Scrape a website (paginated)

This notebook has code to scrape every page of alerts from the Queensland Workplace Health and Safety website and write the data to a CSV file.

We will reuse the core extraction logic of our simple scraper but extend the script to capture multiple pages. The steps will be:

1. Fetch the initial landing page and parse the HTML to determine how many pages we need to spin through
2. Request each page in turn and download a copy onto the computer
3. Open each downloaded page and parse the HTML into data to write to 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
# a standard library module for dealing with time-based tasks -- in
# our case, to "sleep" for a second between requests
# https://docs.python.org/3/library/time.html#time.sleep
import time
# a standard library module to run local file searches
import glob


# 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.c
from bs4 import BeautifulSoup
```

Step 1: Request the base page and figure out how many pages to grab

First, we're going to fetch the first page of alerts so we can see how many pages we need to fetch. Looking at how the URL changes as you page through the results, you can see that the query parameter that changes is `start_rank` -- the offset.

Each page shows 12 results, so page 2 has a `start_rank` of 13, page 3 has a `start_rank` of 25, and so on. That's going to be the key pattern we'll hook into to capture all of the pages.

The pagination controls are at the bottom of the page. You'll notice a link that allows you to jump to the last page of results -- that link will answer our question of "how many pages do we need to visit to get everything?"

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

In [ ]:
```
# parse the HTML into soup
```

```
soup = BeautifulSoup(req.text, 'html.parser')
```

In [ ]:
```
# target the element that has the link to the last page
last_page = soup.find('li', {'class': 'pagination__item--last'})
```

In [ ]:
```
# target the anchor tag (a) and grab the 'href' link attribute
last_page_link = last_page.find('a')['href']
```

In [ ]:
```
# use the split() function to grab just the value for the final page start_rank
# and pass it to the int() function to turn the string into a number
last_page_offset = int(last_page_link.split('start_rank=')[-1])
```

In [ ]:
```
print(last_page_offset)
```

Step 2: Request the pages and save them to file

We're going to save each page into a folder called `scraped-pages` (I set this up ahead of time). Why? This way, if our script crashes partway through, we'll at least have those pages cached and can pick up where we left off.

The process:

1. Generate the URL for each page based on its offset using a range() with an offset of 12
2. Retrieve each page and save the HTML contents to file

In [ ]:
```
# how many items on each page?
items_per_page = 12
```

In [ ]:
```
# set up the URL template with only the offset value missing
url_pattern_base = 'https://www.worksafe.qld.gov.au/news-and-events/alerts?start_rank='
```

In [ ]:
```
# the name of the folder where our files will land
save_folder = 'scraped-pages'
```

In [ ]:
```
# now let's test out the logic for the range to make sure the `start_rank` params will be
# we have to grab the last page offset PLUS ONE for the second value
# because ranges are exclusive at the top end of the range --
# more details: https://docs.python.org/3/library/stdtypes.html#range
for i in range(1, last_page_offset+1, items_per_page):
    print(i)
```

In [ ]:
```
# now that we know the query parameters are generated correctly,
# we can weave in the actual "grab this page and download it" code

# loop over the range
for i in range(1, last_page_offset+1, items_per_page):

    # build the URL using an f-string
    # https://docs.python.org/3/tutorial/inputoutput.html#tut-f-strings
    url = f'{url_pattern_base}{i}'
```

```
    # define the filepath for where we're going to save this page
    filepath = f'{save_folder}/qld-workplace-alerts-{i}.html'

    # fetch the page
    req = requests.get(url)

    # save to file
    with open(filepath, 'w') as outfile:
        outfile.write(req.text)

    # let us know what's up
    print(filepath)

    # wait a tick
    time.sleep(1)
```

Step 2: Parse the pages and extract the data

Next, we'll open those files that we downloaded onto our computer and extract the data using the same logic we used in our previous scraper.

In [ ]:
```
# get a handle to all those files with the glob module
# https://docs.python.org/3/library/glob.html

files = glob.glob(f'{save_folder}/*.html')
```

In [ ]:
```
print(files)
```

Step 3: Parse the data and write to file

In this example, we're doing everything in one fell swoop:

- Open each HTML file I saved onto my computer
- Parse the HTML and target the data we want
- Write that data to file

In [ ]:
```
with open('qld-workplace-alerts-all.csv', 'w') as outfile:
    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)

    # now go through each file we downloaded
    for file in files:
        with open(file, 'r') as infile:
            html = infile.read()

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

        # use the find() method to isolate the unordered list (ul)
        # that contains the alerts
```

```
        results_list = soup.find('ul', {'class': 'search-results__grid'})

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

        # 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 attribu
            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: Write a function

So there's this concept in computer programming called DRY -- Don't Repeat Yourself -- and the general idea is, if you find yourself copying and pasting a bunch of the same code into different contexts, it might be time to extract that bit of logic into a reusable function.

Python comes with a bunch of handy built-in functions, like `print()`, but you can also define your own functions that take inputs and return outputs. You can check out some more explanation and examples in the Functions notebook in the `reference-notebooks` folder.

In this case, it might be useful to define a function that accepts a string of HTML from one of the pages of alert results as an input, parses that HTML with BeautifulSoup, extracts the target data and returns the list of data. Then you could *call* that function at any step in your script where you have a hunk of HTML you need parsed.

In [ ]: