Introduction

I find the best way for me to learn something new is to set myself a really challenging task. When learning webscraping with selenium I decided on the following:

Make a simple movie booking website for the two largest movie chains in the state of Victoria, Australia. The two largest chains are Village and Hoyts. You can find the website here:

https://stevenlmeyer.com/vicmovies/

Of these, scraping the Hoyts website proved to be by far the most challenging. Every movie showing in the Hoyts chain has its own web page. Every movie web page is a single page application constructed with Angular.

Here are some of the lessons I learned:

1. Forget all the elementary stuff. Wherever possible use "expected conditions". To find a list of expected conditions **click this**.

2. If you cannot use expected conditions use **implicit waits**.

3. Assume that when executing a long and complex set of set of selenium commands **there will be failures on the way** and embed all your code in "try except". The most common cause of a failure will be a "time out" and "element stale". This was especially the case when scraping the Hoyts website which is often quite slow.

4. But always set a maximum for the number of tries otherwise you may find your program going into an infinite loop!

The end result of my webscraping was a list of lists. Every list contained the following information:

Element Number		Element info
0		movie_key. Constructed from movie name and used to compare movies from different websites with different naming conventions. Also used as a sort key
1		movie_name
2		house_key. Constructed from name of cinema
3		house_name
4		chain. Either Hoyts or Village for now
5		sess_link. A link to the session on the movie chain website to enable booking
6		sess_id. (Village only)
7		house_id (Village only) Used in conjunction with sess_id to construct a URL for the booking page on Village
8		sess_year
9		sess_month
10		sess_day
11		sess_hour
12		sess_minute
13		sess_ymd .

14		sess_hm
15		sess_dt (session date time in python format)

The webscraping program runs overnight and generates a text file from the list of lists described above. It is uploaded to the website. The site itself is strictly minimal constructed using PHP, jQuery, Ajax, Bootstrap and SASS.

Snippets

Webscraping the Hoyts website proved to be the most challenging part of the project. I have condensed some of the most important lessons I learned in the following snippets of code. I shall repeat what I wrote earlier:

- Every movie on the Hoyts website has its own page.
- Every movie's page is a **single page application** using Angular to present data.

First we need to get the links (hrefs) for every movie page on the Hoyts website. The program, 010_get_hoyts_movie_links does this.

We find this at the webpage with the following url:

https://www.hoyts.com.au/movies

Here is the relevant HTML obtained by inspection in the Chrome browser

The element in the first, brown, rectangle is our starting point

The element in the second, red, element is our first target. We check that the text in this element is not blank. (In the example the text is "Ant-Man and the Wasp" so it is not blank)

If the element's text is not blank we go back up to the first (red) element using the "ancestor" feature of xpath

Our ultimate target is the third, blue, element. We want the value of the href attribute.

Here is the xpath we use:

```
xpath = '//div[contains(@class, "movie-list-item")]'
xpath+= '//div[contains(@class, "item-header-container")]'
# We need to check that the text containing the movie name is not blank
xpath+= '//span[contains(@class, "item-header") and text() != ""]'
# Now go back up the tree
xpath+= '//ancestor::div[contains(@class, "movie-list-item")]'
xpath+= '//div[contains(@class, "item-info-container")]//a'
```

And here is the selenium code we use:

elem_list = wait.until(EC.presence_of_all_elements_located((By.XPATH, xpath)))

EC was obtained from the following import statement:

from selenium.webdriver.support import expected_conditions as EC

By was imported as follows

from selenium.webdriver.common.by import By

"wait" is a two-step process:

from selenium.webdriver.support.ui import WebDriverWait

And then, in the body of the code:

wait = WebDriverWait(browser, 30)

Selenium will wait for up to 30 seconds to see whether the expected conditions are met. This is an implicit wait. If the expected condition, in this case "**presence_of_all_elements_located**" is met in, say, 1 second selenium will move on.

In an operational program you would probably not allow up to 30 seconds. Mostly I find 5 seconds works fine.

Next we iterate through the list of hrefs visiting the page of every movie. 020_iterate.py does this.

We extract the following information from each page

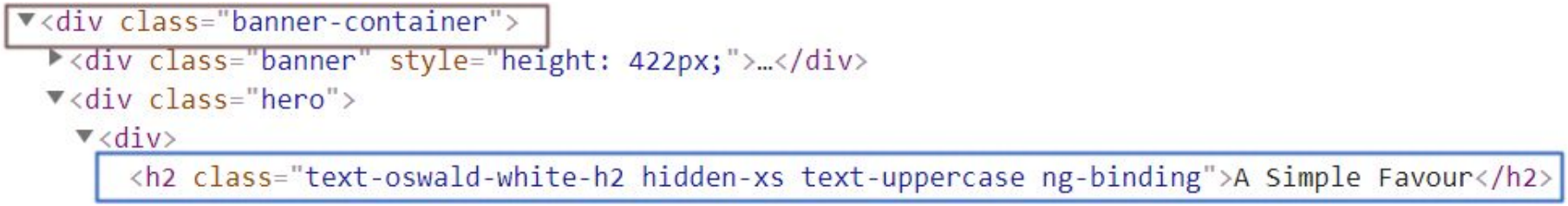
- Movie Name
- Name of every movie house showing the movie in Victoria. (There may be none. The movie may only be on show in other states)

Before proceeding I suggest you watch "iterate_the_movie.mp4" in this repository. It shows what you will see if you leave your browser visible while running 020_iterate.py. If you follow the movie you will see that the program failed a number of times and then recovered because the failure prone parts were inside a "try - except" block.

Step 1: Extract the movie name

Below we have an image of the relevant html obtained by inspecting the name in the Chrome browser

.



The first rectangle shows our starting point

The second the target element. We are interested in the text which, in this case, is "A Simple Favour"

This is a simple. There are no complex conditions so we use a CSS selector

Here is the CSS selector:

```
selector = 'div[class="banner-container"] '
selector+= 'div[class="hero"] '
selector+= 'h2[class*="text"]'
```

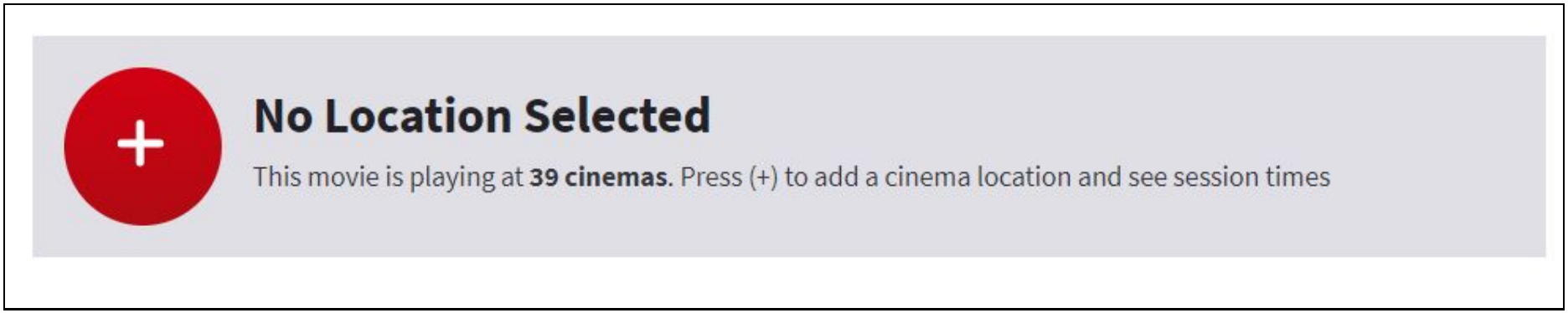
We extract the text from this element

Here is the selenium statement we use

elem = wait.until(EC.presence_of_element_located((By.CSS_SELECTOR, selector)))

Step 2

Next we have to expose the movie houses. To do this we must click the "+" button. (See image below)



Again we obtain the relevant html by inspection

```
▼<div class="container body-content no-padding-h">
    ::before
  ▼<div class="top no-padding-v movie-details">
    ▼<div class="playing-at-cinemas" ng-show="sessionVm.showPlayingCinemas">
      ▼<button class="add red playing-cinemas" data-toggle="modal" data-target="#cinemaModal">
        <i class="sprite-plus-white"></i>
```

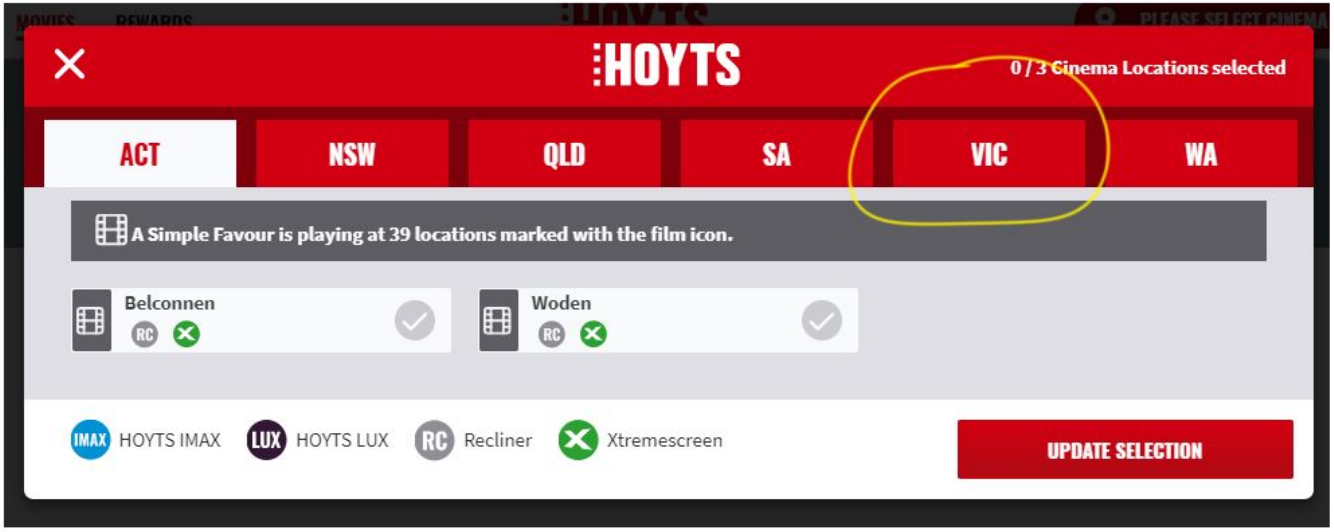
Again we see the starting and end points

Here is the xpath:
 	
#We will use this later to expose movie houses
xpath1 = '//div[contains(@class, "container") and contains(@class, "body-content")]'
xpath1+= '//div[contains(@class, "top")]'
xpath1+= '//button[contains(@class, "add") and contains(@class, "red")]'
xpath1+= '//i[contains(@class, "sprite")]'
 	
We click this element

Here is the selenium statement **elem = wait.until(EC.element_to_be_clickable((By.XPATH, xpath)))** Note the use of the condition "**element_to_be_clickable"**. This is a very useful condition

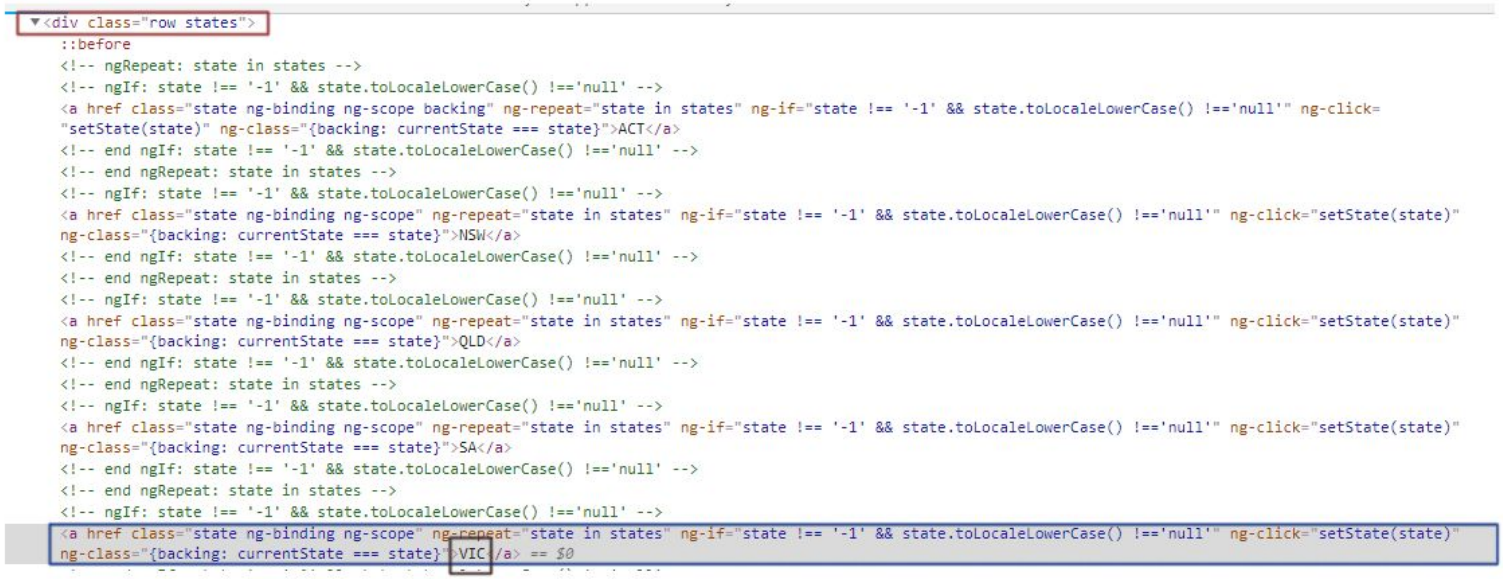
Step3:

Find the VIC movies



We need to find the circled one from the list states

Again, find relevant HTML by inspection



I could only fit this in by making the text small. However it is not difficult. We start with:

This contains a number of anchor elements, one for each state. The one we want is:

VIC

It has "VIC" in the text

We click this element exposing the VIC movie houses

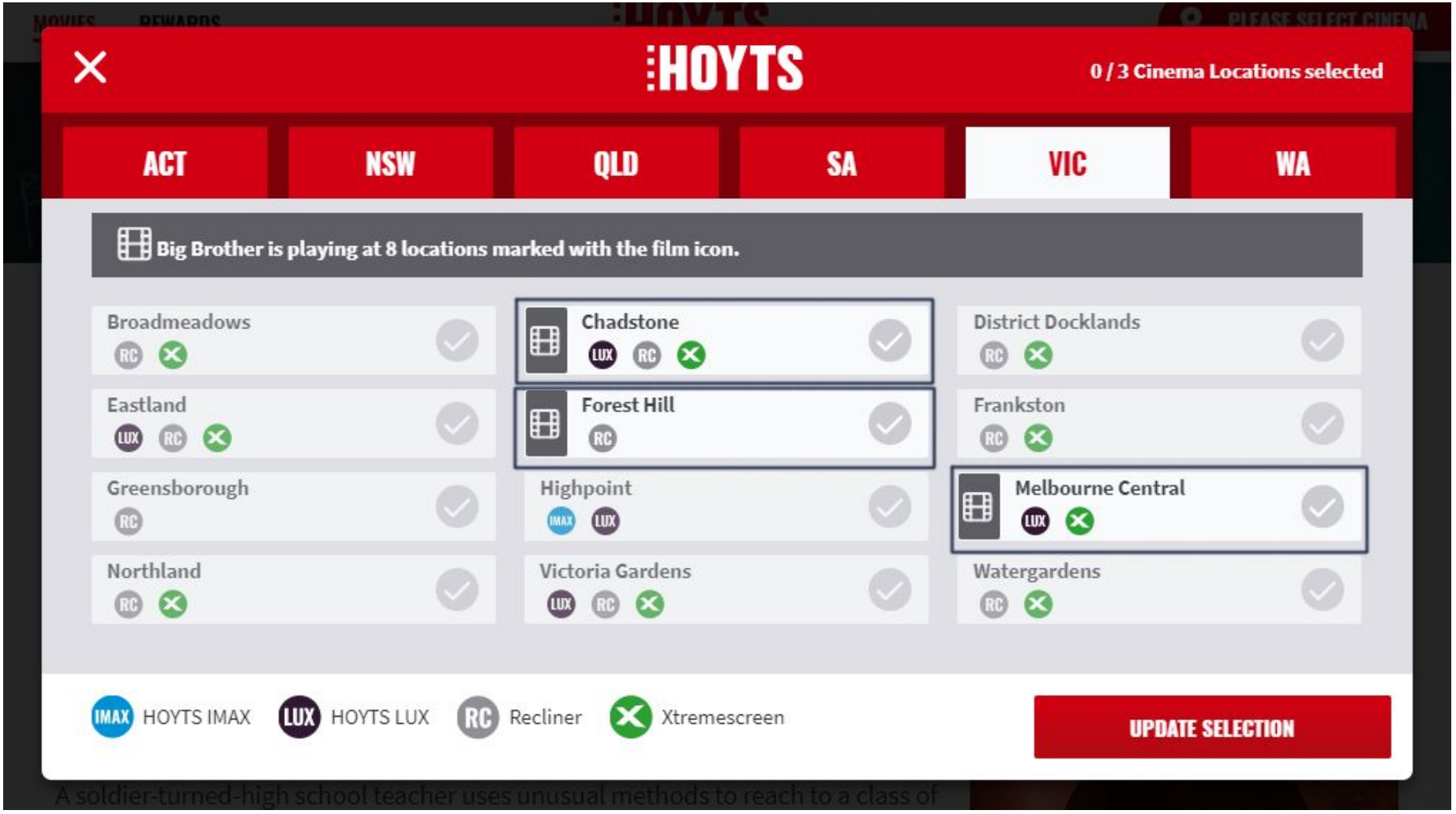
Here is the xpath:

#We will use this to expose VIC movie houses
xpath2 = '//div[contains(@class, "row") and contains(@class, "states")]'
xpath2+= '//a[contains(@class, "state") and text() = "VIC"]'

