Playwright Automation

Open Browser:

- ❖ **Synchronous vs Asynchronous Programming**
- **Synchronous =** Do one task, wait for it to finish, then do the next task.
  Asynchronous = Start a task, don't unnecessarily wait, and allow other work to continue.

- ❖ **Playwright Test — Cheat Sheet:**

Item	**Type**	**Purpose**	**Example**
@playwright/test	Package	Provides Playwright Test APIs	import { test, expect } from '@playwright/test'
Test	Function	Creates and executes test cases	test("Login Test", async ({page}) => {})
Expect	Function	Performs assertions/verification	await expect(page).toHaveTitle("Home")
Page	Fixture	Represents a browser tab/page	await page.goto("https://example.com")

Syntax : test("Test Case Name", async ({ page }) => {});

- **async:** It is a keyword in JavaScript that is used to declare a function as an asynchronous function.
- The main reason is that an async function allows us to use the "**await**" keyword inside it.
- `test` → Playwright's function used to define a test case.
- `"Test Case Name"` → name/description of the test case.
- `async` → makes the test function asynchronous.
- `({ page })` → gets the **page fixture** provided by Playwright.
- `=>` → arrow function.
- `{ }` → contains the test steps.

Program-I: Open Browser and get URL and get Title

```
import {test, expect} from '@playwright/test'


//@playwright/test: Package
//test: Playwright test Function, it is used to created and execute test cases
//expect: Playwright assertion Function, it is used to perform verification

//Syntax: test("Test Case name", async({page})=>{});

// page : fixture(Built in Object in playwright)/Object/ Page Class Object
// page : It represents Tab/Browser new Tab, It helps you to intract with
browser


//async : Asynchronous

// Synchronous Vs Asynchronous
// Use waiting : "await" keyword

// npx playwright test Open_Browser1.spec.js --project=chromium --headed

test("Verification of Facebook title", async({page})=>{
    await page.goto("https://www.facebook.com/");
    let WebpageTitle = await page.title();  //Facebook
    console.log("Title of Webpage :", WebpageTitle)
});


//Syntax: test("Test Case name", async({page})=>{});

test("Verify Kite Zerodha URL", async({page})=>{
    await page.goto("https://kite.zerodha.com/")
    let WebpageURL = await page.url();  //https://kite.zerodha.com/
    console.log("URL of Webpage: ", WebpageURL)
});
```

Program-II: Navigate between Application

```
import {test, expect} from '@playwright/test'

//npx playwright test Open_Browser2.spec.js --project=chromium --headed

test("Navigation Between Application", async({page})=>{

//Enter Facebook URL
await page.goto("https://www.facebook.com/")
await page.waitForTimeout(2000);

//Enter Kite zrodha URL
await page.goto("https://kite.zerodha.com/")
await page.waitForTimeout(2000);

//go back to the Facebook
await page.goBack();
await page.waitForTimeout(2000);

//Go to Kite Zeodha
await page.goForward();
await page.waitForTimeout(2000);

//reload webpage
await page.reload();
await page.waitForTimeout(2000);

});
```

Built in Locators in playwright:

```
import {test, expect} from '@playwright/test'

//Syntax: test("Test Case name", async({page})=>{});

// npx playwright test Built_in_locators1.spec.js --project=chromium --headed
//1) getByRole() : Locate using accessibility role
test("Get By Role on textbox and button",async({page})=>{
    await page.goto("https://kite.zerodha.com/")
    //Enter UserID
    await page.getByRole('textbox',{name: 'Phone'}).fill("Virat12345");
    //Enter Password
    await page.getByRole('textbox', {name : 'Password'}).fill("Kohli18");
    //Click Login Button
    await page.getByRole('button', {name :'Login '}).click();
    await page.waitForTimeout(4000);
});


//2) getByAltText() : to locate an element, usually image, by its text
alternative.(i.e alttext attribute)
test("Get by Alt text on Logo", async({page})=>{
    await page.goto("https://kite.zerodha.com/");
    let logo = page.getByAltText("Kite logo");
    await expect(logo).toBeVisible();  //Pass  //fail
});

//3) getByPlaceholder(): to locate an input by placeholder.(i.e Placeholder
Attribute)

test("Get By Placeholder on Username and Password", async({page})=>{
    await page.goto("https://opensource-
demo.orangehrmlive.com/web/index.php/auth/login");
    await page.waitForTimeout(2000);
    //Enter Username
    await page.getByPlaceholder("Username").fill("Admin");
    await page.waitForTimeout(2000);
    //Enter Password
    await page.getByPlaceholder("Password").fill("admin123");
    await page.waitForTimeout(2000);

});
```

```
//4) getByText(): to locate by text content. (i.e Inner Text)

test("get By Text on Heading", async({page})=>{

    await page.goto("https://www.saucedemo.com/");
    let HeadingElement = page.getByText("Swag Labs")
    await expect(HeadingElement).toBeVisible();  //Pass fail

});

//5) getByLabel() to locate a form control by associated label's text.(i.e
Inner text but should have "label" tagname)

test.only("get By Label of textbox", async({page})=>{

await page.goto("https://testautomationpractice.blogspot.com/p/playwright
practice.html")
await page.waitForTimeout(2000);
await page.getByLabel("Email Address:").fill("Virat@gmail.com");
await page.waitForTimeout(2000);
await page.getByLabel("Password:").fill("Virat@12345")
await page.waitForTimeout(2000);
});
```

```
import {test, expect} from '@playwright/test'

//Syntax: test("Test Case name", async({page})=>{});

// npx playwright test Built_in_locators2.spec.js --project=chromium --headed

//Built_in_locators1.spec
//6) getByTitle() to locate an element by its title attribute. (i.e "Title"
attribute)

test("Get by Title on link", async({page})=>{
    await
page.goto("https://testautomationpractice.blogspot.com/p/playwrightpractice.ht
ml")
    await page.waitForTimeout(2000);
    await page.getByTitle("Home page link").click();
    await page.waitForTimeout(3000);
 });
```

```
//7) getByTestId() to locate an element based on its "data-testid" attribute
("i.e data-testid" attribute).

test("Get By TestID on text element", async({page})=>{
    await
page.goto("https://testautomationpractice.blogspot.com/p/playwrightpractice.ht
ml")
    await page.waitForTimeout(3000);
    let text = await page.getByTestId("profile-name").textContent();  //John
Doe
    console.log(text);

});
```

Xpath:

1. **Xpath by Attribute:**
   Syntax: //tagname[@attribute name = 'attribute value']

```
import {test, expect} from '@playwright/test'

//Syntax: test("Test Case name", async({page})=>{});
// npx playwright test Xpath1.spec.js --project=chromium --headed

//=============================================================
// Xpath by Attribute
// Syntax: //tagname[@attribute name = 'attribute value']
//=============================================================

test("Xpath by Attribute", async({page})=>{
    await page.goto("https://kite.zerodha.com/")
    //Syntax: //tagname[@attribute name = 'attribute value']
    await page.locator("//input[@id='userid']"). fill("ABCD12345");
    await page.waitForTimeout(2000);
});
```

2. Xpath by text:

Syntax: //Tagname[text()='text value']

```
import {test, expect} from '@playwright/test'

// Syntax: test("Test Case name", async({page})=>{});
// npx playwright test Xpath2.spec.js --project=chromium --headed

//=============================================================
// Xpath by text
// Syntax: //Tagname[text()='text value']
//=============================================================

test("Xpath by text", async({page})=>{

    await page.goto("https://kite.zerodha.com/")
    await page.waitForTimeout(2000);
    //Syntax: //Tagname[text()='text value']
    await page.locator("//a[text()='Forgot user ID or password?']").click()
    await page.waitForTimeout(2000);
});
```

3. Xpath by contains by using attributes-

Syntax: //tagname[contains(@attribute name, 'attribute value')]

- Instead of using full 'attribute value', we can use substring of 'attribute value'.

```
import {test, expect} from '@playwright/test'

//Syntax: test("Test Case name", async({page})=>{});
// npx playwright test Xpath3.spec.js --project=chromium --headed

//=============================================================================
// Xpath By contains using Attribute
// Syntax: //tagname[contains(@attribute name, 'attribute value' )]
// Instead of using full 'attribute value', we can use substring of 'attribute
value'
//=============================================================================

test("Xpath By contains using Attribute", async({page})=>{

    await page.goto("https://opensource-
demo.orangehrmlive.com/web/index.php/auth/login")
    await page.waitForTimeout(2000);
    //syntax: //tagname[contains(@attribute name, 'attribute value' )]
    await page.locator("//p[contains(@class,'oxd-text oxd-text--p orangehrm-
login')]").click();
    await page.waitForTimeout(2000);
});
```

4. Xpath by contains by using text-

Syntax://tagname[contains(text(), 'text value')]

- Instead of using full ‘text value’, we can use substring of ‘text value'.

```
import {test, expect} from '@playwright/test'

//Syntax: test("Test Case name", async({page})=>{});
// npx playwright test Xpath3.spec.js --project=chromium --headed

//=============================================================================
// Xpath By contains using Text
// Syntax: //tagname[contains(text(), 'text value' )]
// Instead of using full ‘text value’, we can use substring of ‘text value'.
//=============================================================================

test("Xpath By Contains using Text", async({page})=>{
    await page.goto("https://kite.zerodha.com/")
    await page.waitForTimeout(2000);
    //syntax: //tagname[contains(text(), 'text value' )]
    await page.locator("//a[contains(text(),'Forgot user ID')]").click();
    await page.waitForTimeout(2000);

});
```

5. Xpath by index:

Syntax : (xpath expression)[index]

```
import {test, expect} from '@playwright/test'

//Syntax: test("Test Case name", async({page})=>{});
// npx playwright test Xpath3.spec.js --project=chromium --headed

//=============================================================================
// Xpath By Index
// Syntax = (xpath expression)[index]
//=============================================================================
test("Xpath By Index", async({page})=>{
    await page.goto("https://www.facebook.com/reg/?entry_point=login")
    await page.waitForTimeout(2000);
    //Syntax = (xpath expression)[index]
    await page.locator("(//input[@type='text'])[2]").fill("Kohli")
    await page.waitForTimeout(2000);

});
```

6.Xpath by starts-with

Syntax = //tagname[starts-with(@attribute name, 'attribute value')]

- Instead of using full 'attribute value', we can use substring of 'attribute value'

```
import {test, expect} from '@playwright/test'

//Syntax: test("Test Case name", async({page})=>{});
// npx playwright test Xpath3.spec.js --project=chromium --headed

//=============================================================================
// Xpath By Start- With
// Syntax = //tagname[starts-with(@attribute name, 'attribute value' )]
// Instead of using full 'attribute value', we can use substring of 'attribute
value'
//=============================================================================

test.only("Xpath By Starts With", async({page})=>{

    await page.goto("https://kite.zerodha.com/")
    await page.waitForTimeout(2000);
    //Syntax = //tagname[starts-with(@attribute name, 'attribute value' )]
    await page.locator("//a[starts-with(@class,'text-light ')]").click();
    await page.waitForTimeout(2000);
});
```

CSS Selector:

CSS Selector is a pattern used to identify and locate HTML elements on a web page.

a) tag id :
   Syntax: #id or tagname#id
b) tag class
   Syntax: .class or tagname.class
c) tag attribute
   Syntax: [attaribute name ='attribute value'] or tagname[attaribute name ='attribute value']
d) tag class and attribute
   Syntax: tagname.class[attaribute name ='attribute value']

```
import {test, expect} from '@playwright/test'

// Syntax: test("Test Case name", async({page})=>{});
// npx playwright test Css1.spec.js --project=chromium --headed

//CSS selectors:
//a) tag id
//b) tag class
//c) tag attribute
//d) tag class and attribute

//================================================
// a) CSS selector : tag id
//    Syntax:  #id  or tagname#id
//================================================
test("Css-tag id", async({page})=>{

    await page.goto("https://demowebshop.tricentis.com/")
    await page.waitForTimeout(2000);
    //await page.locator("#small-searchterms").fill("Books")
                //Or
    await page.locator("input#small-searchterms").fill("Loptop")
    await page.waitForTimeout(2000);
});
```

```
//=============================================
// b) CSS selector : tag class
//    Syntax: .class or tagname.class
//=============================================

test("Css-tag class", async({page})=>{

    await page.goto("https://demowebshop.tricentis.com/")
    await page.waitForTimeout(2000);
    //await page.locator(".search-box-text").fill("Mobile");
                //Or
    await page.locator("input.search-box-text").fill("Bag");
    await page.waitForTimeout(2000);
});

//=============================================================================
//c) CSS selector : tag attribute
// Syntax: [attaribute name ='attribute value'] or tagname[attaribute name
='attribute value']
//=============================================================================

test("Css tag attribute", async({page})=>{

    await page.goto("https://demowebshop.tricentis.com/")
    await page.waitForTimeout(2000);
    //await page.locator("[name= 'q']").fill("Keyboard")
            //OR
    await page.locator("input[name= 'q']").fill("Bottle")
    await page.waitForTimeout(2000);

});

//=============================================================================
// d) CSS selector : tag class and attribute
// Syntax: tagname.class[attaribute name ='attribute value']
//=============================================================================

test.only("Css tag class and attribute", async({page})=>{

    await page.goto("https://demowebshop.tricentis.com/")
    await page.waitForTimeout(2000);
    await page.locator("input.search-box-text[name='q']").fill("Charger")
    await page.waitForTimeout(2000);
});
```

