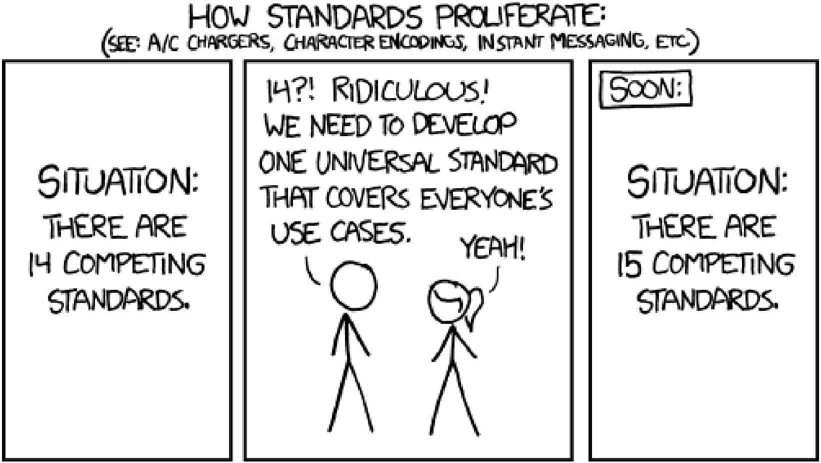
Applied Data Science

Data Ingress

Niall Twomey

niall.twomey@bristol.ac.uk

University of Bristol


HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC.)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

`https://xkcd.com/2054/`

By the end of this lecture you should be...

- ... familiar with the most used data structures in data science
- ... familiar with the most used data formats in data science
- ... familiar about how to work with APIs

By the end of this lecture you should be...

- ... familiar with the most used data structures in data science
- ... familiar with the most used data formats in data science
- ... familiar about how to work with APIs

Later lectures will introduce...

- ... cleaning data
- ... working with databases
- ... visualising data
- ... ‘data science’ :-)

Jupyter notebooks

- Jupyter notebooks will be very useful in the unit
- These will demonstrate the concepts/examples directly in the browser

- Useful links:

Git	`https://www.atlassian.com/git/tutorials/install-git`
Python	`https://www.python.org/`
Jupyter	`http://jupyter.org/install.html/`
In-browser	`https://repl.it/languages/python`
	`https://colab.research.google.com/`

Outline of the lecture

Data structures

- List, Array, matrix, Dictionary

Data formats

- CSV, pandas, JSON, HDF5

Web-scraping and APIs

- Beautiful Soup, Regular expressions, Scrapy

Data structures for data science

Native data structures

- `list`: python list
- `set`: python set
- `dict`: python dictionary

Data structures from other modules

- `np.array`: numpy array
- `pandas.DataFrame`: pandas dataframe

Object persistence between sessions

Serialisation

Serialisation is the process of translating data structures or objects from memory into a format that can be stored

Deserialisation

Deserialisation is the inverse process; translating data structures that have been stored in a particular format to memory

https://en.wikipedia.org/wiki/Serialization

Serialisation of data structures:

- Bespoke serialisation and deserialisation methods can be crafted manually

Serialisation of data structures:

- Bespoke serialisation and deserialisation methods can be crafted manually
- Example: Define a simple serialisation format for list or array:
  - Instantiate an output file object
  - Write each element of the list to file, letting one and only one element be written per line
  - Close the file

Serialisation

```
# Create list
v = [1, 2, 3, 4, 5]

# Write it to file
f = open("d.ivec", "w")
for el in v:
    f.write("%d\n" % el)
f.close()
```

```
# Alternatively:
with open("d.ivec", "w") as f:
    f.write("\n".join(map(str, v)))
```

Serialisation

```python
# Create list
v = [1, 2, 3, 4, 5]

# Write it to file
f = open("d.ivec", "w")
for el in v:
    f.write("%d\n" % el)
f.close()
```

```python
# Alternatively:
with open("d.ivec", "w") as f:
    f.write("\n".join(map(str, v)))
```

Deserialisation

```python
# Instantiate list
v = []

# Read the file
f = open("d.ivec", "r")
for l in f.readlines():
    v.append(int(l))
f.close()

# Print features of the data
print(v)          # [1, 2, 3, 4, 5]
print(len(v))     # 5
print(v[2])       # 3
```

```python
# Alternatively:
with open("d.ivec", "r") as f:
    v = map(int, f.readlines())
```

