LECTURE 8
STRUCTURED DATA

Fundamentals of Programming - COMP1005

Semester 1, 2020

Department of Computing

Curtin University

Copyright Warning

Learning Outcomes

- Understand and implement structured data processing in Python using the pandas library
- Understand and critique the value of reproducible research
- Apply and create Python notebooks to support exploratory research

MORE DATA TYPES

Lecture 8

Data Types

- We've been working with:
  - Strings
  - Floats, integers, Booleans
  - Lists
  - Arrays
- We will now extend this by learning about:
  - Tuples
  - Sets
  - Dictionaries

Tuples

- We've used tuples already:
  - data = np.zeros((3, 3, 3))
  - **(3, 3, 3)** is a tuple
- Tuples are created as comma separated values, usually enclosed in brackets
- Tuples look a bit like lists, but they are immutable – structure cannot be changed
  - They can include mutable objects (e.g. lists) so, in those cases, their contents can be changed
- So, we can't append or del with Tuples
- They are ordered, so we can work with them as sequences.

Tuple example

```
tup1 = ('spam', 'eggs', 42)
tup2 = (1, 4, 9, 16, 25 )
tup3 = "yes", "oui", "ja", "si"

print(tup1)
print(tup2)
print(tup3)
```

```
('spam', 'eggs', 42)
(1, 4, 9, 16, 25)
('yes', 'oui', 'ja', 'si')
```


Tuples are sequences

```
tup1 = ('spam', 'eggs', 42)
tup2 = (1, 4, 9, 16, 25 )
tup3 = "yes", "oui", "ja", "si"

print(tup1[2])

print(tup2[1:-1])

for i in tup3:
   print(i)

print(tup1 + tup2)

print(tup1 * 2)

print(len(tup2))
```

```
42

(4, 9, 16)

yes
oui
ja
si

('spam', 'eggs', 42, 1, 4, 9, 16, 25)

('spam', 'eggs', 42, 'spam', 'eggs', 42)

5
```

Sets

- We've already used set operations to check if a letter is a vowel, e.g.

```
vowels = 'aeiou'
if letter in vowels:
  vowelcount += 1
```

- Sets are unordered (not a sequence), and there are no duplicates
- Sets are defined using {} ie.

```
vowels = {'a','e','i','o','u'}
```

- We can check if an item is **in** a set or **not in** it

```
if letter not in vowels:
```

https://www.learnpython.org/en/Sets

Set Theory – just here as background

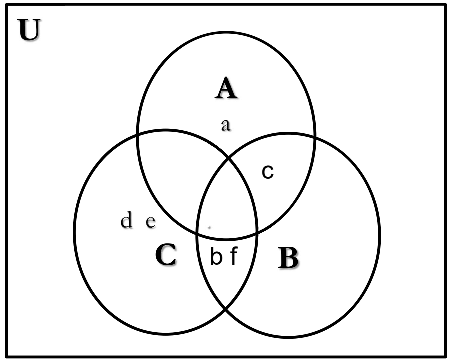
- A set is any collection of objects, e.g. a set of vertices
- The objects in a set are called the **elements** of the set
- Repetition and order are not important
  - {2, 3, 5} = {5, 2, 3} = {5, 2, 3, 2, 2, 3}
- Sets can be written in predicate form:
  - {1, 2, 3, 4} = {x : x is a positive integer less than 5}
  - Read the colon as "such that" such that, also {x|x is a….}
- The empty set is {} = $\emptyset$, all empty sets are equal

Set Theory based on http://www.maths.manchester.ac.uk/~avb/0n1_pdf/0N1_All.pdf

Set Theory – Operations on Sets

- Union
  - $A \cup B = \{ x : x \in A \text{ or } x \in B \}$
- Intersection
  - $A \cap B = \{ x : x \in A \text{ and } x \in B \}$
- Universal Set
  - All sets under consideration will be subsets of a background set, called the Universal Set, U
- Complement
  - $A' = \{ x : x \in U \text{ and } x \notin A \}$

Set Theory – Example



- Let:
  - U = {a, b, c, d, e, f}
  - A = {a, c}, B = {b, c, f}
    C = {b, d, e, f}.
- Then:
  - $B \cup C$ = {b, c, d, e, f}
  - $A \cap (B \cup C)$ = {c}
  - A' = {b, d, e, f}
    = C
  - $A' \cap (B \cup C) = C \cap (B \cup C)$ = {b, d, e, f} = C

