

---

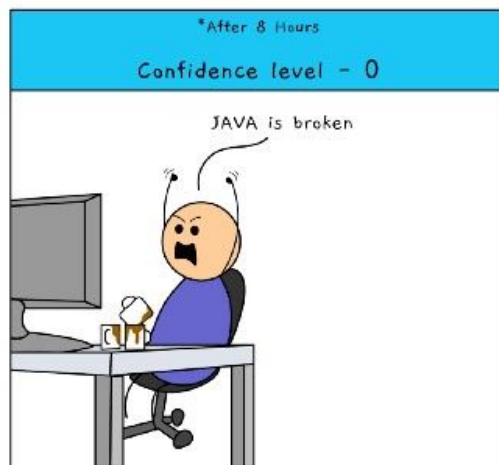
---

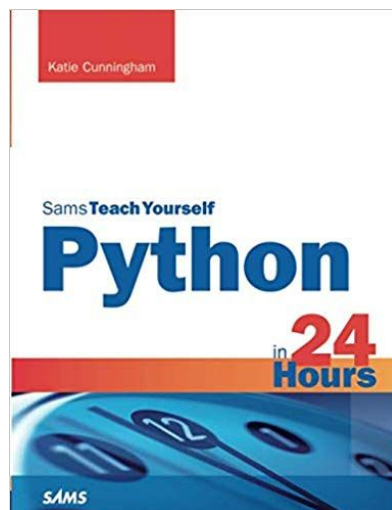
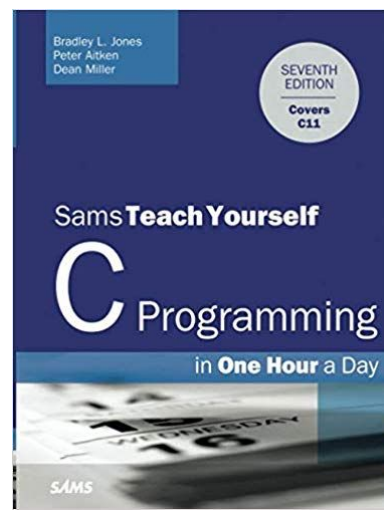
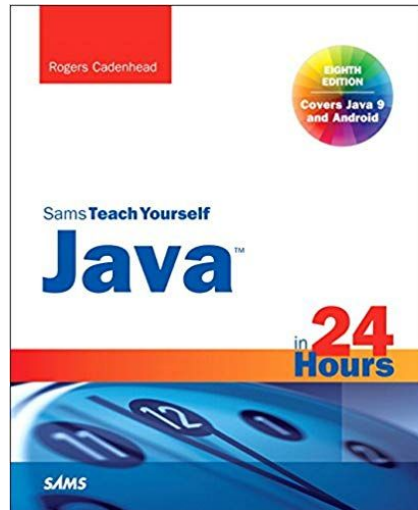
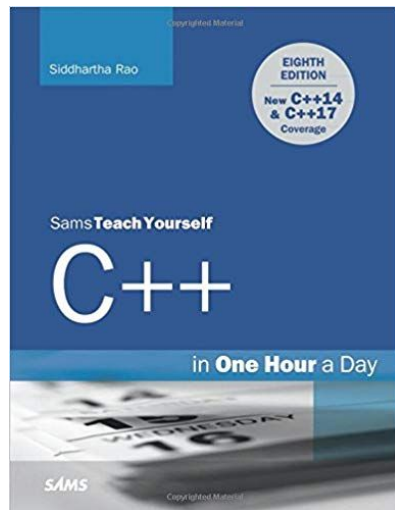
# DATA7001 Programming Bootcamp

— Nan Ye —

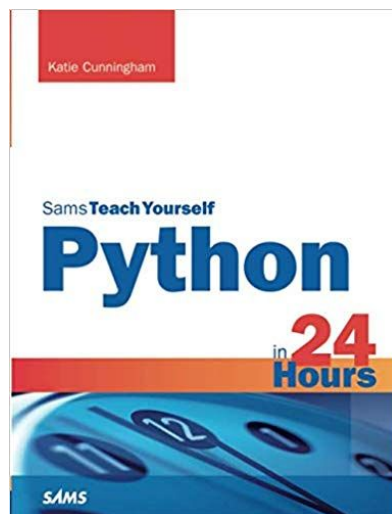
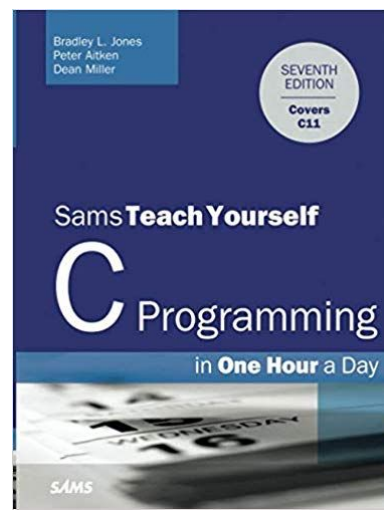
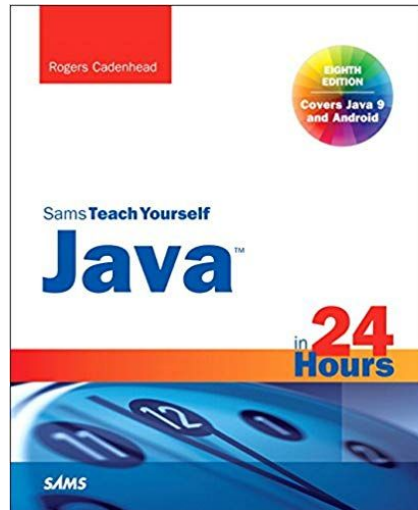
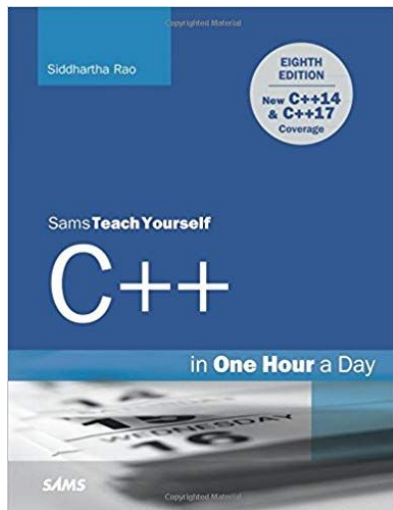
---

---





**Programming is easy!**



~~Programming is easy!~~

**We won't teach you how to program in 3 hours!**

# Teach Yourself Programming in Ten Years

Peter Norvig

## Why is everyone in such a rush?

Walk into any bookstore, and you'll see how to *Teach Yourself Java in 24 Hours* alongside endless variations offering to teach C, SQL, Ruby, Algorithms, and so on in a few days or hours. The Amazon advanced search for [\[title: teach, yourself, hours, since: 2000\]](#) and found 512 such books. Of the top ten, nine are programming books (the other is about bookkeeping). Similar results come from replacing "teach yourself" with "learn" or "hours" with "days."

The conclusion is that either people are in a big rush to learn about programming, or that programming is somehow fabulously easier to learn than anything else. Felleisen *et al.* give a nod to this trend in their book [How to Design Programs](#), when they say "Bad programming is easy. *Idiots* can learn it in *21 days*, even if they are *dummies*." The Abtruse Goose comic also had [their take](#).

## Translations

Thanks to the following authors, translations of this page are available in:

[Arabic](#)  
([Mohamed A. Yahya](#))

العربية

# How Can We Learn to Program Well

## *Deliberate Practice*

*"the most effective learning requires a **well-defined task** with an **appropriate difficulty level** for the particular individual, **informative feedback**, and opportunities for **repetition** and **corrections of errors**."*

# What You Can Expect From The Labs

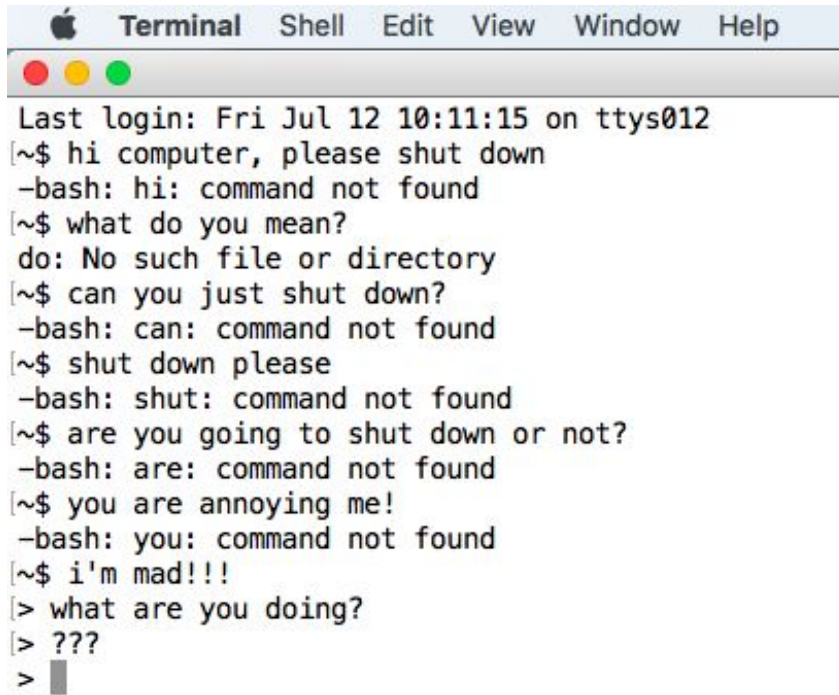
- Well-defined tasks
  - We will give you many well-defined lab exercises throughout the course.
  - But we expect you to self-study features of programming languages.
- Appropriate difficult level (well, not entirely)
  - The difficulties of the exercises will be gradually increasing
  - Some of you already have quite a bit of programming experience, some of you may have not programmed before. Do ask questions if you don't know how to get started!
- Informative feedback
  - Exercises are graded, and your TAs will be around in the lab to help.
- Repetition and error correction
  - Spend time to revisit the exercises and see whether you can get everything right.
  - You won't get more marks for a marked exercise, but you'll get better and better at solving new problems.

# What Will We Cover Here

- We focus on developing the **programmer's mindset** through examples.
- We will cover some basics of programming
  - This is to help you get started, and for the purpose of illustration.
- We will also cover some data science related tools/libraries, and also Web Scraping.

This bootcamp is about **how to approach programming** rather than learning everything about a programming language.



