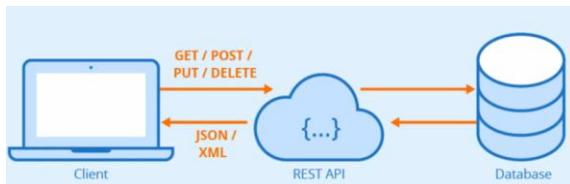


Test Automation using:

- Python
- Playwright
- PyCharm
- PyTest
- GitHub repository

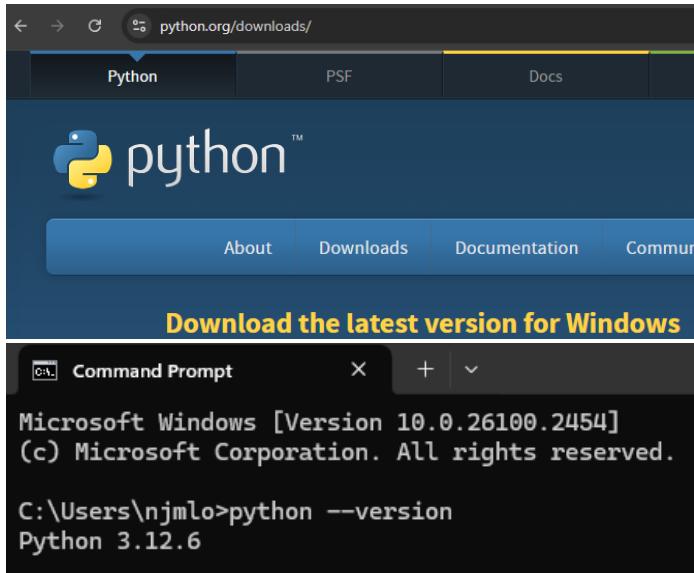
Document highlight:

- Tools download and setup
- Working on a web page:
 - Identifying element using css locators
 - Identifying element using XPath locators
 - Selecting values from dropdowns
 - Selecting radio buttons and checkboxes
 - Alerts and dialog box
 - Windows/page handling
 - Cookies and taking page screenshots
 - Page interactions
 - Exception handling and storing elements in lists
 - Ajax handling
 - Page tables
 - Upload and download of files
 - Page desktop video recording and screenshots
 - Web page with Playwright and PyTest
 - Page with fixtures (module and functions)
 - Page with fixtures (class and functions)
 - Parameterizing tests in PyTest with Playwright
 - Test reporting in PyTest with Playwright (PyTest HTML plugin)

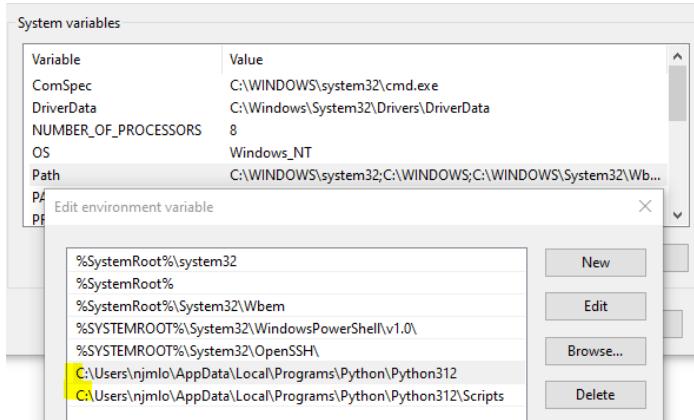
1. API Concept (interface between client and server (database), request and response)**2. Tools download and setup**

Python download and setup:

<https://www.python.org/downloads/>

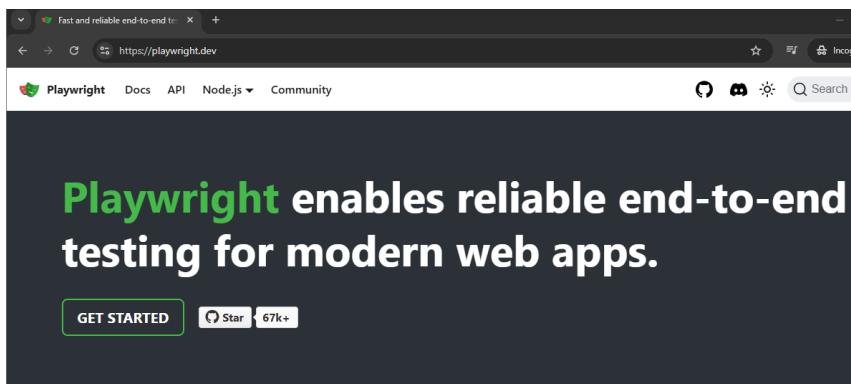


In machine's system variable path:

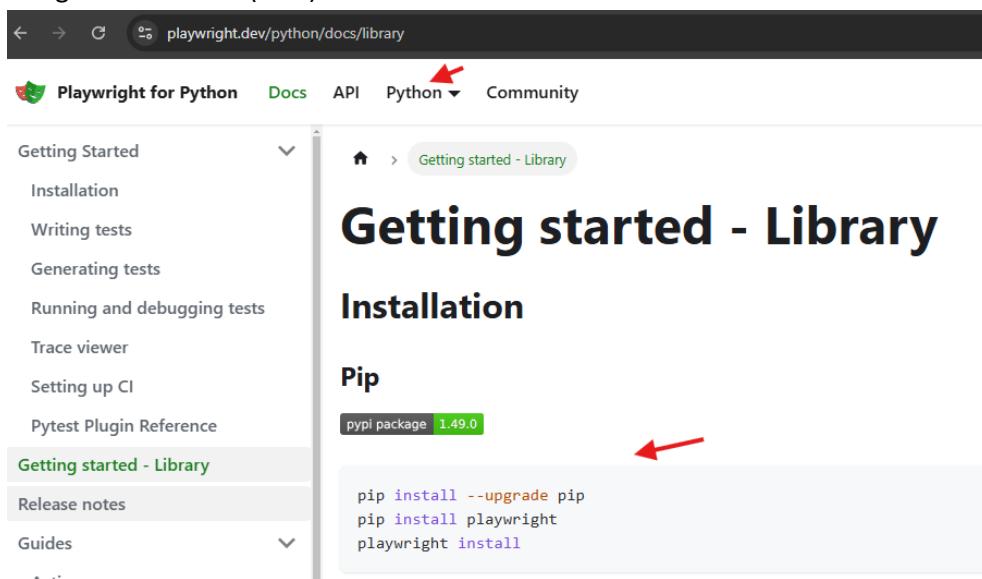


Playwright download and setup:

<https://playwright.dev/>



Using the command (cmd) line:



The screenshot shows the official Playwright documentation website for Python. The navigation bar at the top includes links for 'Playwright for Python', 'Docs', 'API', 'Python' (which has a red arrow pointing to it), and 'Community'. The main content area is titled 'Getting started - Library' and contains sections for 'Installation', 'Pip', and 'FFmpeg'. In the 'Pip' section, there is a code block with a red arrow pointing to the command 'pip install playwright'. Below this, a terminal window shows the execution of various pip commands to install Playwright and its dependencies.

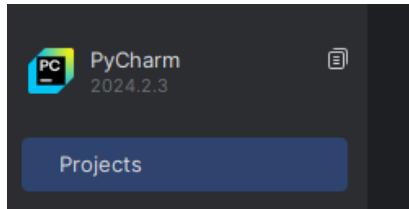
```
PS C:\WINDOWS\system32> pip --version
pip 24.3.1 from C:\Users\njmlo\AppData\Loc
PS C:\WINDOWS\system32> pip install playwright
Collecting playwright
  Downloading playwright-1.49.0-py3-none-win_amd64.whl.metadata (3.5 kB)
Collecting greenlet==3.1.1 (from playwright)
  Downloading greenlet-3.1.1-cp312-cp312-win_amd64.whl.metadata (3.9 kB)
Collecting pyee==12.0.0 (from playwright)
  Downloading pyee-12.0.0-py3-none-any.whl.metadata (2.8 kB)
Requirement already satisfied: typing-extensions in c:\users\njmlo\appdata\loc
ages (from pyee==12.0.0->playwright) (4.12.2)
Downloading playwright-1.49.0-py3-none-win_amd64.whl (34.0 MB)
----- 34.0/34.0 MB 36.0 MB/s eta 0:00:00
Downloading greenlet-3.1.1-cp312-cp312-win_amd64.whl (299 kB)
Downloading pyee-12.0.0-py3-none-any.whl (14 kB)
Installing collected packages: pyee, greenlet, playwright
Successfully installed greenlet-3.1.1 playwright-1.49.0 pyee-12.0.0
PS C:\WINDOWS\system32> playwright --version
Version 1.49.0

PS C:\WINDOWS\system32> playwright install
Downloading Chromium 131.0.6778.33 (playwright build v1148) from https://playwright.azureedge.net/builds/chromium/1148/c
chromium-win64.zip
136.9 MiB [=====] 100% 0.0s
Chromium 131.0.6778.33 (playwright build v1148) downloaded to C:\Users\njmlo\AppData\Local\ms-playwright\chromium-1148
Downloading Chromium Headless Shell 131.0.6778.33 (playwright build v1148) from https://playwright.azureedge.net/builds/
chromium/1148/chromium-headless-shell-win64.zip
87.7 MiB [=====] 100% 0.0s
Chromium Headless Shell 131.0.6778.33 (playwright build v1148) downloaded to C:\Users\njmlo\AppData\Local\ms-playwright\
chromium_headless_shell-1148
Downloading Firefox 132.0 (playwright build v1466) from https://playwright.azureedge.net/builds/firefox/1466/firefox-win
64.zip
85.8 MiB [=====] 100% 0.0s
Firefox 132.0 (playwright build v1466) downloaded to C:\Users\njmlo\AppData\Local\ms-playwright\firefox-1466
Downloading Webkit 18.2 (playwright build v2104) from https://playwright.azureedge.net/builds/webkit/2104/webkit-win64.z
ip
52.7 MiB [=====] 100% 0.0s
Webkit 18.2 (playwright build v2104) downloaded to C:\Users\njmlo\AppData\Local\ms-playwright\webkit-2104
Downloading FFmpeg playwright build v1010 from https://playwright.azureedge.net/builds/ffmpeg/1010/ffmpeg-win64.zip
1.3 MiB [=====] 100% 0.0s
FFmpeg playwright build v1010 downloaded to C:\Users\njmlo\AppData\Local\ms-playwright\ffmpeg-1010
PS C:\WINDOWS\system32>
```

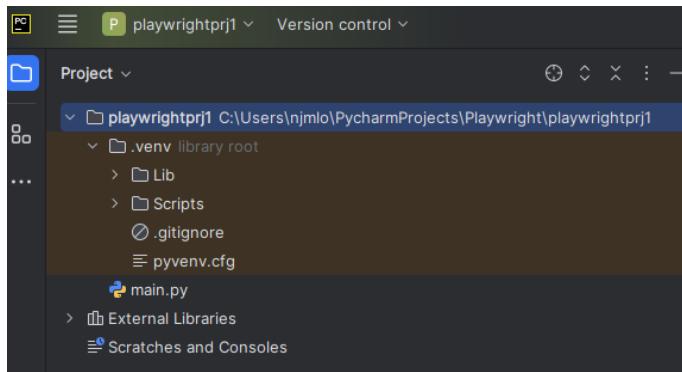
C:\Users\njmlo>playwright --version
Version 1.49.0

PyCharm download and setup:

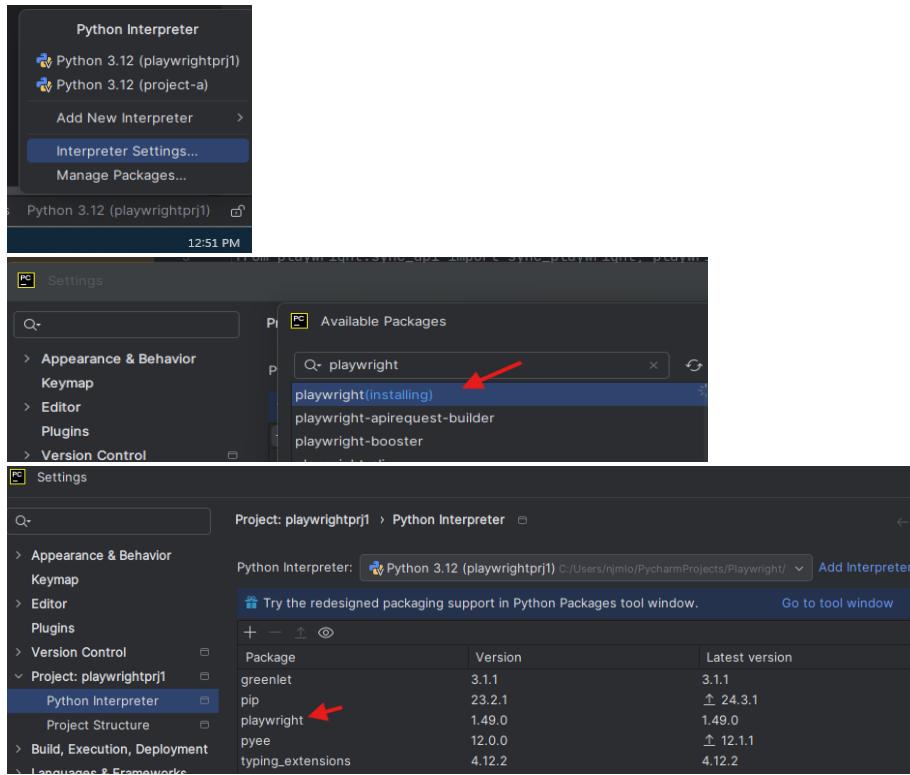
<https://www.jetbrains.com/pycharm/download>



Create a project in PyCharm:



In the interpreter settings, install playwright package:



After all the installation and setup, run a quick test:

The screenshot shows the PyCharm IDE interface. On the left is the Project tool window displaying a file structure for a project named 'playwrightprj1'. In the center is the code editor with a file named 'chrome_browser_launch.py' containing Python code for launching a browser and navigating to a URL. On the right is a context menu with various options like 'Show Context Actions', 'AI Actions', 'Paste', 'Column Selection Mode', 'Find Usages', 'Go To', 'Folding', 'Rename...', 'Refactor', 'Generate...', and 'Run "chrome_browser_launch..."'. A red arrow points to the 'Generate...' option. Below the code editor is the Run tool window showing the output of the script: 'Google' and 'Success!', with 'Success!' highlighted by a red box. At the bottom, it says 'Process finished with exit code 0'.

```
# == import and install files and libraries ==
from playwright.sync_api import sync_playwright

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False)
    page = browser.new_page()
    page.goto("https://www.google.ca")
    # other actions...
    print(page.title())
    print('Success!')
    page.wait_for_timeout(3000)
    browser.close()
```

3. Working on a web page identifying element using css locators

Test page url:

<https://demo.automationtesting.in/>

Locate (inspect) the element attributes for use as its locator:

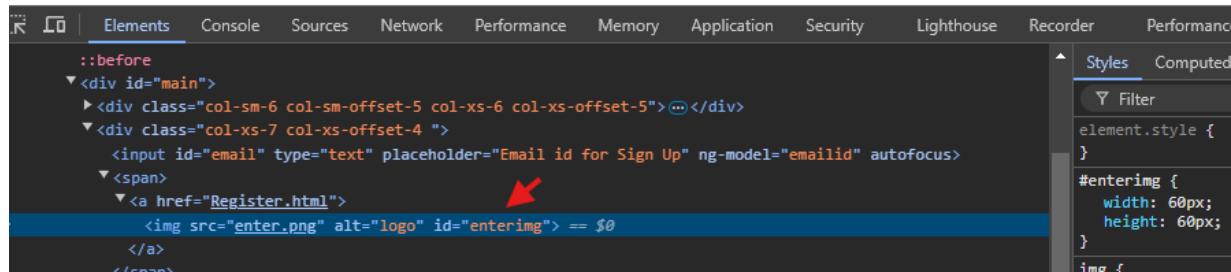
Email textbox...



A screenshot of the browser's developer tools showing the DOM tree under the 'Elements' tab. A red arrow points to the 'email' input element in the tree. The element has the following attributes: id='email', type='text', placeholder='Email id for Sign Up', ng-model='emailid', and autofocus. The browser's address bar shows the URL 'https://demo.automationtesting.in/'.

```
<!DOCTYPE html>
<html lang="en">
  <head> ...
  </head>
  <body ng-app="Register-form" ng-controller="FormCtrl">
    <div id="abcd"> ...
      <div class="container">
        ...
        <div class="row">
          ...
          <div id="main">
            <div class="col-sm-6 col-sm-offset-5 col-xs-6 col-xs-offset-5"> ...
              <div class="col-xs-7 col-xs-offset-4">
                ...
                <input id="email" type="text" placeholder="Email id for Sign Up" ng-model="emailid" autofocus> == $0
                  <span> ...
                    </span>
              </div>
            </div>
          </div>
        </div>
      </div>
    </div>
  </body>
</html>
```

Login button...

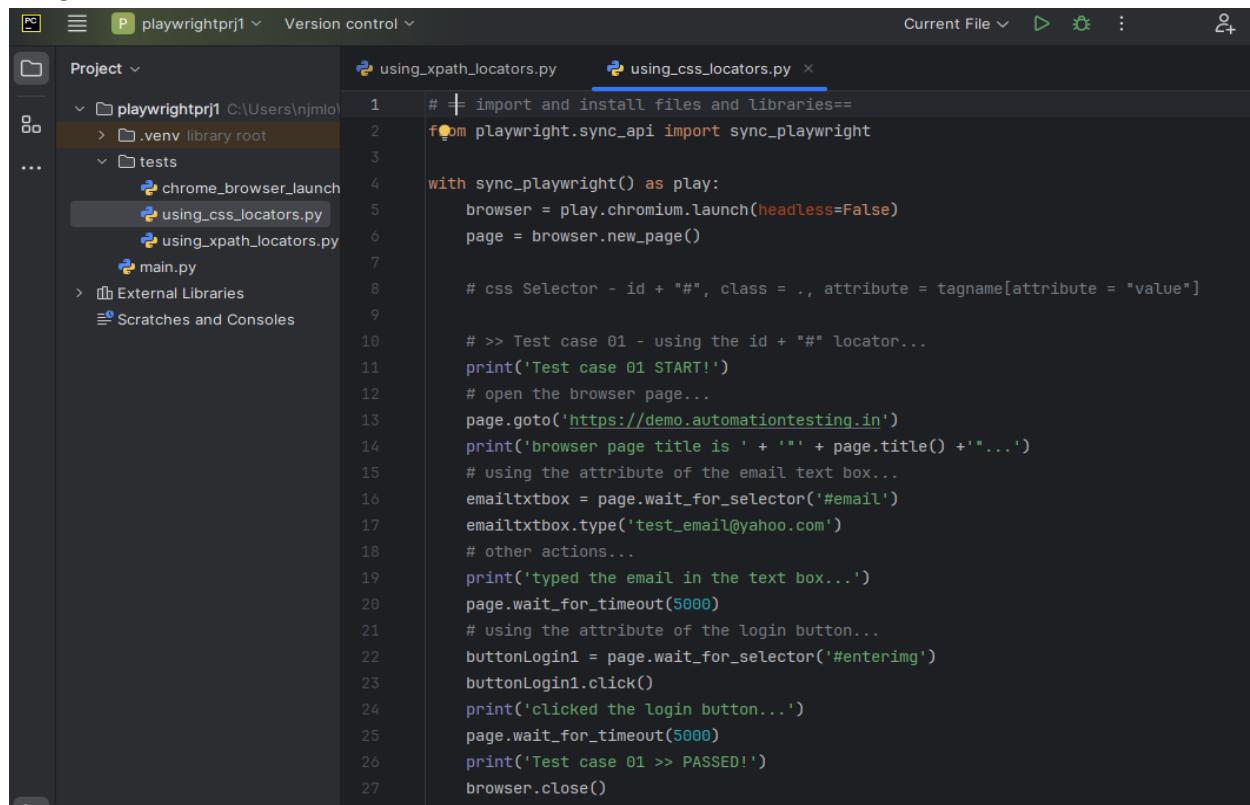


```
Elements | Console | Sources | Network | Performance | Memory | Application | Security | Lighthouse | Recorder | Performance
::before
<div id="main">
  <div class="col-sm-6 col-sm-offset-5 col-xs-6 col-xs-offset-5">...</div>
  <div class="col-xs-7 col-xs-offset-4 ">
    <input id="email" type="text" placeholder="Email id for Sign Up" ng-model="emailid" autofocus>
    <span>
      <a href="Register.html">
         == $0
      </a>
    </span>
  </div>
```

Syntax:

a. (Test case 01) using the id + "#"

Using the "id" locator, create and run the code:



```
Project ▾
  playwrightpr1 C:\Users\njml0
    .env library root
    tests
      chrome_browser_launch
      using_css_locators.py
      using_xpath_locators.py
      main.py
    External Libraries
    Scratches and Consoles
```

```
using_css_locators.py
1 # + import and install files and libraries=
2 from playwright.sync_api import sync_playwright
3
4 with sync_playwright() as play:
5   browser = play.chromium.launch(headless=False)
6   page = browser.new_page()
7
8   # css Selector - id + "#", class = ., attribute = tagname[attribute = "value"]
9
10  # >> Test case 01 - using the id + "#" locator...
11  print('Test case 01 START!')
12  # open the browser page...
13  page.goto('https://demo.automationtesting.in')
14  print('browser page title is ' + '' + page.title() + '...')
15  # using the attribute of the email text box...
16  emailtxtbox = page.wait_for_selector('#email')
17  emailtxtbox.type('test_email@yahoo.com')
18  # other actions...
19  print('typed the email in the text box...')
20  page.wait_for_timeout(5000)
21  # using the attribute of the login button...
22  buttonLogin1 = page.wait_for_selector('#enterimg')
23  buttonLogin1.click()
24  print('clicked the login button...')
25  page.wait_for_timeout(5000)
26  print('Test case 01 >> PASSED!')
27  browser.close()
```

```

Run  using_css_locators (1) ×

C:\Users\njml0\PycharmProjects\Playwright\playwrightprj1\.venv\Scripts\python.exe C:\Users\njml0\PycharmProjects\Playwrightprj1\tests\test_login.py
Test case 01 START!
browser page title is "Index"...
typed the email in the text box...
clicked the login button...
Test case 01 >> PASSED!

Process finished with exit code 0

```

Test page url:

<https://opensource-demo.orangehrmlive.com/web/index.php/auth/login>

Locate (inspect) the element attributes for use as its locator:

Username text box...