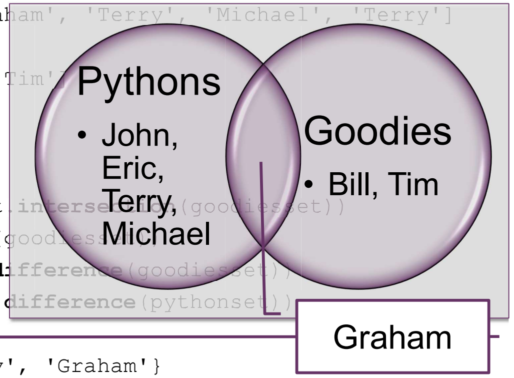
Set creation and operations

```
pythonlist = ['John', 'Eric', 'Graham', 'Terry', 'Michael', 'Terry']
pythonset = set(pythonlist)
goodieslist = ['Bill', 'Graham', 'Tim']
goodiesset = set(goodieslist)
print(pythonset)
print(goodiesset)
print('Intersection = ', pythonset.intersection(goodiesset))
print('Union = ', pythonset.union(goodiesset))
print('Difference = ', pythonset.difference(goodiesset))
print('Difference = ', goodiesset.difference(pythonset))
```

```
{'Eric', 'John', 'Michael', 'Terry', 'Graham'}
{'Tim', 'Bill', 'Graham'}
Intersection =  {'Graham'}
Union =  {'John', 'Michael', 'Tim', 'Bill', 'Eric', 'Terry', 'Graham'}
Difference =  {'Eric', 'John', 'Terry', 'Michael'}
Difference =  {'Bill', 'Tim'}
```


Set creation and operations

```
pythonlist = ['John', 'Eric', 'Graham', 'Terry', 'Michael', 'Terry']
pythonset = set(pythonlist)
goodieslist = ['Bill', 'Graham', 'Tim']
goodiesset = set(goodieslist)
print(pythonset)
print(goodiesset)
print('Intersection = ', pythonset.intersection(goodiesset))
print('Union = ', pythonset.union(goodiesset))
print('Difference = ', pythonset.difference(goodiesset))
print('Difference = ', goodiesset.difference(pythonset))
```


```
{'Eric', 'John', 'Michael', 'Terry', 'Graham'}
{'Tim', 'Bill', 'Graham'}
Intersection =  {'Graham'}
Union =  {'John', 'Michael', 'Tim', 'Bill', 'Eric', 'Terry', 'Graham'}
Difference =  {'Eric', 'John', 'Terry', 'Michael'}
Difference =  {'Bill', 'Tim'}
```


Dictionaries

- Dictionaries are mapping from a key, to a value
- Traditional dictionaries map words to meanings
- We might want to map towns to populations, years or months to total rainfall, student ID to student name
- The dictionary itself is not ordered - it is a set of key:value pairs
- Keys must be immutable
- We can add, **del**ete and overwrite

https://learnpythonthehardway.org/book/ex39.html

Dictionary – The Meaning of Liff

```
liff = {'Duddo': 'The most deformed potato in any
given collection of potatoes.',
        'Fring': 'The noise made by a lightbulb that
has just shone its last.',
        'Tonypandy': ' The voice used by presenters
on children\'s television programmes.'}
liff['Wawne'] = 'A badly supressed yawn.'
liff['Woking'] = 'Standing in the kitchen wondering
what you came in here for.'
print(liff)
print(liff['Duddo'])
print(liff['Fring'])
print(liff.keys())
del liff['Fring']
print(liff.keys())
```

The Meaning of Liff and
The Deeper Meaning of Liff,
by Douglas Adams and John Lloyd

http://liff.hivemind.net/

Dictionary – The Meaning of Liff

```
OUTPUT
{'Fring': 'The noise made by a lightbulb that has just
shone its last.', 'Wawne': 'A badly supressed yawn.',
'Duddo': 'The most deformed potato in any given collection
of potatoes.', 'Tonypandy': " The voice used by presenters
on children's television programmes.", 'Woking': 'Standing
in the kitchen wondering what you came in here for.'}

The most deformed potato in any given collection of
potatoes.

The noise made by a lightbulb that has just shone its
last.
dict_keys(['Woking', 'Fring', 'Wawne', 'Tonypandy',
'Duddo'])
dict_keys(['Woking', 'Wawne', 'Tonypandy', 'Duddo'])
```


Dictionary - Populations

```
pops = {'New South Wales': 7757843,
        'Victoria' : 6100877,
        'Queensland' : 4860448,
        'South Australia' : 1710804,
        'Western Australia' : 2623164,
        'Tasmania': 519783,
        'Northern Territory' : 245657,
        'Australian Capital Territory': 398349}
print(pops['Victoria'])
for p in pops:
   print(p)
for k in pops.keys():
   print(k, ' : ', pops[k])
```

Australian Demographic Statistics, Sep 2016
http://www.abs.gov.au/ausstats/abs@.nsf/mf/3101.0

Dictionary – Populations (output)

```
6100877

Tasmania
Western Australia
Victoria
Queensland
South Australia
Northern Territory
Australian Capital Territory
New South Wales

Tasmania  :  519783
Western Australia  :  2623164
Victoria  :  6100877
Queensland  :  4860448
South Australia  :  1710804
Northern Territory  :  245657
Australian Capital Territory  :  398349
New South Wales  :  7757843
```

```
print(pops['Victoria'])
```

```
for p in pops:
   print(p)
```

```
for k in pops.keys():
   print(k, ' : ', pops[k])
```


Dictionaries

- We can list the keys, or the values, or both...

```
for p in pops:
  print(p)
```

Tasmania
Western Australia
…
Australian Capital Territory
New South Wales

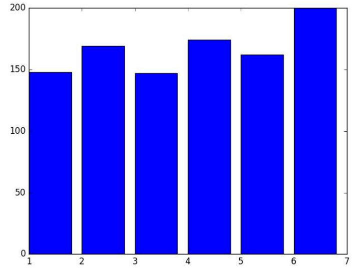
```
for k in pops.keys():
  print(pops[k])
```

519783
2623164 ← **CLUE FOR PRAC QUESTION**
…
7757843

```
for k in pops.keys():
  print(k, ': ', pops[k])
```

Tasmania: 519783
Western Australia: 2623164
…
New South Wales: 7757843

Dice toss with maps



- 1000 random dice tosses and plot the results:

```
import random
import matplotlib.pyplot as plt

dicecount = {1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0}

for i in range(1000):
    toss = random.randint(1,6)
    dicecount[toss] += 1

plt.bar(dicecount.keys(), dicecount.values())
plt.show()
```


And something more challenging…

- Find the frequency of each of the words in a text…

```
import sys

punctuation = '~!@#$%^&*()_+{}|:"<>?`=[]\\;\',./'

if len(sys.argv) <2 :
  filename = 'grimm.txt'
else:
  filename = sys.argv[1]

book = open(filename).read()
bookP = book.translate(str.maketrans('','',punctuation))
words = bookP.lower().split()
print(len(words))
print(words[:10])
```

1139
['rumpelstiltskin', 'by', 'the', 'side', 'of', 'a', 'wood', 'in', 'a', 'country']

And something more challenging…

- Then calculate frequencies using a dictionary…

```
wordfreq = {}                  # empty dictionary
for word in words:             # for each word
   if word not in wordfreq:    # if it's not in dict
      wordfreq[word] = 0       # create a key/val pair
   wordfreq[word] += 1         # increment count[word]

print(len(wordfreq))           # 390 unique, 1139 total
print(wordfreq)
```

- There are many alternative packages with extensive support for analysing text (e.g. nltk) – but this is a good starting point

STRUCTURED DATA

Fundamentals of Programming

Lecture 8

Structured data

- When we downloaded the weather data, we ignored the header lines… ...but they gave more information about the structure of the data
- We can go beyond lists and arrays for storing data
- Data frames look after the structure of the data – column labels, grouping, operations
- We will use pandas to create data frames

Python Data Analysis Library

- The Pandas library:
  - provides data structures
  - produces high-quality plots with matplotlib
  - integrates nicely with other libraries that use NumPy
- To use pandas, we start with an import:

```
import pandas as pd
```


Dataframes

- A DataFrame is a 2-dimensional data structure that can store data of different types in columns
  - including characters, integers, floating point values, factors and more...
- It is similar to a spreadsheet or an SQL table or the data.frame in R
- A DataFrame always has an index (0-based)
- An index refers to the position of an element in the data structure
- The index values can be overridden, but can cause problems – don't do it!

http://www.datacarpentry.org/python-ecology-lesson/01-starting-with-data/

Building a dataframe

- Dataframes can be defined as dictionaries, keys are labels, values are lists

```
import pandas as pd
df = pd.DataFrame({'AAA' : [4, 5, 6, 7],
   'BBB' : [10,20,30,40], 'CCC': [100,50,-30,-50]})
print(df)
print(df.describe())
```

	AAA	BBB	CCC
0	4	10	100
1	5	20	50
2	6	30	-30
3	7	40	-50

	AAA	BBB	CCC
count	4.000000	4.000000	4.000000
mean	5.500000	25.000000	17.500000
std	1.290994	12.909944	69.940451
min	4.000000	10.000000	-50.000000
25%	4.750000	17.500000	-35.000000
50%	5.500000	25.000000	10.000000
75%	6.250000	32.500000	62.500000
max	7.000000	40.000000	100.000000

Reading in a CSV file

- surveys_df = pd.read_csv("surveys.csv")
- No need for splitting etc – all done for you!

```
record_id  month  day  year  plot_id species_id sex  hindfoot_length  weight
0          1      7   16  1977        2         NL   M               32  NaN
1          2      7   16  1977        3         NL   M               33  NaN
2          3      7   16  1977        2         DM   F               37  NaN
3          4      7   16  1977        7         DM   M               36  NaN
4          5      7   16  1977        3         DM   M               35  NaN
...
35544      35545     12   31  2002       15     AH  NaN             NaN  NaN
35545      35546     12   31  2002       15     AH  NaN             NaN  NaN
35546      35547     12   31  2002       10     RM    F              15   14
35547      35548     12   31  2002        7     DO    M              36   51
35548      35549     12   31  2002        5    NaN  NaN             NaN  NaN

[35549 rows x 9 columns]
```


Types within dataframes

type(surveys_df)
- returns:

<class 'pandas.core.frame.DataFrame'>

surveys_df.dtypes
- returns:

```
record_id        int64
month            int64
day              int64
year             int64
plot_id          int64
species_id       object
sex              object
hindfoot_length  float64
weight           float64
dtype: object
```


Describing dataframes

surveys_df.columns

Index(['record_id', 'month', 'day', 'year', 'plot_id', 'species_id', 'sex','hindfoot_length', 'weight'],
 dtype='object')

surveys_df.shape

(35549, 9)

surveys_df.head()

```
 record_id  month  day   year  plot_id species_id sex  hindfoot_length  \
0       1     7   16  1977     2       NL   M          32.0
1       2     7   16  1977     3       NL   M          33.0
2       3     7   16  1977     2       DM   F          37.0
3       4     7   16  1977     7       DM   M          36.0
4       5     7   16  1977     3       DM   M          35.0

   weight
0    NaN
1    NaN
2    NaN
3    NaN
4    NaN
```

surveys_df.head(15)
surveys_df.tail()

