# Scrape a website (simple)

This notebook has code to scrape the 12 most recent alerts from the Queensland Workplace Health and Safety website and write the data to a CSV file.

In [ ]:
```
'''
the convention is to put imports at the top of your script --
specifically, to put imports from the standard library first, then a
blank line, then imports from any third-party libraries you've installed
'''

# the standard library module for working with tabular data
import csv

# third-party library for handling HTTP traffic: https://docs.python-requests.org/en/maste
import requests
# third-party library for parsing strings of HTML into Python data objects: https://www.ci
from bs4 import BeautifulSoup
```

## Step 1: Request the page

1. Call the `requests.get()` method to request the page we're interested in, and save the results of that operation to a new variable
2. Verify that it worked by checking the `text` attribute of the object returned -- this is the HTML underlying the page we're scraping

In [ ]:
```
# use the requests.get() method to request the page
req = requests.get('https://www.worksafe.qld.gov.au/news-and-events/alerts')
```

In [ ]:
```
# take a look at the .text attribute of the request, which is the HTML code behind the pag
print(req.text)
```

## Step 2: Parse the HTML

1. Feed that HTML string into a new BeautifulSoup object.
2. Poke through the HTML on the page to figure out the elements we need to target -- Chrome/Firefox browser tools are great for this.
3. Having found the alerts in an unordered list ( `<ul>` ) element that has a class attribute of `search-results__grid` , use BeautifulSoup's `find()` method to extract it from the larger tree of HTML elements.
4. Use BeautifulSoup's `find_all()` method to extract all of the list items ( `<li>` ) inside the unordered list we just extracted -- this will return multiple elements stored in a Python *list*.
5. Using a Python for loop, loop over each list item and extract the pieces of data we want to capture for each alert.

In [ ]:
```
# parse that string of HTML into a BeautifulSoup object using the default
# 'html.parser' parsing engine
soup = BeautifulSoup(req.text, 'html.parser')
```

In [ ]:
```
# use the find() method to isolate the unordered list (ul)
# that contains the alerts
results_list = soup.find('ul', {'class': 'search-results__grid'})
```

In [ ]:
```
# within that ul, use the find_all() method to retrieve a list of
# list items (li)
results_items = results_list.find_all('li')
```

In [ ]:
```
# check to see how many items in the list
print(len(results_items))
```

In [ ]:
```
# loop over the list of items
for item in results_items:

    # find() the h4 level header within this list item
    header = item.find('h4')

    # grab the text from that header and strip any whitespace
    header_text = header.text.strip()

    # find() the anchor tag ('a') in the header and access the 'href' link attribute
    link = header.find('a')['href']

    # find() the paragraph tag, access the text attribute and strip whitespace
    description = item.find('p').text.strip()

    # find() the date span and grab the stripped text
    date = item.find('span', {'class': 'single-date'}).text.strip()

    # print everything we captured with a newline break between each thing
    print(date, header_text, link, description, sep='\n')

    # and a blank line to make things easier to see
    print('')
```

## Step 3: Write to file

Next, instead of just printing the pieces of data we've targeted, we'll write them to file using the `csv` package, which is part of Python's standard library.

---

⚠ **Note**: Typically, you'd write this code in the same loop you wrote above, just replacing the print statements with instructions to write to file, but for the purposes of this workshop we're breaking this into two code blocks. Generally, the process involves writing your code step by step, stopping frequently to test that you're capturing the data correctly, then once everything looks good, adapting the code to write the data to file (or whatever your output is).

---

1. Inside a `with` block, open a file to write the data into in write ( `w` ) mode.
2. Create a CSV writer object and write the headers to file.
3. Loop over the items as before, targeting the pieces of data to write out.
4. Instead of printing them, drop them into a Python list and write to file.

In [ ]:
```
# open a new CSV file in write mode
with open('qld-workplace-alerts-latest.csv', 'w') as outfile:
```

```
    # create a writer object
    writer = csv.writer(outfile)

    # make a list of headers for our CSV
    headers = ['date', 'hed', 'description', 'link']

    # ... and write them to file
    writer.writerow(headers)

    # loop over the list of items
    for item in results_items:

        # find() the h4 level header within this list item
        header = item.find('h4')

        # grab the text from that header and strip any whitespace
        header_text = header.text.strip()

        # find() the anchor tag ('a') in the header and access the 'href' link attribute
        link = header.find('a')['href']

        # find() the paragraph tag, access the text attribute and strip whitespace
        description = item.find('p').text.strip()

        # find() the date span and grab the stripped text
        date = item.find('span', {'class': 'single-date'}).text.strip()

        # build a list of data to write out to file
        data_to_write = [date, header_text, description, link]

        # ... and write to file
        writer.writerow(data_to_write)
```

## 🚨 Extra credit: Parse dates

Use the Python standard library's `datetime.datetime` module -- specifically, its `strptime()` method -- to convert the date string for each entry ("1 April 2020") into a Python date object as you loop over the items.

Here's an example of how to do this:

```
from datetime import datetime

test_date = "1 April 2020"

date_object = datetime.strptime(test_date, "%d %B %Y")

print(date_object)
```

Here's a handy website to bookmark with all of Python's date/time formatting directives.

And here's a link to the Python documentation on the `strftime()` and `strptime()` methods.

In [ ]:

## 🚨 Extra extra credit: Send your output elsewhere

Instead of just writing this data to file, you could also:

- Send newsworthy alerts to a Twitter bot using Tweepy or another Python wrapper over the Twitter API
- Send new alerts to a Slack channel using the official Python SDK (Software Development Kit) or by setting up a webhook URL and using the `requests` library to POST your message data to that URL
- Ingest your new CSV data into a data analysis workflow using pandas

In [ ]: