Selenium
for everyone

FIRST EDITION – Release 0.1

Kevin Thomas

Forward

It was Christmas morning, 1988, I opened the largest box under the tree and low and behold I saw a Commodore 64 personal computer!

I frantically ripped the box apart to get at the C-64 console and hijacked the main living room television to hook it up. I turned it on and saw for the first time a light blue screen and a blinking prompt which showed a READY and a blinking cursor.

I was able to acquire a small handful of games but I was obsessed with connecting with other people in the world that I can communicate with outside of the, "solo game experience".

I got my hands on a 300 baud modem and was able to get it hooked up and connected to Quantum Link which was a U.S. and Canadian online service for the Commodore 64 and 128 personal computers that operated starting November 5, 1985. In October 1989 the service was renamed to "America Online" and the service was made available to users of PC systems in addition to Commodore users.

I thought, "what if I can have my own online system". This was well before the days of the Internet. I found a piece of software called DMBBS 4.8 which was a Commodore 64 BBS or bulletin board system that allowed me to create my own online experience that others could log into. I found a BBS or bulletin board system called, "The Boozers Place" which checked all the boxes of what I was looking for.

I reached out to the owner or SYSOP of, "The Boozer's Place", and he showed me how I could create my own BBS which I called, "THE ALLNIGHTER". I got it set up and launched it within a day.

Many friends would scoff and say, "only geeks use these computers and this is a fad, this will die out like every other fad in the past". Time proved them starkly incorrect.

As we fast forward to today we find ourselves not only with one or more household computers but multiple cell phones, tablets, work and personal computers along with a ton of microcontrollers which have shaped our entire existence.

Today's software is not like the software of yesterday. Unlike the days of old where you had one application that had a very limited feature scope, we ever increasingly divorce ourselves from the monolithic architecture and gravitating toward the microservices architecture.

In modern software, our interface is quite different to where it once was with either a command-line or even native OS GUI user experience to that of a web-centric user experience.

Despite all of the careful unit testing that occurs for each of these back-end components no matter how big or small the product, we find an increasing need for a comprehensive integration testing framework to which we have a solution with Selenium.

If we think about any modern app such as YouTube we take for granted all of the complex back-end services that exist which connect up to the web interface. No matter how comprehensive the individual micro service components are designed and unit tested they alone exist within only their narrow focus of their respective scope.

In other words you could have multiple databases tied up to a single web interface. Let's examine one single instance of a database to which there will be code that communicates with the that single database instance and then connects to yet another database and/or other segments of code. Each would represent a micro service which could live in a Docker container or Kubernetes pod which then communicates with each other possibly using some message broker like Kafka or ZMQ to which you find yourself in an extremely complex integration.

The need for Automation Engineering and a comprehensive Automation Testing Framework has never been more real as we continue to find ourselves with larger and larger integrated micro service products.

Think of it this way, if a single component which was designed well and was properly unit tested which stands alone provides no real way of working properly when integrated into a larger system.

Think of a single unit test as a brand new house where it is just you and your one of three dogs. You walk into the house, with dog 1 and you walk in and out of each room. You test the dogs reaction and see how she functions in this new environment.

After your traversal throughout the house you find that the dog responds positively and all is well.

You repeat this step with dog 2 and three with the same result and you feel confident that when you integrate all three dogs all will be well!

Imagine if those dogs never met or interacted with each other before? This is much like individual software components.

You now are confident that everything is perfect and you bring in the entire family and the three dogs and BOOM everything blows up and you scratch your head thinking what could have went wrong?

This simple analogy demonstrates the increasing demand of good integration testing with Automation and the most robust solution exists within a Java Selenium environment.

This text will be a journey where we start by setting up our development environment on Windows, as MAC will be an almost identical setup, and creating our first automation where we automate a Google web search for coffee.

We then spend time learning about how information is captured from the web interface such that we could automate it by uniquely identifying individual locators that can be used in an Automation Framework to properly test integration and/or scrape data on a custom web site designed solely for this course working with XPATH.

We then move into some Java basics and develop a simple Test Automation to automate this web site and then move on to developing a more comprehensive object-oriented Automation Framework which can scale in a production environment.

This course will be a comprehensive solution to learn and develop a sophisticated Automation Framework which could be used in a Test Automation production environment. I am excited to start this journey with you! Let's get started!

Table Of Contents

Chapter 1: Getting Started

How often do you use the web? How often do your friends, family, coworkers and really anyone you come into contact with?

Quite simply everything we do interfaces with the web in some way and will continue to do so going forward. Each website is a unique product to which the importance of testing the integration of all of the moving parts of each site is critical to ensure a cohesive user experience.

When you hear the words, "Front-End Automation", we typically refer to Automated Testing or web automation. This is a topic we will be covering in this course however web automation is much more than testing the integration of a site as it also provides an ability to literally automate any possible interaction between a human being and a website.

Understanding web automation can help you also to scrape data from literally any source and then perform basic data science on that data which could be fed into a machine-learning model.

The, "why", has so many answers in this case and taking the time to learn web automation will give you an in-demand set of skills which could help you obtain a career as a SEiT (Software Engineer in Test).

In this course we are going to use the Chrome and Firefox browser and focus on Selenium which is a free (open-source) automated testing framework which will provide us the ability to achieve everything we have discussed thus far.

Let's first download Eclipse which will be our integrated development environment.

https://www.eclipse.org/downloads/

Once installed, let's create a new Maven project. What is Maven you say? Maven means accumulator of knowledge in Yiddish and developed by Apache has become one of the most powerful software project management tools available today. It is built on the concept of a POM or project object model. Maven is also open-source.

Let's open up Eclipse and get started by following the below steps.

```
File
New
Maven Project
CHECK Create a simple project (skip archetype selection)
CHECK Use default Workspace location
Next
Group Id: Selenium Framework
Artifact Id: Selenium Framework
Finish

OPEN pom.xml
Dependencies

(VISIT https://mvnrepository.com)
(SEARCH Selenium Java)
(CLICK latest version i.e. 4.0.0-rc-1)
Add
Group Id: org.seleniumhq.selenium
Artifact Id: selenium-java
Version: 4.0.0-rc-1
OK

(VISIT https://mvnrepository.com)
(SEARCH Junit Jupiter API)
(CLICK latest version i.e. 5.8.0)
Add
Group Id: org.junit.jupiter
Artifact Id: junit-jupiter-api
Version: 5.8.0
OK

RT CLICK /src/test/java
New
File
BrowserTest.java
Finish

RT CLICK SeleniumFramework
New
Folder
drivers
Finish

RT CLICK drivers
New
Folder
chromedriver
Finish

RT CLICK drivers
New
Folder
geckodriver
Finish
```

(VISIT https://sites.google.com/chromium.org/driver/downloads)
(Current Releases – click on link to your current Chrome version)
(Extract chromedriver.exe)
DRAG AND DROP /drivers/chromedriver
CLICK Copy files
OK

(VISIT https://github.com/mozilla/geckodriver/releases)
(Assets – click latest version i.e. geckodriver-v0.30.0-win64.zip)
(Extract geckodriver.exe)
DRAG AND DROP /drivers/geckodriver
CLICK Copy files
OK

Now let's populate our **BrowserTest.java** file which will open a Chrome browser, navigate to *google.com* and wait 3 seconds and close.

```java
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.firefox.FirefoxDriver;

public class BrowserTest {
    public static void main(String[] args) throws InterruptedException {
        String projectPath = System.getProperty("user.dir");
        System.setProperty("webdriver.chrome.driver",
projectPath+"//drivers/chromedriver/chromedriver.exe");
        // System.setProperty("webdriver.gecko.driver",
projectPath+"//drivers/geckodriver/geckodriver.exe");

        WebDriver driver = new ChromeDriver();
        // WebDriver driver = new FirefoxDriver();

        driver.get("https://google.com");
        Thread.sleep(3000);
        driver.close();
    }
}
```

CTRL+F11

Congratulations! You just created your first Automation code in Java! Time for cake!

Let's take a moment and review the official Selenium docs.

(VISIT https://www.selenium.dev/documentation)

We will use these docs to create our entire framework step-by-step.

Chapter 2: Basic Automation

Now that we have our setup complete, let's dive into some basic automation with our own custom site. Let's begin by copying the Site directory within our GitHub repo https://github.com/mytechnotalent/Selenium-For-Everyone to our Desktop.

Assuming you have Python3 installed, navigate to your Site folder on your Desktop within a terminal and type the following.

python -m http.server

Open your Chrome browser and navigate to the following.

http://localhost:8000

Now we have a simple web server and site available for local testing.

Let's right-click on our search box and click inspect.

```
<input autocomplete="on" class="form-control search" name="q" placeholder="Search in google.com" required="required" type="text">
```

Here we see a number of identifiers. The one we want to target is the *name* as it has a unique value of *q*.

Let's open our **BrowserTest.java** file and add the following code and run.

```java
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.firefox.FirefoxDriver;

public class BrowserTest {
    public static void main(String[] args) throws InterruptedException {
        String projectPath = System.getProperty("user.dir");
        System.setProperty("webdriver.chrome.driver",
projectPath+"//drivers/chromedriver/chromedriver.exe");
        // System.setProperty("webdriver.gecko.driver",
projectPath+"//drivers/geckodriver/geckodriver.exe");

        WebDriver driver = new ChromeDriver();
        // WebDriver driver = new FirefoxDriver();

        driver.get("http://localhost:8000");

        WebElement textBox = driver.findElement(By.name("q"));

        textBox.sendKeys("coffee");

        Thread.sleep(3000);

        driver.close();
    }
}
```

CTRL+F11

Here we see the word *coffee* being typed into our text box and then it waits 3 seconds and closes.

Within Java we have the *WebElement* class which represents an HTML element. Generally, all interesting operations to do with interacting with a page will be performed through this interface.

As a professional SEiT, the MOST IMPORTANT skill you will develop will be your ability to read and parse the official Selenium documentation.

(VISIT https://www.selenium.dev/documentation/webdriver/locating_elements)

Turn your attention to the Locating one element area at the top of the page.

10

We learn here how to find elements on a page. We have a number of built-in selector types as illustrated in the docs.

We simply instantiate the *WebElement* class with the *textBox* object by calling the *findElement* method and passing the param *By.name("q")*.

We can call methods on the textBox object. Let's refer to the docs.

(VISIT https://www.selenium.dev/documentation/webdriver/keyboard)

We see the *sendKeys* method which types a key sequence in the respective DOM element. We simply type `textBox.sendKeys("coffee");` this will handle our text input.

Our next step is to programmatically press the *Search* button.

```
<button class="button" type="submit">Search</button>
```

Here we create a searchBtn object and call the By.className method as we have a class called button `WebElement searchBtn = driver.findElement(By.className("button"));` this will setup our object to be called.

We then call the click method on the searchBtn object `searchBtn.click();` this will click our button.

The entire code block is as follows.

```java
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.firefox.FirefoxDriver;

public class BrowserTest {
    public static void main(String[] args) throws InterruptedException {
        String projectPath = System.getProperty("user.dir");
        System.setProperty("webdriver.chrome.driver",
projectPath+"//drivers/chromedriver/chromedriver.exe");
        // System.setProperty("webdriver.gecko.driver",
projectPath+"//drivers/geckodriver/geckodriver.exe");

        WebDriver driver = new ChromeDriver();
        // WebDriver driver = new FirefoxDriver();

        driver.get("http://localhost:8000");

        WebElement textBox = driver.findElement(By.name("q"));

        textBox.sendKeys("coffee");

        WebElement searchBtn = driver.findElement(By.className("button"));

        searchBtn.click();

        Thread.sleep(3000);

        driver.quit();
    }
}
```

Congrats! We successfully took our first steps into some basic interaction through automation!

Chapter 3: Reverse Engineering The DOM

Take a moment and think about everything you do on the web today.
Imagine having the ability to automate that.

Our goal for this course is to have a professional proficiency in
comprehensive web automation. Web Automation can literally reproduce
anything a human can achieve working within a web browser.

Web Automation is now used in the Automation career field to which
the Software Engineer in Test professionals write Automation code to
properly test the advanced integration between components in their
respective product application which is a large distributed system.

In addition, Web Automation can also capture data. Web data consists
of everything including obtaining product and price information from
any number of web sites to capturing tweets and other social media
data to anything that literally exists on the internet.

The trick about capturing web data is the fact that web sites change
literally by the minute. Each update on a web site will cause web
code to change constantly.

In this chapter we will not be doing any hard coding but instead
completely reverse engineering the Document Object Model or DOM in
our web application.

In Web Automation, XPath is used to find an a unique element, or
unique piece of information, on a web page.

XPath, XML Path Language, is the language for finding ANY element on
a web page using the HTML DOM (Document Object Model) structure.

XPath contains the path of the element situated on the web page.
Standard syntax for creating XPath is as follows.

```
XPath=//tagname[@attribute='value']
```

The first is, Absolute XPath, which is the direct way to find the
element. The crippling disadvantage of the absolute XPath is that if
there are any changes made to web code, the XPath will fail. This
happends several times an hour on a large web site.

The second, Relative XPath, allows you to uniquely target parts of
the web site easily which will be more dependable over time. If the
relative XPath fails you can easily fix this.

To learn how to use XPath properly, I have designed a custom web site
we will be learning on that we used in the prior lesson. I first ask
that you use the Chrome browser during this course and in particular
here for consistency and ease of the course material.

The ONLY way we can automate is to uniquely identify an element which
means that when we craft our XPath code it targets only 1 unique
element at a time. This will become more obvious as we begin. Just
keep this firmly in mind as this is the single most important concept
of this lesson.

Navigate to your Site folder on your Desktop within a terminal and
type the following.

python -m http.server

Open your Chrome browser and navigate to the following.

http://localhost:8000

Let's begin by visiting our site.

STEP 1: Open Chrome's Developer Tool

Press F12 (fn F12 on MAC)

STEP 2: Enable Chrome's Developer XPath Search Tool

Press CTRL F (Command F on MAC)
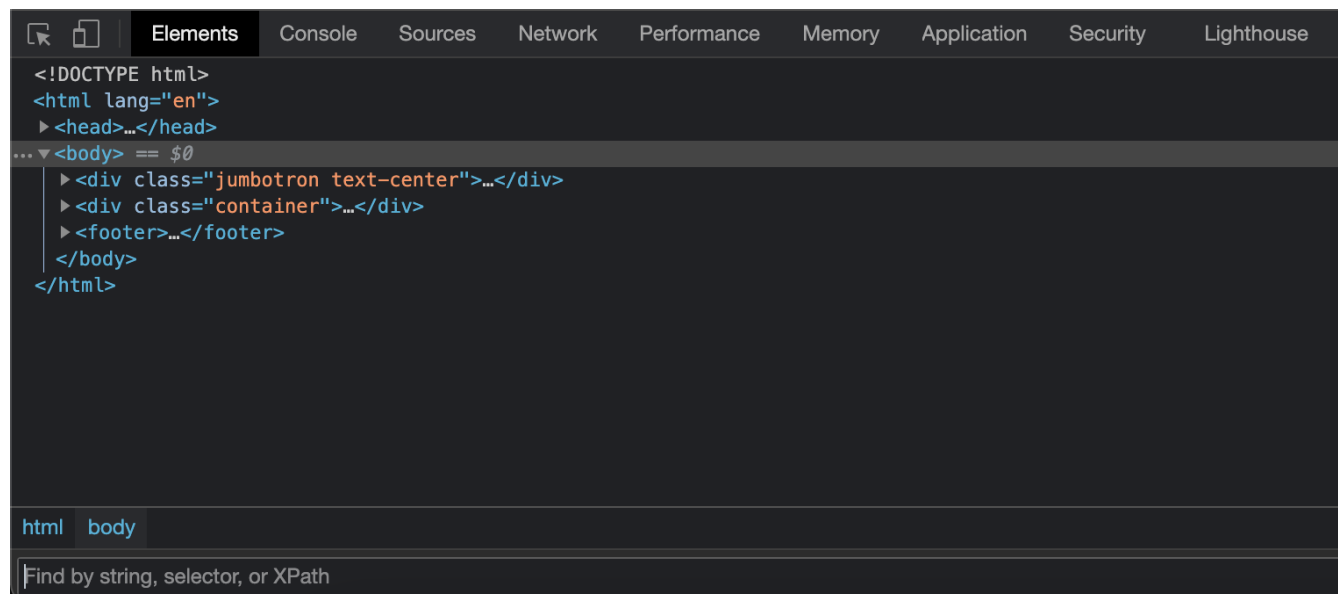
You will see a, *Find by string*, *selector*, or *XPath*, window to which you will type in all of your XPath's.

Let's begin our journey by working with the following tools.

Identify Element Using Attribute

Web Code

```
<p class="first">First Element</p>
<p>
<div class="first">Second Element</div>
</p>
```

To uniquely identify the *First Element*, we start with the // meaning search the entire site where there is a *p* tag that has a class attribute = *"first"*.

XPath Code

```
//p[@class="first"]
```
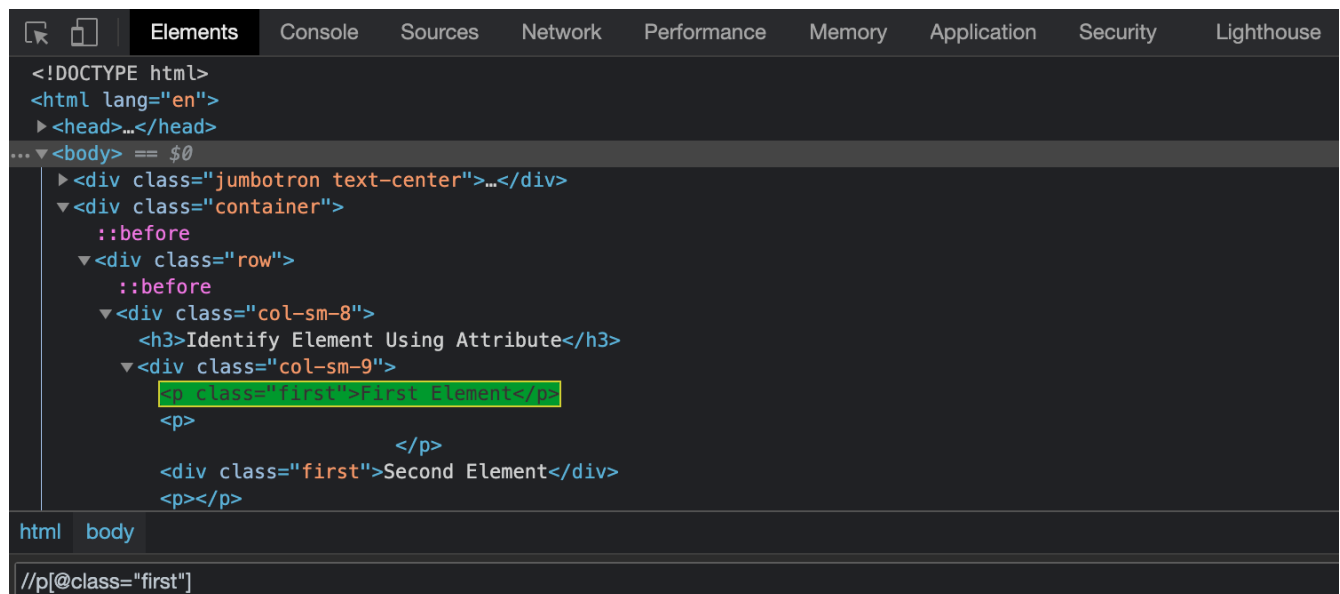

Hover your mouse over the green box and watch it uniquely identify your element in the web site above.

The MOST IMPORTANT think when we identify an element is to ensure it is unique. Located at the bottom right hand side of the **Chrome Developer XPath Search Tool** window you will see the following.

I wanted to mention this now as you will be using this with literally every XPath going forward in this lesson. If you see, *1 of 1*, then we have uniquely identified the element.

To uniquely identify the *Second Element*, we start with the *//* where there is a *div* tag that has a *class* attribute = *"first"*.

XPath Code

```
//div[@class="first"]
```
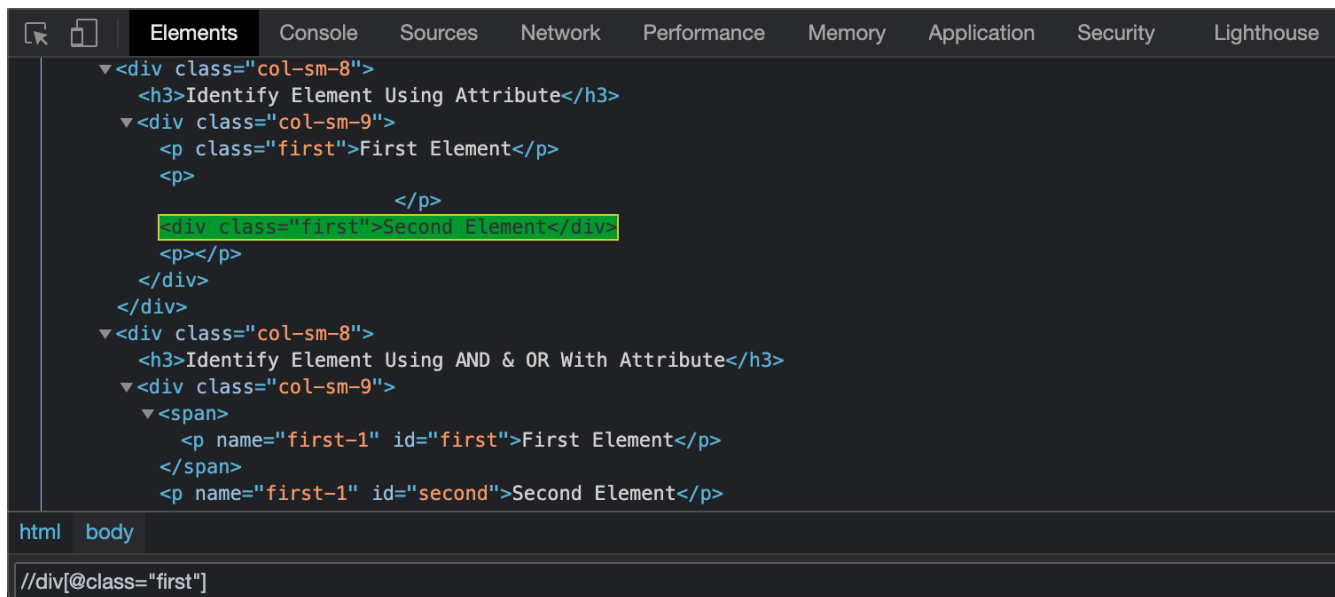

Verify 1 of 1 and hover your mouse over the green box and watch it uniquely identify your element in the web site above.

Identify Element Using AND & OR With Attribute

Web Code

```
<span>
    <p name="first-1" id="first">First Element</p>
</span>
<p name="first-1" id="second">Second Element</p>
<p name="second-1" id="first">Third Element</p>
<p name="second-1" id="second">Fourth Element</p>
```

16

To uniquely identify the *First Element*, we start with the // where there is a *p* tag that has a *name* attribute = *"first-1"* and an *id* attribute = *"first"*.

You can alternatively identify the, *First Element*, and start with the // where the *span* tag has a child *p* tag and has a name attribute = *"first-1"* or has a name attribute = *"second-1"* and an *id* attribute = *"first"*.

XPath Code

```
//p[@name="first-1" and @id="first"]
//span/p[(@name="first-1" or @name="second-1") and @id="first"]
```

```
//span/p[(@name="first-1" or @name="second-1") and @id="first"]
```

Verify 1 of 1 and hover your mouse over the green box and watch it
uniquely identify your element in the web site above.

To uniquely identify the *Second Element*, we start with the // where
there is a *p* tag that has a *name* attribute = *"first-1"* and an *id*
attribute = *"second"*.

XPath Code

```
//p[@name="first-1" and @id="second"]
```

Verify 1 of 1 and hover your mouse over the green box and watch it uniquely identify your element in the web site above.

To uniquely identify the *Third Element*, we start with the // where there is a *p* tag that has a *name* attribute = *"second-1"* and an *id* attribute = *"first"*.

XPath Code

```
//p[@name="second-1" and @id="first"]
```



```
//p[@name="second-1" and @id="first"]
```

Verify 1 of 1 and hover your mouse over the green box and watch it uniquely identify your element in the web site above.

To uniquely identify the, *Fourth Element*, we start with the // where there is a *p* tag has a *name* attribute = *"second-1"* and an *id* attribute = *"second"*.

XPath Code

```
//p[@name="second-1" and @id="second"]
```

```
        <p></p>
      </div>
    </div>
  ▼<div class="col-sm-8">
      <h3>Identify Element Using AND & OR With Attribute</h3>
    ▼<div class="col-sm-9">
      ▼<span>
          <p name="first-1" id="first">First Element</p>
        </span>
        <p name="first-1" id="second">Second Element</p>
        <p name="second-1" id="first">Third Element</p>
        <p name="second-1" id="second">Fourth Element</p>
      </div>
    </div>
  ▶<div class="col-sm-8">…</div>
  ▶<div class="col-sm-8">…</div>
  ▶<div class="col-sm-8">…</div>
```
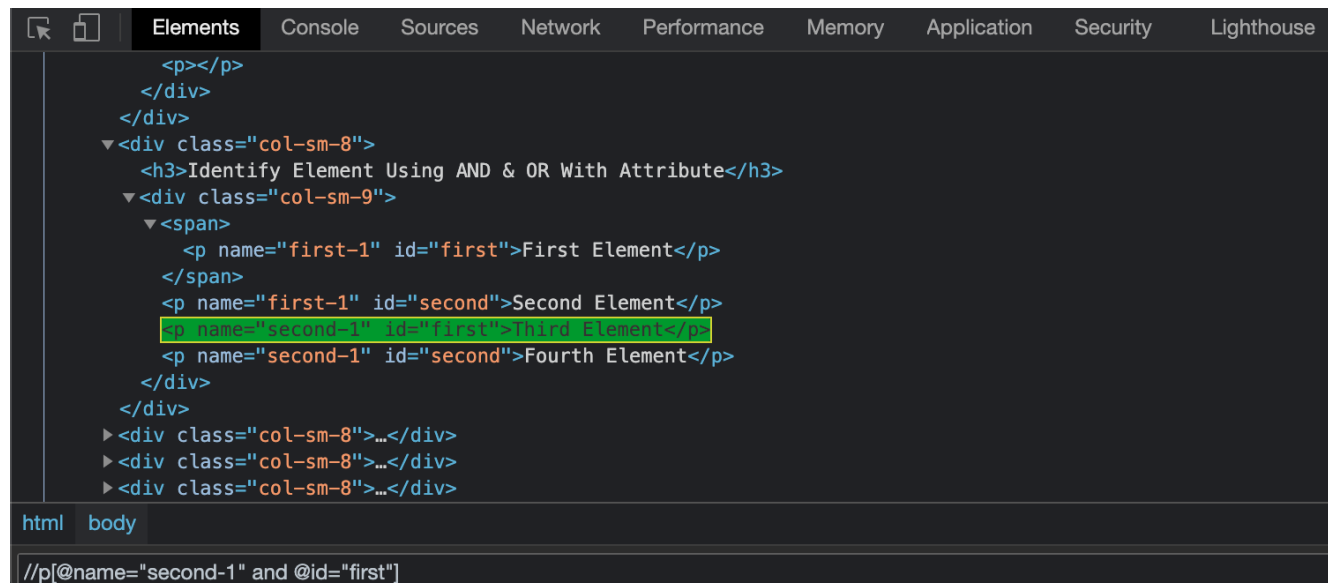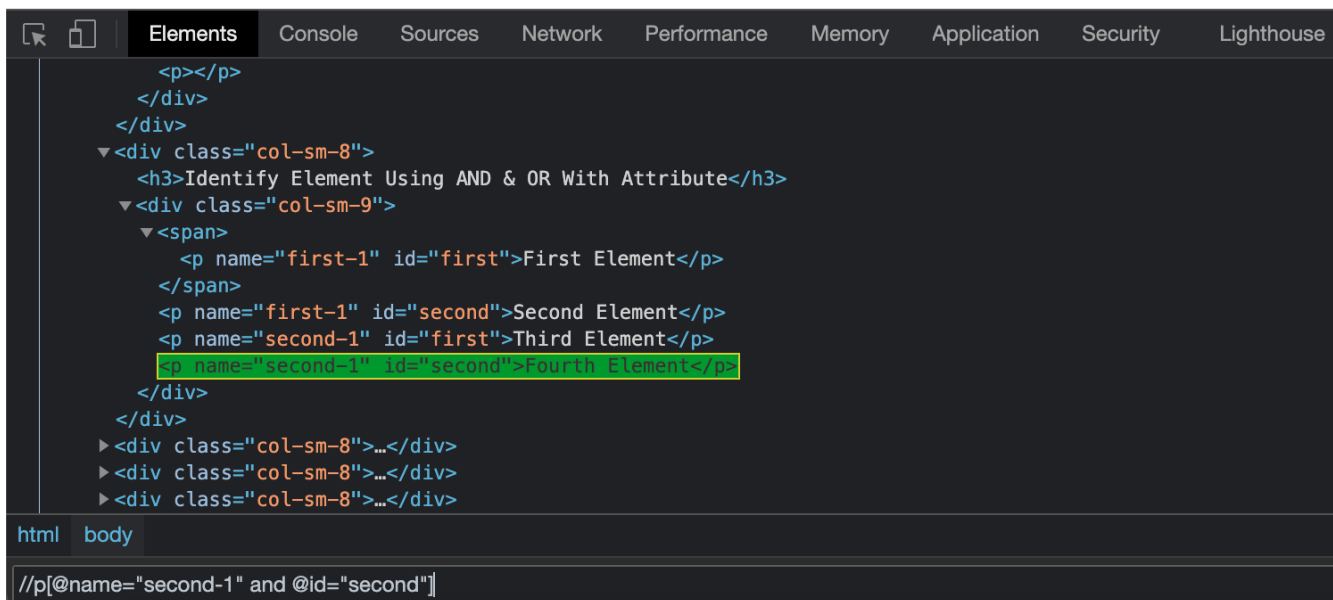
html   body

//p[@name="second-1" and @id="second"]

Verify 1 of 1 and hover your mouse over the green box and watch it uniquely identify your element in the web site above.

Identify Element Using text() Function With Attribute

Web Code

```
<div class="col-sm-9">
    <p name="first" id="first">First Element Text</p>
</div>
```

To uniquely identify the *First Element*, we start with the // where there is a *p* tag that has a *text* function searching for the text, *"First Element Text"*.

You can alternatively identify the *First Element*, starting with the // where the *p* tag has a *text* function searching for the text, *"First Element Text"*, and a *name* attribute = *"first"*.

You can alternatively identify the *First Element*, starting with the // where the *p* has a *text* function searching for the text, *"First Element Text"*, and a *name* attribute = *"first"* or id attribute = *"first"*.

XPath Code

```
//p[text()="First Element Text"]
//p[text()="First Element Text" and @name="first"]
//p[text()="First Element Text" and (@name="first" or @id="first")]
```

```
        <p name="first-1" id="second">Second Element</p>
        <p name="second-1" id="first">Third Element</p>
        <p name="second-1" id="second">Fourth Element</p>
      </div>
    </div>
  ▼<div class="col-sm-8">
      <h3>Identify Element Using text() Function With Attribute</h3>
    ▼<div class="col-sm-9">
        <p name="first" id="first">First Element Text</p>
      </div>
    </div>
  ▶<div class="col-sm-8">…</div>
  ▶<div class="col-sm-8">…</div>
  ▶<div class="col-sm-8">…</div>
  ▶<div class="col-sm-8">…</div>
  ▶<div class="col-sm-8">…</div>
  ▶<div class="col-sm-8">…</div>
```

html body

//p[text()="First Element Text"]

```
        <p name="first-1" id="second">Second Element</p>
        <p name="second-1" id="first">Third Element</p>
        <p name="second-1" id="second">Fourth Element</p>
      </div>
    </div>
  ▼<div class="col-sm-8">
      <h3>Identify Element Using text() Function With Attribute</h3>
    ▼<div class="col-sm-9">
        <p name="first" id="first">First Element Text</p>
      </div>
    </div>
  ▶<div class="col-sm-8">…</div>
  ▶<div class="col-sm-8">…</div>
  ▶<div class="col-sm-8">…</div>
  ▶<div class="col-sm-8">…</div>
  ▶<div class="col-sm-8">…</div>
  ▶<div class="col-sm-8">…</div>
```

html body

//p[text()="First Element Text" and @name="first"]

```
  Elements    Console    Sources    Network    Performance    Memory    Application    Security    Lighthouse

        <p name="first-1" id="second">Second Element</p>
        <p name="second-1" id="first">Third Element</p>
        <p name="second-1" id="second">Fourth Element</p>
      </div>
    </div>
  ▼<div class="col-sm-8">
      <h3>Identify Element Using text() Function With Attribute</h3>
    ▼<div class="col-sm-9">
        <p name="first" id="first">First Element Text</p>
      </div>
    </div>
  ▶<div class="col-sm-8">…</div>
  ▶<div class="col-sm-8">…</div>
  ▶<div class="col-sm-8">…</div>
  ▶<div class="col-sm-8">…</div>
  ▶<div class="col-sm-8">…</div>
  ▶<div class="col-sm-8">…</div>

html  body

//p[text()="First Element Text" and (@name="first" or @id="first")]
```

Verify 1 of 1 and hover your mouse over the green box and watch it uniquely identify your element in the web site above.

Identify Element Using ancestor With Child Attribute

Web Code

```
<div id="first-a">
    <div id="first-a-c">
        <div id="first-a-gc">
            <p>First Element</p>
        </div>
    </div>
</div>
```
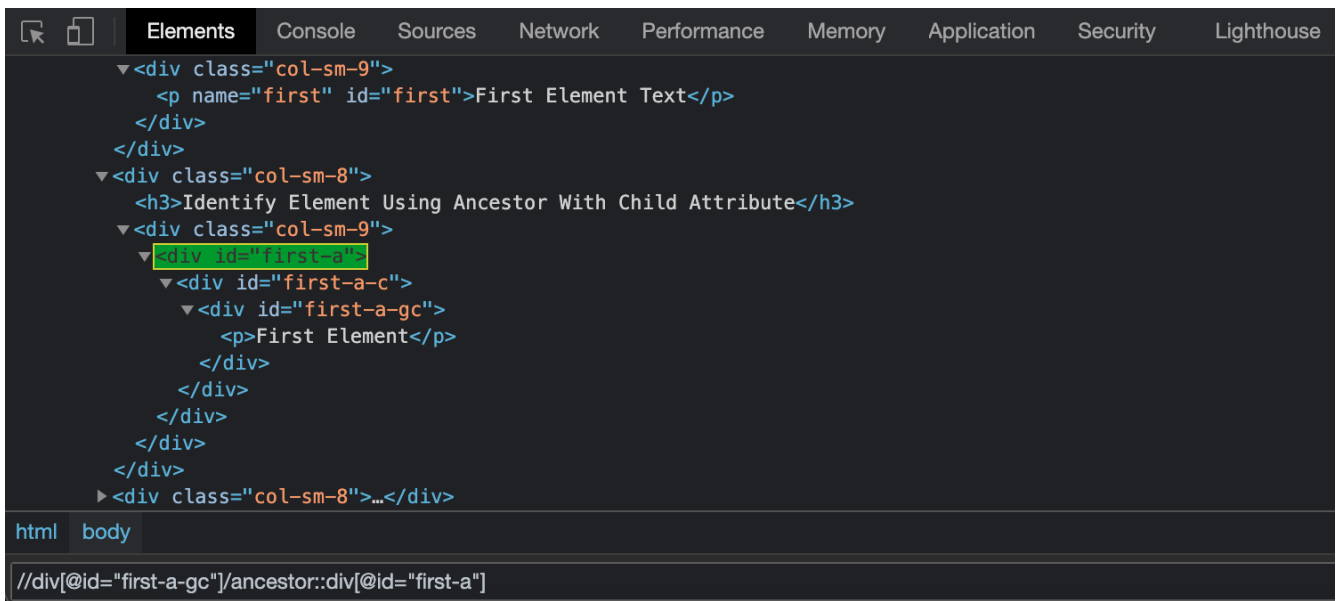
To uniquely identify the div tag that has an id attribute = *"first-a"*, we start with the // where there is a *div* tag that has an *id* attribute = *"first-a-gc"* and an *ancestor* attribute and a *div* tag with an *id* attribute = *"first-a"*.

*** NOTE ***

This is the first time we are not uniquely identifying text within a *p* tag or attribute. This example is designed to show you how to uniquely identify an ancestor if you needed to obtain this unique element in your automation. This particular example is not practical but it is very relevant for your understanding if you ever needed to do such.

XPath Code

```
//div[@id="first-a-gc"]/ancestor::div[@id="first-a"]
```


Verify 1 of 1 and hover your mouse over the green box and watch it uniquely identify your element in the web site above.

Identify Element Using descendent With Parent Attribute
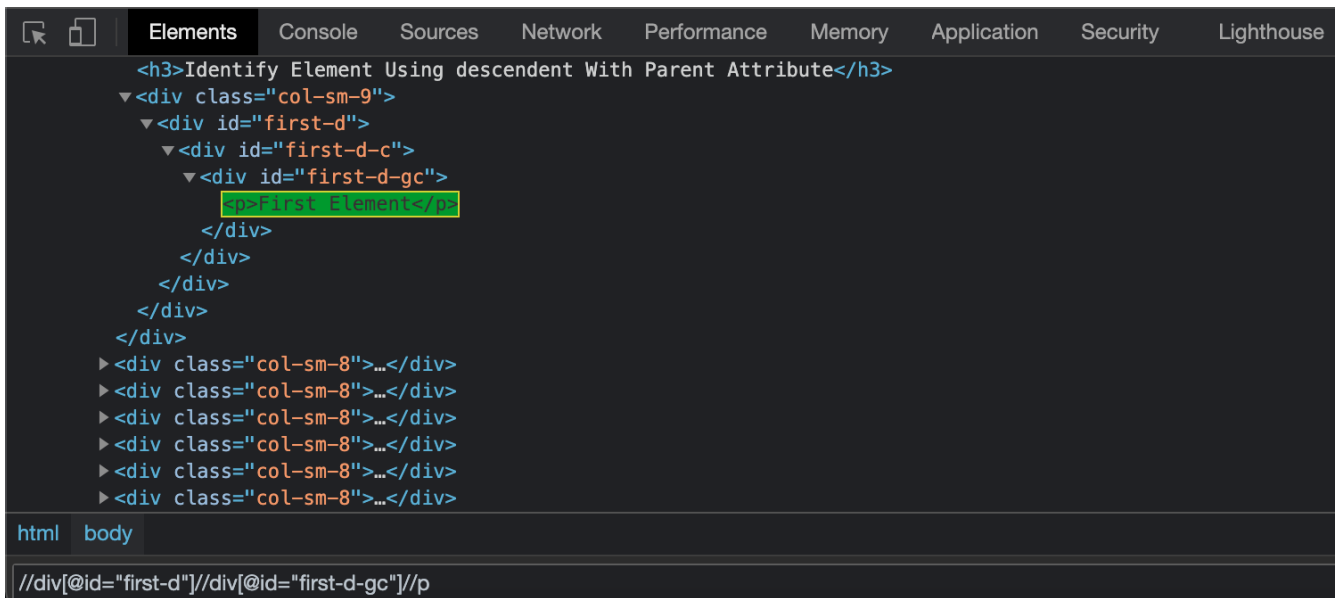
Web Code

```
<div id="first-d">
    <div id="first-d-c">
        <div id="first-d-gc">
            <p>First Element</p>
        </div>
    </div>
</div>
```

To uniquely identify the *First Element*, we start with the // where there is a *div* tag that has an *id* attribute = *"first=d"* and *div* tag that has an *id* attribute = *"first-d-gc"* and a *p* tag.

XPath Code

```
//div[@id="first-d"]//div[@id="first-d-gc"]//p
```

```
<h3>Identify Element Using descendent With Parent Attribute</h3>
▼<div class="col-sm-9">
  ▼<div id="first-d">
    ▼<div id="first-d-c">
      ▼<div id="first-d-gc">
          <p>First Element</p>
        </div>
      </div>
    </div>
  </div>
  ▶<div class="col-sm-8">…</div>
  ▶<div class="col-sm-8">…</div>
  ▶<div class="col-sm-8">…</div>
  ▶<div class="col-sm-8">…</div>
  ▶<div class="col-sm-8">…</div>
  ▶<div class="col-sm-8">…</div>
```
html body

//div[@id="first-d"]//div[@id="first-d-gc"]//p

Verify 1 of 1 and hover your mouse over the green box and watch it uniquely identify your element in the web site above.

Identify Element Using parent With Child Attribute

Web Code

```
<div id="first-p">
    <div id="first-c">
        <p>First Element</p>
    </div>
</div>
```

To uniquely identify the div tag that has an id attribute = *"first-p"*, we start with the // where there is a *div* tag that has has an *id* attribute = *"first-c"* and a *parent* attribute and a *div* tag.

You can alternatively identify the *div* tag that has an *id* attribute = *"first-p"*, starting with the // where there is a *div* tag that has an *id* attribute = *"first-c"* and .. (representing the parent directory).
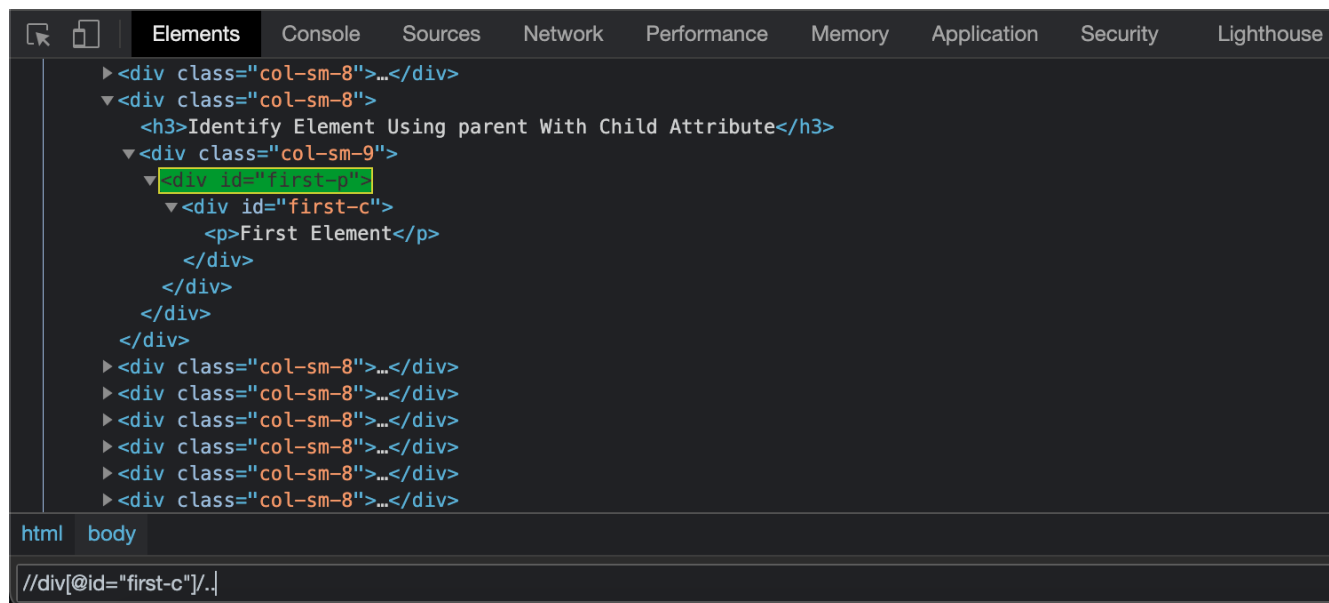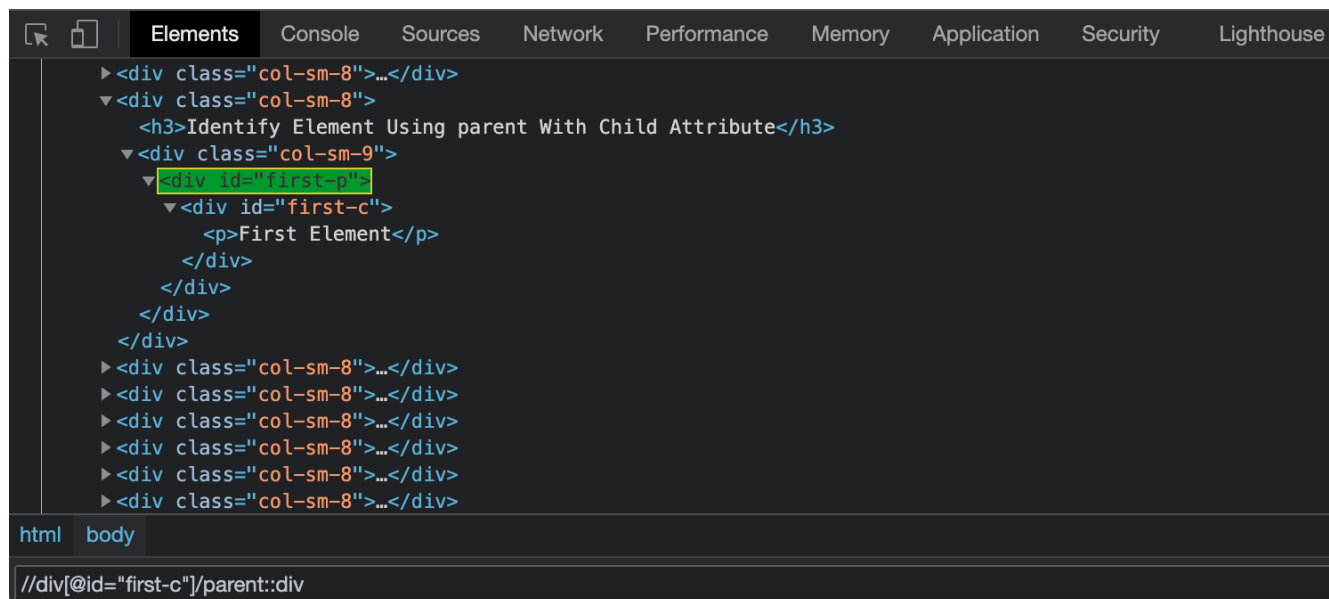
*** NOTE ***

This is the second time we are not uniquely identifying text within a *p* tag or attribute. This example is designed to show you how to uniquely identify a parent if you needed to obtain this unique element in your automation. This particular example is not practical but it is very relevant for your understanding if you ever needed to do such.

24

XPath Code

```
//div[@id="first-c"]/parent::div
//div[@id="first-c"]/..
```


Verify 1 of 1 and hover your mouse over the green box and watch it uniquely identify your element in the web site above.

Identify Element Where Parent Has Unique Child Attribute

Web Code

```
<a href="https://python.org">
    <p><span id="link">First Element</span></p>
</a>
```

To uniquely identify the a tag that has an *href* attribute =
"https://python.org", we start with the // where there is a *span* tag
that has an *id* attribute = *"link"* and a .. .. (representing the
parent's parent directory).

*** NOTE ***

This is the third time we are not uniquely identifying text within a
p tag or attribute. This example is designed to show you how to
uniquely identify a parent if you needed to obtain this unique
element in your automation. This particular example shows you how to
capture a web site address and is very relevant for your
understanding if you ever needed to do such.

XPath Code

```
//span[@id="link"]/../..
```
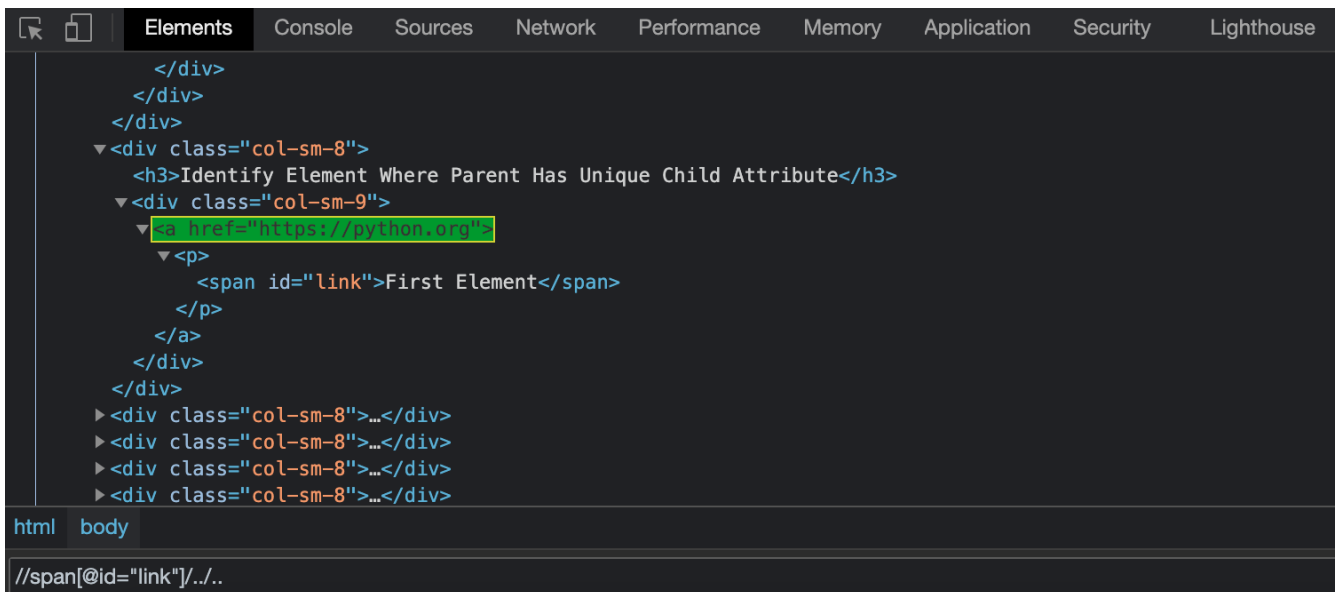

Verify 1 of 1 and hover your mouse over the green box and watch it
uniquely identify your element in the web site above.

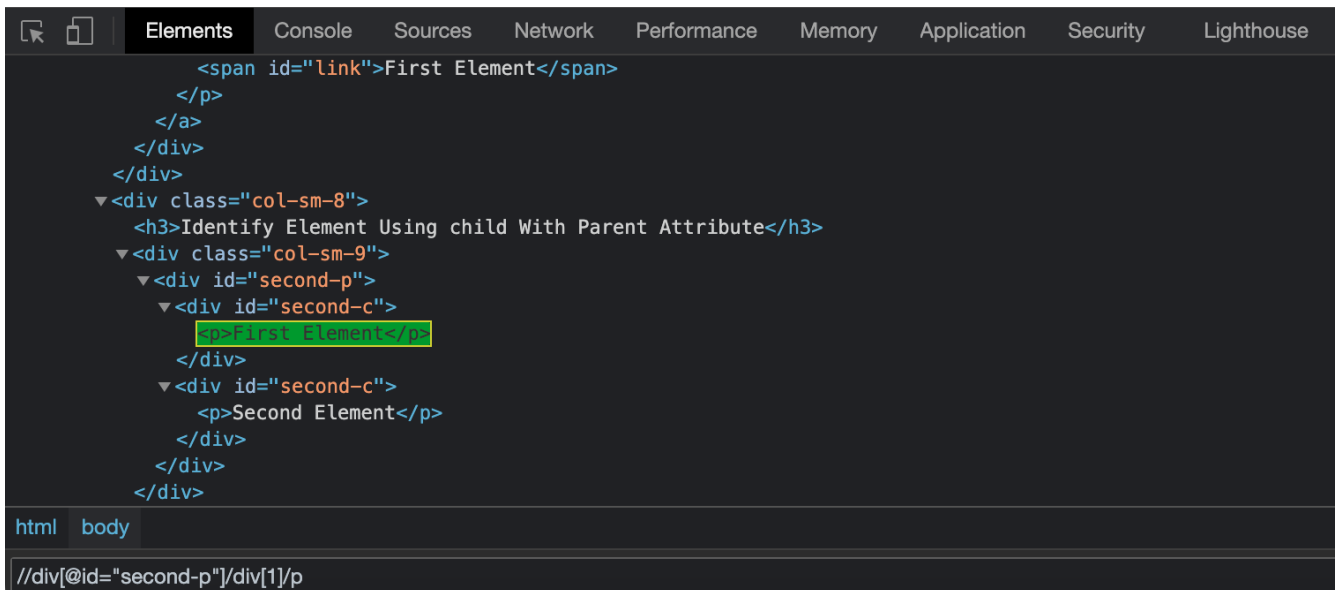**Identify Element Using child With Parent Attribute**

Web Code

```
<div id="second-p">
    <div id="second-c">
        <p>First Element</p>
    </div>
    <div id="second-c">
        <p>Second Element</p>
    </div>
</div>
```

To uniquely identify the *First Element*, we start with the // where
there is a *div* tag that has an *id* attribute = *"second-p"* and a *div*
tag 1st element and a *p* tag.

XPath Code
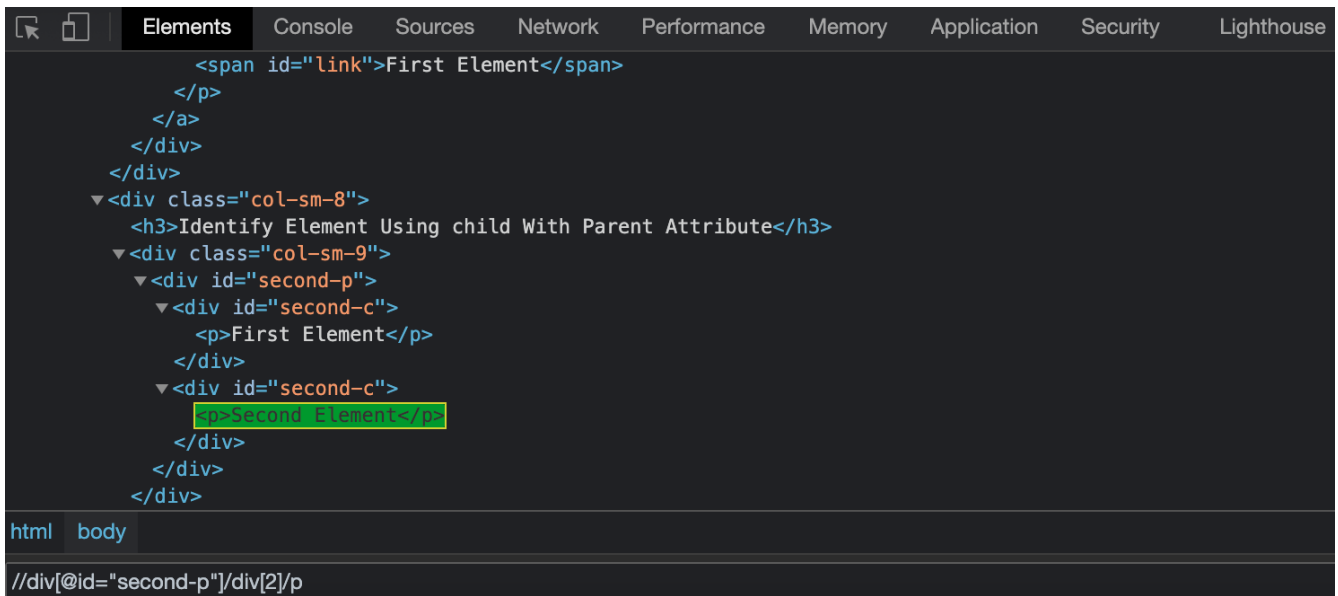
```
//div[@id="second-p"]/div[1]/p
```


Verify 1 of 1 and hover your mouse over the green box and watch it
uniquely identify your element in the web site above.

To uniquely identify the *Second Element*, we start with the // where
there is a *div* tag that has an *id* attribute = *"second-p"* and a *div*
tag 2nd element and a *p* tag.

XPath Code

```
//div[@id="second-p"]/div[2]/p
```

```
            <span id="link">First Element</span>
          </p>
        </a>
      </div>
    </div>
  ▼<div class="col-sm-8">
      <h3>Identify Element Using child With Parent Attribute</h3>
    ▼<div class="col-sm-9">
      ▼<div id="second-p">
        ▼<div id="second-c">
            <p>First Element</p>
          </div>
        ▼<div id="second-c">
            <p>Second Element</p>
          </div>
        </div>
      </div>
    </div>
```

html   body

//div[@id="second-p"]/div[2]/p

Verify 1 of 1 and hover your mouse over the green box and watch it uniquely identify your element in the web site above.

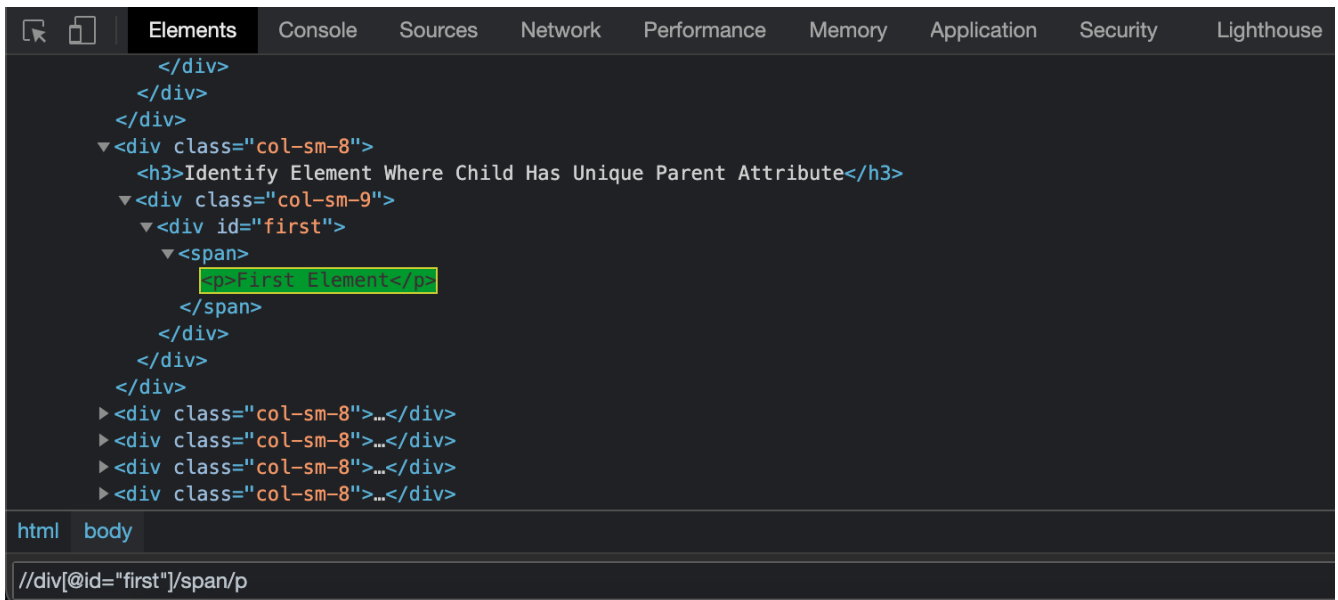**Identify Element Where Child Has Unique Parent Attribute**

Web Code

```
<div id="first">
    <span>
        <p>First Element</p>
    </span>
</div>
```

To uniquely identify the *First Element*, we start with the // where there is a *div* tag that has an *id* attribute = *"first"* and a *span* tag and a *p* tag.

You can alternatively identify the *First Element*, starting with the // where there is a *div* tag that has an *id* attribute = *"first"* and a *p* tag.

XPath Code

```
//div[@id="first"]/span/p
//div[@id="first"]//p
```

```
          </div>
        </div>
      </div>
    ▼<div class="col-sm-8">
        <h3>Identify Element Where Child Has Unique Parent Attribute</h3>
      ▼<div class="col-sm-9">
        ▼<div id="first">
          ▼<span>
              <p>First Element</p>
            </span>
          </div>
        </div>
      </div>
    ▶<div class="col-sm-8">…</div>
    ▶<div class="col-sm-8">…</div>
    ▶<div class="col-sm-8">…</div>
    ▶<div class="col-sm-8">…</div>
```
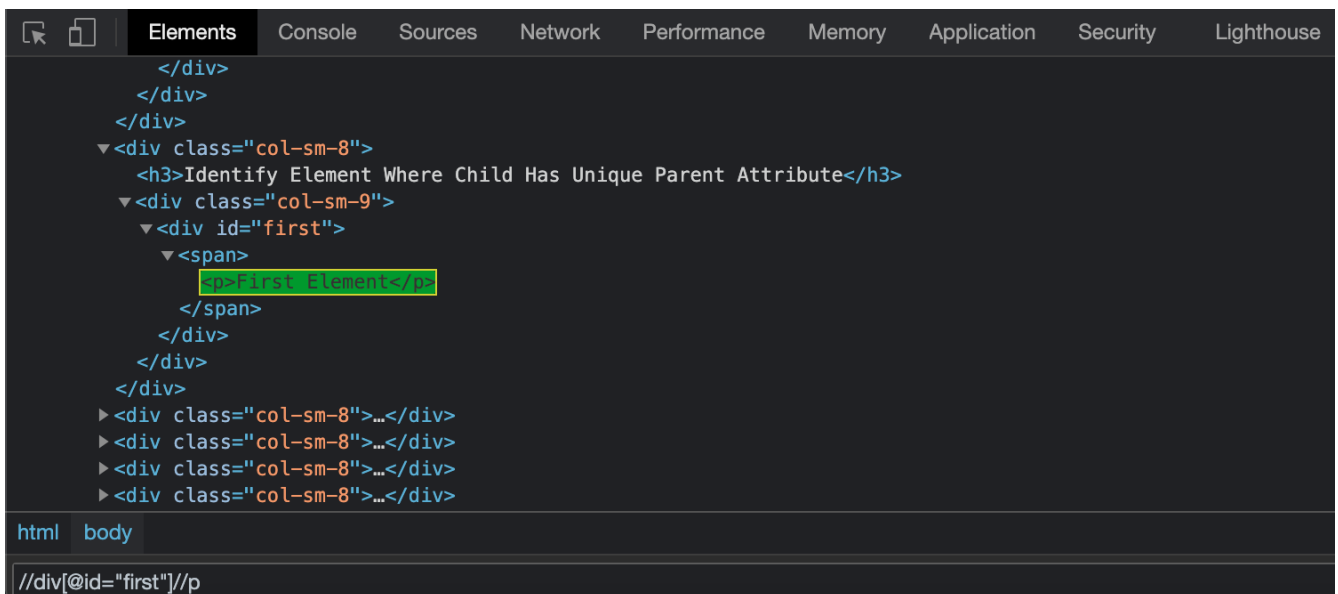
html   body

//div[@id="first"]/span/p

```
          </div>
        </div>
      </div>
    ▼<div class="col-sm-8">
        <h3>Identify Element Where Child Has Unique Parent Attribute</h3>
      ▼<div class="col-sm-9">
        ▼<div id="first">
          ▼<span>
              <p>First Element</p>
            </span>
          </div>
        </div>
      </div>
    ▶<div class="col-sm-8">…</div>
    ▶<div class="col-sm-8">…</div>
    ▶<div class="col-sm-8">…</div>
    ▶<div class="col-sm-8">…</div>
```

html   body

//div[@id="first"]//p

Verify 1 of 1 and hover your mouse over the green box and watch it
uniquely identify your element in the web site above.

Identify Element Using preceding With A Child Attribute

Web Code

```
<div class="first-pre"></div>
<div id="first-pre">
    <div id="first-pre-c">
        <div id="first-pre-gc">
            <div id="first-pre-ggc">
                <p>First Element</p>
            </div>
        </div>
    </div>
</div>
```

To uniquely identify the *div* tag that has a class attribute = *"first-p"*, we start with the // where there is a *div* tag that has has an *id* attribute = *"first-pre-ggc"* and a preceding attribute and a *div* tag with a *class* attribute = *"first-pre"*.

*** NOTE ***

This is the fourth time we are not uniquely identifying text within a *p* tag or attribute. This example is designed to show you how to uniquely identify a parent before your current node, except ancestors and attribute nodes, if you needed to obtain this unique element in your automation. This particular example is not practical but it is very relevant for your understanding if you ever needed to do such.

XPath Code

```
//div[@id="first-pre-ggc"]/preceding::div[@class="first-pre"]
```
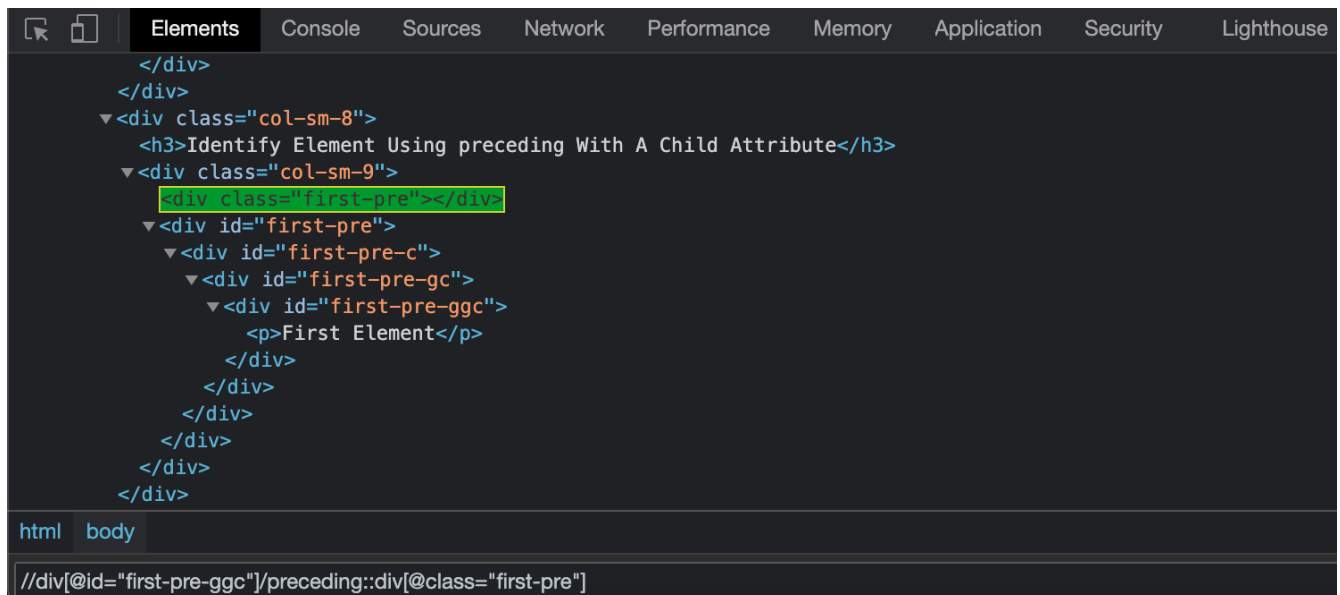

30

Verify 1 of 1 and hover your mouse over the green box and watch it uniquely identify your element in the web site above.

*** NODES ***

Before we move on I want to make sure you are clear on the concept of the current node. If you look at the above image the element that is in green is BEFORE the current node or the node that has a *div* tag with an *id* attribute = *"first-pre-ggc"*.

To illustrate, you can click on the arrow of the node which contains our, *"first-pre-ggc"*, to collapse all the elements in the current node as follows. Here we can CLEARLY see the element in green precedes or comes BEFORE the current node containing the element, *"first-pre-ggc"*.

```
▼<div class="col-sm-9">
    <div class="first-pre"></div>
  ▶<div id="first-pre">…</div>
```

Identify Element Using preceding-sibling With A Sibling Attribute

Web Code

```
<div class="first-pre-s"></div>
<div id="first-pre-s">
    <div id="first-pre-s-c">
        <div id="first-pre-s-gc">
            <div id="first-pre-s-ggc">
                <p>First Element</p>
            </div>
            <div id="second-pre-s-ggc">
                <p>Second Element</p>
            </div>
            <div id="third-pre-s-ggc">
                <p>Third Element</p>
            </div>
        </div>
    </div>
</div>
```

To uniquely identify the *div* tag that has a *class* attribute = *"first-pre-s-ggc"*, we start with the // where there is a *div* tag that has has an *id* attribute = *"third-pre-s-ggc"* and a *preceding-sibling* attribute and a *div* tag with an *id* attribute = *"first-pre-s-ggc"*.

*** NOTE ***

This is the fifth time we are not uniquely identifying text within a *p* tag or attribute. This example is designed to show you how to

uniquely identify a sibling before your current node, if you needed
to obtain this unique element in your automation. This particular
example is not practical but it is very relevant for your
understanding if you ever needed to do such.

XPath Code

```
//div[@id="third-pre-s-ggc"]/preceding-sibling::div[@id="first-pre-s-ggc"]
```

Verify 1 of 1 and hover your mouse over the green box and watch it
uniquely identify your element in the web site above.

We can collapse the nodes to see more clearly that they are all
siblings in the DOM (Document Object Model) web tree structure.

Identify Element Using following With A Parent Attribute

Web Code

```
<div class="first-fol"></div>
<div id="first-fol">
    <div id="first-fol-c">
        <div id="first-fol-gc">
            <div id="first-fol-ggc">
                <p>First Element</p>
            </div>
        </div>
    </div>
</div>
```
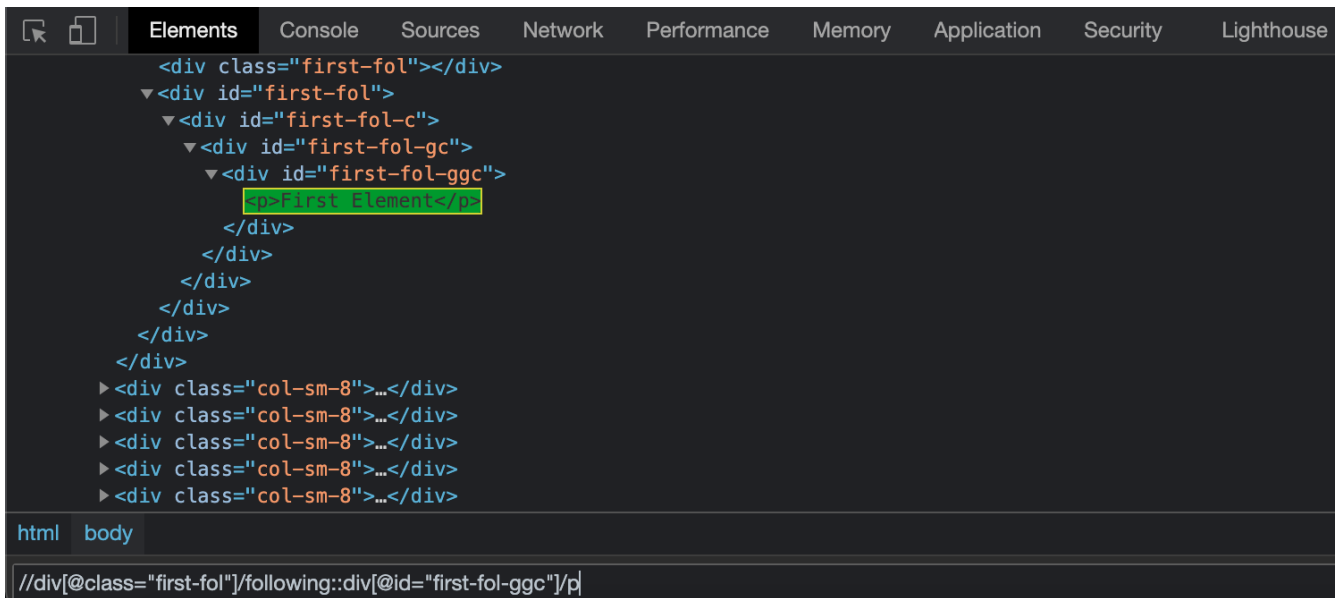
To uniquely identify the *First Element*, we start with the // where
there is a *div* tag that has a *class* attribute = *"first-fol"* and a
following attribute and a *div* tag that has an *id* attribute = *"first-
fol-ggc"* and a *p* tag.

XPath Code

```
//div[@class="first-fol"]/following::div[@id="first-fol-ggc"]/p
```

```
<div class="first-fol"></div>
▼<div id="first-fol">
   ▼<div id="first-fol-c">
      ▼<div id="first-fol-gc">
         ▼<div id="first-fol-ggc">
            <p>First Element</p>
         </div>
      </div>
   </div>
</div>
</div>
▶<div class="col-sm-8">…</div>
▶<div class="col-sm-8">…</div>
▶<div class="col-sm-8">…</div>
▶<div class="col-sm-8">…</div>
▶<div class="col-sm-8">…</div>
```

html   body

//div[@class="first-fol"]/following::div[@id="first-fol-ggc"]/p

Verify 1 of 1 and hover your mouse over the green box and watch it uniquely identify your element in the web site above.

This is simply the opposite of preceding as it selects everything in the document after the closing tag of the current node.

Identify Element Using following-sibling With A Sibling Attribute

Web Code
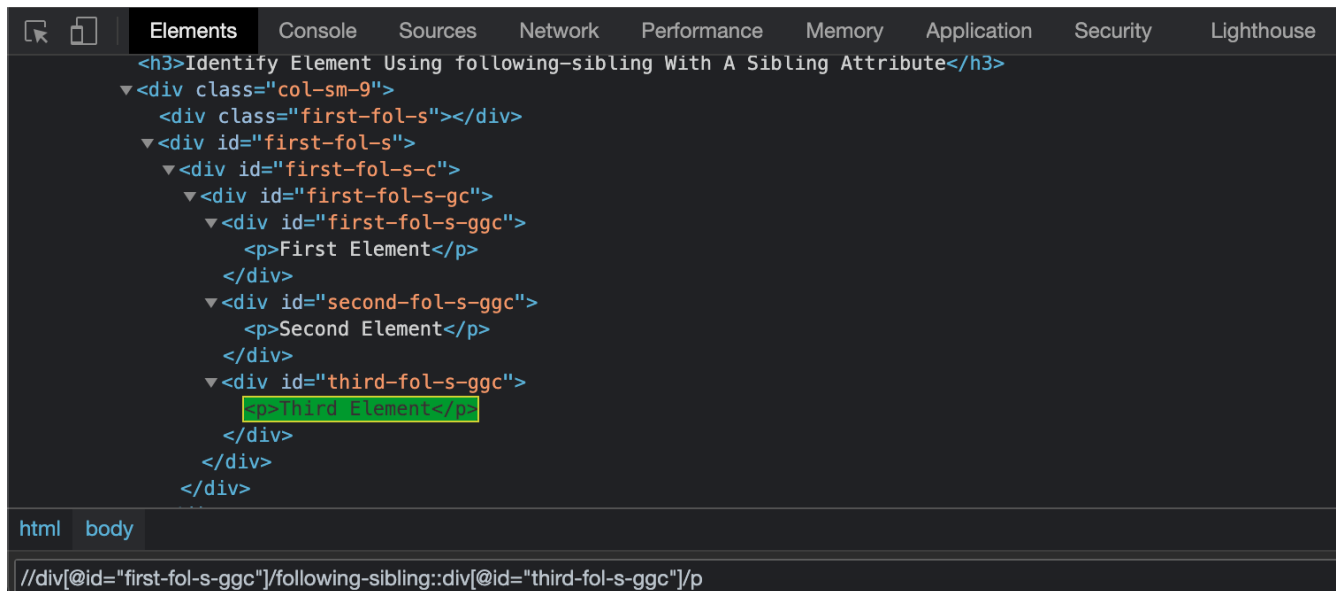
```
<div class="first-fol-s"></div>
<div id="first-fol-s">
    <div id="first-fol-s-c">
        <div id="first-fol-s-gc">
            <div id="first-fol-s-ggc">
                <p>First Element</p>
            </div>
            <div id="second-fol-s-ggc">
                <p>Second Element</p>
            </div>
            <div id="third-fol-s-ggc">
                <p>Third Element</p>
            </div>
        </div>
    </div>
</div>
```

To uniquely identify the *Third Element*, we start with the *//* where there is a *div* tag that has an *id* attribute = *"first-fol-s-ggc"* and a *following-sibling* attribute and a *div* tag that has an *id* attribute = *"third-fol-s-ggc"* and a *p* tag.

33

XPath Code

```
//div[@id="first-fol-s-ggc"]/following-sibling::div[@id="third-fol-s-ggc"]/p
```


Verify 1 of 1 and hover your mouse over the green box and watch it uniquely identify your element in the web site above.

This is simply the opposite of *preceding-sibling* as it selects all siblings after the current node.

Identify Element Using contains With Partial Attribute

Web Code

```
<p name="first" id="first-1">First Element</p>
<p name="first" id="first-2">Second Element</p>
```

To uniquely identify the *First Element*, we start with the // where there is a *p* tag that has a *contains* function and an *id* attribute = *"-1"*.

You can alternatively identify the *First Element*, starting with the // where there is a *p* tag that has a *contains* function and an *id* attribute = *"-1"* and a *text* function searching for the text, *"First Element"*.

XPath Code

```
//p[contains(@id, "-1")]
//p[contains(@id, "-1") and text()="First Element"]
```

34

```
▼<div class="col-sm-8">
    <h3>Identify Element Using contains With Partial Attribute</h3>
  ▼<div class="col-sm-9">
      <p name="first" id="first-1">First Element</p>
      <p name="first" id="first-2">Second Element</p>
    </div>
  </div>
  ▶<div class="col-sm-8">…</div>
  ▶<div class="col-sm-8">…</div>
  ▶<div class="col-sm-8">…</div>
  ▶<div class="col-sm-8">…</div>
  ▶<div class="col-sm-8">…</div>
  ▶<div class="col-sm-8">…</div>
  ▶<div class="col-sm-8">…</div>
  ▶<div class="col-sm-8">…</div>
  ▶<div class="col-sm-8">…</div>
  ▶<div class="col-sm-8">…</div>
```

html   body

//p[contains(@id, "-1")]


```
▼<div class="col-sm-8">
    <h3>Identify Element Using contains With Partial Attribute</h3>
  ▼<div class="col-sm-9">
      <p name="first" id="first-1">First Element</p>
      <p name="first" id="first-2">Second Element</p>
    </div>
  </div>
  ▶<div class="col-sm-8">…</div>
  ▶<div class="col-sm-8">…</div>
  ▶<div class="col-sm-8">…</div>
  ▶<div class="col-sm-8">…</div>
  ▶<div class="col-sm-8">…</div>
  ▶<div class="col-sm-8">…</div>
  ▶<div class="col-sm-8">…</div>
  ▶<div class="col-sm-8">…</div>
  ▶<div class="col-sm-8">…</div>
  ▶<div class="col-sm-8">…</div>
```

html   body

//p[contains(@id, "-1") and text()="First Element"]

Verify 1 of 1 and hover your mouse over the green box and watch it uniquely identify your element in the web site above.
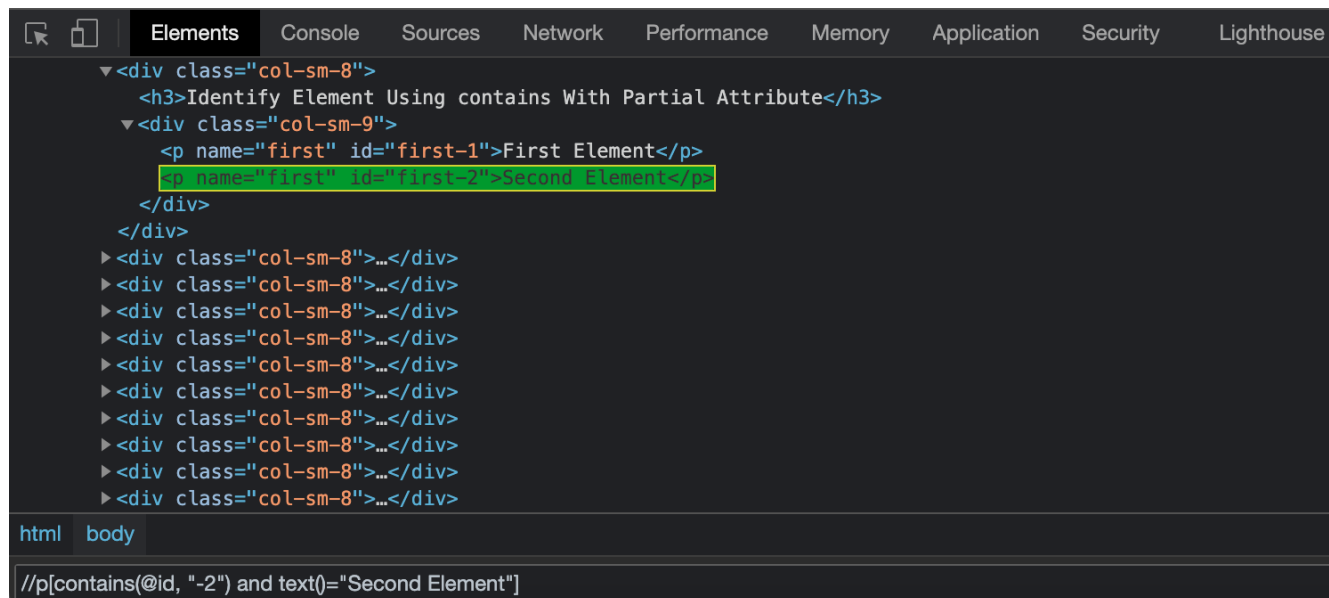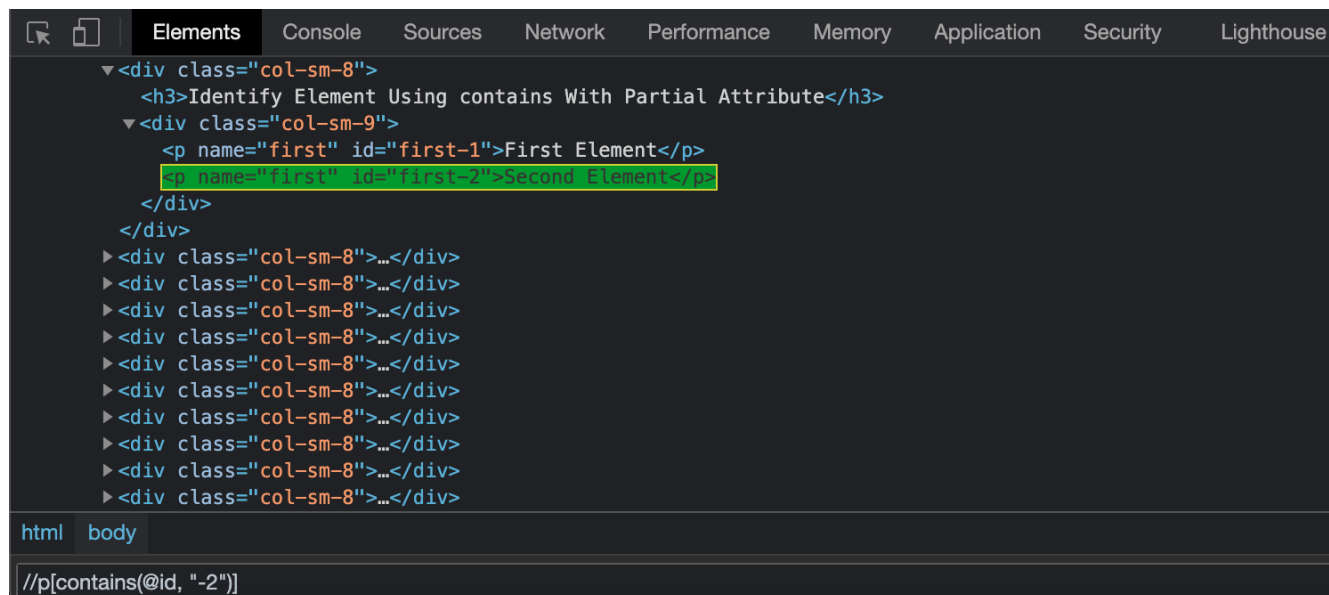
To uniquely identify the *Second Element*, we start with the // where there is a *p* tag that has a *contains* function and an *id* attribute = *"-2"*.

You can alternatively identify the *Second Element*, starting with the // where there is a *p* tag that has a *contains* function and an *id* attribute = *"-2"* and a *text* function searching for the text, *"Second Element"*.

XPath Code

XPath Code

```
//p[contains(@id, "-2")]
//p[contains(@id, "-2") and text()="Second Element"]
```


Verify 1 of 1 and hover your mouse over the green box and watch it uniquely identify your element in the web site above.

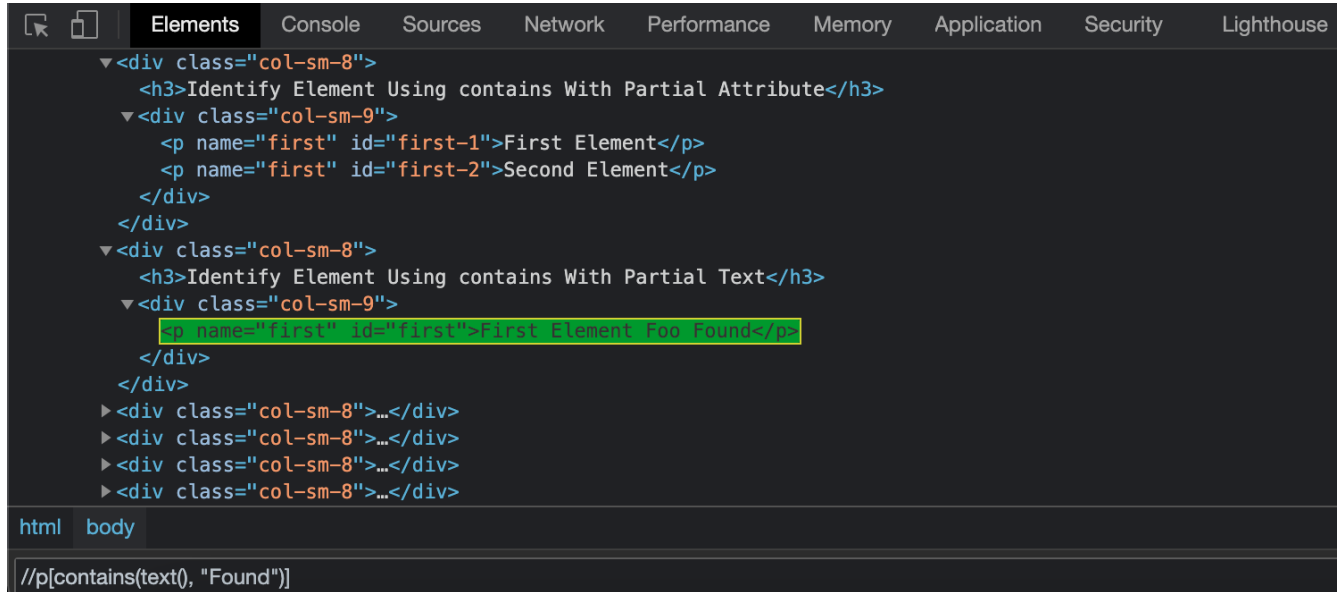*Identify Element Using contains With Partial Text*

Web Code

```
<p name="first" id="first">First Element Foo Found</p>
```

To uniquely identify the *First Element Foo Found*, we start with
the // where there is a *p* tag that has a *contains* function and a *text*
function searching for the text, *"Found"*.

XPath Code

```
//p[contains(text(), "Found")]
```


Verify 1 of 1 and hover your mouse over the green box and watch it
uniquely identify your element in the web site above.

Identify Element Using starts-with With Partial Text

Web Code

```
<p name="first" id="first">First Element Text - Dynamic Text</p>
```

To uniquely identify the *First Element Text - Dynamic Text*, we start
with the // where there is a *p* tag that has a *starts-with* function
and a *text* function searching for the text, *"First Element Text -"*.

XPath Code

```
//p[starts-with(text(), "First Element Text -")]
```
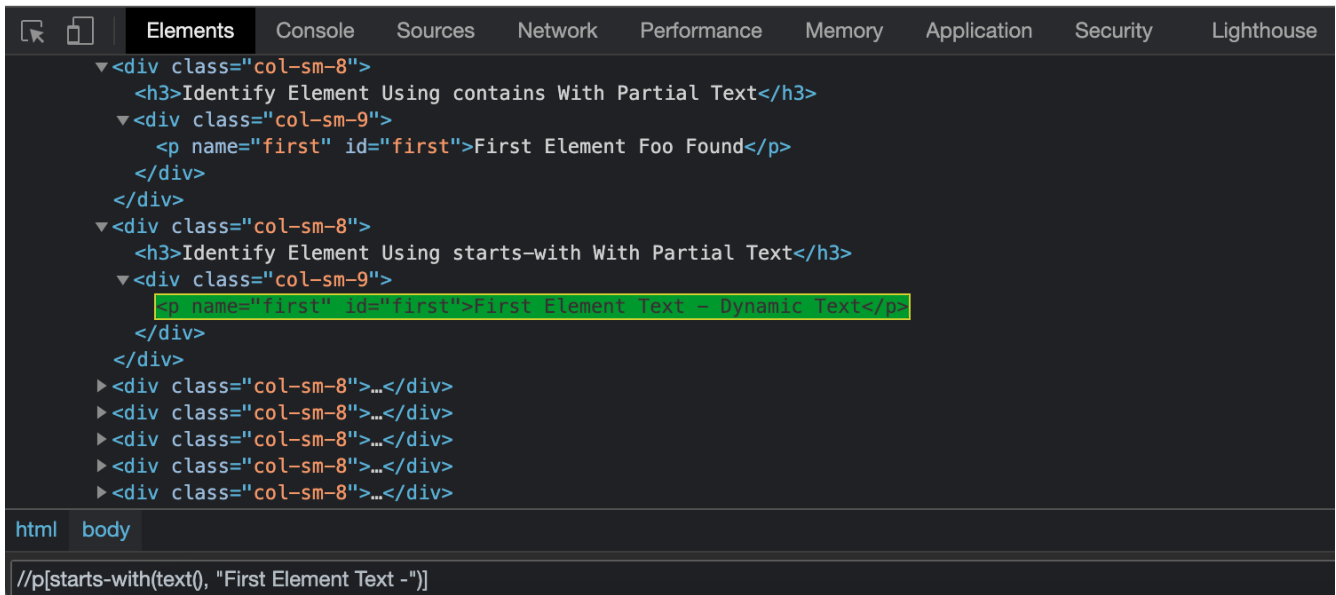
Verify 1 of 1 and hover your mouse over the green box and watch it uniquely identify your element in the web site above.

Identify Element Using position & last With Attribute

Web Code

```
<table id="first" style="width:100%">
    <tbody>
        <tr>
            <td>
                <p><label class="checkbox-inline"><input type="checkbox" value="">First
                        Element</label></p>
            </td>
        </tr>
        <tr>
            <td>
                <p><label class="checkbox-inline"><input type="checkbox" value="">Second
                        Element</label></p>
            </td>
        </tr>
        <tr>
            <td>
                <p><label class="checkbox-inline"><input type="checkbox" value="">Third
                        Element</label></p>
            </td>
        </tr>
    </tbody>
</table>
```
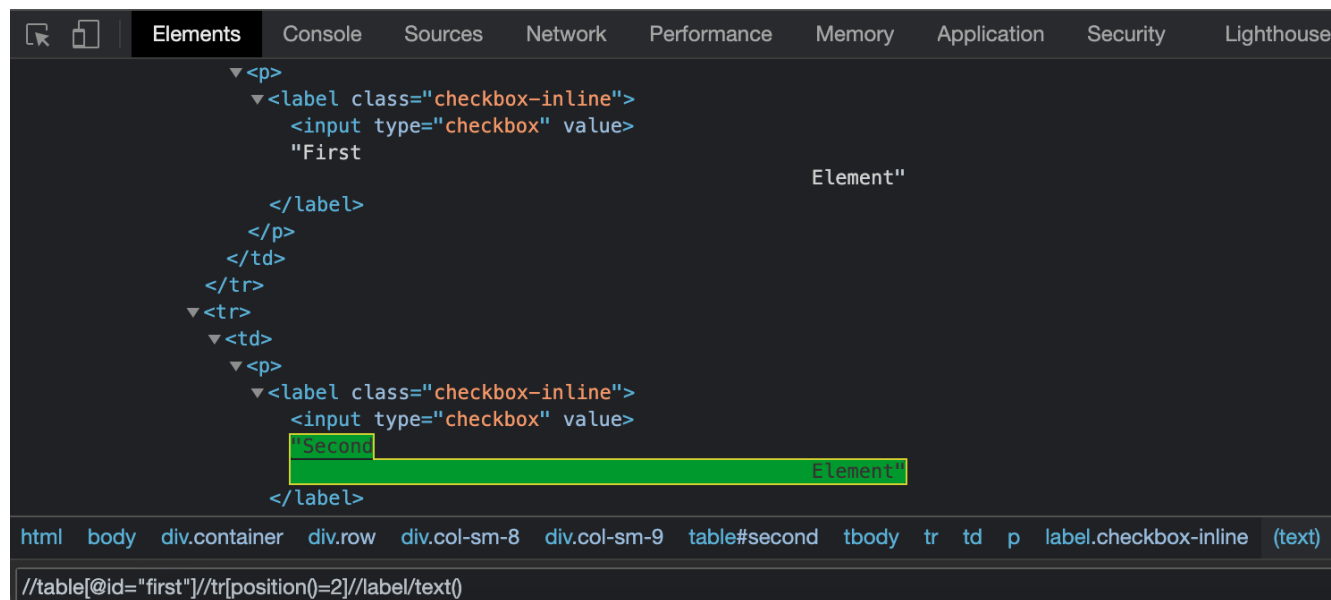
To uniquely identify the *Second Element*, we start with the *//* where there is a *table* tag that has a *id* attribute = *"first"* and a *tr* tag

38

with a *position* function searching for *2* and a *label* tag and a *text* function searching for the text inside the label.

XPath Code

```
//table[@id="first"]//tr[position()=2]//label/text()
```
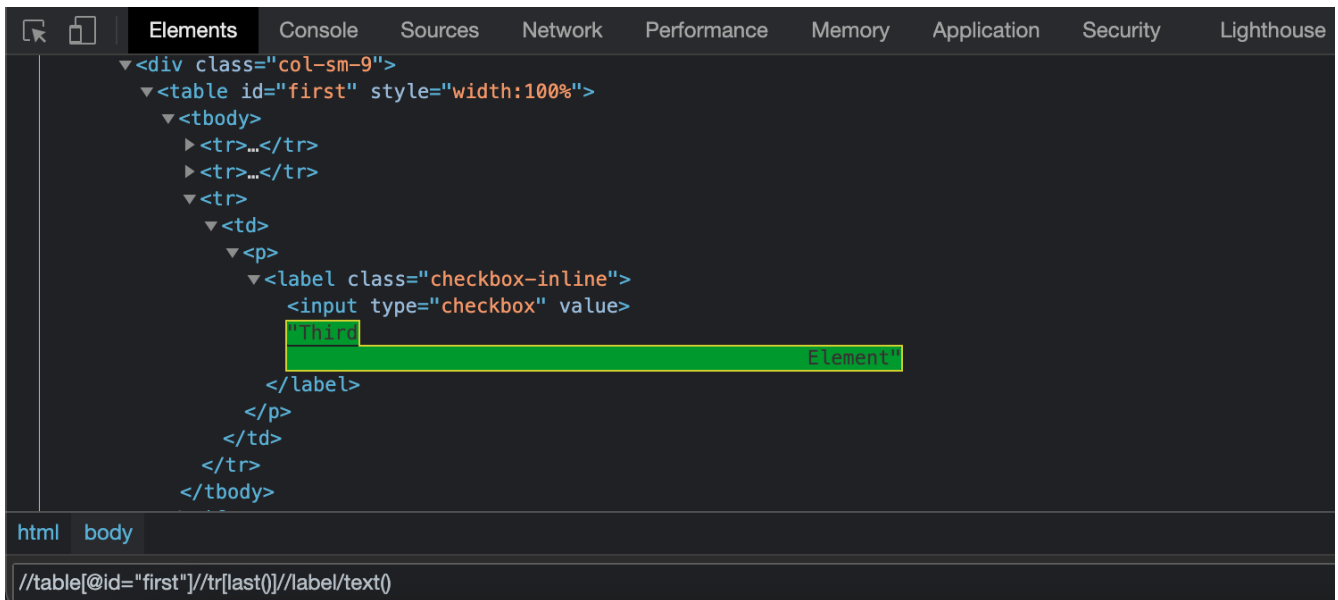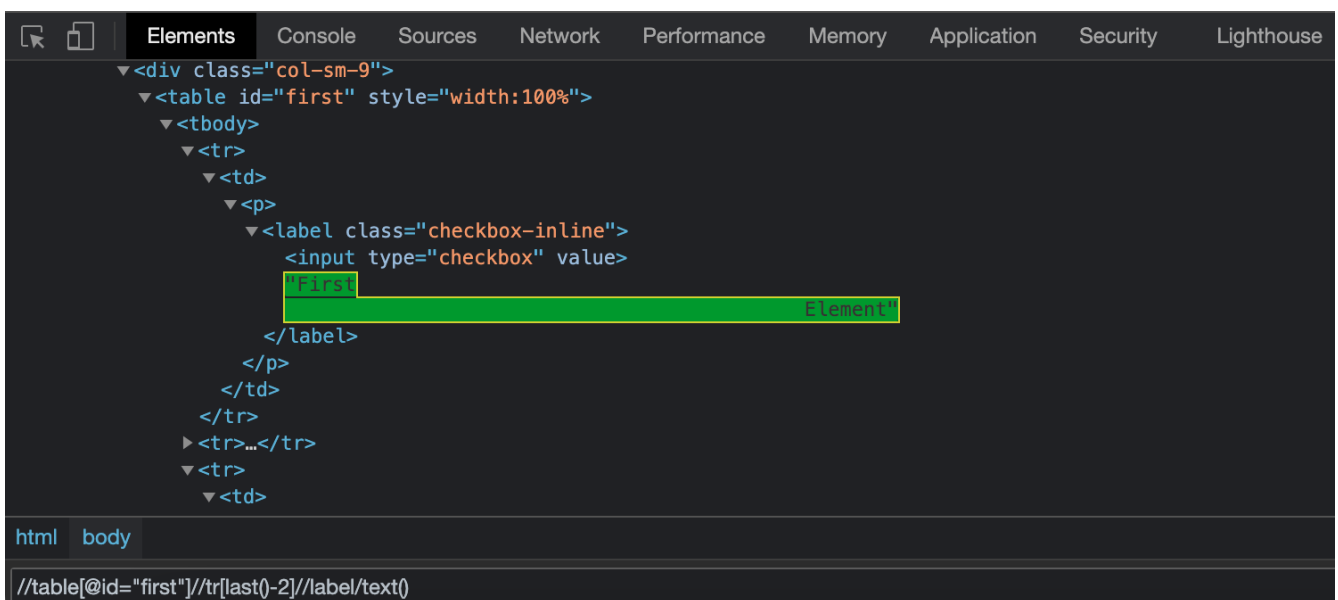

Verify 1 of 1 and hover your mouse over the green box and watch it uniquely identify your element in the web site above.

To uniquely identify the *Third Element*, we start with the *//* where there is a *table* tag that has an *id* attribute = *"first"* and a *tr* tag with a *last* function searching for the last element and a *label* tag and a *text* function searching for the text inside the label.

XPath Code

```
//table[@id="first"]//tr[position()=2]//label/text()
```

Verify 1 of 1 and hover your mouse over the green box and watch it uniquely identify your element in the web site above.

To uniquely identify the *First Element*, we start with the *//* where there is a *table* tag that has a *id* attribute = *"first"* and a *tr* tag with a *last* function searching for *-2* and a *label* tag and a *text* function searching for the text inside the label.

XPath Code

```
//table[@id="first"]//tr[last()-2]//label/text()
```

Verify 1 of 1 and hover your mouse over the green box and watch it
uniquely identify your element in the web site above.

Identify Element Using normalize-space With Text
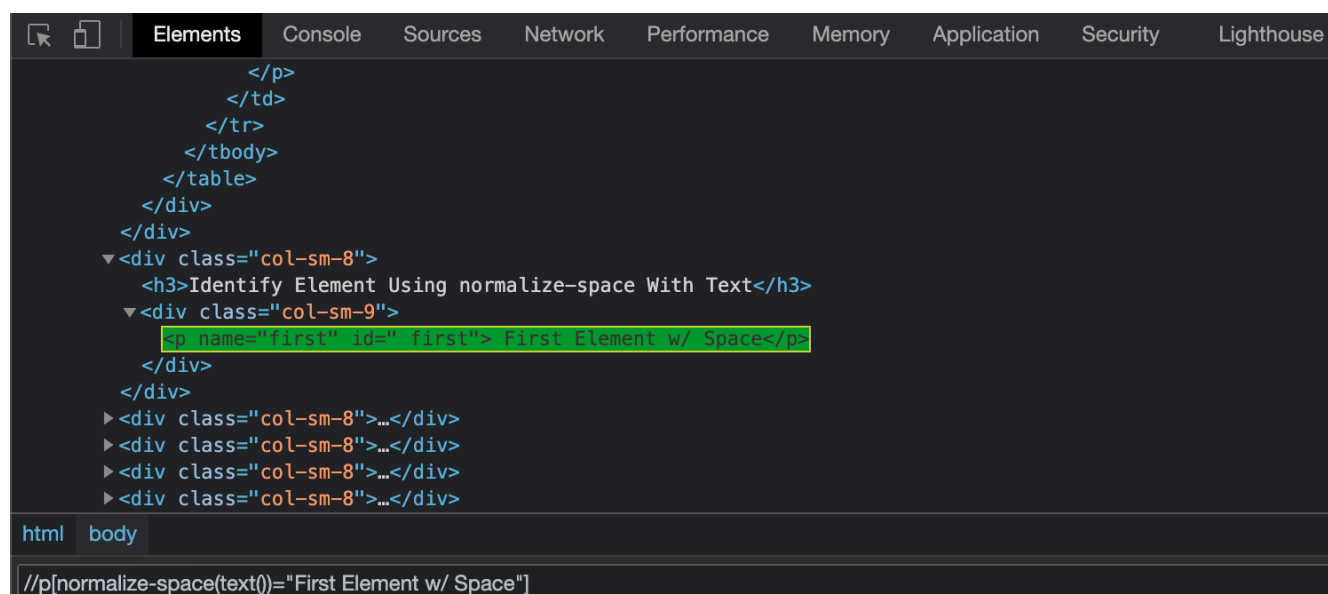
Web Code

```
<p name="first" id=" first"> First Element w/ Space</p>
```

To uniquely identify the *First Element w/ Space*, we start with the //
where there is a *normalize-space* function and a *text* function
searching for the text, *"First Element w/ Space"*.

XPath Code

```
//p[normalize-space(text())="First Element w/ Space"]
```


Verify 1 of 1 and hover your mouse over the green box and watch it
uniquely identify your element in the web site above.

Identify Element Using normalize-space With Attribute
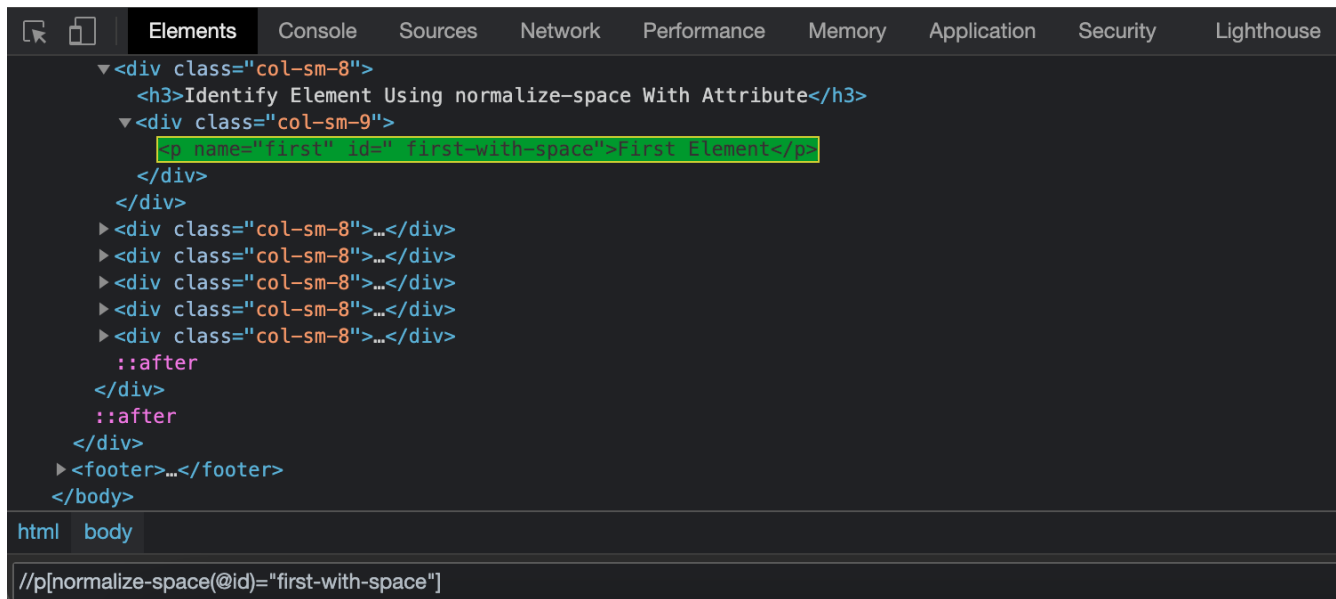
Web Code

```
<p name="first" id=" first-with-space">First Element</p>
```

To uniquely identify the *First Element*, we start with the // where
there is a *normalize-space* function and an *id* attribute = *"first-
with-space"*.

XPath Code

```
//p[normalize-space(@id)="first-with-space"]
```


Verify 1 of 1 and hover your mouse over the green box and watch it uniquely identify your element in the web site above.
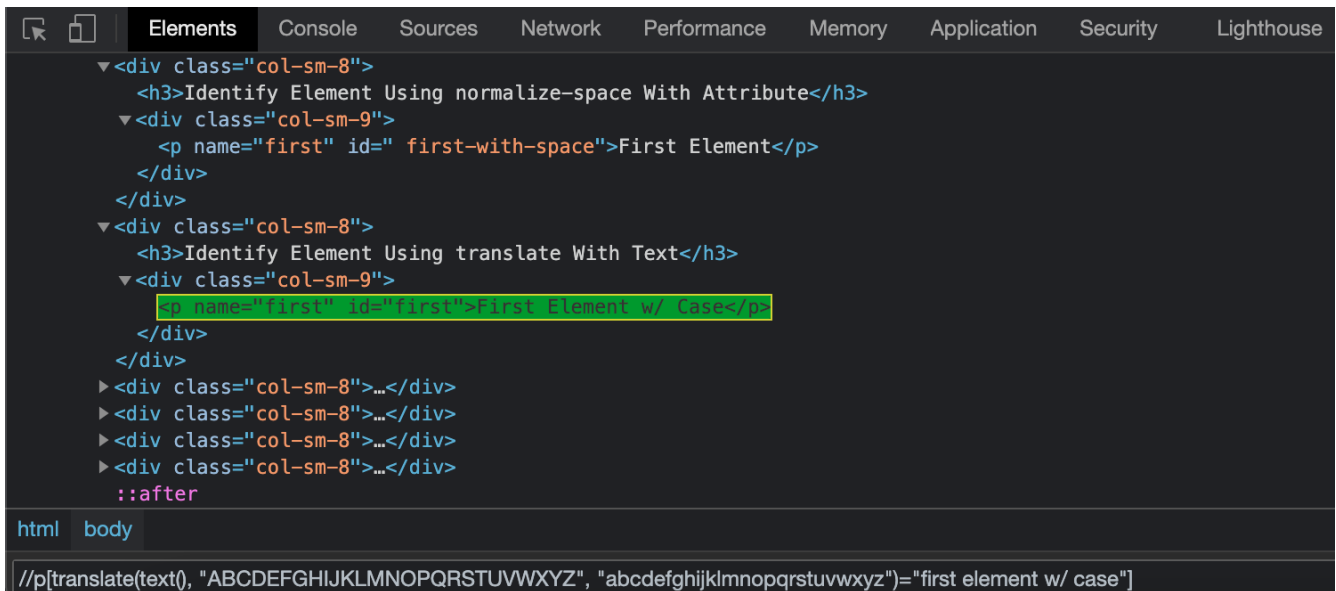
Identify Element Using translate With Text

Web Code

```
<p name="first" id="first">First Element w/ Case</p>
```

To uniquely identify the *First Element w/ Case*, we start with the *//* where there is a *translate* function and *text* function checking against all capital and lowercase letters, searching for the text, *"first element w/ case"*.

XPath Code

```
//p[normalize-space(translate(text(), "ABCDEFGHIJKLMNOPQRSTUVWXYZ",
"abcdefghijklmnopqrstuvwxyz"))="first element w/ space"]
```

```
//p[translate(text(), "ABCDEFGHIJKLMNOPQRSTUVWXYZ", "abcdefghijklmnopqrstuvwxyz")="first element w/ case"]
```

Verify 1 of 1 and hover your mouse over the green box and watch it uniquely identify your element in the web site above.

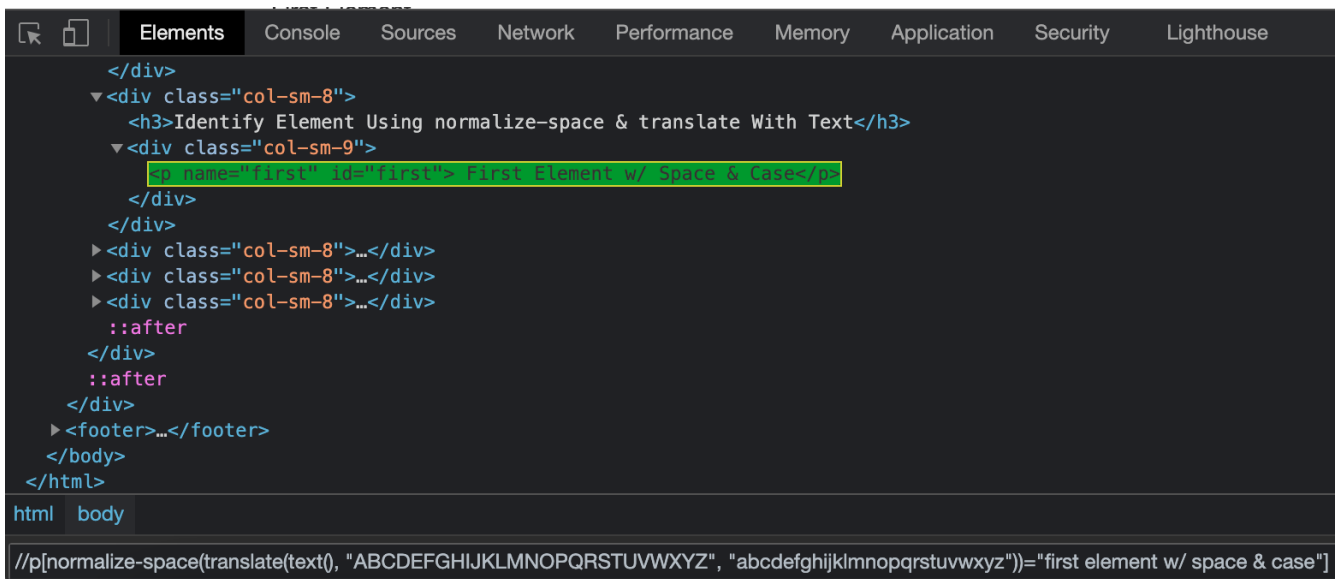**Identify Element Using normalize-space & translate With Text**

Web Code

```
<p name="first" id="first"> First Element w/ Space & Case</p>
```

To uniquely identify the *First Element w/ Space & Case*, we start with the // where there is a *normalize-space* function and a *translate* function and *text* function checking against all capital and lowercase letters, searching for the text, *"first element w/ case"*.

XPath Code

```
//p[normalize-space(translate(text(), "ABCDEFGHIJKLMNOPQRSTUVWXYZ",
"abcdefghijklmnopqrstuvwxyz"))="first element w/ space & case"]
```

43

</div>

```
     </div>
   ▼<div class="col-sm-8">
       <h3>Identify Element Using normalize-space & translate With Text</h3>
     ▼<div class="col-sm-9">
         <p name="first" id="first"> First Element w/ Space & Case</p>
       </div>
     </div>
   ▶<div class="col-sm-8">…</div>
   ▶<div class="col-sm-8">…</div>
   ▶<div class="col-sm-8">…</div>
     ::after
   </div>
   ::after
  </div>
 ▶<footer>…</footer>
 </body>
</html>

html   body

//p[normalize-space(translate(text(), "ABCDEFGHIJKLMNOPQRSTUVWXYZ", "abcdefghijklmnopqrstuvwxyz"))="first element w/ space & case"]
```

Verify 1 of 1 and hover your mouse over the green box and watch it uniquely identify your element in the web site above.

Identify Element Using NOT With Attribute

Web Code

```
<p name="first-not" id="first-not">First Element</p>
<p name="second-not" id="second-not">Second Element</p>
```

To uniquely identify the *First Element*, we start with the // where there is a *p* tag with a name attribute = *"first-not"* and a *not* attribute and an *id* attribute = *"second-not"*.

XPath Code

```
//p[@name="first-not" and not (@id="second-not")]
```

```
</div>
▼<div class="col-sm-8">
    <h3>Identify Element Using normalize-space & translate With Text</h3>
  ▼<div class="col-sm-9">
      <p name="first" id="first"> First Element w/ Space & Case</p>
    </div>
  </div>
▼<div class="col-sm-8">
    <h3>Identify Element Using NOT With Attribute</h3>
  ▼<div class="col-sm-9">
      <p name="first-not" id="first-not">First Element</p>
      <p name="second-not" id="second-not">Second Element</p>
    </div>
  </div>
▶<div class="col-sm-8">…</div>
▶<div class="col-sm-8">…</div>
    ::after
```

html body

`//p[@name="first-not" and not (@id="second-not")]`

Verify 1 of 1 and hover your mouse over the green box and watch it
uniquely identify your element in the web site above.

To uniquely identify the *Second Element*, we start with the // where
there is a *p* tag with a *name* attribute = *"second-not"* and a *not*
attribute and an *id* attribute = *"first-not"*.

XPath Code

```
//p[@name="second-not" and not (@id="first-not")]
```



```
</div>
▼<div class="col-sm-8">
    <h3>Identify Element Using normalize-space & translate With Text</h3>
  ▼<div class="col-sm-9">
      <p name="first" id="first"> First Element w/ Space & Case</p>
    </div>
  </div>
▼<div class="col-sm-8">
    <h3>Identify Element Using NOT With Attribute</h3>
  ▼<div class="col-sm-9">
      <p name="first-not" id="first-not">First Element</p>
      <p name="second-not" id="second-not">Second Element</p>
    </div>
  </div>
▶<div class="col-sm-8">…</div>
▶<div class="col-sm-8">…</div>
    ::after
```

html body

`//p[@name="second-not" and not (@id="first-not")]`

Verify 1 of 1 and hover your mouse over the green box and watch it uniquely identify your element in the web site above.

Identify Element Using substring-before With Text
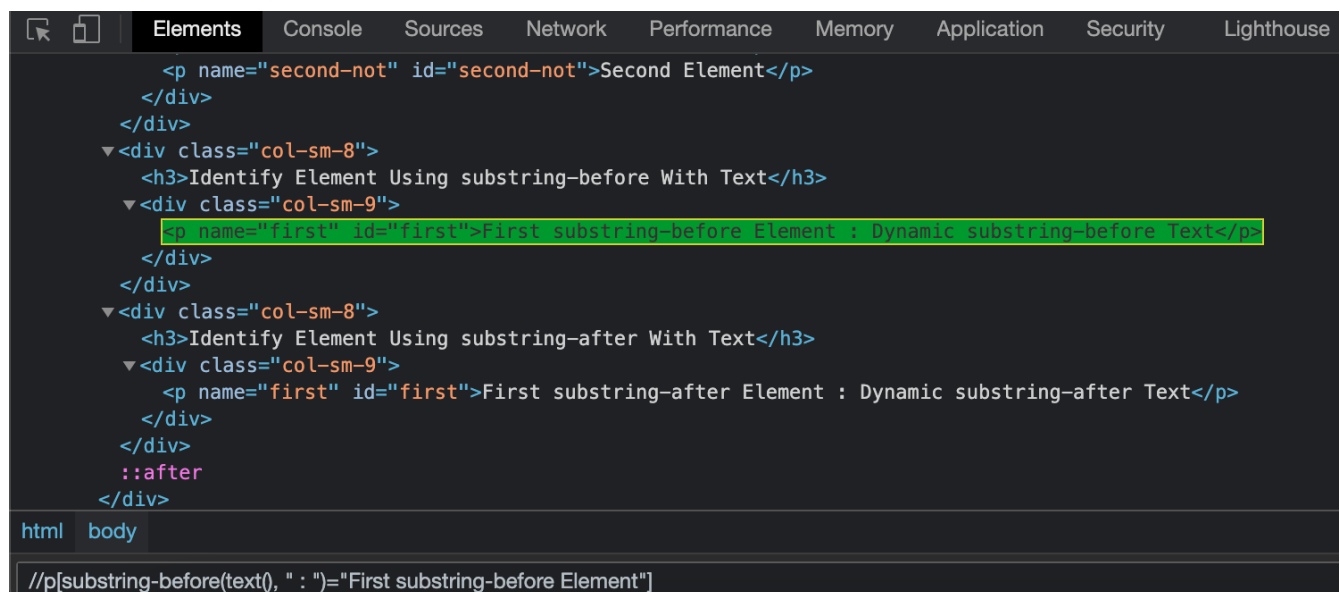
Web Code

```
<p name="first" id="first">First substring-before Element : Dynamic substring-before Text</p>
```

To uniquely identify the *First substring-before Element : Dynamic substring-before Text*, we start with the *//* where there is a *p* tag with a *substring-before* function and a *text* function, searching for the text, *" : "* and then the *substring-before* function searching for the text, *"First substring-before Element"*.

XPath Code

```
//p[substring-before(text(), " : ")="First substring-before Element"]
```


Verify 1 of 1 and hover your mouse over the green box and watch it uniquely identify your element in the web site above.

Identify Element Using substring-after With Text

Web Code

```
<p name="first" id="first">First substring-after Element : Dynamic substring-after Text</p>
```

To uniquely identify the *First substring-after Element : Dynamic substring-after Text*, we start with the *//* where there is a *p* tag with a *substring-after* function and a *text* function, searching for the text, *" : "* and then the *substring-after* function searching for the text, *"Dynamic substring-after Text"*.

XPath Code

```
//p[substring-after(text(), " : ")="Dynamic substring-after Text"]
```
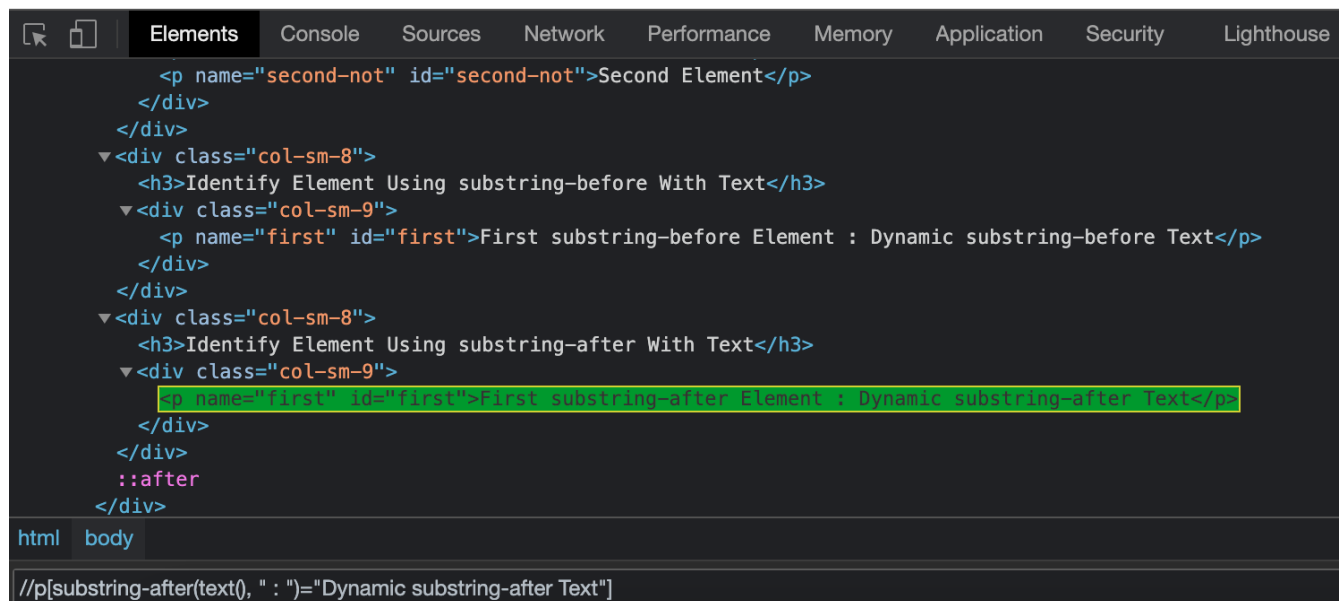

Verify 1 of 1 and hover your mouse over the green box and watch it uniquely identify your element in the web site above.

Congratulations for making all this way! This was an extremely intense lesson and I know it was not easy. The key is to continue to experiment with this knowledge to try new scenarios.

As you continue to practice over the coming weeks this will come more naturally to you. Having a firm understanding of XPath and how to reverse engineer a relative XPath is the most important skill of an Automation Engineer.

Chapter 4: Page Object Model

Now that we have the basics of Selenium as well as DOM reversing we can now discuss a design architecture that can scale.

The *Page Object Model* is a design pattern to properly handle all web elements for an entire site.

For every web page there should be a separate page class.

Open up our framework in Eclipse.

```
RT CLICK src/test/java
New
Package
pages
Finish
```

```
RT CLICK src/test/java
New
Package
test
Finish
```

```
RT CLICK pages
New
Class
SeleniumForEveryonePage
Finish
```

```
RT CLICK test
New
Class
TestSeleniumForEveryonePage
Finish
```

Let's fire up our web server.

```
python -m http.server
```

Let's open our **SeleniumForEveryonePage.java** file and add the following code.

```java
package pages;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;

public class SeleniumForEveryonePage {
	private static WebElement element = null;

	public static WebElement text_box(WebDriver driver) {
		element = driver.findElement(By.name("q"));
		return element;
	}

	public static WebElement button(WebDriver driver) {
		element = driver.findElement(By.className("button"));
		return element;
	}
}
```

Let's open our **TestSeleniumForEveryonePage.java** file and add the following code and run.

```java
package test;

import org.openqa.selenium.By;
import org.openqa.selenium.Keys;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

import pages.SeleniumForEveryonePage;

public class TestSeleniumForEveryonePage {
	private static WebDriver driver = null;

	public static void main(String[] args) {
		testSeleniumForEveryone();
	}

	public static void testSeleniumForEveryone() {
		String projectPath = System.getProperty("user.dir");
		System.setProperty("webdriver.chrome.driver", projectPath +
"//drivers/chromedriver/chromedriver.exe");
		driver = new ChromeDriver();
		driver.get("http://localhost:8000");
		SeleniumForEveryonePage.text_box(driver).sendKeys("coffee");
		SeleniumForEveryonePage.button(driver).sendKeys(Keys.RETURN);
		driver.quit();
	}
}
```

CTRL+F11

Here we have a very basic POM automation!

Let's refactor a bit to better scale our architecture.

RT CLICK pages
New
Class
SeleniumForEveryonePageObjects
Finish

The advantage is whenever you have to make changes to your locators, you go to ONE place.

Let's open our **SeleniumForEveryonePageObjects.java** file and add the following code.

```java
package pages;

import org.openqa.selenium.By;
import org.openqa.selenium.Keys;
import org.openqa.selenium.WebDriver;

public class SeleniumForEveryonePageObjects {
	WebDriver driver = null;

	By text_box = By.name("q");
	By button = By.className("button");

	public SeleniumForEveryonePageObjects(WebDriver driver) {
		this.driver = driver;
	}

	public void setTextInTextBox(String text) {
		driver.findElement(text_box).sendKeys(text);
	}

	public void clickButton() {
		driver.findElement(button).sendKeys(Keys.RETURN);
	}
}
```

Let's edit our **TestSeleniumForEveryonePageObjects.java** file and update the following code and run.

```java
package test;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

import pages.SeleniumForEveryonePageObjects;

public class TestSeleniumForEveryonePage {
    private static WebDriver driver = null;

    public static void main(String[] args) {
        testSeleniumForEveryone();
    }

    public static void testSeleniumForEveryone() {
        String projectPath = System.getProperty("user.dir");
        System.setProperty("webdriver.chrome.driver", projectPath +
"//drivers/chromedriver/chromedriver.exe");
        driver = new ChromeDriver();
        SeleniumForEveryonePageObjects seleniumForEveryonePageObject = new
SeleniumForEveryonePageObjects(driver);
        driver.get("http://localhost:8000");
        seleniumForEveryonePageObject.setTextInTextBox("coffee");
        seleniumForEveryonePageObject.clickButton();
        driver.quit();
    }
}
```

CTRL+F11

Hooray! We see our automation work as we did earlier but now we have a cleaner and better designed API.

Chapter 5: TestNG

We are at the final stage now! We are going to implement the TestNG framework to now write our tests.

Open up our framework in Eclipse.

CLICK Help
Install New Software
Add...
TestNG
https://testng.org/testng-eclipse-update-site
Check ALL
Next
Accept License
Finish
Restart Eclipse

Let's fire up our web server.

python -m http.server

Let's open our **TestSeleniumForEveryonePage.java** file and update the following code.

```java
package test;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.Assert;
import org.testng.annotations.AfterTest;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;

import pages.SeleniumForEveryonePageObjects;

public class TestSeleniumForEveryonePage {
	WebDriver driver = null;

	@BeforeTest
	public void setUpTest() {
		String projectPath = System.getProperty("user.dir");
		System.setProperty("webdriver.chrome.driver", projectPath +
"//drivers/chromedriver/chromedriver.exe");
		driver = new ChromeDriver();
	}

	@Test
	public void testSeleniumForEveryoneTitle() {
		String expectedTitle;
		String actualTitle;
		driver.get("http://localhost:8000");
```

```
            expectedTitle = "SFE";
            actualTitle = driver.getTitle();
            Assert.assertEquals(actualTitle, expectedTitle);
      }

      @Test
      public void testSeleniumForEveryoneSearchPageTitle() {
            String expectedTitle;
            String actualTitle;
            SeleniumForEveryonePageObjects seleniumForEveryonePageObject = new
SeleniumForEveryonePageObjects(driver);
            driver.get("http://localhost:8000");
            seleniumForEveryonePageObject.setTextInTextBox("coffee");
            seleniumForEveryonePageObject.clickButton();
            for(String winHandle : driver.getWindowHandles()) {
                driver.switchTo().window(winHandle);
            }
            expectedTitle = "coffee site:google.com - Google Search";
            actualTitle = driver.getTitle();
            Assert.assertEquals(actualTitle, expectedTitle);
      }

      @AfterTest
      public void tearDownTest() {
            driver.quit();
            System.out.println("Test Completed Successfully");
      }
}
```

CTRL+F11

Hooray! Now we have two simple tests written.

```
Test Completed Successfully
PASSED: testSeleniumForEveryoneSearchPageTitle
PASSED: testSeleniumForEveryoneTitle

===============================================
    Default test
    Tests run: 2, Failures: 0, Skips: 0
===============================================




===============================================
Default suite
Total tests run: 2, Passes: 2, Failures: 0, Skips: 0
===============================================
```

At this point we have a firm understanding of how to reverse engineer
the DOM and use the Page Object Model to scale our Front-End
Automation and finally we have our testing framework in place.

To see the test report.

RT CLICK SeleniumFramework
Refresh
test-output
RT CLICK index.html
Open With
Web Browser

Now we have a professional test report!

Thank you for taking this journey with me and learning the basics of Front-End Automation with Java and Selenium. At this point you can build out this test page and then build your own framework within your organization.