Selenium Python Developer Notes

1 Fundamentals of Automation & Selenium

1.1 What is Automation Testing?

Automation testing matlab apne software ya website ko **automatically test** karwana—matlab human ko baar baar manually check nahi karna padta. Ek script banate ho, woh khud sab test steps follow karti hai.

Why use karein?

- Manual testing **boring ho jata hai**, time lagta hai.
- Repeat karne mein ✗ mistake ho sakti hai.
- ✓ Fast, consistent, accurate results chahiye toh automation best hai.

Kab use karein?

- Jab project bada ho ya bhut baar same cheez check karni ho—jaise regression, smoke test, etc.

Agar na karein toh?

- Test karte karte bore ho jaoge, choti choti galtiyan ho sakti hain.
- Release cycle slow ho jayegi.

1.2 Manual vs Automated Testing

Manual Testing: Insaan khud manually har ek step karta hai, report karta hai. Small projects, ya one-time testing mein useful.

Automated Testing: Script sab kuch khud karti hai, repeat kar sakte ho jitni baar chaaho—jaise ek robot har baar same kaam automatically kare.

1.3 SDLC, STLC, Automation Life Cycle

- **SDLC (Software Development Life Cycle):** Software kaise banta hai—plan, code, test, deploy, maintain.

- **STLC (Software Testing Life Cycle):** Sirf testing wali steps—plan, create tests, execute, bugs dhundo, fix karao.
- **Automation Life Cycle:** Kaunse test automate karne hain decide karo, tool select karo, script banao, test chalhao, maintain karo.

1.4 Automation Testing ka ROI (Return on Investment)

- Starting mein time lagta hai scripts banane mein, lekin baad mein ✓**bahut time bacha sakte ho**.
- Jaldi release ho sakte ho.
- Manual galtiyan kam ho jati hain.
- Team ka kaam aasan ho jata hai.

1.5 Types of Testing: Smoke, Regression, Integration, etc.

Smoke Testing: Aisa test jisse pata chale major feature kam se kam work kar raha hai ya nahi.

Regression Testing: Code mein change ke baad kuch aur toota toh nahi check karna.

Integration Testing: Alag alag modules milke kaam kar rahe hain ki nahi.

Aur bhi: functional, load, UAT, sanity testing, etc.

1.6 Selenium Overview

- **Selenium Suite:** IDE, WebDriver, Grid (RC ab nahi chalta)
- **Selenium IDE:** Simple record & playback tool hai, jaise macro, basic automation ke liye.
- **WebDriver:** Yeh main tool hai jo programming languages (Python, Java, etc.) se browser ko control karta hai.
- **Grid:** Multiple machines/browsers pe parallel testing ke liye.

1.7 Selenium WebDriver kaise kaam karta hai? (Architecture)

Aapka Python script → Selenium library → Driver (jaise ChromeDriver) → Browser control hota hai (jaise Chrome ya Firefox).

Har browser ka apna driver hota hai.

1.8 Supported Browsers, Languages, Platforms

- **Browsers:** Chrome, Firefox, Edge, Safari, Opera, etc.
- **Languages:** Python, Java, C#, Ruby, JavaScript (Node.js)
- **Platforms:** Windows, Mac, Linux sab pe chalta hai.

2 Environment Setup

2.1 Python & Tooling

Python Install karna hai: Official python.org se download karo, install karo.

Virtual Environment (virtualenv): Har project alag environment mein rakho toh dependency issues nahi aate.

Virtual Environment Creation Example

```
python -m venv venv
```

Activate:

Activate Virtual Environment

```
# Windows:
.\venv\Scripts\activate

# Mac/Linux:
source venv/bin/activate
```

IDE use karo: PyCharm ya VSCode (Python plugin ke saath)—isme code likhna, run karna easy ho jata hai.

Project structure: Aise files/folders banao:

- `src/` (source code)
- `tests/` (test scripts)
- `resources/` (data/files)
- `logs/` (output ya failures track karne ke liye)

2.2 Selenium Install Karna

Install Selenium

```
pip install selenium
```

Browser driver (jaise ChromeDriver) chahiye:

- Chrome: ChromeDriver
- Firefox: GeckoDriver
- Edge: EdgeDriver

Download karke PATH mein dal do, ya easy way:

webdriver_manager: Yeh Python package driver ko automatically download/use karta hai.

webdriver_manager usage example

```
from selenium import webdriver
from webdriver_manager.chrome import ChromeDriverManager

driver = webdriver.Chrome(ChromeDriverManager().install())
```

Environment Variables & PATH setup: Driver ka path system ke environment variable mein hona chahiye, taki Selenium use kar sake.

webdriver_manager use karoge toh yeh step mostly skip ho jayega.

Aap mujhe next specific topic batao ya poochho, ya example code ki jarurat ho toh batao main aapke level pe explain kar dunga!

---

3 Selenium WebDriver Core (Complete Explanation)

3.1 1. Browser Control

1. Launching Browsers (Chrome, Firefox, Edge)

Why Important?

- Jab tak browser launch nahi hoga, tab tak koi bhi web testing impossible hai.
- Selenium ko browser control karna hota hai taaki woh apne scripts me jo automate karna hai woh kar sake.

When To Use?

- Har baar test script run karte waqt browser ko launch karna hota hai.
- Different browsers pe cross-browser compatibility check karna ho.

Agar Na Use Kiya Toh?

- Browser open nahi hoga, test kisi tarah se run nahi hoga.
- Manual testing ke bina automation impossible.

Code (Chrome Example):

Chrome Launch Example

```
from selenium import webdriver  # Selenium ke webdriver module ko import karte hain
from webdriver_manager.chrome import ChromeDriverManager  # Chrome driver manager import karte hain

driver = webdriver.Chrome(ChromeDriverManager().install())  # Chrome browser launch karte hain aur driver automatically set karte hain
```

Line-by-line Explanation:

- `from selenium import webdriver`: Selenium ke webdriver module ko import kiya, yeh module browsers ko automate karne ke liye APIs deta hai.
- `from webdriver_manager.chrome import ChromeDriverManager`: Webdriver manager ek utility hai jo ChromeDriver ko automatically download aur update karta hai, taaki manually driver manage na karna pade.
- `driver = webdriver.Chrome(ChromeDriverManager().install())`: Yeh line Chrome browser ko launch karti hai. `ChromeDriverManager().install()` se driver ka path milta hai jo browser ko control karta hai.

Note: Firefox ya Edge ko launch karne ke liye bhi yehi concept hai, bas webdriver_manager ke corresponding module use karoge.

Firefox Example

```
from webdriver_manager.firefox import GeckoDriverManager
driver = webdriver.Firefox(executable_path=GeckoDriverManager().install())
```

2. Navigating to URLs

Why Important?

- Browser ko kisi specific website pe le jana hota hai jisko test karna hai.

When To Use?

- Har test case ke starting me ya kisi bhi step me URL open karna pade to.

Agar Na Use Kiya Toh?

- Test script kisi page pe nahi jayega, automation steps fail honge.

Code:

URL Navigation

```
driver.get("https://www.google.com")  # Jo browser ab open hai usko google.com pe le jate hain
```

Explanation: .get() method browser ke URL bar me link dalne ke barabar kaam karta hai. Jo URL diya hai, browser us page ko load kar leta hai.

3. Browser Operations: Back, Forward, Refresh, Close, Quit

Why Important?

- Browser navigation control karna, test script ko automate browser ke button jaisa operate karwana.

When To Use?

- Browser me peeche ya aage jana ho.
- Page ko refresh karna ho (reload karna ho).
- Test ke end me browser band karna ho, ya tab close karna ho.

Agar Na Use Kiya Toh?

- Navigation nahi hoga, contaminations ho sakte hain.
- Browser khula rahega, resource waste hoga.

Codes with Explanations:

Browser Operations

```
driver.back()      # Browser ko ek page peeche le jana hai, jaisa
    back button dabate hain
driver.forward()   # Browser ko ek page aage le jana hai, jaisa
    forward button dabate hain
driver.refresh()   # Current page ko dubara load karna hai (reload)
driver.close()     # Jo active tab ya window hai use band karta hai
driver.quit()      # Pura browser aur jitne bhi tabs khule hain
    sabko band karta hai
```

4. Incognito & Headless Mode

Why Important?

- **Incognito Mode:** Testing cookies/cache/reset state pe impact karne ke liye blueprint hota hai.
- **Headless Mode:** Background me browser chalana bina GUI ke, useful for CI/CD pipeline jahan UI show karna zaruri nahi.

When To Use?

- Jab clean session chahiye without cache/cookies.
- Jab automation servers pe run karna ho jaha GUI available nahi hota (headless).

Agar Na Use Kiya Toh?

- Cookies/cached data interfere kar sakta hai tests me.
- Automation servers pe GUI na hone ke wajah se script fail ho sakta hai.

Incognito Mode Example:

Incognito Mode

```
from selenium.webdriver.chrome.options import Options

options = Options()                    # Browser options object
    create karte hain
options.add_argument("--incognito")   # Incognito mode enable karne
    ke liye argument dete hain

driver = webdriver.Chrome(options=options,
    executable_path=ChromeDriverManager().install())
```

Options() object banaya, usme incognito argument add kiya jo browser ko private mode me kholta hai.

Headless Mode Example:

Headless Mode

```
options = Options()
options.add_argument("--headless")   # Browser GUI bina khole
    background me run karega

driver = webdriver.Chrome(options=options,
    executable_path=ChromeDriverManager().install())
```

–headless flag se browser window visible nahi hoti, background me test run hota hai.

5. Multiple Tabs/Windows pe Kaam Karna

Why Important?

- Kabhi browser me ek se zyada tabs ya windows open karne padte hain (social login pop-ups, ads, payment windows).
- Selenium ko pata hona chahiye kaunsa window/tab active karna hai taaki waha commands bheje.

When To Use?

- Jab naye tabs/windows open hote hain during user flows.

Agar Na Use Kiya Toh?

- Selenium commands galat window/tab pe chale jayenge, tests fail ho jayenge.

Code Example with Explanation:

Multiple Tabs/Windows

```
driver.execute_script("window.open('https://www.youtube.com',
    '_blank')")  # Naya blank tab open kar ke usme YouTube URL load
    karo

windows = driver.window_handles      # Browser ke sab khule
    window/tab ke handles le aao (list me)
driver.switch_to.window(windows[1])  # Window handles list me se
    second tab pe switch ho jao, taki ab us tab par control mile
```

execute_script se browser me JavaScript execute kar rahe hain — naya tab khola gaya. window_handles me tabs ka list, switch_to.window se focus change.

WebDriver APIs Advanced Concepts

1. WebDriver Object, Options, Capabilities

Why Important?

- WebDriver object browser automation ka main interface hai.
- Options aur Capabilities set karne se hum browser behavior customize kar sakte hain (jaise SSL certificates accept karwana).

When To Use?

- Jab browser ko specific mode me chalana ho ya kisi feature ko enable/disable karna ho.

Agar Na Use Kiya Toh?

- Browser ka default behavior use hoga, kuch cases me automation fail ho sakti hai, jaise SSL errors.

Code Example:

Capabilities Example

```
from selenium.webdriver.common.desired_capabilities import
    DesiredCapabilities
from selenium import webdriver

cap = DesiredCapabilities().CHROME.copy()  # Chrome capabilities
    copy karte hain
cap['acceptSslCerts'] = True               # SSL certificate errors
    ko accept karne ka flag set karte hain

driver = webdriver.Chrome(desired_capabilities=cap,
    executable_path=ChromeDriverManager().install())
```

DesiredCapabilities se browser ke advanced options set hote hain. acceptSslCerts se insecure SSL certificates ke warnings ko bypass kar sakte hain.

2. JavaScript Execution with `execute_script()`

Why Important?

- Standard WebDriver methods se kaam nahi hota, ya kisi element ko special handle karna ho (scroll, alerts, page ke variables, etc).
- JS execute karne ka option flexibility deta hai.

When To Use?

- Scroll karna, hidden element dikhaana, alert box dikhana, JS ke variables ko access karna, custom framework ke ops.

Agar Na Use Kiya Toh?

- Limited browser access hota hai, kuch advanced ops possible nahi hoti.

Code Example:

JavaScript Execution

```
1 driver.execute_script("alert('Hello from Selenium!')")  # Browser
    me ek alert pop-up show karo
```

execute_script() JS code ko direct browser me chalata hai. Yahan ek alert box open hota hai.

Summary Table

Feature	Why Important	When To Use	Core Usage Example
Browser Launch	Browser control ke liye	Har automation run me	`webdriver.Chrome()`
URL Navigation	Web page open karne ke liye	Test case start ya step me	`driver.get(url)`
Browser Nav Ops (Back/Forward)	User navigation replicate karne ke liye	Navigation test scenario	`driver.back()`, `driver.forward()`
Browser Close/Quit	Resource cleanup	Test end me	`driver.quit()`
Incognito Mode	Clean session for tests	Cache, cookies problems ke liye	`Options().add_argument("\-incc`
Headless Mode	GUI bina test run	CI/CD pipelines, servers	`Options().add_argument("--heac`
Multiple Tabs Handling	Multi-tab automation	Naya tab/window open hone par	`driver.switch_to.window()`
WebDriver Options/Capabilities	Browser behavior customize	Advanced scenarios	`DesiredCapabilities()`
JavaScript Execution	Advanced browser manipulation	Custom scroll, alerts, JS vars	`driver.execute_script()`

Agar tumhe chahiye toh chhote-chhote projects ya step-by-step exercises bhi mil sakte hain. Bas topic batao! Ready ho? Shuru karte hain?

==========

4 Locators & Element Identification in Selenium (Complete Guide)

Why Locator Identification Important Hai?

- Selenium ka main kaam hota hai browser ke elements (buttons, input boxes, links, etc.) ko find karke unpe actions karna.
- Agar elements ko sahi tarah locate nahi karoge, toh test fail ho jayega kyunki Selenium ko pata nahi chalega kis element pe kaam karna hai.
- Locator selection test ki speed, reliability aur maintainability ko directly affect karta hai.

When Use Karein?

- Jab bhi interaction karna ho page ke kisi element ke saath (click, type, read text).
- Har ek test step ke liye element locate karna zaroori hota hai.
- Dynamic web pages me locator strategy smart honi chahiye.

Agar Locator sahi na chuna ho (If Not Used Correctly)?

- Tests break honge frequently.
- Element not found exceptions ayenge.
- Flaky tests bante hain, jisme kabhi pass kabhi fail ho.
- Maintenance bohut difficult ho jayega.

Basic Locators in Selenium

1. ID Locator

What?

- Element ka HTML attribute `id` ka use karta hai. Har element ka id usually unique hota hai.

Why?

- Fastest, sabse reliable locator.

When?

- Jab element ka id available aur static ho.

Code Example:

ID Locator Example

```
element = driver.find_element("id", "username")
```

Line Explanation:

- `find_element()` method use karte hain.
- `"id"` specify karta hai locator strategy ko.
- `"username"` woh unique id value hai jise match karna hai.
- Selenium is element ko find karke `element` variable mein store karta hai.

2. Name Locator

What?

- Element ka `name` attribute use karta hai.

Why?

- ID ke bad option, particularly form elements me common hai.

When?

- Jab id nahi ho aur name unique ho.

Code Example:

Name Locator Example

```
element = driver.find_element("name", "password")
```

Explanation: Same way jaise id ke liye, par yahan locator strategy `"name"` hai aur `"password"` attribute value.

3. Class Name Locator

What?

- Element ka `class` attribute use karta hai.

Why?

- Tab use karo jab class unique ya specific ho.

When?

- Jab id/name nahi ho.

Caution:

- Class name usually shared hota hai, toh first match milega.

Code Example:

Class Name Locator Example

```
element = driver.find_element("class name", "btn-primary")
```

4. Tag Name Locator

What?

- HTML tag jaise `input`, `button`, `div` se locate karta hai.

Why?

- Jab sirf tag basis par search karna ho, ya parent element ke andar multiple similar tags ho.

When?

- Complex scenarios me jab aur options nahi mil rahe.

Code Example:

Tag Name Locator Example

```
element = driver.find_element("tag name", "button")
```

5. Link Text Locator

What?

- <a> tags me jo visible link text hota hai, uske basis par element locate karta hai.

Why?

- Simple aur readable locator links ke liye.

When?

- Jab link text fix ho aur woh accurately identifiable ho.

Code Example:

Link Text Locator Example

```
element = driver.find_element("link text", "Click here")
```

6. Partial Link Text Locator

What?

- Link text ka exact nahi, kuch part ka match karta hai.

Why?

- Jab pura text yaad nahi ho ya change ho sakta ho.

When?

- Flexible partial match ke liye.

Code:

Partial Link Text Locator Example

```
element = driver.find_element("partial link text", "Click")
```

7. CSS Selector

What?

- CSS selector syntax ka use karke elements locate kar sakte hain.

Why?

- Powerful, flexible and fast.

When?

- Jab id/name/class nahi mile ya complex hierarchy chahiye.

Examples:

CSS Selector Examples

```
element1 = driver.find_element("css selector", "#email")
    # id ke liye #
element2 = driver.find_element("css selector", ".input-field")
    # class ke liye .
element3 = driver.find_element("css selector",
    "input[type='text'][name='email']")
```

Explanation:

- `#email`: id `email` wala element.
- `.input-field`: class `input-field` wala element.
- Complex selector jisme input tag ke andar `type='text'` aur `name='email'` dono condition apply hain.

8. XPath

What?

- XML path syntax se element location karte hain.
- Sabse flexible locator hai.

Why?

- Sab complex aur dynamic structure me useful.

When?

- Jab koi basic locator kaam na kare, ya nested element chahiye.

Example:

XPath Example

```
element = driver.find_element("xpath", "//button[@id='submitBtn']")
```

Explanation:

- `//button`: page ke jitne bhi `button` tags hain.
- `[@id='submitBtn']`: jinka `id='submitBtn'` attribute hai.

Advanced Locators

1. Dynamic XPath: contains(), starts-with(), axes

Why?

- Element attribute dynamic ya change hota rahta hai, toh simple exact match fail ho jata hai.

When?

- Jab element ke id/class dynamic strings generate kar raha ho.

Examples:

contains():

contains() Example

```
element = driver.find_element("xpath", "//*[contains(@id,
    'user_')]")
```

Iska matlab: Kisi bhi tag ka wo element jiska `id` me 'user_' substring ho.

starts-with():

starts-with() Example

```
element = driver.find_element("xpath", "//*[starts-with(@name,
    'pass')]")
```

Iska matlab: Jo elements ka `name` attribute 'pass' se start hota ho.

Axes (Sibling, Parent, Child):

XPath Axes Example

```
element = driver.find_element("xpath",
    "//label[text()='Username']/following-sibling::input")
```

Explanation: XPath me `label` jiska text 'Username' hai, uske just baad jo input sibling element ho usko select karna.

2. Custom Attributes: data-, aria-

Why?

- Web accessibility aur testing ke liye custom attributes use karte hain jise identify karna easy hota hai.

Example:

Custom Attributes CSS Selector

```
element = driver.find_element("css selector",
    "div[data-test='loginButton']")
```

3. SelectorHub Plugin

What?

- Browser extension jo automatic XPath/CSS selector generate karta hai.

Why Important?

- Beginner-friendly tool locator banana aur test karna asaan ho jata hai.

When Use?

- Jab complex locator chahiye ya time-save karna hai.

4. Accessibility Locators (ARIA roles)

What?

- `aria-*` attributes se elements ko locate kar sakte hain.

Example XPath:

ARIA Accessibility Locator Example

```
element = driver.find_element("xpath", "//*[@role='button' and
    @aria-label='Submit']")
```

When?

- Jab app me accessibility features implemented hain, inhe use karna reliable hota hai.

Summary Table

Locator Type	Use Case / When to Use	Why Important	Example
ID	Unique id attribute	Sabse fast, reliable	`find_element("id", "username")`
Name	Name attribute suitable mostly forms	Good fallback	`find_element("name", "password")`
Class Name	Class attribute, often multiple matches	Use when unique class	`find_element("class name", "btn-primary")`
Tag Name	Tag type locator	Simple element list	`find_element("tag name", "button")`
Link Text	Exact link text	Link identification	`find_element("link text", "Click here")`
Partial Link Text	Partial text match for link	Flexible link matching	`find_element("partial link text", "Click")`
CSS Selector	CSS selectors hierarchy or attribute	Powerful & faster	`find_element("css selector", "#email")`
XPath	Complex queries & relationships	Most flexible locator	`find_element("xpath", "//button[id='submitBtn']"`
Dynamic XPath	Dynamic attributes (contains, starts-with)	Handle dynamic UX	`/*[contains(@id, 'user_')]`
Custom Attributes	data-, aria- attributes	Accessibility & testing	`div[data-test='loginButtor`
Accessibility Locators	ARIA labels & roles	Inclusive design, reliable	`/*[role='button' and aria-label='Submit']`

==========

5 Web Element Interactions in Selenium Python — Complete Guide

What are Web Element Interactions?

- Jab hum Selenium se browser me kisi element ko locate kar lete hain (jaise textbox, button, checkbox, dropdown etc.), tab hume un elements ke saath interact karna hota hai.
- Interaction matlab user ki tarah us element pe click karna, usme text type karna, kisi dropdown me option select karna, checkbox check/uncheck karna, ya us element se koi

information lena.

- Selenium ka purpose hi hai browser elements ke saath automatic interaction karna, bina human ke manual clicks/inputs ke.

Why Use Web Element Interactions?

- Manual testing me har ek interaction manually karna padta hai, ye bohot slow aur repetitive hota hai.
- Automation se ye sab interactions programmatically perform hote hain jo fast, accurate aur repeatable hote hain.
- Testing ke liye ye interactions required hain taaki app ka behavior verify kar sako user ke jaise.

When Use Web Element Interactions?

- Jab bhi test case ka koi step hota hai jisme input dena ho ya button dabana ho.
- Form filling, login, navigation, filtering, submitting, selecting, aur reading outputs sab ke liye use hota hai.
- Har functional flow automation me ye basic building blocks hain.

Agar Na Use Karein (If Not Used)?

- Test scripts incomplete rahenge — koi input nahi hoga, buttons click nahi honge.
- Automation ka purpose hi khatam.
- Manual intervention ke bina test cases chalana mushkil ho jayega.
- Sabhi core UI actions automate karna possible nahi hoga.

6 1. send_keys(), click(), clear() Methods

Ye teen sabse common methods hain Selenium me element ke saath interaction ke liye.

a) send_keys()

What?

- Input field (textbox, textarea) me text type karne ke liye.

Why?

- User input ko replicate karta hai automation ke through.

When?

- Jab form fill kar rahe ho, login kar rahe ho, koi search box me typing kar rahe ho.

Example Code + Explanation:

send_keys() Example

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from webdriver_manager.chrome import ChromeDriverManager

# Browser open karna
driver = webdriver.Chrome(ChromeDriverManager().install())

# Website open karna jisme interact karenge
driver.get("https://example.com/login")

# Username input field locate karna (id='username' assume kar rahe
    hain)
username_input = driver.find_element(By.ID, "username")

# Us input box me "myusername" type karna
username_input.send_keys("myusername")
```

- `driver.find_element(By.ID, "username")`: Browser me element locate kar raha hai jiska id `username` hai.
- `.send_keys("myusername")`: Us element me text "myusername" type kar diya.

b) clear()

What?

- Input field me already jo value/hai use clear (delete) karta hai.

Why?

- Kabhi-kabhi fields pre-filled hote hain, toh purana text hata ke naya type karna hota hai.

Example:

clear() Example

```
username_input.clear()
```

- Ye line `username_input` field se pehle wali value hata degi taaki nayi value sahi jaye.

c) click()

What?

- Button, checkbox, link, ya kisi bhi clickable element pe click karta hai.

Why?

- User clicks ko automate karta hai.

When?

- Jab form submit karna ho, button click karna ho, link open karna ho, checkbox select/unselect karna ho.

Example:

click() Example

```
password_input = driver.find_element(By.ID, "password")
password_input.send_keys("mypassword")

login_button = driver.find_element(By.ID, "loginBtn")
login_button.click()
```

- Pehle password field locate karke password type kiya.
- Phir login button locate karke uspe click kiya.

7 2. Get Text / Get Attributes

Kabhie element se uska visible text ya koi attribute chahiye hota hai.

a) Get Visible Text

What?

- Element par jo dikh raha visible text hai usko lena.

Why?

- Verify karne ke liye ki sahi message ya label dikh raha hai.

When?

- Form submission ke baad success message check karna ho.
- Page header, error messages, labels ko verify karna ho.

Code Example:

Get Visible Text Example

```
heading = driver.find_element(By.TAG_NAME, "h1")
print(heading.text)
```

- `heading.text`: us element ke andar ka jo visible text hai, wo fetch karta hai.

b) Get Attribute Value

What?

- Element ke kisi attribute ki value lena, jaise href, value, title.

Why?

- Link URL verify karna ho, placeholder text check karna ho, koi custom attribute validate karna ho.

When?

- Verification or assertion ke liye.

Example:

Get Attribute Example

```
link = driver.find_element(By.LINK_TEXT, "Click Here")
url = link.get_attribute("href")
print(url)
```

- Yahaan `href` attribute ka value link ke liye liya gaya.

8 3. Handling Checkboxes

What?

- Checkboxes select/unselect karna.

Why?

- Forms me options select karna zaruri hota hai.

When?

- Jab forms fill kar rahe ho ya kisi filter option select karna ho.

Code + Explanation:

Checkbox Handling Example

```
checkbox = driver.find_element(By.ID, "subscribe")

# Checkbox select hai ya nahi check karna
if not checkbox.is_selected():
    checkbox.click()  # Selected nahi hai toh click karke select
        karo

# Agar unselect karna ho
if checkbox.is_selected():
    checkbox.click()  # Selected hai toh click karke unselect karo
```

- `.is_selected()` check karta hai ki checkbox selected hai ya nahi.
- `click()` se toggle hota hai.

9 4. Handling Radio Buttons

What?

- Radio buttons me se ek choose karna.

Why?

- User ke choices capture karne ke liye.

When?

- Jab gender, payment method, ya koi exclusive option select karna ho.

Code:

Radio Button Handling Example

```
radio_male = driver.find_element(By.ID, "gender_male")

radio_male.click()                # Select karna
print(radio_male.is_selected()) # Check karna kya select hua hai
```

- Radio buttons me multi-select allowed nahi hota.

10 5. Dropdowns via Select Class

What?

- HTML dropdown element me option select karna.

Why?

- User ko list me se ek option dena hota hai.

When?

- Country select karo, product category choose karo, etc.

Code Example + Explanation:

Dropdown Select Example

```
from selenium.webdriver.support.ui import Select

# Dropdown element locate karo (id='country' assume kar rahe hain)
dropdown = Select(driver.find_element(By.ID, "country"))

# Option select karne ke 3 tarike
dropdown.select_by_visible_text("India")    # Jo text dikhta hai usse select karo
dropdown.select_by_value("IN")              # Value attribute ke basis par select karo
dropdown.select_by_index(5)                 # Index se (0-based) select karo
```

- `Select` class ko import karna hota hai.
- Alag-alag tarike se select kar sakte hain depending on use case.

11 6. Working with Sliders, Date Pickers, Drag and Drop

a) Sliders

What?

- Slider control move karne ke liye.

Why?

- Volume, brightness, price filter jaise sliders control karte hain.

When?

- Test case me slider ko adjust karna ho.

Code + Explanation:

Slider Control Example

```
from selenium.webdriver import ActionChains

slider = driver.find_element(By.ID, "volumeSlider")
action = ActionChains(driver)

# Slider ko 50 pixels right drag karo
action.click_and_hold(slider).move_by_offset(50, 0).release().perform()
```

- `ActionChains` mouse/keyboard advanced actions ke liye hota hai.
- `.click_and_hold()`: mouse button down karke
- `.move_by_offset(50,0)`: 50 pixels right move karna

- `.release()`: button chhodna
- `.perform()`: action execute karwana

b) Date Pickers

What?

- Date input field me date dalna.

Why?

- Forms me date of birth, appointment date select karna hota hai.

When?

- Jab input field simple type ka ho — datepicker nahi complex pop-up.

Code:

Date Picker Example

```
date_field = driver.find_element(By.ID, "dob")
date_field.send_keys("2025-07-26")
```

- Agar datepicker complex ho tab JavaScript execute karna pad sakta hai.

c) Drag and Drop

What?

- Ek element ko drag kar ke doosre element par drop karna.

Why?

- UI me reorder lists, file upload fields, ya custom drag-drop controls me.

Code:

Drag and Drop Example

```
source = driver.find_element(By.ID, "drag")
target = driver.find_element(By.ID, "drop")

action = ActionChains(driver)
action.drag_and_drop(source, target).perform()
```

- `drag_and_drop()` ek simple method hai drag-drop ke liye.

12 7. File Uploads & Downloads Automation

a) File Upload

What?

- File select karna (upload karna) browser me.

Why?

- Form inputs me resume upload, profile photo etc.

When?

- Jab HTML input type=file ho.

Code:

File Upload Example

```
upload = driver.find_element(By.ID, "file-upload")
upload.send_keys("C:\\Users\\User\\Desktop\\testfile.txt")
```

- Yahan `.send_keys()` ko file ka **absolute** path dena hota hai.
- Selenium native OS file dialogs ko handle nahi karta, isliye input element ki help se hi file upload hota hai.

b) File Download Automation

- Selenium khud direct control nahi karta download pop-ups pe kyunki browser control bahar hota hai.
- Par browser preferences set karke downloads folder change kar sakte hain.
- Advance automation tools/plugins ke saath download automation hoti hai.

Summary Table (Quick Reference)

Feature	Method/Approach	Use Case
Text input	`send_keys()`	Type text in input fields
Button/link click	`click()`	Click buttons, links
Clear input	`clear()`	Purana text remove karna
Get visible text	`.text`	Text retrieve karna
Get attribute	`.get_attribute("attr")`	href, value, title, custom attrs
Checkbox select/unselect	`click()` + `.is_selected()`	Checkbox control
Radio button select	`click()` + `.is_selected()`	Radio buttons
Dropdown select	`Select` class + `select_by_*` methods	Select option from dropdown
Slider move	`ActionChains.move_by_offset()`	Slider drag
Drag & Drop	`ActionChains.drag_and_drop()`	Drag element to target
File Upload	`.send_keys(file_path)` on file input	Upload file automation

13 Handling Alerts, Frames, Windows

Why important hai?

- Web applications mein bahut baar JavaScript alerts, confirm boxes, ya prompt dialogs aate hain user interaction ke liye, jinke bina test automation ruk sakta hai.
- Websites mein frames or iframes hote hain jinke andar page ke nested content hota hai — bina unhe switch kiye, elements ko access nahi kar sakte automation se.
- Multiple browser tabs/windows hoti hain—automation script ko pata hona chahiye kaunsi window/tab pe kaam karna hai, warna errors aayenge.

When to use?

- Jab alert/dialog aaye usko accept ya dismiss karna ho (jaise confirm dialog).
- Jab iframe ke andar ke elements ko interact karna ho.
- Jab aapka test case multiple windows ya tabs ke beech switch karta ho (pop-ups, social login windows, payment gateways).

Agar na karein toh?

- Alert aane pe agar handle na karein toh script popup ke wajah se ruk jaata hai (Exception error).
- Frame ke andar ke elements ko bina switch kiye locate nahi kar paoge.
- Multiple windows ke beech switch na karoge toh galat window pe commands bhejoge, test fail ho sakta hai.

JavaScript Alerts, Confirm, Prompt Popups

JavaScript alerts are simple popups with a message and OK button, confirm boxes have OK and Cancel, prompts ask for user input.

Code example + explanation:

JavaScript Alert Handling Example

```python
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from webdriver_manager.chrome import ChromeDriverManager

driver = webdriver.Chrome(ChromeDriverManager().install())
driver.get("https://example.com/alert-demo")

# Assume button pe click karke alert open hota hai
alert_button = driver.find_element(By.ID, "alertButton")
alert_button.click()

# Switch to alert box:
alert = driver.switch_to.alert

# Alert ka text get karo
print(alert.text)  # Alert me kya likha hai woh dikhayega

# Alert accept karo (OK button press)
alert.accept()

# Agar confirm box ho toh reject karna ho (cancel), use karo:
# alert.dismiss()

# Agar prompt box hai jisme input dena hai:
# alert.send_keys("Hello Alert")
# alert.accept()
```

- `driver.switch_to.alert` se Selenium ko alert box ka control milta hai.
- `alert.text` alert mein jo message hai wo fetch karta hai.
- `alert.accept()` alert ke OK button ko press karta hai.
- `alert.dismiss()` Cancel button click karta hai.
- Prompt box me input ke liye `send_keys()` bhi use karte hain.

Handling Frames & IFrames

Frames simply ek webpage ke andar embedded documents hote hain. Selenium ke liye, agar aap element frame ke andar ho, toh us frame pe switch karna zaroori hota hai.

Why important?

- Agar frame me content hai aur frame switch nahi karoge, Selenium `NoSuchElementException` dega; elements nahi milenge.

When to use?

- Jab aapko frames/iframes wale webpage pe automate karna ho.

Basic methods to switch:

- `driver.switch_to.frame()` — frame me switch karna.
- `driver.switch_to.default_content()` — parent/main page me wapas aana.

Example:

Frame Handling Example

```
driver.get("https://example.com/frame-demo")

# Frame ko locate karo (id/name ke through)
frame = driver.find_element(By.ID, "frame1")

# Frame switch karna
driver.switch_to.frame(frame)

# Ab frame ke andar elements access kar sakte hain
inner_element = driver.find_element(By.ID, "inside_frame")
print(inner_element.text)

# Frame se bahar (main page) switch karna
driver.switch_to.default_content()
```

- `switch_to.frame(frame)` se driver us specific frame ke andar ka context le leta hai.
- `default_content()` se wapas main page pe aa jaate hain.

Frames ko index ya name se bhi switch kar sakte hain:

Frame Switch by Index or Name

```
driver.switch_to.frame(0)              # pehla frame (index 0) me
    switch
driver.switch_to.frame("frameName")    # frame name se switch
```

Switching Between Windows/Tabs

Why important?

- Jab test me new tab/window open ho, toh default selenium driver usi original tab/window par rahta hai.
- Aapko script me explicitly nayi window/tab par switch karna hota hai.
- Nahi kare toh element nahi milega ya galat window pe commands dene lagenge.

When to use?

- Popups, social logins, payment gateways jaha window/tab switch hota hai.

Methods:

- `driver.window_handles` — list deta hai sab open windows/tabs ke handles.
- `driver.current_window_handle` — current window handle.
- `driver.switch_to.window(handle)` — switch karna specific window/tab pe.

Example:

Window/Tab Switching Example

```python
driver.get("https://example.com")

# maan lo button click se new tab open hota hai
new_tab_button = driver.find_element(By.ID, "newTabBtn")
new_tab_button.click()

# Sab window handles le lo
windows = driver.window_handles
print(windows)  # e.g. ['CDwindow-1', 'CDwindow-2']

# Current window handle
current_window = driver.current_window_handle

# New tab/window pe switch karna (jo current se alag ho)
for window in windows:
    if window != current_window:
        driver.switch_to.window(window)
        break

# Ab new tab me jaa ke kaam karo
print(driver.title)

# Jab kaam ho jaye, wapas main tab par aana:
driver.switch_to.window(current_window)
```

Summary

Feature	Method/Concept	Use Case / Why Important
JavaScript alerts/confirm/prompt	`driver.switch_to.alert`	Alert/dialog aane pe usko handle karna (accept/dismiss/input)
Frames/Iframes	`driver.switch_to.frame()`	Frame ke andar element access ke liye frame switch karna
Frames se bahar aana	`driver.switch_to.default_content()`	Frame se main page pe wapas jana
Multiple windows/tabs switching	`driver.switch_to.window(handle)`	Naye tab/window pe automate karne ke liye switch karna

==========

14 Advanced Actions & Mouse Events

Selenium mein basic click/send_keys toh aa gaya, lekin real-world automation mein mouse hover, double click, right-click, keyboard shortcuts, ya page pe scroll bahut zaruri hai. In ke liye ActionChains aur kuch built-in methods use hote hain.

Why Important?

- **Mouse Hover (hover actions):** Menus, tooltips, dropdowns ya hidden items ko dikhane ke liye zaruri hai.
- **Double/right click:** Kuch features ya menus sirf double-click ya right-click pe open hote hain.
- **Keyboard actions:** Form automation, navigation, hotkey usage test karne ke liye.
- **Scroll:** Bahut saare web pages pe elements scroll ke baad hi visible hote hain—bina scroll kiye interact nahi kar sakte.

Agar na use karein toh: Test wahan atak jaayega jahan page ya element visible nahi hai, ya action expected nahi ho raha.

ActionChains: Mouse Hover, Double Click, Right Click

Selenium me advanced mouse actions ke liye `ActionChains` ka use hota hai, jis se aap mouse se jo bhi manually karte the, woh sab automate ho jata hai.

Example Setup (Import):

Imports for Advanced Actions

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.action_chains import ActionChains
from webdriver_manager.chrome import ChromeDriverManager
```

- `webdriver`: Main driver, browser control karta hai.
- `By`: Element ka locator strategy (id, name, etc.) ke liye.
- `ActionChains`: Advanced mouse/keyboard actions ke liye.
- `ChromeDriverManager`: Driver ka auto-setup.

1. Mouse Hover (move_to_element)

Mouse Hover Example

```
driver = webdriver.Chrome(ChromeDriverManager().install())
driver.get("https://example.com")

menu = driver.find_element(By.ID, "menu")            # Menu element
    locate kiya
actions = ActionChains(driver)                       # ActionChains
    ka object banaya
actions.move_to_element(menu).perform()              # Mouse pointer
    menu pe hover kiya
```

- `move_to_element(menu)`: Mouse ko "menu" ke upar le gaya.
- `perform()`: Jo bhi actions ab tak queue hue, woh run kiye.

2. Double Click (double_click)

Double Click Example

```
el = driver.find_element(By.ID, "doubleClickBtn")    # Element
    locate kiya
actions.double_click(el).perform()                   # Uspe
    double-click kiya
```

`double_click(el)`: Us element pe tez dubble-click hua.

3. Right Click (context_click)

Right Click Example

```
el = driver.find_element(By.ID, "rightClickBtn")     # Element
    locate kiya
actions.context_click(el).perform()                  # Uspe
    right-click (context-click) kiya
```

`context_click(el)`: Us element pe right mouse button click kiya.

Keyboard Actions: Keys.TAB, Keys.ENTER

Selenium me keyboard keys send karne ke liye `Keys` class hoti hai.

Keyboard Actions Example

```
from selenium.webdriver.common.keys import Keys

input_box = driver.find_element(By.ID, "inputBox")
input_box.send_keys("Hello Bro")
input_box.send_keys(Keys.TAB)          # Tab dabaya (next field pe
    jump)
input_box.send_keys(Keys.ENTER)        # Enter dabaya (jaise form
    submit)
```

- `send_keys(Keys.TAB)`: Tab key send karta hai—form navigation ke liye.
- `send_keys(Keys.ENTER)`: Enter key send karta hai—button click ya form submit ke liye.

Agar aap bina element directly keyboard key bhejna chahte ho:

Direct Keyboard Key Example

```
actions.send_keys(Keys.ENTER).perform()
```

Direct browser ke focused element pe key send ho jaata hai.

Scroll to Element

Agar element page pe niche ya upar chhupa hai, toh usko scroll karke view mein lana zaruri hai. Selenium do tareeke deta hai:

1. **ActionChains se scroll:**

Scroll using ActionChains

```
element = driver.find_element(By.ID, "myID")
actions.move_to_element(element).perform()    # Element ke paas
    scroll ho jayega
```

`move_to_element(element)`: Mouse ko element tak le jaata hai, aur page ko us position pe scroll kar deta hai.

2. **JavaScript se scroll:**

Scroll using JavaScript

```
element = driver.find_element(By.ID, "myID")
driver.execute_script("arguments[0].scrollIntoView(true);", element)
```

`execute_script("arguments[0].scrollIntoView(true);", element)`: Direct JavaScript chala ke element ko screen ke view mein le aata hai.

Useful jab ActionChains ka scroll work na kare, ya precise position chahiye.

Scrolling with JavaScript: Custom Scroll

Agar aap page pe custom vertical/horizontal scroll karna chahte hai (e.g., end pe, top pe, pixel-wise):

JavaScript Custom Scroll Examples

```
# Scroll to bottom of page
driver.execute_script("window.scrollTo(0,
    document.body.scrollHeight);")

# Scroll to top of page
driver.execute_script("window.scrollTo(0, 0);")

# Scroll by 200 pixels vertically
driver.execute_script("window.scrollBy(0, 200);")
```

JavaScript ke `window.scrollTo` aur `window.scrollBy` se aap page ko kisi bhi position pe scroll kara sakte ho.

Summary Table

Action	Method/Function	Kyun Zaruri Hai
Mouse hover	`ActionChains.move_to_element().perform()`	Hidden menus, hover
Double click	`ActionChains.double_click().perform()`	Special interaction
Right-click	`ActionChains.context_click().perform()`	Context menu
Keyboard (TAB/ENTER/other keys)	`send_keys(Keys.TAB/ENTER)`	Navigation
Keyboard (actions, no element focus)	`ActionChains.send_keys(Keys.KEY).perform()`	Direct key input
Scroll to element	`move_to_element()` or JS `scrollIntoView()`	Element not visible
Custom Scrolling (JavaScript)	`driver.execute_script("scroll...")`	Pixel-wise, full-page

==========

Waits & Synchronization in Selenium

Why Is Synchronization Important?

Modern web apps slow ya dynamically load hote hai—kuch cheezein turant nahi dikhai deti, ya kuch actions background mein chal rahe hote hain.

Agar Selenium script ne element pe action kar diya aur element abhi page pe aya hi nahi, toh error aa jayegi.

Synchronization techniques ensure ki tumhaari script tab tak ruk jaye jab tak element ready na ho, taki test reliable rahe123.

Kab use karein?

- Jab elements dynamically aa rahe ho.
- Jab web app slow hai ya network ke chaal pe depend karta hai.
- Jab apne test stable aur robust banane hain.

Agar use nahi karein toh?

- Scripts unreliable, kabhi pass kabhi fail.
- False negatives (bugs nahi milte, ya sahi test case fail ho jata hai).
- CI/CD pipeline mein problems aati hain.

Sync Problems in Web Testing

- Page load hota hai, but element baad mein aata hai.
- Animations, Ajax, ya JS events ke wajah se timing mismatch ho jata hai.

- Selenium DOM ko real-time poll nahi karta, isliye mismatch easily ho jaate hain12.

Selenium Wait Types (with Python examples and step-by-step explanation)

1. Implicit Wait

Kya hai? Global timeout—jitne bhi `find_element()` ya similar locators hain, un sab pe ek wait lag jaata hai. Agar element turant mil jaata hai toh aage badh jayega, nahi toh set time tak try karega.

Syntax:

```
driver.implicitly_wait(5)
```

Yahan driver 5 secs tak try karega element dhoondhne ka, har `find_element()` ke liye4567.

Pura session global hai—jab set kar diya, jab tak close nahi karte, apply hota hai.

Pros: Simple

Cons: SLOW, har element pe lagega, kabhi-kabhi expected nahi hota.

Best for: Static ya mostly predictable websites.

2. Explicit Waits (WebDriverWait, ExpectedConditions)

Kya hai? Tum khud define karo kisko (kaunsi condition) kitni der tak wait karna hai.

Syntax:

```
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

driver = webdriver.Chrome()
driver.get("https://example.com")

# Wait up to 10 sec till element with ID 'myID' is present
element = WebDriverWait(driver, 10).until(
    EC.presence_of_element_located((By.ID, "myID"))
)
```

- `WebDriverWait`: Maximum 10 secs tak wait karega.
- `EC.presence_of_element_located`: Jaise hi condition true ho gayi (element mil gaya), wait break ho jayega.
- Other conditions: visibility, clickable, text present, title, etc.489710

Pros: Targeted, efficient, only as long as needed

Cons: Thoda code zyada lagta hai

Best for: Dynamic web apps, real-world use

3. Fluent Waits

Kya hai? Explicit ka advanced version—yahan tum define kar sakte ho:

- Max time kitna,
- Kitni baar poll kare,
- Kaunse exceptions ignore karein.

Syntax:

```
from selenium.webdriver.support.ui import WebDriverWait
from selenium.common.exceptions import NoSuchElementException

wait = WebDriverWait(
    driver,
    timeout=10,
    poll_frequency=2,                    # Har 2 sec me check karega
    ignored_exceptions=[NoSuchElementException]
)

element = wait.until(
    lambda driver: driver.find_element(By.ID, "myID")
)
```

Maximum 10 seconds tak, har 2 seconds me retry karega.

Agar `NoSuchElementException` aayi toh ignore karega, next poll karega11712.

Pros: Zyada control, perfect for flaky/dynamically loaded elements

Cons: Complexity thodi badh jati hai

4. Custom Waits using Lambda Functions

Agar tumhara condition predefined ExpectedConditions me nahi hai, toh tum lambda ya custom function de sakte ho:

```
element = WebDriverWait(driver, 10).until(lambda d:
    d.find_element(By.CLASS_NAME, "custom-class").is_displayed())
```

Jab tak element visible nahi hota, yeh check chalta rahega.

Kisi bhi custom condition ke liye use kar sakte ho137.

5. Page Load Strategies

By default Selenium poori page loading (CSS/images/js) ka wait karta hai. But kabhi-kabhi tumhe jaldi proceed karna hota hai (like only DOM loaded but images pending).

Strategies:

- `normal` (default): Wait till everything loaded (recommended for most).
- `eager`: Wait only till DOMContentLoaded.
- `none`: Bilkul bhi wait na karo, jaise hi navigation ho jaye, code aage badh jaye.

Python Example:

```
from selenium.webdriver.chrome.options import Options

options = Options()
options.page_load_strategy = 'eager'  # 'normal', 'eager', or 'none'
driver = webdriver.Chrome(options=options)
```

Aap script ki requirement ke hisaab se set kar sakte ho, lekin none risky hai1415.

Summary Table

Wait Type	Scope	Use-Case	Pros	Cons
Implicit Wait	Global	Mostly static sites	Easy, set once	Can slow suite, not targeted
Explicit Wait	Per action	Dynamic elements, AJAX, etc.	Accurate, reliable	More code needed
Fluent Wait	Per action	Unstable/delayed elements	Max control	Little complex
Custom Waits	Per action	Unusual conditions	Any logic	Must handle error
PageLoadStrategy	Session	Fast page(s), partial loads	Speed boost	Risk of missing elements

Golden Rules:

- **Do NOT mix implicit and explicit waits.** Unpredictable results milte hain!7
- Be specific: Only wait as much as needed, warna scripts slow ho jayengi.
- Debug properly: Wrong waits == hard to fix flaky scripts.

---

15 Working with Web Tables in Selenium Python

Why Important Hai?

- Web tables mein data organize hota hai rows aur columns mein.
- Testing mein zaruri hota hai tables se data nikalna, check karna, ya koi row filter karna.
- Agar table dynamic hai (rows change hote hain), toh handle karna thoda tricky hota hai.

- Agar na samjho toh data verification mushkil ho jayega, ya test fail hoga.

1. Table Data Read Karna (Rows aur Columns)

Step-by-step example

Suppose ek simple HTML table hai jisme rows ke andar columns hain. Humko poora table data read karna hai:

Table Data Read Example

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from webdriver_manager.chrome import ChromeDriverManager

driver = webdriver.Chrome(ChromeDriverManager().install())
driver.get("https://example.com/webltablepage")

# Table locate karo - maan lo ID se mil raha hai
table = driver.find_element(By.ID, "exampleTable")

# Table ke sare rows lo tbody ke andar se
rows = table.find_elements(By.TAG_NAME, "tr")

# Loop se har ek row ke cells read karna
for row in rows:
    cells = row.find_elements(By.TAG_NAME, "td")  # td matlab
        columns
    for cell in cells:
        print(cell.text, end=" | ")  # har cell ka text print karo
    print()   # naya line next row ke liye
```

Explanation:

- `find_element(By.ID)` table locate karta hai.
- `find_elements(By.TAG_NAME, "tr")` se table ke rows milte hain.
- Har row mein `find_elements(By.TAG_NAME, "td")` se columns milte hain.
- `cell.text` se us column ka text milta hai.
- Print karke table ka pura data console mein dekh sakte ho.

2. Search/Filter Specific Rows

Agar aapko table me se kuch specific data wala row find karna ho, jaise kisi column me "India" hona chahiye, toh aise karenge:

Search Specific Row Example

```
for row in rows:
    cells = row.find_elements(By.TAG_NAME, "td")
    for cell in cells:
        if "India" in cell.text:      # Agar kisi cell me 'India' hai
            print("Found Row:", [c.text for c in cells])
            break  # row mil gaya toh next row pe jao
```

Advanced: XPath se directly kisi condition wale rows dhoondhna

Search Rows with XPath

```
rows = driver.find_elements(By.XPATH,
    "//table[@id='exampleTable']//tr[td[contains(text(),'India')]]")
for row in rows:
    print([cell.text for cell in row.find_elements(By.TAG_NAME,
        "td")])
```

Explanation:

- XPath me condition lagake woh rows dhoondh raha hai jisme kisi bhi column ka text 'India' ho.

3. Dynamic Content Handling in Web Tables

Dynamic table ka matlab rows/columns page load ke baad change ho sakte hain—jaise pagination, sorting ya Ajax calls ki wajah se data update hota hai.

Key Tips:

- Explicit Wait lagao taaki table ke rows fully load ho jayein.
- Pagination aaye toh uske buttons ko automate karo aur data extract karo.
- Rows ya columns count har baar calculate karo, na ki fixed number treat karo.

Wait example for dynamic content:

Explicit Wait for Table

```
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

wait = WebDriverWait(driver, 10)
table = wait.until(EC.presence_of_element_located((By.ID,
    "exampleTable")))
rows = table.find_elements(By.TAG_NAME, "tr")
```

Pagination example (agar table multiple pages me data rakhta hai):

Pagination Handling Example

```
while True:
    rows = driver.find_elements(By.XPATH,
        "//table[@id='exampleTable']/tbody/tr")
    for row in rows:
        print([cell.text for cell in row.find_elements(By.TAG_NAME,
            'td')])
    next_button = driver.find_element(By.ID, "nextPageBtn")
    if "disabled" in next_button.get_attribute("class"):
        break  # end of pages
    else:
        next_button.click()
        wait.until(EC.staleness_of(rows[0]))  # Wait for table to
            reload new data
```

Explanation:

- Har page ke rows print kiye jaate hain.
- Next button disable hone par loop end ho jata hai.

Summary Table

Feature	Selenium Methods Used	Use Case / Explanation
Locate Table	`find_element(By.ID or XPATH)`	Locate the full table element
Get Rows	`table.find_elements(By.TAG_NAME, "tr")`	Get all rows in table
Get Columns/Cells	`row.find_elements(By.TAG_NAME, "td")`	Get cells in each row
Read Cell Text	`cell.text`	Get visible data from each cell
Search Rows	Loop condition or `XPath` with `contains(text(),...)`	Filter rows containing specific text
Handle Dynamic Content	Use Explicit Waits, Recaculate rows each time	Handle changing table data or pagination
Pagination Handling	Click next buttons, wait for reload, continue	Traverse all pages of table automatically

Data-Driven Testing (DDT) in Selenium Python — Full Detailed Explanation

1. DDT Kya Hai? (What is Data-Driven Testing?)

- **DDT matlab** test automation ka ek aisa approach jisme hum apne test scripts ko alag-alag data se chala sakte hain.
- Matlab ek hi test case ko multiple data sets ke saath run kar sakte hain, bina code change kiye.
- Data alag source (Excel, CSV, JSON, APIs) se liya jata hai aur test input/output uske hisaab se hota hai.

Example: Agar tumhara login page hai, toh alag-alag usernames/passwords se authentication test kar sakte ho bina baar-baar alag code likhe.

2. Kyun Use Karte Hain DDT? (Why Use Data Driven Testing?)

- Manual data ko baar baar change karna time-consuming hota hai aur galtiyan ho sakti hai.
- Automation me data ko alag rakh kar test maintain karna easy ho jaata hai.
- Same test script se multiple scenarios (valid, invalid, edge cases) ko efficiently test kar sakte hain.
- Scalability badh jaati hai, aur tests zyada reliable aur repeatable ho jaate hain.

3. Kab Use Karein? (When to Use Data-Driven Testing?)

- Jab test cases me bar-bar alag inputs se test karna ho (login, registration, search, filters).
- Jab smoke/regression boundary aur negative test cases banana ho.
- Jab large test data ho jo baar-baar input lena ho.
- Jab test automation ko modular aur maintainable banana ho.

4. Agar Na Karo DDT Toh Kya Problem Hai? (If Not Used, Then?)

- Bar-bar test script ka code copy-paste ya hardcoding mein extra time lagega.
- Test data update karna complicated ho jayega.
- Test cases jitne ho utne complex management ho jayega.
- Test coverage limited ho jayega edge cases ke liye.
- Automation slow aur error prone banega.

5. Kaise Implement Karte Hain DDT? (How to Implement Data-Driven Testing?)

Hum alag-alag data sources se data le sakte hain, phir us data ko test cases me use karte hain.

Option 1: Excel File se Data Read/Write karna (using openpyxl or pandas)

Excel Data Read karna (openpyxl ke saath) **Setup:** Pehle openpyxl install karo:

```
pip install openpyxl
```

Code Example:

```
import openpyxl

# Excel file load karo
wb = openpyxl.load_workbook("testdata.xlsx")

# Specific sheet select karo
sheet = wb['Sheet1']

# Row 2 se start karte hain (assuming row1 headings hain)
for i in range(2, sheet.max_row + 1):
    username = sheet.cell(row=i, column=1).value  # Column 1 ka data (username)
    password = sheet.cell(row=i, column=2).value  # Column 2 ka data (password)
    print(username, password)
```

Line-by-Line Explanation:

- `load_workbook("testdata.xlsx")` : Excel file ko memory me load karta hai.
- `sheet = wb['Sheet1']` : 'Sheet1' naam ke tab ka content access karta hai.
- `sheet.max_row` : Sheet me total rows ki count deta hai.
- Loop me har row ka username aur password uthaya gaya.
- `.cell(row=x, column=y).value` : specific cell ka value extract karta hai.

Excel Data Write karna Example

```
# Row 2 aur column 3 me "Login Pass" likh rahe hain
sheet.cell(row=2, column=3, value="Login Pass")

# Excel file ko save karna na bhoolo, tabhi changes rakhe jayenge
wb.save("testdata.xlsx")
```

Pandas Se Excel Data Read Karna (Simple aur Fast)

```
import pandas as pd

df = pd.read_excel("testdata.xlsx")

for index, row in df.iterrows():
    print(row["username"], row["password"])
```

- `pd.read_excel()` se pure Excel ko dataframe me convert karta hai.
- `df.iterrows()` se row by row data read karte hai.

Option 2: CSV File se Data Read Karna

```
import csv

with open("testdata.csv", newline='') as file:
    reader = csv.DictReader(file)  # Header ke hisaab se dictionary bana
        deta hai
    for row in reader:
        print(row["username"], row["password"])
```

- CSV files simple text format hote hain.
- `DictReader` se har row ek dict ban jata hai with keys as header names.

Option 3: JSON File se Data Read Karna

```
import json

with open("testdata.json") as file:
    data = json.load(file)
    for entry in data:
        print(entry["username"], entry["password"])
```

- JSON files structured data format hai, API responses me common use hota hai.
- Python `json.load()` se us data ko dictionary/list structure me load kar lete hain.

6. Pytest ke Saath Data Injection (parametrize decorator)

Kya Hai? (What is pytest.mark.parametrize?)

- Pytest ka ek powerful decorator hai jo ek test function ko multiple data sets ke saath baar-baar run karne deta hai.
- Manual loop ya multiple test likhne ki zarurat nahi.

Kab Use Karein?

- Jab same test case ko multiple data se run karna ho (login with multiple username/-password).
- Test code clean aur maintainable chahiye.

Agar Na Karein?

- Repetitive aur messy code
- Data ko manually change karna padega
- Test coverage kam ho jayega

Parametrize Example (Simple)

```
import pytest

@pytest.mark.parametrize("username,password", [
    ("user1", "pass1"),
    ("user2", "pass2"),
```

```
    ("user3", "pass3")
])
def test_login(username, password):
    print(username, password)
```

- Yahan test_login function 3 baar alag-alag data ke saath chalega.
- Har tuple me username aur password define kiye hain.

Excel/CSV Data pytest ke Saath Link Karna

Agar tum Excel/CSV se data directly chahte ho inject karna, toh ek helper function likh kar data lo, fir parametrize me use karo:

```
import pytest
import openpyxl

def get_excel_data():
    wb = openpyxl.load_workbook("testdata.xlsx")
    sheet = wb['Sheet1']
    data = []
    for i in range(2, sheet.max_row+1):
        username = sheet.cell(row=i, column=1).value
        password = sheet.cell(row=i, column=2).value
        data.append((username, password))
    return data

@pytest.mark.parametrize("username,password", get_excel_data())
def test_login(username, password):
    print(username, password)
```

7. Automated Form Submission with DDT

Example Code

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

@pytest.mark.parametrize("username,password", [
    ("user1", "pass1"),
    ("user2", "pass2"),
])
def test_form_submission(username, password):
    driver = webdriver.Chrome()
    driver.get("http://example.com/login")

    driver.find_element(By.ID, "username").send_keys(username)
    driver.find_element(By.ID, "password").send_keys(password)
    driver.find_element(By.ID, "btnLogin").click()

    # yahan assertions karo login success ke liye

    driver.quit()
```

Explanation

- Har test run me browser chal raha hai.
- Login page open hota hai.
- Username aur password input fields me data fill karte hain.
- Login button click karte hain.
- Multiple data sets ke saath baar-baar ye test repeat hota hai.

Final Summary Table: DDT Quick Reference

Task	Library / Tool	Description / Example
Excel read/write	openpyxl / pandas	Excel data read karke test me use karna
CSV read	csv	CSV data file se data lena
JSON read	json	JSON se structured data lena
Data injection in pytest	pytest.mark.parametrize	Multiple data sets ek test function me
Form submission automation	Selenium webdriver	send_keys, click ke saath data use karna

Golden Rules for DDT:

- Test data ko **code se alag rakho** (Alag files ya DB me).
- Data source fix karo, format standardize karo.
- Test failures ke liye data aur results ko track & log karna zaroori hai.
- Repeatable, scalable aur maintainable tests banao.

==========

Page Object Model (POM) Framework — Complete Guide with Examples

16 What is Page Object Model (POM)?

- POM ek design pattern hai jo test automation me use hota hai.
- Iska purpose hai **UI elements ka access aur actions ko organize karna ek structured tarike se**.
- Matlab: Har webpage ya screen ka ek **Python class** banta hai jisme us page ke **sare locators** (buttons, fields, links) aur **actions** (click, type karna, read karna) likhe jaate hain.

Example:

Login page ka ek `LoginPage` class banega jisme username/password field ke locators aur login button click karne ka method hoga.

17 Why POM Use Karna Zaroori Hai?

- **Code Reusability**: Same UI element locator baar-baar har test me likhne ki zarurat nahi. Ek jagah update karo, sab jagah update ho jaayega.
- **Maintainability**: Agar UI me change aaye, toh sirf us page class me locator change karna padega, test scripts ko touch karne ki zarurat nahi.
- **Readability**: Test scripts bahut clean aur readable ho jaate hain, kyunki wo business steps par focus karte hain, technical UI details par nahi.
- **Separation of Concerns**: UI representation (Page class) aur test logic (test scripts) alag-alag rehta hai, jisse code organized banta hai.

18 When Should You Use POM?

- Jab **complex ya bade projects** pe kaam kar rahe ho.
- Jab **multiple testers/developers** ek saath automation code pe kaam karte ho.
- Jab test cases **long term ke liye maintainable** hone chahiye.
- Jab aap chahte ho ki **automation framework professional aur scalable** bane.

19 Agar POM Na Use Karo: Problem Kya Hogi?

- Test code me **bohot sari duplicate locators/actions** honge.
- UI me chhote bade changes par sari test scripts toot sakti hain.
- Project ka maintenance **husarang mushkil aur costly** ho jayega.
- Code **readability aur modularity khatam** ho jaayegi — har test me UI details fir se likhne padenge.

20 POM Structure Ke Core Concepts

20.1 Page Classes (Locators + Actions)

- Ek webpage ko represent karta hai.
- Har element ka locator define hota hai (eg: `By.ID`, `By.XPATH`, etc.).
- Jo actions user page pe karta hai (text type karna, buttons click karna) wo methods ke form me likhe hote hain.

Example: `LoginPage.py`

LoginPage.py Example

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

class LoginPage:
    def __init__(self, driver):
        self.driver = driver
        # Locators: har element ke liye tuple (Locator type,
           locator value)
        self.username_input = (By.ID, "username")
        self.password_input = (By.ID, "password")
        self.login_button = (By.ID, "loginBtn")

    # Actions as methods:
    def enter_username(self, username):
        self.driver.find_element(*self.username_input).send_keys(username)

    def enter_password(self, password):
        self.driver.find_element(*self.password_input).send_keys(password)

    def click_login(self):
        self.driver.find_element(*self.login_button).click()
```

Line-by-line Explanation:

- `__init__(self, driver)`: Constructor jisme browser driver pass hota hai, jiske saath ye page kaam karega.

- `self.username_input = (By.ID, "username")` : Yetuplelocatordefinekartahai, jismepehlelocatorke

20.2 Reusability & Maintainability

- Page class me ek baar locator aur actions define karo, fir **sab test scripts me reuse karo**.
- Agar locator change hota hai to sirf page class me change karo—baaki test scripts automatic update ho jaayenge.
- Test scripts zyada clean aur business logic par focused hote hain, UI complexity mein nahi uljte.

20.3 BasePage & TestBase Structure

BasePage.py

- Common helper methods sabhi pages ke liye. Jaise explicit wait, error handling, scrolling etc.

BasePage.py Example

```
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

class BasePage:
    def __init__(self, driver):
        self.driver = driver

    def wait_for_element(self, locator, timeout=10):
        return WebDriverWait(self.driver,
            timeout).until(EC.presence_of_element_located(locator))
```

- Isko sab pages inherit kar sakte hain — code repeat nahi hoga.
- Example: LoginPage could inherit BasePage to use wait methods.

TestBase.py

- Test suite run karne ke liye setup/teardown ka kaam karta hai:

TestBase.py Example

```
import unittest
from selenium import webdriver
from webdriver_manager.chrome import ChromeDriverManager

class TestBase(unittest.TestCase):
    def setUp(self):
        self.driver =
            webdriver.Chrome(ChromeDriverManager().install())
        self.driver.maximize_window()

    def tearDown(self):
        self.driver.quit()
```

- Har ek test class ko `TestBase` inherit kar lena chahiye — tool setup and cleanup handle ho jayega.

20.4 Utility Classes — Waits, Logger, Screenshots

- **WaitUtils:** Explicit, fluent wait ke methods ko yahan rakho for reuse.
- **Logger:** Automation run time info ko log karne ke liye logger setup.
- **ScreenshotUtils:** Test fail hone par screenshot capture aur save karne ke liye helper class.

ScreenshotUtils Example

```python
import time

class ScreenshotUtils:
    def __init__(self, driver):
        self.driver = driver

    def take_screenshot(self, name='screenshot'):
        timestamp = time.strftime("%Y%m%d-%H%M%S")
        filename = f"{name}_{timestamp}.png"
        self.driver.save_screenshot(filename)
```

Screenshot Example:

- Har test ke fail hone par screenshot lena easy ho jaata hai.

21 Ek Simple End-to-End Example — Test Case Using POM and TestBase

test_login.py Example

```python
# test_login.py
from TestBase import TestBase
from LoginPage import LoginPage

class TestLogin(TestBase):
    def test_valid_login(self):
        self.driver.get("http://example.com/login")

        login_page = LoginPage(self.driver)
        login_page.enter_username("testuser")
        login_page.enter_password("testpass")
        login_page.click_login()

        # yahan assertions lagao jaise dashboard load hua ya nahi
```

Explanation:

- `TestLogin` class `TestBase` inherit karta hai jisme driver setup/teardown hai.
- `self.driver.get()` browser ko page open karne ke liye.
- `LoginPage` object create kiya, jise driver pass kiya.
- UI interaction methods ko call kiya, test ka main flow simple aur readable bana.
- Assertions add karke verify kar sakte hain ki login successful hua ya nahi.

22 Summary Table (POM Components)

POM Component	Kya Kaam Karta Hai / Kya Contain Karta Hai	Kyun Zaroori Hai / Use Kyun Karte Hain
Page Class	Locators + Page specific actions/methods	Reusability, locator update easy
BasePage	Common utilities jaise waits, scrolling, error handling	DRY code, shared common flows
TestBase	Browser setup, initialization, teardown	Har test ke liye common driver/session management
Utility Classes	Logging, screenshots, waits etc.	Framework ki maintainability, debugging enhance karne ke liye

23 Golden Rule

> Agar POM use nahi karoge, toh tumhara automation code **duplicate**, **mushkil maintain karne wala**, aur **unstable** ho jayega. Chhoti chhoti UI changes me puri test suite fail hogi.
>
> Isliye, **POM sikhna aur use karna job-ready automation ka sabse zaroori step hai**.

Agar chaho main ready-to-use folder structure, multiple page class examples, advance features jaise inheritance use karke common utility kaise banao, ya kisi aur specific doubt clear karu toh batao bro! Main tumhari madad ekdum step-by-step karne ko tayyar hoon!

---

24 PyTest Framework (Industry-Standard) — Beginner Friendly Guide

1. PyTest Installation & Usage

What?

- PyTest ek popular Python testing framework hai.
- Ye simple hai, lightweight hai aur bahut powerful features provide karta hai.
- Automated test likhne aur chalane ke liye use hota hai.

Why?

- PyTest me test likhna bohot simple aur expressive hota hai.
- `assert` ka use karke expected vs actual check kar sakte ho.
- Fixtures, markers, parametrization easy hain.
- Large projects me scalable aur modular banata hai.

When?

- Jab bhi Python me automated tests banana ho, chahe unit ho, integration ho, ya Selenium automation ho.

If Not Used?

- Instead of pytest, agar basic `unittest` use karoge toh boilerplate zyada hota.
- Complex fixtures aur parametrization manually implement karna mushkil.
- Test execution aur reporting limited hoti hai.

Installation Command:

PyTest Installation

```
pip install pytest
```

Basic Usage:

- PyTest automatically test files ko find karta hai jiska naam `test_` se shuru ho ya `_test` se khatam ho.
- Test functions ka naam `test_` se start hona chahiye, jaise `def test_addition():`.

Simple Test Example:

Simple Test Function

```
1 def test_addition():
2     assert 2 + 3 == 5
```

Explanation:

- `def test_addition()` : Ye ek test function hai jo PyTest ko run karna hai.
- `assert 2 + 3 == 5`: Assert Python ka keyword hai, check karta hai expression true hai ya nahi. True hai toh test pass hoga.

PyTest Run Command:

Run PyTest

```
pytest
```

Ye command current folder me test files dhundh ke run karta hai.

2. Writing Tests with Assert

What?

- `assert` keyword built-in Python ka hota hai.
- Iska use to verify actual vs expected results.

Why?

- Simple assertion se pata chalta hai test pass hai ya fail.
- PyTest fail hone pe error message dete hai.

When?

- Har test me use karo to verify correctness.

If Not Used?

- Test invalid hoga, ya Google/test framework ko pata nahi chalega test success ya failure ka.

Assertion Examples:

Passing assertion:

Passing Assertion

```
def test_subtract():
    result = 10 - 5
    assert result == 5    # Test will pass
```

Failing assertion:

Failing Assertion

```
def test_fail():
    assert 1 == 0     # Test fail hoga with AssertionError
```

Jab assertion fail hota hai, PyTest detailed traceback aur error show karta hai console me.

3. Organizing Tests into Modules

What?

- Tests ko logically organize karne ke liye alag-alag Python files ya folders me rakhna.

Why?

- Large projects me test management asaan hota hai.

- Bugs trace karna, specific test group chalana easy hota hai.

When?

- Jab test suite bada ho.
- Jab multiple test categories ho.

If Not Used?

- Sab tests ek file me hone se code messy ho jayega.
- Run/select karna specific subset mushkil hoga.

Folder Structure Example:

```
tests/
    test_login.py
    test_checkout.py
    test_search.py
```

PyTest recursively `tests/` directory scan kar ke sari test files aur functions run karega.

4. Fixtures: Setup & Teardown at Different Levels

What?

- Fixtures predefined functions hote hai jo test se pehle setup aur test ke baad teardown me chalate hain.
- Ye reusable components hote hain jisse test code clean aur DRY banta hai.

Why?

- Repetitive setup/cleanup code ko ek jagah manage karna asaan hota hai.
- Browser launch, database connect, environment prepare karna fixtures me hota hai.

When?

- Jab common pre-condition aur post-condition chahiye ho tests ke liye.

If Not Used?

- Har test me setup/cleanup likhna padega — code duplicate hoga.
- Maintenance tough hoga.

Simple Fixture Example:

Fixture Example

```python
import pytest

@pytest.fixture
def setup_browser():
    print("Setup before test")
    yield
    print("Teardown after test")

def test_example(setup_browser):
    print("Test running")
    assert True
```

Explanation:

- `@pytest.fixture` decorator se function fixture ban gaya.
- `yield` ke pehle ki line setup ke liye, baad ki line teardown ke liye hoti hai.
- $\texttt{test}_{e}xamplefunctionmefixture\texttt{setup\_browser}koargumentkitarahdiyagaya|PyTestuskoautomat$

Fixture Scopes:

Scope	Explanation	Example
function	Har test function ke liye alag call (default)	`scope="function"`
class	Ek test class ke sab tests ke liye ek baar call	`scope="class"`
module	Ek file ke sab tests ke liye ek baar call	`scope="module"`
session	Pure test session ke liye ek baar chalata hai	`scope="session"`

Example Fixture with Scope:

Scoped Fixture Example

```python
@pytest.fixture(scope="module")
def setup_module():
    print("Setup module")
    yield
    print("Teardown module")
```

5. Hooks: Custom Code During Test Lifecycle

What?

- Hooks special functions hote hain jo PyTest ke test run ke different phases me automatically call hote hain.

Why?

- PyTest run ko customize karne ke liye.
- Jaise failures pe logs capture karna, report me kuch add karna, ya custom setup karna.

When?

- Advanced test framework banate wakt.
- Custom logging & reporting me.

If Not Used?

- Test execution me flexibility kam hogi.
- Some features ko automate nahi kar paoge.

Example Hooks in `conftest.py`:

Hook Examples

```python
def pytest_runtest_setup(item):
    print(f"Setting up {item.name}")

def pytest_runtest_makereport(item, call):
    if call.when == 'call':
        if call.excinfo is not None:
            print(f"Test {item.name} failed")
        else:
            print(f"Test {item.name} passed")
```

Explanation:

- `item.name`: current test ka name.
- `call.when`: test ke phase (setup, call, teardown).
- `call.excinfo`: exception info agar test fail hua to.

6. Markers: Test Functions Ko Categorize Karna

What?

- Markers decorations hote hain jisse test ko category/class assign karte hain.

Why?

- Different suite run karna easy hota hai.
- Kuch test ko skip/xfail karne me madad karta hai.

When?

- Smoke, regression, sanity tests ko alag chalana ho.
- Kuch tests temporarily skip karna ho.

If Not Used?

- Sab test ek saath chalayenge, time zyada lagega.
- Filters aur reports me confusion hoga.

Common Marker Examples:

Marker Examples

```python
import pytest

@pytest.mark.smoke
def test_smoke1():
    assert True

@pytest.mark.regression
def test_regression1():
    assert True

@pytest.mark.skip(reason="Skipping temporarily")
def test_skip():
    assert False

@pytest.mark.xfail(reason="Known bug")
def test_expected_fail():
    assert False
```

Run only smoke tests:

Run Only Smoke Tests

```
pytest –m smoke
```

- `skip` wale test run nahi honge.
- `xfail` expected failed tests ko mark karta hai taaki report me pass/fail wisely dikhe.

Custom Markers Add Karna (`pytest.ini`):

```
[pytest]
markers =
    smoke: smoke tests
    regression: regression tests
    sanity: sanity tests
```

7. Parallel Execution With pytest-xdist

What?

- Multiple test cases ek saath (parallel) chalana.

Why?

- Test execution time kam karna.
- Large test suites ke liye must.

When?

- Jab test suite bada ho.
- Continuous integration pipelines me fast feedback chahiye.

If Not Used?

- Sequential execution slow hoga.
- Productivity kam hogi.

Installation:

Parallel Execution Installation

```
pip install pytest-xdist
```

Run Tests in Parallel:

Run Tests in Parallel

```
pytest -n 4
```

- 4 CPU threads pe parallel tests chalenge.
- `-n auto` se CPU cores auto detect karke chalata hai.

Best Practices:

- Tests ko isolate karo.
- Shared global state avoid karo.
- Data conflicts na ho aise likho tests.

Summary Table

Feature	Purpose / Explanation	Example
PyTest Installation	PyTest install karna	`pip install pytest`
Writing Tests	Assert se test likhna	`def test_func(): assert True`
Organizing Tests	Multiple files and folders me organize karna	Folder jaise `tests/test_login.py`
Fixtures	Setup/teardown reusable code	`@pytest.fixture` use karo
Hooks	Custom actions during test lifecycle	`pytest_runtest_setup`, `pytest_runtest_makereport`
Markers	Test categorize karne ke liye	`@pytest.mark.smoke`
Parallel Execution	Tests ko ek saath chalana	`pytest -n 4`

Ek Simple Example:

Simple PyTest Example with Fixture and Markers

```python
import pytest

@pytest.fixture(scope="module")
def setup_data():
    print("Setup module data")
    yield
    print("Teardown module data")

@pytest.mark.smoke
def test_addition(setup_data):
    assert 2 + 2 == 4

@pytest.mark.regression
def test_subtraction():
    assert 5 - 3 == 2

@pytest.mark.skip(reason="Not ready yet")
def test_skip_me():
    assert False
```

Run command (Smoke tests verbose):

Run Smoke Tests Verbose

```
pytest -m smoke -v
```

Agar Chahiye Toh

Main aapko advanced PyTest frameworks, detailed examples, Selenium integration, reporting logging, aur CI/CD pipelines ke liye PyTest ka best use bata sakta hoon. Bas

batayega!

==========

25 Logging, Reporting & Debugging - Complete Structure se Explanation

1. Logging with `logging` Module

Why Logging?

- Jab test chal rahe hote hain toh har action, step, warning, aur error ko record karna zaroori hota hai.
- Agar koi bug ya failure aaye toh logs dekh kar pata lagta hai kahan dikkat hai.
- Logs debugging, maintenance aur audit ke liye best practice hain.
- Team members ko bhi samajh aata hai ki test execution me kya hua.

When to Use?

- Har test automation framework me use karo.
- Especially bade projects me jahan multiple test cases aur users involved ho.
- Jab flaky tests ya complex failures diagnose karne ho.

Agar Na Use Karein Toh?

- Failures ki wajah samajhna mushkil ho jayega.
- Test results inconsistent lagenge.
- Debug karne me time aur effort zyada lagega.
- Automation ka fayda kam ho jayega.

Code Example + Line-by-Line Explanation:

Logging Setup Example

```python
import logging  # Python ka logging module import kar rahe hain

logging.basicConfig(
    level=logging.DEBUG,  # Sabse detailed level se logging shuru karenge (DEBUG se upar ke sab log honge)
    format='%(asctime)s - %(levelname)s - %(message)s',  # Log message ka format (timestamp, level, message)
    filename='test_log.log',  # Logs ek file me save karenge jiska naam test_log.log hoga
    filemode='w'  # Har baar naye run pe file overwrite hogi, purani delete ho jayegi
)

logging.debug("Ye debug message hai")
# DEBUG ka matlab sabse low level ka detail message     mostly development ke liye useful hai

logging.info("Test start ho gaya")
# INFO level ka message     jab test start ho jaye ya koi important milestone ho

logging.warning("Thoda dhyan se! Unexpected behavior")
# WARNING ka matlab kuch ajeeb ya unexpected hua, lekin abhi problem nahi

logging.error("Error aaya element locate karte waqt")
# ERROR ka message     jab error aaye ya koi step fail ho

logging.critical("Critical failure, pura test hang")
# CRITICAL sabse bada error, system ya test execution pura fail ho raha ho
```

2. Screenshot Capture (Full Page / Elements)

Why Important?

- Jab test fail ho jata hai to us waqt ka screenshot lena debugging ke liye bahut help karta hai.
- Visual proof milta hai ki screen pe kya dikh raha tha.
- Developers ya testers ko samajhne me ease hoti hai problems ko.

When to Use?

- Jab test case fail ho.
- Kabhi kabhi pass hone wale cases me bhi document karne ke liye.
- Automated screenshot generation on failure hooks me.

Agar Na Use Karein?

- Failures ka reason samajhna bohot mushkil hoga.

- Bug reproduce karne me time waste hoga.
- Report me proof kam hoga.

Code Example + Explanation:

Screenshot Capture Examples

```
driver.save_screenshot("full_page.png")
# Ye pura browser window ka screenshot "full_page.png" file me save
    karta hai

element = driver.find_element(By.ID, "submitBtn")
element.screenshot("element_screenshot.png")
# Sirf specific element ka hi screenshot "element_screenshot.png"
    file me save karega
```

3. PyTest HTML Reports (`pytest-html`)

Why Important?

- Readable aur user-friendly report chahiye jisme pass/fail clearly dikhai de.
- Automatic screenshot attach kar sakte hain failure ke time.
- Team members ko results easily share karne me help karta hai.

When to Use?

- Jab detailed test reports chahiye.
- Integration with CI/CD pipelines me.
- Jab manual intervention ke bina test results samajhne hain.

Agar Na Use Karein?

- Sirf console logs pe depend karna padega.
- Complex failures ko samajhna mushkil hoga.
- Dushre stakeholders ke liye sharing muskil hoti hai.

Code Example & Usage:

Install & Run pytest-html

```
pip install pytest-html

pytest --html=report.html --self-contained-html
```

Is command se `report.html` file generate hogi, jisme test results, errors, screenshots sab honge.

4. Allure Reports Integration

Why Important?

- Allure rich, interactive dashboard deta hai with graphs, history and detailed logs.
- Industry standard hai professional reporting ke liye.
- Jenkins jaisi CI tools ke saath easily integrate hota hai.

When to Use?

- Jab detailed analytics aur visual feedback chahiye.
- Large scale projects mein testing metrics track karne ke liye.

Agar Na Use Karein?

- Basic reporting hi milega, jo complex projects me insufficient hota hai.

Code Example & Usage:

Allure Installation & Usage

```
pip install allure-pytest

pytest --alluredir=allure-results

allure serve allure-results
```

Screenshot on failure (Allure integration):

Allure Screenshot Hook Example

```
import allure

@pytest.hookimpl(hookwrapper=True)
def pytest_runtest_makereport(item, call):
    outcome = yield
    rep = outcome.get_result()
    if rep.when == "call" and rep.failed:
        driver = item.funcargs['driver']  # Assuming 'driver'
            fixture name
        screenshot = driver.get_screenshot_as_png()
        allure.attach(screenshot, name="Screenshot",
            attachment_type=allure.attachment_type.PNG)
```

5. Screenshot on Failure (Hook-Based)

Why Important?

- Automatically screenshot lena jab bhi test fail ho jaye, taaki manual check na karna pade.

- Debugging easy ho jaati hai.

When to Use?

- Jab test suite me failures ho rahe hain aur fix karna ho.
- Continuous testing environment me.

Agar Na Use Karein?

- Failures me screenshot nahi milega.
- Debug karna time-consuming hoga.

Code Example with explanation:

Hook to Capture Screenshot on Failure

```python
import pytest

@pytest.hookimpl(hookwrapper=True)
def pytest_runtest_makereport(item, call):
    outcome = yield                          # Test execution result ka wait kar raha hoon
    rep = outcome.get_result()               # Result ko get kar liya
    if rep.when == 'call' and rep.failed:  # Agar actual test call phase me failure hai toh
        driver = item.funcargs.get('driver', None)  # Driver fixture ko access kiya
        if driver:
            screenshot_path = f"screenshots/{item.name}.png"
            driver.save_screenshot(screenshot_path) # Screenshot file me save kar diya
            print(f"Screenshot saved to {screenshot_path}")  # Console me message diya
```

6. Debugging Flaky Tests

Why Important?

- Flaky tests unpredictable hote hain — kab pass, kab fail.
- Production level automation ke liye flaky tests bohot problem create karte hain.

When to Use?

- Jab test results inconsistent aayein.
- Jab environment aur timings unstable ho.

Agar Na Use Karein?

- Automation suite unreliable ho jayega.
- Confidence lost hoga testing results pe.
- Time waste hoga failures ko repeat karke check karne me.

Common Fixes & Best Practices

- Use Explicit Waits (`WebDriverWait`) instead of `time.sleep()`.
- Improve locator reliability.
- Increase timeout and poll frequency.
- Isolate tests to avoid interference.
- Use retry plugins like `pytest-rerunfailures`.

Retry Example

Retry Plugin Installation and Usage

```
pip install pytest-rerunfailures

pytest --reruns 2
```

Agar test fail ho jaye to 2 baar automatically retry karega.

tabularx

Summary Table

—p3.5cm—p3.5cm—X—p3.5cm—

Concept	Why Important	When to Use	Code/Method Example
Logging	Steps, errors track karne ke liye	Har automation framework me	`logging.basicConfig()`, `logging.info()`
Screenshot Capture	Visual error proof	Failure ya debugging time	`driver.save_screenshot()`
PyTest HTML Reports	User-friendly report	CI/CD me report hone ke liye	`pytest --html=report.html`
Allure Reports	Advanced, interactive dashboards	Large scale professional projects	`pytest --alluredir=allure-results`
Screenshot on Failure (Hook)	Auto screenshot on test fail	Robust debugging	`pytest_runtest_maker hook`
Debugging Flaky Tests	Stability and reliability	Flaky tests track karne	`WebDriverWait`, `pytest-rerunfailures`

26 Visual & Accessibility Testing — Step-by-Step Guide

1. Full-Page Screenshots with Image Comparison (using PIL, OpenCV)

What is this?

- Ye technique hoti hai jisme hum poore web page ya application ka screenshot lete hain.
- Us screenshot ko baseline (pehle se store image) ke sath compare karte hain.
- Difference detect kar ke dekhte hain ki UI ya design me koi unexpected change to nahi hua.

Why important hai?

- Software ke UI (colors, fonts, layout, spacing) me chhote-chhote badlav ko manually pakadna mushkil hota hai.
- Manual inspection me visual bugs miss ho sakte hain, jo production me ja sakte hain.
- Automated comparison se pata lagta hai ki kya change hua, aur exact jagah color ya shape ka fark kaha hai.
- Isse hum production release se pehle bugs catch kar sakte hain jo visual quality ko effect karte hain.

When use karein?

- Har build ya release ke baad UI regression test karte waqt.
- Jab website me CSS, front-end code frequent change hote hain.
- Responsive design testing, cross-browser compatibility check karne ke liye.
- Jab user experience critical ho (E-commerce, banking app, complex portals).

Agar na karein toh?

- UI bugs unnoticed rahenge.
- Production me visual inconsistencies aayengi, user disappoint hoga.
- Brand reputation nuksaan me ja sakta hai.

Code Example (Screenshot + Image Comparison):

Image Comparison Example

```python
import pyautogui                # Screen capture ke liye
from PIL import Image           # Image processing ke liye
import numpy as np              # Array operations ke liye
import cv2                      # OpenCV, image difference aur processing ke liye

# Step 1: Screenshot lena
screenshot = pyautogui.screenshot()      # Screen ka screenshot capture karte hain
screenshot.save("current.png")           # Screenshot ko file me save karte hain

# Step 2: Pehle se saved baseline image (reference image) load karo
img1 = np.array(Image.open("baseline.png"))    # Pehle se stored image ko numpy array me convert karte hain
img2 = np.array(Image.open("current.png"))     # Abhi ka screenshot bhi numpy array me convert karte hain

# Step 3: Do images ka difference nikaalo (pixel level)
diff = cv2.absdiff(img1, img2)             # Absolute pixel difference, jahan pixels same nahi hote wahan white color create hota hai

# Step 4: Differences ko grey scale me convert karo
gray = cv2.cvtColor(diff, cv2.COLOR_BGR2GRAY)    # Color difference ko grayscale me badalna

# Step 5: Threshold apply karo jisse chhote mote changes ignore ho aur badi differences highlight hon
_, thresh = cv2.threshold(gray, 30, 255, cv2.THRESH_BINARY)

# Step 6: Difference ko nayi image file me save karo
cv2.imwrite("diff.png", thresh)

# Step 7: Check karo dono images bilkul same hain ya nahi
if img1.shape == img2.shape and np.all(img1 == img2):
    print("Images are identical!")
else:
    print("Images are different.")
```

Line-by-line explanation:

- `pyautogui.screenshot()` se current screen ka screenshot liya.
- `screenshot.save("current.png")` us screenshot ko disk pe "current.png" naam se save kiya.
- `Image.open()` se PNG image open kar ke `np.array()` se NumPy array banaya, jisse pixel wise calculations easy hote hain.
- `cv2.absdiff(img1, img2)` se dono images ke pixels ka absolute difference liya.
- `cv2.cvtColor(..., cv2.COLOR`$_B GR2GRAY)$$difference image ko grayscale me convert kiya.cv$
- `cv2.imwrite()` se difference image file me save kiya.

- Last me check kiya ki dono images bilkul same hain ya nahi.

2. Visual Regression Testing (Percy, Applitools)

What?

- Ye tools advanced visual testing ke liye automation handle karte hain.
- Apne build system ya test framework me integrations provide karte hain jisse har test run pe screenshots le ke automatically compare karte hain.
- AI/Smart diffing karte hain taaki false positives (chhoti chhoti insignificant changes) kam report ho.

Why important hai?

- Visual bugs ko jaldi identify karne me madad karta hai.
- Manual mistakes ya missed regression reduce hota hai.
- Cross-browser ya device wise UI changes manual baar-baar check karne ki zarurat nahi.

When use karein?

- Jab Continuous Integration/Delivery (CI/CD) pipeline me regression test hai.
- Feature-rich UI applications me jisme frequent UI changes hote hain.
- Cross platform, cross browser UI consistency ensure karna ho.

Agar na karo toh?

- Visual bugs production me chala jayega.
- Manual testing zyada time lega.
- UI consistency issues rahenge jo brand value ko hurt karenge.

Popular Tools:

- **Percy (BrowserStack)**: Automated screenshot + baseline compare system with CI/CD integration.
- **Applitools Eyes**: AI-powered visual diffing with intelligent ignore rules and easy maintenance.

Workflow:

- Test ke har UI screen ya state pe screenshot lete hain.
- Pehle se stored baseline images se comparison karte hain.
- Differences highlight karte hain ek dashboard me.
- Reviewers changes accept/reject karte hain.

3. Detect Broken Images, Links, Missing Text

What?

- Web page me jo images nahi load hote (broken images) ya links invalid ho (dead links).
- Missing important text ya UI components nahi dikh rahe.

Why important hai?

- Broken images aur links user experience kharab karte hain.
- Functional issues lead kar sakte hain (wrong navigation, errors).
- Early detection bugs ko jaldi fix karna allow karta hai.

When use karein?

- Har regression or smoke test me.
- New UI/content deployments ke baad.

Agar na use karo toh?

- Users ko broken UI milega.
- Critical bugs late detect honge.
- Business lose kar sakta hai user trust.

Code Example (Broken Images Detection with Selenium + requests):

Broken Images Detection

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

driver = webdriver.Chrome()
driver.get("https://example.com")

# Step 1: Saare image elements locate karo
images = driver.find_elements(By.TAG_NAME, "img")

# Step 2: Har image ka 'src' attribute leke HTTP request bhejo
for img in images:
    src = img.get_attribute("src")
    if src:
        try:
            response = requests.get(src)
            # Step 3: Agar status code 200 nahi, toh image broken hai
            if response.status_code != 200:
                print(f"Broken image found: {src} - Status Code: {response.status_code}")
        except requests.exceptions.RequestException as e:
            print(f"Error checking image {src}: {e}")

driver.quit()
```

Code Explanation:

- `driver.find_elements(By.TAG_NAME, "img")`: Page ke sare image elements find kiye.
- Har image ka src URL liya `get_attribute("src")` se.
- `requests.get(src)` se us URL pe HTTP request bheja.
- Server se response code 200 hai ya nahi check kiya.
- Code 200 nahi hone pe broken image print kar diya.
- Exceptions ko catch kiya taaki script crash na ho.

Note: Isi logic se links (anchor tags ¡a¿) ke href attribute check kar sakte hain similarly. Agar href URL 404 ya error deta hai toh broken link hai.

4. Accessibility Testing Tools: axe-core, Lighthouse CLI

What?

- Accessibility testing ka matlab hai website ko aise banana jo sab log use kar saken — specially visually impaired, screen-reader users, keyboard users etc.
- **a11y testing** legal requirement hota jaa raha hai (WCAG standards).
- Tools like axe-core (JavaScript library) aur Google Lighthouse CLI automatically accessibility errors detect karte hain.

Why Important?

- Aapka product sab users ke liye accessible ho.
- Legal compliance aur best practices follow karna.
- User base aur reputation badhane ke liye.

When Use Karein?

- Har major release ke pehle.
- Jab bhi front-end UI me changes aaye.
- CI/CD pipeline me automated accessibility tests integrate karna.

Agar Na Karein Toh?

- Accessibility bugs production me rahenge.
- Legal fine and penalty ka risk badhta hai.
- Users base restrict ho jayega, brand damage hoga.

Tools & Usage:

a) axe-core (Deque)

- Javascript based accessibility engine.

- Selenium testing me JS execute kar ke accessibility violations detect karta hai.

Sample Usage (Selenium Python):

axe-core Integration

```
# Selenium me axe-core JS inject kar ke run karna:

driver.execute_script(open("axe.min.js").read())  # axe-core
    library ko page me inject karo
results = driver.execute_script("return axe.run(document)")  #
    axe-core run karo aur results lo
print(results)  # Violations ka summary print karo
```

- Axe-core JS ko web page me insert karte hain.
- axe.run(document) call se accessibility violations list milti hai.
- Aap is output ko parse karke test assertions bana sakte hain.

b) Lighthouse CLI (Google)

- Chrome ka ek command line tool jo in-depth accessibility, performance, SEO audit karta hai.

Run command:

Lighthouse CLI Accessibility Audit

```
lighthouse https://example.com --only-categories=accessibility
```

- Ye report generate karta hai with scores and actionable items.
- Automated or manual use dono kar sakte ho.

Summary Table: Quick Reference

Topic	Why Important	When Use	If Not Used	E T
Visual Screenshot Compare	UI bugs detect kare	Build ke baad, regression	UI bugs slip production	Pl Pe
Visual Regression Testing	Automated UI comparison	Continuous testing, UI changes	Manual testing zyada	Pe
Broken Images/Links	UX aur function failure detect kare	Regression / Smoke tests	User broken UI face kare	Se qu
Accessibility Testing	User coverage aur legal compliance	Major releases, CI/CD	Accessibility issues miss	ax Li

27 Cross-Browser, Parallel, and Remote Testing

1. Cross-Browser Testing (Chrome, Firefox, Edge, Safari)

Why important hai?

- Har user har browser ya device pe tumhare app ko alag-alag dekh raha hota hai.
- Kuch features ya CSS/JS sabhi browsers pe same nahi dikhte — Chrome me chale, Firefox me error aa jaaye, etc.
- Real users ka experience ensure karne ke liye cross-browser testing critical hai.

When use karein?

- Har major release pe, production deploy karne se pehle.
- Responsive, design, authentication jaise features ke liye zaroor.

Agar na karo toh?

- Unknown bugs slip ho jaayenge.
- Real users ka trust lose — production me broken UI/bugs.

Selenium Python Example: Different Browsers pe run karna

Cross-Browser Selenium Example

```python
from selenium import webdriver

# Chrome
driver = webdriver.Chrome()

# Firefox
driver = webdriver.Firefox()

# Edge
driver = webdriver.Edge()

# Safari (SafariDriver pe Mac pe preinstalled, Safari mein "Allow
    Remote Automation" ON karo)
driver = webdriver.Safari()

driver.get("https://example.com")
driver.quit()
```

Har driver se respective browser open hota hai. Tum ek hi script mein browser variable pass karke bhi cross-browser loop bana sakte ho.

2. Parallel Testing — pytest-xdist & multiprocessing

Why?

- Test bohot ho gaye toh sequentially run karoge toh time waste.
- Parallel run se test execution time bohot kam ho jaata hai.

When?

- Large test suite — 10-1000+ cases.
- CI/CD integration, fast feedback.

If not used?

- Test suite slow, developers ko feedback late.
- Deployment speed kam.

pytest-xdist ka use (Parallel run in Selenium tests):

Install karo:

pytest-xdist Installation

```
pip install pytest pytest-xdist
```

Simple usage (4 CPU cores pe parallel run):

Run PyTest Parallel

```
pytest -n 4
```

Tum `-n auto` bhi likh sakte ho — jitne cores system detect kare, utne pe parallel run karega.

Multiprocessing (Python ka inbuilt module) bhi use hota hai, lekin pytest-xdist Selenium-automation me zyada popular/flexible hai.

3. Selenium Grid: Remote, Parallel, and Multi-Browser infra

Why?

- Multiple browsers & OS pe parallel run karna ho — local system pe impossible ho jaata hai.
- Distributed infra ya remote machines use karna ho.

When?

- Cross-team, enterprise scale, remote VM infra, Cloud clusters.
- Lagatar test runs for every commit/PR.

If not used?

- Local resource pe hi stuck rahoge.
- Scalability, coverage less hoga.

Concept: Hub & Node

- Hub: Control center. Test requests idhar aate hain.

- Node: Actual browser/OS wala machine. Jo bhi hub assign kare, woh node execute karta hai.

Basic Setup (Manual) Step-by-Step:

Selenium Grid Manual Setup

```
# Selenium Server jar download karo (selenium.dev)

# Hub start karo:
java -jar selenium-server-<version>.jar hub

# Node join karo:
java -jar selenium-server-<version>.jar node --hub
    http://<hub-ip>:4444
```

Apne Selenium script me remote WebDriver ka config do, browser & desired capabilities set karo.

Docker ke Saath Selenium Grid Setup:

Why Docker?

- Setup ekdum easy, repeatable, scalable.
- 1 command me grid infra create/destroy.

How to:

Docker Grid Setup

```
# Docker install karo.

# Grid network banao:
docker network create grid

# Hub container run karo:
docker run -d -p 4444:4444 --net grid --name selenium-hub
    selenium/hub

# Node (browser) containers:
docker run -d --net grid -e HUB_HOST=selenium-hub
    selenium/node-chrome
docker run -d --net grid -e HUB_HOST=selenium-hub
    selenium/node-firefox
```

Apne test me remote url: `http://localhost:4444/wd/hub` likho driver setup me.

4. Cloud Selenium Grid Platforms

Why cloud?

- Har browser/OS instantly milta hai, no infra headache.
- Tests parallel, scalable ho jaate hain (cloud infra, local nahi).

Popular Platforms:

- **BrowserStack:**
  - Real browsers/devices, instant access.
  - Code se direct cloud drivers pe test run.
  - Advanced debugging — video, logs, screenshots.
  - CI/CD integration easy.
- **Sauce Labs:**
  - Hundreds of browser/OS combos, video recording, advanced reporting.
  - Secure infra, great community support.
  - Parallel, scalable runs.
- **LambdaTest:**
  - 3000+ environments (desktop/mobile).

on, Java, C#, Robot sab support.

  - Local testing tunnel for localhost apps, geolocation, video logs.
  - Parallel runs, analytics, CI/CD integrations.

Sample Python code (BrowserStack Remote WebDriver):

BrowserStack Remote WebDriver Example

```python
from selenium import webdriver

caps = {
    'browser': 'chrome',
    'browser_version': 'latest',
    'os': 'Windows',
    # Add more capabilities per need
}

driver = webdriver.Remote(
    command_executor='https://YOUR_USERNAME:YOUR_ACCESS_KEY@hub-cloud.browserstack.c
    desired_capabilities=caps
)
driver.get("https://example.com")
print(driver.title)
driver.quit()
```

Summary Table

Concept	Why Important	When To Use	Example/Tool
Cross-Browser (local)	Real user experience, bug catch	UI/feature test har build	Selenium: Chrome/-Firefox
Parallel Execution	Time-save, scalable	Badii test suite	pytest-xdist, multiple procs
Selenium Grid	Multi-browser, parallel remote	Team infra, scale needed	Hub+Node infra, Docker
Cloud Selenium	Infra-free, scalable	Release, big team, all OS	BrowserStack, Sauce-Labs

==========

28 Security & Resiliency Checks — Easy Step-by-Step Guide

1. Run UI Tests Behind Login (Authentication Testing)

What Hai Ye?

- Bahut websites aur apps me real important features (jaise My Account, Orders, Payments) login karne ke baad hi dikhte hain.
- "Behind Login" matlab: pehle user login karta hai, phir hi aage ke pages ya actions automate karne hain.

Why Important Hai?

- Agar tum sirf public pages test karoge toh user journey incomplete rahegi.
- Login ke baad aane wala content, role-based access, session expiry jaisi cheezein sirf login ke baad hi test ho sakti hain.
- Security bugs (jaise unauthorized access) sirf login ke baad detect hoti hain.

When Use Karna Hai?

- Har end-to-end (E2E) test ke liye jab user authentication zaroori ho.
- Regression cycles me jab login-protected areas bhi cover karne ho.
- Jab administrator ya user-specific features test karne ho.

Agar Na Kiya Toh Kya Hoga?

- Bahut bugs production me chhut jayenge.
- User experience partial hoga.

- Security vulnerabilities unnoticed rahengi.

Code Example (Line-by-line Explanation):

UI Login Automation

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

driver = webdriver.Chrome()                # Browser launch kar rahe hain (Chrome)
driver.get("https://example.com/login") # Login page open kar rahe hain

username_box = driver.find_element(By.ID, "username")   # Username input field locate kar rahe hain (id="username")
username_box.send_keys("myuser")                        # Username input me "myuser" type kar rahe hain

password_box = driver.find_element(By.ID, "password")   # Password input field locate kar rahe hain (id="password")
password_box.send_keys("mypassword")                    # Password input me "mypassword" type kar rahe hain

login_btn = driver.find_element(By.ID, "loginBtn")      # Login button locate kar rahe hain (id="loginBtn")
login_btn.click()                                        # Login button pe click kar rahe hain (form submit hoga)

# Ab yahan se aap logged-in user actions automate kar sakte hain, jaise order place karna, profile edit karna, etc.

driver.quit()                                           # Browser band karte hain
```

Step-wise logic:

- Browser ko kholda hain aur login page pe jaate hain (driver.get).
- Username aur password ke input boxes dhundte hain aur username/password bhejte hain (send_keys).
- Login button ko dhund kar uspe click karte hain (click).
- Login successful hone par session create ho jaata hai, tab agla test step chal sakta hai.

2. Basic Vulnerability Checks (e.g., XSS Inputs)

What Hai?

- Cross-Site Scripting (XSS) jaise basic vulnerabilities check karna jisme attacker malicious scripts inject karta hai input fields me.
- Automation me commonly use hone wale XSS payloads ko input fields me daalte hain aur check karte hain ki alert box ya script execute ho rahi hai ya nahi.

Why?

- Developers aksar input sanitization nahi karte, isse security loopholes bante hain.
- Automation se cyclic security scans kar ke vulnerabilities jaldi pakad sakte hain.

When Use Karein?

- Jab bhi release aaye, jab bhi forms ya user inputs change ho rahe ho.
- Periodic security audit ke liye CI/CD ke sath.

Agar Na Karein?

- Hackers easily app compromise kar sake hain.
- Bug bounty aur reputation risk badhega.

Code Example with Explanation:

XSS Check Automation

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

xss_payload = "alert('XSS!')"  # Yeh malicious script hai

driver = webdriver.Chrome()
driver.get("https://example.com/contact")       # Contact page open
    karte hain jahan comment box hai

input_box = driver.find_element(By.NAME, "comment") # Comment input
    field locate kar rahe hain
input_box.send_keys(xss_payload)                        # Malicious
    script input me dal rahe hain

input_box.submit()                                      # Form submit
    kar rahe hain (could also click submit btn)

time.sleep(1)    # Thoda rukna padta hai alert box aane ke liye

try:
    alert = driver.switch_to.alert                  # Browser alert
        box pe switch karte hain (agar alert aaya hai)
    print("XSS Vulnerable! Alert text:", alert.text) # Alert text
        print karte hain     matlab vulnerability hai
    alert.accept()                                      # Alert box
        band kar dete hain
except:
    print("XSS Not Detected.")                          # Alert nahi
        aya matlab vulnerability nahi mil rahi hai

driver.quit()
```

Step-wise Logic:

- Malicious script text ko input box me daala.
- Form submit kara, jisse web app ke input processing hui.
- Agar alert box aya, toh matlab XSS vulnerability hai (javascript injection successful).
- Agar alert nahi aya toh app properly sanitize kar raha hai input.

3. Integrate OWASP ZAP with Selenium

What Hai?

- OWASP ZAP open source security scanner tool hai jo Selenium tests ke background me realtime vulnerability scan karta hai.
- Automated pen-testing jaisa kaam karta hai.

Why?

- Manually security testing karna tedious aur inconsistent hota hai.
- Automation ke sath integration se continuous security ensure hota hai.

When Use Karein?

- Har build ke baad security scan automatic karne ho.
- CI/CD pipeline me integrate karna ho.

Agar Na Karein?

- Security checks inconsistent rahe.
- Manual pen-testing costly hota hai.

Basic Setup and Code:

1. Install and run OWASP ZAP locally (default port 8080).
2. Configure Selenium to use ZAP as a Proxy:

Selenium through ZAP Proxy

```python
from selenium import webdriver

zap_proxy = "localhost:8080"                          # ZAP proxy URL and port

chrome_options = webdriver.ChromeOptions()            # Chrome options create kar rahe hain
chrome_options.add_argument(f'--proxy-server=http://{zap_proxy}')  # Proxy ko Chrome me set kiya

driver = webdriver.Chrome(options=chrome_options) # Driver ko options ke sath create kiya

driver.get("https://your-app-url.com")                # App URL open kara (saari traffic ZAP ke through jayegi)

# Selenium usual tests run karo...

driver.quit()
```

Logic:

- Browser setup me ZAP ko proxy bana diya.
- Selenium browser se jo bhi HTTP/S request hogi, wo ZAP ke through jayegi.
- ZAP automatically scan karke vulnerabilities report karega.

4. Secure Credential Management (Using dotenv, Vault)

What Hai?

- Credentials (username/password/API keys) directly code me store karna insecure hota hai.
- Secure way me environment variables ya encrypted vaults ka use karte hain.

Why Important Hai?

- Secrets leak hone se security risk badhta hai.
- Best practice hai ki confidential info codebase se alag rakho.

When Use Karein?

- Har project me jab credentials ya keys use karna ho, especially public repo ya remote environments me.

Agar Na Karo Toh?

- Credentials accidentally expose ho sakti hain.
- Security breach ka high risk hai.

Credential Management with dotenv

1. Install:

Install dotenv

```
pip install python-dotenv
```

2. Create `.env` file in project root:

```
USERNAME=yourusername
PASSWORD=yourpassword
```

3. Use in Python:

Load Credentials in Python

```
from dotenv import load_dotenv
import os

load_dotenv()                          # .env file se variables load
    karo

username = os.getenv("USERNAME")   # Environment se username get karo
password = os.getenv("PASSWORD")   # Password bhi lo

print(f"Username is: {username}")
```

4. Use these variables in Selenium:

Use Secure Vars in Selenium

```
driver.find_element(By.ID, "username").send_keys(username)
driver.find_element(By.ID, "password").send_keys(password)
```

Vault / Secret Manager (Advanced)

- HashiCorp Vault jaise tools encrypted secrets store karte hain.
- API ya CLI ke zariye secrets ko securely fetch karte hain.
- CI/CD pipelines me credentials secure rakhne me help karta hai.

Summary Table

Feature	Why Important	When Use	If Not Used
UI Tests Behind Login	Auth flow test karne ke liye zaroori	E2E, regression tests	Risk of missing re bugs
Basic Vulnerability Checks	Security flaws jaise XSS detect karne	Releases, critical inputs	App vulnerable sakta hai
OWASP ZAP Integration	Automated real-time security testing	Build pipeline, CI/CD	Manual only, incons tent tests
Secure Credential Management	Secrets leak protect karne ke liye	Sab projects jahan keys hain	Leaks & security ri badhta hai

29 Mobile Testing with Appium — Full Guide (Beginner to Industry Ready)

1. Appium Architecture: Kya Hai & Kyun Important?

What?

- Appium ek open-source **Mobile Automation Tool** hai jo **Android, iOS apps (native, hybrid, web)** ko automate karne ke liye use hota hai.
- Yeh **Client-Server Architecture** pe kaam karta hai:
  - Aapka test script (Python, Java, JS, etc.) ek Client hota hai jo commands bhejta hai Appium Server ko (jo Node.js pe run karta hai).
  - Server fir OS-specific drivers ko commands forward karta hai — Android ke liye `UIAutomator2`, iOS ke liye `XCUITest`.
  - Fir ye drivers device/emulator/simulator pe user interactions perform karte hain.

Why?

- Mobile apps ko manual test karna time-consuming aur error-prone hota hai, especially jab baar-baar chhote-chhote changes test karne ho.
- Appium se ek hi test script ko Android aur iOS dono par chala sakte hain, **cross-platform automation** setup ho jaata hai.
- Repetitive regression, smoke, E2E testing automate ho jaati hai jo releases fast banata hai.

When?

- Jab aapko mobile app testing **automate** karni ho.
- Jab regression ya smoke tests bar-bar execute karne ho.
- Jab multi-platform coverage (Android + iOS) chahiye.

- Continuous Integration (CI/CD) ke part me jab automated mobile tests chalaane ho.

If Not Used?

- Manual testing slow aur incomplete hoga.
- Release cycles lambey ho jayenge.
- Bugs easily production me slip ho sakte hain.
- Job market ke liye ye skill na hone se aap competition me peeche rah jaoge.

2. Appium Server Setup — Kaise Start Karen & Kyun?

What?

- Appium server ek bridge hai jo **client commands ko devices/emulators tak pahunchata hai**.
- Server bina chalaye tests device pe nahi chalenge.

Why?

- Server chalu hona zaroori hai, tabhi automation scripts device ke saath communicate kar payengi.

When?

- Har baar Jab automation script chalana ho.
- Jab naya device ya emulator add karein.

If Not Used?

- Scripts connect nahi kar payenge device/emulator se.
- Automation fail ho jayegi.

Step-by-Step Setup

1. **Node.js install karo** (kyunki Appium Node.js pe run karta hai)

Node.js Version Check

```
node -v
npm -v
```

2. **Appium global install karo**:

Appium Install

```
npm install -g appium
```

3. **Appium server start karo**:

Appium Server Start

```
appium
```

- Yeh command default port (`4723`) pe server start kar deta hai.

4. **Android setup**:

- Android SDK / Android Studio install karo.
- `adb devices` run karke connected devices/emulator ka status check karo.
- Env variables set karo (ANDROID_HOME, PATH).

5. **iOS setup (sirf Mac pe)**:

- Xcode / Command line tools install karo.
- `instruments` ya XCUITest driver install karo.
- USB se device connect karo aur trusted karo.

3. Writing Appium Tests in Python — Simple Code + Each Line Explained

Why?

- Python simple aur easy to learn hai.
- Appium official Python client support karta hai.
- Ek hi code Android aur iOS pe chal sakta hai.

When?

- CI/CD ke liye regression runs.
- Daily automation.
- Cross-platform apps ke liye.

If Not Used?

- Manual testing hoga.
- Code duplicate/fixed hoga for each platform.
- Error prone & time-consuming testing.

Step-by-step Example:

Appium Python Example

```python
import unittest
from appium import webdriver
from appium.webdriver.common.appiumby import AppiumBy

# Device or app capabilities define karna hota hai
capabilities = {
    "platformName": "Android",              # Android ya iOS platform specify karo
    "automationName": "uiautomator2",       # Android automation driver
    "deviceName": "Android Emulator",       # Device ya emulator ka naam
    "appPackage": "com.android.settings",   # App ka Android package name
    "appActivity": ".Settings",              # Starting activity of app
    "language": "en",
    "locale": "US"
}

appium_server_url = "http://localhost:4723"

class TestAppium(unittest.TestCase):
    def setUp(self):
        # Appium server ko connect karke session create karta hai driver object
        self.driver = webdriver.Remote(appium_server_url, capabilities)

    def tearDown(self):
        # Test end me session close kar deta hai
        if self.driver:
            self.driver.quit()

    def test_find_battery(self):
        # Battery option locate karo through XPath aur click karo
        el = self.driver.find_element(by=AppiumBy.XPATH, value='//*[@text="Battery"]')
        el.click()

if __name__ == "__main__":
    unittest.main()
```

Line-by-Line Explanation:

- `capabilities`: Ye dict tell karta hai ki kaunsa platform, device, aur app test hoga.
- `webdriver.Remote`: Appium server se session connect karta hai.
- `setUp()`: Har test case se pehle browser/session init hota hai.
- `tearDown()`: Test ke baad session close hota hai.

- `find_element(...)`: Mobile app screen ke UI element ko locate karta hai.
- `click()`: Element pe tap karta hai.
- `unittest.main()`: Python test runner hota hai.

4. Emulator vs. Real Device — Kya Farak Hai?

Parameter	Emulator	Real Device
Cost	Free, software based	Expensive, need to buy/-manage hardware
Accuracy	Almost real but kuch hard-ware features missing	Real world hardware expe-rience
Maintenance	Easy to reset and configure	Device firmware updates, battery etc.
Parallelism	Easy multiple emulators run kar sakte hain	Hardware ki limitation hoti hai
Hardware Features	Limited (sensor emulation, camera limited)	Full hardware access (GPS, camera)

Why do we need both?

- **Emulator** fast dev/testing ke liye.
- **Real Device** real user conditions ke liye.

When to use?

- Development aur early regression me emulator.
- Final testing, bug reproduction, real feedback ke liye real devices.

Agar na use karein?

- Sirf emulator: Real world ke bugs miss kar sakte hain.
- Sirf real device: Costly aur testing slow ho sakti hai.

5. Cross-Platform Locators (Android / iOS)

Why?

- Android aur iOS app UI structure alag hota hai.
- Locator strategy bhi platform wise alag ho sakti hai isliye common approach chahiye jo dono me kaam kare.
- Locators reliable hone chahiye taaki test stable rahe.

When?

- Jab single automation framework dono platforms pe run kar raha ho.

- Jab cross-platform code reuse chahiye.

Agar na use karein?

- Duplicate code likhna padega har platform ke liye.
- Maintenance zyada hoga, cost badhegi.

Common Locator Types

Locator Type	Explanation	Code Example
Accessibility ID	Cross-platform best locator, content-desc/iOS label	`driver.find_element(AppiumBy.A`
ID/Resource-ID (Android)	Android-specific unique id	`driver.find_element(AppiumBy.I`
Name (iOS)	iOS element name / label attribute	`driver.find_element(AppiumBy.N`
Class Name	UI element's class type, platform agnostic	`driver.find_element(AppiumBy.C`
XPath	Flexible but slow, last option	`driver.find_element(AppiumBy.X`

Notes on Flutter / React Native / Kotlin Apps

- **Flutter Apps:**
  - Traditional locators work kum ya nahi karte.
  - Flutter specific plugin `Flutter Driver` use hota hai, jo Flutter widgets ko identify karta hai.
  - Appium me bhi Flutter ke custom locators available hain (`flutter finders`), but thoda complex hota hai.
- **React Native Apps:**
  - Mostly Android/iOS jaisa hi hota hai.
  - Developers accessibility labels pe dhyan dete hain, taaki app UI testable rahe.
  - Accessibility IDs ya content-desc use karna best practice hote hain.
- **Kotlin/Java Android Apps:**
  - Usually resource IDs, content-desc, XPath sab normal hi use karte hain.

Summary Table

Feature	Why Important	When to Use	If Not Used
Appium Architecture	Multi-platform support, client-server	Mobile automation run	Manual testing o tial
Appium Server	Connect script & devices	Every test run	Scripts fail to co
Python Appium Tests	Easy scripting and code reuse	E2E, regression, CI	Manual or fragm code
Emulator vs Real Device	Fast dev & real UX validation	Dev vs final testing	Bugs miss or testing
Cross-Platform Locators	One code multiple platforms	Cross-platform projects	Duplicate pla code
Special Framework Apps	Flutter, React Native need special locators	Mobile frameworks	Locator failures, tests

==========

[a4paper,12pt]article [margin=1in]geometry xcolor sectsty parskip tcolorbox listings,skins listings courier enumitem hyperref pifont

Continuous Integration & Delivery (CI/CD)

30 Git Integration: Branching & Version Control for Test Scripts

What?

Git ek version control system hai jisme aap apne test automation code ko manage karte ho — history maintain karte ho, different versions banao, changes track karte ho.

Why Important?

- Team me ek saath kai log codes pe kaam kar sakte hain without conflict.
- Agar galti se kuch big bug aa jaye toh purani stable version pe revert kar sakte hain.
- Code changes traceable hote hain, bugs ya issues jaldi identify hote hain.

When Use?

- Jab multiple testers/devs ho.
- Jab code regularly update hota ho.
- Jab automation ko production ke sath sync rakhna ho.

If Not Used?

- Code duplicates increase honge.
- Manual merging errors honge.
- Collaboration mushkil hoga.

31 Jenkins

What?

Jenkins ek popular open-source **Continuous Integration (CI)** tool hai jo automation run karne, code pull karne, tests chalane, reports generate karne, schedule karne ke liye use hota hai.

Why Important?

- Tests ko automated banata hai, baar baar bina manual chalaye.
- Developers ko continuous feedback milta hai (pass/fail).
- Pipelines define karke process automatic banata hai.

When Use?

- Medium ya bade projects jahan automation bar bar chahiye.
- Jenkins server configured ho.
- Multiple tools integrate karne ho (SonarQube, Slack, Emails).

If Not Used?

- Manual testing/process slow hoga.
- Test result feedback late milega.
- Human error badhega.

Jenkins Pipeline Declarative Example

Listing 1: Jenkinsfile Example - Pipeline

```
pipeline {
    agent any

    stages {
        stage('Checkout') {
            steps {
                git branch: 'main', url: 'https://github.com/yourrepo.git'
            }
        }
        stage('Setup') {
            steps {
```

```
                sh 'python -m venv venv'
                sh '. venv/bin/activate && pip install -r requirements.txt'
            }
        }
        stage('Test') {
            steps {
                sh '. venv/bin/activate && pytest --html=report.html --self
                    -contained-html'
            }
            post {
                always {
                    publishHTML([
                        allowMissing: false,
                        alwaysLinkToLastBuild: true,
                        keepAll: true,
                        reportDir: '.',
                        reportFiles: 'report.html',
                        reportName: 'PyTest Report'
                    ])
                }
            }
        }
    }

    post {
        always {
            archiveArtifacts artifacts: 'report.html', fingerprint: true
        }
    }
}
```

Explanation:

- `pipeline { agent any }`: Pipeline kisi bhi available node par run hoga.
- `Checkout`: Git se code pull ho raha hai.
- `Setup`: Virtual env banake dependencies install ho rahe hain.
- `Test`: Pytest report generate karke publish kar raha hai.
- `post`: Report archive ho rahi hai.

32 Schedule Jobs

Jenkins me "Build periodically" feature se cron syntax specify kar sakte ho, jaise `0 0 * * *` (har raat 12 baje).

33 HTML Reports & Screenshots

- `pytest-html` se detailed reports banti hain.
- Jenkins me `publishHTML` plugin se integrate hoti hain.

- Failures par screenshot bhi reports me attach kar sakte hain.
- Logs Jenkins console me realtime dikhte hain.

34 GitHub Actions

What?

GitHub Actions ek cloud-based CI/CD service hai, GitHub ke liye, jisme automation workflows likhe jate hain YAML me.

Why?

- Fully cloud hosted, zero infra maintenance.
- GitHub repo se direct integrate hota hai.
- YAML workflows se flexible automation.
- Multiplatform runners with parallelism.

When?

- Repo GitHub pe ho.
- Cloud based CI chahiye.
- Multiplatform testing.

If Not Used?

- Infra khud maintain karna hoga.
- Scale aur speed limited ho sakti hai.
- Cloud benefits miss honge.

GitHub Actions YAML

Listing 2: GitHub Actions Workflow

```yaml
name: Python CI

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v3
    - name: Setup Python
```

```
      uses: actions/setup-python@v4
      with:
        python-version: 3.10
    - name: Install dependencies
      run: |
        python -m venv venv
        source venv/bin/activate
        pip install -r requirements.txt
    - name: Run Tests
      run: |
        source venv/bin/activate
        pytest --html=report.html --self-contained-html
    - name: Upload Report
      uses: actions/upload-artifact@v3
      with:
        name: test-report
        path: report.html
```

Explanation:

- Triggers on push and PR to main branch.
- Runs on Ubuntu latest.
- Checks out code, sets python, installs deps.
- Runs tests and generates HTML report.
- Uploads report as artifact.

35 Trigger on Push / PR / Merge

CI runs test automatically on code push or PR events to catch errors early.

36 Slack / Email Notifications

Slack Setup in Jenkins

- Install Slack plugin.
- Create Slack app, get webhook URL.
- Configure URL in Jenkins settings.

```
post {
    failure {
        slackSend(color: 'danger', message: "Build failed: ${env.JOB_NAME}
            ${env.BUILD_NUMBER}")
    }
    success {
        slackSend(color: 'good', message: "Build success: ${env.JOB_NAME} $
            {env.BUILD_NUMBER}")
    }
}
```

Slack Setup in GitHub Actions

```yaml
- name: Slack Notification
  uses: 8398a7/action-slack@v3
  with:
    status: ${{ job.status }}
    fields: repo, commit, author
  env:
    SLACK_WEBHOOK_URL: ${{ secrets.SLACK_WEBHOOK_URL }}
```

Email Notifications

Jenkins Email-ext plugin se email configure kar sakte hain. Customize templates and notify team instantly on failures.

Summary Table

Feature	Why Important	When Use	Example / Tool
Git Integration	Collaboration, version control	Team projects	Git
Jenkins	Automated CI, pipelines	Medium to large projects	Jenkins
Scheduled Jobs	Regular runs, regression	Nightly / periodic jobs	Jenkins cron
HTML Reports	Friendly, detailed results	Reporting	pytest-html, Jenkins plugins
GitHub Actions	Cloud CI/CD, easy setup	GitHub hosted repos	GitHub Actions
Notifications	Faster feedback	Failures and successes	Slack, Email

37 Real-World End-to-End Projects in Selenium with Python

Project 1: E-Commerce Website Testing

Features covered:

- User login
- Product search & filtering
- Add products to cart
- Checkout process (payment simulation)
- Data-driven order placements

- Reports generation for orders & test results

Why important?

- E-commerce sites real-world me bahut complex hote hain.
- End-to-end (E2E) tests user journey ka real scenario replicate karte hain.
- Bugs yahan destination tak pahuchte hain, toh testing strict honi chahiye.

When use karo?

- Release se pehle full regression karni ho.
- Payment, cart functionality regularly check karni ho.
- Multi-user, multi-data scenario validate karna ho.

Basic Approach & Examples

POM Design Har page ko alag class banayenge:

Page	**Features / Actions**
LoginPage	`enter_username()`, `enter_password()`, `click_login()`
HomePage	`search_product()`, `apply_filter()`, `open_product()`
ProductPage	`add_to_cart()`, `select_quantity()`
CartPage	`verify_cart_items()`, `proceed_to_checkout()`
CheckoutPage	`fill_address_details()`, `select_payment()`, `place_order()`
OrderConfirmationPage	`verify_order_success()`, `get_order_id()`

Data-Driven Orders

- Excel/CSV me products details, user info ready rakhoge.
- `pytest.mark.parametrize` ya external data provider se data fetch karke test chalaoge.

Sample Code Snippet (LoginPage as Example)

LoginPage Example

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

class LoginPage:
    def __init__(self, driver):
        self.driver = driver
        self.username_input = (By.ID, "username")
        self.password_input = (By.ID, "password")
        self.login_button = (By.ID, "loginBtn")

    def enter_username(self, username):
        self.driver.find_element(*self.username_input).send_keys(username)

    def enter_password(self, password):
        self.driver.find_element(*self.password_input).send_keys(password)

    def click_login(self):
        self.driver.find_element(*self.login_button).click()
```

Test Example with Data-Driven

Data-Driven Login Test

```python
import pytest

@pytest.mark.parametrize("username, password", [
    ("user1", "pass1"),
    ("user2", "pass2")
])
def test_login(driver, username, password):
    login_page = LoginPage(driver)
    login_page.enter_username(username)
    login_page.enter_password(password)
    login_page.click_login()
    # Assert dashboard visible, etc
```

CI/CD Integration

- Jenkins/GitHub Actions me run karoge.
- PyTest-html ya Allure se report generate.
- Jenkins me HTML report publish karo aur Slack notify karo.

Project 2: HR Portal or CMS Testing

Features covered:

- Form submissions (employee registration, leave requests)
- File uploads (CV uploads, document attachments)
- Approval workflows (manager approval buttons, status changes)

Why important?

- Business-critical workflows hain jo manually bohot time lete hain.
- Automated testing se error-free, fast workflow validation.

When use karo?

- Release ke baad ya changes ke baad validate karo.
- Automation regression part banana ho.

Automation Approach

POM Classes

Page	Actions
LoginPage	Same as Project 1
EmployeeFormPage	`fill_form(data)`, `upload_documents(path)`
ApprovalPage	`approve_request()`, `reject_request()`
DashboardPage	`check_request_status()`, `filter_requests()`

File Upload Example

File Upload Method

```
def upload_document(self, file_path):
    upload_element = self.driver.find_element(By.ID, "fileUpl
    upload_element.send_keys(file_path)  # File path absolute
        chahiye
```

Additional Notes:

- Waits and synchronization important hai – workflow me buttons/dynamic elements pe action karne se pehle visible and clickable hone ka wait lagao.
- Report generation and logs – process ke steps ka logging karna useful hota hai workflows me.

Project 3: Regression Test Suite

Features Covered:

- Comprehensive test suite for core features
- Built using POM + PyTest for modularity and maintainability
- Reports (pytest-html/allure) and CI/CD integration for continuous testing

Why important?

- Regression testing every build me mandatory hai taaki naye changes purane features break na karein.
- Automated and maintainable test suite efficient delivery ke liye zaroori.

Structure & Best Practices

```
project_root/
 pages/                # POM classes
     login_page.py
     home_page.py
     cart_page.py
 tests/                # PyTest test cases
     test_login.py
     test_cart.py
     test_checkout.py
 utils/                # Waits, logging, reports helpers
 data/                 # Excel/CSV/JSON test data files
 reports/              # Generated reports folder
 conftest.py           # PyTest fixtures (setup/teardown)
 requirements.txt
 pytest.ini            # PyTest configurations
```

Running All Tests

Run Entire Suite

```
pytest --html=reports/report.html --self-contained-html
```

CI/CD Pipeline

- GitHub Actions/Jenkins to pull, setup env, install dependencies, run tests and publish reports.
- Slack/Email notifications configured for test status alerts.

Code Reusability

- BasePage and Utility classes for common actions (waiting for elements, logging, screenshot on failure) use karo.
- Fixtures me driver setup/teardown properly handle karo.

Final Tips

- Real-world projects me modularity, reusability, maintainability sabse important hai.
- Test data ko alag rakho (Excel/CSV/JSON).
- Logs/reports pe focus karo for easy debugging.
- CI/CD integration nahi bhoolo — automation ka future wahi hai!

==========

[a4paper,12pt]article [margin=1in]geometry xcolor sectsty parskip tcolorbox listings,skins listings courier enumitem amsmath hyperref pifont

Advanced & Job-Oriented Topics in Automation Testing

1. Test Tagging & Suite Management

Why Important?

- Bahut saare tests hote hain, sabko ek saath chalana efficient nahi hota.
- Tagging se tests categorize kar sakte hain (smoke, regression, sanity, critical etc).
- Easily required tests ko run karo based on tag without changing code.

When to Use?

- Jab test suite bada ho.
- CI/CD pipelines mein selective test execution ke liye.

If Not Used?

- Testing slow ho jata hai.
- Troubleshooting mushkil hota hai.

Pytest Tagging Example

```
import pytest

@pytest.mark.smoke
def test_login():
    assert True

@pytest.mark.regression
def test_checkout():
    assert True
```

Run smoke tests only:

```
pytest -m smoke
```

2. Test Retry Mechanism

Why Important?

- Kabhi kabhi flaky tests network/dynamic content ki wajah se fail hotey hain.
- Retry se flaky failures reduce hote hain aur stable results milte hain.

When to Use?

- Flaky tests ya unstable environments me.

If Not Used?

- CI/CD me false negatives aate hain.
- Time waste hota hai failures ko repeat karne me.

Pytest Retry Plugin

```
pip install pytest-rerunfailures

pytest --reruns 2
```

Explanation: Failed test ko max 2 baar automatically retry karega.

3. Flaky Test Resolution

Why Important?

- Flaky tests confidence kam karte hain; automation ka fayda kam hota hai.
- Root cause identify karna zaroori hai.

When to Use?

- Jab test results unpredictable ho.

If Not Used?

- Developers ignore karte hain flaky failures ko.
- Bugs slip ho sakte hain into production.

Common Strategies:

- Use explicit waits (‘WebDriverWait‘) instead of static sleeps.
- Use stable locators avoid karna chahiye dynamic XPaths.
- Test isolation and data parameterization.
- Manage parallel execution carefully.
- Use retries cautiously.

4. Locator Auto-Healing Techniques

Why Important?

- UI changes hone pe hardcoded locators break ho jate hain.
- Auto-healing se test khud alternate locators try karta hai.

When to Use?

- Large / dynamic UI projects jahaan frequent UI changes hote hain.

If Not Used?

- Tests frequently fail honge.
- Maintenance zyada ho jayega.

Common Approaches:

- Store multiple locators for an element with fallback logic.
- Use AI-powered tools (Mabl, Testim).
- Wrap element finding in custom functions.

Basic Auto-Healing Function

```
def find_element(driver, locators):
    for locator in locators:
        try:
            element = driver.find_element(*locator)
            return element
        except:
            continue
    raise Exception("Element not found using any locator")

# Usage:
locators = [
    (By.ID, "submitBtn"),
    (By.XPATH, "//button[text()='Submit']"),
    (By.CSS_SELECTOR, "button.submit")
]

element = find_element(driver, locators)
element.click()
```

5. Code Reviews & Pull Requests

Why Important?

- Code quality improve hoti hai.
- Bugs early detect hote hain.
- Team knowledge sharing badhata hai.

When to Use?

- Har commit ke liye, specially large teams / projects me.

If Not Used?

- Bad coding practices spread ho sakti hain.
- Code duplication aur inconsistencies badhen.

Best Practices:

- Regular peer review karo.
- Code styling (PEP8) maintain karo.
- Meaningful comments aur docs likho.
- Unit integration tests include karo.

6. Creating Reusable Libraries

Why Important?

- Code reuse badhata hai.
- DRY principle follow hota hai.
- Maintenance easy hota hai.

When to Use?

- Jab multiple tests me common functionalities ho.

If Not Used?

- Code duplication,
- Changes multiple jagah karne ki need.

Utility Function Example

```
def login(driver, username, password):
    driver.find_element(By.ID, "username").send_keys(username)
    driver.find_element(By.ID, "password").send_keys(password)
    driver.find_element(By.ID, "loginBtn").click()
```

Use is function ko multiple scripts me import kar ke.

7. Test Strategy Document Preparation

Why Important?

- Stakeholders ko roadmap deta hai.
- Scope, objectives, resources define karta hai.
- Quality metrics aur risk management specify karta hai.

When to Use?

- Project start me.
- Client approval ke liye.

If Not Used?

- Confusion in scope and expectations.
- Poor resource/time management.

Typical Sections:

- Scope & Objectives
- Tools & Frameworks
- Test Types & Coverage
- Test Data & Environments
- Deliverables & Reporting
- Risks & Mitigation
- Timeline & Milestones

8. Writing Test Cases for Automation Coverage

Why Important?

- Without well-written test cases automation adhura hai.
- Coverage ensures important scenarios are tested.

When to Use?

- Before and during automation implementation.

If Not Used?

- Important test scenarios miss ho jayengi.
- Coverage poor; bugs undetected rahenge.

Tips:

- Clear objectives likho.
- Input data specify karo.
- Expected outcome clearly mention karo.
- Preconditions & postconditions include karo.
- Make test cases reusable and parameterized.

Test Case ID	TC001
Description	Verify login with valid user
Steps	1. Open Login Page 2. Enter valid username 3. Enter valid password 4. Click Login
Expected Result	User should be redirected to Dashboard Page

Summary Table: Advanced Topics

Topic	Why Important	When To Use	Typical Tool
Test Tagging & Suite Management	Efficient selective execution	For large suites and CI/CD	Pytest mar
Test Retry Mechanism	Reduce flaky failures	For unstable tests	pytest-reru
Flaky Test Resolution	Stability and confidence	For flaky	Better loca tion
Locator Auto-Healing	Reduce test breaks due to UI changes	Frequent UI churn	Multiple lo
Code Reviews & PRs	Code quality, bug catching	Every commit	Peer review
Reusable Libraries	Speed, maintainability	Multiple test reuse	Utility func
Test Strategy Document	Roadmap clarity	Project start	Word doc,
Writing Test Cases	Coverage	Before/during automation	Clear steps

==========

[a4paper,12pt]article [margin=1in]geometry xcolor sectsty parskip tcolorbox listings,skins listings courier enumitem hyperref pifont

Job Interview Preparation for Selenium + Python Automation Roles

38 Selenium & Python Theory Questions

Why? Theory se pata chalta hai tumhari basic understanding kitni strong hai. Interview me basic questions zarur poochte hain to assess foundation.

Common Topics and Sample Questions

Topic	Sample Questions	Tips & Important Points
Selenium WebDriver Basics	*WebDriver kya hai? WebDriver vs Selenium RC?*	WebDriver new, RC deprecated; WebDriver browser control karta hai.
Locators	*XPath vs CSS selectors, dynamic locators kaise handle karte?*	Always prefer stable and fast locators. Use `contains()`, `starts-with()` for dynamic elements.
Waits	*Implicit vs Explicit waits, fluent wait kya hota hai?*	Explicit wait most reliable, avoid mixing implicit and explicit.
Selenium Architecture	*Selenium suite ke components (IDE, WebDriver, Grid)?*	Grid for parallel & remote; IDE mostly deprecated.
Python Scripting Basics	*List, dict, classes, decorators basics*	Useful for writing clean automation code.
Exception Handling	*Selenium ki common exceptions (NoSuchElement, Timeout)?*	Use try-except blocks, explicit waits preempt errors.
Handling Popups/Frames	*Alert handling, iframe switch kaise karte?*	`switch_to.alert`, `switch_to.frame` usage.

39 Debugging & Optimization Questions

Why? Debugging skill test hoti hai ki tum errors kaise trace karte ho, flaky tests ko kaise fix karte ho.

Common Scenarios and Approaches

Problem	Approach to Debug & Optimize
ElementNotVisible or StaleElementRef	Use explicit waits, avoid caching elements for long time
Flaky Tests	Add retries, use fluent waits, stabilize locators
Slow Test Execution	Parallel execution (pytest-xdist), selective tests run
Memory leaks/Driver leaks	Proper `driver.quit()`, avoid multiple driver instances

40 Practical Scenarios

Locators

Example Question: Agar xpath dynamic hai, handle kaise karoge?
Best Answer: Use `contains()`, `starts-with()`, or better stable attributes like accessibility-id, aria-labels.

Waits

Example: Implicit vs explicit wait kab use karoge? Agar dono mix kiye toh kya hoga?
Answer: Explicit waits targeted elements ke liye sabse best hain. Dono mix karna risk hai flaky tests ka.

POM Questions

Example: POM kya hai? Advantages kya hain? Kaise implement karte ho?
Answer: POM ek design pattern hai jisme har page ka ek class banta hai with locators & actions, jis se reusability & maintainability badhta hai.

41 Git, Jenkins, Docker, API Knowledge for Automation Roles

Git

- Branching basics, merging, resolving conflicts.
- Using branch strategies (feature, develop, main).
- Setup `.gitignore` for test artifacts.

Jenkins

- Pipeline setup basics.
- Scheduling jobs and triggering on Git commit.
- Archiving reports and sending notifications.

Docker

- Containerize automation environment.
- Use Selenium Grid with Docker for scalable parallel test execution.

REST API Basics

- Testing APIs using `requests` module.
- Validate status codes, response contents in tests.
- Integrate API calls into automation flows.

42 System Design for Automation Frameworks

Important Concepts

- Modular, scalable architecture (POM + Utilities + Data Driven + Reports).
- Parallelism & grid/distributed testing.
- CI/CD integration pipeline.
- Test data source management.
- Error handling & retry mechanisms.

Prepare to Explain

- Folder structure.
- How fixtures manage setup/teardown.
- How test suites are grouped and executed selectively.
- Reporting and logging approach.

43 Assignment-Level Projects from Companies

Common Requirements

- Login + CRUD operations automation.
- Data-driven scenarios with multiple datasets.
- Reporting + screenshots + CI pipeline.
- Clean, commented, maintainable code.

Best Practice

- Use POM + PyTest.
- Write meaningful asserts and error messages.
- Handle dynamic elements patiently.
- Automate environment setup using Docker if possible.

44 Bonus Tools to Learn (Job-Ready Plus)

Tool	Purpose	Why Important
requests	REST API testing	API validation integral part of modern automation
openpyxl, pandas	Excel automation	Data-driven testing, test data handling
faker	Test data generation	Generate realistic random test data
pytest-xdist	Parallel test execution	Speed up test runs
allure-pytest	Beautiful test reports	Interactive, shareable reports
dotenv	Secure credentials storage	Avoid hardcoding secrets
SelectorHub, ChroPath	XPath/CSS locator generation	Fast, error-free locator creation
Selenoid, Zalenium	Scalable Grid + video recording	Parallel & remote cross-browser testing
Docker	Containerized test environments	Consistent environment setup

You Are Job-Ready If You Can:

- Selenium WebDriver APIs deeply samjho, confidently use karo.
- POM + PyTest framework effectively banake run kar sakte ho.
- Dynamic elements, waits, popups, frames handle properly karte ho.
- Readable, maintainable aur reusable test code likh sakte ho.
- Cross-browser tests automate karke CI/CD pipeline me integrate kar chuke ho.
- Automation reports generate kar sakte ho (pytest-html, allure).
- Flaky tests ko identify aur fix karte ho.
- Real data se test karte ho aur multiple user journeys automate karte ho.

Extra Tips for Interview Preparation

- Real-life project examples ready rakho (GitHub repo links preferred).
- Code ko samjho, har function ka kya kaam hai clarity rakho.
- Python basics (data types, functions, OOP) thoda revise karo.
- Practice debugging with real Selenium errors.
- Mock interview questions focus on scenario-based answers.

[a4paper,12pt]article [margin=1in]geometry xcolor sectsty parskip tcolorbox listings,skins listings courier enumitem amsmath hyperref pifont

Most Helpful Plugins & Tools for Selenium + Python Automation

45 SelectorHub (Browser XPath/CSS Tool)

What?

- SelectorHub ek Chrome/Firefox extension hai jo **XPath aur CSS selectors** automatically aur accurately generate karta hai for any web element[1].

Why use karein?

- XPath/CSS locator banana manual process mein galtiyan hoti hain – especially dynamic, lengthy, ya complex DOM structure me. - SelectorHub real-time valid selectors suggest karta hai, aur verify bhi karta hai ki selector unique hai ya nahi.

When to use karein? - Jab new locator banana ho, ya dynamic element ke liye alternate locator chahiye ho. - Jab manually XPath likhna error-prone ho raha ho.

If not used toh kya problem? - Locators galat ho sakte hain, automation script frequently break hogi. - Development/debugging time badh jayega.

How to Install/Use:

1. Chrome/Edge/Firefox browser mein jao, **SelectorHub extension** search karo.
2. Install karlo (official store, trusted source only).
3. Kisi bhi web page pe jao, right click → Inspect (DevTools open).
4. SelectorHub tab dikh raha ho to use open karo. Koi web element select karo → aapko recommended XPath/CSS mil jayega.
5. Copy-paste selector code in your Selenium scripts.

46 ChroPath (XPath/CSS Finder)

What? - ChroPath ek aur popular Chrome extension hai for generating and validating XPath, CSS Selector, Absolute XPath, etc.

Why use karein? - Difficult-to-locate elements me instant/helpful selector bana deta hai. - Auto-suggests selectors and allows one-click copy.

When to use karein? - Jab complex forms, tables, popups, iframes handle kar rahe ho.

If not used toh? - Galat selector se flaky/unreliable tests ban jayenge.

How to Install/Use:

- Similar to SelectorHub: Extension install karo, DOM pe element select karo, recommended locator copy karo.

47 Axe DevTools (Accessibility Testing Plugin)

What? - Chrome/Firefox extension for **automated accessibility audits** using axe-core engine.

Why use karein? - WCAG/ARIA/accessibility guidelines ki violations easily detect ho jaati hain—manual a11y testing tedious ho sakta hai.

When to use karein? - Jab aapko a11y compliance (legal ya usability reason se) ensure karna ho.

If not used toh? - Visually impaired users ko app accessible nahi rahega; legal/compliance risks ho sakte hain.

How to Install/Use:

1. Install Axe DevTools extension.
2. Target web page open karo, DevTools → Axe tab pe click karo.
3. "Analyze" button dabao—report mil jayega.

48 Testim or Mabl (AI-Based Locator Auto-Healing Tools - Advanced/Optional)

What? - Cloud-based tools jo **AI se locator break hone pe auto-heal/auto-suggest** alternative selectors dete hain and reduce script maintenance. - Selenium/Python ecosystem me optional, advanced team ke liye.

Why use karein? - UI change hone pe scripts bar-bar break hoti hain; auto-healing maintainability improve karta hai.

When to use? - Large scale projects, rapidly changing frontends.

If not used? - Manual locator update/maintenance frequently required.

How to Use: - Testim/Mabl ka trial setup karo, Selenium integration steps follow karo. AI-based healing tabhi useful hai jab UI fragile ho.

49 Allure Commandline & Pytest-HTML (Reporting Plugins)

What? - **Allure**: Rich, beautiful reporting dashboard. - **pytest-html**: Simple HTML report plugin for pytest.

Why use karein? - Pass/fail, screenshots, logs, steps—all kuch visual format me share kar sakte ho team/clients ke saath.

When to use? - Automated reporting after every run, CI/CD integration.

If not used toh? - Output sirf console me yaa low-detail HTML; traceability poor; sharing difficult.

How to Install/Use:

- `pip install allure-pytest pytest-html`
- Pytest run karte waqt: `pytest --html=report.html --self-contained-html` `pytest --alluredir=allure-results` (then Allure serve command)
- For Allure dashboard, Java Allure do install and use `allure serve allure-results`.

50 Requests Interceptor (Network API Monitoring Plugin)

What? - Chrome extension jo **network requests capture, modify, analyze** karne me madad karta hai.

Why/When to use? - UI + API combined automation ya API endpoint debugging ke liye helpful hai. - Network failures, broken API ya security vulnerability quickly detect karte hain.

How to Use: - Install "Request Interceptor" extension, page reload karo, all HTTP(s) requests analyze karo.

51 Fake Data Generator Plugins (e.g., Fake Filler)

What? - **Fake Filler** Chrome extension: Instantly form fields me random but realistic data fill karta hai.

Why? - Data-driven test aur regression me form input fill karne ka time bachaata hai.

If not used? - Test dev slow, manual typing repetitive actions.

52 Other Useful Tools Quick List

Tool/Plugin	What/Why/When	Install/Use Steps
Postman	API endpoint rapid testing & automation	Download app, import/export collec
BrowserStack Local	Cloud device/browser testing	Install CLI agent, connect with Sele
Jenkins GitHub Plugins	CI/CD triggers, notifications, status checks	Jenkins manage plugins, GitHub Ac setup
Python dotenv	Secure credentials/env variable manage	`pip install python-dotenv`, `.env` files
Visual Studio Code Plugins	Python, Selenium, Pylance, Allure Test Reporters	Install via VSCode marketplace

53 FAQs / Common Interview Tasks

- **SelectorHub ya ChroPath interview me kaise explain karo?**

Bol sakte ho — "I use SelectorHub/ChroPath for robust XPath/CSS creation and validation. It saves time and reduces flaky test script failures by generating accurate locators, especially in dynamic web apps."

- **Allure/pytest-html reporting zaruri hai?**

  Industry me test traceability, CI/CD integration and result sharing ke liye reporting plugin essential hai.

- **Accessibility plugin a11y compliance ke liye kaise integrate karo?**

  Manual and automated audit dono aap Axe DevTools/axe-core JS based automation se integrate kar sakte ho.

54 Final Golden Rules

- Har automation setup me **minimum SelectorHub/ChroPath, Allure/pytest-html, and Axe DevTools** install rakho.
- Plugins/Extensions properly trusted source se install karo.
- Plugins sirf debugging/locator/rapid form filling/reports/analysis ke liye hi use karo—core test logic hamesha apne codebase me likho.

==========

[a4paper,12pt]article [margin=1in]geometry xcolor sectsty parskip tcolorbox listings,skins listings courier enumitem amsmath hyperref pifont

Web & Mobile Element Locators in Selenium & Appium: Beginner to Job-Ready Guide

55 1. Web Element Locators in Selenium (For Beginners)

A. Basic Types of Locators

- **ID:** Unique page element identity.
- **Name:** Form fields mein jyada use hota hai.
- **Class Name:** CSS class values (multiple elements ho sakte hain).
- **Tag Name:** E.g., input, div, button.
- **Link Text:** Anchor (<a>) text.
- **Partial Link Text:** Anchor text ka partial match.
- **XPath:** Powerful, custom locator (even complex elements).
- **CSS Selector:** CSS-based matching, fast & flexible.

B. Locators Collect Karne Ka Step by Step Tarika (Chrome Example)

1. Manual Inspection (Without Plugin)

- **Website kholo** (target page).
- **Right click element** → "Inspect" (Chrome DevTools open hoga).
- DevTools mein highlighted line pe hover karo:
  - **ID** present hai? E.g., `<button id="submit">` → Use `By.ID`
  - **Class**? E.g., `<button class="main-btn">` → Use `By.CLASS_NAME` (agar class unique ho)
  - **Name**? E.g., `<input name="user">`
  - **Tag**? E.g., use `By.TAG_NAME('button')`
  - **XPath/CSS**: Right click element in DevTools → "Copy" → Select "Copy XPath" or "Copy selector"
- **Paste karo value apne Selenium script me**.

Python Example:

Basic Selenium Locators Example

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

driver = webdriver.Chrome()
driver.get("https://example.com")

element = driver.find_element(By.ID, "submit")              # For ID
element = driver.find_element(By.CLASS_NAME, "main-btn")    # For class name
element = driver.find_element(By.XPATH, "//button[@id='submit']")   # For XPath
element = driver.find_element(By.CSS_SELECTOR, ".main-btn")         # For CSS Selector
```

