A Quick Introduction to Python

Chase AHDA Python workshop
FutureLearn Offices, Camden, UK
Monday 14 February 2019

Purpose

This workshop is not going to teach you how to be a Python programmer in one day. However, *it will* introduce you to enough Python, and supply you with enough sample code, to let you make progress in using and writing scripts for your research. Further, it will introduce you to the three useful resources from the introductory slides that will let you work independently afterwards.

As a recap the resources are:

- W3C Python tutorial
- Python Manual - bundled with Python and in the `Doc` sub-folder of Python itself
- nltk online book

Datatypes

There are many datatypes in Python. We will look at the three that are most useful in the digital humanities: string, lists and dict[ionarie]s.

Strings

Strings are enclosed by quotes, advisable to use single quotes.

- `part1_01.py` simple strings and variables
- `part1_02.py` and how they compare to integers, so look closely!
- `part1_03.py` and this what happens if you try to use an undefined variable
- `part1_03.py` and what happens if you try to use an undefined variable

Lists

A list is marked by square brackets, and can contain any other datatype, even another list.

- `part1_04.py` a simple list, and how computers count
  OBOE Off By One Error - computers count from zero, humans from one
  Tuples are similar to lists, but are *immutable*.
  You cannot change their contents.
  Marked by round brackets instead of square brackets.
  They have their place, but for our purposes we shall stick with lists.

- `part1_05.py` shows some of the operations you can perform on a list Go to Python manual for page of list operations

- `part1_06.py` introducing if…else… in a literate form
- `part1_07.py` some more style options, aimed at making scripts easier to read

Bonus - autocomplete

IDLE – use Ctrl + space

Spyder – use tab

Naming things - good practice

With autocomplete there's no need to save typing:

```
python w = ['Mary', 'had', 'a', 'little', 'lamb']
```

Using **w** instead of **words** only confuses. Using good names makes the code self-documenting.

Note also the use of plurals, **words** for the list, and **word** for the individual entry in the list.

What does this Python function do?

```
python ts('trump', 20, False, True)
```

And this one?

```
python twitter_search('trump', numtweets=20, retweets=False, unicode=True)
```

Note also the use of **snake_case**, that's the joining together of words by an underscore. The other common way of doing this in computer languages is **CamelCase**, using capital letters to distinguish joined words. Pythonic code uses **snake_case**.

Hands on

Experiment with lists guided by W3C or Python manual using the IDLE shell.

Dictionaries

A unique key followed by data, which can be a string, an integer, a list, even another dictionary. Marked by braces.

- `part1_08.py` a simple example of a dictionary

Dictionaries are very useful when working with structured data, as we will see.

Note, order of entries in a dictionary is not guaranteed. For that, you need to use an `OrderedDict`.

Sets

Sets are a special type of dictionary that have only keys. As a special type of dictionary they are also marked by braces.

An easy way to deduplicate a list is to convert it to a set, because the keys of a set must be unique.

- `part1_09.py` deduplication

Note, 'a' is not the same as 'A' as far as the computer is concerned. We will deal with handling case later when looking at natural language processing.

Working with files

Plain text

The old, simple way is to explicitly open and close the file yourself. Better now to use `with` to manage the ‘context’.

- `part2_01.py` read a file, not quite as intended
- `part2_02.py` read a file, better

Talk through making mistakes and having the courage to try Always have two copies of your data, especially if writing data. Well, you always have a backup, don't you.

For completeness, here's how to write to a file:

- `part2_03.py` write a file

Talk through `print()` having a newline at the end of a line by default. In contrast `.write()` doesn't.

Parsing structured files

On the whole, pretty straightforward, just import the appropriate package.

json

- `part2_04.py` read a json file
- `part2_05.py` an easier way to read a json file

Pretty print does what it says, and is useful when looking at structured data. However, this data is really meant for computers to read, not humans.

While you read and write to a plain text file, you load and dump to a json file.

The data is built from the same datatypes of string, list and dict. Hence can reference 'title' as a key to retrieve its value. Similarly, with 'rioxx2_author' retrieves list of authors, so need to loop through.

One author has an ORCID id listed. Do you have yours yet?

xml

- `part2_06.py` read an xml file

Not terribly useful, just an Element. Show how to look this up in Python manual, and see useful methods, eg tag and text.