The screenshot shows the OrangeHRM login page. The URL in the address bar is <https://opensource-demo.orangehrmlive.com/web/index.php/auth/login>. The page title is 'Login'. There is a grey info box with 'Username : Admin' and 'Password : admin123'. Below it is a form with 'Username' and 'Password' inputs. A red arrow points to the 'Username' input. At the bottom is a large orange 'Login' button. The browser's developer tools are open, with the 'Elements' tab selected. The DOM tree shows the HTML structure. A red arrow points to the 'input' element under the 'username' placeholder. The right panel shows the element's styles, including a CSS rule for the active state:

```
.oxd-input--active{border: 1px solid #e8eaef;}
```

.

Password text box...

The screenshot shows a login interface with a header "Login". Below it is a form containing fields for "Username" and "Password". The "Password" field is highlighted with a red arrow pointing to its input box. The background features an orange and white design with a circular logo on the right.

Username : Admin
Password : admin123

Username

>Password

Login

Elements Console Sources Network Performance Memory Application Security Lighthouse Recorder Performance insights

```
</div>
<div data-v-2130bd2a data-v-0af708be class="oxd-form-row">
  <div data-v-957b4417 data-v-0af708be class="oxd-input-group oxd-input-field-bottom-space">
    <div data-v-957b4417 class="oxd-input-group__label-wrapper"></div>
    <div data-v-957b4417 class="oxd-input-group__input">
      <input data-v-1f99f73c class="oxd-input oxd-input--active" type="password" name="password" placeholder="Password" />
    </div>
  </div>
</div>
```

Styles Computed Layout Event Listeners

```
element.style { }
.oxd-input--active[data-v-1f99f73c] {
  border: 1px solid #e8eaef;
}
```

Login button...

The screenshot shows the same login interface as the previous one, but now the "Login" button is highlighted with a red arrow pointing to its center. The background design remains the same.

Username : Admin
Password : admin123

Username

Password

Login

Elements Console Sources Network Performance Memory Application Security Lighthouse Recorder Performance insights

```
<input data-v-1f99f73c class="oxd-input oxd-input--active" type="password" name="password" placeholder="Password" />
</div>
<!-->
</div>
</div>
<div data-v-19c2496b data-v-0af708be class="oxd-form-actions orangehrm-login-action">
  <button data-v-10d463b7 data-v-0af708be type="submit" class="oxd-button oxd-button--medium oxd-button--main orangehrm-login-button">Login</button>
</div>
```

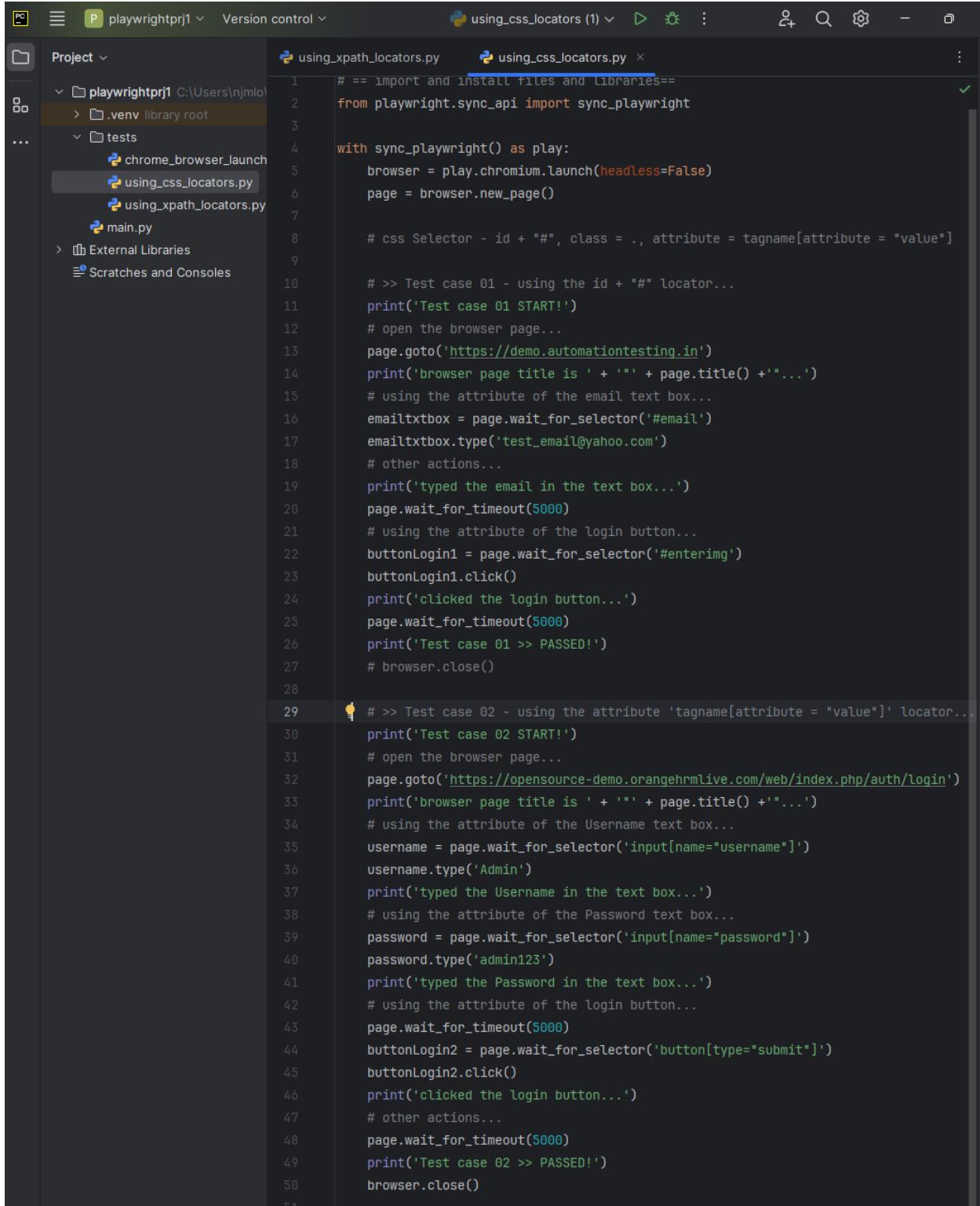
Styles Computed Layout Event Listeners

```
element.style { }
.orangehrm-login-button[data-v-0af708be] {
  flex: 1;
  padding: 1rem 0;
}
```

Syntax:

b. (Test case 02) use the attribute 'tagname[attribute = "value"]'

Using the “attribute” locator, create and run the code:



The screenshot shows a code editor interface with a dark theme. On the left is a project tree for 'playwrightprj1' containing files like 'main.py', 'chrome_browser_launch.py', 'using_css_locators.py', and 'using_xpath_locators.py'. The main pane displays two Python scripts: 'using_css_locators.py' and 'using_xpath_locators.py'. The 'using_css_locators.py' script is active and contains code demonstrating various locators, including the 'tagname[attribute = "value"]' locator used in Test Case 02. The code includes comments explaining the steps and prints the browser title at each step.

```
# == import and install files and libraries ==
from playwright.sync_api import sync_playwright

with sync_playwright() as play:
    browser = play.chromium.launch(headless=False)
    page = browser.new_page()

    # css Selector - id + "#", class = ., attribute = tagname[attribute = "value"]
    # >> Test case 01 - using the id + "#" locator...
    print('Test case 01 START!')
    # open the browser page...
    page.goto('https://demo.automationtesting.in')
    print('browser page title is ' + ' ' + page.title() + ' ')
    # using the attribute of the email text box...
    emailtxtbox = page.wait_for_selector('#email')
    emailtxtbox.type('test_email@yahoo.com')
    # other actions...
    print('typed the email in the text box...')
    page.wait_for_timeout(5000)
    # using the attribute of the login button...
    buttonLogin1 = page.wait_for_selector('#enterimg')
    buttonLogin1.click()
    print('clicked the login button...')
    page.wait_for_timeout(5000)
    print('Test case 01 >> PASSED!')
    # browser.close()

# >> Test case 02 - using the attribute 'tagname[attribute = "value"]' locator...
print('Test case 02 START!')
# open the browser page...
page.goto('https://opensource-demo.orangehrmlive.com/web/index.php/auth/login')
print('browser page title is ' + ' ' + page.title() + ' ')
# using the attribute of the Username text box...
username = page.wait_for_selector('input[name="username"]')
username.type('Admin')
print('typed the Username in the text box...')
# using the attribute of the Password text box...
password = page.wait_for_selector('input[name="password"]')
password.type('admin123')
print('typed the Password in the text box...')
# using the attribute of the login button...
page.wait_for_timeout(5000)
buttonLogin2 = page.wait_for_selector('button[type="submit"]')
buttonLogin2.click()
print('clicked the login button...')
# other actions...
page.wait_for_timeout(5000)
print('Test case 02 >> PASSED!')
browser.close()
```

```
Run using_css_locators x
C:\Users\njmlo\PycharmProjects\Playwright\playwrightprj1\.venv\Scripts\python.exe C:\Users\njmlo\PycharmProjects\Playwright\playwrightprj1\src\tests\test_login.py
Test case 01 START!
browser page title is "Index"...
typed the email in the text box...
clicked the login button...
Test case 01 > PASSED!
Test case 02 START!
browser page title is "OrangeHRM"...
typed the Username in the text box...
typed the Password in the text box...
clicked the login button...
Test case 02 > PASSED!
Process finished with exit code 0
```

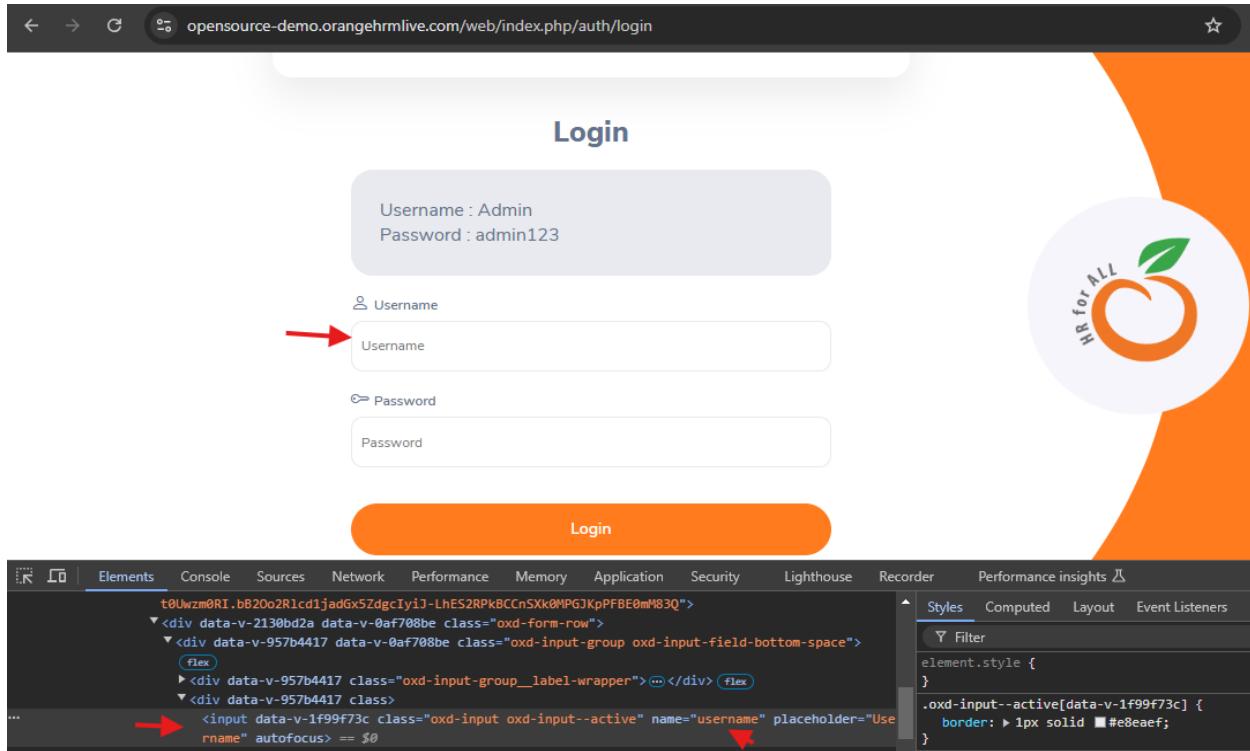
4. Working on a web page identifying element using XPath locators

Test page url:

<https://opensource-demo.orangehrmlive.com/web/index.php/auth/login>

Locate (inspect) the element attributes for use as its XPath locator

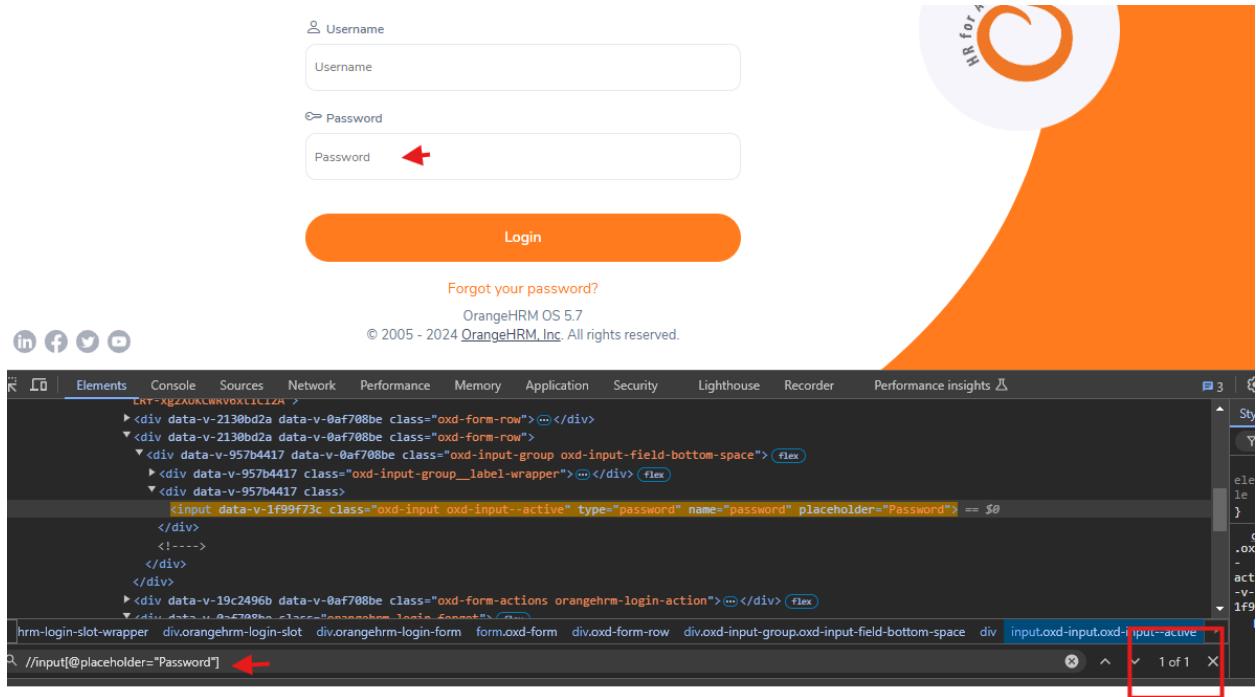
Username text box...



The screenshot shows a browser window displaying the OrangeHRM login page. The page has a header 'Login' and two input fields: 'Username' and 'Password'. A red arrow points to the 'Username' input field. Below the page, the browser's developer tools are open, specifically the 'Elements' tab. The DOM tree shows the structure of the login form. A red arrow points to the 'input' element under the 'username' placeholder in the DOM tree. The right panel of the developer tools shows the 'Styles' tab with CSS rules applied to the element.

```
<input data-v-1f99f73c class="oxd-input oxd-input--active" name="username" placeholder="Username" autofocus>
```

Password text box...

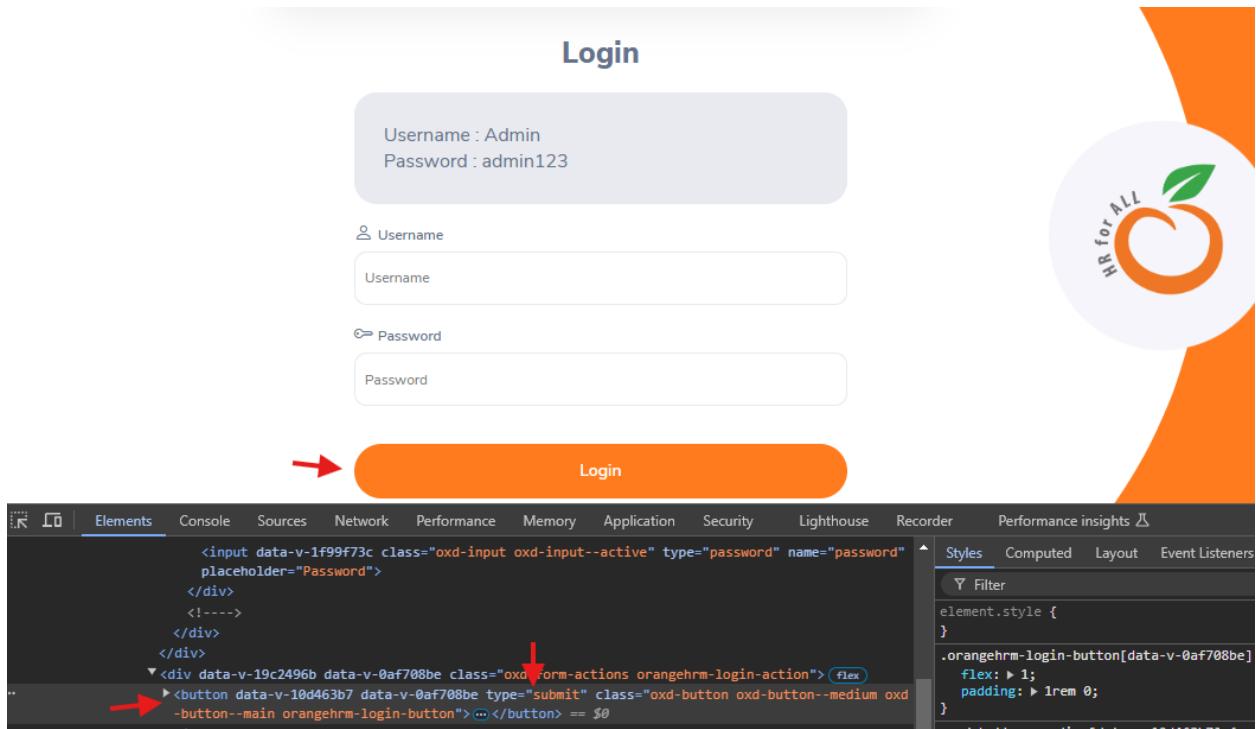


The screenshot shows the OrangeHRM login page. A red arrow points to the 'Password' input field, which contains the placeholder text 'Password'. The browser's developer tools are open, with the 'Elements' tab selected. The search bar in the developer tools also contains the query '/input[@placeholder="Password"]', with another red arrow pointing to it. The results list shows the password input field highlighted with a red border.

```
<input data-v-1f99f73c class="oxd-input oxd-input--active" type="password" name="password" placeholder="Password"> == $0
```

```
//input[@placeholder="Password"]
```

Login button...

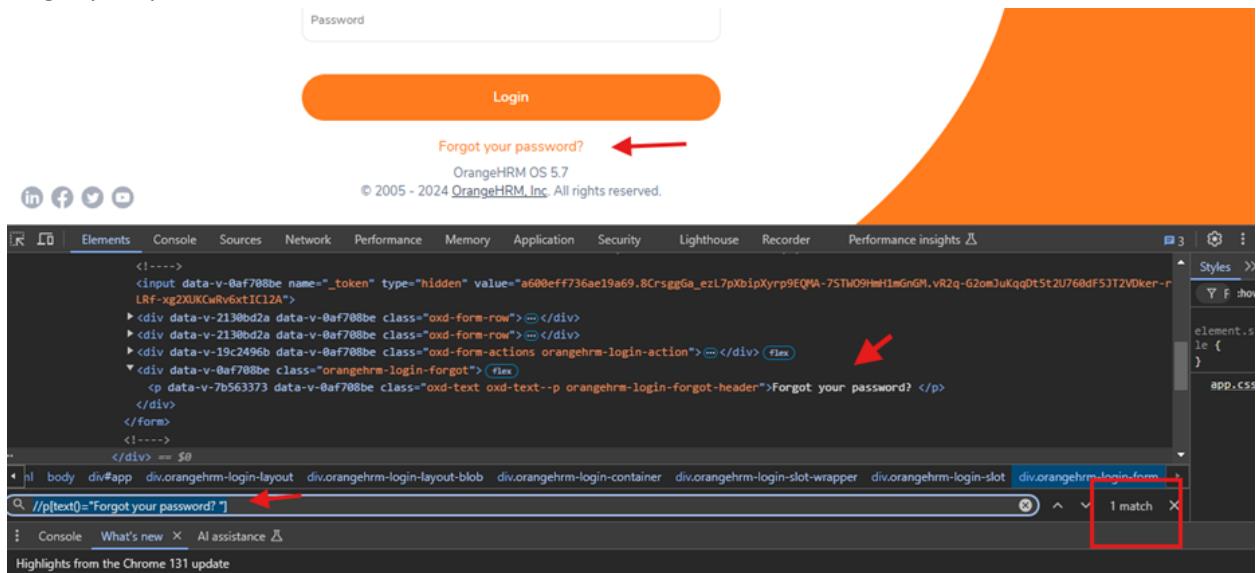


The screenshot shows the OrangeHRM login page with a red arrow pointing to the large orange 'Login' button at the bottom center. The browser's developer tools are open, with the 'Elements' tab selected. A red arrow also points to the 'orangehrm-login-button' class in the styles panel, indicating the CSS selector used for the button.

```
.orangehrm-login-button[data-v-0af708be] { flex: 1; padding: 1rem 0; }
```

```
<button data-v-10d463b7 data-v-0af708be type="submit" class="oxd-button oxd-button--medium oxd-button--main orangehrm-login-button"> == $0
```

Forgot your password link:



Syntax:

- (Test case 01) use the attribute - '//tagname[@name="value"]'
- (Test case 02) use the visible text attribute - '//tagname[text()="text"]'

Using the XPath locator, create and run the code:

```

# == import and install files and libraries ==
from playwright.sync_api import sync_playwright

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False)
    page = browser.new_page()

    # Note that for the XPath, Relative path is used because of its flexibility vs Absolute path

    # >> Test case 01 - use the attribute - '//tagname[@name="value"]'
    print('Test case 01 START!')
    # open the browser page...
    page.goto('https://opensource-demo.orangehrmlive.com/web/index.php/auth/login')
    print('browser page title is ' + "'" + page.title() + "'")
    # using the attribute of the Username text box...
    username_xpath = page.wait_for_selector('//input[@name="username"]')
    username_xpath.type('Admin')
    print('typed the Username in the text box...')
    # using the attribute of the Password text box...
    password_xpath = page.wait_for_selector('//input[@placeholder="Password"]')
    password_xpath.type('admin123')
    print('typed the Password in the text box...')
    # using the attribute of the login button...
    page.wait_for_timeout(5000)
    buttonSubmit_xpath = page.wait_for_selector('//button[@type="submit"]')
    buttonSubmit_xpath.click()
    print('clicked the login button...')
    # other actions...
    page.wait_for_timeout(5000)
    print('Test case 01 >> PASSED!')
    # browser.close()

```

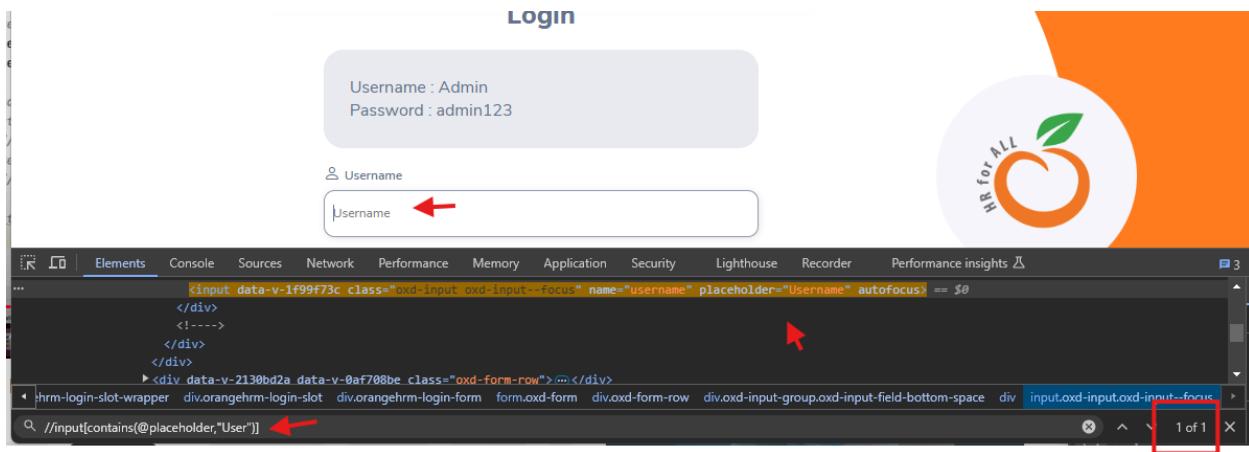
```
    # browser.close()
32
33     # >> Test case 02 - use the visible text attribute - '//tagname[text()="text"]'
34     print('Test case 02 START!')
35     # open the browser page...
36     page = browser.new_page()
37     page.goto('https://opensource-demo.orangehrmlive.com/web/index.php/auth/login')
38     print('browser page title is ' + "'" + page.title() + "'")
39     # using the attribute of the Username text box...
40     page.wait_for_timeout(5000)
41     page.wait_for_selector('//p[text()="Forgot your password? "]').click()
42     print('clicked the "Forgot your password?" link...')
43     # other actions...
44     page.wait_for_timeout(5000)
45     print('Test case 02 >> PASSED!')
46     browser.close()
```

```
Run using_xpath_locators x

C:\Users\njmlo\PycharmProjects\Playwright\playwrightprj1\.venv\Scripts\python.exe C:\Users\njmlo\PycharmProjects\Playwright\playwrightprj1\tests\test_login.py
Test case 01 START!
browser page title is "OrangeHRM"...
typed the Username in the text box...
typed the Password in the text box...
clicked the login button...
Test case 01 >> PASSED!
Test case 02 START!
browser page title is "OrangeHRM"...
clicked the "Forgot your password?" link...
Test case 02 >> PASSED!

Process finished with exit code 0
```

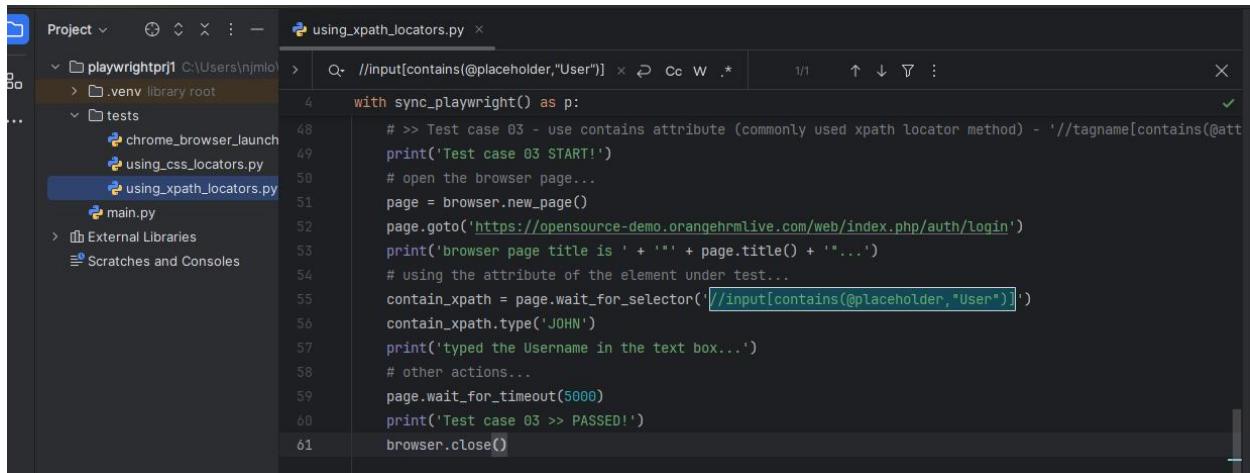
Username text box...



Syntax:

c. (Test case 03) use contains attribute (commonly used xpath locator method) -
`'//tagname[contains(@attribute, "value")]'`

Using the XPath locator, create and run the code:



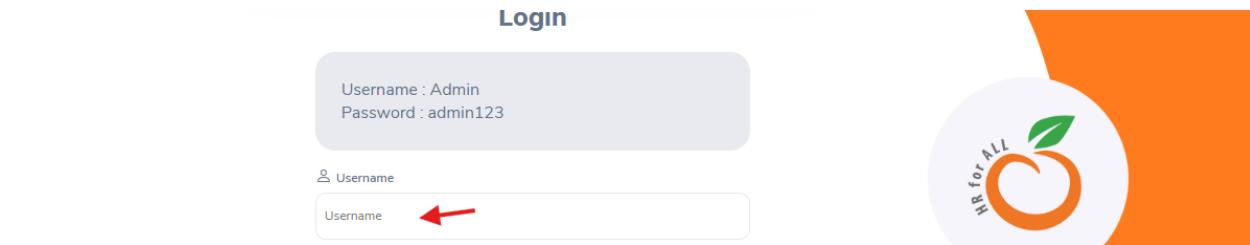
```
Project > using_xpath_locators.py > //input[contains(@placeholder,"User")] > C:\Users\njml0\playwrightprj1\.venv\library root > tests > using_css_locators.py > using_xpath_locators.py > main.py > External Libraries > Scratches and Consoles
```

```
4 with sync_playwright() as p:  
# >> Test case 03 - use contains attribute (commonly used xpath locator method) - '//tagname[contains(@attribute, "value")]'  
    print('Test case 03 START!')  
    # open the browser page...  
    page = browser.new_page()  
    page.goto('https://opensource-demo.orangehrmlive.com/web/index.php/auth/login')  
    print('browser page title is ' + "'" + page.title() + "'...')  
    # using the attribute of the element under test...  
    contain_xpath = page.wait_for_selector('//input[contains(@placeholder, "User")]')  
    contain_xpath.type('JOHN')  
    print('typed the Username in the text box...')  
    # other actions...  
    page.wait_for_timeout(5000)  
    print('Test case 03 >> PASSED!')  
    browser.close()
```



```
Test case 03 START!  
browser page title is "OrangeHRM"...  
typed the Username in the text box...  
Test case 03 >> PASSED!  
  
Process finished with exit code 0
```

Username text box...

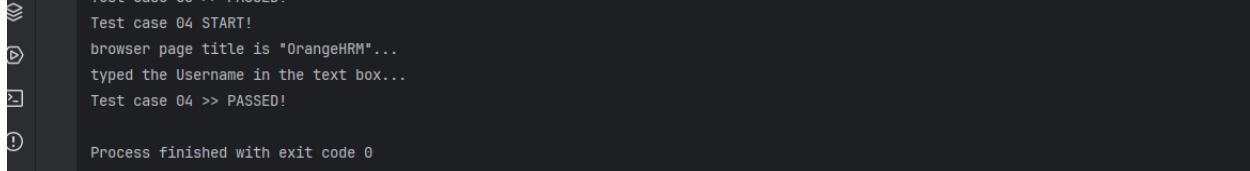


The screenshot shows the OrangeHRM login page. A red arrow points to the 'Username' input field. The browser's developer tools Elements tab is open, showing the HTML structure of the page. The search bar in the developer tools has the XPath expression `//input[starts-with(@placeholder,"User")]`. The status bar at the bottom right of the developer tools window shows '1 of 1'.

Syntax:

c. (Test case 04) use contains attribute for starts-with/ends-with (used for dynamic or array of data with common value) - `//tagname[starts-with(@attribute, "value")]`

Using the XPath locator, create and run the code:



The terminal window shows the output of the script execution:

```
Test case 04 >> PASSED!
Test case 04 START!
browser page title is "OrangeHRM"...
typed the Username in the text box...
Test case 04 >> PASSED!
Process finished with exit code 0
```

Samples of common XPath locators:

- `/bookstore/book` - Selects all book elements in the root bookstore.
- `//title[text()='XPath']` - Selects title elements with text 'XPath' anywhere in the document.
- `//*[@id='myId']` - Selects elements with the attribute id equal to 'myId'.
- `/bookstore/book[position()=1]` - Selects the first book element.
- `//div[@class='highlight']//p` - Selects p elements within div with class 'highlight'.
- `//a[contains(@href, 'example.com')]` - Selects a elements with 'example.com' in the href attribute.

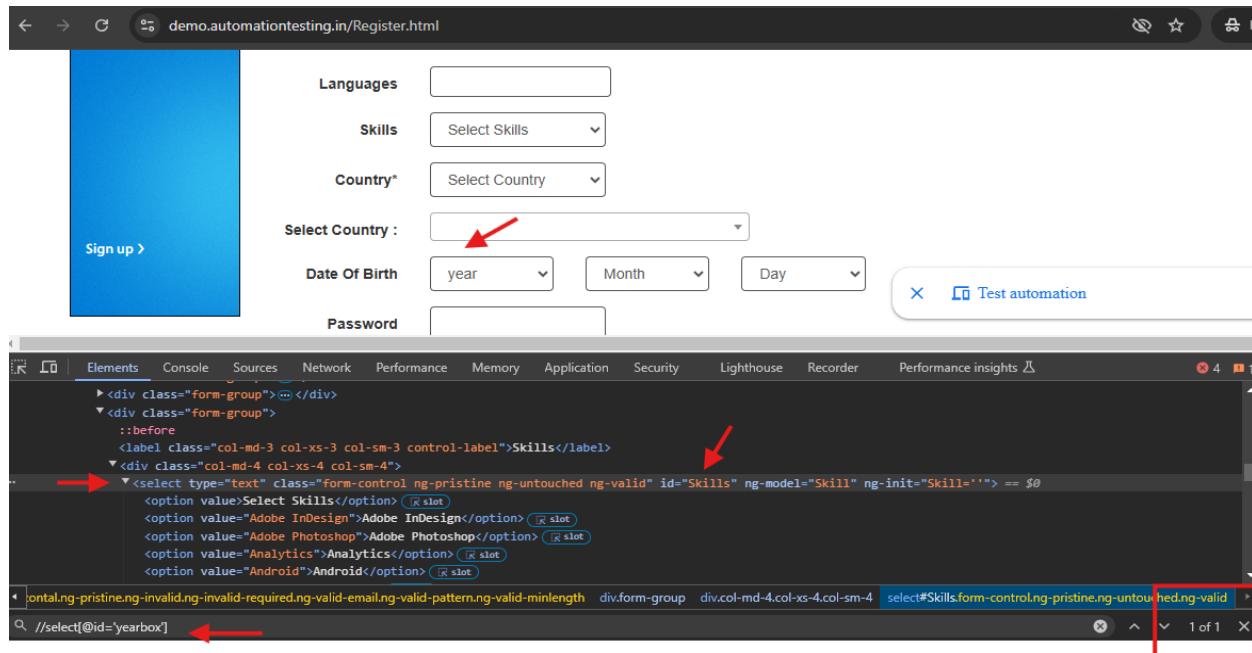
5. Working on a web page in selecting values from dropdowns

Test page url:

<https://demo.automationtesting.in/>

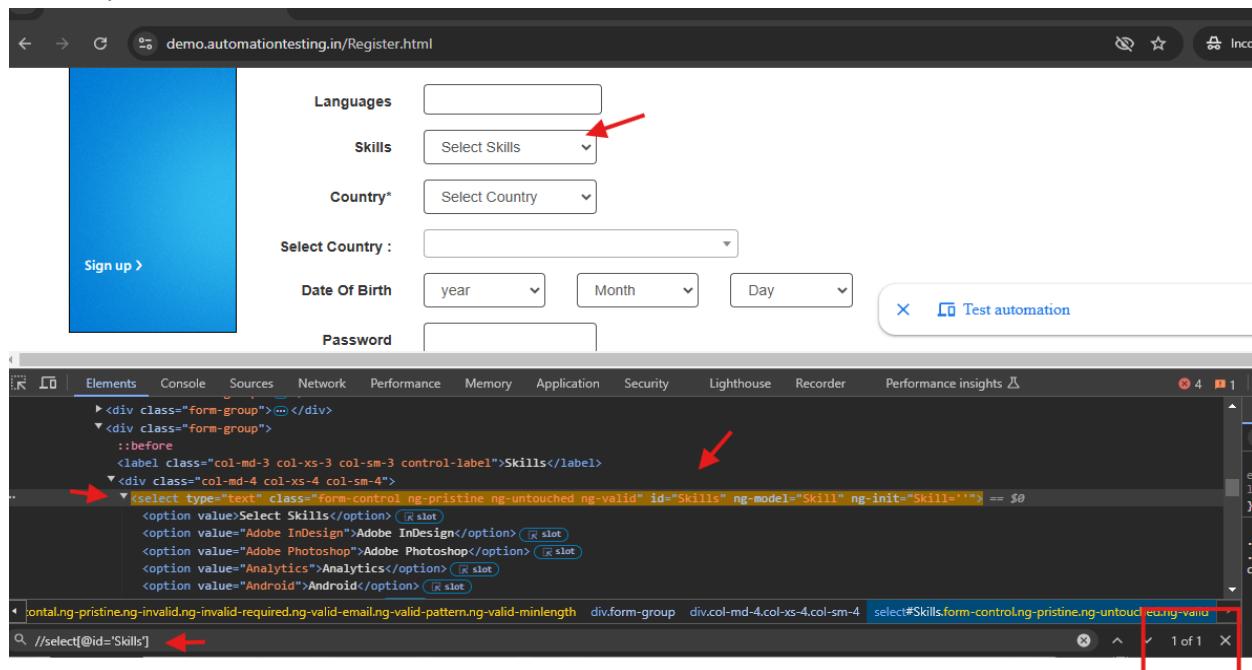
Locate (inspect) the element attributes for use as its locator:

Skills dropdown...



The screenshot shows a registration form with fields for Languages, Skills, Country, Select Country, Date Of Birth (with year, month, and day dropdowns), and Password. A 'Sign up >' button is on the left. The developer tools are open, showing the Elements tab with the DOM structure. A red arrow points to the 'Skills' dropdown. Another red arrow points to the 'Skills' element in the DOM tree. The CSS selector `select#Skills.form-control.ng-pristine.ng-untouched.ng-valid` is highlighted in yellow in the search bar.

DOB dropdown...



The screenshot shows the same registration form. A red arrow points to the 'Date Of Birth' dropdown. Another red arrow points to the 'Date Of Birth' element in the DOM tree. The CSS selector `//select[@id='yearbox']` is highlighted in yellow in the search bar.

Syntax:

- Test case 01 - use the attribute - '//*[@name="value"]'
- Test case 02 - use the attribute - '//*[@name='value'], @name='value')'

Using the XPath locator, create and run the code:

The screenshot shows the PyCharm IDE interface with the following details:

- Project:** playwrightprj1
- File:** dropdown_select.py
- Code Content:**

```
# == import and install files and libraries ==
from playwright.sync_api import sync_playwright

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False)
    page = browser.new_page()

    # >> Test case 01 - use the attribute - '//*[@name="value"]'
    print('Test case 01 START!')
    # open the browser page...
    page.goto('https://demo.automationtesting.in/Register.html')
    print('browser page title is ' + "'" + page.title() + "'")
    # using the attribute of the element under test...
    select_dropdown1 = page.query_selector("//select[@id='Skills']")
    # select the option
    select_dropdown1.select_option(label='Adobe Photoshop')
    print('Dropdown value selected...')
    # other actions...
    page.wait_for_timeout(10000)
    print('Test case 01 >> PASSED!')

    # >> Test case 02 - use the attribute - '//*[@name='value'], @name='value')'
    print('Test case 02 START!')
    # open the browser page...
    page.goto('https://demo.automationtesting.in/Register.html')
    print('browser page title is ' + "'" + page.title() + "'")
    # using the attribute of the element under test...
    select_dropdown2 = page.select_option(selector: "//select[@id='yearbox']", label='1918')
    print('Dropdown value selected...')
    # other actions...
    page.wait_for_timeout(10000)
    print('Test case 02 >> PASSED!')
    browser.close()
```
- Run Tab:** dropdown_select
- Output Log:**

```
C:\Users\njmlo\PycharmProjects\Playwright\playwrightprj1\.venv\Scripts\python.exe C:\Users\njmlo\PycharmProjects\Playwright\dropdown_select.py
Test case 01 START!
browser page title is "Register"...
Dropdown value selected...
Test case 01 >> PASSED!
Test case 02 START!
browser page title is "Register"...
Dropdown value selected...
Test case 02 >> PASSED!
```
- Status Bar:** Process finished with exit code 0

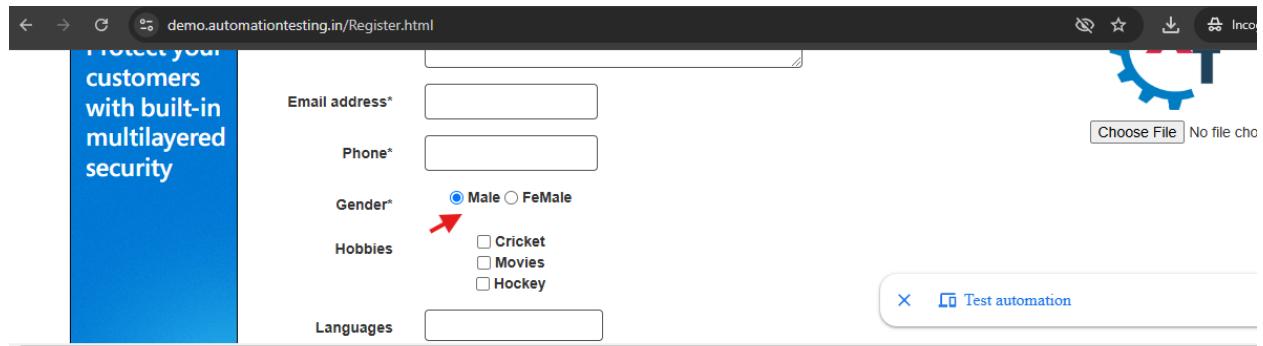
6. Working on a web page in selecting radio buttons and checkboxes

Test page url:

<https://demo.automationtesting.in/Register.html>

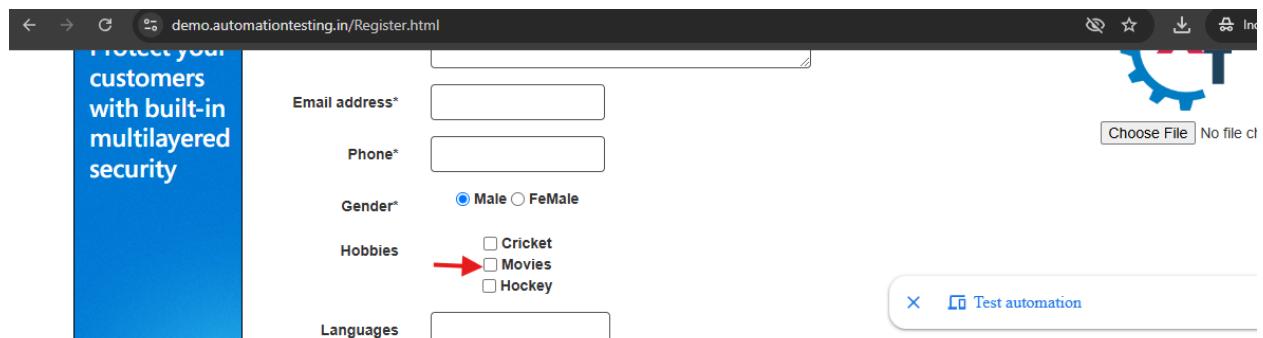
Locate (inspect) the element attributes for use as its locator:

Gender radio button...



The screenshot shows a web form with a sidebar on the left containing the text "Protect your customers with built-in multilayered security". The main form has fields for Email address*, Phone*, Gender*, Hobbies, and Languages. The Gender field contains radio buttons for Male and Female, with Male selected. Below the gender field is a list of hobbies: Cricket, Movies, and Hockey. The 'Movies' checkbox is highlighted with a red arrow. The browser's developer tools are open, showing the HTML structure. A red arrow points to the line of code for the 'Movies' checkbox in the Elements tab. The search bar at the bottom of the developer tools also shows the locator //input[@value="Movies"].

Hobbies checkbox...

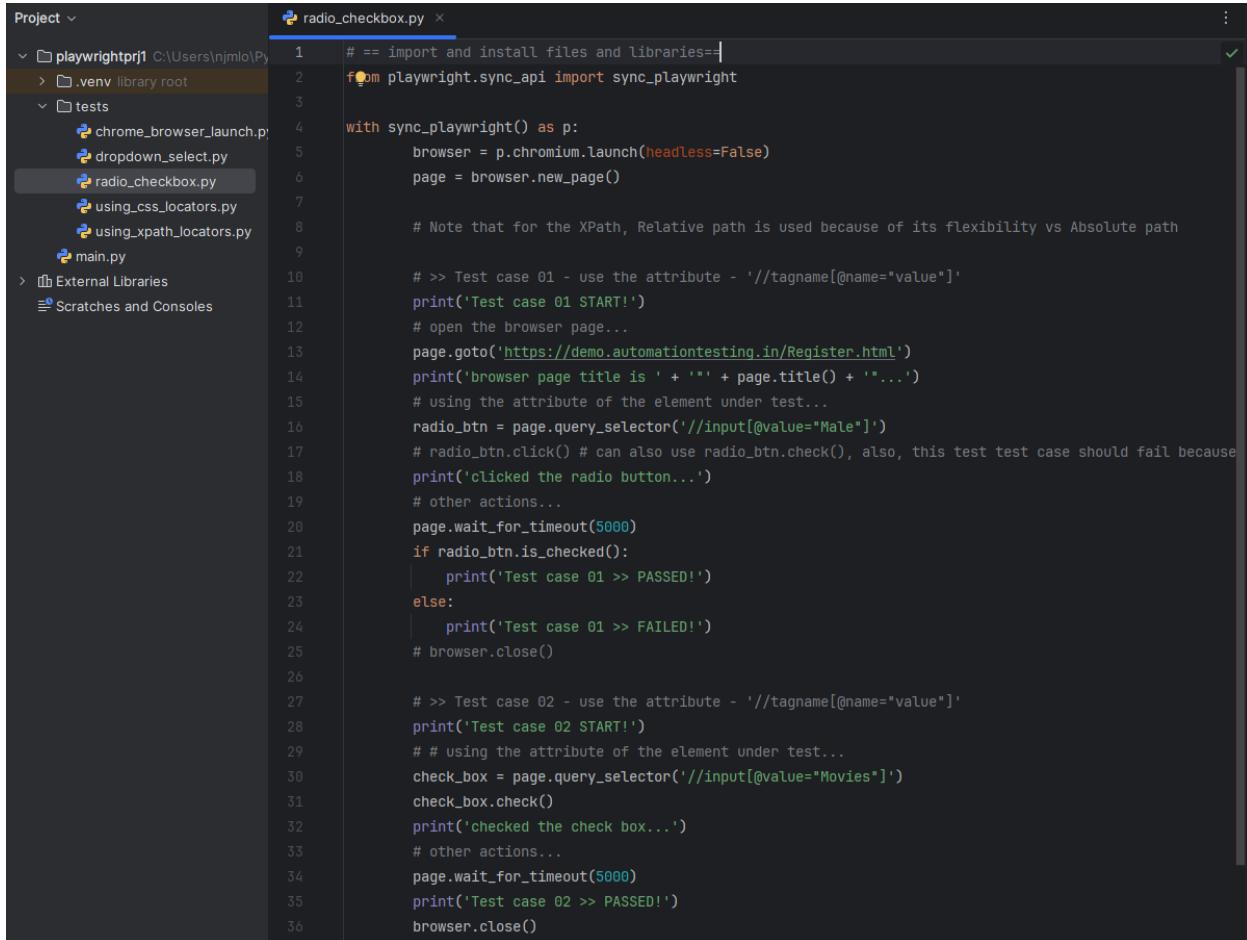


The screenshot shows the same web form as the previous one, but now the 'Movies' checkbox is unselected. The developer tools are open again, showing the HTML structure. A red arrow points to the line of code for the 'Movies' checkbox in the Elements tab. The search bar at the bottom of the developer tools also shows the locator //input[@value="Movies"].

Syntax:

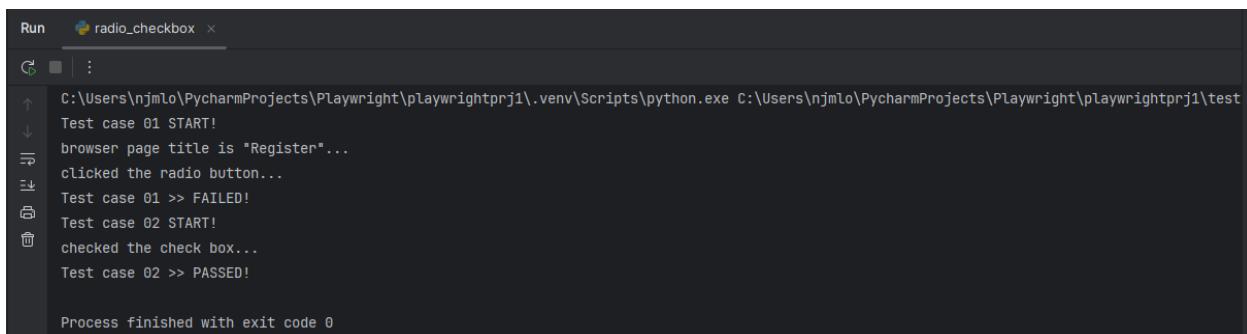
- Test case 01 - use the attribute - '//*[@name="value"]'
- Test case 02 - use the attribute - '//*[@name="value"]'

Using the XPath locator, create and run the code:



```
Project ▾
  playwrightprj1 C:\Users\njmlo\PycharmProjects\Playwright\playwrightprj1
    .venv library root
    tests
      chrome_browser_launch.py
      dropdown_select.py
      radio_checkbox.py
      using_css_locators.py
      using_xpath_locators.py
    main.py
  External Libraries
  Scratches and Consoles
```

```
radio_checkbox.py ×
1  # == import and install files and libraries=|
2  from playwright.sync_api import sync_playwright
3
4  with sync_playwright() as p:
5      browser = p.chromium.launch(headless=False)
6      page = browser.new_page()
7
8      # Note that for the XPath, Relative path is used because of its flexibility vs Absolute path
9
10     # >> Test case 01 - use the attribute - '//*[@name="value"]'
11     print('Test case 01 START!')
12     # open the browser page...
13     page.goto('https://demo.automationtesting.in/Register.html')
14     print('browser page title is ' + ' ' + page.title() + '...')
15     # using the attribute of the element under test...
16     radio_btn = page.query_selector('//*[@value="Male"]')
17     # radio_btn.click() # can also use radio_btn.check(), also, this test test case should fail because
18     print('clicked the radio button...')
19     # other actions...
20     page.wait_for_timeout(5000)
21     if radio_btn.is_checked():
22         print('Test case 01 >> PASSED!')
23     else:
24         print('Test case 01 >> FAILED!')
25     # browser.close()
26
27     # >> Test case 02 - use the attribute - '//*[@name="value"]'
28     print('Test case 02 START!')
29     # # using the attribute of the element under test...
30     check_box = page.query_selector('//*[@value="Movies"]')
31     check_box.check()
32     print('checked the check box...')
33     # other actions...
34     page.wait_for_timeout(5000)
35     print('Test case 02 >> PASSED!')
36     browser.close()
```



```
Run radio_checkbox ×
C:\Users\njmlo\PycharmProjects\Playwright\playwrightprj1\.venv\Scripts\python.exe C:\Users\njmlo\PycharmProjects\Playwright\playwrightprj1\test
↑
↓
Test case 01 START!
browser page title is "Register"...
clicked the radio button...
Test case 01 >> FAILED!
Test case 02 START!
checked the check box...
Test case 02 >> PASSED!

Process finished with exit code 0
```

7. Working on a web page on alerts and dialog box

Test page url:

<https://demo.automationtesting.in/Alerts.html>

Locate (inspect) the element attributes for use as its locator:

Alert screen options for the Alerts with OK & Cancel dialog popup window...

The screenshot shows a browser window with the URL <https://demo.automationtesting.in/Alerts.html>. The page displays two tabs: 'Alert with OK' and 'Alert with OK & Cancel'. The 'Alert with OK & Cancel' tab is active. A red arrow points to this tab. In the top right corner of the browser, there is a message: 'click the button to display a confirm box' and 'You Pressed Cancel'. Below the browser, the developer tools are open, specifically the Elements tab. A red arrow points to the 'CancelTab' entry in the list of tabs. The search bar at the bottom of the developer tools has a red arrow pointing to it with the query 'a[href="#CancelTab"]'. The status bar at the bottom right shows '1 of 1'.

The screenshot shows a browser window with the URL <https://demo.automationtesting.in/Alerts.html>. The page displays three tabs: 'Alert with OK', 'Alert with OK & Cancel', and 'Alert with Textbox'. The 'Alert with OK & Cancel' tab is active. A red arrow points to this tab. In the top right corner of the browser, there is a message: 'click the button to display a confirm box' and 'You Pressed Cancel'. Below the browser, the developer tools are open, specifically the Elements tab. A red arrow points to the 'CancelTab' entry in the list of tabs. The search bar at the bottom has a red arrow pointing to it with the query 'a[href="#CancelTab"]'.

The screenshot shows a browser window with the URL <https://demo.automationtesting.in/Alerts.html>. The page displays three tabs: 'OKTab', 'CancelTab', and 'Textbox'. The 'CancelTab' tab is active. A red arrow points to this tab. In the top right corner of the browser, there is a message: 'click the button to display an alert box:' followed by a button and 'click the button to display a confirm box' followed by another button. Below the browser, the developer tools are open, specifically the Elements tab. A red arrow points to the 'CancelTab' entry in the list of tabs. The search bar at the bottom has a red arrow pointing to it with the query '//div[@id="CancelTab"]/button'.

Alert screen options for the Alerts with OK & Cancel dialog popup window...

Alert with OK

Alert with OK & Cancel

Alert with Textbox

click the button to demonstrate the prompt box

Elements Console Sources Network Performance Memory Application Security Lighthouse Recorder Performance insights

//a[@href="#Textbox"]

1 match

Alert with OK

Alert with OK & Cancel

Alert with Textbox

click the button to demonstrate the prompt box

Elements Console Sources Network Performance Memory Application Security Lighthouse Recorder Performance insights

//div[@id="Textbox"]

1 match

Alert with OK

Alert with OK & Cancel

Alert with Textbox

click the button to demonstrate the prompt box

Elements Console Sources Network Performance Memory Application Security Lighthouse Recorder Performance insights

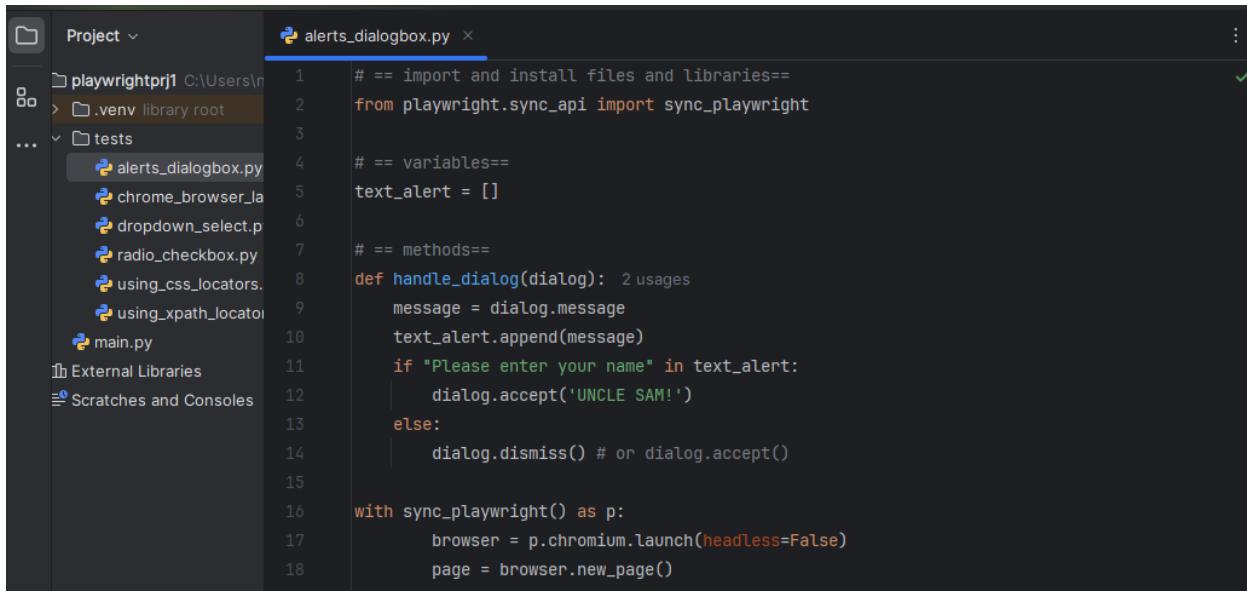
//div[@id="Textbox"]/button

1 of 1

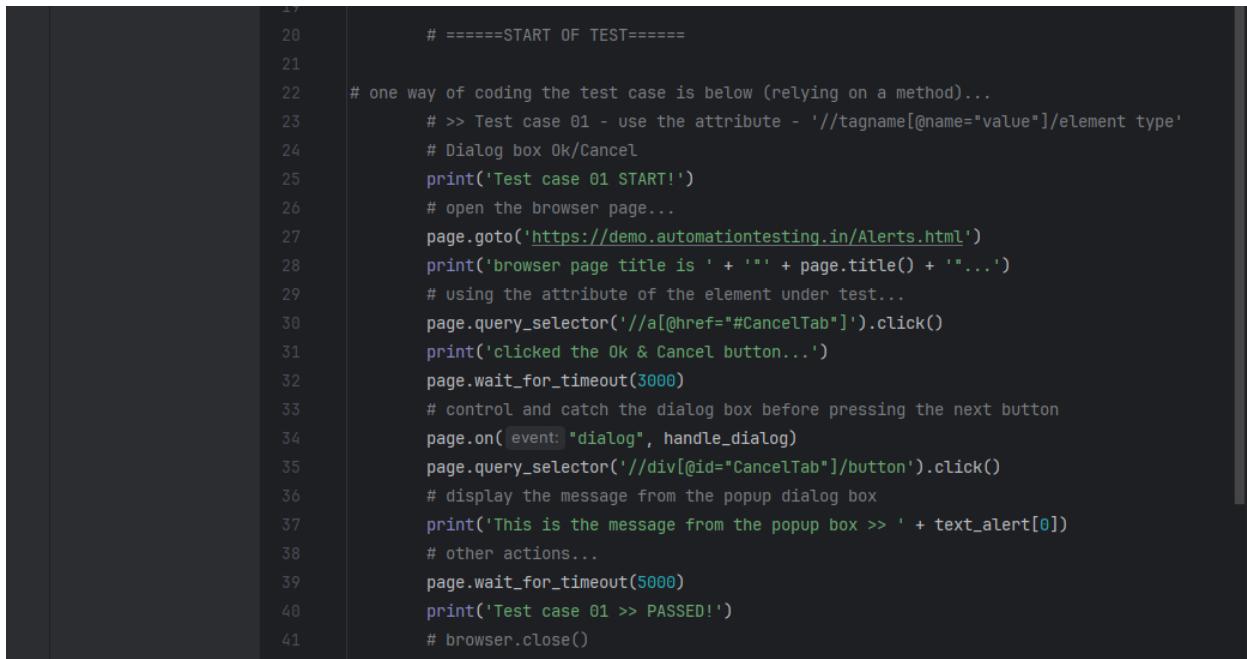
Syntax:

- Test case 01 - use the attribute - '//tagname[@name="value"]/child tagname', dialog box Ok/Cancel
- Test case 02 - use the attribute - '//tagname[@name="value"]/child tagname', dialog box with Textbox

Using the XPath locator, create and run the code:



```
1 # == import and install files and libraries==
2 from playwright.sync_api import sync_playwright
3
4 # == variables ==
5 text_alert = []
6
7 # == methods ==
8 def handle_dialog(dialog): 2 usages
9     message = dialog.message
10    text_alert.append(message)
11    if "Please enter your name" in text_alert:
12        dialog.accept('UNCLE SAM!')
13    else:
14        dialog.dismiss() # or dialog.accept()
15
16 with sync_playwright() as p:
17     browser = p.chromium.launch(headless=False)
18     page = browser.new_page()
```



```
17
18
19
20     # =====START OF TEST=====
21
22 # one way of coding the test case is below (relying on a method)...
23 # >> Test case 01 - use the attribute - '//tagname[@name="value"]/element type'
24 # Dialog box Ok/Cancel
25 print('Test case 01 START!')
26 # open the browser page...
27 page.goto('https://demo.automationtesting.in/Alerts.html')
28 print('browser page title is ' + "" + page.title() + "...")
29 # using the attribute of the element under test...
30 page.query_selector('//a[@href="#CancelTab"]').click()
31 print('clicked the OK & Cancel button...')
32 page.wait_for_timeout(3000)
33 # control and catch the dialog box before pressing the next button
34 page.on( event: "dialog", handle_dialog)
35 page.query_selector('//div[@id="CancelTab"]/button').click()
36 # display the message from the popup dialog box
37 print('This is the message from the popup box >> ' + text_alert[0])
38 # other actions...
39 page.wait_for_timeout(5000)
40 print('Test case 01 >> PASSED!')
41 # browser.close()
```

```
Run alerts_dialogbox x

C:\Users\njml0\PycharmProjects\Playwright\playwrightprj1\.venv\Scripts\python.exe C:\Users\njml0\PycharmProjects\Playwright\playwrightprj1\tests\alerts_dialogbox.py

Test case 01 START!
browser page title is "Alerts"...
clicked the Ok & Cancel button...
This is the message from the popup box >> Press a Button !
Test case 01 >> PASSED!

Test case 02 START!
browser page title is "Alerts"...
clicked the Alerts with Textbox button...
This is the message from the popup box >> Please enter your name
Test case 02 >> PASSED!

Process finished with exit code 0
```

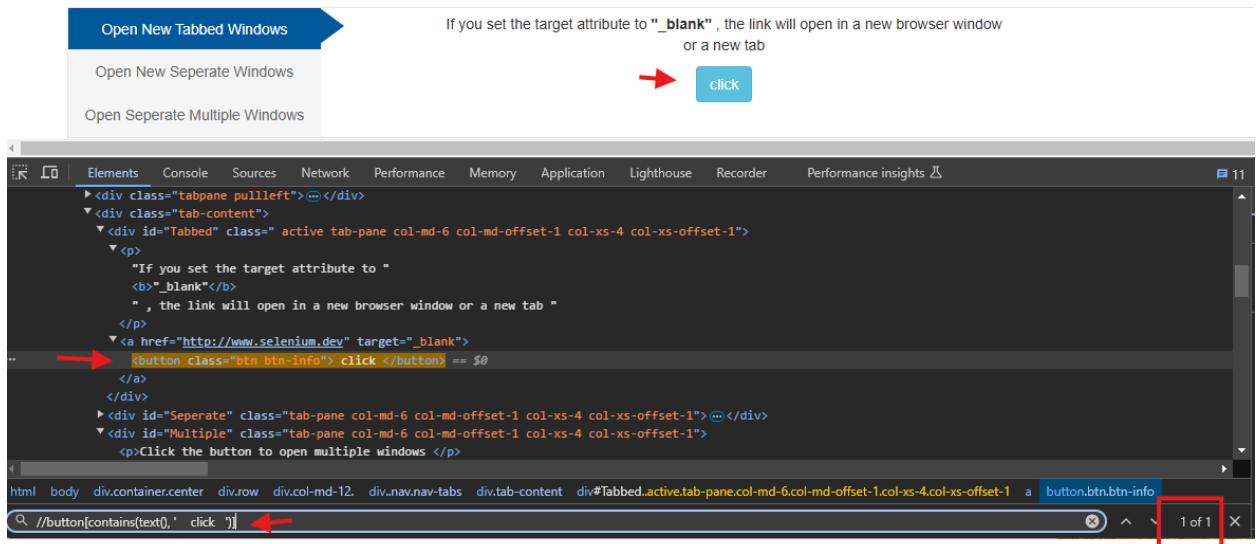
8. Working on a web page windows/page handling

Test page url:

<https://demo.automationtesting.in/Windows.html>

Locate (inspect) the element attributes for use as its locator:

Open new tabbed windows click button...



Syntax:

a. Test case 01 - use the attribute - "//tagname[contains=(text(), 'value')]"

Using the XPath locator, create and run the code:

The screenshot shows the PyCharm IDE interface. The top bar displays the project name "playwrightprj1" and the file "page_switch.py". The left sidebar shows the project structure with files like alerts_dialogbox.py, chrome_browser_.py, dropdown_select_.py, main.py, and others. The main editor window contains the Python code for switching between browser pages using the XPath locator.

```
# == import and install files and libraries==
from playwright.sync_api import sync_playwright

# == variables==

# == methods==

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False)
    context = browser.new_context() # context handles multiple page, new page handles only
    page = context.new_page()

    # =====START OF TEST=====

    # >> Test case 01 - use the attribute - "//tagname[contains(text(), 'value')]"
    # Dialog box Ok/Cancel
    print('Test case 01 START!')
    # open the browser page...
    page.goto('https://demo.automationtesting.in/Windows.html')
    # using the attribute of the element under test...
    page.wait_for_selector("//button[contains(text(), ' click ')]").click()
    page.wait_for_timeout(3000)
    # this is to find/collect (and a list is returned) of all the pages under test
    total_pages = context.pages
    print('Total opened browser page (see below urls) >> ' + str(len(total_pages)))
    for i in total_pages:
        print(i)
    print('parent page title is ' + ' ' + page.title() + '...')
    # this is how to store new pages in a variable
    new_page = total_pages[1]
    # this brings the child page in focus or active mode
    new_page.bring_to_front()
    print('child page title is ' + ' ' + new_page.title() + '...')
    # other actions...
    new_page.wait_for_timeout(3000)
    # switch back to child page, this closes only the child page
    new_page.close()
    print('All opened child pages are now closed.')
    # this brings the main page back in focus or active mode
    page.bring_to_front()
    page.wait_for_timeout(3000)
    # close all other opened pages
    browser.close()
    print('All opened browser pages are now closed.')
    print('Test case 01 >> PASSED!')
```

The screenshot shows the PyCharm Run tab with the output of the "page_switch" run configuration. It displays the console logs from the script execution, showing the test case starting, listing the two opened browser pages, switching between them, and finally closing them all, resulting in a successful test case.

```
C:\Users\njmlo\PycharmProjects\Playwright\playwrightprj1\.venv\Scripts\python.exe C:\Users\njmlo\PycharmProjects\Playwright\page_switch.py
Test case 01 START!
Total opened browser page (see below urls) >> 2
<Page url='https://demo.automationtesting.in/Windows.html'>
<Page url='https://www.selenium.dev/'>
parent page title is "Frames & windows"...
child page title is "Selenium"...
All opened child pages are now closed.
All opened browser pages are now closed.
Test case 01 >> PASSED!

Process finished with exit code 0
```

9. Working on a web page cookies and taking page screenshots

Test page url:

<https://demo.automationtesting.in/Register.html>

The screenshot shows the PyCharm IDE interface. The left sidebar displays the project structure for 'playwrightprj1' with files like 'alerts_dialogbox.py', 'chrome_browser_laun...', 'cookies_attributes.py', 'dropdown_select.py', 'page_switch.py', 'radio_checkbox.py', 'using_css_locators.py', 'using_xpath_locators.py', and 'main.py'. The right pane shows the code for 'cookies_attributes.py':

```
# == import and install files and libraries==
from playwright.sync_api import sync_playwright

# == variables==

# == methods==

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False)
    context = browser.new_context() # context handles multiple page, new page handles only
    page = context.new_page()

    # =====START OF TEST=====

    # >> Test case 01 - use the attribute - "//tagname[contains=(text(), 'value')]"
    # Dialog box OK/Cancel
    print('Test case 01 START!')
    page.goto('https://demo.automationtesting.in/Register.html')
    # get and display all the cookies
    my_cookies = page.context.cookies()
    print(my_cookies)
    # clear all the cookies
    page.context.clear_cookies()
    print('All cookies cleared.')

    # syntax in insert cookies
    # new_cookies = {
    #     'name': 'test_sample',
    #     'value': 'asdflkasdhiflsoadfjsdlfjsdklfdsd'
    # }
    # pass the new cookies into the target page
    # page.context.add_cookies([new_cookies])

    #taking page screenshot
    page.screenshot(path='test.png', full_page=False)
    print('Screenshots printed.')
    # close all other opened pages
    browser.close()
    print('All opened browser pages are now closed.')
    print('Test case 01 >> PASSED!')
```

The screenshot shows the PyCharm 'Run' tab with the output of the executed script. The terminal window displays the following log:

```
C:\Users\njmlo\PycharmProjects\Playwright\playwrightprj1\.venv\Scripts\python.exe C:\Users\njmlo\PycharmProjects\Playwright\playwrightprj1\tests\cookies_attributes.py
Test case 01 START!
[{'name': 'IDE', 'value': 'AHWqTULWdbJJ7gefAbQvyPcR5f85hSNxxoyXzLpiF6HyJF8cjBM3Z7aRX-SEjhD', 'domain': '.doubleclick.net'}
All cookies cleared.
Screenshots printed.
All opened browser pages are now closed.
Test case 01 >> PASSED!

Process finished with exit code 0
```

10. Working on web page interactions

Test page url:

<https://demo.automationtesting.in>Selectable.html>

Locate (inspect) the element attributes for use as its locator:

Switch To list men ...

The screenshot shows a browser window for the "Automation Demo Site". The navigation bar includes links for Home, Register, WebTable, SwitchTo, Widgets, Interactions, Video, WYSIWYG, More, and Practice. A red arrow points to the "SwitchTo" link. Below the navigation bar, a dropdown menu is open under the "SwitchTo" link, showing options like "Default Functionality" and "Serialize". The main content area displays a message: "You've selected: Readability , Method Chaining , Extent Reports" and a section titled "Sakinalium - Readability". The developer tools (F12) are open, showing the Elements tab with the DOM structure. A red box highlights the search bar at the bottom, which contains the XPath expression "//a[contains(text(), 'SwitchTo')]" and has a red arrow pointing to it. The status bar at the bottom right shows "1 of 1".

Syntax:

a. Test case 01 - use the attribute - "//tagname[contains=(text(), 'value')]"

Using the XPath locator, create and run the code:

The screenshot shows the PyCharm IDE interface. The left sidebar displays the project structure for 'playwrightprj1' with files like alerts_dialogbox.py, chrome_browser_laun..., cookies_values.py, dropdown_select.py, main.py, and several test files under 'tests'. The right pane shows the code editor for 'page_actions.py'. The code imports playwright.sync_api and sync_playwright, then uses sync_playwright() to launch a browser and navigate to a demo automation testing URL. It performs various mouse and keyboard actions on the page, such as hovering over links, clicking, and pressing keys. Finally, it closes the browser and prints a success message.

```
# == import and install files and libraries==
from playwright.sync_api import sync_playwright

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False)
    context = browser.new_context() # context handles multiple page, new page handles only
    page = context.new_page()

# =====START OF TEST=====

# >> Test case 01 - use the attribute - "//tagname[contains(text(), 'value')]"
print('Test case 01 START!')
# go to browser under test
page.goto('https://demo.automationtesting.in>Selectable.html')
# ----mouse action below...
# find the page element and hover the mouse
page.wait_for_selector('//a[contains(text(), "SwitchTo")]').hover()
page.wait_for_timeout(3000)
# # single click on a selected element value
page.wait_for_selector('//a[contains(text(), "Frames")]').click()
page.wait_for_timeout(3000)
# # double-click on a selected element value
# page.wait_for_selector('//a[contains(text(), "SwitchTo")]').dblclick()
# # right-click on a selected element value
# page.wait_for_selector('//a[contains(text(), "SwitchTo")]').click(button="right")
# # right-click on a selected element value
# page.wait_for_selector('//a[contains(text(), "SwitchTo")]').click(modifiers=["Shift"])
# ----keyboard action below...
# ie., A-Z, 0-9, F1-F12, special characters, ArrowRight, PageUp, Enter, Control, and so
# page.wait_for_selector('//input[@type="text"]').press("A")
# # close all other opened pages
browser.close()
print('All opened browser pages are now closed.')
print('Test case 01 >> PASSED!')
```

The screenshot shows the 'Run' tab in PyCharm. It displays the command used to run the script ('C:\Users\njmlo\PycharmProjects\Playwright\playwrightprj1\.venv\Scripts\python.exe C:\Users\njmlo\PycharmProjects\Playw...'), followed by the output of the script's print statements: 'Test case 01 START!', 'All opened browser pages are now closed.', and 'Test case 01 >> PASSED!'. The bottom of the tab shows the status 'Process finished with exit code 0'.

```
C:\Users\njmlo\PycharmProjects\Playwright\playwrightprj1\.venv\Scripts\python.exe C:\Users\njmlo\PycharmProjects\Playw...
Test case 01 START!
All opened browser pages are now closed.
Test case 01 >> PASSED!
Process finished with exit code 0
```

11. Working on web page exception handling and storing elements in lists

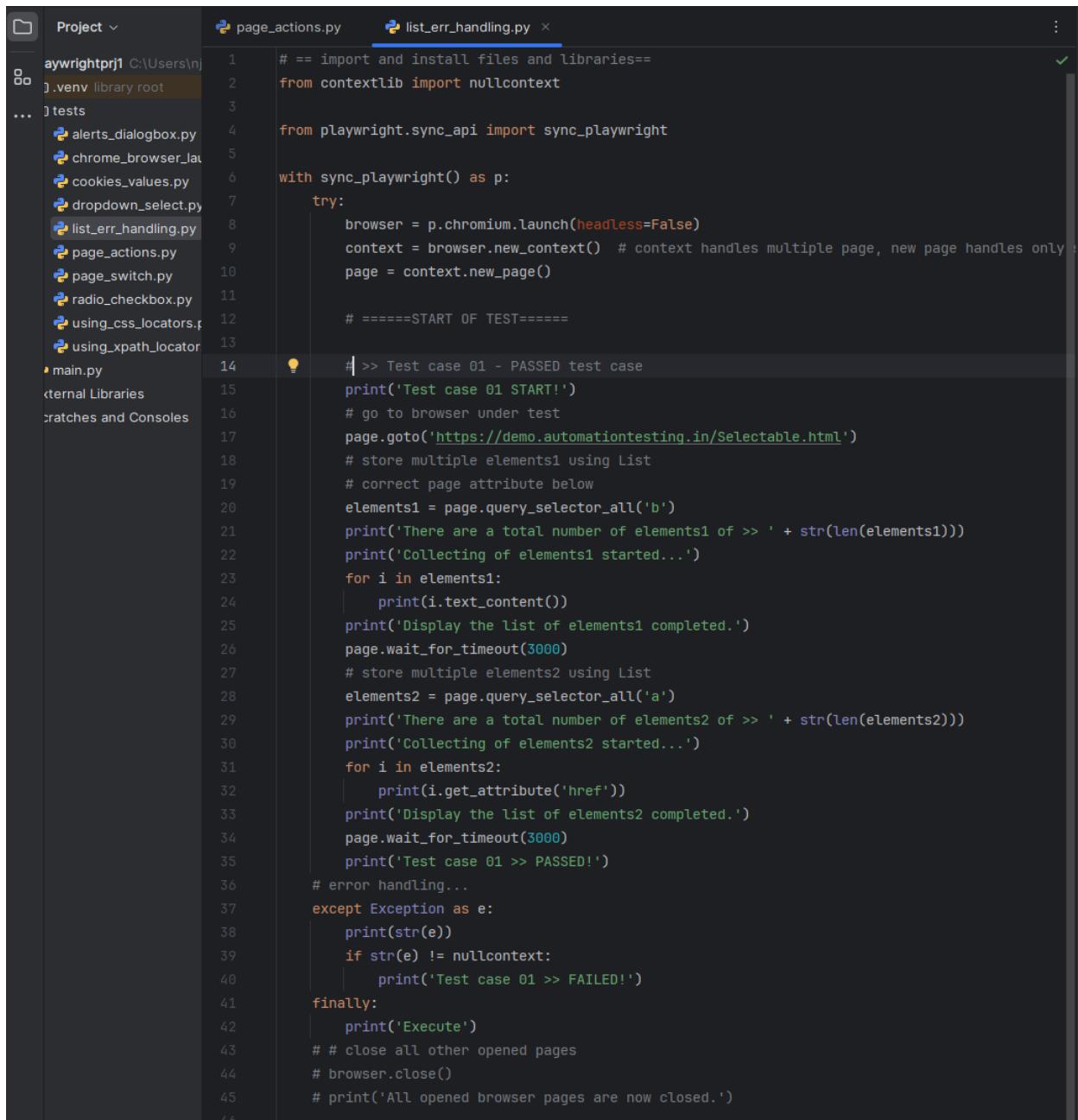
Test page url:

<https://demo.automationtesting.in>Selectable.html>

Syntax:

- a. Test case 01 – PASSED test case
- b. Test case 02 – FAILED test case (to test error handling syntax)

Using the locator, create and run the code:



The screenshot shows a code editor with a dark theme. On the left is a file tree for a project named 'aywrightprj1' located at 'C:\Users\ni'. The tree includes a '.venv' folder, a 'tests' folder containing several files like 'alerts_dialogbox.py', 'chrome_browser_lat.py', 'cookies_values.py', 'dropdown_select.py', 'list_err_handling.py', 'page_actions.py', 'page_switch.py', 'radio_checkbox.py', 'using_css_locators.py', and 'using_xpath_locator.py', and a 'main.py' file. The 'list_err_handling.py' file is currently open in the main editor area. The code is as follows:

```
# == import and install files and libraries==
from contextlib import nullcontext
from playwright.sync_api import sync_playwright

with sync_playwright() as p:
    try:
        browser = p.chromium.launch(headless=False)
        context = browser.new_context() # context handles multiple page, new page handles only
        page = context.new_page()

        # =====START OF TEST=====

        # >> Test case 01 - PASSED test case
        print('Test case 01 START!')
        # go to browser under test
        page.goto('https://demo.automationtesting.in>Selectable.html')
        # store multiple elements1 using List
        # correct page attribute below
        elements1 = page.query_selector_all('b')
        print('There are a total number of elements1 of >> ' + str(len(elements1)))
        print('Collecting of elements1 started...')
        for i in elements1:
            print(i.text_content())
        print('Display the list of elements1 completed.')
        page.wait_for_timeout(3000)
        # store multiple elements2 using List
        elements2 = page.query_selector_all('a')
        print('There are a total number of elements2 of >> ' + str(len(elements2)))
        print('Collecting of elements2 started...')
        for i in elements2:
            print(i.get_attribute('href'))
        print('Display the list of elements2 completed.')
        page.wait_for_timeout(3000)
        print('Test case 01 >> PASSED!')

        # error handling...
        except Exception as e:
            print(str(e))
            if str(e) != nullcontext:
                print('Test case 01 >> FAILED!')
        finally:
            print('Execute')
            # # close all other opened pages
            # browser.close()
            # print('All opened browser pages are now closed.')
    
```

```
Project page_actions.py list_err_handling.py
Run list_err_handling ×

C:\Users\njmlo\PycharmProjects\Playwright\playwrightprj1\.venv\Scripts\python.exe C:\Users\njmlo\PycharmProjects\Playwright\playwrightprj1\list_err_handling.py
↑ Test case 01 START!
↓ There are a total number of elements1 of >> 14
Collecting of elements1 started...
Sakinalium - Readability
Sakinalium - Single Line Coding
Sakinalium - Method Chaining
Sakinalium - Cross Browser Testing
Sakinalium - Extent Reports
Sakinalium - Data Driven Testing
Sakinalium - Functional Testing
Sakinalium - Readability
Sakinalium - Single Line Coding
Sakinalium - Method Chaining
Sakinalium - Cross Browser Testing
Sakinalium - Extent Reports
Sakinalium - Data Driven Testing
Sakinalium - Functional Testing
Display the list of elements1 completed.
There are a total number of elements2 of >> 45
Collecting of elements2 started...
http://www.automationtesting.in
Index.html
Register.html
WebTable.html
SwitchTo.html
Alerts.html
Windows.html
Frames.html
Widgets.html
Accordion.html
AutoComplete.html
Datepicker.html
Slider.html
Interactions.html
None
Static.html
Dynamic.html
Selectable.html
Resizable.html
SwitchTo.html
Youtube.html
Vimeo.html
WYSIWYG.html
TinyMCE.html
CKEditor.html
SummerNote.html
CodeMirror.html
#
Charts.html
DynamicData.html
FileDownload.html
FileUpload.html
JqueryProgressBar.html
Loader.html
Modals.html
ProgressBar.html
http://practice.automationtesting.in/
#Default
#Serialize
#
https://www.facebook.com/automationtesting2016/
https://twitter.com/krishnasakinala
https://www.linkedin.com/nhome/?trk=hb\_signin
https://plus.google.com/105206300926085335367
https://www.youtube.com/channel/UCmQRa3pWM9zsB474URz8ESg
Display the list of elements2 completed.
Test case 01 >> PASSED!
Execute
```

```
47     try:
48         # >> Test case 02 - FAILED test case
49         print('Test case 02 START!')
50         # go to browser under test
51         page.goto('https://demo.automationtesting.in>Selectable.html')
52         # store multiple elements1 using List
53         # incorrect page attribute below (to test the error handling syntax)
54         page.query_selector('d//sf="wrong"]').click()
55         # correct page attribute below
56         elements1 = page.query_selector_all('b')
57         print('There are a total number of elements1 of >> ' + str(len(elements1)))
58         print('Collecting of elements1 started...')
59         for i in elements1:
60             print(i.text_content())
61         print('Display the list of elements1 completed.')
62         page.wait_for_timeout(3000)
63         # store multiple elements2 using List
64         elements2 = page.query_selector_all('a')
65         print('There are a total number of elements2 of >> ' + str(len(elements2)))
66         print('Collecting of elements2 started...')
67         for i in elements2:
68             print(i.get_attribute('href'))
69         print('Display the list of elements2 completed.')
70         page.wait_for_timeout(3000)
71         print('Test case 02 >> PASSED!')
72         # error handling...
73     except Exception as e:
74         print(str(e))
75         if str(e) != nullcontext:
76             print('Test case 02 >> FAILED!')
77     finally:
78         print('Execute')
79         # # close all other opened pages
80         browser.close()
81         print('All opened browser pages are now closed.')
82
```

```
Test case 02 START!
Page.query_selector: Unexpected token "/" while parsing selector "d//sf="wrong"]"
Test case 02 >> FAILED!
Execute
All opened browser pages are now closed.
```

```
Process finished with exit code 0
```

12. Working on web page Ajax handling

Test page url:

https://www.plus2net.com/php_tutorial/ajax_drop_down_list-demo.php

Locate (inspect) the element attributes for use as its locator:

Main dropdown...

Sub dropdown...

plus2net

HOME PHP Ajax Search Submit

We use cookies to improve your browsing experience. Learn more

Login

plus2net

Demo of linked drop down list by using Ajax & PHP

form 935 x 184.78

Name: [input type="text"]

Select one

Category : Colors ↗

SubCategory : Green ↗

submit

You HAVE ONLY IQ: 155

TRY

YOU HAVE ONLY IQ: 155

Submit this form and see how the form components data are available in action page of the form.

Elements Console Sources Network Performance Memory Application Lighthouse Recorder Performance insights

<form name="testform" method="POST" action="mainck.php">

"Name:"

<input type="text" name="fname">

"Select one "

" Category : "

<select name="cat" onchange="ajaxFunction();" id="s1"> == \$0 ↗

<option value>Select One</option>

<option value="1">Fruits</option>

<option value="2">Colors</option>

<option value="3">Games</option>

<option value="4">Vehicles</option>

</select>

" SubCategory : "

<select name="subcat" id="s2">@</select>

html body div.row div.col-md-7.col-md-offset-1 div#body_content table tbody tr td span plus2net_index form select#s1

/select[@id="s1"] ↗

Headers Payload Preview Response Initiator Timing Cookies

General

Request URL: https://www.plus2net.com/php_tutorial/dd-ajax.php?cat_id=1&sid=0.8215641309476789

Request Method: GET

Status Code: 200 OK

Remote Address: 68.178.227.154:443

Referrer Policy: strict-origin-when-cross-origin

Response Headers

Content-Encoding: br

Content-Length: 84

```

Name
dd-ajax.php?cat_id=1&sid=0.8215641309476789
dd-ajax.php?cat_id=2&sid=0.17490690679270138
interaction/?ai=CpsFijpNZ8isH4Oq2OMP-J6DwA...
interaction/?ai=CtNzWijpNZ5eYlsCZ20MPiLlimA...
interaction/?ai=CtQoJipNZ-vFl6m52OMP55SrsAT...
interaction/?ai=C7ChTijpNZ7jaJP3220MPpqXtSO...
interaction/?ai=Cd05tlinNZ-6w4f6nvn88PxQ2-eOr...

X Headers Payload Preview Response Initiator Timing Cookies
▼ {data: [{"subcategory": "Red", "subcat_id": "5"}, {"subcategory": "Blue", "subcat_id": "6"}, ...]
  ▼ data: [{"subcategory": "Red", "subcat_id": "5"}, {"subcategory": "Blue", "subcat_id": "6"}, ...]
    ▷ 0: {"subcategory": "Red", "subcat_id": "5"}
    ▷ 1: {"subcategory": "Blue", "subcat_id": "6"}
    ▷ 2: {"subcategory": "Green", "subcat_id": "7"}
    ▷ 3: {"subcategory": "Yellow", "subcat_id": "8"}}

```

Syntax:

a. Test case 01 - AJAX handling

Using the XPath locator, create and run the code:

```

Project ajax_call.py
playwrightprj1 C:\Users\njml0\PycharmProjects\Playwright\playwrightprj1\.venv\lib\site-packages\playwright\sync\_api\__init__.py
tests
  ajax_call.py
  alerts_dialog.py
  chrome_brow.py
  cookies_valu.py
  dropdown_se.py
  list_err_hand.py
  page_actions.py
  page_switch.py
  radio_checkb.py
  using_css_lo.py
  using_xpath_.py
  main.py
External Libraries
Scratches and Con
# == import and install files and libraries ==
from playwright.sync_api import sync_playwright

# == methods ==
def handle_ajax(response): 1 usage
    if 'https://www.plus2net.com/php_tutorial/dd-ajax.php?' in response.url:
        status = response.status
        data = response.text()
        print(f'status:{status}, data:{data}')

    with sync_playwright() as p:
        browser = p.chromium.launch(headless=False)
        context = browser.new_context()
        page = context.new_page()

# =====START OF TEST=====
>>> Test case 01 - AJAX handling
print('Test case 01 START!')
page.goto('https://www.plus2net.com/php_tutorial/ajax_drop_down_list-demo.php')
print('browser page title is ' + page.title() +'...')
select = page.wait_for_selector('//select[@id="s1"]')
# setup a "listener" to display the list values of selected category
page.on(event: 'response', lambda response : handle_ajax(response))
select.select_option('3')
print('Dropdown value selected...')
# other actions...
print('Test case 01 >> PASSED!')
page.wait_for_timeout(5000)
browser.close()

```

```

Run ajax_call
C:\Users\njml0\PycharmProjects\Playwright\playwrightprj1\.venv\Scripts\python.exe C:\Users\njml0\PycharmProjects\Playwright\playwrightprj1\tests\ajax_call.py
Test case 01 START!
browser page title is "Demo of two Dependent drop down list using Ajax and PHP"...
Dropdown value selected...
Test case 01 >> PASSED!
status:200, data:{"data":[{"subcategory":"Cricket","subcat_id":"9"}, {"subcategory":"Football","subcat_id":"10"}, {"subcategory":"Basketball","subcat_id":"11"}, {"subcategory":"Hockey","subcat_id":"12"}]}
Process finished with exit code 0

```

13. Working on web page Web Tables

Test page url:

<https://cosmocode.io/automation-practice-webtable/>

Locate (inspect) the element attributes for use as its locator:

Demo table...

The screenshot shows a browser window with the title "Automation Practice | WebTable". The address bar contains the URL "cosmocode.io/automation-practice-webtable/". The main content area displays a table with the following data:

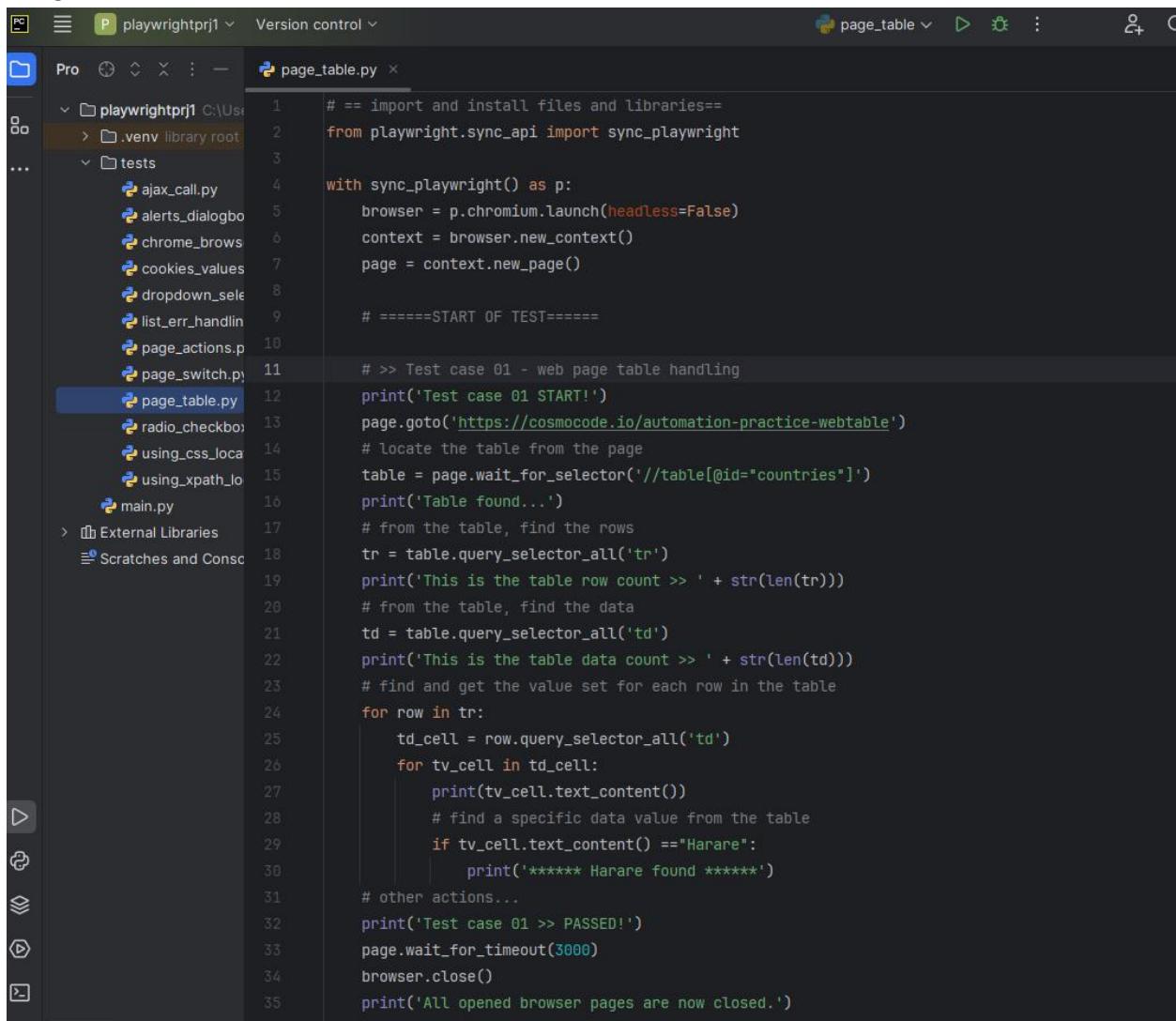
Visited	Country	Capital(s)	Currency	Primary Language(s)
<input type="checkbox"/>	Afghanistan	Kabul	Afghani	Dari Persian; Pashto
<input type="checkbox"/>	Albania	Tirane	Lek	Albanian
<input type="checkbox"/>	Algeria	Algiers	Algerian Dinar	Arabic; Tamazight; French
<input type="checkbox"/>	Andorra	Andorra la Vella	Euro	Catalan
<input type="checkbox"/>	Angola	Luanda	Kwanza	Portuguese

Below the table, the developer tools' Elements tab is open, showing the DOM structure. A red arrow points to the table element in the tree view. Another red arrow points to the search bar at the bottom left, which contains the XPath expression "//table[@id='countries']". A third red box highlights the status bar at the bottom right which says "1 of 1".

Syntax:

a. Test case 01 - web page table handling

Using the XPath locator, create and run the code:



```
# == import and install files and libraries ==
from playwright.sync_api import sync_playwright

# =====START OF TEST=====
with sync_playwright() as p:
    browser = p.chromium.launch(headless=False)
    context = browser.new_context()
    page = context.new_page()

    # >> Test case 01 - web page table handling
    print('Test case 01 START!')
    page.goto('https://cosmocode.io/automation-practice-webtable')
    # locate the table from the page
    table = page.wait_for_selector('//table[@id="countries"]')
    print('Table found...')

    # from the table, find the rows
    tr = table.query_selector_all('tr')
    print('This is the table row count >> ' + str(len(tr)))
    # from the table, find the data
    td = table.query_selector_all('td')
    print('This is the table data count >> ' + str(len(td)))
    # find and get the value set for each row in the table
    for row in tr:
        td_cell = row.query_selector_all('td')
        for tv_cell in td_cell:
            print(tv_cell.text_content())
            # find a specific data value from the table
            if tv_cell.text_content() == "Harare":
                print('***** Harare found *****')
    # other actions...
    print('Test case 01 >> PASSED!')
    page.wait_for_timeout(3000)
    browser.close()
    print('All opened browser pages are now closed.')

# == import and install files and libraries ==
# from playwright.sync_api import sync_playwright
```

```
C:\Users\njmlo\PycharmProjects\Playwright\playwrightprj1\.venv\Scripts\python.exe C:\Users\njmlo\PycharmProjects\Playwright\playwrightprj1\test
Test case 01 START!
Table found...
This is the table row count >> 197
This is the table data count >> 985
Visited

Country

Capital(s)

Currency

Primary Language(s)

Afghanistan
Kabul
Afghani
Dari Persian; Pashto

Albania
Tirane
Lek
Albanian

Algeria
Algiers
Algerian Dinar
Arabic; Tamazight; French
```

```
Yemen
Sanaa
Yemeni Rial
Arabic

Zambia
Lusaka
Zambian Kwacha
English

Zimbabwe
Harare
***** Harare found *****
United States Dollar
English
Test case 01 >> PASSED!
All opened browser pages are now closed.

Process finished with exit code 0
```

14. Working on web page upload and download of files

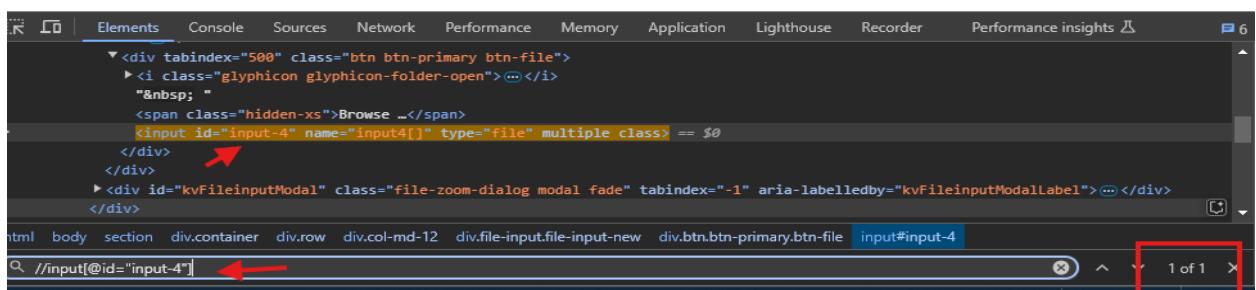
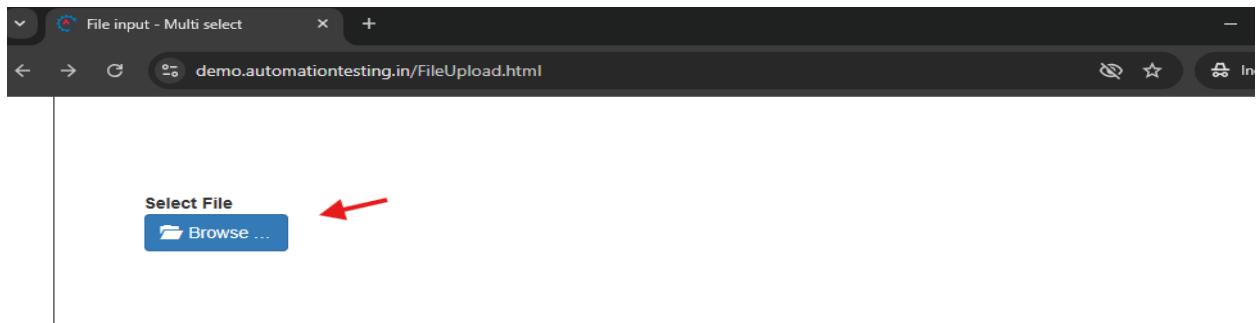
Test page url:

<https://demo.automationtesting.in/FileUpload.html>

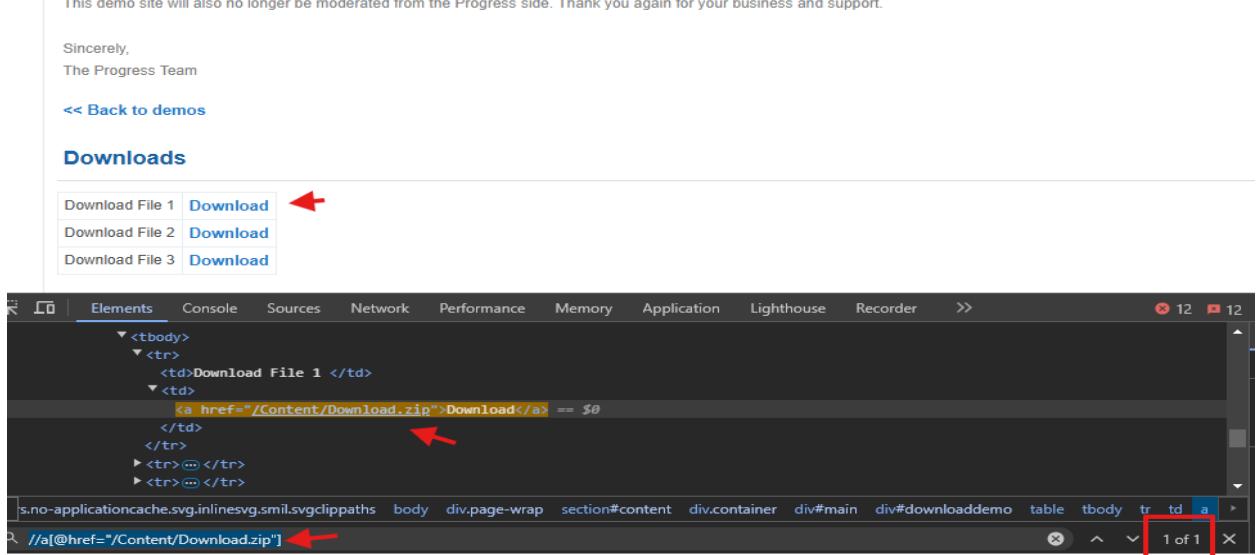
<https://demo.imacros.net/Automate/Downloads>

Locate (inspect) the element attributes for use as its locator:

Upload page...



Download page...



Syntax:

- a. Test case 01 – file upload
- b. Test case 01 – file download

Using the XPath locator, create and run the code:

The screenshot shows the PyCharm IDE interface. On the left is the Project tool window displaying a file structure under 'playwrightprj1'. It includes files like 'test.zip', 'test_file.txt', 'file_upload.py' (which is selected), and 'main.py'. On the right is the code editor window showing the 'file_upload.py' script. The script uses the Playwright library to upload a file. Red arrows point to the file 'test_file.txt' in the project tree and to the line of code where it is specified in the script.

```
4     with sync_playwright() as p:
5         browser = p.chromium.launch(headless=False)
6         context = browser.new_context()
7         page = context.new_page()
8
9         # =====START OF TEST=====
11     # >> Test case 01 - file upload
12     print('Test case 01 START!')
13     page.goto('https://demo.automationtesting.in/FileUpload.html')
14     print('browser page title is ' + page.title() + '...')
15
16     # file to upload
17     file_upload = './files/test_file.txt'
18
19     # file upload object/elements
20     upload_location = page.wait_for_selector('//input[@id="input-4"]')
21
22     # to upload the file
23     upload_location.set_input_files(file_upload)
24     print('File uploaded.')
25     # other actions...
26     page.wait_for_timeout(10000)
27     print('Test case 01 >> PASSED!')
28     browser.close()
```

The screenshot shows the PyCharm terminal window. It displays the command used to run the script and the resulting output. The output shows the test case starting, the browser title, the file being uploaded, and the test case passing.

```
C:\Users\njmlo\PycharmProjects\Playwright\playwrightprj1\.venv\Scripts\python.exe C:/Users/njmlo/PycharmProjects/Playwright/playwrightprj1/test
Test case 01 START!
browser page title is "File input - Multi select"...
File uploaded.
Test case 01 >> PASSED!
Process finished with exit code 0
```

The screenshot shows the PyCharm IDE interface. On the left is the Project tool window displaying a file tree. A red arrow points to the file 'test.zip' under the 'files' directory. Another red arrow points to the file 'file_download.py' under the 'tests' directory. The main editor window on the right contains Python code for file download testing using the playwright sync API.

```
# == import and install files and libraries==
from playwright.sync_api import sync_playwright

# == methods ==
def download_handle(download):
    location_file = './files/test.zip'
    download.save_as(location_file)

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False)
    context = browser.new_context()
    page = context.new_page()

    # =====START OF TEST=====
    # |> Test case 01 - file download
    print('Test case 01 START!')
    page.goto('https://demo.imacros.net/Automate/Downloads')
    print('browser page title is ' + page.title() + '...')
    # to download the file
    page.on('download', download_handle)
    page.wait_for_selector('//a[@href="/Content/Download.zip"]').click()
    print('File downloaded.')
    # other actions...
    page.wait_for_timeout(5000)
    print('Test case 01 >> PASSED!')
    browser.close()
```

The screenshot shows the Run tool window in PyCharm. It displays the output of the executed test script. The output text is as follows:

```
C:\Users\njmlo\PycharmProjects\Playwright\playwrightprj1\.venv\Scripts\python.exe C:\Users\njmlo\PycharmProjects\Playwright\playwrightprj1\tests\file_download.py
Test case 01 START!
browser page title is "Web Automation Test Pages : Downloads"...
File downloaded.
Test case 01 >> PASSED!
```

At the bottom of the window, it says 'Process finished with exit code 0'.

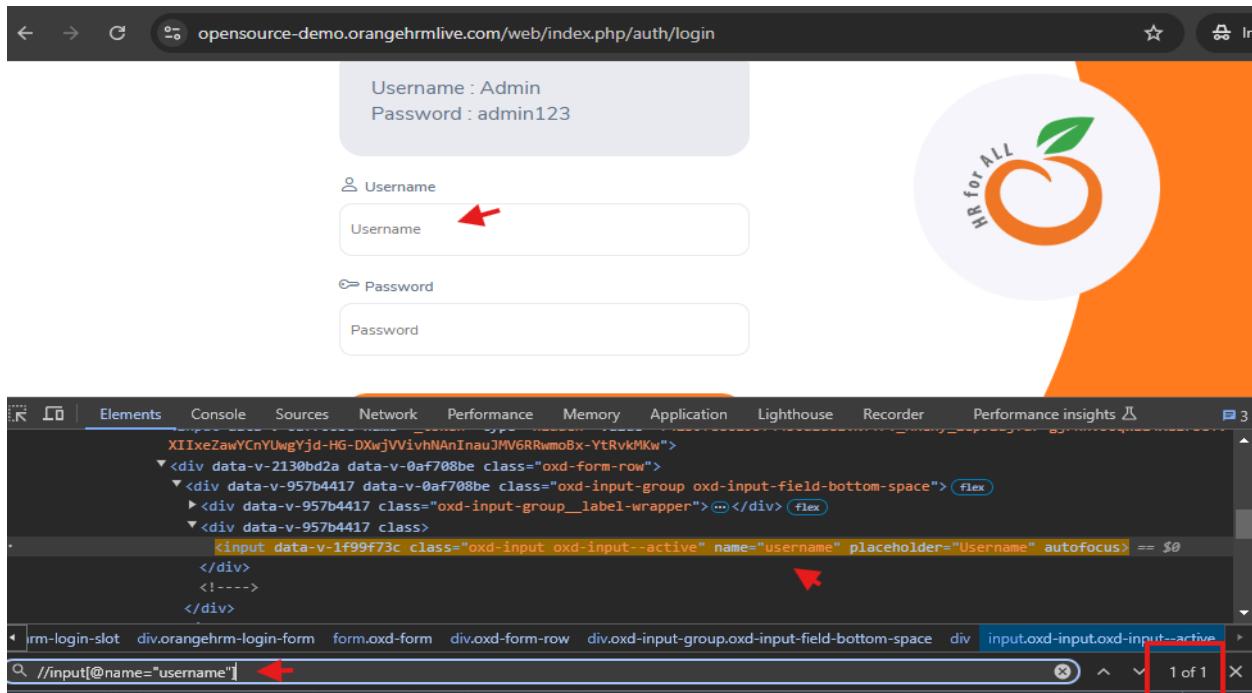
15. Working on web page desktop video recording and screenshots

Test page url:

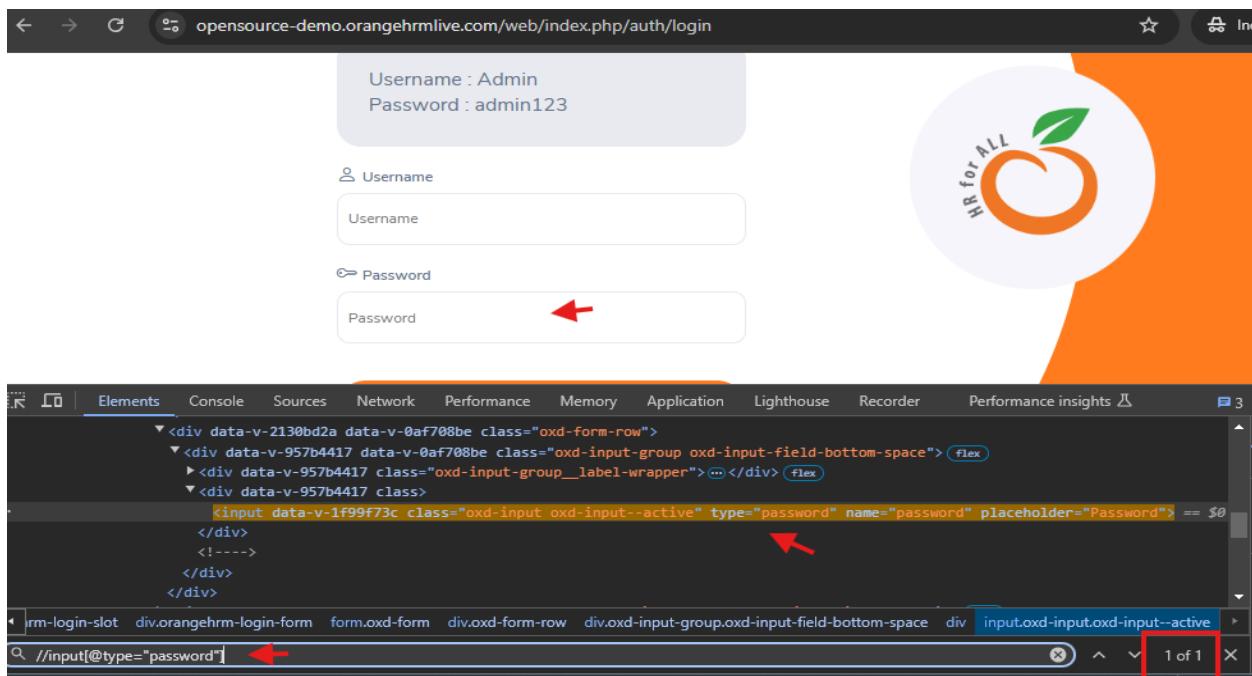
<https://opensource-demo.orangehrmlive.com/web/index.php/auth/login>

Locate (inspect) the element attributes for use as its locator:

Username and Password textboxes...



The screenshot shows a web browser displaying a login form. The URL in the address bar is <https://opensource-demo.orangehrmlive.com/web/index.php/auth/login>. The page has a light gray header with the text "Username : Admin" and "Password : admin123". Below this is a form with two input fields: "Username" and "Password". Red arrows point to both of these fields. At the bottom of the screen, the browser's developer tools are visible, specifically the "Elements" tab. In the search bar of the tools, the XPath expression `/input[@name="username"]` is entered. The results pane shows one match, indicated by a red box around the "1 of 1" text.



This screenshot is similar to the previous one, showing the same login page and developer tools. The main difference is that the red arrows now point to the "Password" field instead of the "Username" field. The developer tools show the same XPath expression `/input[@type="password"]` in the search bar, and the results pane indicates "1 of 1" result, with a red box highlighting this text.

Submit button...

The screenshot shows the OrangeHRM login page with a password field and a large orange 'Login' button. Below the page, the developer tools Elements tab is open, displaying the DOM structure. A red arrow highlights the 'button[@type="submit"]' selector in the search bar of the tools. The status bar at the bottom right of the browser window indicates '1 of 1'.

Syntax:

a. Test case 01 - page video recording and screenshot

Using the XPath locator, create and run the code:

```
playwrightprj1 C:\Users\njmlo\PycharmProjects\playwrightprj1
  .venv library root
    tests
      files
        videos
          4dce6bf064c2b1232c9
            screenshot.png
            test.png
            test.zip
            test_file.txt
            ajax_call.py
            alerts_dialogbox.py
            chrome_browser_launch.py
            cookies_screenshot.py
            dropdown_select.py
            file_download.py
            file_upload.py
            list_err_handling.py
            page_actions.py
            page_switch.py
            page_table.py
            radio_checkbox.py
            record_screenshot.py
            using_css_locators.py

record_screenshot.py
with sync_playwright() as p:
    browser = p.chromium.launch(headless=False)
    context = browser.new_context(record_video_dir='./files/videos')
    page = context.new_page()
    # >> Test case 01 - page video recording and screenshot
    print('Test case 01 START!')
    page.goto('https://opensource-demo.orangehrmlive.com/web/index.php/auth/login')
    print('browser page title is ' + ' ' + page.title() + '...')
    # fill the login fields
    page.wait_for_timeout(5000)
    page.wait_for_selector('//input[@name="username"]').fill('Admin')
    page.wait_for_selector('//input[@type="password"]').fill('admin123')
    # take a screenshot before logging in
    page.screenshot(path='./files/screenshot.png')
    print('Screenshot captured.')
    page.wait_for_timeout(5000)
    # login into the page
    page.query_selector('//button[@type="submit"]').click()
    # other actions...
    page.wait_for_timeout(5000)
    print('Video captured.')
    print('Test case 01 >> PASSED!')
    browser.close()
```

```

Run record_screenshot ×

C:\Users\njmlo\PycharmProjects\Playwright\playwrightprj1\.venv\Scripts\python.exe C:\Users\njmlo\PycharmProjects\Playwright\playwrightprj1\
Test case 01 START!
browser page title is "OrangeHRM"...
Screenshot captured.
Video captured.
Test case 01 >> PASSED!

Process finished with exit code 0

```

16. Working on web page with Playwright and PyTest

Check PyTest if installed and its version:

```

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

(.venv) PS C:\Users\njmlo\PycharmProjects\Playwright\playwrightprj2> pytest --version
pytest 8.3.3

```

The PyCharm interface shows the project structure with a file named `test_chrome_browser.py`. The code within the file is as follows:

```

# ===== IMPORT FILES AND LIBRARIES =====
from playwright.sync_api import Playwright
def test_login(playwright: Playwright) -> None:
    browser = playwright.chromium.launch(headless=False)
    context = browser.new_context()
    page = context.new_page()
    page.goto('https://opensource-demo.orangehrmlive.com/web/index.php/auth/login')
    assert page.title() == 'OrangeHRM'
    print(page.title())
    page.wait_for_timeout(5000)

```

The run results show the test passed:

```

Python tests for test_chrome_browser.test_login ×
Test Results
  ✓ Tests passed: 1 of 1 test – 6 sec 996 ms
  ✓ test_chrome_browser
    ✓ test_login
      6 sec 996 ms
C:\Users\njmlo\PycharmProjects\Playwright\playwrightprj2\.venv\Scripts\python.exe "C:/Program Files/...
Testing started at 1:08 p.m. ...
Launching pytest with arguments test_chrome_browser.py::test_login --no-header --no-summary -q in C:...
=====
collecting ... collected 1 item
test_chrome_browser.py::test_login PASSED [100%]OrangeHRM
=====
1 passed in 7.90s =====
Process finished with exit code 0

```

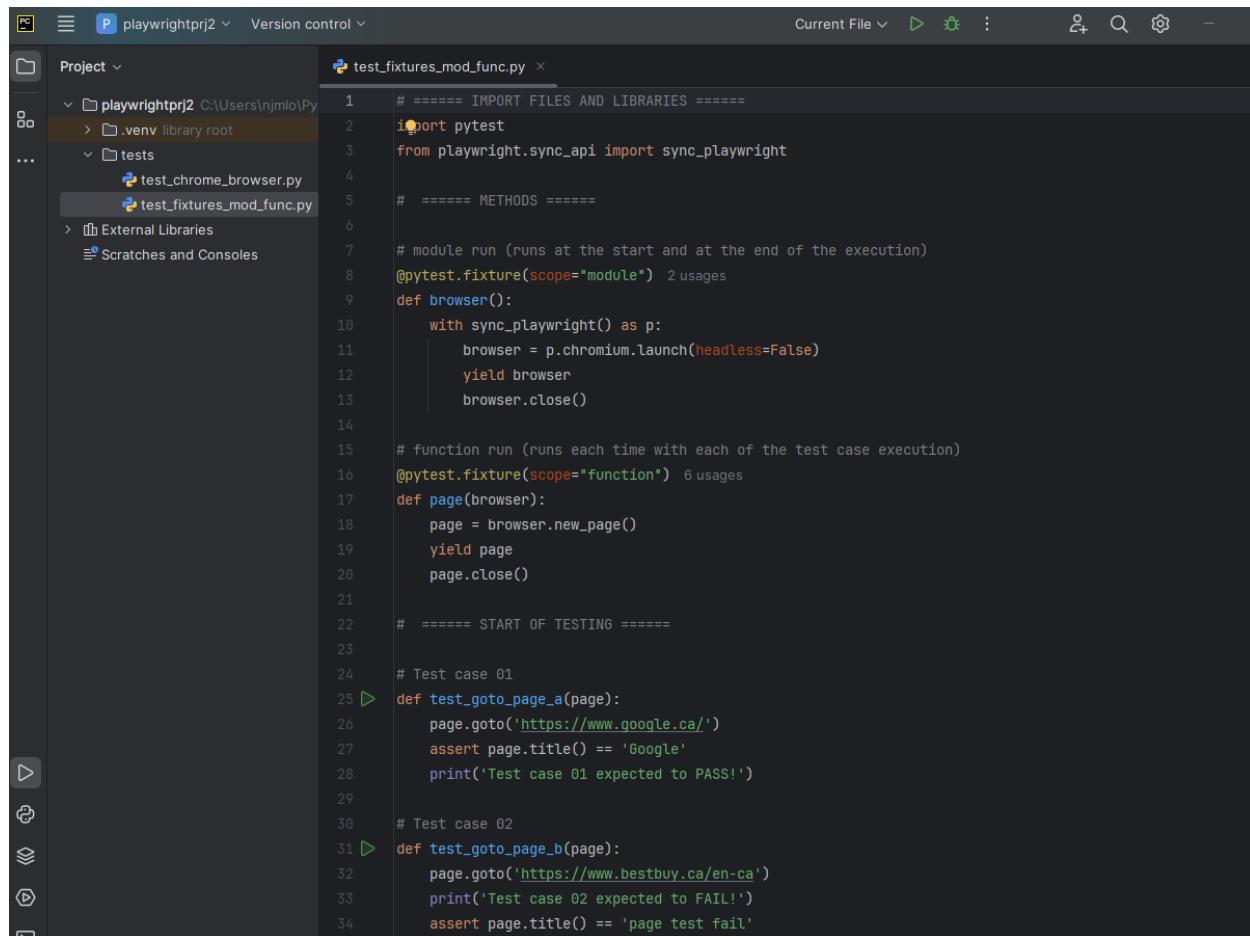
```

(.venv) PS C:\Users\njmlo\PycharmProjects\Playwright\playwrightprj2> python -m pytest
=====
platform win32 -- Python 3.12.6, pytest-8.3.4, pluggy-1.5.0
rootdir: C:\Users\njmlo\PycharmProjects\Playwright\playwrightprj2
plugins: base-url-2.1.0, playwright-0.6.2
collected 1 item
tests\test_chrome_browser.py .
=====
1 passed in 8.18s =====
(.venv) PS C:\Users\njmlo\PycharmProjects\Playwright\playwrightprj2>

```

Bottom status bar: 3:1 CRLF UTF-8 4 spaces Python 3.12 (playwrightprj2)

17. Working on web page with Fixtures (module and functions)



The screenshot shows the PyCharm IDE interface with the following details:

- Project:** playwrightprj2
- File:** test_fixtures_mod_func.py
- Code Content:** The code defines fixtures for a browser and a page, and two test cases (test_goto_page_a and test_goto_page_b) using the sync API of playwright.

```
# ===== IMPORT FILES AND LIBRARIES =====
import pytest
from playwright.sync_api import sync_playwright

# ===== METHODS =====

# module run (runs at the start and at the end of the execution)
@pytest.fixture(scope="module") 2 usages
def browser():
    with sync_playwright() as p:
        browser = p.chromium.launch(headless=False)
        yield browser
        browser.close()

# function run (runs each time with each of the test case execution)
@pytest.fixture(scope="function") 6 usages
def page(browser):
    page = browser.new_page()
    yield page
    page.close()

# ===== START OF TESTING =====

# Test case 01
def test_goto_page_a(page):
    page.goto('https://www.google.ca/')
    assert page.title() == 'Google'
    print('Test case 01 expected to PASS!')

# Test case 02
def test_goto_page_b(page):
    page.goto('https://www.bestbuy.ca/en-ca')
    print('Test case 02 expected to FAIL!')
    assert page.title() == 'page test fail'
```

The screenshot shows a PyCharm interface with a terminal window displaying the results of a pytest test run. The terminal output is as follows:

```
Tests failed: 1, passed: 1 of 2 tests - 6 sec 797 ms
C:\Users\njmlo\PycharmProjects\Playwright\playwrightprj2\.venv\Scripts\python.exe "C:/Program Files/JetBrains/PyCharm Community Edition 2021.2.1\helpers\pycharm\pytest\pytest.py" -v
Testing started at 1:43 p.m. ...
Launching pytest with arguments C:\Users\njmlo\PycharmProjects\Playwright\playwrightprj2\tests\test_fixtures.py

===== test session starts =====
collecting ... collected 2 items

test_fixtures.py::test_goto_page_a PASSED [ 50%]Test case 01 expected to PASS!

test_fixtures.py::test_goto_page_b FAILED [100%]Test case 02 expected to FAIL!

test_fixtures.py:28 (test_goto_page_b)
'Best Buy: Shop Online For Deals & Save | Best Buy Canada' != 'page test fail'

Expected :'page test fail'
Actual   :'Best Buy: Shop Online For Deals & Save | Best Buy Canada'
<Click to see difference>

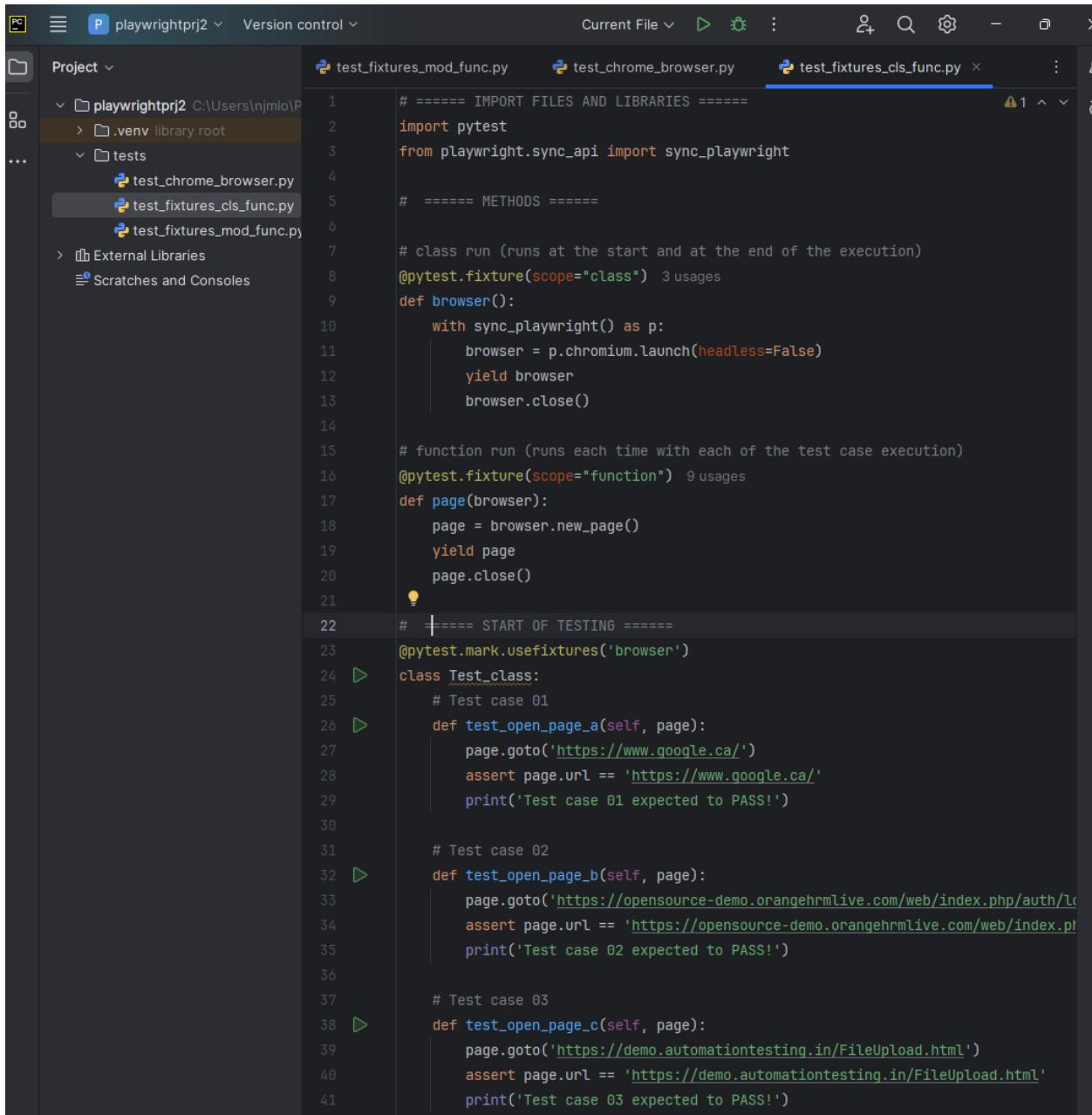
page = <Page url='https://www.bestbuy.ca/en-ca'>

def test_goto_page_b(page):
    page.goto('https://www.bestbuy.ca/en-ca')
    print('Test case 02 expected to FAIL!')
>     assert page.title() == 'page test fail'
E     AssertionError: assert 'Best Buy: Shop Online For Deals & Save | Best Buy Canada' == 'page test fail'
E
E         - page test fail
E         + Best Buy: Shop Online For Deals & Save | Best Buy Canada

test_fixtures.py:32: AssertionError

===== 1 failed, 1 passed in 9.36s =====
```

18. Working on web page with Fixtures (class and functions)



The screenshot shows the PyCharm IDE interface with the following details:

- Project:** playwrightprj2
- Current File:** test_fixtures_cls_func.py
- Code Content:** Python code using the playwright sync API. It defines fixtures for browser and page, and a Test class with three test methods (test_open_page_a, test_open_page_b, test_open_page_c) that assert the URL of the opened page.

```
# ===== IMPORT FILES AND LIBRARIES =====
import pytest
from playwright.sync_api import sync_playwright

# ===== METHODS =====

# class run (runs at the start and at the end of the execution)
@pytest.fixture(scope="class") 3 usages
def browser():
    with sync_playwright() as p:
        browser = p.chromium.launch(headless=False)
        yield browser
        browser.close()

# function run (runs each time with each of the test case execution)
@pytest.fixture(scope="function") 9 usages
def page(browser):
    page = browser.new_page()
    yield page
    page.close()

# ===== START OF TESTING =====
@pytest.mark.usefixtures('browser')
class Test_class:
    # Test case 01
    def test_open_page_a(self, page):
        page.goto('https://www.google.ca/')
        assert page.url == 'https://www.google.ca/'
        print('Test case 01 expected to PASS!')

    # Test case 02
    def test_open_page_b(self, page):
        page.goto('https://opensource-demo.orangehrmlive.com/web/index.php/auth/login')
        assert page.url == 'https://opensource-demo.orangehrmlive.com/web/index.php/auth/login'
        print('Test case 02 expected to PASS!')

    # Test case 03
    def test_open_page_c(self, page):
        page.goto('https://demo.automationtesting.in/FileUpload.html')
        assert page.url == 'https://demo.automationtesting.in/FileUpload.html'
        print('Test case 03 expected to PASS!')
```

Run Python tests for test_fixtures_cls_func.Test_class

Test Result 6 sec 764 ms

Tests passed: 3 of 3 tests – 6 sec 764 ms

C:\Users\njmlo\PycharmProjects\Playwright\playwrightprj2\.venv\Scripts\python.exe "C:/Program
Testing started at 2:32 p.m. ...
Launching pytest with arguments test_fixtures_cls_func.py::Test_class --no-header --no-summary
===== test session starts =====
collecting ... collected 3 items

test_fixtures_cls_func.py::Test_class::test_open_page_a
test_fixtures_cls_func.py::Test_class::test_open_page_b
test_fixtures_cls_func.py::Test_class::test_open_page_c

===== 3 passed in 9.28s =====
PASSED [33%] Test case 01 expected to PASS!
PASSED [66%] Test case 02 expected to PASS!
PASSED [100%] Test case 03 expected to PASS!

Process finished with exit code 0

Terminal Local +

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! <https://aka.ms/PSWindows>

(.venv) PS C:\Users\njmlo\PycharmProjects\Playwright\playwrightprj2> pytest
===== test session starts =====
platform win32 -- Python 3.12.6, pytest-8.3.4, pluggy-1.5.0
rootdir: C:\Users\njmlo\PycharmProjects\Playwright\playwrightprj2
plugins: base-url-2.1.0, playwright-0.6.2
collected 3 items

tests\test_fixtures_cls_func.py ... [100%]

===== 3 passed in 8.67s =====
(.venv) PS C:\Users\njmlo\PycharmProjects\Playwright\playwrightprj2>

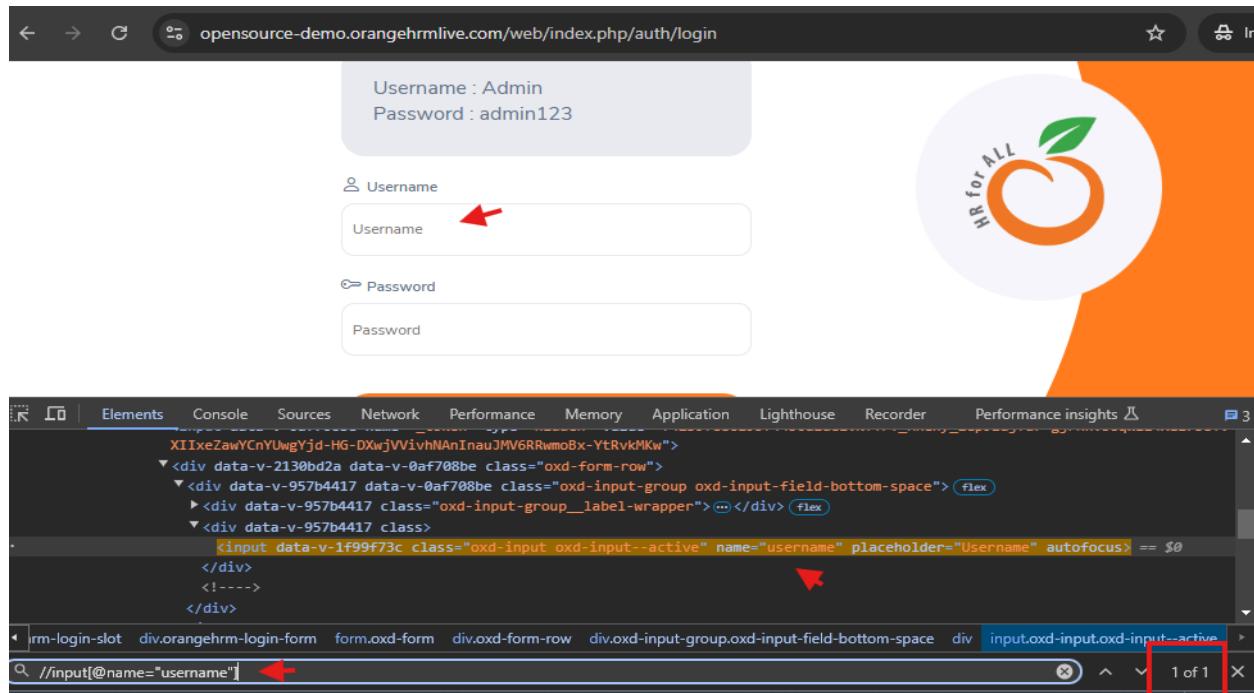
19. Working on web page parameterizing tests in PyTest with Playwright

Test page url:

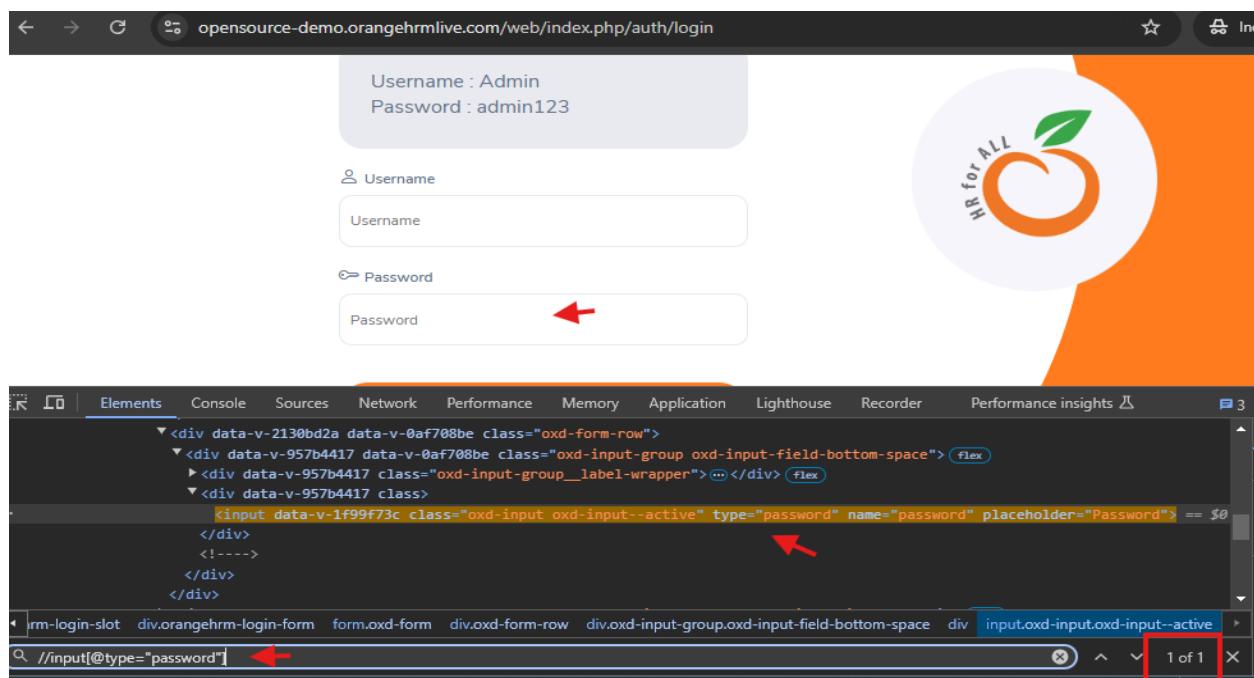
<https://opensource-demo.orangehrmlive.com/web/index.php/auth/login>

Locate (inspect) the element attributes for use as its locator:

Username and Password textboxes...

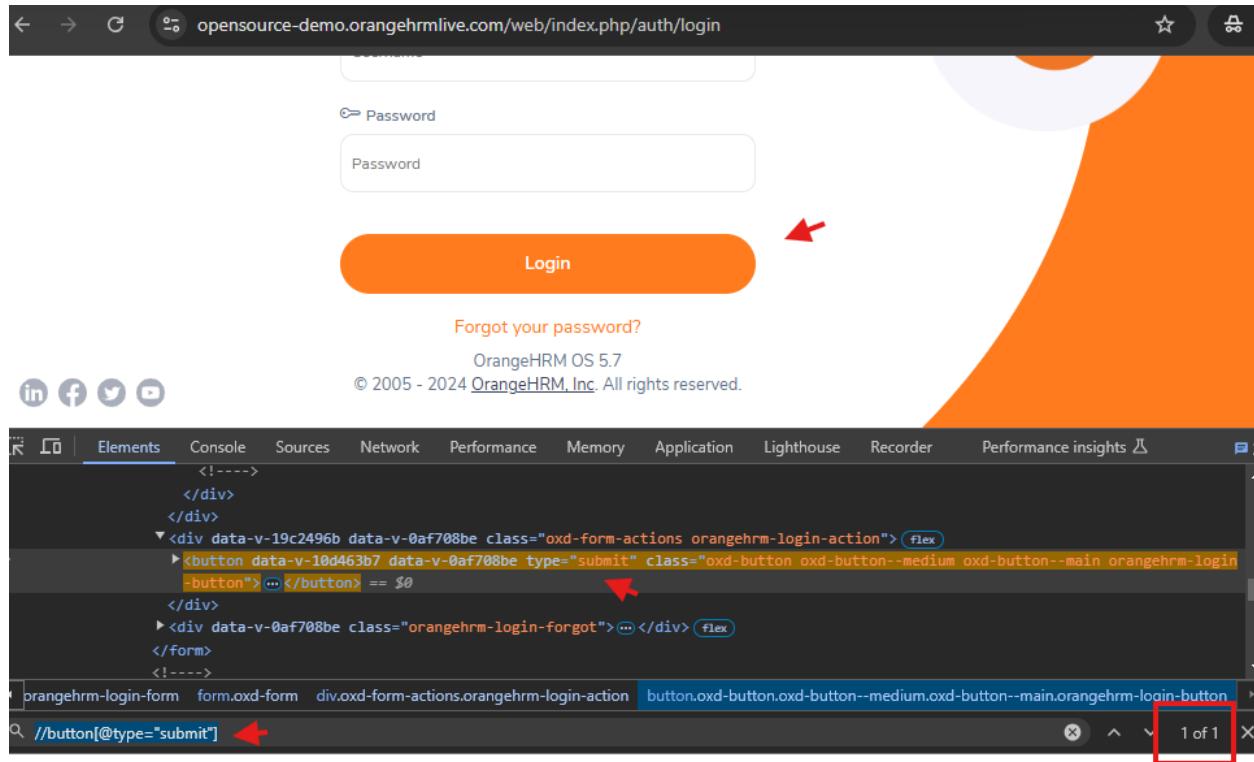


The screenshot shows the OrangeHRM login page. A tooltip at the top indicates the current user is 'Admin' with password 'admin123'. The 'Username' input field is highlighted with a red arrow. In the developer tools' Elements tab, the search bar contains the locator '/input[@name="username"]', which has a red arrow pointing to it. A red box highlights the '1 of 1' result count on the right.



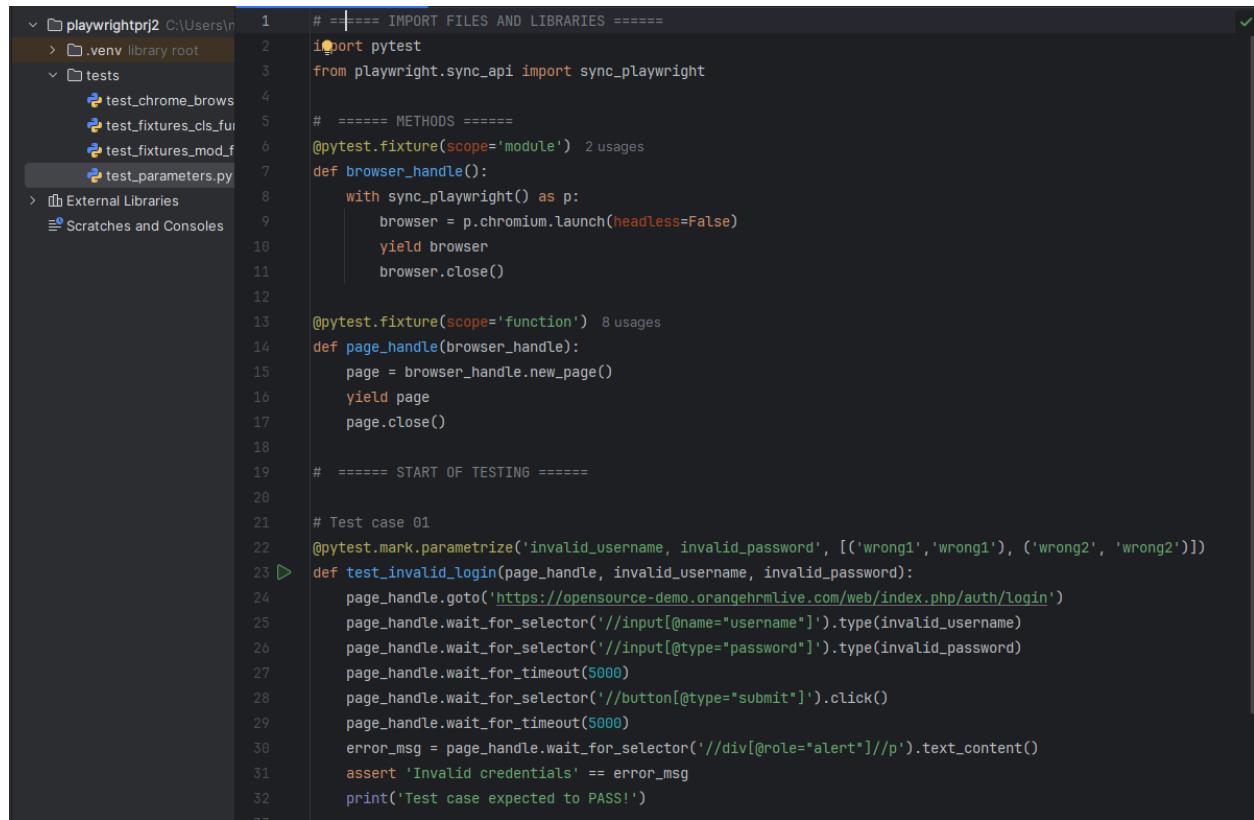
The screenshot shows the OrangeHRM login page. The 'Password' input field is highlighted with a red arrow. In the developer tools' Elements tab, the search bar contains the locator '/input[@type="password"]', which has a red arrow pointing to it. A red box highlights the '1 of 1' result count on the right.

Submit button...



The screenshot shows a web browser displaying the OrangeHRM login page. The URL is opensource-demo.orangehrmlive.com/web/index.php/auth/login. The page features a large orange header with the OrangeHRM logo. Below it is a form with a password input field and a large orange 'Login' button. A red arrow points to the 'Login' button. The developer tools are open, with the 'Elements' tab selected. The search bar at the bottom left contains the query `//button[@type='submit']`. The results list shows one item, which is highlighted with a red box. The full code snippet for the button is:

```
<button data-v-10d463b7 data-v-0af708be type="submit" class="oxd-button oxd-button--medium oxd-button--main orangehrm-login-button"></button>
```



```
# ===== IMPORT FILES AND LIBRARIES =====
import pytest
from playwright.sync_api import sync_playwright

# ===== METHODS =====
@pytest.fixture(scope='module') 2 usages
def browser_handle():
    with sync_playwright() as p:
        browser = p.chromium.launch(headless=False)
        yield browser
        browser.close()

@pytest.fixture(scope='function') 8 usages
def page_handle(browser_handle):
    page = browser_handle.new_page()
    yield page
    page.close()

# ===== START OF TESTING =====

# Test case 01
@pytest.mark.parametrize('invalid_username, invalid_password', [('wrong1', 'wrong1'), ('wrong2', 'wrong2')])
def test_invalid_login(page_handle, invalid_username, invalid_password):
    page_handle.goto('https://opensource-demo.orangehrmlive.com/web/index.php/auth/login')
    page_handle.wait_for_selector('//input[@name="username"]').type(invalid_username)
    page_handle.wait_for_selector('//input[@type="password"]').type(invalid_password)
    page_handle.wait_for_timeout(5000)
    page_handle.wait_for_selector('//button[@type="submit"]').click()
    page_handle.wait_for_timeout(5000)
    error_msg = page_handle.wait_for_selector('//div[@role="alert"]//p').text_content()
    assert 'Invalid credentials' == error_msg
    print('Test case expected to PASS!')
```

```

    ✓  ✓ Test Results          24 sec 522 ms
    ✓  ✓ test_parameters      24 sec 522 ms
    ✓  ✓ test_invalid_login   24 sec 522 ms
        ✓ (wrong1-wrong1)     12 sec 262 ms
        ✓ (wrong2-wrong2)     12 sec 260 ms

    ✓ Tests passed: 2 of 2 tests – 24 sec 522 ms

C:\Users\njml0\PycharmProjects\Playwright\playwrightprj2\.venv\Scripts\python.exe "C:/Program Files/J...
Testing started at 3:17 p.m. ...
Launching pytest with arguments C:\Users\njml0\PycharmProjects\Playwright\playwrightprj2\tests\test_p...
=====
collecting ... collected 2 items

test_parameters.py::test_invalid_login[wrong1-wrong1]
test_parameters.py::test_invalid_login[wrong2-wrong2]

=====
2 passed in 26.54s =====
PASSED      [ 50%]Test case expected to PASS!
PASSED      [100%]Test case expected to PASS!

Process finished with exit code 0

```

```

(.venv) PS C:\Users\njml0\PycharmProjects\Playwright\playwrightprj2> pytest
=====
platform win32 -- Python 3.12.6, pytest-8.3.4, pluggy-1.5.0
rootdir: C:\Users\njml0\PycharmProjects\Playwright\playwrightprj2
plugins: base-url-2.1.0, playwright-0.6.2
collected 2 items

tests\test_parameters.py .. [100%]

=====
2 passed in 26.45s =====
(.venv) PS C:\Users\njml0\PycharmProjects\Playwright\playwrightprj2> █

```

20. Working on web page test reporting in PyTest with Playwright (PyTest HTML plugin)

Install the PyTest-HTML

```

(.venv) PS C:\Users\njml0\PycharmProjects\Playwright\playwrightprj2> pip install pytest-html
Collecting pytest-html
  Using cached pytest_html-4.1.1-py3-none-any.whl.metadata (3.9 kB)
Collecting jinja2>=3.0.0 (from pytest-html)
  Using cached jinja2-3.1.4-py3-none-any.whl.metadata (2.6 kB)
Collecting pytest-metadata>=2.0.0 (from pytest-html)
  Using cached pytest_metadata-3.1.1-py3-none-any.whl.metadata (8.6 kB)
Requirement already satisfied: pytest>=7.0.0 in c:\users\njml0\pycharmprojects\playwright\playwrightprj2\.venv\lib\site-packages (from pytest-html)
 (8.3.4)
Collecting MarkupSafe>=2.0 (from jinja2>=3.0.0->pytest-html)
  Downloading MarkupSafe-3.0.2-cp312-cp312-win_amd64.whl.metadata (4.1 kB)
Requirement already satisfied: colorama in c:\users\njml0\pycharmprojects\playwright\playwrightprj2\.venv\lib\site-packages (from pytest>=7.0.0->py...
test-html) (0.4.6)
Requirement already satisfied: iniconfig in c:\users\njml0\pycharmprojects\playwright\playwrightprj2\.venv\lib\site-packages (from pytest>=7.0.0->py...
test-html) (2.0.0)
Requirement already satisfied: packaging in c:\users\njml0\pycharmprojects\playwright\playwrightprj2\.venv\lib\site-packages (from pytest>=7.0.0->py...
test-html) (42.4)
Requirement already satisfied: pluggy<,>=1.5 in c:\users\njml0\pycharmprojects\playwright\playwrightprj2\.venv\lib\site-packages (from pytest>=7.0...
.0->pytest-html) (1.5.0)
Using cached pytest_html-4.1.1-py3-none-any.whl (23 kB)
Using cached jinja2-3.1.4-py3-none-any.whl (133 kB)
Using cached pytest_metadata-3.1.1-py3-none-any.whl (11 kB)
Downloading MarkupSafe-3.0.2-cp312-cp312-win_amd64.whl (15 kB)
Installing collected packages: MarkupSafe, pytest-metadata, jinja2, pytest-html
Successfully installed MarkupSafe-3.0.2 jinja2-3.1.4 pytest-html-4.1.1 pytest-metadata-3.1.1

```

Cmd line syntax:

```
pytest --html=./reports/report1.html
```

The screenshot shows a terminal window with the following content:

```
Terminal Local x
=====
===== 1 passed, 7 errors in 8.60s =====
(.venv) PS C:\Users\njmlo\PycharmProjects\Playwright\playwrightprj2> pytest --html=./reports/report1.html ↗
=====
===== test session starts =====
platform win32 -- Python 3.12.6, pytest-8.3.4, pluggy-1.5.0
rootdir: C:\Users\njmlo\PycharmProjects\Playwright\playwrightprj2
plugins: base-url-2.1.0, html-4.1.1, metadata-3.1.1, playwright-0.6.2
collected 8 items

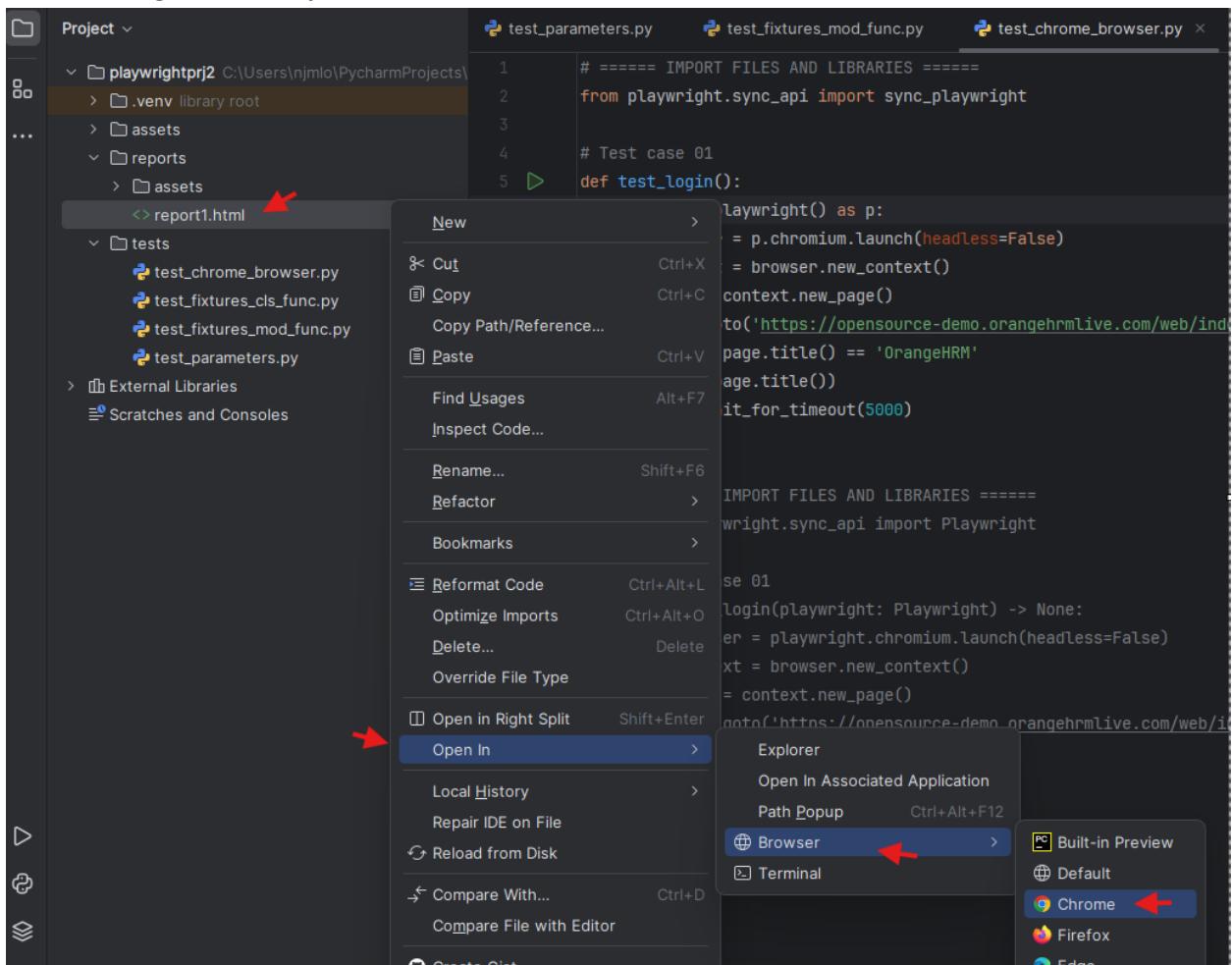
tests\test_chrome_browser.py . [ 12%]
tests\test_fixtures_cls_func.py ... [ 50%]
tests\test_fixtures_mod_func.py .F [ 75%]
tests\test_parameters.py .. [100%]

=====
===== FAILURES =====
----- test_goto_page_b -----
page = <Page url='https://www.bestbuy.ca/en-ca'>

def test_goto_page_b(page):
    page.goto('https://www.bestbuy.ca/en-ca')
    print('Test case 02 expected to FAIL!')
>     assert page.title() == 'page test fail'
E     AssertionError: assert 'Best Buy: Sh...st Buy Canada' == 'page test fail'
E
E         - page test fail
E         + Best Buy: Shop Online For Deals & Save | Best Buy Canada

tests\test_fixtures_mod_func.py:34: AssertionError
----- Captured stdout call -----
Test case 02 expected to FAIL!
----- Generated html report: file:///C:/Users/njmlo/PycharmProjects/Playwright/playwrightprj2/reports/report1.html -----
=====
===== short test summary info =====
FAILED tests\test_fixtures_mod_func.py::test_goto_page_b - AssertionError: assert 'Best Buy: Sh...st Buy Canada' == 'page test fail'
=====
===== 1 failed, 7 passed in 51.89s =====
(.venv) PS C:\Users\njmlo\PycharmProjects\Playwright\playwrightprj2> []
```

Look for the generated report



localhost:63342/playwrightprj2/reports/report1.html?_jt=pbI4d3u7stnemtjh55o8l8mfk&_jj_reload=RELOAD... ☆ ⚡ 🌐 🎯 🗃 🗺 🗺 🗺 🗺 🗺

report1.html

Report generated on 02-Dec-2024 at 15:39:45 by [pytest-html](#) v4.1.1

Environment

Python	3.12.6
Platform	Windows-11-10.0.26100-SP0
Packages	<ul style="list-style-type: none"> pytest: 8.3.4 pluggy: 1.5.0
Plugins	<ul style="list-style-type: none"> base-urt: 2.1.0 html: 4.1.1 metadata: 3.1.1 playwright: 0.6.2
Base URL	

Summary

8 tests took 00:00:52.
 (Un)check the boxes to filter the results.

1 Failed, 7 Passed, 0 Skipped, 0 Expected failures, 0 Unexpected passes, 0 Errors, 0 Reruns

Show all details / Hide all details

Result ▾	Test	Duration	Links
Passed	tests/test_parameters.py::test_invalid_login[wrong2-wrong1]	00:00:13	
Passed	tests/test_parameters.py::test_invalid_login[wrong1-wrong1]	00:00:14	
Passed	tests/test_fixtures_mod_func.py::test_goto_page_a	00:00:03	
Passed	tests/test_fixtures_cls_func.py::Test_class::test_open_page_c	00:00:05	
Passed	tests/test_fixtures_cls_func.py::Test_class::test_open_page_b	00:00:02	
Passed	tests/test_fixtures_cls_func.py::Test_class::test_open_page_a	00:00:03	
Passed	tests/test_chrome_browser.py::test_login	00:00:08	
Failed	tests/test_fixtures_mod_func.py::test_goto_page_b	00:00:05	

```
page = <Page url='https://www.bestbuy.ca/en-ca'>
def test_goto_page_b(page):
    page.goto('https://www.bestbuy.ca/en-ca')
    print('Test case 02 expected to FAIL!')
    assert page.title() == 'page test fail'
E   AssertionError: assert 'Best Buy: Sh...st Buy Canada' == 'page test fail'
E
E       - page test fail
E       + Best Buy: Shop Online For Deals & Save | Best Buy Canada
tests\test_fixtures_mod_func.py:34: AssertionError
----- Captured stdout call -----
Test case 02 expected to FAIL!
```

expand [+]

*****END*****