Problems with bespoke serialisation:

- Very specific use case
- Format not standardised
- The above example is not robust in its current (naïve) state
- Needs to be tested against many test cases
- No object metadata encoded (e.g. data type, length)
- Every data structure (e.g. matrices, dictionaries, list of strings) requires a (de)serialisation method

Problems with bespoke serialisation:

- Very specific use case
- Format not standardised
- The above example is not robust in its current (naïve) state
- Needs to be tested against many test cases
- No object metadata encoded (e.g. data type, length)
- Every data structure (e.g. matrices, dictionaries, list of strings) requires a (de)serialisation method

... but should be fine if using in well-controlled situations

Comma-separated values (CSV)

- Very suited to tabular data, particularly matrices
- A row is stored as a line
- Each element in the row is separated by a comma

- Example CSV File:

```
1, 2, 3,
4, 5, 6,
7, 8, 9,
```

Loading CSV[1] file with python

- Easy to write own parser, but will use the pandas python package to load CSV data: `http://pandas.pydata.org/`
- The `pandas` library performs intelligent type conversion and checking
- Provides a powerful `DataFrame` object

Source code

```python
from pandas import read_csv
df = read_csv("csv.csv")
print df
```

Output

```
   0  1  2
0  1  2  3
1  4  5  6
2  7  8  9
```

- CSV files can be a very time efficient and space efficient format choice for tabular data

[1] `https://tools.ietf.org/html/rfc4180`

Data Types in Data Science

Data Types

- Dense data
- Sparse data
- Text data
- Structured/relational data
- Categorical/Ordinal
- Date/time
- Lat/lon

Data Types in Data Science

Data Types	Data characteristics
Dense data	CSV should be fine :-)
Sparse data	Don’t store in dense format
Text data	How to store efficiently
Structured/relational data	Handle relationships
Categorical/Ordinal	Handling categorical constraints
Date/time	Retrieving time zone
Lat/lon	Retrieving location

Serialising generic objects:

[2] https://en.wikipedia.org/wiki/JSON

[3] https://en.wikipedia.org/wiki/Hierarchical_Data_Format

[4] https://en.wikipedia.org/wiki/Category:Data_serialization_formats

Serialising generic objects:

- JSON (JavaScript Object Notation)[2]
  - Human readable, `dict`-like format
  - Very robust language; suits many purposes

[2] https://en.wikipedia.org/wiki/JSON

[3] https://en.wikipedia.org/wiki/Hierarchical_Data_Format

[4] https://en.wikipedia.org/wiki/Category:Data_serialization_formats

Serialising generic objects:

- JSON (JavaScript Object Notation)[2]
  - Human readable, `dict`-like format
  - Very robust language; suits many purposes
- HDF5 (Hierarchical Data Format)[3]
  - Binary format
  - File system-like access

Will not cover other formats in this lecture (e.g. XML (and related variants), Protocol buffers or YAML and others[4])

[2] https://en.wikipedia.org/wiki/JSON

[3] https://en.wikipedia.org/wiki/Hierarchical_Data_Format

[4] https://en.wikipedia.org/wiki/Category:Data_serialization_formats

JavaScript Object Notation (JSON)[5]

- JSON is a syntax for storing and exchanging data
- JSON is text, written with JavaScript object notation standard
- Although initially designed for javascript, JSON is a common serialisation in many languages, APIs, and communication frameworks, e.g. REST APIs.
- We can convert JSON into objects in memory
  - May need to create specific conversion process.
- JSON is a very well defined standard
  - "Because it is so simple, it is not expected that the JSON grammar will ever change. This gives JSON, as a foundational notation, tremendous stability" [6]

[5] www.w3schools.com/js/js_json_intro.asp

[6] www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf

JSON code:

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "mobile",
      "number": "123 456-7890"
    }
  ],
  "children": [],
  "spouse": null
}
```

JSON code:

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "mobile",
      "number": "123 456-7890"
    }
  ],
  "children": [],
  "spouse": null
}
```

Python code:

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "mobile",
      "number": "123 456-7890"
    }
  ],
  "children": [],
  "spouse": None
}
```

- Some distinctions between JSON and Python `dicts`

	`dict`	JSON
Missing values	`None`	`null`
String character	’ or "	" only
Dictionary keys	any hashable object	strings

- Demonstrations:
- JSON validation: `http://www.jsonlint.com`
- JSON files can become large (due to `key` repetition). Transposing lists of dictionaries into a dictionary of lists will save space in general.

Original JSON:

```
[
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "mobile",
      "number": "123 456-7890"
    },
    {
      "type": "home2",
      "number": "212 555-1234"
    },
    {
      "type": "mobile2",
      "number": "123 456-7890"
    }
]
```

Transposed JSON:

```
{
    "type": [
      "home",
      "mobile",
      "home2",
      "mobile2"
    ],
    "number": [
      "212 555-1234",
      "123 456-7890",
      "212 555-1234",
      "123 456-7890"
    ]
}
```

Hierarchical Data Format 5 (HDF5):

- HDF5 is a format for serialising data
- Core concepts:
  - Datasets: array-like collections of data
  - Groups: folder-like structures that contain datasets and other groups
  - Metadata: add information that pertains to all datasets
- HDF5 lets you store huge amounts of numerical data, and easily manipulate that data from `numpy`.
- Thousands of datasets can be stored in a single file, categorised and tagged however you want.
- Unlike `numpy` arrays, they support a variety of transparent storage features such as compression, error-detection, and chunked I/O.

```
import h5py
import numpy as np

# Create a HDF5 file
f = h5py.File("mytestfile.hdf5", "w")

# Add a new dataset to the file: integer array of length 100
dset1 = f.create_dataset("mydataset", (100,), dtype="i")

# Assign values to the dataset
dset[...] = np.arange(100)

# Add a group called subgroup, with a dataset underneath
dset2 = f.create_dataset("subgroup/dataset_two", (10,), dtype="i")

# Store metadata in the HDF5 file object
dset.attrs["author"] = "nt"
dset.attrs["date"] = "24/01/2018"
```

http://docs.h5py.org/en/latest/quick.html

Effectively, you can see HDF5 as a file system within a file, where files are datasets and folders are groups. However, the HDF Group doesn't seem to like this comparison. The major differences are as follows:

- An HDF5 file is portable: the entire structure is contained in the file and doesn't depend on the underlying file system. However it does depend on the HDF5 library.
- HDF5 datasets have a rigid structure: they are all homogeneous (hyper)rectangular numerical arrays, whereas files in a file system can be anything.
- You can add metadata to groups, whereas file systems don't support this.

`http://cyrille.rossant.net/moving-away-hdf5/`

Web Scraping

Web scraping should be done in accordance with the website's terms of use.

https://www.crummy.com/software/BeautifulSoup/zine/

Web scraping:

- Web scraping (web harvesting or web data extraction) is the term given for acquiring data from websites
- Scraping technologies must be tolerant to several artefacts of real-world data (so-called ‘wrangling’ will be covered in later lectures)
- The erroneous data makes parsing information difficult
- In the context of scraping data from the web:
  - `BeautifulSoup`[7]; and
  - `scrapy`[8]
  - `Selenium`
- First we need to know a little about webpages, however.

[7] https://www.crummy.com/software/BeautifulSoup/

[8] https://scrapy.org

```html
<!DOCTYPE html>
<html>
  <head>
    <title>Scraping</title>
  </head>
  <body class="col-sm-12">
    <h1>section1</h1>
    <p>paragraph1</p>
    <p>paragraph2</p>
    <div class="col-sm-2">
      <h2>section2</h2>
      <p>paragraph3</p>
      <p>unclosed
    </div>
  </body>
</html>
```