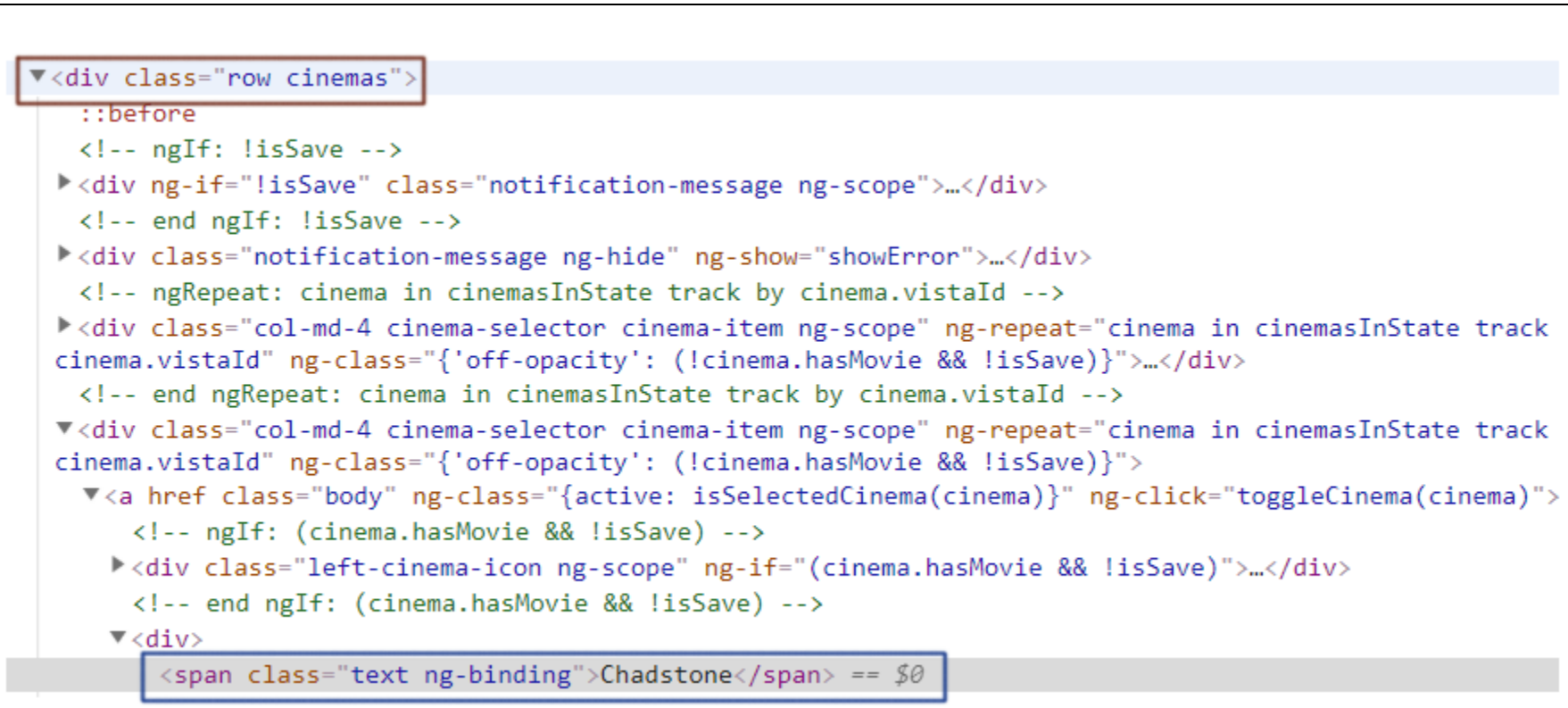
We click this element to expose the VIC movie houses

The selenium statement is, again:

elem = wait.until(EC.element_to_be_clickable((By.XPATH, xpath)))



This is a list of VIC movie houses for the movie Big Brother. The movie is only playing at the houses indicated with the rectangle.



The red rectangle is where we start.

We want the text of the element in the blue rectangle

The xpath

```
#We will use this to get list of VIC movie houses
#Where movie is playing (There may be none in VIC)
xpath3 = '//div[contains(@class, "row") and contains(@class, "cinemas")]'
#exclude elements where the class contains "off-opacity"
xpath3+= '//div[contains(@class, "cinema-selector") and not(contains(@class, "off-opacity"))]'
xpath3+= '//a//span[contains(@class, "text")]'
```

We add the text of the target elements to the list of movie houses for the current movie.
Note the use of:

and not(...)

Xpath is truly flexible

Here are the selenium statements:

This tells the browser to wait for up to "wait_time" seconds before throwing a time out.
browser.implicitly_wait(wait_time)

This is a simple browser statement:
elem_list = browser.find_elements_by_xpath(xpath)

Why not use "expected conditions" instead of an implicit wait?

Because this movie may not be showing in any movie houses in Victoria. In that case expected conditions throws a timeout error when all we want is an empty list.

Comments:

If the element of list of elements we want is simple to extract I use a CSS selector. It is more straightforward. Where complex conditions apply the flexibility of xpath makes this the logical choice

In order of preference I use:

- Expected conditions
- Implicit wait
- Explicit wait such as time.sleep (This is strictly a last resort. It can blow out execution times by hours)