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 three useful resources that will let you work independently afterwards.

Datatypes

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
- `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

IDLE bonus - autocomplete

Using Ctrl + space

```
>>> words = ['Mary', 'had', 'a', 'little']
>>> words
['Mary', 'had', 'a', 'little']
>>> words.append('lamb')
>>> words
```

```
['Mary', 'had', 'a', 'little', 'lamb']
>>>
```

Naming things - good practice

With autocomplete there's no need to save typing:

```
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?

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

And this one?

```
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**.

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.

```
>>> words = ('Mary', 'had', 'a', 'little', 'lamb')
>>> words
('Mary', 'had', 'a', 'little', 'lamb')
>>>words[2]
a
>>> words[3] = 'big'
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    words[3] = 'big'
TypeError: 'tuple' object does not support item assignment
>>>
```

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.

xml

- `part2_06.py` read an xml file

Not terribly useful, just an Element. Show how to look this up in Python manual.

Similarly Python re Match object.

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 sscrfipt 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

csv

Sample data taken from `BL_Labs_Broadcast_News_2010.csv`, supplied directly by Mahendra Mahey on one of this visits to the OU.

- `part2_09.py` read a csv file
- `part2_10.py` read a csv file as a dictionary
- `part2_11.py` read and write a csv file

This type of file comes in different **dialects**, At the simplest, all columns are separated by a comma. Any symbol that is not in the data can be used as a separator. They could also be separated by a tab, \t, or pipe, |. If your text has commas this is useful. Another way to address this problem is to enclose text within columns in double quotes. This is how Excel saves textual data for example. These different separators and use of quotes are what distinguish **dialects**. The `sample.csv` file in Excel dialect. The Python Manual guides you through specifying the appropriate **dialect** to the csv `reader` and `writer`.

sql, and other database formats

Python supports many databases, from old simple key/value ones such as Berkeley, through mainstream relational databases such as sqlite3, to contemporary document-centric noSQL databases such as MongoDB.

HTML and web scraping

You may have noticed that html was not covered in the above run through of structured data.

Python has a useful built-in web access package, `urllib`. If you have particularly demanding html requests, maybe you have to log in for example, then use the external package `requests`. This package is bundled with the standard Anaconda distribution,

To process the html use the external package BeautifulSoup documenation. This package is also bundled with the standard Anaconda distribution,

- `part3_01.py` download and save a local copy of an html web page
- `part3_02.py` extract text from html - what does the computer understand as text?
- `part3_03.py` extract other data from html

Typically modern web pages are dynamic, they are built on the fly from relevant content drawn from databases and web services. Older style web pages typically were static. The page was written, without any elements that changed without the intervention of a developer to rewrite the code.

- `part3_04.py` download and save a local copy of a static html web page
- `part3_05.py` extract text from static html

Hold on to the downloaded data because we're going to use it later in this session when we look at data cleaning.

There are better methods of sharing data over the web than writing web pages. Typically, web sites will expose data directly in a computer readable format other than html.

Other formats

json

Building on the popularity of JavaScript, data is encoded in **JavaScript Object Notation**. This is a subset of the datatypes available in JavaScript.

It requires special handling:

- `part3_06.py` download and save a local copy of a json web resource

xml

XML is handled in a similar fashion to json because both are text files.

- `part3_07.py` download and save a local copy of an xml web resource

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
- `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

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 for 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...

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...

Coming back to words...

- `part5_06.py` concordance

Understanding words##

Some common NLP tasks

- `part5_07.py` stem words
- `part5_08.py` lemmatize words
- `part5_09.py` POS (parts of speech) tagging
- `part5_10.py` NER (named entity recognition)

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