Handling of Inputbox:

```
import {test, expect} from '@playwright/test'

// Syntax: test("Test Case name", async({page})=>{});
// npx playwright test Handling_of_Inputbox.spec.js --project=chromium --
headed

test("Handling of Input field", async({page})=>{

    await page.goto("https://testautomationpractice.blogspot.com/")

    //Identify name field
   let NameInputField = await page.locator("#name");

   //1)Verify Inputbox is Visible or not
    await expect(NameInputField).toBeVisible();  //Pass  //Fail

    //2)Verify Inputbox is Empty or not
    await expect(NameInputField).toBeEmpty();  //Pass Fail

    //3) Verify Inputbox is Editable or not
    await expect(NameInputField).toBeEditable();  //Pass Fail

    //4) Verify Inputbox is Enabled or not
    await expect(NameInputField).toBeEnabled();  //Pass  //Fail

    //5) Enter Value
    await NameInputField.fill("Rohit");

    //6) Get Entered Value
   let EnteredText = await NameInputField.inputValue();  //Rohit
   console.log("Entered Value in inputbox :"+ EnteredText)

   //Verify entered Value
   await expect(NameInputField).toHaveValue("Rohit");  //Rohit==Rohit -->Pass
Fail

});
```

Handling of Checkboxes:

```
import {test, expect} from '@playwright/test'

// Syntax: test("Test Case name", async({page})=>{});
// npx playwright test Handling_of_Checkboxes.spec.js --project=chromium --
headed

test("Handling of Checkboxes", async({page})=>{

await page.goto("https://testautomationpractice.blogspot.com/")

//1) Verify Monday checkbox is Visible or not
let MondayCheckbox = await page.locator("#monday");
await expect(MondayCheckbox).toBeVisible();  //Pass  //Fail

//2) Verify Monday checkbox Enabled or not
await expect(MondayCheckbox).toBeEnabled();  //Pass  //Fail

//3) Select/check single checkbox
await MondayCheckbox.check();

//4) Verify checkbox is checked or not
await expect(MondayCheckbox).toBeChecked();  //Pass  //Fail

//5) Verify Sunday checkbox is Not-checked
let SundayCheckbox = await page.locator("#sunday")
await expect(SundayCheckbox).not.toBeChecked();  //Pass  //Fail

// 6) Count of checkboxes
let Allcheckboxes = await page.locator("//input[@type='checkbox' and
@class='form-check-input']")  //7 Checkboxes addresses/Locators
await expect(Allcheckboxes).toHaveCount(7)  //7==7  //Pass Fail


//Print Count of Checkboxes
let Countofcheckboxes = await Allcheckboxes.count();  //7
console.log(`Total no of Checkboxes : ${Countofcheckboxes}`)


//7) Check/select multiple checkboxes
let checkboxlist = await Allcheckboxes.all()  // //7 Checkboxes
addresses/Locators //[sunday,monday,thu,wed,thur,friday,sat]

for(let checkbox of checkboxlist)  //[sunday,monday,thu,wed,thur,friday,sat]
{
    await checkbox.check();
}
await page.waitForTimeout(4000);
```

```
//Uncheck Sunday Checkbox
await SundayCheckbox.uncheck();
await page.waitForTimeout(2000);

});
```

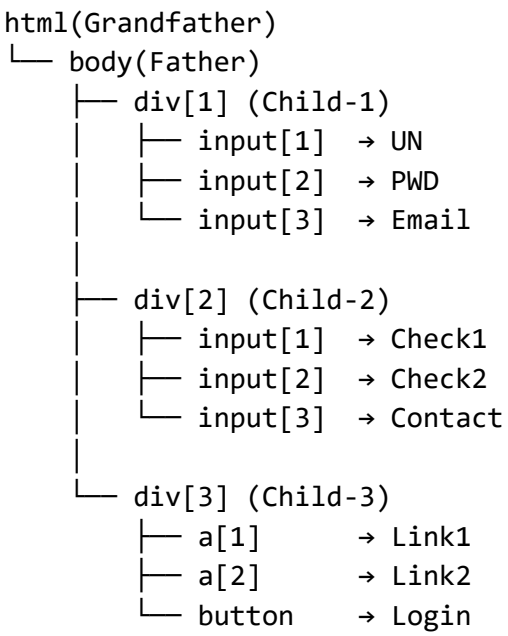
Handling Of Radio Button:

```
import {test, expect} from '@playwright/test'

//================================================
// Handling of Radio Button
//================================================

// Syntax: test("Test Case name", async({page})=>{});
// npx playwright test Handling_of_RadioButton.spec.js --project=chromium --
headed

test("Handling of Radio Button", async({page})=>{

    await page.goto("https://testautomationpractice.blogspot.com/")

    //Identify male Radio Button
    let MaleRadioButton = await page.locator("#male");

    //Verify Male radio Button is Visible or not
    await expect(MaleRadioButton).toBeVisible(); //Pass Fail

    //Verify Male radio Button Enabled or not
    await expect(MaleRadioButton).toBeEnabled();  //Pass Fail

    //Select/Check Male Radio Button
    await MaleRadioButton.check();

    //Verify Male radio Button is checked or not
    await expect(MaleRadioButton).toBeChecked();  //Pass Fail

    //Verify Female radio button is not Checked
    await expect(await page.locator("#female")).not.toBeChecked();  //Pass
//Fail

    //Verify Count of Radio Button
    let Radios = await page.locator("//input[@type='radio']")  //2 Addresses
    await expect(Radios).toHaveCount(2);  //2===2  //Pass  Fail

    await page.waitForTimeout(4000);

});
```