A screenshot of a macOS Terminal window. The title bar shows the Apple logo, the word "Terminal", and menu items: Shell, Edit, View, Window, Help. The window has three colored window control buttons (red, yellow, green) on the left. The text inside the terminal shows a user's conversation with a computer. The user says "hi computer, please shut down", "what do you mean?", "can you just shut down?", "shut down please", "are you going to shut down or not?", "you are annoying me!", and "i'm mad!!!". The computer responds with "command not found" for each of the user's commands. The user then says "> what are you doing?" and "> ???". The computer responds with "> " followed by a cursor.

```
Terminal  Shell  Edit  View  Window  Help
Last login: Fri Jul 12 10:11:15 on ttys012
[~$ hi computer, please shut down
-bash: hi: command not found
[~$ what do you mean?
do: No such file or directory
[~$ can you just shut down?
-bash: can: command not found
[~$ shut down please
-bash: shut: command not found
[~$ are you going to shut down or not?
-bash: are: command not found
[~$ you are annoying me!
-bash: you: command not found
[~$ i'm mad!!!
> what are you doing?
> ???
> █
```

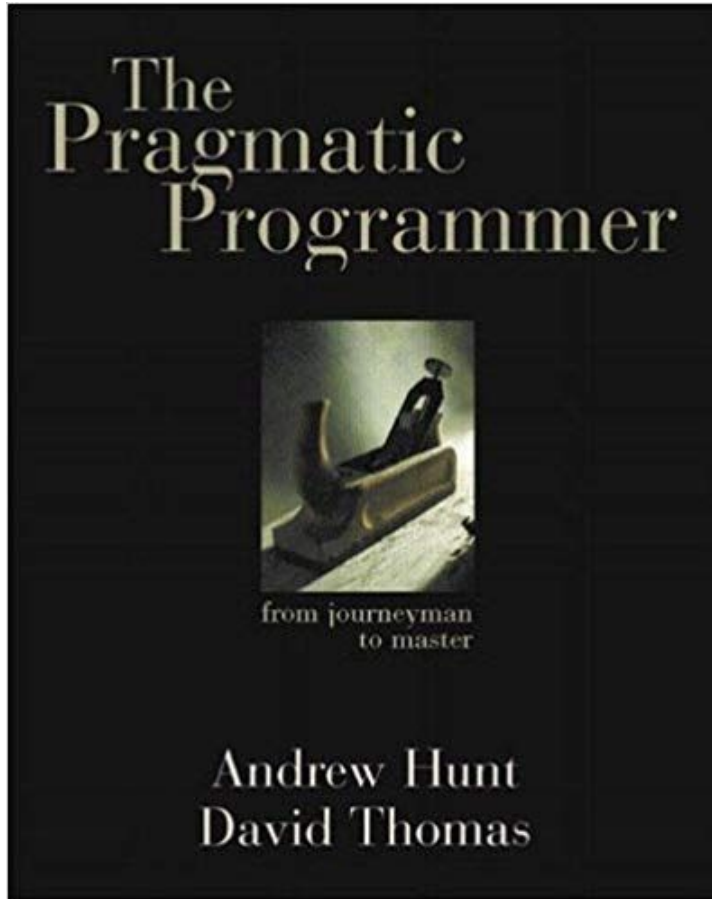
~~Program by wish~~

How NOT to be a programmer



The right way to put the computer to sleep immediately.

A computer is reliable and trustful, but only if you talk to it in its language.



## **First rule**

- You need to know that a computer is fairly limited in the way it can receive instructions.
- You need to know the syntax and semantics of the languages understood by computers.
- Think in the shoes of the computers!

## **Pragmatic programmer**

- Early adopter/fast adapter
- Inquisitive
- Critical thinker
- Realistic

# Checking Your Understanding

Which one is an objective of this bootcamp?

- A. Develop a programmer's mindset
- B. Learn how to use tools including Jupyter Notebook, Python, and Unix shell
- C. Learn how to use support vector machines
- D. Learn how to debug
- E. Teach you how to be a great programmer in 3 hours

Which describes the programmer's mindset?

- A. Programming requires speaking a language understandable by the computers
- B. A pragmatic programmer writes perfect programs
- C. A pragmatic programmer writes working programs
- D. A pragmatic programmer keeps on learning new tools

# Your Task

- Take notes for this bootcamp using a Jupyter Notebook
  - You'll note down important points noted down
  - You'll have some code snippets to illustrate those points
  - These are properly organized into sections

## Getting started

Let's start with calculating the sum of 1 to 10.

```
In [1]: sum(range(1, 11))
```

```
Out[1]: 55
```

## Basics of Python

Numbers

```
In [2]: x = 2
        y = 3.
        print(x, type(x))
        print(y, type(y))
        print(x + y, x - y, x * y, x / y, x//y)

2 <class 'int'>
3.0 <class 'float'>
5.0 -1.0 6.0 0.6666666666666666 0.0
```

**An example of expected outcome for this bootcamp**

Yours may be more detailed

# Overview

- **Getting started with your tools**
  - Jupyter Notebook, Python, Linux Shell
- Good coding practices
- Debugging your code
- Reading others' code
- Programming for data science
  - Matplotlib, pandas, NumPy
  - Data Sources
  - Web Scraping
- How to effectively ask us for help

# Getting Started with Your Tools

- Jupyter Notebook
  - Learn how to use Jupyter Notebook to develop, execute and document code, and communicate results.
  - This is a major framework that we will be using in this course.
- Python
  - Learn basics of Python (one of the most popular programming languages)
  - We only demonstrate how to pick up the basics, and will leave it to you to discover charms of Python.
- Unix shell
  - Learn ssh and basic terminal commands (geeky/nerdy way of connecting to another computer, copy/delete/view files,...)
  - When you become a power user, you'll be able to do many things a lot faster!

**A workman is known by his tools.**



TOOLBOX

# INTERACTIVE NOTEBOOKS: SHARING THE CODE

*The free IPython notebook makes data analysis easier to record, understand and reproduce.*

ILLUSTRATION BY THE PROJECT THINGS



BY HELEN SHEN

researchers to keep a detailed lab notebook for their computational work.

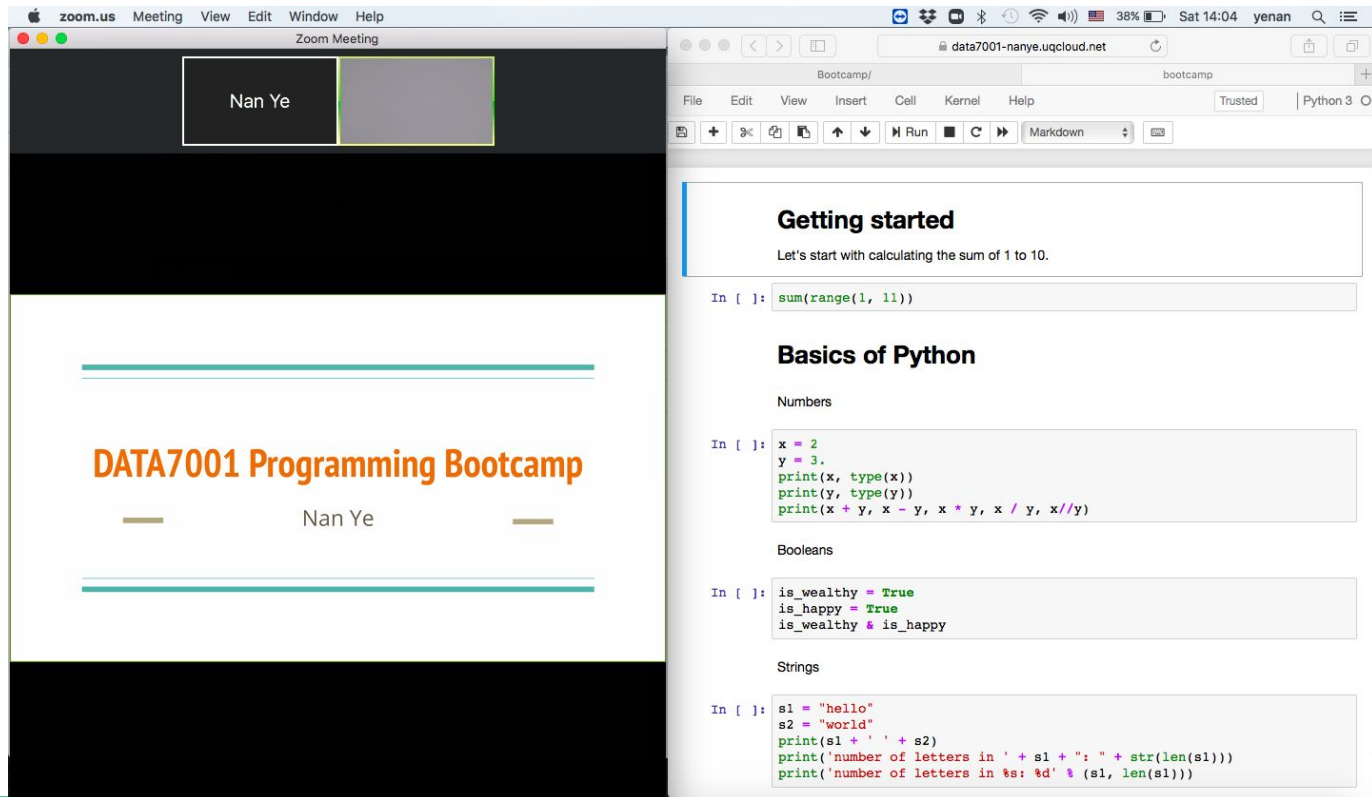
used increasingly by scientists who want to keep detailed records of their work, devise teaching

This is a Nature News article about Jupyter (then IPython) published in 2014.

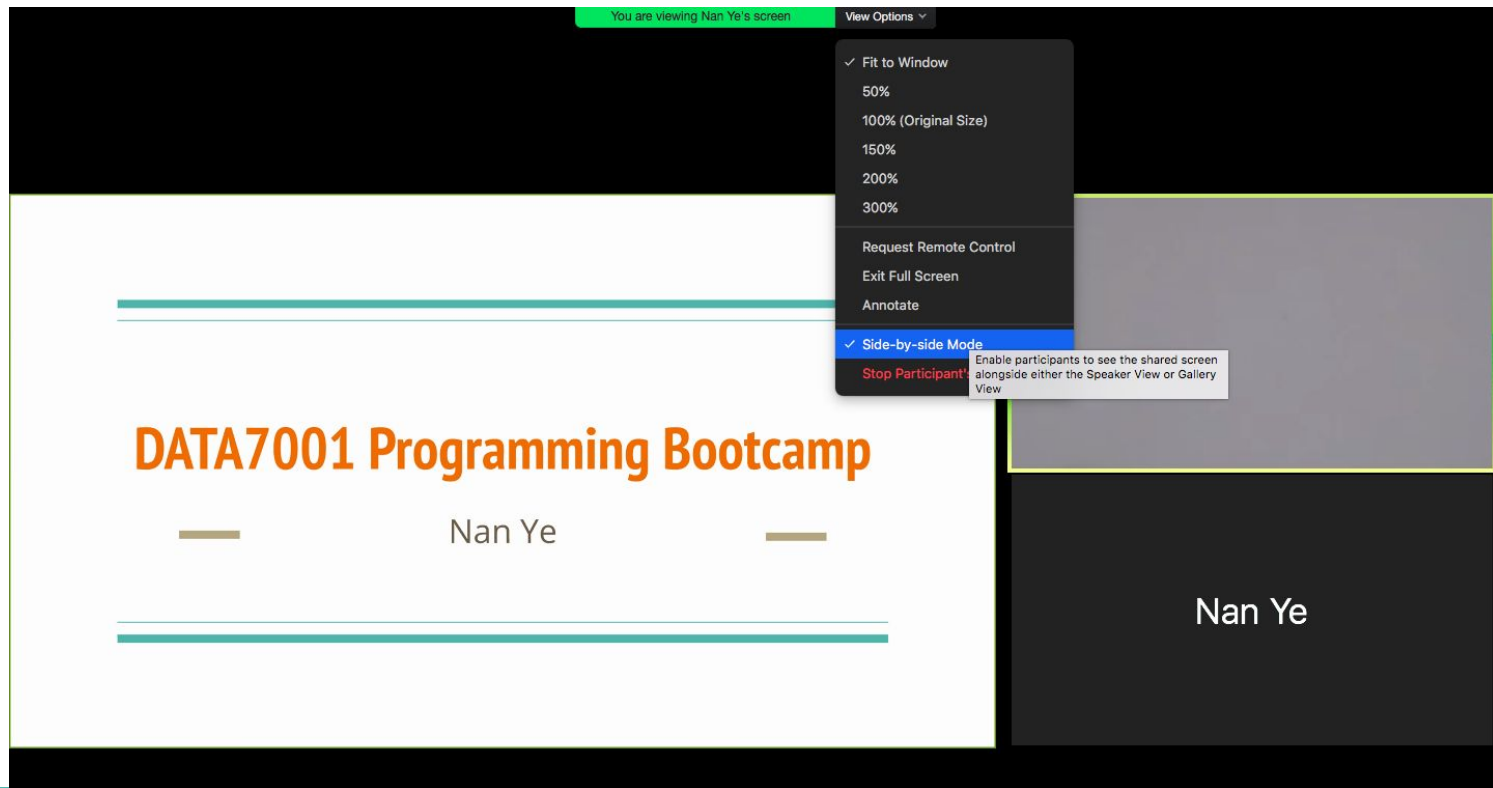
**Scientists love Jupyter!**

# Getting Read for Hands-on Exercises

- Put your zoom window and browser window side by side



- Step 1. Untick side-by-side mode

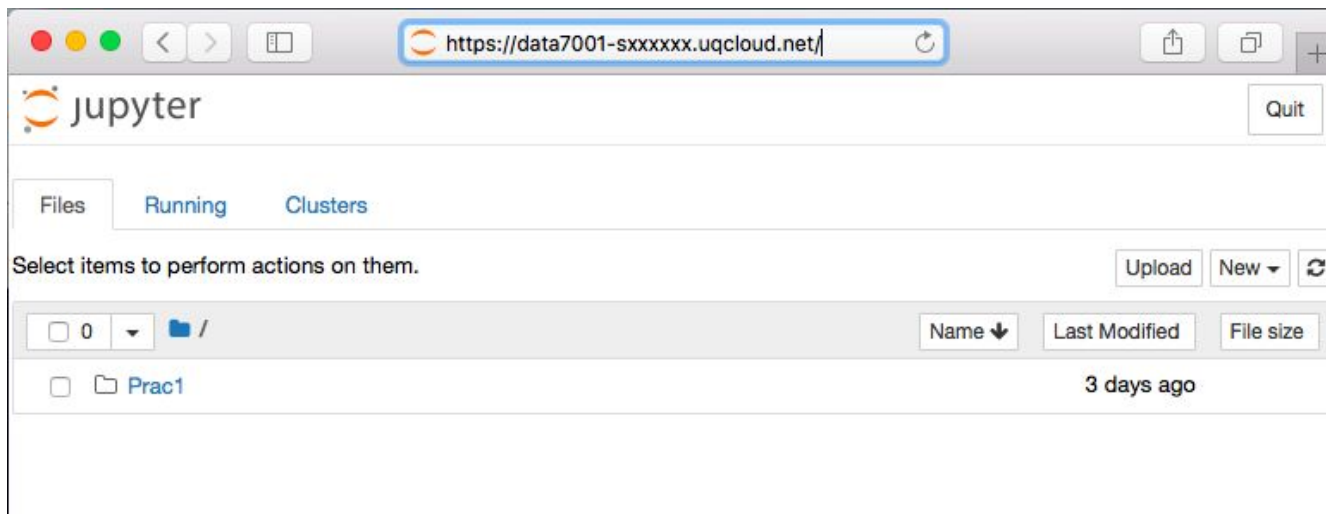


- Step 2. Press the ESC key to exit full screen mode for Zoom
- Step 3. Shrink and move Zoom window so that it takes up the left half of your computer screen
- Step 4. Open a browser, then shrink and move the browser so that it takes up the right half of your computer screen

**Congratulations for completing  
a large-scale distributed  
program!**

# Your Playground for DATA7001

- Your playground (or zone): <https://data7001-xxxxxxx.uqcloud.net/>
  - You should have the link at <https://coursemgr.uqcloud.net/data7001>
  - Open the above URL in a browser
  - You'll do a lot of programming in your zone for this course.

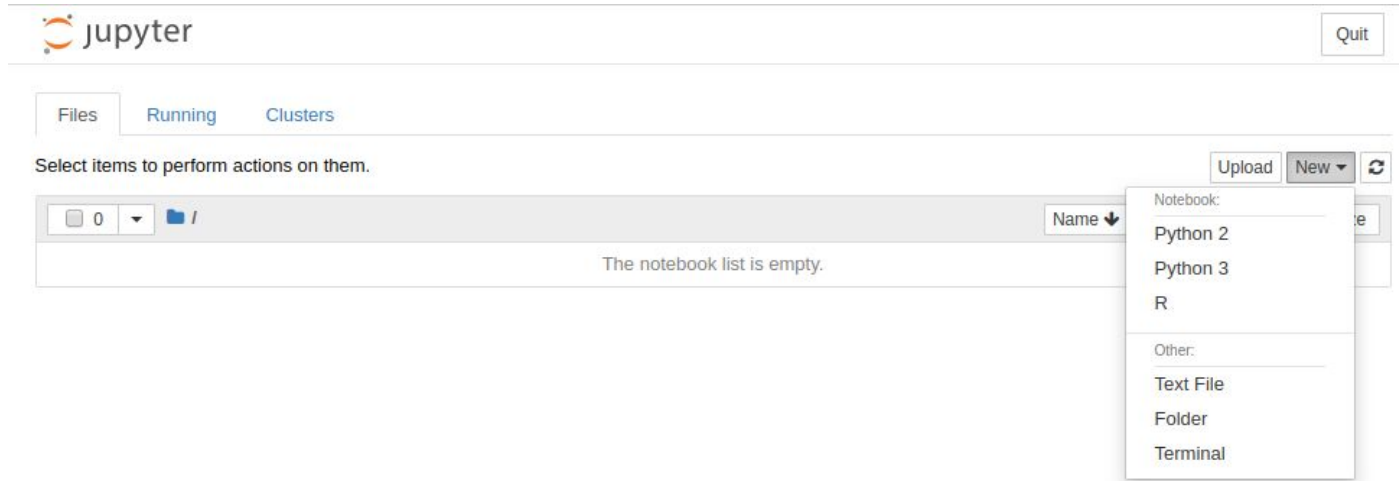


# Jupyter Notebook Basics

- They can be run on a browser - you can even access your notebooks on your smart phone!
- Jupyter Notebook has two components
  - Notebook documents
    - These are files containing both code and elements like figures, inks and equations.
  - Web application
    - This allows you to run your notebooks via a browser
    - It has two main components
      - Dashboard: the web interface for viewing and managing code and kernels
      - Kernels, which are programs for running and inspecting code
- Develop, execute and document code, and communicate results, all at one place.

# Demo

- Create a Jupyter Notebook to compute the sum of 1 to 10 in Python.

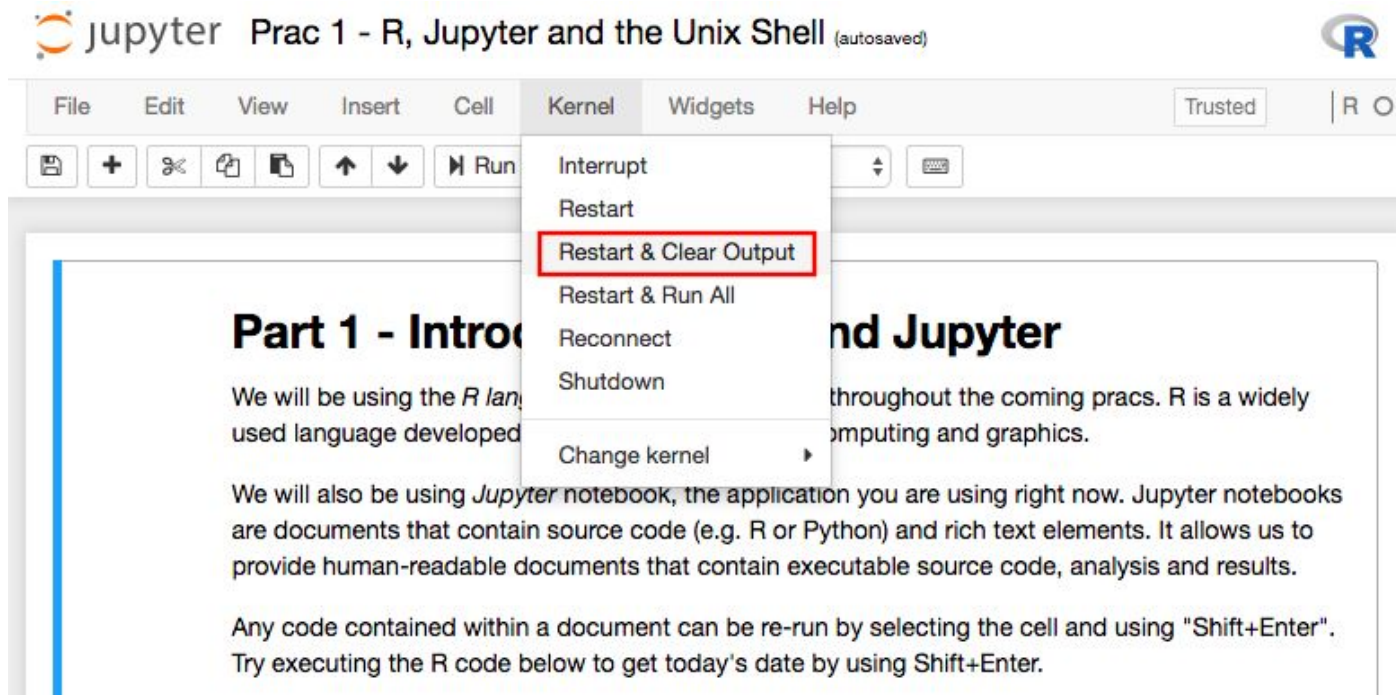


- Click Python3, then you will have a new Jupyter Notebook. Rename it to bootcamp.



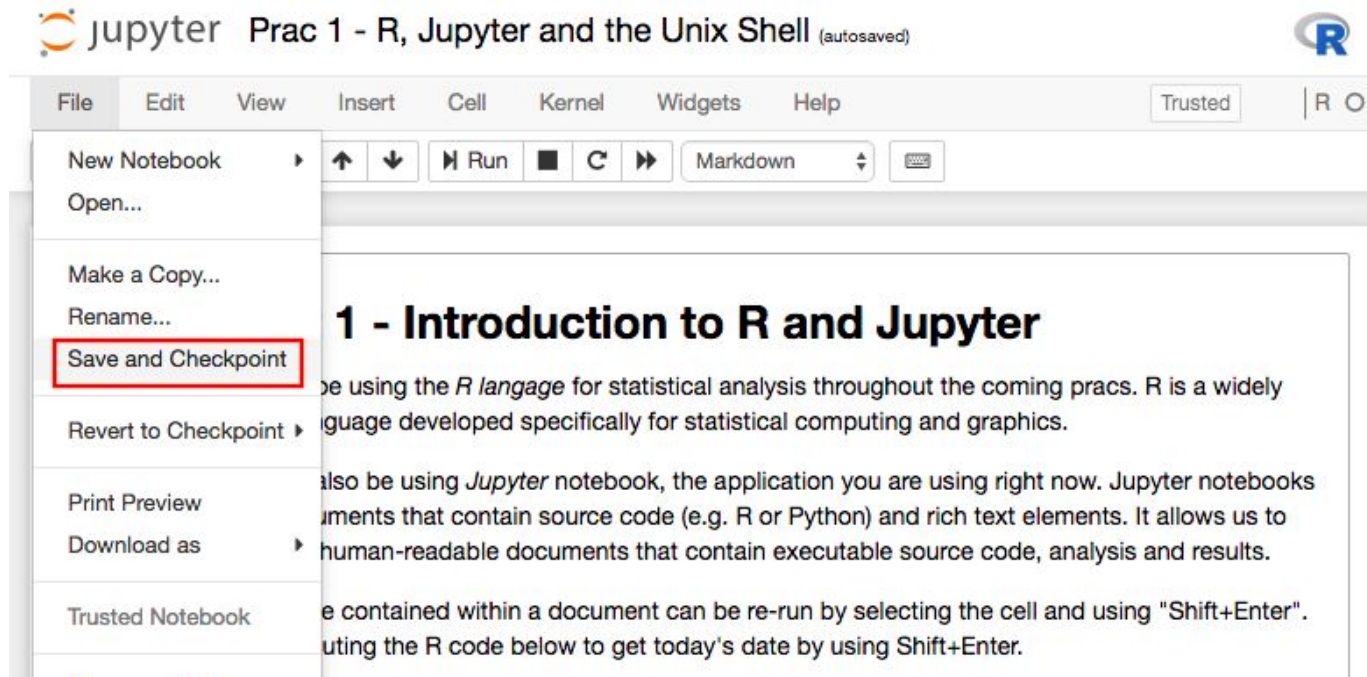
- Click the 2 links below to see gif images demonstrating basic operations
  - E.g. rename file, edit cell, add cell, remove cell
  - [http://community.datacamp.com.s3.amazonaws.com/community/production/ckeditor\\_assets/pictures/200/content\\_jupyternotebook3b.gif](http://community.datacamp.com.s3.amazonaws.com/community/production/ckeditor_assets/pictures/200/content_jupyternotebook3b.gif)
  - [http://community.datacamp.com.s3.amazonaws.com/community/production/ckeditor\\_assets/pictures/202/content\\_jupyternotebook7.gif](http://community.datacamp.com.s3.amazonaws.com/community/production/ckeditor_assets/pictures/202/content_jupyternotebook7.gif)
- These are taken from this tutorial:  
<https://www.datacamp.com/community/tutorials/tutorial-jupyter-notebook>

- If you want to restart the kernel and clear the outputs
  - E.g. after changing some code at the beginning of the notebook



# Important!

- Do save and checkpoint when you're done!
  - Your last checkpoint before deadline will be graded.



- Shutdown your kernel!
  - Otherwise computing resources may be quickly eaten up.



The image shows the Jupyter web interface. At the top left is the Jupyter logo. At the top right is a 'Quit' button. Below the logo are three tabs: 'Files', 'Running', and 'Clusters'. The 'Running' tab is selected. Below the tabs is a toolbar with buttons: 'Duplicate', 'Shutdown' (highlighted with a red box), 'View', 'Edit', and a trash icon. To the right of these buttons are 'Upload', 'New' (with a dropdown arrow), and a refresh icon. Below the toolbar is a table showing the current directory and its contents. The table has columns for 'Name', 'Last Modified', and 'File size'. The current directory is '/ Prac1'. The table lists three items: '..' (parent directory), 'img' (subdirectory), and 'Prac 1 - R, Jupyter and the Unix Shell.ipynb' (a file that is currently 'Running').

Name	Last Modified	File size
..	seconds ago	
img	2 years ago	
Prac 1 - R, Jupyter and the Unix Shell.ipynb	Running 2 minutes ago	19.1 kB

# Setting up Jupyter Notebook On Your Computer

- You may be eager to play with your notebooks locally on your computer.
- Follow the guide from the official website  
<https://jupyter.readthedocs.io/en/latest/install.html>
- You can install various extensions to make Jupyter Notebook more powerful
  - See <https://jupyter-contrib-nbextensions.readthedocs.io/en/latest/index.html>

- We will use R in the course as well.
- If you want to have R kernel
  - Install R first
  - Start R console and run the following two commands  
`install.packages('IRkernel')`  
`IRkernel::installspec()`
  - Read more at <https://irkernel.github.io/installation/>

# How to Learn More about Jupyter Notebook

- Read this excellent tutorial in detail  
<https://www.datacamp.com/community/tutorials/tutorial-jupyter-notebook>
- Practice
  - You'll use Jupyter Notebook for the practicals - this is not enough.
  - Use it for programming frequently
    - E.g. when you want to try out language features - write down your notes as well.
    - E.g. use it for your own little projects

# Checking Your Understanding

- The dashboard has three tabs: Files, Running, and Clusters. True or false?



The screenshot shows the Jupyter dashboard interface. At the top left is the Jupyter logo and the word "jupyter". At the top right is a "Quit" button. Below the logo, there are three tabs: "Files", "Running", and "Clusters". The "Running" tab is selected and highlighted with a red box. Below the tabs, there is a text prompt "Select items to perform actions on them." and three buttons: "Upload", "New", and a refresh icon. Below this is a table of running notebooks. The table has columns for "Name", "Last Modified", and "File size". The first row shows a folder icon and the text ".." with "seconds ago" in the last modified column. The second row shows a folder icon and the text "img" with "2 years ago" in the last modified column. The third row shows a notebook icon and the text "Prac 1 - R, Jupyter and the Unix Shell.ipynb" with "Running" in green text in the last modified column and "19.1 kB" in the file size column.

jupyter

Quit

Files Running Clusters

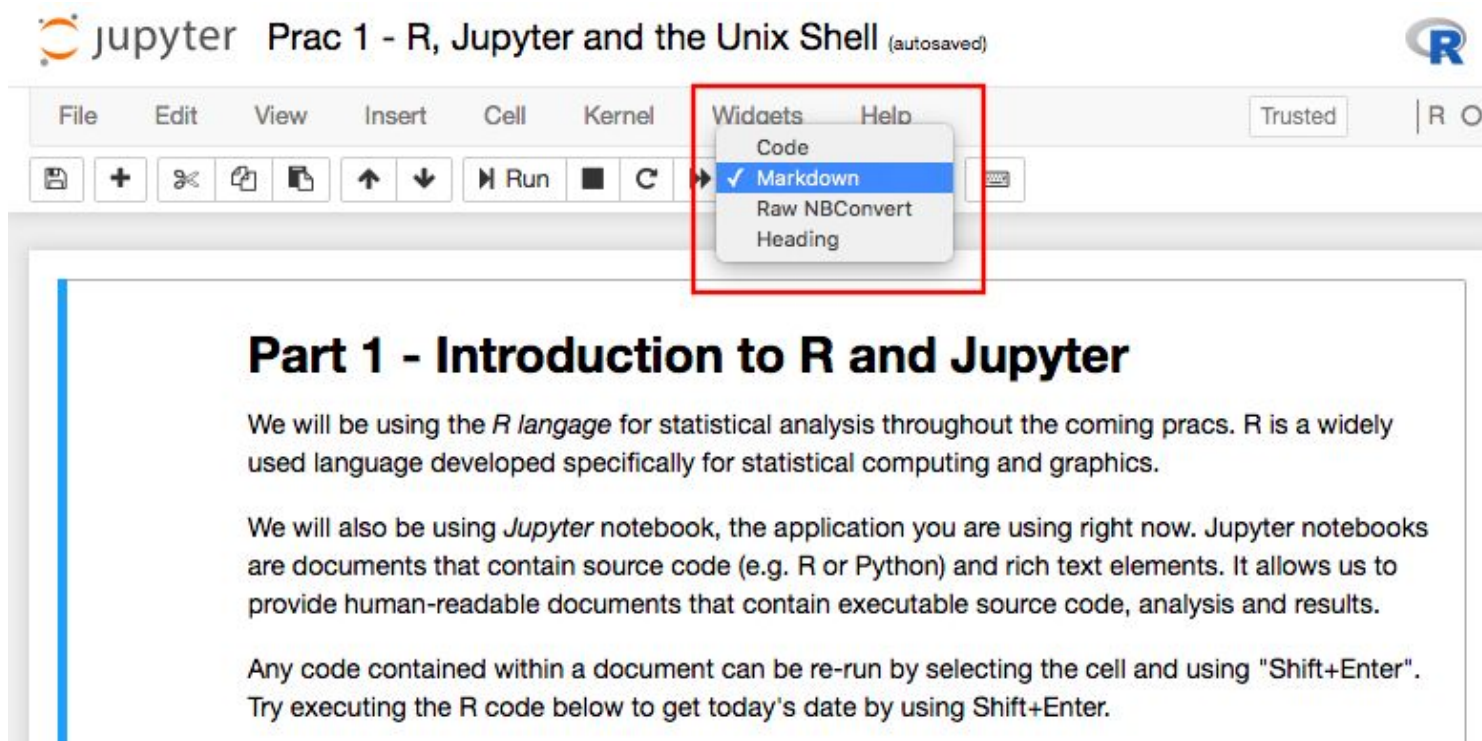
Select items to perform actions on them.

Upload New ↕ ↻

	Name ↓	Last Modified	File size
<input type="checkbox"/> 0	/ Prac1		
<input type="checkbox"/> ..		seconds ago	
<input type="checkbox"/> img		2 years ago	
<input type="checkbox"/> Prac 1 - R, Jupyter and the Unix Shell.ipynb		Running an hour ago	19.1 kB



- How many types of cells are in Jupyter Notebooks?



The screenshot shows the Jupyter Notebook interface. At the top, the title bar reads "jupyter Prac 1 - R, Jupyter and the Unix Shell (autosaved)". Below the title bar is a menu bar with "File", "Edit", "View", "Insert", "Cell", "Kernel", "Widgaets", and "Help". The "Widgaets" menu is open, showing options: "Code", "✓ Markdown", "Raw NBConvert", and "Heading". The "Markdown" option is highlighted with a blue background. Below the menu bar is a toolbar with icons for saving, adding, deleting, and running cells. The main content area displays a notebook with the title "Part 1 - Introduction to R and Jupyter". The text in the notebook describes the use of R for statistical analysis and Jupyter notebooks for combining code and text.

jupyter Prac 1 - R, Jupyter and the Unix Shell (autosaved)

File Edit View Insert Cell Kernel Widgaets Help

Trusted | R O

Code  
✓ Markdown  
Raw NBConvert  
Heading

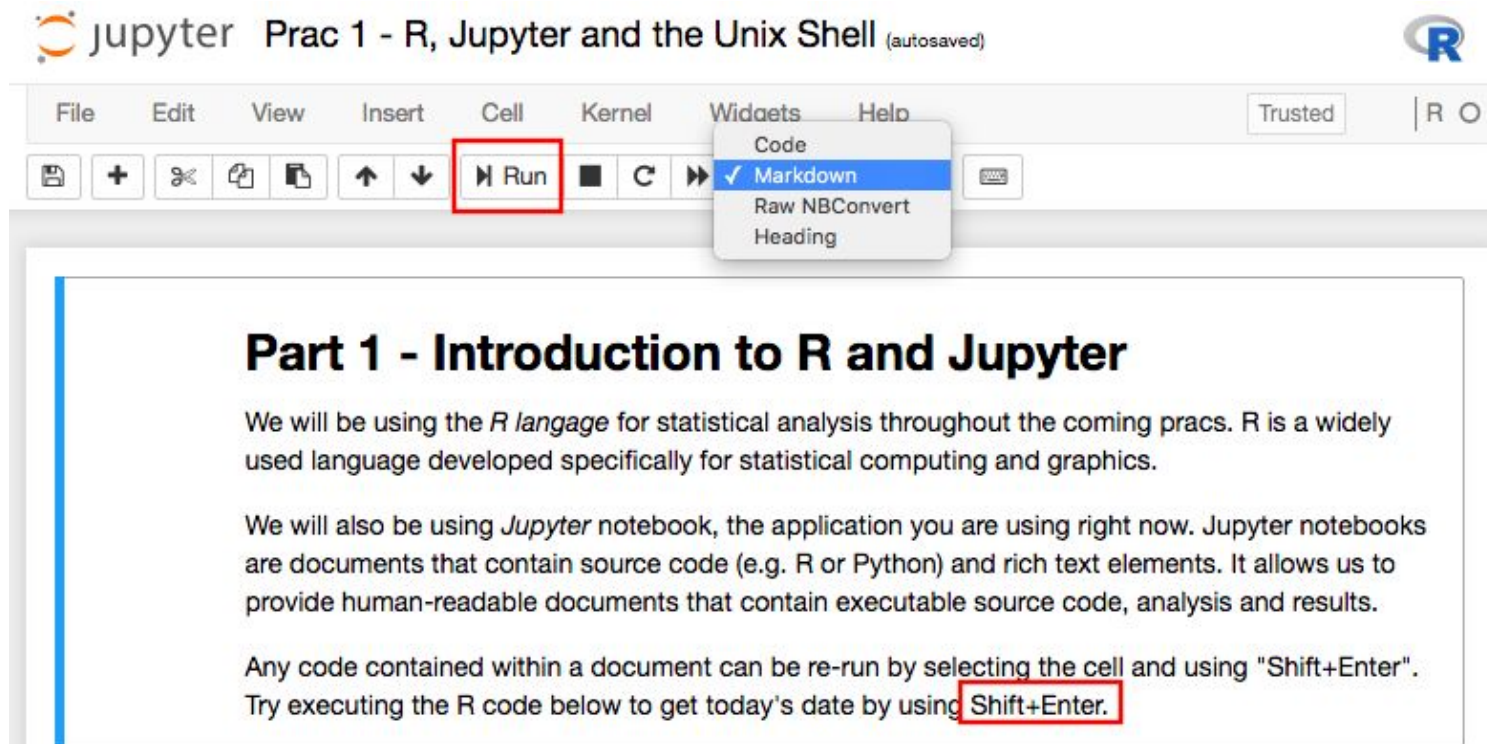
## Part 1 - Introduction to R and Jupyter

We will be using the *R language* for statistical analysis throughout the coming pracs. R is a widely used language developed specifically for statistical computing and graphics.

We will also be using *Jupyter* notebook, the application you are using right now. Jupyter notebooks are documents that contain source code (e.g. R or Python) and rich text elements. It allows us to provide human-readable documents that contain executable source code, analysis and results.

Any code contained within a document can be re-run by selecting the cell and using "Shift+Enter". Try executing the R code below to get today's date by using Shift+Enter.

- How do you run a cell?



The screenshot shows the Jupyter Notebook interface. The title bar reads "jupyter Prac 1 - R, Jupyter and the Unix Shell (autosaved)". The top menu bar includes "File", "Edit", "View", "Insert", "Cell", "Kernel", "Widawets", and "Help". The "Cell" menu is open, showing options: "Code", "Markdown" (selected), "Raw NBConvert", and "Heading". The "Run" button (a play icon) is highlighted with a red box. Below the menu bar, the notebook content is displayed in a light blue box. It contains a heading "Part 1 - Introduction to R and Jupyter" and three paragraphs of text. The last paragraph mentions "Shift+Enter" and has it highlighted with a red box.

**Part 1 - Introduction to R and Jupyter**

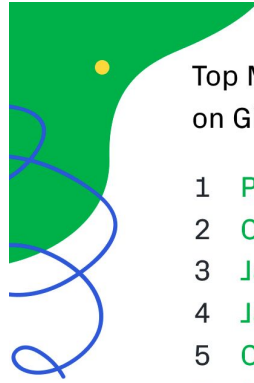
We will be using the *R language* for statistical analysis throughout the coming pracs. R is a widely used language developed specifically for statistical computing and graphics.

We will also be using *Jupyter* notebook, the application you are using right now. Jupyter notebooks are documents that contain source code (e.g. R or Python) and rich text elements. It allows us to provide human-readable documents that contain executable source code, analysis and results.

Any code contained within a document can be re-run by selecting the cell and using "Shift+Enter". Try executing the R code below to get today's date by using **Shift+Enter**.

# Python

- One of the most popular programming languages for machine learning and data science



## Top Machine Learning Languages on GitHub

- 1 Python
- 2 C++
- 3 JavaScript
- 4 Java
- 5 C#
- 6 Julia
- 7 Shell
- 8 R
- 9 TypeScript
- 10 Scala



- Python is general-purpose, and designed with an emphasis on readability.

## **The Zen of Python**

```
Beautiful is better than ugly.  
Explicit is better than implicit.  
Simple is better than complex.  
Complex is better than complicated.  
Flat is better than nested.  
Sparse is better than dense.  
Readability counts.
```

- Different languages have very different features, just like natural languages
- Sometimes the differences are huge
  - E.g. between procedural languages and declarative languages
- Still, there are a lot of features that are shared across many different languages
  - These are abstractions that people have found very useful
- We cover some of these features for Python, which are shared with languages like R, Java, C++, C

# Primitive Data Types

- Numbers

```
x = 2
y = 3.
print(x, type(x))
print(y, type(y))
print(x + y, x - y, x * y, x / y, x//y)
```

Can you guess the answers?

Try them out on your Jupyter Notebook to check!

- Booleans

```
is_wealthy = True
is_happy = True
is_wealthy & is_happy
```

- Strings

```
s1 = "hello"
s2 = "world"
print(s1 + ' ' + s2)
print('number of letters in ' + s1 + ": " + str(len(s1)))
print('number of letters in %s: %d' % (s1, len(s1)))
```

# Containers

- List: as the name suggests, this is used for storing a list of items

```
colors = ['red', 'green', 'blue']
print('We have', len(colors), 'colors:', colors)
colors.reverse()
print('Reversed list:', colors)
print('yellow appears ', colors.count('yellow'), 'times in the list')
print('green is at position', colors.index('green'))
del colors[1]
print('List after deleting item 1:', colors)
```

- Dictionaries: these are good when you want to quickly look up some information given a key

```
contacts = {'Taylor Swift': '0711112222', 'George Bush': '0711113333'}  
print('The complete contacts:', contacts)  
print('Contact names:', contacts.keys())  
print("Taylor Swift's phone number:", contacts['Taylor Swift'])  
print("Alex Taylor is a contact:", ('Alex Taylor' in contacts))
```



# Loops

- We often need to loop through elements in some container

```
for c in ['red', 'green', 'blue']:
    print(c)
```

```
for i, c in enumerate(['red', 'green', 'blue']):
    print(i, c)
```

```
colors = ['red', 'green', 'blue']
for i in range(len(colors)):
    print(colors[i])
```

- Try replacing a list by a dictionary. What values will be printed out?

# Conditionals

- Conditionals are used to control which statements are executed by testing some conditions

```
x, y = 1, 1
if x > 0 and y > 0:
    print('The point is in 1st quadrant')
elif x <= 0 and y > 0:
    print('The point is in 2nd quadrant')
elif x <= 0 and y <= 0:
    print('The point is in 3rd quadrant')
else:
    print('The point is in 4th quadrant')
```

# Functions

- Functions

```
def print_quadrant(x, y):  
    if x > 0 and y > 0:  
        print('The point is in 1st quadrant')  
    elif x <= 0 and y > 0:  
        print('The point is in 2nd quadrant')  
    elif x <= 0 and y <= 0:  
        print('The point is in 3rd quadrant')  
    else:  
        print('The point is in 4th quadrant')  
  
print_quadrant(1, 1)
```

# Checking Your Understanding

Which statement is correct?

- A. Python's basic data types include int, float, boolean, str
- B. Python supports list and dictionaries
- C. It is not possible to write conditional statements, loops and functions in Python

What is the `23//3` in Python?

- A. 2
- B. 6
- C. 7
- D. 8

# Unix Shell

- A Unix shell is a command-line interpreter or shell that provides a command line user interface for Unix-like operating systems.

```
demo$ ls
1.pdf  2.pdf  4.pdf  6.pdf  8.pdf  doc.txt
10.pdf 3.pdf  5.pdf  7.pdf  9.pdf
demo$ cat doc.txt
Hello World!
demo$ wc doc.txt
      1      2    13 doc.txt
```

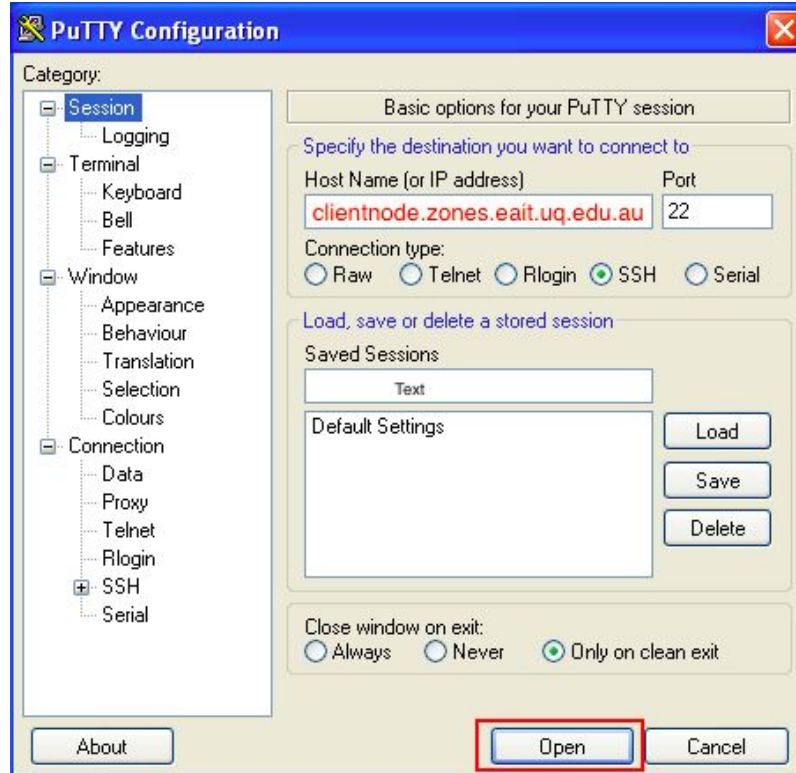
Example: list files in current folder and examine the content of a file

# ssh (Secure Shell)

- You have a Unix account associated with your zones.
  - Access from a Unix/Mac
    - Open a terminal
    - Inside UQ network (remember to replace sxxxxxxx by your student ID)
- ssh [xxxxxxx@clientnode.zones.eait.uq.edu.au](mailto:xxxxxxx@clientnode.zones.eait.uq.edu.au)

```
user@computer:~$ ssh sxxxxxxx@clientnode.zones.eait.uq.edu
.au
The authenticity of host 'clientnode.zones.eait.uq.edu.au
(172.23.82.200)' can't be established.
RSA key fingerprint is SHA256:6l/ZLGbfutq8DyLPEEZnh0DXV9DY
2Q59BSTNKGv6x8U.
Are you sure you want to continue connecting (yes/no)? █
```

- What if you have a Windows system -> use Putty



### Example

To connect to clientnode.zones.eait.uq.edu.au, enter the host name, click Open, and then input your user name (student ID starting with s) and password.

### Download Putty

<https://www.putty.org/>



# If You're Outside UQ's Network...

- If you are outside UQ's network, you can't directly login to your UNIX account.
- You need to do one of the following
  - Login to UQ's network using a VPN client
    - download and install the AnyConnect VPN client from <https://vpn.uq.edu.au>
    - connect to vpn.uq.edu.au using your UQ username and password
    - now try to connect to the server as in previous slide
  - Login to remote.labs.eait.uq.edu.au
    - follow the same procedure as connecting to clientnode.zones.eait.uq.edu.au
    - now you can run: ssh [sxxxxxxx@clientnode.zones.eait.uq.edu.au](#)

# Some Cool Utilities

- Batch renaming with rename (prepend all pdf files name with DATA7001-notes)

```
demo$ rename 's/^/DATA7001-notes/' *.pdf
demo$ ls
DATA7001-notes1.pdf  DATA7001-notes4.pdf  DATA7001-notes8.pdf
DATA7001-notes10.pdf DATA7001-notes5.pdf  DATA7001-notes9.pdf
DATA7001-notes2.pdf  DATA7001-notes6.pdf  doc.txt
DATA7001-notes3.pdf  DATA7001-notes7.pdf
```

- Search text file with grep

```
[demo$ grep 'Hello' *
doc.txt:Hello World!
```

# More on Python, R and Unix Shell

- Python

- Official Python tutorial  
<https://docs.python.org/3/tutorial/>
- Datacamp interactive course  
<https://www.datacamp.com/courses/intro-to-python-for-data-science>

- R

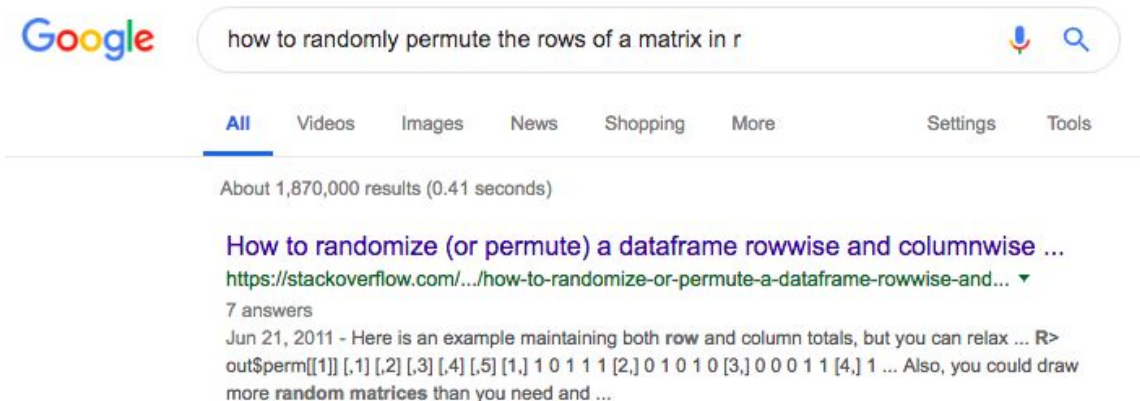
- Official introduction  
<https://cran.r-project.org/doc/manuals/r-release/R-intro.pdf>
- Datacamp interactive course  
<https://www.datacamp.com/courses/free-introduction-to-r>

- Unix shell

- <https://swcarpentry.github.io/shell-novice/>

# How to Learn Features of A Language/Library

- Skim through documentation or user guide to get started
  - You don't need to remember everything
  - This is mainly to get yourself familiar with what's officially available, so that you can look up the information you need later
- In reality, we often find it hard to search a documentation/user guide
  - Google, StackOverflow are your friends.



# Overview

- Getting started with your tools
  - Jupyter Notebook, Python, Linux Shell
- **Good coding practices**
- Debugging your code
- Reading others' code
- Programming for data science
  - NumPy, matplotlib, pandas
  - Data Sources
  - Web Scraping
- How to effectively ask us for help

# Example 1

- Task: calculate the sum and product of integers from 1 to 10.
- First attempt (let's code everything at once and run)

```
total = 0
for i in range(10):
    total = i
print('sum =', total)

product = 1
for i in range(10):
    product *= i
print('product =', product)
```

# Tip 1. Incremental Development

```
total = 0
for i in range(1,11):
    total += i
print('sum =', total)
```

```
product = 1
for i in range(1,11):
    product *= i
print('product =', product)
```

## Example 2

- Task: calculate the sum of  $1 + \dots + n$  for  $n=10, 100, 1000$
- Sample solution

```
total = 0
for i in range(1,11):
    total += i
print(total)

total = 0
for i in range(1,101):
    total += i
print(total)

total = 0
for i in range(1,1001):
    total += i
print(total)
```



## Tip 2. DRY (Don't Repeat Yourself)

- The DRY solution

```
def summation(n):  
    total = 0  
    for i in range(1,n+1):  
        total += i  
    return total  
  
print(summation(10))  
print(summation(100))  
print(summation(1000))
```

# Example 3

- Task: calculate the sum of integers from 1 to 10 million
- Sample solution

```
%time  
total = 0  
for i in range(1, int(1e7)+1):  
    total += i  
print(total)
```

50000005000000

CPU times: user 2.03 s, sys: 20 ms, total: 2.05 s

Wall time: 2.11 s

- You'll find it pretty slow if you have experience in C/C++ - in fact, Python loops are indeed very slow!

## Tip 3. Avoid Loops

- Loop-free solution

```
%time  
total = sum(range(1, int(1e7)+1))  
print(total)
```

```
50000005000000
```

```
CPU times: user 305 ms, sys: 2.2 ms, total: 308 ms
```

```
Wall time: 307 ms
```

- Python has high-level functions that allow us to process lists/arrays more efficiently than their Python equivalent with loops
  - Check these out (built-in functions, <https://docs.python.org/3/library/functions.html>):  
any(), all(), map(), sum()
  - We will see more when talking about Numpy later.

# More Good Practices

- Use meaningful names, though they may be long
- Write comments, though some code may seem trivial to you
  - If possible, ask someone to review your code
- Get your task done first, optimize later.
- Design test cases, and use them to test your program.

Keep learning, and avoid coding when you're tired or in a bad mood

- How to write good programs is what the field of software engineering is about.
- People have invented numerous design patterns, which are used as solution templates for common problems in programming.
- In a lot of cases, there are no clearly best winners
  - You make the call based on what's the most important for the task
- Programmers also write extensively about insights learned from their own experience
  - E.g. Alan Perlis' epigrams: <http://www.cs.yale.edu/homes/perlis-alan/quotes.html>

# Checking Your Understanding

Which is a good programming practice?

- A. Incremental development
- B. Don't Repeat Yourself
- C. Use tmp, instead of number\_of\_patients, as a variable name
- D. Design test cases to verify correctness of your code

# Overview

- Getting started with your tools
  - Jupyter Notebook, Python, Linux Shell
- Good coding practices
- **Debugging your code**
- Reading others' code
- Programming for data science
  - NumPy, matplotlib, pandas
  - Data Sources
  - Web Scraping
- How to effectively ask us for help

# Debugging Your Code

- There are two types of errors
  - Syntax errors
  - Logical errors
- Syntax errors are automatically reported when your program is run
  - The error messages indicates the types and locations of the errors
- Logical errors are much harder to detect
  - You need to trace out the errors by carefully reading your programs carefully and/or inspecting the intermediate results



# Example 1

- Read the following program and its execution results and answer the following questions
  - What is the error type and what does it mean?
  - Which line is causing the error?
  - How can this type of error be fixed?

```
1 x, y = 10, 2
2 if x + y > 10:
3 x = 1
```

```
File "<ipython-input-22-ee18a59f60c4>", line 4
  x = 1
  ^
```

```
IndentationError: expected an indented block
```

- Corrected code

```
x, y = 10, 2  
if x + y > 10:  
    x = 1
```

# Example 2

- Read the following program and its execution results and answer the following questions
  - What is the error type and what does it mean?
  - Which line is causing the error?
  - How can this type of error be fixed?

```
1 agentnum = 1
2 "Hello, Agent No. " + agentnum
```

---

```
TypeError                                Traceback (most recent call last)
<ipython-input-36-6bde4687e9fa> in <module>()
      1 agentnum = 1
----> 2 "Hello, Agent No. " + agentnum

TypeError: must be str, not int
```

- Corrected code

```
agentnum = 1  
"Hello, Agent No. " + str(agentnum)  
'Hello, Agent No. 1'
```

## Example 3

- Task: print a given list of strings of color names, with "red" inserted at the end.

```
def f(colors=[]):  
    colors.append('red')  
    print(colors)  
  
f(['green'])  
f()  
f()
```

Output

```
['green', 'red']  
['red']  
['red', 'red']
```

Can you see what's wrong?

- Corrected program

- It is not very intuitive, but sometimes languages features are subtle. That's why we need to test our code carefully.

```
def f(colors=None):  
    if colors is None:  
        colors = []  
    colors.append('red')  
    print(colors)
```

```
f(['green'])  
f()  
f()
```

```
['green', 'red']  
['red']  
['red']
```

- Logical errors are often difficult to detect for various reasons
  - A subtle language feature is interpreted differently by you and the computer.
  - You may have mistyped some statements which do not trigger any syntax error (e.g. + typed as -).
  - There may be logical errors in your ideas.
- Often, we trace the computation to identify where the errors occur.
  - Work out a few small test cases by hand
  - Insert statements to print intermediate results
    - You may want to learn how to use a debugger to inspect intermediate results without inserting print statements
  - Check whether they agree with your manual solutions

# How to Fix Errors

- Read the error message (if any)
  - What is the error type and what does it mean?
  - Which line is causing the error?
  - How can this type of error be fixed?
- Try your best to resolve it
  - Do some research using Google if you are not sure
  - In particular, you can often find good answers on StackOverflow
  - Not just because you're required to work out an assignment independently, but more importantly, it's the only way through which you can become good at programming
- Ask around when you run out of wit
  - Preferred way to ask questions: post them on Piazza



# Overview

- Getting started with your tools
  - Jupyter Notebook, Python, Linux Shell
- Good coding practices
- Debugging your code
- **Reading others' code**
- Programming for data science
  - NumPy, matplotlib, pandas
  - Data Sources
  - Web Scraping
- How to effectively ask us for help

# Reading Others' Code

- Just like English, if you read a lot, you are likely to write better.
- Some places to look at
  - <https://github.com/jakevdp/PythonDataScienceHandbook>
  - <https://github.com/donnemartin/data-science-ipython-notebooks>
  - <https://mdatascience-notebooks.uqcloud.net/view/>
- When you read, interact with the code
  - Run through the code in small chunks
  - For each chunk, form an idea of what it does and what the outputs will be
  - Compare the actual results with your anticipated results
  - Make changes (e.g. insert print statement, taking out an expression in a complex statements and run it)

# Overview

- Getting started with your tools
  - Jupyter Notebook, Python, Linux Shell
- Good coding practices
- Debugging your code
- Reading others' code
- **Programming for data science**
  - **(Python) The SciPy ecosystem: NumPy, matplotlib, pandas**
  - **(R) The tidyverse family**
  - **Data Sources**
  - **Web Scraping**
- How to effectively ask us for help

# Programming for Data Science

- The SciPy ecosystem
  - NumPy
    - Great tool for working with numerical arrays
  - Matplotlib
    - Great tool for producing high-quality visualization and plots
  - Pandas
    - Great tool for working with tabular data
- The tidyverse family
- Data sources
  - You need to have data to start your analysis.
- Web-scraping
  - There are a lot of useful but unstructured data on the web, you may want to use them for your project!