Calculating statistics

surveys_df.columns.values

array(['record_id', 'month', 'day', 'year', 'plot_id', 'species_id', 'sex', 'hindfoot_length', 'weight'], dtype=object)

- The pd.unique function tells us all of the unique values in the species_id column.

pd.unique(surveys_df['species_id'])

array(['NL', 'DM', 'PF', 'PE', 'DS', 'PP', 'SH', 'OT', 'DO', 'OX', 'SS', 'OL', 'RM', nan, 'SA', 'PM', 'AH', 'DX', 'AB', 'CB', 'CM', 'CQ', 'RF', 'PC', 'PG', 'PH', 'PU', 'CV', 'UR', 'UP', 'ZL', 'UL', 'CS', 'SC', 'BA', 'SF', 'RO', 'AS', 'SO', 'PI', 'ST', 'CU', 'SU', 'RX', 'PB', 'PL', 'PX', 'CT', 'US'], dtype=object)

Summary statistics

- **surveys_df['weight'].describe()**

count 32283.000000
mean 42.672428
std 36.631259
min 4.000000
25% 20.000000
50% 37.000000
75% 48.000000
max 280.000000
Name: weight, dtype: float64

The **pandas function describe** will return descriptive stats including: mean, median, max, min, std and count for a column in the data (if it has numeric data)

- We can also extract one specific metric using...

surveys_df['weight'].**min()**
surveys_df['weight'].**max()**
surveys_df['weight'].**mean()**
surveys_df['weight'].**std()**
surveys_df['weight'].**count()**

Grouping data

- # Group data by sex

sorted_data = surveys_df.groupby('sex')

- # summary statistics for all numeric columns by sex
- **sorted_data.describe()**
- # provide the mean for each numeric column by sex
- **sorted_data.mean()**

```
 record_id   month   day   year   plot_id \
sex
F 18036.412046 6.583047 16.007138 1990.644997 11.440854
M 17754.835601 6.392668 16.184286 1990.480401 11.098282
 hindfoot_length weight
sex
F 28.836780 42.170555
M 29.709578 42.995379
```


Counting and Maths functions

- We can count the number of samples by species

species_counts = surveys_df.groupby('species_id')\
['record_id'].count()
print(species_counts)

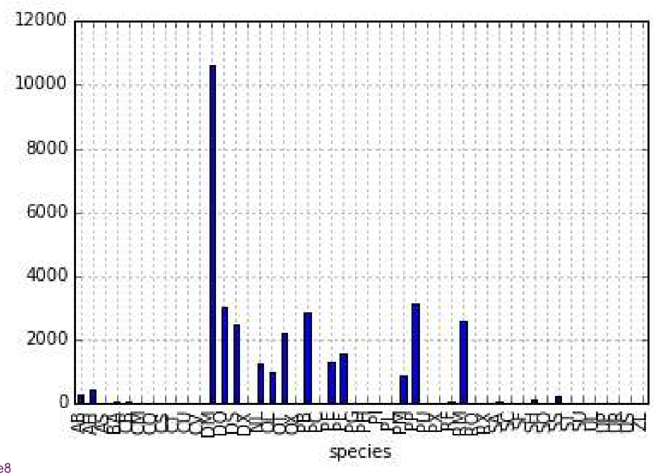
- Or, we can count just the rows that have the species "DO"

surveys_df.groupby('species_id')['record_id'].count()['DO']

- Basic Maths Functions
  - We can perform maths functions on entire columns of our data
  - # multiply all weight values by 2.2 surveys_df['weight']*2.2

Simple plotting

species_counts.plot(kind='bar')



Slicing

select rows 0, 1, 2 (row 3 is not selected)
surveys_df[0:3]
select the first 5 rows (rows 0, 1, 2, 3, 4)
surveys_df[:5]
select the last element in the list
surveys_df[-1]

- **loc** indexes by labels, **iloc** indexes by position (integers)

iloc[row slicing, column slicing]
surveys_df.iloc[0:3, 1:4]
select all columns for rows of index values 0 and 10 surveys_df.loc[[0, 10], :]
select three columns for row 0
surveys_df.loc[0, ['species_id', 'plot_id', 'weight']]
All columns for rows, 0, 10 and 35549
surveys_df.loc[[0, 10, 35549], :]

Subsetting data using criteria

surveys_df[surveys_df.year == 2002]
record_id month day year plot_id species_id sex hindfoot_length weight
33320 33321 1 12 2002 1 DM M 38 44
33321 33322 1 12 2002 1 DO M 37 58
33322 33323 1 12 2002 1 PB M 28 45
...
35546 35547 12 31 2002 10 RM F 15 14
35547 35548 12 31 2002 7 DO M 36 51
35548 35549 12 31 2002 5 NaN NaN NaN NaN
[2229 rows x 9 columns]

- Or we can select all rows that do not contain the year 2002:

surveys_df[surveys_df.year **!= 2002**]

- We can define sets of criteria too:

surveys_df[(surveys_df.year **>= 1980**) & (surveys_df.year **<=1985**)]

Copying dataframes

- As with other objects we've worked with (arrays, lists…) assigning an object to a variable just points them to the same place.
- Need to use **copy()** to make a separate object.
- **Copy** uses the dataframe's copy() method

true_copy_surveys_df = surveys_df.copy()

- A **Reference** is created using the = operator

ref_surveys_df = surveys_df

- Slices and views of a dataframe are using a reference to the original data – any changes will change the original

That's it for now…

- We'll do more with dataframes in the practicals
- There is a lot more they can do…

- Think about how this might affect how we work with datasets we have used already – do they have column names that we could work with... and csv files can be read in using pandas...

REPRODUCIBLE RESEARCH

Fundamentals of Programming

Lecture 8

Reproducible Research

- A key value of scientific research is that it is reproducible
- When we share or publish our research, others should be able to reproduce our results
- Many journals now ask for data and code along with submitted papers
- Reviewers can then check the data and code to verify the results

About Project Jupyter

- Project Jupyter is an open source project was born out of the IPython Project in 2014
- The project aims to support interactive data science and scientific computing across all programming languages
- Jupyter is 100% open source software, free for all to use and released under the liberal terms of the modified BSD license
- Dynamic developers, cutting edge scientists as well as everyday users work together to further Jupyter's best-in-class tools.

http://jupyter.org/about.html

Jupyter notebooks

- Notebook documents (or “notebooks”, all lower case) are documents which contain both computer code (e.g. python) and rich text elements (paragraph, equations, figures, links, etc...).
- Notebook documents are both
  - human-readable documents containing the analysis description and the results (figures, tables, etc..)
  - executable documents which can be run to perform data analysis

http://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html

Jupyter notebook app

- The *Jupyter Notebook App* is a server-client application that allows editing and running notebook documents via a web browser
- The *Jupyter Notebook App* can be executed on a local desktop requiring no internet access or can be installed on a remote server and accessed through the internet
- In addition to displaying/editing/running notebook documents, the *Jupyter Notebook App* has a **dashboard**
  - a "control panel" showing local files and allowing to open notebook documents or shutting down their kernels.

Notebook kernels

- A notebook *kernel* is a "computational engine" that executes the code contained in a Notebook document.
- The *ipython kernel* executes python code
- Kernels for many other languages exist
- When you open a Notebook document, the associated *kernel* is automatically launched
- When the notebook is *executed* (either cell-by-cell or with menu *Cell -> Run All*), the *kernel* performs the computation and produces the results
- We could work on Python 3.6 or 2.7 (etc) kernels

Notebook dashboard

- The *Notebook Dashboard* is the component which is shown first when you launch Jupyter Notebook App
- The *Notebook Dashboard* is mainly used to open notebook documents, and to manage the running kernels (visualize and shutdown)
- The *Notebook Dashboard* has other features similar to a file manager, namely navigating folders and renaming/deleting files

Running Jupyter

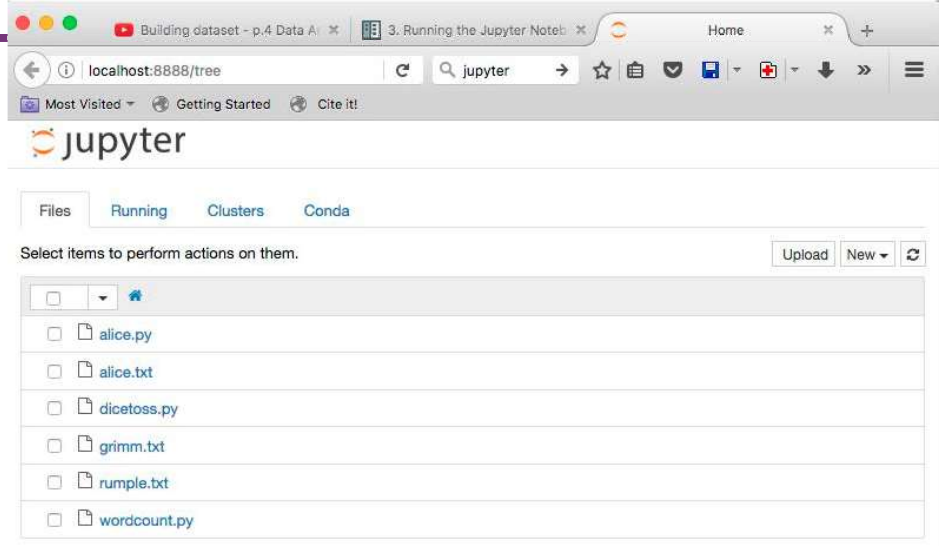
- Type **jupyter notebook** at the command line in the directory with your notebooks

```
M-A0009607-S:Lecture8 $ jupyter notebook
[I 13:01:47.692 NotebookApp] [nb_conda_kernels] enabled, 2 kernels found
[I 13:01:49.571 NotebookApp] ✓ nbpresent HTML export ENABLED
[W 13:01:49.572 NotebookApp] ✗ nbpresent PDF export DISABLED: No
module named 'nbbrowserpdf'
[I 13:01:49.619 NotebookApp] [nb_conda] enabled
[I 13:01:49.832 NotebookApp] [nb_anacondacloud] enabled
[I 13:01:49.865 NotebookApp] Serving notebooks from local directory:
/Users/username/Fundamentals/Lecture8
[I 13:01:49.865 NotebookApp] 0 active kernels
[I 13:01:49.865 NotebookApp] The Jupyter Notebook is running at:
http://localhost:8888/
[I 13:01:49.865 NotebookApp] Use Control-C to stop this server and shut down
all kernels (twice to skip confirmation).
```