```html
<!DOCTYPE html>
<html>
  <head>
    <title>Scraping</title>
  </head>
  <body class="col-sm-12">
    <h1>section1</h1>
    <p>paragraph1</p>
    <p>paragraph2</p>
    <div class="col-sm-2">
      <h2>section2</h2>
      <p>paragraph3</p>
      <p>unclosed
    </div>
  </body>
</html>
```

- `<!DOCTYPE html>`: HTML documents must start with a type declaration.
- The HTML document is contained between `<html>` and `</html>`.
- The meta and script declaration of the HTML document is between `<head>` and `</head>`.
- The visible part of the HTML document is between `<body>` and `</body>`.
- Title headings are defined with the `<h1>` to `<h6>` tags.
- The section/division tags `<div>` are often used to segment the source.
- Paragraphs are defined with the `<p>` tag.

- Interactive DOM explorer: `https://javascript.info/dom-nodes`

Summary of Beautiful Soup:

- Beautiful Soup is a powerful library for parsing XML/HTML documents
- With this library, one can create information extraction engines that can run autonomously
- However, HTML scraping has many serious problems:
  - Selecting the ‘correct’ link to follow is a strong function of the ability to identify the link of interest based only on its HTML tags. This can be difficult in poorly designed websites.
  - If the layout or formatting of information on a webpage changes, it will become necessary to reconfigure the web scraping mechanism from scratch
- Tests for format changes should be defined

Web APIs:

- Web APIs provide a portal for explicit data acquisition, and are generally less prone to the issues from HTML scraping since:
  - Code is optimised for retreival and not for visual layout/aesthetics
  - Standard serialisation tools (*e.g.* JSON) are typically used
  - The core items of interest have been extracted (*e.g.* dates, URLs)

Web API examples:

- `https://github.com/toddmotto/public-apis`

Web API examples:

- `https://github.com/toddmotto/public-apis`
- Web APIs, although similar in principle, will have very different schemas.
- However, writing code for APIs is generally simpler than parsing HTML, requires less maintenance, are often documented, and results in faster overall code

Web API examples:

- `https://github.com/toddmotto/public-apis`
- Web APIs, although similar in principle, will have very different schemas.
- However, writing code for APIs is generally simpler than parsing HTML, requires less maintenance, are often documented, and results in faster overall code

Example

- Task: acquire a list of recent posts from the 'technology' section of a newspaper
  `http://open-platform.theguardian.com/`
  `http://open-platform.theguardian.com/documentation/`

RESTful APIs:

- Representational state transfer (REST) or RESTful Web services are one way of providing interoperability between computer systems on the Internet
- In most circumstances API keys are required before data can be accessed

https://en.wikipedia.org/wiki/Representational_state_transfer

RESTful APIs:

- Representational state transfer (REST) or RESTful Web services are one way of providing interoperability between computer systems on the Internet
- In most circumstances API keys are required before data can be accessed

- RESTful APIs define a collection of resources

  `https://content.guardianapis.com/sections?api-key=test`

- For each resource, RESTful APIs additionally provide a list of items

  `https://content.guardianapis.com/technology?api-key=test`

https://en.wikipedia.org/wiki/Representational_state_transfer

```
# Specify the arguments
args = {
    "section": "technology",
    "order-by": "newest",
    "api-key": "test",
    "page-size": "100"
}

# Construct the URL
base_url = create_guardian_url(args)

# Make the request and extract the source
response = json.loads(requests.get(url).text)

# Print the data that is available
print response.keys()
#  ["currentPage", "orderBy", "pageSize", "pages", "results",
#   "startIndex", "status", "total", "userTier"]
```

Generic queries: regular expressions:

- Regular expressions are sequences of characters that define a search pattern
- In Python there are two main options foe executing regular expressions: `re.match` and `re.search`
- These two functions are similar, but with distinct differences

```
# This function attempts to match RE pattern to the whole string
re.match(pattern, string, flags=0)

# This function searches for *first* occurrence of a pattern
re.search(pattern, string, flags=0)
```

https://en.wikipedia.org/wiki/Regular_expression

```
import re

line = "Cats are smarter than dogs"
matchObj = re.match( r"(.*) are (.*?) .*", line, re.I)

print "matchObj.group():", matchObj.group()
# "Cats are smarter than dogs"

print "matchObj.group(1):", matchObj.group(1)
# "Cats"

print "matchObj.group(2):", matchObj.group(2)
# "dogs"
```

http://pythex.org/

Summary on APIs:

- API-based querying is robust, reliable, well maintained and documented with a static schema
  - HTML-based web scraping is not
- Information extraction is not based on fickle naming conventions of tag attributes
- Since only content is acquired (and no images, javascript or style files) the bandwidth spent to acquire data is reduced significantly. This naturally lends itself to faster processing
- Since access is acquired through API keys, it is easy for the service provider to manage the bandwith and throughput of its service as necessary
- The data structures being received from APIs are complex, and need specific serialisation tools to be built.

Resources for the lecture

- Python tutorials:
  `https://docs.python.org/2/tutorial/`
- ‘Python for Data Analysis’
  `http://shop.oreilly.com/product/0636920023784.do`
- JSON Verification:
  `http://jsonlint.com/`
- Serialisation techniques:
  `https://en.wikipedia.org/wiki/Comparison_of_data_serialization_formats`