78 Object identification

78.1 About

In order to automate a website, a script needs to be able to interact with the controls on the pages of the site. To do that, the script first needs to be able to locate the controls. This can be a difficult task, especially on a website that changes frequently- in fact, object identification is one of the biggest challenges of automated testing. Finding the best way to identify an object (or group of objects) often requires some experimentation and some knowledge of the application under test is required. Note there is often more than one way to identify an object. In that case, choose the one which is likely to be the most robust.

Selenium has a number of different ways to identify an object:

- By class name
- By CSS selector
- By id
- By link text
- By name
- By partial link text
- By tag name
- By XPath

Some of these may be more useful than others for different types of control, and each has their own advantages and disadvantages- for example, link text is very convenient for identifying hyperlinks, but may be less useful if the AUT has an option to switch languages, for example.

In this activity you will use some of these methods to identify objects on a page.

In these sections, use **time.sleep** to slow down the script execution so you can see clearly what is happening. Also comment out any code that is not relevant (remember in Python the **#** is used to mark comments).

78.2 Before you start

Have your editor open but close any scripts which you may have open.

78.3 Create a script

Create a new project in your editor called **seleniumObjectsProject** or similar.
Add the necessary import statement for Selenium:

```
from selenium import webdriver
```

Add the code to initialise the browser and go to the home page of the website.

```
wd = webdriver.Chrome()
wd.get("http://<AUT IP>/index.html")
```

Normally we would probably also add a line at the end of the test to close the browser (**wd.quit()**) but in this case it will be useful to keep the browser open to see how the code is working.

78.4 Link text

We will start by adding some code to click on links. This will allow us to navigate through pages.
Add code to open a page:

```
obj_page_link = wd.find_element(By.LINK_TEXT, "Page with a bit of text and a little picture")
obj_page_link.click()
```

Add code to return to the home page:

```
wd.find_element(By.LINK_TEXT, "Go back to index page").click()
```

Note that the syntax of these two lines is slightly different. The first stores the object into a variable before using it, the second identifies it and uses it on the same line.

It’s also possible to use just a part of the link text. Try changing the first link to the following:

```
obj_page_link = wd.find_element(By.PARTIAL_LINK_TEXT, "little picture")
```

Note that if there is more than one link with the same text, it is the first one found that is used. Replace **little picture** in the line above with just **picture** and run the script again to see what happens.

78.5 Name

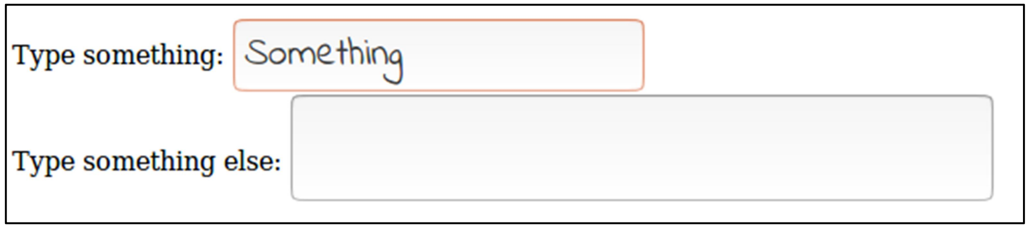
Elements can be identified by name.
First add some code to jump to a page that has some data entry fields:

```
wd.find_element(By.LINK_TEXT, "Page with some CSS applied").click()
```

Next, add a line to type something into one of the data fields.

```
wd.find_element(By.NAME, "datafield").send_keys("Something")
```

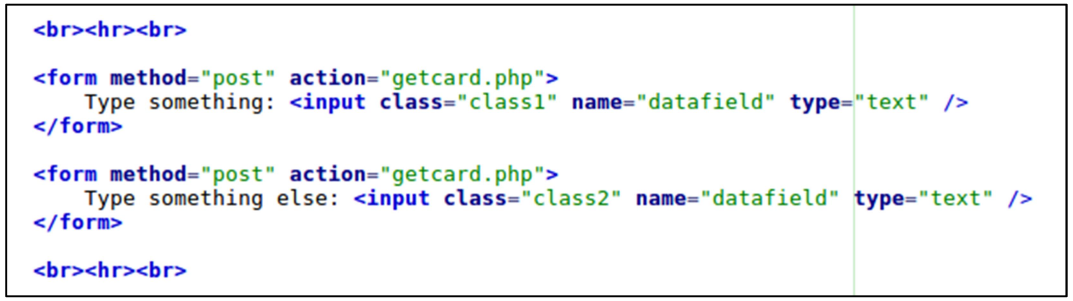
Run the script and make sure the text appears.


Type something: Something

Type something else:

78.6 Class names

Open a browser and navigate to the page with CSS styling. Right click and select **View Page Source** to see the source code of the page.
Find this section:

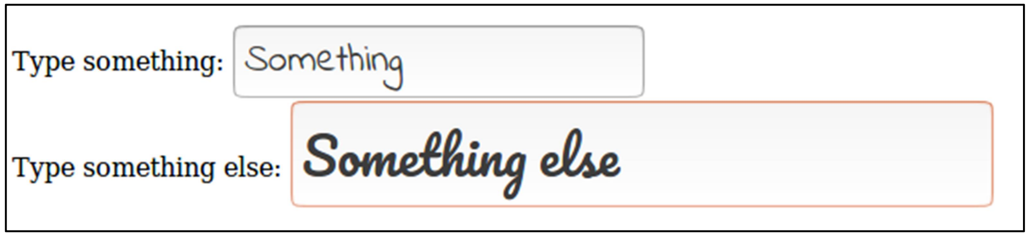


```
<br><hr><br>

<form method="post" action="getcard.php">
    Type something: <input class="class1" name="datafield" type="text" />
</form>

<form method="post" action="getcard.php">
    Type something else: <input class="class2" name="datafield" type="text" />
</form>

<br><hr><br>
```

Note that the two data entry fields have the same **name** which is **datafield**- but have different class names, **class1** and **class2**. If we use the name to identify the field then it is the first one that is used. If we want to type something into the second field, we have to use a different approach.
Add this line:

```
wd.find_element(By.CLASS_NAME, "class2").send_keys("Something else")
```

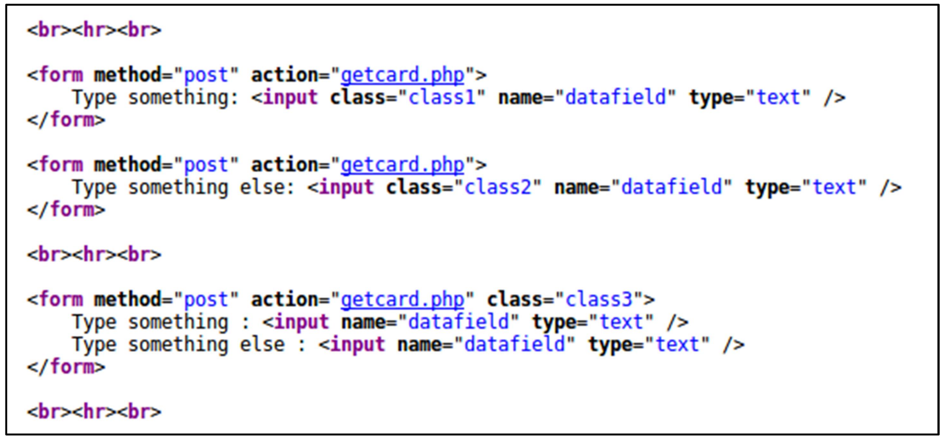
Run the script and ensure the data appears correctly:



78.7 Finding multiple elements

In the previous sections we used the **find_element…** methods to find individual controls. Sometimes, we may want to deal with several elements at the same time. For each of the **find_element…** methods, there is an equivalent **find_elements…** method. Because **find_elements…** can find several items, the method returns a **list** of **webelement**s.

Refer back to the source of the page that has CSS styling. There are actually four input fields, all with the same name:



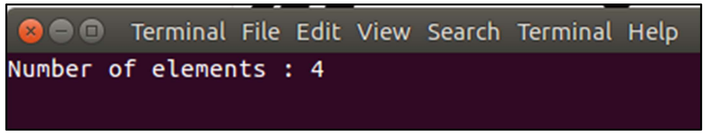
```
<br><hr><br>

<form method="post" action="getcard.php">
    Type something: <input class="class1" name="datafield" type="text" />
</form>

<form method="post" action="getcard.php">
    Type something else: <input class="class2" name="datafield" type="text" />
</form>

<br><hr><br>

<form method="post" action="getcard.php" class="class3">
    Type something : <input name="datafield" type="text" />
    Type something else : <input name="datafield" type="text" />
</form>

<br><hr><br>
```

Add the following line to get a list of all of the elements:

```
elements = wd.find_elements(By.NAME, "datafield")
number_of_elements = len(elements)
```

```
print("Number of elements : {}".format(number_of_elements))
```

Run at this stage and make sure you get the correct number of items:

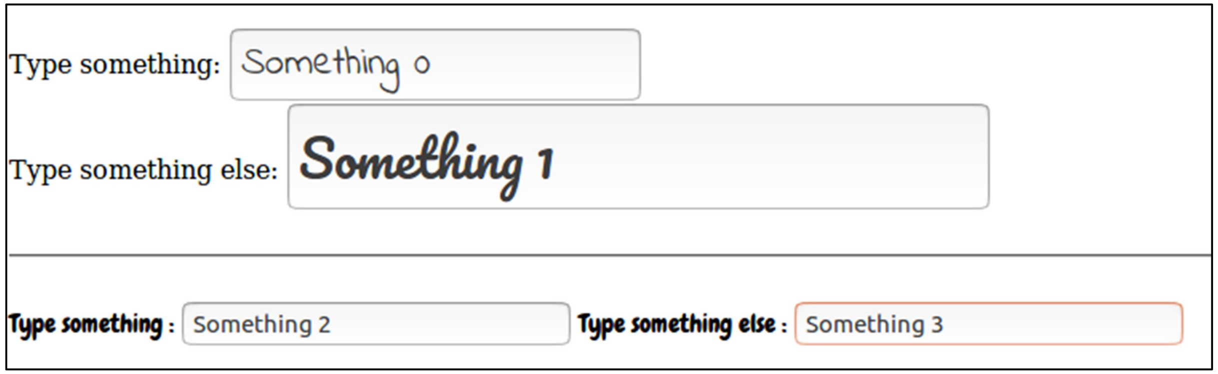


```
Number of elements : 4
```

Now we know how many elements there are, we can deal with them in order. Python has more than one way of iterating around an array. One way is shown here. Add these lines to the script:

```
i = 0
while(i < len(elements)):
    elements[i].send_keys("Something {}".format(i))
    i += 1
```

Run and check the output:



Type something: Something 0

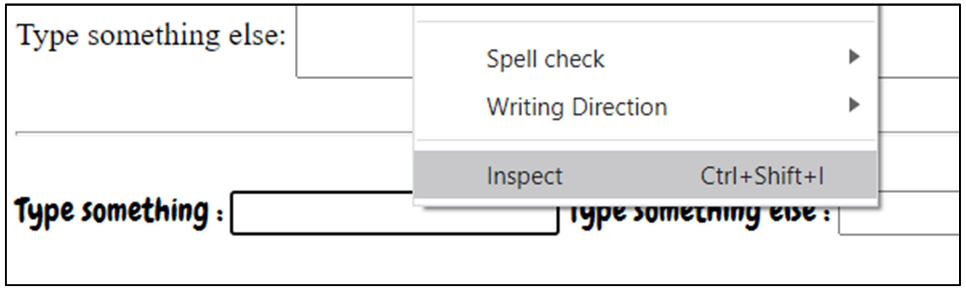
Type something else: Something 1

Type something : Something 2 Type something else : Something 3

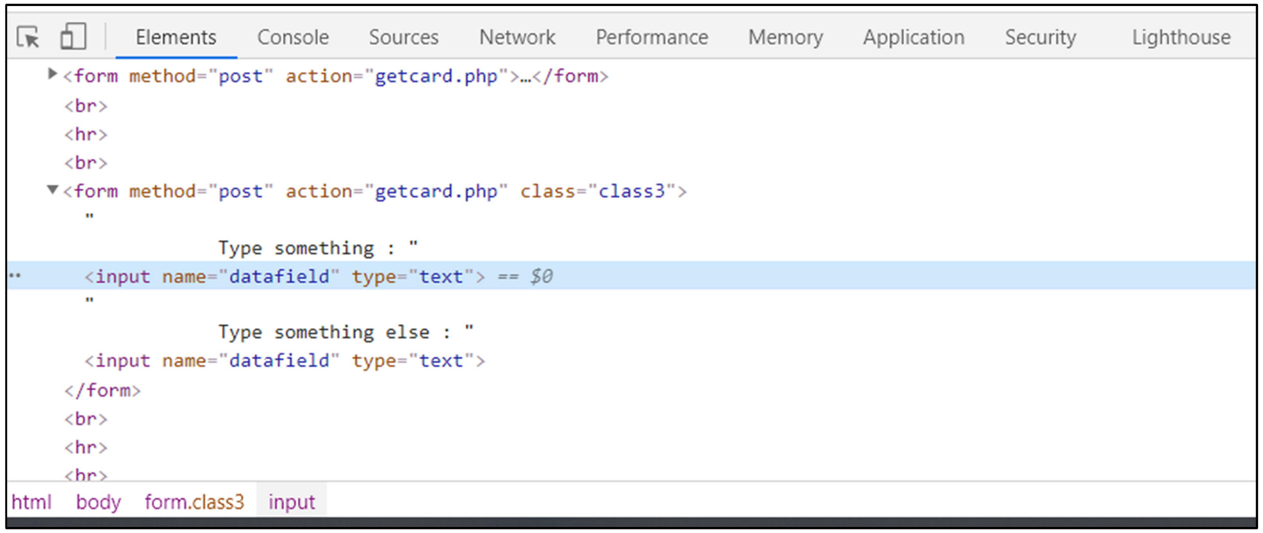
78.8 CSS selectors

CSS (Cascading Style Sheet) is the language used to apply styles to HTML web pages. CSS selectors are identifiers that are used by CSS (such as **class** or **id**) to identify items or sets of items for styling.
CSS selectors can be complex, but fortunately most browsers have built-in tools (or add-ins that can be installed) to help us.

Open Chrome and navigate to the page with CSS styling (**http://<AUT IP>/styled.html**). Right click in one of the edit fields and select **Inspect**.



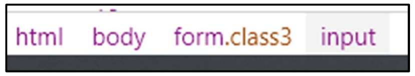
The Inspector screen opens.



```
▶<form method="post" action="getcard.php">…</form>
 <br>
 <hr>
 <br>
▼<form method="post" action="getcard.php" class="class3">
   "
            Type something : "
   <input name="datafield" type="text"> == $0
   "
            Type something else : "
   <input name="datafield" type="text">
 </form>
 <br>
 <hr>
 <br>
```

html body form.class3 input

The Inspector shows the HTML source of the page with the code relating to the currently-selected control highlighted. Move the mouse up and down the HTML file and observe that the selected object is highlighted in the main browser window.
The opposite is also possible- use the icon in the top left of the Inspector window and click an item on the screen to see the HTML highlighted in the Inspector window.
Select one of the two edit fields in the bottom part of the page. Examine the corresponding source code of the page and observe that the two fields both have the same **class** value, **class3**.
In the Inspector, note the path at the bottom of the window:

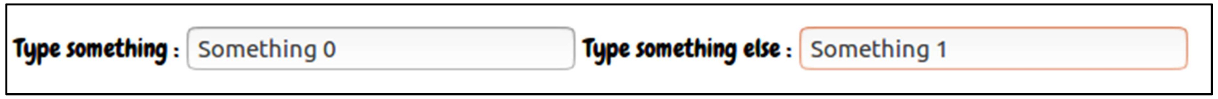


This path can be used directly as a CSS selector.

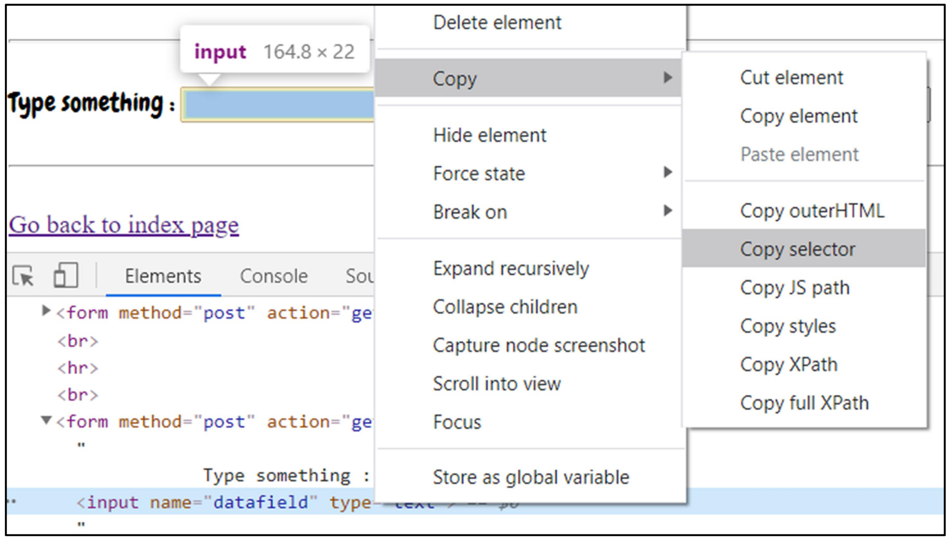
Go back to the script and replace the findElements line with this:

```
elements = wd.find_elements(By.CSS_SELECTOR, "html > body > form.class3 > input")
```

Run the script again and make sure only the two **class3** fields are used.



There is often more than one way to write a CSS selector to identify a particular object or objects. Try writing a different CSS selector that will identify the last two fields on the page- use the browser's inspector tool, right-click on an item and pick **Copy>Copy selector**:



```
▶<form method="post" action="ge
  <br>
  <hr>
  <br>
▼<form method="post" action="ge
   "
                Type something :
    <input name="datafield" type="text"> == $0
   "
```

The CSS selector can then be pasted directly into the script and used in a find_element(s) line in the script.

Note that CSS support varies slightly across browsers. This is one of the reasons we do cross-browser testing! When using CSS selectors in Selenium scripts, keep an eye out for tricks that the application developer may have used in order to make an application work correctly on different browsers.

78.9 XPath identifiers

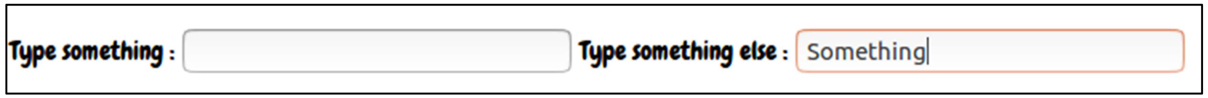
An XPath is a way of specifying a path to an element or elements in an XML file. Elements on HTML pages can also be identified with XPath. An XPath can be used to identify a single element or many elements. You can write XPaths manually, or you can use a tool to create them. The inspector tool in Chrome has an option to copy an XPath for an item.

Open the inspector and use it to select the second of the two edit fields at the bottom of the page with CSS styling. Right-click the item and select **Copy>XPath**.

Go back to the editor and paste the XPath into the script and use it to construct a new line in the scenario. Note that your XPath may be different from the one shown here.

```
wd.find_element(By.XPATH, "/html/body/form[3]/input[2]").send_keys("Something")
```

Run and make sure the correct field gets the text:



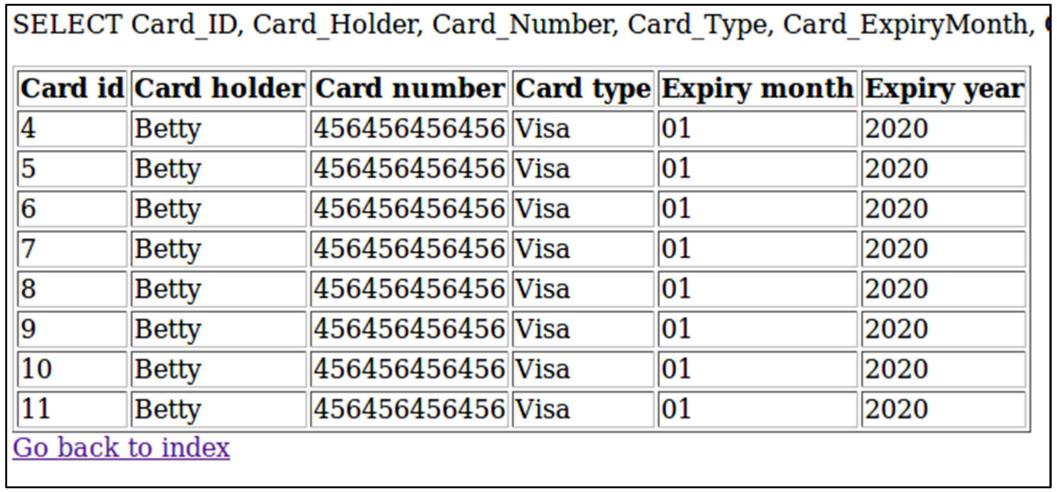
Experiment with **find_elements…** and XPath. Using the Comments website, open the homepage, the page with CSS styling, the page with text and a picture, and the page with several large pictures, and try the following XPaths to see what output you get:

```
//a/@href/
//*
//INPUT[@name='datafield']
//*[@class=’class2’]
//a/img
//img
//p
//p[3]
```

78.10 Tables

Many websites use HTML tables. Pages that have a list of account transactions, or items in a basket, will present that information in a table. In order to automate such pages successfully, we need some mechanism to investigate a table and discover the data it contains.

The demo website in the training environment includes some tables. Open a browser and navigate to **http://<AUT IP>/getcard.html** and enter Betty as the cardholder. The page returned will look similar to this:



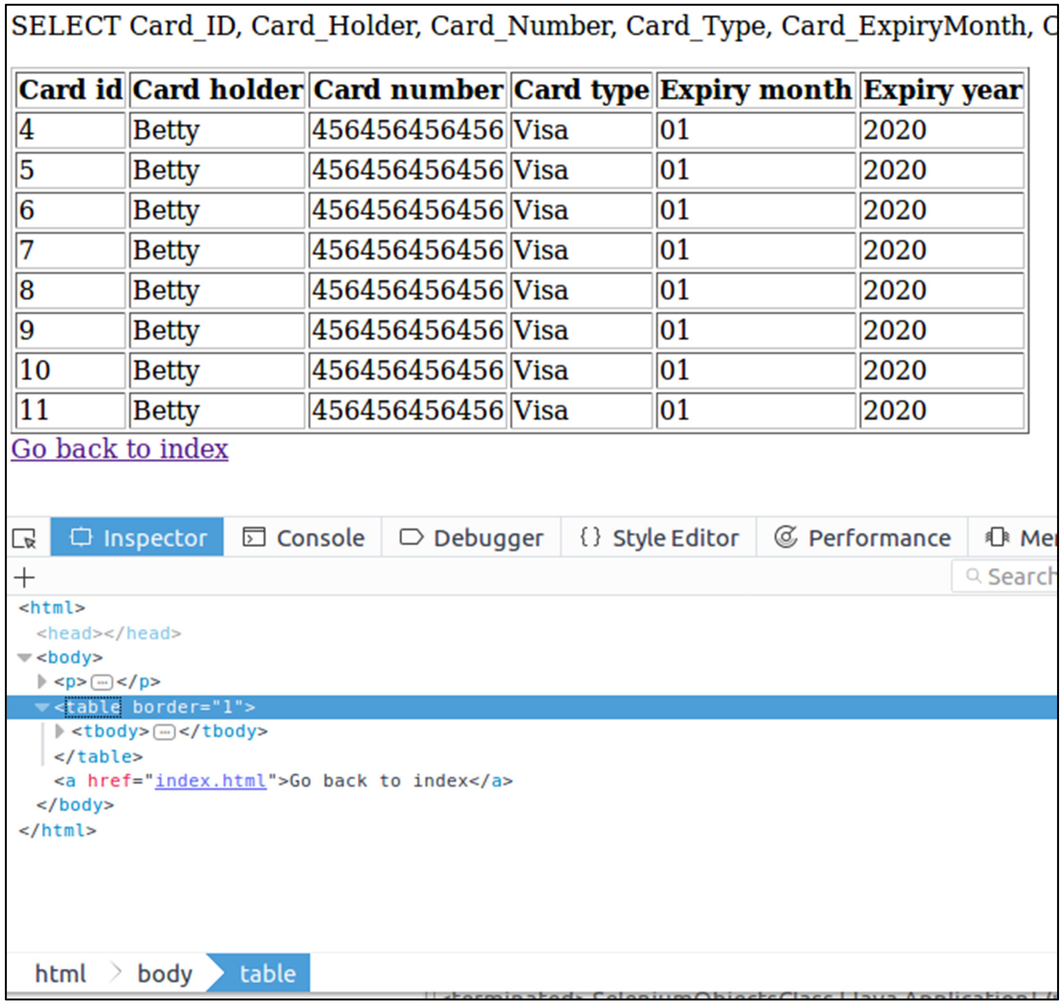
SELECT Card_ID, Card_Holder, Card_Number, Card_Type, Card_ExpiryMonth,

Card id	Card holder	Card number	Card type	Expiry month	Expiry year
4	Betty	456456456456	Visa	01	2020
5	Betty	456456456456	Visa	01	2020
6	Betty	456456456456	Visa	01	2020
7	Betty	456456456456	Visa	01	2020
8	Betty	456456456456	Visa	01	2020
9	Betty	456456456456	Visa	01	2020
10	Betty	456456456456	Visa	01	2020
11	Betty	456456456456	Visa	01	2020

Go back to index

Right click on one of table cells and select **Inspect** and use the Inspector to examine the table.

As with all types of element, we can use various ways to identify a table (name, id, CSS selector, XPath and so on, as we saw previously). In this case, the table is unnamed, but since there is only one table on the page, it has a fairly simple CSS selector, visible at the bottom of the screen: **html > body > table**.



SELECT Card_ID, Card_Holder, Card_Number, Card_Type, Card_ExpiryMonth, C

Card id	Card holder	Card number	Card type	Expiry month	Expiry year
4	Betty	456456456456	Visa	01	2020
5	Betty	456456456456	Visa	01	2020
6	Betty	456456456456	Visa	01	2020
7	Betty	456456456456	Visa	01	2020
8	Betty	456456456456	Visa	01	2020
9	Betty	456456456456	Visa	01	2020
10	Betty	456456456456	Visa	01	2020
11	Betty	456456456456	Visa	01	2020

Go back to index

```
<html>
  <head></head>
  <body>
    <p>⋯</p>
    <table border="1">
      <tbody>⋯</tbody>
    </table>
    <a href="index.html">Go back to index</a>
  </body>
</html>
```

In the script, add a line that navigates to the card query page and enters a cardholder name.

```
wd.get(“http://<AUT IP>/getcard.html”)
wd.find_element(By.NAME, "cardholder").send_keys("Betty")
wd.find_element(By.NAME, "cardholder").submit()
```

Add a line that gets the table and stores a reference to it in a variable:

```
table = wd.find_element(By.CSS_SELECTOR, "html > body > table")
```

Now we can find out how many rows are in the table. Again, how we do this will depend on the structure of the page. In our case, in the source code, rows are identified by the tag **<tr>**, so we'll count the elements with that tag using the **find_elements** method we saw previously.

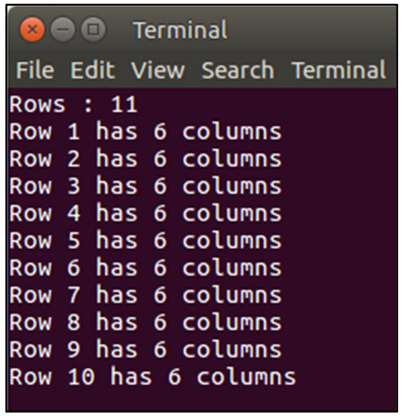
```
rows = table.find_elements(By.XPATH, "//tr")
print("Number of rows : {}".format(len(rows)))
```

Save and run at this point and ensure you get the correct output. Note that in this case, the table headings ("Card id", "Card holder" and so on) count as the first row of the table- the table in the screenshot above would return a value of 9. We can omit the first row (which will be row 0) from the rest of the tasks.

In an HTML table each row can have a different number of columns. To discover the number of columns, we will loop around each row. Columns are identified by the **<td>** tag.

```
i = 1
while i < len(rows):
    columns = rows[i].find_elements(By.XPATH, "td")
    print("Row {} has {} columns".format(i, len(columns)))
    i += 1
```

Run the script and ensure you get an output similar to this:



```
Rows : 11
Row 1 has 6 columns
Row 2 has 6 columns
Row 3 has 6 columns
Row 4 has 6 columns
Row 5 has 6 columns
Row 6 has 6 columns
Row 7 has 6 columns
Row 8 has 6 columns
Row 9 has 6 columns
Row 10 has 6 columns
```

To get the data from a cell, add another loop inside the first (to make a nested loop) directly under the code to get the number of columns. To get the text, we can use the **text** property (which every Element has). Add this code to write out the value of every cell to the terminal window:

```
j = 0
while j < len(columns):
    cell_data = columns[j].text
    print("{}, {} : {}".format(i, j, cell_data))
    j += 1
```

Note that the first column is also numbered 0.

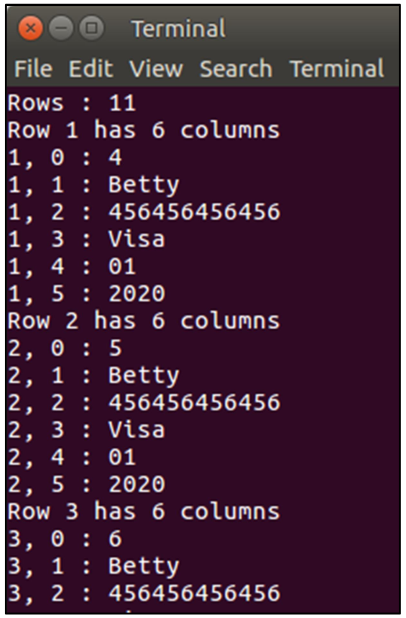
78.10.1 Checkpoint:

At this stage the code should look similar to this:

```
import time
from selenium import webdriver
from selenium.webdriver.common.by import By
```

```
wd = webdriver.Chrome()

# go to the cards query page and enter a query
wd.get("http://<AUT IP>/getcard.html")
wd.find_element(By.NAME, "cardholder").send_keys("Betty")
wd.find_element(By.NAME, "cardholder").submit()

# pause while the page loads
time.sleep(2)

# get a reference to the table
table = wd.find_element(By.CSS_SELECTOR, "html > body > table")

# find the number of rows in the table
rows = table.find_elements(By.XPATH, "//tr")
print("Number of rows : {}".format(len(rows)))

# loop round each row
i = 1
while i < len(rows):
    # count the columns on this row
    columns = rows[i].find_elements(By.XPATH, "td")
    print("Row {} has {} columns".format(i, len(columns)))

    # loop round each column
    j = 0
    while j < len(columns):
        cell_data = columns[j].text
        print("{}, {} : {}".format(i, j, cell_data))
        j += 1

    i += 1
```

Run and check that the output is correct:



```
Rows : 11
Row 1 has 6 columns
1, 0 : 4
1, 1 : Betty
1, 2 : 456456456456
1, 3 : Visa
1, 4 : 01
1, 5 : 2020
Row 2 has 6 columns
2, 0 : 5
2, 1 : Betty
2, 2 : 456456456456
2, 3 : Visa
2, 4 : 01
2, 5 : 2020
Row 3 has 6 columns
3, 0 : 6
3, 1 : Betty
3, 2 : 456456456456
```

Verify the code further by adding a new card to the database and running the script again to make sure the new row is included in the output.

78.11 Challenge

So far in this section we have simply displayed the data from the table in the terminal window. Normally when we are examining a table in a script like this we will do it because we want to verify something. Modify the code so that it produces a message that indicates if a card has expired.

78.12 Explore more

78.12.1 Randomise

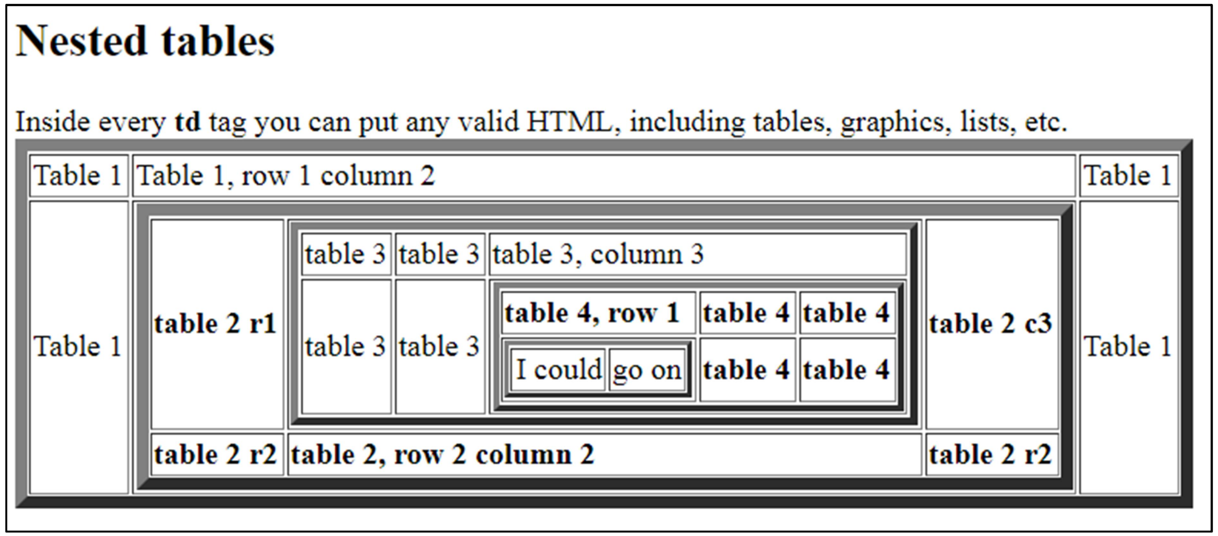
In this section we saw the **find_elements** methods for finding all the items on a page that match a description, and how to iterate and use each one in turn. Modify the code so that it just selects one of the input fields at random.

78.12.2 Other methods of using a list

In this section we used a simple **while** loop to iterate through the array of entry fields. There are other methods of using a list. Modify the code so that it uses a **for** loop instead.

78.12.3 More tables

HTML tables can be complex. They can have complex formatting or be nested inside each other. Open the “table sampler” page https://hissa.nist.gov/~black/tableQuikRef.html and find the example of the nested table:



Nested tables

Inside every **td** tag you can put any valid HTML, including tables, graphics, lists, etc.

Table 1	Table 1, row 1 column 2	Table 1
Table 1		Table 1

table 2 r1		**table 2 c3**
table 2 r2	**table 2, row 2 column 2**	**table 2 r2**

table 3	table 3	table 3, column 3
table 3	table 3	

table 4, row 1	**table 4**	**table 4**
	table 4	**table 4**

I could	go on

How would you get the values of the cells that contain “I could”, “go on” inside a script?

78.12.4 DIVs

Generally, as you might expect, tables are good for displaying data that is appropriate to be displayed in a tabular format. Some web designers also use them for controlling the layout of a page, but that is not generally good design practice. It’s more normal for a web designer to use a framework such as Bootstrap (http://getbootstrap.com) that allows them to divide the page up and control the flow of the different areas of the page, which is important for “mobile first” page layouts.
Have a look at the sample layouts at http://getbootstrap.com/examples/grid/. Examine them with the browser Inspector. How would you get the information from one of the grid columns? Or all the grid columns?

78.12.5 Radio buttons and checkboxes

There are many different kinds of controls on web pages. We have seen some such as links, data entry fields and tables. Radio buttons and checkboxes are also very common.
Often one of these controls can be found using any of the methods shown previously, for example by name or CSS selector, but sometimes it can be a little more challenging. To select a checkbox or radio button, use the **click()** method.

This is an example of using a CSS selector to select a radio button. In this case, the radio group name is **crust** and we are looking for the item with the value **thin**.

```
wd.find_element(By.CSS_SELECTOR, "input[name='crust'][value='thin']").click()
```

Alternatively, we can find all the items in the group, then examine each item in the group in turn until we find the one we want. The **get_attribute** method can be used to get the value of any HTML attribute.

```
elements = wd.find_elements(By.NAME, "crust")
i = 0
while(i < len(elements)):
    if(elements[i].get_attribute("value") == "deep"):
        elements[i].click()
    i += 1
```

The same syntax can be used to find and select checkboxes.

One of the pages on the demo website has some radio buttons and checkboxes. Write a script that starts on the home page and navigates to that page, then orders a large stuffed crust pizza with onions and mushrooms.