Shutting down jupyter

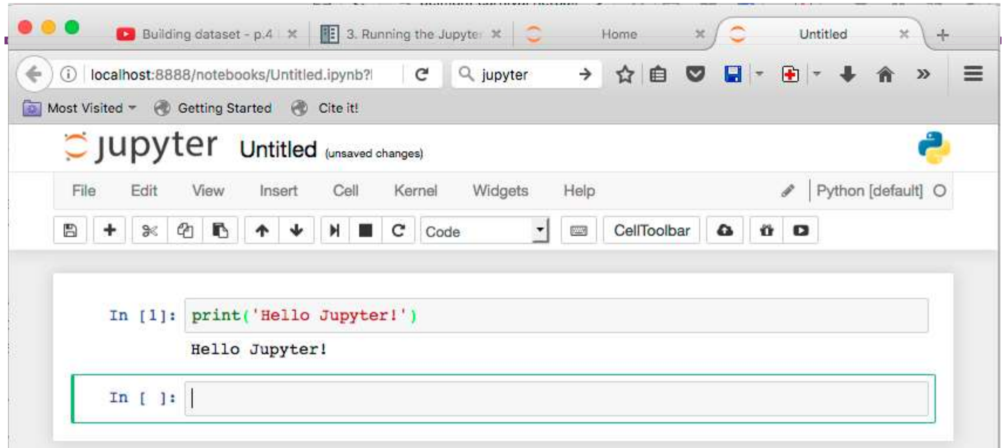
- Type control-C in the terminal window you ran jupyter from…

```
^C[I 13:05:23.645 NotebookApp] interrupted
Serving notebooks from local directory:
/Users/username/Fundamentals/Lecture8
0 active kernels
The Jupyter Notebook is running at:
http://localhost:8888/
Shutdown this notebook server (y/[n])? y
[C 13:05:26.181 NotebookApp] Shutdown confirmed
[I 13:05:26.181 NotebookApp] Shutting down kernels
M-A0009607-S:Lecture8 $
```