Python has several built-in XML parsers, of which `etree` is the easiest to use. If you have particularly demanding XML to processing then use the external package `lxml`. This is bundled with the standard Anaconda distribution,

- `part2_07.py` read an xml file more usefully

Working way down through tree to tags and text. Note use of namespaces, dc and rioxx. You might see more of RIOXX in your career. It's the [HEFCE metadata standard, RIOXX](https://github.com/atmire/RIOXX), and any submission you make to REF will have RIOXX metadata. Indeed, probably anything you publish through your institution will be assigned a RIOXX record by your library. And yes, RIOXX source is hosted on GitHub.

zip

To catch you out from the pattern above, for zip files the magic line is not `import zip` but `import zipfile`.

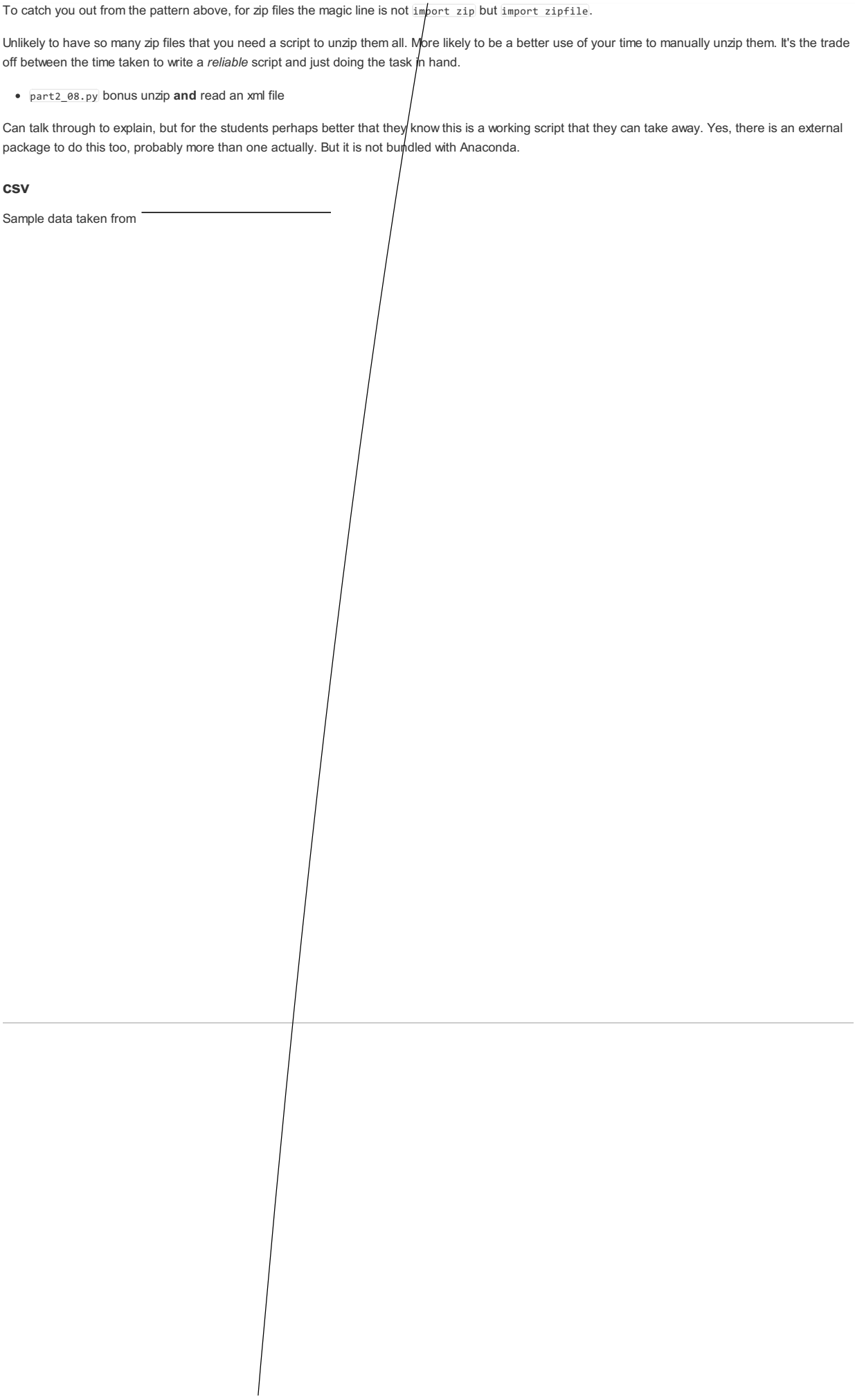
Unlikely to have so many zip files that you need a script to unzip them all. More likely to be a better use of your time to manually unzip them. It's the trade off between the time taken to write a *reliable* script and just doing the task in hand.

