Data Structures and Algorithms

Lecture 5

Heikki Peura

h.peura@imperial.ac.uk

Plan for today

Accessing data on the web:

- Scraping
- APIs

Python:

- Libraries
- Looping practice

Dividing code into modules

my_algorithms.py

```python
def lin_search(L, x):
    for elem in L:
        if elem == x:
            return True
    return False
print('Running my_algorithms.py')
```

working.py

```python
import my_algorithms as alg # "as" part is optional

from my_algorithms import lin_search # specific function

L = [1, 2, 2, -2, 9]
found_10 = alg.lin_search(L, 10)
found_2 = lin_search(L, 2)
```

When you import, Python will run all of `my_algorithms.py`

Using code from libraries

```python
import numpy as np
import matplotlib.pyplot as plt
```

There are thousands of Python libraries

- Anaconda packages "the big ones"
- Command line: `conda list`
- Update a library:
  1. Close everything Python-related (Spyder, Notebook...)
  2. `conda update package_name`
  3. Be careful: updating may sometimes remove old functionality

The web is full of data and services

Quick and repeated access could be useful for:

- Analysing tweets / government spending / ???
- Getting data to/from Google maps / AWS / Bloomberg?
- Finding an apartment in London / best coffee close to you?

We can use APIs or scraping for access

A web page is just text

Most pages are in HTML (HyperText Markup Language)

The difference:

```
Hello, world!

Hello again.

An example page: http://www.example.com/
```

```html
<p>Hello, world!</p>

<p>Hello again.</p>

<p><a href="http://www.example.com/">An example page</a></p>
```

HTML tags tell browser how to display content

Tags are within angle brackets: `<p>` is starting tag, and `</p>` is ending tag. Inside, there is the *paragraph* content.

```
<p>Hello, world!</p>
```

Tags can be nested:

```
<p>Hello, <strong>world!</strong></p>

<p>Hello again.</p>
```

`<p>` and `<strong>` are parent and child tags, and the two `<p>` tags are sibling tags.

Tags can have attributes

For example, the hyperlink tag `<a>` specifies the target URL as the `href` attribute in the `opening` tag:

```
<p><a href="http://www.example.com/">An example page</a></p>

<p id="myid">Hey there</p>
```

Web sites use custom attributes to make content look fancy with CSS (cascading style sheets): for example, change how `myid` looks.

We know how to parse text with Python

Websites use HTML tag structure to look nice

- We can exploit these tags to parse their contents
- For example, find all links in a web page through `<a href...>`
- Great libraries exist!

Parsing HTML with Python

We will use a library called Beautiful Soup 4 (BS)

1. Detective work to identify what we want in the page (right tags)
2. Looping through tags with BS

```python
from bs4 import BeautifulSoup
html = '<p>Hello, world!</p> <p>Hello again.</p>'
soup = BeautifulSoup(html, 'lxml') # lxml is the html parser BS uses
print(soup.text)
```

The `soup` variable contains a `BeautifulSoup` object.

- The `soup.text` attribute contains the text content

Finding things in soup

Let's find all tags:

```
>>> paragraphs = soup.find_all('p') # find all <p> tags
>>> paragraphs
[<p>Hello, world!</p>, <p>Hello again.</p>]
>>> type(paragraphs) # augmented version of a list
bs4.element.ResultSet
>>> first = paragraphs[0]
>>> first.name
'p'
>>> first.text
'Hello, world!'
>>> type(first)
bs4.element.Tag
```

Browsing with Python

We will use a library called requests:

```
import requests
r = requests.get('http://www.example.com')
```

What is r? An object with attributes. For example:

```
r.ok # was the access attempt successful
r.text # raw HTML
```

Together with soup

Let’s go to example.com:

```
import requests
from bs4 import BeautifulSoup
r = requests.get('http://www.example.com')
soup = BeautifulSoup(r.text, 'lxml')
links = soup.find_all('a')
```

Get link locations from tag attributes:

```
>>> first_link = links[0]
>>> first_link.text
'More information...'
>>> first_link.attrs # dictionary of attributes
{'href': 'http://www.iana.org/domains/example'}
>>> first_link.attrs['href']
'http://www.iana.org/domains/example'
```

Soup workflow

1. Use requests to get HTML
2. Create a soup object with BS
3. Inspect the HTML in browser to find the tags you need
4. BS: `find` (first) and `find_all` methods are often useful
5. BS: `text` and `attrs` attributes are often useful
6. Get data into a Python data structure (list, dictionary, ...)
7. Save to a file

BS has a great documentation online.

Scraping ethics

Websites don't always like scraping

- Be polite!
  - Terms for access
  - `from time import sleep`
- Better to use APIs if available
  - Bonus: more robust access and cleaner data

What are APIs?

Application programming interface

A controlled way to access a service. Widely used:

- Google maps, TfL, Spotify, NY Times, Gmail, . . .
- Easy and robust access to data in nice form
- Typically need to register to use
- Limitations: types of data, access frequency

What do you get from an API?

Data often in JSON or XML format.

- Both forms are common for semi-structured data
- Worth getting to know for analytics
- Great Python libraries for parsing both

Today: JSON (JavaScript Object Notation, *'Jason'*)

JSON, have we met before?

In general:

```
{"countries":
  [
  {"Germany": "Berlin"},
  {"France": "Paris"},
  {"Italy": "Rome"}
  ]
}
```

A tweet (some fields):

```
{"source": "Twitter for iPhone",
 "id_str": "815271067749060609",
 "text": "RT @realDonaldTrump: Happy Birthday @DonaldJTrumpJr!\nhttps://t.co/uRx
 "created_at": "Sat Dec 31 18:59:04 +0000 2016",
 "retweet_count": 9529,
 "in_reply_to_user_id_str": null,
 "favorite_count": 0,
 "is_retweet": true}
```

Why not all web services have APIs

Hacker Challenge

Workshop: HTML parsing and Twitter data

After the break...

Work with:

- JSON
- HTML

Workshop

Workshop zip file on the Hub

- Open Jupyter Notebook and find the `.ipynb`-file with skeleton code