C. Auto-Generate Locators Using Plugins (Recommended for Beginners)

1. SelectorHub Extension (Best Plugin)

- Chrome/Edge/Firefox webstore pe jao, search "SelectorHub".
- **Install** karo extension.
- Target page kholo, **right click** → "Inspect".
- **SelectorHub tab select karo**, ab koi web element pe click ya hover karo.

- Tumhe **auto-generated XPath, CSS Selector** aur unka validity status milega.
- **Copy karo** directly from SelectorHub panel and use in Selenium code.

2. ChroPath (Alternative Plugin)

- Same process as above.
- "ChroPath" bhi install kar sakte ho from Chrome Web Store.
- DevTools me ChroPath tab pe element select karte hi locator mil jayega.

3. Fake Filler (For Form Testing)

- Chrome/Firefox extension "Fake Filler" install karo.
- Login/signup form kholke, extension click karo—**form sare fields auto-fill** ho jayenge.
- Isse rapid test data entry possible hai, time bachaata hai.

56 2. Mobile Elements in Appium (App Testing with Python)

A. Locators in Mobile Automation (Appium ke liye)

- **resource-id:** Similar to `id` in web.
- **accessibility id:** User-accessibility label.
- **class name:** Android/iOS UI widget ka class.
- **XPath:** Powerful but a bit slower on mobile, still very useful.

B. Locators Collect Karne Ka Tarika (Android Example)

1. UIAutomatorViewer (Android)

- `android-sdk/tools/bin/UIAutomatorViewer` tool run karo (Android SDK installed ho).
- Mobile device connect rakho, target app open karo.
- UIAutomatorViewer me screenshot lo—**click on any element, uska resource-id, class, XPath, sab dikhega**.
- Copy locator and use in Appium scripts.

Appium Android Locator Example

```
from appium import webdriver

element = driver.find_element("id", "com.app.package:id/submit")
element = driver.find_element("accessibility id", "loginButton")  # Accessibility ID
element = driver.find_element("class name", "android.widget.Button")
element = driver.find_element("xpath", "//android.widget.Button[@content-desc='login']")
```

2. Appium Inspector (Cross-platform GUI Tool)

- Download and install "Appium Inspector".
- Start Appium server.
- Connect device/emulator, open Inspector.
- App screen ka snapshot dikhega; kisi bhi element pe click karne se sab possible locator strategies dikhenge (resource-id, class, xpath, accessibility label, etc.).
- **Copy locator details aur apne scripts me use karo.**

57 3. Pro Beginner Extensions/Tools List

Tool/Extension	Platform	What/Why	Install Steps
SelectorHub	Web (Selenium)	Robust XPath/CSS auto-suggest & validation	Browser extens Store)
ChroPath	Web (Selenium)	Alternate locator auto-suggest	Browser extens Store)
Fake Filler	Web (Testing)	Auto-fill web forms for testing	Browser extens Store)
Axe DevTools	Web	Automated accessibility audit (a11y)	Browser extens Store)
Appium Inspector	Mobile (Appium)	Visual locator tool for Android/iOS apps	Download app (
UIAutomatorViewer	Mobile (Appium)	Android XML hierarchy inspector & locator visualizer	Included with An

Golden Beginner Tips

- Always try **ID/resource-id** locator first (fastest & most reliable).
- Tab tak plugin/extension use karo until you get confident extracting locators manually.
- **Plugin locator copy karne ke baad**, apne script me paste karke test karo ki sahi kaam kar raha hai ya nahi.
- Mobile pe accessibility labels set karwana developers se bolo—automation simple ho jayega.

Ek beginner ko **har step hand-on practice** karna zaruri hai, plugins/Inspector tools se start karna best hai. Zeher notes bana rahe ho bro, ab in extensions ke workflow add kar loge toh full paisa vasool ho jayega!

==========

E-Commerce Selenium Automation Framework - Complete Beginner Guide

Ye ek professional automation framework hai jo Selenium aur PyTest ke saath banaya gaya hai taaki e-commerce website (jaise OpenCart) ko test kar sako. Main tumhe har ek part ko detail mein samjhaunga – kya hai, kyun hai, kaise use karoge – taaki tum confidently isko setup karo aur professional QA tester ban jao!

PROJECT STRUCTURE OVERVIEW for industry ready approach ...

Sabse pehle project ka structure samajhte hain. Ye industry-standard tarike se organized hai taaki code maintain karna aur samajhna aasan ho:

Project Structure

```
ecommerce_automation/                Main project folder
          README.md                        Project ki basic info
          requirements.txt                 Python packages ki list
          pytest.ini                      Test configuration file
          .env                            Secret values jaise URL/
   passwords
          conftest.py                     Browser setup aur teardown

          tests/                          Yahan actual test cases honge
                test_login.py              Login testing
                test_signup.py             Registration testing
                test_cart.py               Cart functionality
                test_checkout.py           Order placement

          pages/                          Page Object Model files
                base_page.py               Common functions
                login_page.py              Login page actions
                home_page.py               Homepage actions
                cart_page.py               Cart page actions
                checkout_page.py           Checkout actions

          utils/                          Helper functions
                logger.py                  Log messages
                screenshot.py              Screenshots capture
                data_reader.py             Excel/CSV data read

          data/                           Test data files
                users.xlsx                 Login credentials
                products.csv               Product info

          reports/                        Test reports (auto-generated)
          screenshots/                    Failure screenshots (auto-
   generated)
```

Ab ek-ek karke har cheez ko detail mein dekhte hain.

ROOT LEVEL FILES - Setup Ka Pehla Step

Ye files project ke root mein hoti hain aur project ko setup karne ke liye zaroori hain.

requirements.txt

- **Kya hai?** Ye ek list hai Python packages ki jo project chalane ke liye chahiye.
- **Kyun chahiye?** Taaki ek hi command se saari dependencies install ho jayein aur team ke saath share karna aasan ho.
- **Kya likha hoga?**

requirements.txt

```
selenium==4.15.0          # Browser automation ke liye
pytest==7.4.3             # Testing framework
pytest-html==4.1.1        # HTML reports ke liye
openpyxl==3.1.2           # Excel files padhne ke liye
webdriver-manager==4.0.1 # Browser driver auto manage karta hai
python-dotenv==1.0.0      # .env file se data padhne ke liye
```

- **Kaise use karoge?** Terminal mein ye command run karo:

Installation Command

```
pip install -r requirements.txt
```

.env

- **Kya hai?** Isme sensitive info store karte hain jaise website URL, username, password.
- **Kyun chahiye?** Taaki ye cheezein code mein hardcode na karni padein aur secure rahein.
- **Kya likha hoga?**

.env File

```
BASE_URL=https://demo.opencart.com  # Website ka base URL
USERNAME=testuser@gmail.com         # Test user ka email
PASSWORD=Test@123                   # Test user ka password
HEADLESS=false                      # Browser UI ke saath chalega
TIMEOUT=10                          # Wait time in seconds
```

- **Kaise use karoge?** Python mein `os.getenv("BASE_URL")` likhkar access karoge.

pytest.ini

- **Kya hai?** PyTest ke liye configuration file hai.
- **Kyun chahiye?** Tests ko run karne ke default settings set karne ke liye.
- **Kya likha hoga?**

pytest.ini

```
[pytest]
addopts = --html=reports/report.html --self-contained-html -v
markers =
    smoke: Quick sanity tests       # Fast basic tests
    regression: Detailed test cases # Complete tests
    cart: Cart related tests        # Cart-specific tests
```

- **Samajhna:**

- `addopts` = Har test run pe HTML report banega aur verbose output milega.
- `markers` = Tests ko category mein divide karne ke liye.

README.md

- **Kya hai?** Project ka introduction aur setup instructions.
- **Kyun chahiye?** Taaki koi bhi nayi team member easily setup kar sake.
- **Kya likha hoga?**

README.md

```
# E-Commerce Automation Framework

## Setup:
1. Install Python 3.8+
2. Run: `pip install -r requirements.txt`
3. Update `.env` file with your URLs

## Run Tests:
- All tests: `pytest`
- Smoke tests: `pytest -m smoke`
- With reports: `pytest --html=reports/report.html`
```

conftest.py - Browser Setup Ka Boss

- **Kya hai?** Ye file PyTest ke liye browser setup aur cleanup karti hai.
- **Kyun chahiye?** Taaki har test se pehle browser khule aur test ke baad band ho jaye.
- **Complete Code:**

conftest.py

```python
import pytest                        # Testing framework
import os                            # Environment variables ke liye
from selenium import webdriver       # Browser control
from selenium.webdriver.chrome.options import Options  # Chrome settings
from webdriver_manager.chrome import ChromeDriverManager  # Driver manager
from dotenv import load_dotenv       # .env file load

load_dotenv()  # .env file se values load karo

@pytest.fixture(scope="function")  # Har test ke liye naya browser
def driver():
    """
    Browser setup aur cleanup
    """
    chrome_options = Options()  # Chrome ke options set karo

    # Headless mode check (bina UI ke)
    if os.getenv("HEADLESS", "false").lower() == "true":
        chrome_options.add_argument("--headless")  # Background mein chalega

    chrome_options.add_argument("--no-sandbox")  # Linux compatibility
    chrome_options.add_argument("--disable-dev-shm-usage")  # Memory fix
    chrome_options.add_argument("--window-size=1920,1080")  # Window size

    # Chrome browser start karo
    driver = webdriver.Chrome(
        ChromeDriverManager().install(),  # Driver auto download
        options=chrome_options
    )

    driver.maximize_window()  # Full screen
    driver.implicitly_wait(int(os.getenv("TIMEOUT", "10")))  # Default wait 10 sec

    yield driver  # Test ko driver do
    driver.quit()  # Test ke baad band karo

@pytest.fixture(scope="session")  # Poore session ke liye ek baar
def base_url():
    """Base URL for all tests"""
    return os.getenv("BASE_URL", "https://demo.opencart.com")  # Default URL
```

- **Line-by-Line:**
  - `@pytest.fixture(scope="function")` = Har test ke liye naya browser khulega.
  - `yield driver` = Test ko browser deta hai, baad mein `quit()` band karta hai.

– `implicitly_wait(10)` = Har action ke liye 10 sec tak wait karega.

pages/ - Page Object Model (POM) Files

POM Kya Hai? Har web page ke liye ek alag class banate hain jisme us page ke elements (locators) aur actions (click, type) define hote hain. **Kyun Use Karte Hain?** Code organized rehta hai, agar ek element change ho to sirf us page ki class mein update karo, baaki sab apne aap theek ho jata hai.

pages/base_page.py - Common Functions

- **Kya hai?** Ye base class hai jo sab pages ke liye common functions rakhti hai.
- **Kyun chahiye?** Taaki har page mein same code baar-baar na likhna pade.
- **Complete Code:**

pages/base_page.py

```python
from selenium.webdriver.support.ui import WebDriverWait  # Wait ke liye
from selenium.webdriver.support import expected_conditions as EC  # Conditions
from selenium.webdriver.common.by import By  # Locators ke liye
import os  # Environment variables

class BasePage:
    """
    Base class for all page objects
    """

    def __init__(self, driver):  # Constructor
        self.driver = driver  # Driver save karo
        self.wait = WebDriverWait(driver, int(os.getenv("TIMEOUT", "10")))  # Wait object

    def find_element(self, by, locator):  # Element dhoondho
        """Element dhoondho with wait"""
        return self.wait.until(EC.presence_of_element_located((by, locator)))

    def click_element(self, by, locator):  # Click karo
        """Element pe click karo"""
        element = self.wait.until(EC.element_to_be_clickable((by, locator)))
        element.click()

    def enter_text(self, by, locator, text):  # Text daalo
        """Text field mein value daalo"""
        element = self.find_element(by, locator)
        element.clear()  # Purana text hatao
        element.send_keys(text)  # Naya text daalo

    def get_text(self, by, locator):  # Text lo
        """Element ka text return karo"""
        return self.find_element(by, locator).text

    def is_element_visible(self, by, locator):  # Visibility check
        """Check karo element dikhta hai ya nahi"""
        try:
            self.wait.until(EC.visibility_of_element_located((by, locator)))
            return True
        except:
            return False
```

pages/login_page.py - Login Page Actions

- **Kya hai?** Login page ke elements aur actions define karta hai.
- **Complete Code:**

pages/login_page.py

```python
from selenium.webdriver.common.by import By  # Locators ke liye
from pages.base_page import BasePage  # BasePage class ko inherit kiya
import os  # Env variables ke liye

class LoginPage(BasePage):
    """
    LoginPage ne BasePage ko inherit kiya hai
    Yahan login page ke saare elements aur actions define hain
    """

    # Elements ke locators - (By method, locator string)
    EMAIL_INPUT = (By.ID, "input-email")
    PASSWORD_INPUT = (By.ID, "input-password")
    LOGIN_BUTTON = (By.CSS_SELECTOR, "input[value='Login']")
    ERROR_MESSAGE = (By.CSS_SELECTOR, ".alert-danger")

    def go_to_login_page(self):
        """Login page pe jaane ke liye URL open karo"""
        base_url = os.getenv("BASE_URL")
        self.driver.get(f"{base_url}/index.php?route=account/login")

    def enter_email(self, email):
        """Email input field mein value daalo"""
        self.enter_text(*self.EMAIL_INPUT, email)
        #         Yahan * ka matlab: tuple ko alag-alag arguments mein todna
        # self.EMAIL_INPUT = (By.ID, "input-email")
        # *self.EMAIL_INPUT     By.ID, "input-email"
        # Final call     self.enter_text(By.ID, "input-email", email)

    def enter_password(self, password):
        """Password input field mein value daalo"""
        self.enter_text(*self.PASSWORD_INPUT, password)

    def click_login_button(self):
        """Login button pe click karo"""
        self.click_element(*self.LOGIN_BUTTON)

    def login(self, email, password):
        """Poora login process complete karo"""
        self.enter_email(email)
        self.enter_password(password)
        self.click_login_button()

    def get_error_message(self):
        """Agar login fail ho toh error message return karo"""
        if self.is_element_visible(*self.ERROR_MESSAGE):
            return self.get_text(*self.ERROR_MESSAGE)
        return ""
```

- *** (Star)**: Tuple jaise (By.ID, "value") ko unpack karke function mein alag arguments bana deta hai.

- Agar * nahi lagayenge to poora tuple ek hi argument ban jaayega, aur error milega.
- Yahan BasePage ke functions ko reuse kiya gaya hai jaise `enter_text()`, `click_element()`.

- **Samajhna:**
  - `*self.EMAIL_INPUT` = Tuple (By.ID, "input-email") ko unpack karta hai.
  - `BasePage` se inherit karke `enter_text`, `click_element` jaise functions use hote hain.

tests/ - Actual Test Cases

Ye folder mein real test cases hote hain jo website ki functionality check karte hain.

tests/test_login.py - Login Test Cases

- **Complete Code:**

tests/test_login.py

```python
import pytest  # Testing framework
import os  # Env variables
from pages.login_page import LoginPage  # Login page class

class TestLogin:
    """
    Login ke test cases
    """

    @pytest.mark.smoke  # Smoke test category
    def test_valid_login(self, driver, base_url):  # Valid login
        """Valid credentials se login check"""
        login_page = LoginPage(driver)  # Object banao
        login_page.go_to_login_page()  # Login page pe jao
        login_page.login(os.getenv("USERNAME"),
            os.getenv("PASSWORD"))  # Login karo
        assert "account" in driver.current_url, "Login nahi hua!"

    @pytest.mark.regression  # Regression test category
    def test_invalid_login(self, driver):  # Invalid login
        """Galat credentials se login check"""
        login_page = LoginPage(driver)
        login_page.go_to_login_page()
        login_page.login("wrong@email.com", "wrongpass")  # Galat
            data
        error = login_page.get_error_message()
        assert "No match" in error, "Error message nahi dikha!"
```

- **Samajhna:**
  - `@pytest.mark.smoke` = Ye test fast sanity check ke liye hai.
  - `driver` aur `base_url` = `conftest.py` se aate hain.