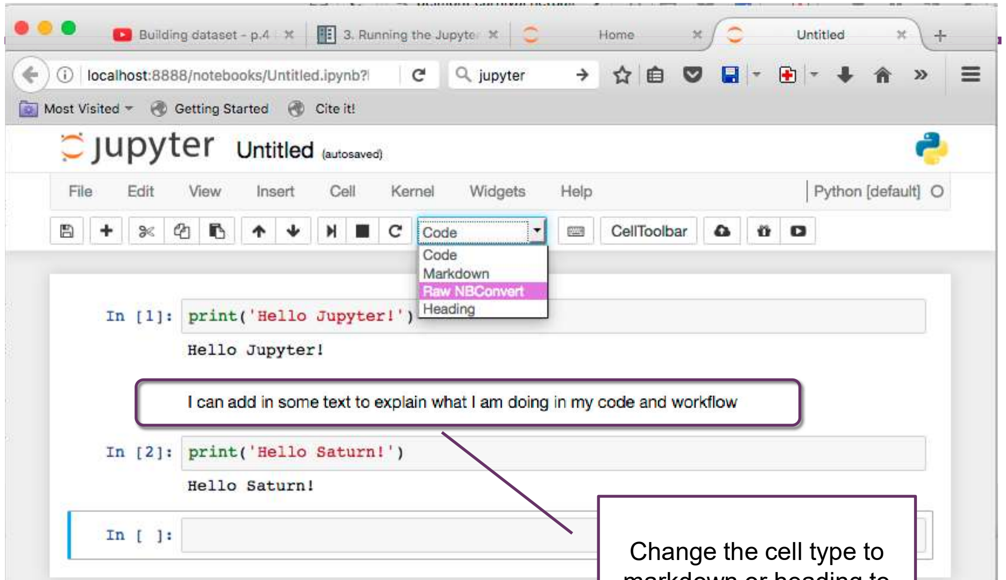
Jupyter dashboard

- Create a new notebook using default kernel…

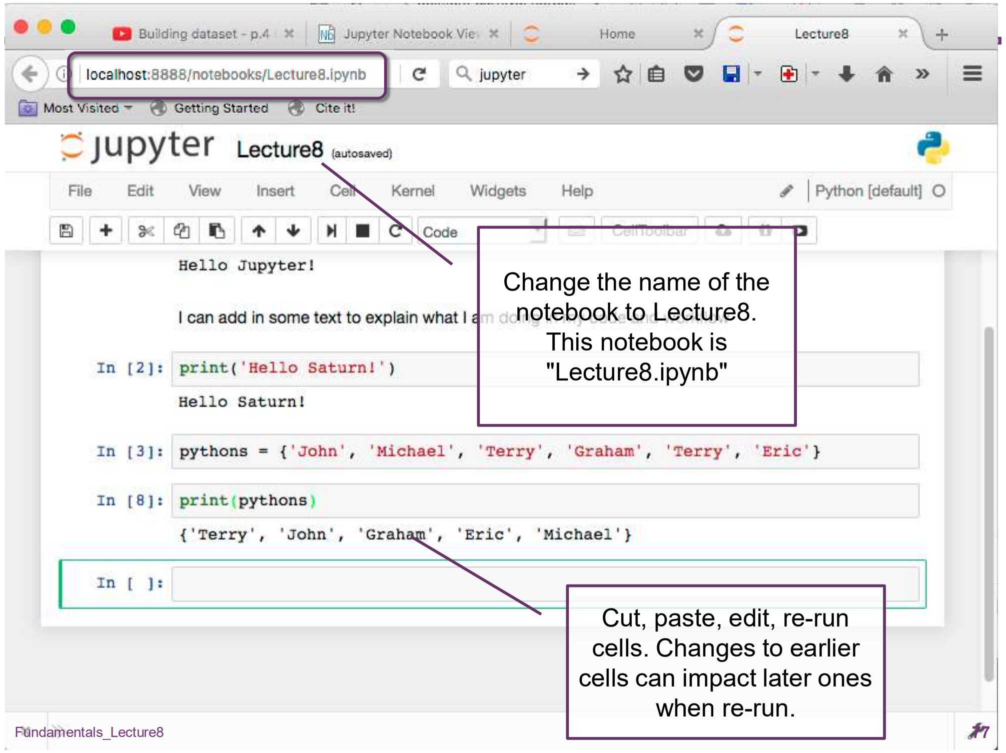


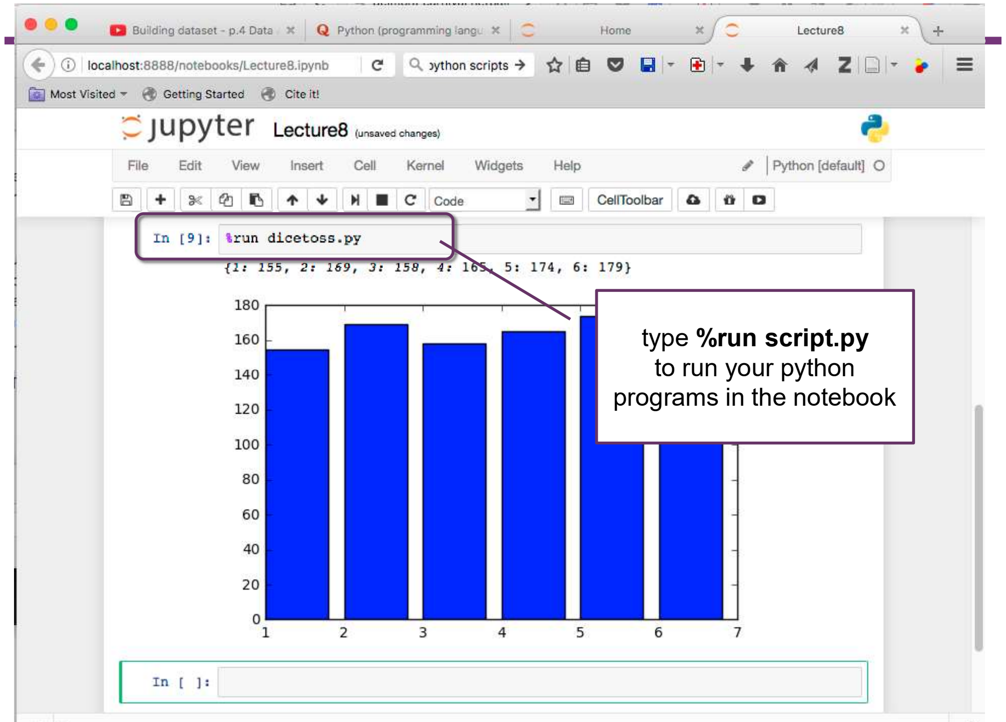
Jupyter notebook

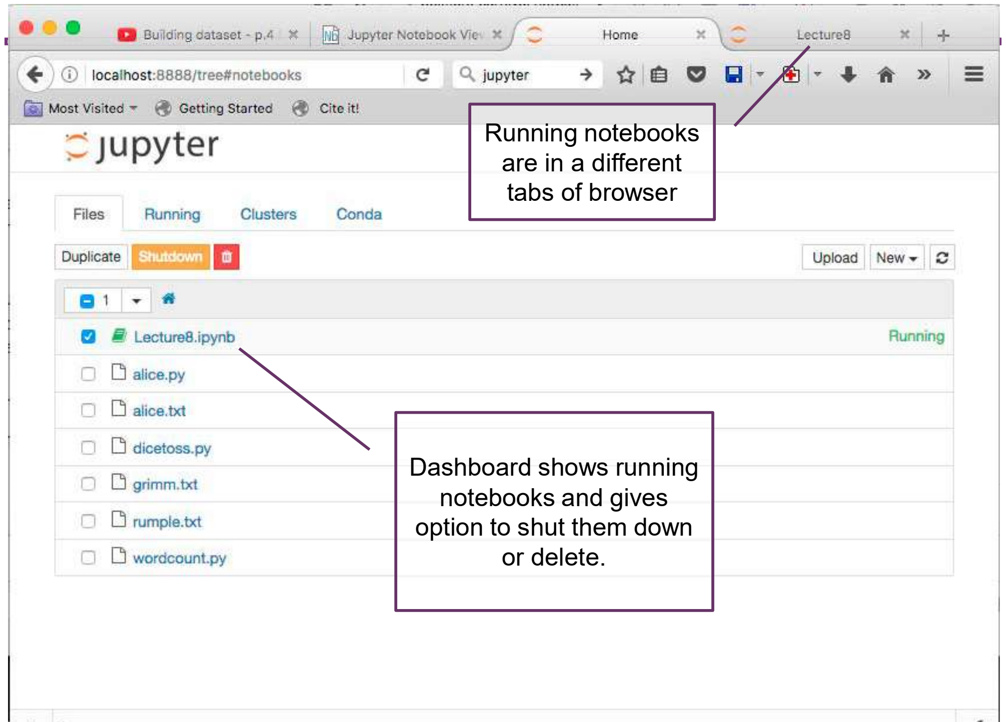
- Type in some Python code
- shift-enter to run the code



Change the cell type to markdown or heading to provide narrative or give structure to your document







Reproducibility and notebooks

- A notebook lets you:
  - explore ideas
  - present workflows
  - show overall logic of research process
  - refine analysis or workflow over time
  - create presentations
- It may even be useful for your assignment!

Reproducibility and provenance

- Notebooks maintain metadata and context information – vital for provenance and reproducibility

Versions

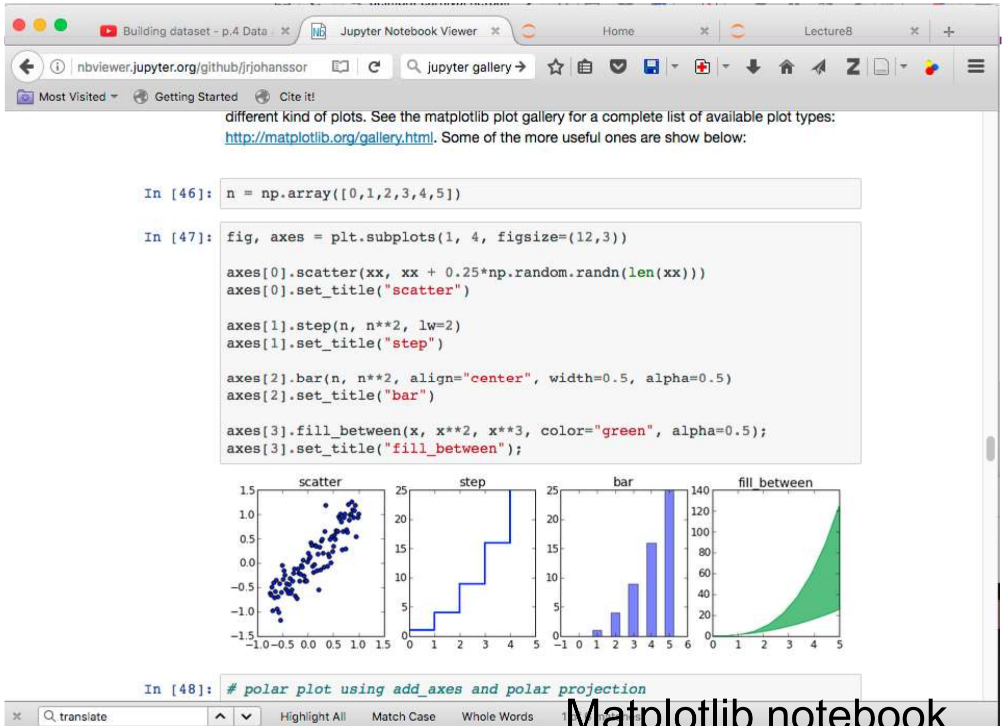
In [1]:
```
%reload_ext version_information
%version_information numpy, scipy, matplotlib
```

Out[1]:

Software	Version
Python	2.7.10 64bit [GCC 4.2.1 (Apple Inc. build 5577)]
IPython	3.2.1
OS	Darwin 14.1.0 x86_64 i386 64bit
numpy	1.9.2
scipy	0.16.0
matplotlib	1.4.3
Sat Aug 15 11:30:23 2015 JST	

Some examples to explore

- Notebooks can be shared, some galleries:
- Main gallery for Jupyter project:
  - https://github.com/jupyter/jupyter/wiki/A-gallery-of-interesting-Jupyter-Notebooks
- Fabian Pedregosa's notebook gallery
  - http://nb.bianp.net/
- Example: an excellent matplotlib notebook
  - http://nbviewer.jupyter.org/github/jrjohansson/scientific-python-lectures/blob/master/Lecture-4-Matplotlib.ipynb



Matplotlib notebook

Summary

- We've learnt a few more datatypes
- We've looked at how to use and implement structured data processing in Python using the pandas library
- We can see the value of reproducible research
- We can create and use Python notebooks to support exploratory research