- `part2_08.py` bonus unzip **and** read an xml file

Can talk through to explain, but for the students perhaps better that they know this is a working script that they can take away. Yes, there is an external package to do this too, probably more than one actually. But it is not bundled with Anaconda.

csv

Sample data taken from _____



pdf

PDFs are different because a PDF is a **byte** file.

- `part3_08.py` download and save a local copy of a pdf web resource

PDFs are *not* a great way of sharing data, but sometimes it's all you'll get.

Processing files

- `part4_01.py` cut some sample data Either to remove header and footer information from a file, or to cut a smaller sample for testing purposes.
- `part4_02.py` remove leading spaces

Missing newline? print() has a new line by default write() doesn't. `f_out.write(line.strip() + '\n')`

- `part4_03.py` remove redundant newlines
- `part4_04.py` remove redundant newlines with a regular expression

- `part4_05.py` find relevant lines

- `part4_06.py` find relevant lines using a different test
- `part4_07.py` find relevant lines and change the line

Break to explore Python's string methods.

Working with words

- `part4_08.py` extract words
- `part4_09.py` remove stopwords

Working with files

- `part4_09.py` looping through all files in a folder

Natural Language Processing

nltk

Originally developed as a teaching tool, so well documented and doesn't reinvent the wheel but makes use of existing NLP tools.

There are many resources and languages, including Arabic and Chinese, supported by nltk.

It has excellent documentation, including a free online book Natural Language Processing with Python Printed versions are available for money.

Tokenizing

See the approach used by nltk, by default punctuation remains but as separate tokens.

- `part5_01.py` tokenize text

Should fail with need to download punkt:
`Line 11: nltk.download('punkt')` Then will fail because word_tokenize not found, need to state it's in the nltk package:
`Line 16: tokens = nltk.word_tokenize(raw_text)`

Word counts

A simple, but useful statistic.

- `part5_02.py` most frequent words

As well as `nltk.FreqDist()` for the *frequency distribution*, there is `nltk.ConditionalFreqDist()`, which compares word frequency distributions across corpora. Very useful as a way into understanding the content and sentiment of a text, and a necessary step in stylistic analysis.

- `part5_03.py` conditional frequency words
- `part5_03_with_function.py` conditional frequency words

In effectively one line of Python, so much is calculated for you… …and from this can plot(), tabulate(), etc because these functions are built in.

nltk has many corpora available. Run nltk.download() from a Python shell to access and download them.

Words in context

The first building block is to create word pairs.

- `part5_04.py` count of bigrams

And then we go down the rabbit hole of metrics.

- `part5_05.py` collocates

Missing stopwords? Follow the instructions in the error message. Are incorporated as line 11 in sample.

Collocates is another useful route into understanding the content and sentiment of a text, with more numbers to substantiate any conclusions.

Coming back to words… here is a more sophisticated way of finding keywords in texts. Especially useful to gain context. Can specify how much surrounding context to retrieve

- `part5_06.py` concordance

In this example, compare the negative words associated with 'monster' to the more positive words associated with 'creature'.

Understanding words

Some common NLP tasks

- `part5_07.py` stem words – *typographical root of a word*
- `part5_08.py` lemmatize words – *semantic root of a word*
- `part5_09.py` POS (parts of speech) tagging – *building block for other analysis*
- `part5_10.py` NER (named entity recognition) – *depends on your identifier*

Conclusion

A real world example of the use of Python in the digital humanities: On Building an Instagram Street Art Dataset and Detection Model

---

Appendix A: Sample data from ORO

BibTeX

Dublin Core

JSON

PDF

XML

Appendix B: Sample web links to harvest

Japanese attitudes to smart cities

Reportage of Syrian conflict in NY Times

Populism in leader's speeches during the 2017 GE

Linguistic descriptors applied to different music genres

Working on how World Bank describes poverty

BBC Pidgin news

UN on international development

Key themes and countries represented in nettime list archives