Absolute XPath & Relative XPath:

FORM LAYOUT	HTML STRUCTURE
UN [_____] PWD [_____] Email [_____] Check1 [ ] Check2 [ ] Contact [_____] Link1 Link2 +-----------+ | Login | +-----------+	<html> <body> UN <input type="text"> PWD <input type="password"> Email <input type="text"> Check1 <input type="checkbox"> Check2 <input type="checkbox"> Contact <input type="text"> Link1 Link2 <button type="button">Login </button> </body> </html>

HTML Tree Diagram



Absolute Xpath: Use "/" and We can move from "parent to immediate child"

Absolute XPath of UN: / html/ body/div[1]/ input[1]

Absolute XPath of Contact: / html/**body/div[2]/** input[3]

Absolute XPath of Login: /html/ body/ div[3]/button

Relative Xpath: Use "//" and We can move from "any Parent to any child"

Relative XPath of UN: //div[1]/ input[1]

Relative XPath of Contact: // div[2]/ input[3]

Relative XPath of Login: // div[3]/ button or //button

Handling of Single Selectable Listbox / Dropdownbox:

Program-I:

```
import {test, expect} from '@playwright/test'

//================================================
// Handling of Single selecteble Listbox/Dropdown
//================================================

// Syntax: test("Test Case name", async({page})=>{});
// npx playwright test Handling_of_listbox.spec.js --project=chromium --headed

test("Handling of listbox", async({page})=>{

    await page.goto("https://testautomationpractice.blogspot.com/")

    //Identify Listbox
    let CountryDropdown = await page.locator("#country");

    //Scroll upto WebPage Element
    await CountryDropdown.scrollIntoViewIfNeeded();

    //Verify country listbox is visible or not
    await expect(CountryDropdown).toBeVisible();  //Pass  //Fail

    //Select Option form listbox
    await CountryDropdown.selectOption("India");

    await page.waitForTimeout(4000)

});
```

Program-II:

```
import {test, expect} from '@playwright/test'

//================================================
// Handling of Single selectable Listbox/Dropdown
//================================================

// Syntax: test("Test Case name", async({page})=>{});
// npx playwright test Handling_of_listbox2.spec.js --project=chromium --
headed

test("Handling of Single selectable listbox", async({page})=>{

    await page.goto("https://testautomationpractice.blogspot.com/")

    //Identify Listbox
    let CountryDropdown = await page.locator("#country");

    //Scroll upto WebPage Element
    await CountryDropdown.scrollIntoViewIfNeeded();

    //Verify country listbox is visible or not
    await expect(CountryDropdown).toBeVisible();  //Pass  //Fail

    //Select Option form listbox
    // await CountryDropdown.selectOption("India");
    // await CountryDropdown.selectOption({label:"France"})
    // await CountryDropdown.selectOption({index: 1})
    await CountryDropdown.selectOption({value: "australia"})

    //Validate selected Option
   let SelectedOption = await CountryDropdown.inputValue() // Australia
    console.log("Selected Option :", SelectedOption)


    //Get Count of all options of listbox
    let AllOptionsaddress = await
page.locator("//select[@id='country']/option")  //10 Country Options
    let Optioncount = await AllOptionsaddress.count()  //10
    console.log("Total No of Options: ", Optioncount)
```

```
//Print all Options

        //i=0      0 < 10            1
                  //1<10              2
                  //2<10              3
                  //3<10              4
                  //4<10              5
                  //5<10              6
                  //6<10              7
                  //7<10              8
                  //8<10              9
                  //9<10              10
                  //10<10
    for(let i=0;    i< Optioncount;    i++)
    {
        let Option = await AllOptionsaddress.nth(i);
        let Optiontext = await Option.textContent();
        console.log("Option: ", Optiontext.trim())

    }
    await page.waitForTimeout(4000)
});
```

Handling of Multi Selectable Listbox:

```
import {test, expect} from '@playwright/test'

//================================================
// Handling of Multi selectable Listbox/Dropdown
//================================================

// Syntax: test("Test Case name", async({page})=>{});
// npx playwright test Handling_of_listbox3.spec.js --project=chromium --
headed

test("Handling of Multi Selectable Listbox", async({page})=>{

    await page.goto("https://testautomationpractice.blogspot.com/")

    //Identify Listbox
    let ColorDropdown = await page.locator("//select[@id='colors']")

    //Verify Visibility
    await expect(ColorDropdown).toBeVisible();  //Pass  //Fail

    //Create an array with Options which need to be selected
    let Colour = ["Red","Green","White"];

    //Select Options of listbox
   await ColorDropdown.selectOption(Colour)

   await page.waitForTimeout(4000);


});
```

Pop-up:

Alert pop-up:

```
import {test, expect} from '@playwright/test'

// Syntax: test("Test Case name", async({page})=>{});
// npx playwright test Pop_up1.spec.js --project=chromium --headed

//=========================================
// Handling of Alert popup
//=========================================

test("Pop_up-Alert", async({page})=>{

    await page.goto("https://testautomationpractice.blogspot.com/")

    await page.waitForTimeout(2000);

    //dialog : Event name (Keyword)
    //async(dialog) : variablename

    page.on('dialog', async(dialog)=>{

        //Type of Popup
        let popupType = await dialog.type();  //alert
        console.log("Type of Popup: ", popupType)
        await expect(popupType).toContain("alert")  //Pass Fail
        await page.waitForTimeout(2000);

        //Get Message/Text Present on Popup
        let dialogText = dialog.message()  //I am an alert box!
        console.log("Alert Message :", dialogText)
        //await expect(dialogText).toContain("I am an alert") //Pass Fail
        await expect(dialogText).toBe("I am an alert box!")  //Pass  Fail
        await page.waitForTimeout(2000);

        //Click ok Button
        await page.waitForTimeout(2000);
        await dialog.accept()
    })

    await page.locator("#alertBtn").first().click();

    await page.waitForTimeout(8000);


});
```

Handling of Confirm Pop-up:

Program-I:

```
import {test, expect} from '@playwright/test'

// Syntax: test("Test Case name", async({page})=>{});
// npx playwright test Pop_up2.spec.js --project=chromium --headed

//=========================================
// Handling of Confirm popup
//=========================================

test("Pop_up-Confirm", async({page})=>{

await page.goto("https://testautomationpractice.blogspot.com/")

page.on('dialog', async(dialog)=>{

    //Type of Popup
    let  PopupType = await dialog.type();  //confirm
    console.log("Type of Popup: ", PopupType)
    await expect(PopupType).toContain("confirm")  //Pass //fail


    //get Text/Message present on Popup
    let Dialogtext = dialog.message();  //Press a button!
    console.log("Message present on Popup: ",Dialogtext)
    await expect(Dialogtext).toBe("Press a button!")  //Pass  //Fail

    //Click Ok Button
    await page.waitForTimeout(4000)
    await dialog.accept()

})


//Click on Confrimation Alert Button
await page.locator("//button[@id='confirmBtn']").click();

//verify Message after click Ok Button
let MessageafterclickOkbutton = await page.locator("#demo");
await expect(MessageafterclickOkbutton).toHaveText("You pressed OK!")//Pass
Fail

await page.waitForTimeout(4000)

});
```

Program-II:

```
import {test, expect} from '@playwright/test'

// Syntax: test("Test Case name", async({page})=>{});
// npx playwright test Pop_up3.spec.js --project=chromium --headed

//=========================================
// Handling of Confirm popup
//=========================================

test("Pop_up-Confirm", async({page})=>{

await page.goto("https://testautomationpractice.blogspot.com/")

page.on('dialog', async(dialog)=>{

    //Type of Popup
    let  PopupType = await dialog.type();  //confirm
    console.log("Type of Popup: ", PopupType)
    await expect(PopupType).toContain("confirm")  //Pass //fail


    //get Text/Message present on Popup
    let Dialogtext = dialog.message();  //Press a button!
    console.log("Message present on Popup: ",Dialogtext)
    await expect(Dialogtext).toBe("Press a button!")  //Pass  //Fail

    //Click Ok Button
    await page.waitForTimeout(4000)
    await dialog.dismiss()

})


//Click on Confrimation Alert Button
await page.locator("//button[@id='confirmBtn']").click();

//verify Message after click Ok Button
let Messageafterclickcancel = await page.locator("#demo")  //You pressed
Cancel!
await expect(Messageafterclickcancel).toHaveText("You pressed
Cancel!")  //Pass  Fail

await page.waitForTimeout(4000)

});
```

Handling of Prompt Popup:

Program-I:

```
import {test, expect} from '@playwright/test'

// Syntax: test("Test Case name", async({page})=>{});
// npx playwright test Pop_up4.spec.js --project=chromium --headed

//=========================================
// Handling of prompt popup
//=========================================

test("Handling of prompt popup", async({page})=>{

await page.goto("https://testautomationpractice.blogspot.com/")

    page.on('dialog', async(dialog)=>{

    //Type of Popup
    let popuptype = await dialog.type(); //prompt
    console.log("Type of Popup: ", popuptype)
    await expect(popuptype).toContain("prompt")  //Pass

    //Get text present on popup
    let dialogtext = await dialog.message()  //Please enter your name:
    console.log("Text present on Popup: ", dialogtext)
    await expect(dialogtext).toBe("Please enter your name:")  //Pass Fail

    //Get default value present on popup
    let defaultvalueininputfield = await dialog.defaultValue()  //Harry Potter
    console.log("Default value on popup: ", defaultvalueininputfield)
    await expect(defaultvalueininputfield).toContain("Harry Potter")  //Pass
Fail

    //Enter Value in input field
    await page.waitForTimeout(4000)
    await dialog.accept("Virat");


    })

    await page.locator("//button[@id='promptBtn']").click()

    //Validate mesage on click ok Button
    let Messageafterclickonokbutton = await page.locator("#demo")
    await expect(Messageafterclickonokbutton).toHaveText("Hello Virat! How are
you today?")  //Pass //fail
});
```

Program-II:

```
import {test, expect} from '@playwright/test'

// Syntax: test("Test Case name", async({page})=>{});
// npx playwright test Pop_up5.spec.js --project=chromium --headed

//=========================================
// Handling of prompt popup
//=========================================

test("Handling of prompt popup", async({page})=>{

await page.goto("https://testautomationpractice.blogspot.com/")

    page.on('dialog', async(dialog)=>{

    //Type of Popup
    let popuptype = await dialog.type(); //prompt
    console.log("Type of Popup: ", popuptype)
    await expect(popuptype).toContain("prompt")  //Pass

    //Get text present on popup
    let dialogtext = await dialog.message()  //Please enter your name:
    console.log("Text present on Popup: ", dialogtext)
    await expect(dialogtext).toBe("Please enter your name:")  //Pass Fail

    //Get default value present on popup
    let defaultvalueininputfield = await dialog.defaultValue()  //Harry Potter
    console.log("Default value on popup: ", defaultvalueininputfield)
    await expect(defaultvalueininputfield).toContain("Harry Potter")  //Pass
Fail

    //Enter Value in input field
    await page.waitForTimeout(4000)
    await dialog.dismiss();


    })

    await page.locator("//button[@id='promptBtn']").click()

    //Validate mesage on click Cancel Button
    let Messageafterclickoncancelbutton = await page.locator("#demo")
    await expect(Messageafterclickoncancelbutton).toHaveText("User cancelled
the prompt.")  //Pass //fail


});
```

Auth Popup:

```
import {test, expect} from '@playwright/test'

// Syntax: test("Test Case name", async({page})=>{});
// npx playwright test Pop_up6.spec.js --project=chromium --headed

//=========================================
// Handling of Auth popup
//=========================================

test("Handling of Auth popup", async({page})=>{

    await page.goto("https:admin:admin@//the-
internet.herokuapp.com/basic_auth")
    await expect(await
page.locator("//p[contains(text(),'Congratulations!')]")).toHaveText("Congratu
lations! You must have the proper credentials.")  //Pass Fail
    await page.waitForTimeout(5000)

});
```

Mouse Actions:

```
import {test, expect} from '@playwright/test'

//=================================================
// Mouse Actions : Left Click
//=================================================

// Syntax: test("Test Case name", async({page})=>{});
// npx playwright test Mouse_Actions.spec.js --project=chromium --headed

test("Mouse Action-Left Click ", async({page})=>{
    await page.goto("https://testautomationpractice.blogspot.com/")
    //Left Click on New Tab Button
    await page.locator("//button[text()='New Tab']").click()
    await page.waitForTimeout(3000)

});

//=================================================
// Mouse Actions : Right Click
//=================================================

test("Mouse Action-Right Click ", async({page})=>{
    await page.goto("https://swisnl.github.io/jQuery-contextMenu/demo.html")
    //Right Click on New Tab Button
    await page.locator("//span[text()='right click
me']").click({button:'right'});
    await page.waitForTimeout(3000)

});

//=================================================
// Mouse Actions : Double Click
//=================================================

test("Mouse Action-Double click", async({page})=>{
    await page.goto("https://testautomationpractice.blogspot.com/")
    //Double Click on Copy text Button
    await page.locator("//button[text()='Copy Text']").dblclick()
    await page.waitForTimeout(3000)

});
```

```
//================================================
// Mouse Actions : Mouse Hover
//================================================

test("Mouse Action-Mouse Hover", async({page})=>{
    await page.goto("https://testautomationpractice.blogspot.com/")
    //Mouse Hover on Point Me Button
    await page.locator("//button[text()='Point Me']").hover()
    await page.waitForTimeout(3000)
});

//================================================
// Mouse Actions : Drag and Drop
//================================================

test.only("Mouse Action-Drag and Drop", async({page})=>{
    await page.goto("https://demo.guru99.com/test/drag_drop.html")
    //Identify Source Element
    let Source = await page.locator("//a[text()=' 5000']")
    //Identify Destination Element
    //let Destination = await page.locator("(//li[@class='placeholder'])[4]")
                    //OR
    let Destination = await page.locator("//li[@class='placeholder']").nth(3)

    //Drag and drop Action
    await Source.dragTo(Destination);
    await page.waitForTimeout(3000)
});
```

KeyBoard Actions:

```
import {test, expect} from '@playwright/test'

// Syntax: test("Test Case name", async({page})=>{});
// npx playwright test KeyBoard_Action.spec.js --project=chromium --headed

//=============================================
//  KeyBoard Actions
//=============================================

test("keyBoard_Actions", async({page})=>{

    await page.goto("https://kite.zerodha.com/")

    //Identify UserId and Password
    let UserID = await page.locator("#userid");
    let Password = await page.locator("#password");

    await page.waitForTimeout(2000);

    //Enter UserID
    await UserID.fill("Rohit12345")

    //Select entered value by using Ctrl+A
    await UserID.press("Control+A")

    //Copy selected value by using Ctrl+C
    await UserID.press("Control+C")

    //Navigate to Password Field by Using Tab Key
    await UserID.press("Tab")

    await page.waitForTimeout(2000);

    //Paste Copied Value By using Ctrl+V
    await Password.press("Control+V")

    await page.waitForTimeout(2000);

    //Verify Entered values
    let UserIDText = await UserID.inputValue()
    let PasswordText = await Password.inputValue();

    console.log("Entered Value in UserID Field: ", UserIDText)
    console.log("Entered Value in Password Field: ", PasswordText)
    await page.waitForTimeout(3000);
});
```

Program-II:

```
import {test, expect} from '@playwright/test'

// Syntax: test("Test Case name", async({page})=>{});
// npx playwright test KeyBoard_Action2.spec.js --project=chromium --headed

//============================================
//  KeyBoard Actions
//============================================
test("KeyBoard Actions", async({page})=>{

    await page.goto("https://the-internet.herokuapp.com/key_presses")

    //await page.waitForTimeout(8000)

    //Identify Input box
    let Inputbox = await page.locator("#target")

    await page.waitForTimeout(4000)

    //Identify Result
    let Result = await page.locator("#result")

    //Click on inputbox
    await Inputbox.click()

    //Press "Shift" Key
    await Inputbox.press("Shift")

    await expect(Result).toContainText("SHIFT")  //Pass  //Fail

    await page.waitForTimeout(4000)

});
```

Handling of Iframe:

Explanation :

`page.goto()` opens webpage. `page.frameLocator()` identifies **frame1** using its XPath and returns a locator for interacting with elements inside that iframe. `frame1.locator()` identifies the input field using its XPath. The `fill("Virat")` method enters **Virat** into the textbox.

```
import {test, expect} from '@playwright/test'

// Syntax: test("Test Case name", async({page})=>{});
// npx playwright test Iframe1.spec.js --project=chromium --headed


//=============================================
// Example 1: Handling of single Iframe
//=============================================

test("Handling of Iframe-frame1", async({page})=>{

    await page.goto("https://ui.vision/demo/webtest/frames/")

    //Identify frame1
    let frame1 = await page.frameLocator("//frame[@src='frame_1.html']")

    //Identify Inputbox and Entered Value in it
    await frame1.locator("//input[@name='mytext1']").fill("Virat")
    await page.waitForTimeout(2000)

});

//=============================================
// Example 2:Handling of single Iframe
//=============================================

test.only("Handling of Iframe-frame4", async({page})=>{

    await page.goto("https://ui.vision/demo/webtest/frames/")

    //Identify frame4
    let frame4 = await page.frameLocator("//frame[@src='frame_4.html']")

    //Identify Inputbox and Entered Value in it
    await frame4.locator("//input[@name='mytext4']").fill("Rohit")
    await page.waitForTimeout(2000)

});
```

Program-II:

Explanation:

page.goto() navigates to the W3Schools JavaScript Tryit page. The **frameLocator("#iframeResult")** method identifies the iframe using its ID and stores it in the frame variable. Since the button exists inside the **iframe, frame.locator("//button[@type='button']")** locates the button using XPath. The .**click()** method clicks the button inside the iframe.

```
import {test, expect} from '@playwright/test'

// Syntax: test("Test Case name", async({page})=>{});
// npx playwright test Iframe2.spec.js --project=chromium --headed

//================================================
// Example 3: Handling of single Iframe
//================================================

test("Handling of Iframe", async({page})=>{

    await
page.goto("https://www.w3schools.com/js/tryit.asp?filename=tryjs_myfirst")

    //Identify Iframe
    let frame = await page.frameLocator("#iframeResult")

    //Identify Button and Click on Button
    await frame.locator("//button[@type='button']").click()

    await page.waitForTimeout(4000)

});
```

Handling of Nested Frames:

Program-I:

```
import {test, expect} from '@playwright/test'

// Syntax: test("Test Case name", async({page})=>{});
// npx playwright test Iframe3.spec.js --project=chromium --headed

//=============================================
// Example 3: Handling of Nested Iframe
//=============================================

test("Handling of nested frames", async({page})=>{

    await page.goto("https://ui.vision/demo/webtest/frames/")

    //Get Total no of Frames available on Webpage
    let allframes = await page.frames();  //[frame1, frame2,frame3......]
    console.log("Count of Frames: ", allframes.length) //7

    //Identify Frame3(Outer Frame)
    let frame3 = await page.locator("[src='frame_3.html']")

    //Convert locator into an Object using Contentframe() method
    let Outerframe = await frame3.contentFrame();  //{frame3 : await
page.locator("[src='frame_3.html']")}

    //Enter Value in input field
    await Outerframe.locator("//input[@name='mytext3']").fill("Rohit")

    await page.waitForTimeout(4000)

    //Identify Innerframe
    let Innergoogleformframe = await
Outerframe.locator("//iframe[@src='https://docs.google.com/forms/d/e/1FAIpQLSf
5WiH3jEQApYku0Rl_nreU6_YMuLKAH5ffHuASyykQSIBjmg/viewform?embedded=true']")


    //Convert locator into an Object using Contentframe() method
    let Innerframe = await Innergoogleformframe.contentFrame(); //
{Innergoogleformframe:
locator("//iframe[@src='https://docs.google.com/forms/d/e/1FAIpQLSf5WiH3jEQApY
ku0Rl_nreU6_YMuLKAH5ffHuASyykQSIBjmg/viewform?embedded=true']}

      await page.waitForTimeout(2000)

    //Check I am a human Radio button
    await Innerframe.locator("(//div[@class='vd3tt'])[2]").check();
```

```
    await page.waitForTimeout(4000)

    //Check General Web Automation Checkbox
    await Innerframe.locator("(//div[contains(@class,'uHMk6b
fsHoPb')])[3]").check();

    await page.waitForTimeout(5000)

});
```

Program-II:

```
import {test, expect} from '@playwright/test'

// Syntax: test("Test Case name", async({page})=>{});
// npx playwright test Iframe4.spec.js --project=chromium --headed

//=============================================
// Example 4: Handling of Nested Iframe
//=============================================

test("Handling of Nested frame", async({page})=>{

    await page.goto("https://demo.automationtesting.in/Frames.html")

    //Click "Iframe with in an Iframe" Button
    await page.locator("(//a[@class='analystic'])[2]").click()

    await page.waitForTimeout(2000)

    //Identify Outerframe
  let  Outerframeaddress = await page.locator('//div[@id="Multiple"]/iframe')

    //Contentframe() : Convert locator into Object
  let OuterFrame = await Outerframeaddress.contentFrame();


    //Identify Innerframe
  let Innerframeaddress = await OuterFrame.locator("//div[@class='iframe-
container']//iframe")

    //Contentframe() : Convert locator into Object
  let Innerframe = await Innerframeaddress.contentFrame(); //{}


    //Enter value into Inputput Field
    await Innerframe.locator("//div[@class='col-xs-6 col-xs-offset-
5']/input").fill("JavaScript");

    await page.waitForTimeout(5000)

});
```

Program-III:

```
import { test, expect } from "@playwright/test";
// Syntax: test("Test Case name", async({page})=>{});
// npx playwright test Iframe5.spec.js --project=chromium --headed

//=============================================
// Example 5: Handling of Nested Iframe
//=============================================

test("Handling of Nested frame", async({page})=>{

    await page.goto("https://demo.automationtesting.in/Frames.html")

    //Click "Iframe with in an Iframe" Button
    await page.locator("(//a[@class='analystic'])[2]").click();

    //Identify Outerframe
    let Outerframeaddress = await
page.locator("//div[@id='Multiple']/iframe");

    //Convert Locator into object by using contentFrame()
    let Outerframe =await
Outerframeaddress.contentFrame()  //{Outerframeaddress: await
page.locator("//div[@id='Multiple']/iframe");;}

    //Identify Innerframe
    let Innerframeaddress = await Outerframe.locator("//div[@class='iframe-
container']//iframe");

    //Convert Locator into object by using contentFrame()
    let Innerframe = await
Innerframeaddress.contentFrame()  //{Innerframeaddress
:Outerframe.locator("//div[@class='iframe-container']//iframe");}

    //Enter value into input box
    await Innerframe.locator("//div[@class='col-xs-6 col-xs-offset-
5']/input").fill("JavaScript")

    await page.waitForTimeout(4000)


})
```

Handling of WebTable:

Tag	Full Form / Meaning	Description
`<table>`	Table	Defines the **entire HTML table**. It is the main container for rows and cells.
`<tbody>`	Table Body	Defines the **body/content section** of the table. It contains one or more `<tr>` rows.
`<tr>`	Table Row	Defines a **row** in the table. A row contains `<th>` or `<td>` elements.
`<th>`	Table Header	Defines a **header cell**. It usually contains column headings such as Name, Age, Salary, etc.
`<td>`	Table Data	Defines a **data cell** containing the actual information in a table row.

Task 1: Total Number of Rows

- First identify the table using the XPath `//table[@name='BookTable']`.
- Then identify all the rows using `//table[@name='BookTable']//tr`.
- The `<tr>` tag represents a table row.

Task 2: Total Number of Columns/Cells

- First identify the table using the XPath `//table[@name='BookTable']`.
- Then identify the first row using `//table[@name='BookTable']//tr[1]`.
- Finally, identify the header cells using `//table[@name='BookTable']//tr[1]/th`.
- The `<th>` tag represents a table header or column heading.

```
import { test, expect } from "@playwright/test";
// Syntax: test("Test Case name", async({page})=>{});
// npx playwright test WebTable1.spec.js --project=chromium --headed

//=============================================
// Example 1: Handling of WebTable
//=============================================

test("Handling of WebTable", async({page})=>{

    await page.goto("https://testautomationpractice.blogspot.com/")

    //Scroll upto Table
    await page.locator("//h2[text()='Static Web
Table']").scrollIntoViewIfNeeded()

    //Get Total no of Rows
    let tableRows = await page.locator("//table[@name='BookTable']//tr")  //7
Rows addresses
    let rowCount = await tableRows.count()  //7
    console.log("Count of Rows: ", rowCount) //7

    //Get Total no of Columns
    let tableColumns = await
page.locator("//table[@name='BookTable']//tr[1]/th")  //4 Coluumns addresses
    let columnCount = await tableColumns.count()  //4
    console.log("Total no of Columns: ",columnCount)//4

    await page.waitForTimeout(4000)
});
```

Program-III:

```
import { test, expect } from "@playwright/test";
// Syntax: test("Test Case name", async({page})=>{});
// npx playwright test WebTable2.spec.js --project=chromium --headed

//================================================
// Example 2: Print all Booknames of WebTable
//================================================

test("Handling of WebTable", async({page})=>{

    await page.goto("https://testautomationpractice.blogspot.com/")

    //Scroll upto Table
    await page.locator("//h2[text()='Static Web
Table']").scrollIntoViewIfNeeded()

    //Row Count
    let TableRows = await page.locator("//table[@name='BookTable']//tr")  //7
Rows Addresses
    let RowCount = await TableRows.count();  //7
    console.log("Row Count of WebTable: ", RowCount)  //7

        //i=2    2<=7              3
                //3<=7              4
                //4<=7              5
                //5<=7              6
                //6<=7              7
                //7<=7              8
                //8<=7
    for(let i=2;    i<= RowCount;     i++)
    {
        let Books = await
page.locator(`//table[@name='BookTable']//tr[${i}]/td[1]`)
        let Bookname =await Books.textContent();
        console.log("Names of Book: ", Bookname)
    }

    await page.waitForTimeout(4000)
```

```
    //================================================
    // Find Price of Learn JS
    //================================================

    for(let i=2;    i<=RowCount;     i++)
    {
        let Bookname= await
page.locator(`//table[@name='BookTable']//tr[${i}]/td[1]`) //Learn JS
        let book = await Bookname.textContent();

        if(book==="Learn JS")  //Learn JS===Learn JS  //true
        {
            let price = await
page.locator(`//table[@name='BookTable']//tr[${i}]/td[4]`)
            let Bookprice = await price.textContent();//300
            console.log("BookName: ",book) //Learn JS
            console.log("Price of Book: ",Bookprice)  //300
        }

}
});
```

Program-III:

```
import { test, expect } from "@playwright/test";
// Syntax: test("Test Case name", async({page})=>{});
// npx playwright test WebTable2.spec.js --project=chromium --headed

//=============================================
// Example 2: Print all Booknames of WebTable
//=============================================

test("Handling of WebTable", async({page})=>{

    await page.goto("https://testautomationpractice.blogspot.com/")

    //Scroll upto Table
    await page.locator("//h2[text()='Static Web
Table']").scrollIntoViewIfNeeded()

    //Row Count
    let TableRows = await page.locator("//table[@name='BookTable']//tr")  //7
Rows Addresses
    let RowCount = await TableRows.count();  //7
    console.log("Row Count of WebTable: ", RowCount)  //7

    for(let i=2;    i<= RowCount;     i++)
    {
        let Books = await
page.locator(`//table[@name='BookTable']//tr[${i}]/td[1]`)
        let Bookname =await Books.textContent();
        console.log("Names of Book: ", Bookname)
    }

    await page.waitForTimeout(4000)

    //=================================================
    // Find Price of Learn JS
    //=================================================

    for(let i=2;    i<=RowCount;     i++)
    {
        let Bookname= await
page.locator(`//table[@name='BookTable']//tr[${i}]/td[1]`) //Learn JS
        let book = await Bookname.textContent();

        if(book==="Learn JS")  //Learn JS===Learn JS  //true
        {
            let price = await
page.locator(`//table[@name='BookTable']//tr[${i}]/td[4]`)
            let Bookprice = await price.textContent();//300
```

```
            console.log("BookName: ",book) //Learn JS
            console.log("Price of Book: ",Bookprice)  //300
        }
}

 await page.waitForTimeout(4000)

//=======================================
// Books Written By Mukesh
//=======================================

     //i=2       //2<=7          3
                 //3<=7          4
                 //4<=7          5
                 //5<=7          6
                 //6<=7          7
                 //7<=7          8
                 //8<=7
for(let i=2;      i<=RowCount;     i++)
{
                                                                    //7
   let Authorname = await
page.locator(`//table[@name='BookTable']//tr[${i}]/td[2]`)
                                        //
//table[@name='BookTable']//tr[7]/td[2]
   let Author =await Authorname.textContent();  //Amit

      //Amit===Mukesh
    if(Author==="Mukesh")  //False
    {
                                                                  //5
      let Book = await
page.locator(`//table[@name='BookTable']//tr[${i}]/td[1]`)
                                    ////table[@name='BookTable']//tr[5]/td[1]
      let Bookname= await Book.textContent();  //Master In Selenium

      //console.log("Author Name: ", Author) //Mukesh  //Mukesh
      //console.log("Book Name: ", Bookname)  //Learn Java  //Master In
Selenium
      console.log(`Book Name: ${Bookname} and Author Name: ${Author}`)

    }


}
```

File Upload:

```
import {test, expect} from '@playwright/test'
import path from "node:path"

// Syntax: test("Test Case name", async({page})=>{});
// npx playwright test Single_File_upload1.spec.js --project=chromium --headed

test("Single File Upload", async({page})=>{

    await page.goto("https://the-internet.herokuapp.com/upload")

    //Create file path
    // __dirname : D\ATT_6_June_Morning_Playwright\tests
    //../ : D\ATT_6_June_Morning_Playwright

   let filepath = path.join(__dirname, "../TestData/AI Github CoPilot.pdf") //
D\ATT_6_June_Morning_Playwright\TestData\AI Github CoPilot.pdf

    await page.locator("#file-upload").setInputFiles(filepath)

    //Click Upload Button
    await page.locator("#file-submit").click()

});
```