with "File", "Edit", "View", "Insert", "Cell", "Kernel", and "Help". To the right of the menu bar are "Trusted" and "Python 3" indicators. Below the menu bar is a toolbar with icons for file operations, running, and a dropdown menu currently set to "Markdown". To the right of the toolbar is a button that says "Enter/Exit RISE Slideshow".

The main content area of the notebook has a title "Python Classes" in a large, bold font. Below the title, there are two code cells. The first cell contains a comment: `In [ ]: # Uncomment to see how helper module works` and `# from employee import Employee`. The second cell contains a class definition: `In [ ]: class Employee:` followed by a docstring, class variables, and a method definition.

```
In [ ]: # Uncomment to see how helper module works
# from employee import Employee
```

```
In [ ]: class Employee:
    """
    This class is a demo of how to setup a class.
    Each object represents an employee of the company.

    name: Name of employee
    address: Address of employee
    role: Role / function of employee in company
    manager: Reporting manager of employee

    set_<attribute>: sets the chosen attributes value
    get_<attribute>: returns the attributes value
    print_attr: prints all attributes of the object
    """
    # Class Variable - shared by all instances
    kind = 'employee'

    def __init__(self, name='Default'):
```

# Unicode Pitfalls





# Ticks

Further Reading:

<https://www.cl.cam.ac.uk/~mgk25/ucs/quotes.html>

U+0022	QUOTATION MARK	"	neutral (vertical), used as opening or closing quotation mark; preferred characters in English for paired quotation marks are U+201C and U+201D
U+0027	APOSTROPHE	'	neutral (vertical) glyph having mixed usage; preferred character for apostrophe is U+2019; preferred characters in English for paired quotation marks are U+2018 and U+2019
U+0060	GRAVE ACCENT	`	
U+00B4	ACUTE ACCENT	´	
U+2018	LEFT SINGLE QUOTATION MARK	‘	
U+2019	RIGHT SINGLE QUOTATION MARK	’	this is the preferred character to use for apostrophe
U+201C	LEFT DOUBLE QUOTATION MARK	“	
U+201D	RIGHT DOUBLE QUOTATION MARK	”	

# Ticks



“Cow”  
‘Chicken’  
`pig`

PEP8 Style Guide - consistent with your quotes  
(<https://www.python.org/dev/peps/pep-0008/> )

Why it's important:

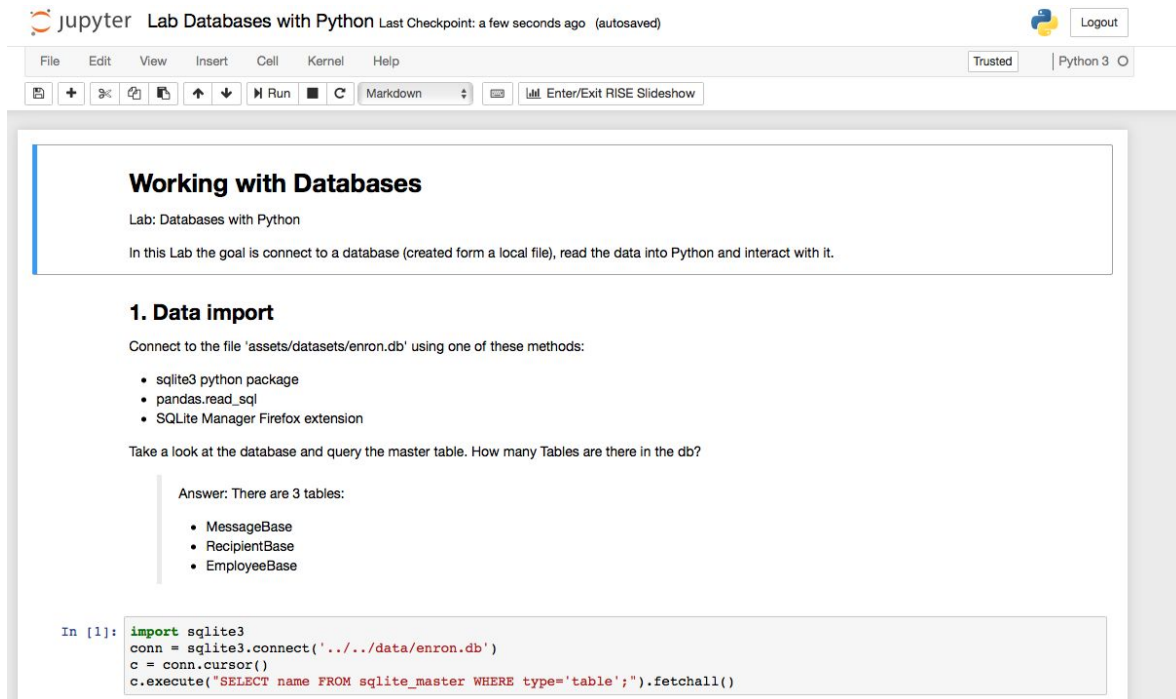
JSON = '{ "message" : "text" }'

# Connecting to Databases



# Connecting to Databases

Open “labs/Lab Databases with Python.ipynb” in jupyter notebooks and complete exercise.



jupyter Lab Databases with Python Last Checkpoint: a few seconds ago (autosaved) Logout

File Edit View Insert Cell Kernel Help Trusted Python 3

Working with Databases

Lab: Databases with Python

In this Lab the goal is connect to a database (created from a local file), read the data into Python and interact with it.

### 1. Data import

Connect to the file 'assets/datasets/enron.db' using one of these methods:

- sqlite3 python package
- pandas.read\_sql
- SQLite Manager Firefox extension

Take a look at the database and query the master table. How many Tables are there in the db?

Answer: There are 3 tables:

- MessageBase
- RecipientBase
- EmployeeBase

```
In [1]: import sqlite3
conn = sqlite3.connect('../data/enron.db')
c = conn.cursor()
c.execute("SELECT name FROM sqlite_master WHERE type='table';").fetchall()
```

EXERCISE

# Data Science





# Data Science

Open “Lab 001 Data Science.ipynb” in Jupyter Notebooks

Jupyter Titanic Worked Example Last Checkpoint: 01/18/2018 (autosaved) Logout

File Edit View Insert Cell Kernel Help Trusted Python 3

Run Enter/Exit RISE Slideshow

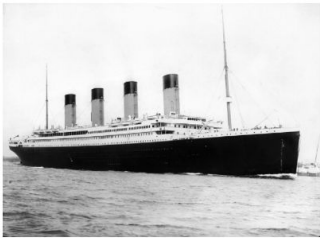
### Titanic Worked Example

Author: Andrew Szewc

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from string import ascii_letters
import seaborn as sns

%matplotlib inline
```

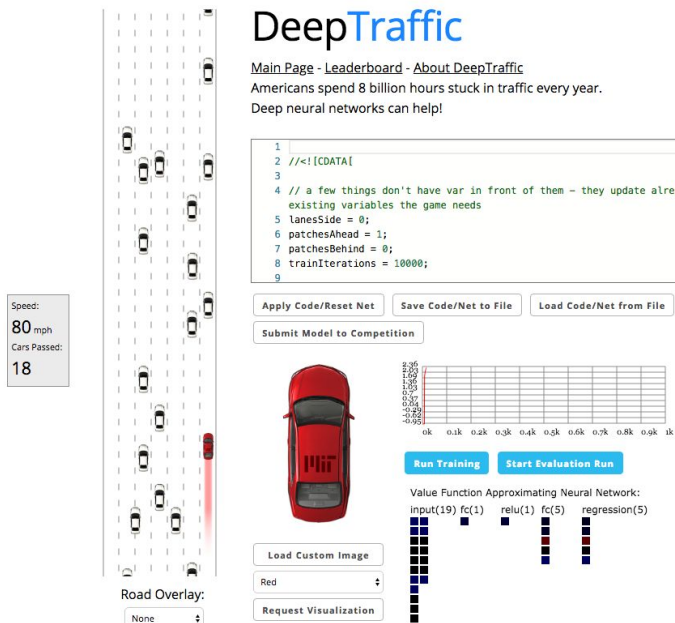


# Machine Learning



# Machine Learning

1. Go to <https://selfdrivingcars.mit.edu/deeptraffic/>
2. Make a neural network that can reach a speed above 65 mph.
3. Screenshot your top speed and submit it to pycrashcourse.slack.com #general channel



The screenshot displays the DeepTraffic web application. On the left, a vertical road simulation shows a red car moving through traffic. A box on the left indicates 'Speed: 80 mph' and 'Cars Passed: 18'. Below the road, there are controls for 'Road Overlay' (set to 'None') and 'Request Visualization' (set to 'Red').

The main area features the 'DeepTraffic' logo and a brief description: 'Main Page - Leaderboard - About DeepTraffic. Americans spend 8 billion hours stuck in traffic every year. Deep neural networks can help!'. Below this is a code editor with the following code:

```
1
2 //<![CDATA[
3
4 // a few things don't have var in front of them - they update already
5 existing variables the game needs
6 lanesSide = 0;
7 patchesAhead = 1;
8 patchesBehind = 0;
9 trainIterations = 10000;
10
```

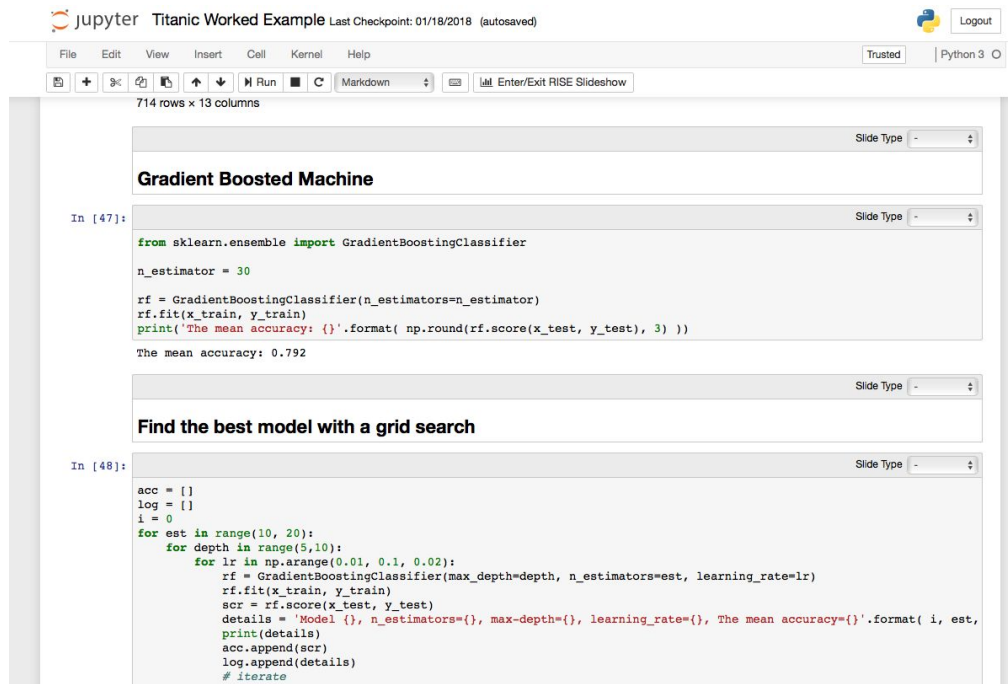
Below the code editor are buttons for 'Apply Code/Reset Net', 'Save Code/Net to File', 'Load Code/Net from File', and 'Submit Model to Competition'. A red car is shown in the center, with a speedometer and a graph of 'Value Function Approximating Neural Network' output. The graph shows a series of bars representing the output of the neural network over time. The x-axis is labeled '0k 0.1k 0.2k 0.3k 0.4k 0.5k 0.6k 0.7k 0.8k 0.9k 1k' and the y-axis is labeled '0.00 0.05 0.10 0.15 0.20 0.25 0.30 0.35 0.40 0.45 0.50 0.55 0.60 0.65 0.70 0.75 0.80 0.85 0.90 0.95 1.00'.

Below the graph are buttons for 'Run Training' and 'Start Evaluation Run'. Below these are labels for the neural network layers: 'input(19) fc(1) relu(1) fc(5) regression(5)'. There are also buttons for 'Load Custom Image' and 'Request Visualization'.

EXERCISE

# Machine Learning

Open “Lab 002 Machine Learning.ipynb” in Jupyter Notebooks



The screenshot shows a Jupyter Notebook interface with the title "Titanic Worked Example" and a status bar indicating "Last Checkpoint: 01/18/2018 (autosaved)". The notebook has a menu bar (File, Edit, View, Insert, Cell, Kernel, Help) and a toolbar with icons for file operations, running, and saving. The notebook content is divided into two sections, each with a title bar and a "Slide Type" dropdown.

**Gradient Boosted Machine**

In [47]:

```
from sklearn.ensemble import GradientBoostingClassifier

n_estimator = 30

rf = GradientBoostingClassifier(n_estimators=n_estimator)
rf.fit(x_train, y_train)
print('The mean accuracy: {}'.format( np.round(rf.score(x_test, y_test), 3) ))

The mean accuracy: 0.792
```

**Find the best model with a grid search**

In [48]:

```
acc = []
log = []
i = 0
for est in range(10, 20):
    for depth in range(5, 10):
        for lr in np.arange(0.01, 0.1, 0.02):
            rf = GradientBoostingClassifier(max_depth=depth, n_estimators=est, learning_rate=lr)
            rf.fit(x_train, y_train)
            scr = rf.score(x_test, y_test)
            details = 'Model {}, n_estimators={}, max-depth={}, learning_rate={}, The mean accuracy={}'.format( i, est,
                                                                                                        scr,
                                                                                                        details)
            print(details)
            acc.append(scr)
            log.append(details)
            # iterate
```

---

PYTHON CRASH COURSE

---

# CONCLUSION

---

# REVIEW & RECAP

---

- › In this workshop, we've covered the following topics:
  - › Git + Github How to
  - › Terminal 101
  - › 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!

---

## TAKEAWAYS

---

# LEARNING PLAN

Evaluate your python programming skills! How confident are you with:

- Github
- Terminal
- Python Syntax
- Programming Fundamentals
- Data Analysis

---

## TAKEAWAYS

---

# WHAT SHOULD YOU DO NEXT?

For beginner programmers:

- Go through [Learn Python the hard way](#)
- Familiarize yourself with the language by going through [A Beginner's Python Tutorial](#)



---

## TAKEAWAYS

---

# 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](#)

---

## TAKEAWAYS

---

# WHAT SHOULD YOU DO NEXT?

For anyone looking for a challenge :)

- Challenge yourself by tackling the [Python Challenge](#)

# PYTHON CRASH COURSE

---

# Q&A

---

PYTHON CRASH COURSE

---

# EXIT TICKETS

DON'T FORGET TO FILL OUT YOUR EXIT TICKET

# Appendix



# Fun Activities

Online game <http://www.pythonchallenge.com>

<https://selfdrivingcars.mit.edu/deeptraffic/>

<http://playground.tensorflow.org/>