# More on the Programmer's Mindset

- An effective programmer need to develop a basic level of multilingualism and some expertise in key data science libraries
- Multilingualism
  - Various programming languages are used to develop data science solutions
  - Being able to read solutions written in different languages helps you to learn better
- Key data science libraries
  - Many libraries have been developed to greatly simplify (complex) routine tasks
  - You can greatly speeding up your coding process by standing on the shoulders of giants

# Bootcamp and Library Courses

- Multilingualism
  - Python intro (bootcamp)
  - R intro (library)
- Key libraries
  - Python: overview (bootcamp), pandas (library)
  - R: overview (bootcamp), dplyr (library)



Install



Getting started



Documentation



Report bugs



Blogs

SciPy (pronounced "Sigh Pie") is a Python-based ecosystem of open-source software for mathematics, science, and engineering. In particular, these are some of the core packages:



NumPy

Base N-dimensional array package



SciPy library

Fundamental library for scientific computing



Matplotlib

Comprehensive 2-D plotting



IPython

Enhanced interactive console



SymPy

Symbolic mathematics



pandas

Data structures & analysis

# NumPy for Numerical Arrays

- NumPy is a very handy library for data science
  - Excellent support for multidimensional numerical array
  - Many routines written in C/C++
    - Orders of magnitude faster than the Python equivalent with loops
  - Support for linear algebra operations



- Vectors

```
import numpy as np
a = np.arange(10)
print('a =', a)
b = np.arange(1, 10, 2)
print('b =', b)
c = np.linspace(0, 1, 6)
print('c =', c)
```

```
a = [0 1 2 3 4 5 6 7 8 9]
b = [1 3 5 7 9]
c = [0.  0.2 0.4 0.6 0.8 1. ]
```

- Arrays

```
a = np.ones((3, 3))
print('a =', a)
b = np.zeros((3, 3))
print('b =', b)
c = np.eye(3)
print('c =', c)
d = np.random.rand(3, 3)
print('d =', d)
e = np.array([[1, 2],
              [3, 4]])
print('e =', e)
```

```
a = [[1. 1. 1.]
      [1. 1. 1.]
      [1. 1. 1.]]
b = [[0. 0. 0.]
      [0. 0. 0.]
      [0. 0. 0.]]
c = [[1. 0. 0.]
      [0. 1. 0.]
      [0. 0. 1.]]
d = [[0.39615806 0.00316834 0.51771741]
      [0.09584409 0.64629246 0.65185526]
      [0.75239463 0.18897194 0.41071496]]
e = [[1 2]
      [3 4]]
```

- Indexing and slicing

```
a = np.arange(12).reshape(3, 4)
print('a =', a)
print('a[0,0] =', a[0,0])
print('a[0,:] =', a[0,:])
print('a[:,0] =', a[:,0])
print('a[1:3,:] =', a[1:3,:])
```

---

```
a = [[ 0  1  2  3]
      [ 4  5  6  7]
      [ 8  9 10 11]]
a[0,0] = 0
a[0,:] = [0 1 2 3]
a[:,0] = [0 4 8]
a[1:3,:] = [[ 4  5  6  7]
             [ 8  9 10 11]]
```

- Linear algebra

```
a = np.arange(12).reshape(3, 4)
v = np.arange(4)
print('a =', a)
print('v =', v)
print('np.dot(a, v) =', np.dot(a, v))
print('a @ v =', a @ v)
print('v + v =', v + v)
print('a + a =', a + a)
print('a + v =', a + v)
```

```
a = [[ 0  1  2  3]
      [ 4  5  6  7]
      [ 8  9 10 11]]
v = [0 1 2 3]
np.dot(a, v) = [14 38 62]
a @ v = [14 38 62]
v + v = [0 2 4 6]
a + a = [[ 0  2  4  6]
          [ 8 10 12 14]
          [16 18 20 22]]
a + v = [[ 0  2  4  6]
          [ 4  6  8 10]
          [ 8 10 12 14]]
```

- Matrix-vector product (with and without loops)

```
a = np.random.rand(1000, 1000)
v = np.random.rand(1000)
```

```
%%time
rslt = np.zeros(len(a))
for i in range(len(a)):
    for j in range(len(a[0])):
        rslt[i] += a[i,j] * v[j]
```

```
CPU times: user 1.14 s, sys: 0 ns, total: 1.14 s
Wall time: 1.14 s
```

```
%%time
rslt = a @ v
```

```
CPU times: user 57 µs, sys: 16.7 ms, total: 16.7 ms
Wall time: 1.99 ms
```

- Advanced linear algebra

```
a = np.array([[1,2], [3,4]])  
w, v = np.linalg.eig(a)  
print('matrix = ', a)  
print('eigenvalues =', w)  
print('eigenvectors =', v)  
print('inverse =', np.linalg.inv(a))  
print('determinant =', np.linalg.det(a))
```

```
matrix = [[1 2]  
          [3 4]]  
eigenvalues = [-0.37228132  5.37228132]  
eigenvectors = [[-0.82456484 -0.41597356]  
                [ 0.56576746 -0.90937671]]  
inverse = [[-2.   1.]  
           [ 1.5 -0.5]]  
determinant = -2.0000000000000004
```

- Avoid loops if possible

- Use simple statements expressed in terms of existing numpy functions instead

```
a = np.random.rand(1000, 1000)
v = np.random.rand(1000)
```

```
%%time
rslt = np.zeros(len(a))
for i in range(len(a)):
    for j in range(len(a[0])):
        rslt[i] += a[i,j] * v[j]
```

```
CPU times: user 1.14 s, sys: 0 ns, total: 1.14 s
Wall time: 1.14 s
```

```
%%time
rslt = a @ v
```

```
CPU times: user 57 µs, sys: 16.7 ms, total: 16.7 ms
Wall time: 1.99 ms
```

- Learning more about NumPy

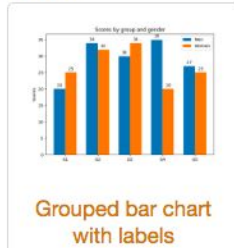
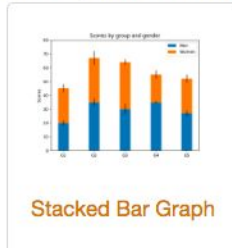
- Official website: <https://numpy.org/>
- User guide: <https://docs.scipy.org/doc/numpy/user/index.html>
- SciPy Lecture Notes (section 1.3): <https://scipy-lectures.org/>



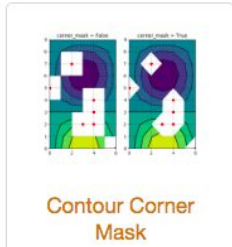
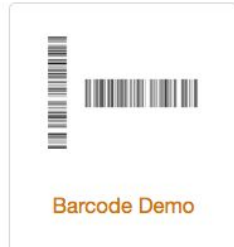
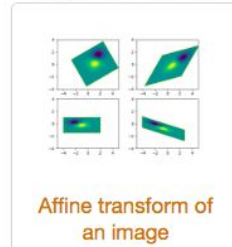
# Matplotlib for Visualization

- The example gallery gives you a sense of what matplotlib is capable of

## Lines, bars and markers



## Images, contours and fields

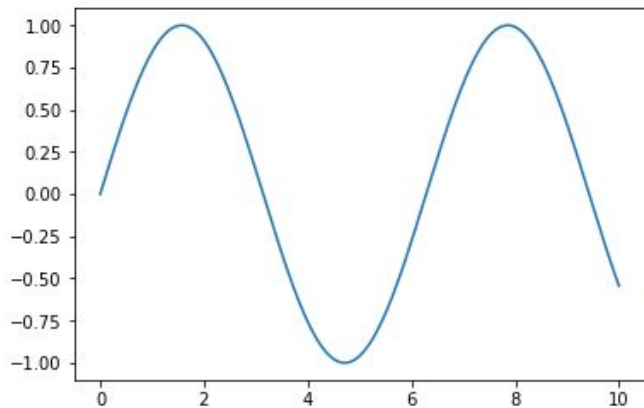


<https://matplotlib.org/gallery/index.html>

- Plot 2D curves

```
import numpy as np  
x = np.linspace(0, 10, 1000)  
y = np.sin(x)
```

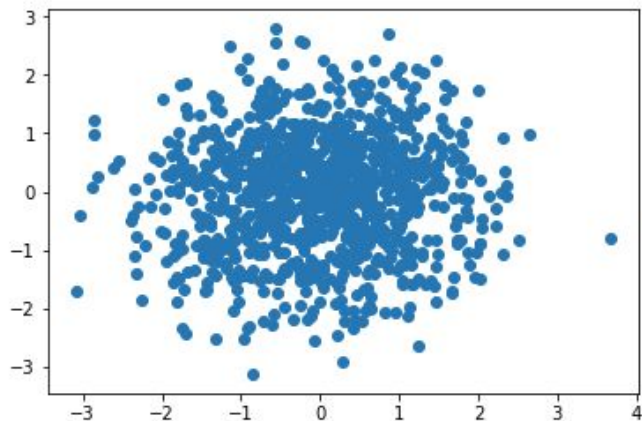
```
import matplotlib.pyplot as plt  
plt.plot(x, y)  
plt.show()
```



- 2D scatter plot

```
points = np.random.randn(2, 1000)  
plt.scatter(points[0], points[1])
```

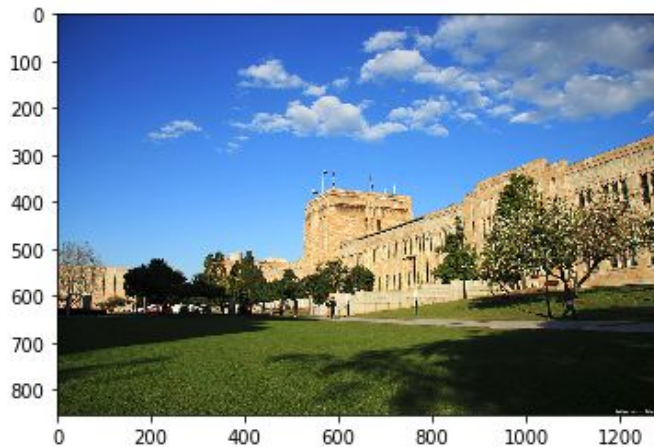
<matplotlib.collections.PathCollection at 0x7f10d7e786d8>



- Display an image

```
import imageio  
url = 'http://tinyurl.com/yxh6sngv'  
img = imageio.imread(url)  
plt.imshow(img)
```

<matplotlib.image.AxesImage at 0x7f110076e710>



- Learning more about Matplotlib

- Official website: <https://matplotlib.org/>
- Pyplot: <https://matplotlib.org/tutorials/introductory/pyplot.htm>
- Sample plots: [https://matplotlib.org/tutorials/introductory/sample\\_plots.html](https://matplotlib.org/tutorials/introductory/sample_plots.html)

# Pandas for Tabular Data

- Many datasets are stored in a tabular format (e.g. csv/Excel files)
- We often need to be able to
  - read in the data from csv/Excel
  - perform various operations on the tabular data, such as creation of new tabular data, selection of subsets
- Pandas is a great tool for working with tabular data
  - The name is derived from “panel data” (multidimensional data involving measurements over time)

- Read a csv file

- [https://stluc.manta.uqcloud.net/mdatascience/public/datasets/HumanResourceAnalytics/HR\\_comma\\_sep.csv](https://stluc.manta.uqcloud.net/mdatascience/public/datasets/HumanResourceAnalytics/HR_comma_sep.csv)

```
import pandas as pd
url = 'https://stluc.manta.uqcloud.net/mdatascience/public/datasets/HumanResourceAnalytics/HR_comma_sep.csv'
df = pd.read_csv(url)
df
```

	satisfaction_level	last_evaluation	number_project	average_monthly_hours	time_spend_company	Work_accident	left	promotion_last_5years	sales
0	0.38	0.53	2	157	3	0	1	0	sales
1	0.80	0.86	5	262	6	0	1	0	sales
2	0.11	0.88	7	272	4	0	1	0	sales
3	0.72	0.87	5	222	5	0	1	0	sales

- Print dataframe headers

```
df.columns
```

```
Index(['satisfaction_level', 'last_evaluation', 'number_project',  
      'average_monthly_hours', 'time_spend_company', 'Work_accident', 'left',  
      'promotion_last_5years', 'sales', 'salary'],  
      dtype='object')
```



- Selection

- first 5 entries in the satisfaction\_level column

```
df['satisfaction_level'][:5]
```

```
0    0.38
```

```
1    0.80
```

```
2    0.11
```

```
3    0.72
```

```
4    0.37
```

```
Name: satisfaction_level, dtype: float64
```

- Selection

- first 5 entries for both the satisfaction\_level and last\_evaluation columns

```
df[['satisfaction_level', 'last_evaluation']][:5]
```

	satisfaction_level	last_evaluation
0	0.38	0.53
1	0.80	0.86
2	0.11	0.88
3	0.72	0.87
4	0.37	0.52

- Selection

- High satisfaction level and low evaluation

```
subset = df[(df['satisfaction_level'] > 0.9) & (df['last_evaluation'] < 0.4)]  
subset[['satisfaction_level', 'last_evaluation']]
```

	satisfaction_level	last_evaluation
2819	0.96	0.37
2920	0.99	0.39
3449	0.99	0.37
5292	1.00	0.39
6329	0.95	0.37
6415	0.95	0.37
6641	0.98	0.38
6783	0.98	0.39
6884	0.95	0.38

- Learning more about pandas

- Official website: <https://pandas.pydata.org/>
- 10 min intro: [https://pandas.pydata.org/pandas-docs/stable/getting\\_started/10min.html](https://pandas.pydata.org/pandas-docs/stable/getting_started/10min.html)
- Cookbook: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/cookbook.html](https://pandas.pydata.org/pandas-docs/stable/user_guide/cookbook.html)



## R packages for data science

The tidyverse is an opinionated **collection of R packages** designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

Install the complete tidyverse with:

```
install.packages("tidyverse")
```

# Checking Your Understanding

Which statement is correct?

- A. A programmer should focus on learning only one programming language
- B. SciPy contains useful Python libraries for data scientists
- C. Tidyverse contains useful R libraries for data scientists
- D. pandas is good for processing tabular data

# Data Sources

- Data is what data scientists need to scientifically answer interesting questions.
- There are numerous publically available datasets available.
- Browse through some of them - they may give you some idea of what to work on for your project.

- Kaggle
  - <https://www.kaggle.com>
  - Numerous datasets, machine learning competitions
- Government datasets
  - <https://www.data.gov/>
  - <https://data.gov.au/>
  - <https://data.gov.sg/>
- UCI machine learning datasets
  - <https://archive.ics.uci.edu/ml/datasets.php>



- Others

- FiveThirtyEight (interactive sports and news site)
  - <https://data.fivethirtyeight.com/>
- BuzzFeedNews (America Internet media, news and entertainment company)
  - <https://github.com/BuzzFeedNews>
- Socrata (cleaned open source data sources ranging from government, business, and education data sets)
  - <https://opendata.socrata.com/>

# Web Scraping

- Sometimes, you may want to build your own dataset by using the vast amount of unstructured data on the web
  - News articles, food reviews, movie reviews, scientific articles,...
  - You can use these data to analyze what people are thinking/talking
- Web scraping is scraping data from the web
- Python provide some generic tools for you to collect data over the web
  - requests: useful for retrieving content from a URL
    - <https://pypi.org/project/requests/>
  - bs4: useful for extracting data from HTML files
    - <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>
- Some websites provide customized tools for users to extract data
  - E.g. Twitter, Facebook

- Retrieving a file on the web using requests

```
import requests
from IPython.display import display
BASE_URL = 'https://stluc.manta.uqcloud.net/mdatascience/public/datasets/NYC_Taxi/data/'
r = requests.get(BASE_URL + 'all_files.txt')
print(r.text)
```

```
fhv_tripdata_2015-01.csv
fhv_tripdata_2015-02.csv
fhv_tripdata_2015-03.csv
fhv_tripdata_2015-04.csv
fhv_tripdata_2015-05.csv
fhv_tripdata_2015-06.csv
fhv_tripdata_2015-07.csv
fhv_tripdata_2015-08.csv
fhv_tripdata_2015-09.csv
```

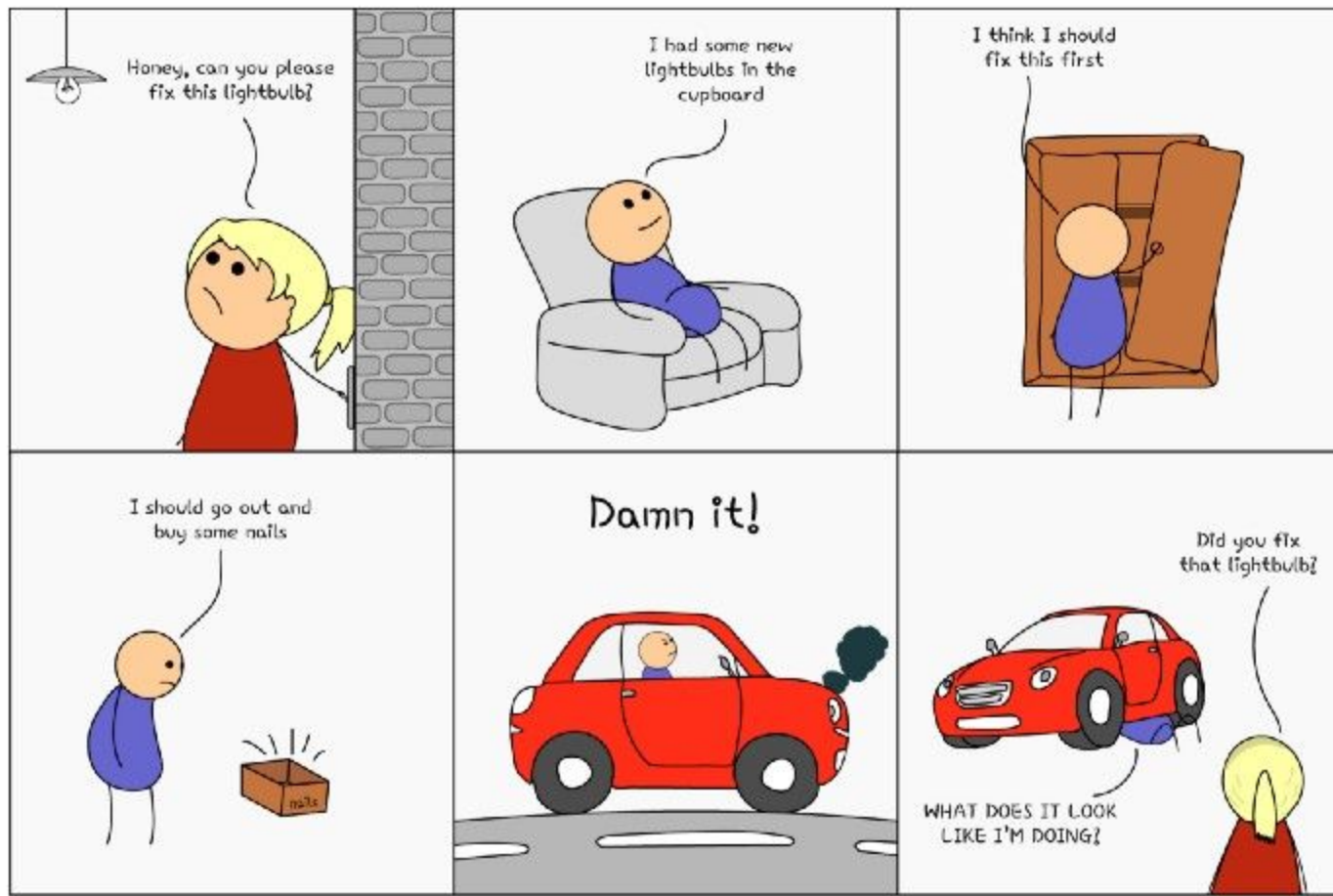
- Scraping some news titles from [www.news.com.au](https://www.news.com.au) using requests + bs4

```
from bs4 import BeautifulSoup
soup = BeautifulSoup(r.text, 'html.parser')
r = requests.get('https://www.news.com.au/')
soup = BeautifulSoup(r.text, 'html.parser')
for i, h in enumerate(soup('h4', class_='heading')):
    print('%2d.' % i, h.getText())
```

0. 'THEY'RE COMPLETELY WRONG': MasterChef trio eyes 'risky' next move
1. Residents forced out of Sydney building
2. Matt and Gary, you've made a mistake
3. Teen murder suspects 'trained in war'
4. Comedian rocked by string of deaths
5. Star-studded Hitler movie reveals trailer

# Overview

- Getting started with your tools
  - Jupyter Notebook, Python, Linux Shell
- Good coding practices
- Debugging your code
- Reading others' code
- Programming for data science
  - NumPy, matplotlib, pandas
  - Data Sources
  - Web Scraping
- **How to effectively ask us for help**



/techindustan



/techindustan



/techindustan

Do some investigation before asking us to fix any issue!

- Before asking questions
  - Rerun your code and make sure the error is reproducible
    - You may need to restart the kernel
  - Try to fix the errors yourself
    - See the section on Debugging Your Code
- What helps us to answer your question
  - Narrow down the scope
    - Sometimes the cause of error may be far away
    - Print out values of relevant variables
  - A minimal working example is extremely helpful
    - to yourself as well, as you can use that to remember the problem

- Can you see the difference between a specific question and a vague question?

- **Bad:** C# Math Confusion
- **Good:** Why does using float instead of int give me different results when all of my inputs are integers?
- **Bad:** [php] session doubt
- **Good:** How can I redirect users to different pages based on session data in PHP?
- **Bad:** android if else problems
- **Good:** Why does `str == "value"` evaluate to false when `str` is set to "value"?

<https://stackoverflow.com/help/how-to-ask>



# Summary

- Getting started with your tools
  - Jupyter Notebook, Python, Linux Shell
- Good coding practices
- Debugging your code
- Reading others' code
- Programming for data science
  - NumPy, matplotlib, pandas
  - Data Sources
  - Web Scraping
- How to effectively ask us for help