– `assert` = Test pass ya fail decide karta hai.

utils/ - Helper Functions

Ye folder mein chhoti-chhoti helper files hoti hain.

utils/logger.py

- **Kya hai?** Test ke logs banata hai.
- **Kyun chahiye?** Taaki pata chale test mein kya hua.
- **Code:**

utils/logger.py

```python
import logging  # Logging module
import os  # Folder banane ke liye
from datetime import datetime  # Timestamp ke liye

def get_logger(name):
    logger = logging.getLogger(name)
    if not logger.handlers:  # Agar handler nahi hai
        os.makedirs("logs", exist_ok=True)  # Logs folder banao
        handler =
            logging.FileHandler(f"logs/test_{datetime.now().strftime('%Y%m%d_%H%M%S'
        handler.setFormatter(logging.Formatter('%(asctime)s |
            %(message)s'))  # Format
        logger.addHandler(handler)
        logger.setLevel(logging.INFO)  # Info level logs
    return logger
```

utils/screenshot.py

- **Kya hai?** Test fail hone pe screenshot save karta hai.
- **Kyun chahiye?** Debugging ke liye.

utils/data_reader.py

- **Kya hai?** Excel ya CSV se test data padhta hai.
- **Kyun chahiye?** Data-driven testing ke liye.

data/ - Test Data Files

- **Kya hai?** Test data files jaise `users.xlsx` ya `products.csv`.
- **Kyun chahiye?** Alag-alag test scenarios ke liye data provide karne ke liye.
- **Example (users.xlsx):**

username	password
test@email.com	Test@123

Tests Kaise Run Karoge?

Terminal mein ye commands use karo:

Test Run Commands

```
# Setup karo
pip install -r requirements.txt

# Sab tests run karo
pytest

# Specific test file
pytest tests/test_login.py

# Smoke tests only
pytest -m smoke

# Report ke saath
pytest --html=reports/report.html

# Verbose output (details)
pytest -v
```

Framework Ke Fayde

1. **Organized:** Har cheez alag-alag rakhi hai.
2. **Easy Maintenance:** Ek change se sab fix ho jata hai.
3. **Professional:** Industry mein aise hi frameworks use hote hain.

Common Doubts Clear Kiye

@pytest.fixture(scope="function") Kya Hai?

- Ye PyTest ko batata hai ki ye function setup ke liye hai.
- `scope="function"` = Har test ke liye naya browser khulega.

POM Kyun Use Karte Hain?

- Page Object Model se har page ke elements aur actions alag class mein hote hain.
- Agar ek locator change ho, to sirf us page ko update karo, baaki tests safe!

Inheritance Ka Role Kya Hai?

- `LoginPage` class `BasePage` se inherit karti hai taaki common functions baar-baar na likhne padein.

XPath ka .// Kya Matlab Hai

Step-by-step Explanation in Simple Words

Jab hum Selenium ya XPath likhte hain, kabhi kabhi aise likha hota hai:

Example XPath

```
.//span[text()='Sign In']
```

Aao isko **step-by-step aur simple words** mein samjhte hain:

1. . (Dot) ka Matlab

Meaning of . (Dot)

`dot (.)` ka matlab hai: "Current element" yaani jahan abhi tum ho.

- Agar tum ek `<button>` ke andar ho, to `.` ka matlab hota hai: **wohi button** khud.
- Iska use tab hota hai jab tum kisi specific element ke **andar** hi search karna chahte ho.

Example:

> "Main abhi `button` ke andar hoon." Toh XPath expression `.//span` ka matlab hai, iss button ke andar kahin bhi `<span>` search karo.

2. // ka Matlab

Meaning of // (Double Slash)

`//` ka matlab hai: "Is jagah ke andar jitne bhi children, grandchildren hain — sabko check karo".

- `//` recursive search karta hai.
- Matlab kisi bhi level pe jo match mile, **use le lega**.

Example:

`//div` matlab, document ke har jagah se `<div>` elements dhundho — chahe wo kisi bhi level pe ho.

3. `.//` ka Matlab

Combined Meaning: `.//`

`.//` ka matlab hai: "Current element ke andar kahi bhi (chahe nested kitna bhi deep ho) jo bhi matching element ho, use khojho."

- Yeh `.` se shuru hota hai toh search starting point **current node** hai, na ki pura document.
- `//` matlab anywhere inside, recursive search.

Example XPath:

Usage Example

```
//button[.//span[text()='Sign In']]
```

Iska matlab:

> Select karo `<button>` jiske andar kahin bhi ek `<span>` ho jiska text ho `"Sign In"`.

XPath `//` vs `.//` Ka Farak Aur Galatiyan

Avoid This Common Mistake

Galti: `//button[//span[text()='Sign In']]`
Yeh XPath sahi nahi hai kyunki:

- `//span[...]` pure document mein **sab jagah** `<span>` dhundhta hai.
- Iska matlab hai ki **button ke andar hona** ka koi matlab nahi.
- Yeh condition XPath me **sahi tarah se limit nahi karta** ki span button ke andar hona chahiye.

Correct: `//button[.//span[text()='Sign In']]`

Ismein `.//span` button ke andar hi restrict karta hai search ko, jisse expected result milega.

Summary Table

2tablealtwhite
c

tablehead

`.`	Current element, jahan abhi tum ho
`//`	Anywhere inside (children, grandchildren, etc.)
`.//`	Current element ke andar kahin bhi (nested)
`//button[.//span]`	Button jisme span nested ho

Final Tip

XPath likhte waqt:

- Agar tum kisi element ke andar **nested** elements dhundhna chahte ho, toh `.//` ka use karo.
- Agar tum document ka pura scope **global** check karna chahte ho, toh sirf `//` ka use karo.

Ye guide beginner friendly hai, aur XPath ke inner working ko Hindi-English mix me simple tarike se samjhane ki koshish karta hai.

==========

`pytest-watch` – Nodemon jaisa tool for Python Tests

Objective:

Jaise Node.js projects mein `nodemon` use karte hain taaki har file change pe server auto-restart ho jaaye — waise hi **Python tests** ko auto-run karwane ke liye hum use karte hain `pytest-watch`.

1. Tool Install Karna

Installation Command

```
pip install pytest-watch
```

- Ye tool install karte hi tumhare system mein `ptw` command available ho jaata hai.

- `ptw` = **pytest watch**

2. Basic Command: Auto Run Tests

Basic Auto-Run Command

```
ptw
```

- Isse `watch mode` activate ho jaata hai.
- Ab jaise hi tum test file ya source code file change karoge — tests **automatically run** ho jaayenge.

3. Realistic Test Example:

test_calculator.py

```python
# test_calculator.py

def test_addition():
    assert 2 + 3 == 5

def test_subtraction():
    assert 5 - 2 == 3
```

Agar tum `ptw` chala doge, to har baar file save karne par dono tests dobara run honge.

4. Run Only Specific Tests (`-k` flag)

Agar tum sirf ek specific test run karwana chahte ho (jaise sirf `test_addition`) tab:

Specific Test Filter

```
ptw -- -k "addition"
```

Explanation:

- `-k` is a filter.
- `"addition"` ka matlab: sirf wo test jinke naam mein "addition" word ho — wo hi run honge.
- Ye **partial match** hota hai.

More Examples with `-k`:

—X—X—

Command	What it does
`ptw -- -k "login"`	Sirf tests run honge jinke naam mein `login` aata hai
`ptw -- -k "invalid"`	Sirf invalid waale test run honge (jaise `test_invalid_input`)

5. Ignore Folder from Watch

Agar tum chahte ho ki koi folder ke changes ignore ho jaayein (e.g., migrations, venv, etc.):

Ignore Folders

```
ptw --ignore "venv/"
```

6. Use Custom Runner (e.g., for verbose output)

Custom Runner with Verbose Output

```
ptw --runner "pytest -v"
```

- `-v` (verbose) flag deta hai zyada detailed output har test ka.

7. Alternative for Linux/macOS:

Agar `ptw` nahi chalta (ya Windows mein issue ho) to:

Linux/macOS Alternative

```
watch -n 2 pytest
```

- Ye har **2 second** mein `pytest` run karega, chahe file badli ho ya nahi.
- Downside: Ye smart nahi hai — har 2 second mein run karta rahega unnecessarily.

8. Advanced Tip – Multiple Options Combine Karke

Combined Options

```
ptw -- -v -k "auth"
```

Matlab:

- Verbose output ke saath,
- Sirf `auth` related test hi run karne hain

Summary Table:

—X—X—

Use Case	Command
Auto-run all tests	`ptw`
Run only specific tests	`ptw -- -k "keyword"`
Verbose test output	`ptw -- -v`
Ignore folder changes	`ptw --ignore "foldername/"`
Custom runner with flags	`ptw --runner "pytest -v -k 'login'"`
Linux fallback	`watch -n 2 pytest`

Suitable for:

- ✓ Developers doing TDD (Test Driven Development)
- ✓ QA Engineers automating regression tests
- ✓ Python devs migrating from Node.js and missing `nodemon`
- ✓ Security testers validating test cases frequently

`ptw` (pytest-watch) sirf ek file run karta hai ya saare test files?

Answer:

`ptw` **saare test files run karta hai** (jinmein test cases hain) — **jaise hi koi bhi relevant file change hoti hai**.

Thoda Detail mein:

Jab tum command chalaate ho:

Basic Command

```
ptw
```

To:

- Yeh tumhare **poore project directory ko watch** karta hai.
- Jaise hi **kisi bhi Python file** mein change hota hai (especially test files ya unse linked source files), `pytest` dobara **saare test cases** ko run karta hai — **sirf wo test nahi jisme change hua ho**.

Example:

Agar tumhare paas yeh test files hain:

```
tests/
 test_auth.py
 test_cart.py
 test_checkout.py
```

Agar tum `test_cart.py` file mein kuch change karte ho (aur save karte ho), to:

- `ptw` puri directory ke test cases dobara run karega:
  - test_auth
  - test_cart
  - test_checkout

Yeh ek tarah se **full test run** hota hai **on every save**.

Want to Limit? Use Filter:

Agar tum sirf ek particular test file ko run karna chahte ho — chahe koi file change ho ya nahi — to use karo:

Specific File Run

```
ptw tests/test_cart.py
```

Ya phir test name filter:

Test Name Filter

```
ptw -- -k "cart"
```

Summary:

—X—X—

Situation	Behavior
Normal `ptw`	✓ All tests run jab bhi koi change hota hai
Change sirf ek file mein	✓ Tab bhi all tests run honge
Specific file chahiye run karna	Use `ptw test_filename.py`
Specific test run karna	Use `ptw -- -k "testname"`

==========

pytest -s Command — Most important command...

Bahut badhiya question bhai! `pytest -s` command testing karte waqt input/output ya print statements ko samajhne ke liye kaafi kaam aata hai. Chalo simple hinglish mein full explain karta hoon:

Syntax:

Basic Syntax

```
pytest -s
```

Kya karta hai -s?

-s ka matlab hota hai:

"Don't capture output."

Jab tum normally pytest chalaate ho, to:

- `print()` ke output ya `input()` calls screen par nahi dikhte.
- Kyunki pytest by default stdout/stderr capture karta hai.

Lekin agar tum:

With -s Flag

```
pytest -s
```

chalaate ho, to:

- `print()` ka output terminal mein dikhne lagega.
- `input()` se tum input le sakte ho during test.

Kab use karein -s?

Useful when:

- Debug karna ho test code (print statements se)
- `input()` lena ho user se during test
- pdb (Python debugger) ya interactive code chalana ho

Example:

test_example.py

```
# test_example.py

def test_addition():
    name = input("Enter your name: ")  # Without -s, ye chalega hi
        nahi
    print("Hello", name)
    assert 2 + 2 == 4
```

Run:

Run Command

```
pytest -s test_example.py
```

Output:

```
Enter your name: Satyam
Hello Satyam
.
1 passed
```

Without -s kya hota?

Agar tum just `pytest test_example.py` chalaoge, to:

- `input()` par test hang ya crash ho jaayega.
- `print()` ka output nahi dikhega.

Summary Table:

—X—X—

Command	Kya karta hai
`pytest`	Normal test run, print/input hidden
`pytest -s`	Print visible, input allow karta hai
`pytest -s -v`	Print visible + detailed test output

Agar tum debug kar rahe ho ya koi interactive behavior test kar rahe ho, to `-s` zaroor lagao
Aur batana agar tumko pdb debugger bhi chahiye samajhna, vo bhi professional testing mein kaam aata hai.

==========

How `pytest` Automatically Runs Your `test_` Functions (Without Manual Call)

1. Tumne Ek Function Banaya:

Function Definition

```
def test_dashboard_navigation(driver):
    ...
```

Tumne is function ko **kahin pe bhi manually call nahi kiya**:

```
test_dashboard_navigation(driver)   #     Aisa kuch nahi likha
```

Phir bhi jab `pytest` command chalate ho — ye function **run ho jaata hai**!

2. Pytest Ka Internal Kaam: Test Discovery System

Jab Tum Command Chalate Ho:

Pytest Command

```
pytest
```

Tab `pytest` kuch kaafi important steps karta hai:

—c—X—

Step	Kaam
1	Current directory aur subfolders scan karta hai
2	Sirf wo files select karta hai jinka naam `test_*.py` hota hai
3	Fir un files ke andar functions dhundta hai jo `test_` se start hote hain
4	Un functions ko automatically **run/call** karta hai
5	Har test ka pass/fail result CLI pe show karta hai

Example

`test_login.py`

test_login.py

```
def test_dashboard_navigation(driver):
    print("Testing dashboard navigation")
```

Run Command:

Run Command

```
pytest -s
```

Output:

```
Testing dashboard navigation
1 passed in 1.02s
```

Tumne function manually call nahi kiya, phir bhi run hua!

Kyu Run Hua Without Manual Call?

Answer: `pytest` automatically samajh jaata hai ki:

- File ka naam `test_*.py` → ye test file hai
- Function ka naam `test_` se start ho raha hai → ye test function hai

Isliye usko **automatically run** karta hai — bina kisi manual function call ke.

Internally Pytest Ye Kaam Kar Raha Hota Hai:

Internal Working

```
import test_login                          # tumhari test file import
    karta hai
test_login.test_dashboard_navigation(driver=fixture_object)
```

Tumhe kuch manually likhne ki zarurat nahi hoti — `pytest` ye sab **automatically** handle karta hai.

Important Rule:

—X—X—

Function Naming	Pytest Behavior
`test_abc()`	Run hoga automatically
`abc_test()`	Run nahi hoga pytest ke through

Bonus: `driver` Ka Value Kahan Se Aata Hai?

- Agar function ke andar `driver` diya hai, to uska data **pytest fixture** se aata hai.
- Ye fixtures tum `conftest.py` file me define karte ho.
- Example:

Fixture Example

```
@pytest.fixture
def driver():
    # setup browser driver
    yield driver_instance
    # teardown code
```

Summary Table:

—X—X—

Question	Answer
Function manually call nahi kiya, fir bhi run kaise?	Pytest automatically discover karke run karta hai `test_` se start hone wale functions
Kaunsi files pytest uthata hai?	Sirf those files jinka naam `test_*.py` hota hai
Kaunsa function pytest run karta hai?	Wo functions jo `test_` se start ho rahe hain
`driver` ya arguments ka value kaun deta hai?	Pytest fixtures system through setup hota hai
Agar naam `dashboard_test()` ho toh?	Wo function pytest se run nahi hoga

==========

Lambda Functions & Expected Conditions in Selenium - Complete Guide

PART 1: `lambda` Function in Python (Especially for Selenium)

What is lambda?

Lambda ek **anonymous (nameless)** function hota hai jo **one-line expression** ke liye use hota hai.

Syntax:

Lambda Syntax

```
lambda arguments: expression
```

Example (Normal vs Lambda):

Normal vs Lambda Function

```
# Normal function
def add(x, y):
    return x + y

# Lambda version
add = lambda x, y: x + y
print(add(2, 3))  # Output: 5
```

Why we use `lambda` in Selenium?

Selenium mein humein kai baar aise **functions pass karne hote hain as arguments** — jaise `WebDriverWait` ke andar.

Example:

Lambda in Selenium WebDriverWait

```
from selenium.webdriver.support.ui import WebDriverWait
wait = WebDriverWait(driver, 10)

wait.until(lambda driver: driver.find_element(By.ID, "submit"))
```

Explanation:

- `lambda driver:` matlab hum driver object input le rahe hain
- `driver.find_element(...)` — agar element mil gaya toh wait success
- Jab tak element nahi milta, Selenium wait karta rahega

Iska fayda: Lambda se hum **short inline logic likh sakte hain** bina alag function define kiye.

When NOT to use `lambda`?

Agar tumhara logic 1 line se zyada ka hai, toh **normal function** hi use karo:

Normal Function Alternative

```
def wait_for_submit(driver):
    return driver.find_element(By.ID, "submit")

wait.until(wait_for_submit)
```

PART 2: `EC` (Expected Conditions) in Selenium

Import:

EC Import Statement

```
from selenium.webdriver.support import expected_conditions as EC
```

1. EC.presence_of_element_located(locator)

Meaning:

DOM mein element **exist karta hai** ya nahi — visible ho ya na ho.

Presence of Element Example

```
WebDriverWait(driver, 10).until(
    EC.presence_of_element_located((By.ID, "email"))
)
```

Use when:

- Tumhara element page mein aa gaya ho but abhi visible na ho.

2. EC.visibility_of_element_located(locator)

Meaning:

Element **present bhi ho AND user ko visible bhi ho.**

Visibility of Element Example

```
WebDriverWait(driver, 10).until(
    EC.visibility_of_element_located((By.XPATH,
        "//input[@type='text']"))
)
```

Use when:

- Tum **user interaction** chahte ho — jaise typing ya clicking.

3. EC.element_to_be_clickable(locator)

Meaning:

Element:

- DOM mein ho

- Visible ho
- Enabled ho

Element to be Clickable Example

```
WebDriverWait(driver, 15).until(
    EC.element_to_be_clickable((By.ID, "loginBtn"))
).click()
```

Use when:

- Button ya link click karwana hai after full page load

4. EC.invisibility_of_element(locator)

Meaning:

Check karta hai ki element **hidden ho gaya ya remove ho gaya** DOM se.

Invisibility of Element Example

```
WebDriverWait(driver, 10).until(
    EC.invisibility_of_element((By.CLASS_NAME, "loading-spinner"))
)
```

Use when:

- Wait kar rahe ho kisi loader ya overlay ke hide hone ka

5. EC.text_to_be_present_in_element(locator, text)

Meaning:

Yeh tab tak wait karega jab tak kisi specific element mein koi **text aa jaaye.**

Text to be Present Example

```
WebDriverWait(driver, 10).until(
    EC.text_to_be_present_in_element((By.ID, "status"), "Order
        Complete")
)
```

Use when:

- Tum kisi text ke update hone ka wait kar rahe ho

6. `EC.alert_is_present()`

Meaning:

JavaScript alert pop-up present hai ya nahi.

Alert is Present Example

```
WebDriverWait(driver, 5).until(EC.alert_is_present()).accept()
```

Use when:

- Alert ya confirmation popup aane ka wait karna hai.

Bonus: Full EC Conditions Cheat Sheet

—X—X—X—

EC Condition	Explanation	Use-case Example
`presence_of_element_located`	DOM mein ho	Page load ke baad element aya ya nahi
`visibility_of_element_located`	User ke liye visible ho	Typing ya UI interaction se pehle
`element_to_be_clickable`	Clickable (visible + enabled)	Buttons, tabs, menu
`invisibility_of_element`	Element chhup gaya ya gaya	Loader, spinners
`text_to_be_present_in_element`	Specific text appear ho gaya	Status messages
`alert_is_present`	JavaScript alert show ho gaya	Accept, dismiss alerts
`frame_to_be_available_and_switch_to_it`	Frame ready hai aur switch kar sakte ho	iFrames ke saath automation

Final Words:

Yeh cheezein **automation ko reliable banati hain** — bina random sleep ke.
Agar tum `lambda` aur `EC` samajh jaate ho, toh Selenium ke saare `wait` logic powerful ho jaate hain.
Shortcuts jaise `ptw`, `pytest.mark`, fixtures — yeh sab combine karke tum ek **robust test framework** build kar sakte ho.

Agar chaho toh:

- `conftest.py` fixtures
- `pytest.ini` setup
- Page Object Model
- CI/CD integration with Selenium

bhi Hinglish + real-world project ke examples ke saath bana ke de sakta hoon.

==========

`.click()` Behavior in Selenium

`.click()` tabhi kaam karta hai jab:

1. **Element DOM mein ho** ✓
2. **Element screen pe visible ho agar nahi hai then scroll karke visible karo uusko..** ✓
3. **Element kisi aur cheez se blocked na ho** ✓ (jaise popup, modal, loader, etc.)
4. **Element pe koi animation ya transition na chal rahi ho** ✓

Agar ye conditions fail hoti hain, toh ye errors milte hain:

—X—X—X—

Situation	Error Type	Example Message
Element screen ke bahar hai	`ElementClickInterceptedException`	"Element is not clickable at point..."
Element kisi overlay ke niche hai	`ElementClickInterceptedException`	"Click intercepted by modal/overlay..."
Element visible nahi hai	`ElementNotInteractableException`	"Element not interactable..."
Element DOM mein hi nahi hai	`NoSuchElementException`	"Unable to locate element..."

Best Practices to Avoid These Errors

1. **Scroll into view:**

Scroll to Element

```
driver.execute_script("arguments[0].scrollIntoView({block:
    'center'});", element)
```

2. **Wait for visibility:**

Wait for Visibility

```
WebDriverWait(driver, 10).until(EC.visibility_of(element))
```

3. **Wait for clickability:**

Wait for Clickable

```
WebDriverWait(driver,
    10).until(EC.element_to_be_clickable((By.XPATH, xpath)))
```

4. **Wait for overlays to disappear:**

Wait for Overlay to Disappear

```
WebDriverWait(driver,
    10).until(EC.invisibility_of_element_located((By.CLASS_NAME,
    "modal-class")))
```

5. **Fallback to JS click (if needed):**

JavaScript Click

```
driver.execute_script("arguments[0].click();", element)
```

Key Selenium WebElement Methods & Their Behaviors

—l—X—X—

Method	Behavior Summary	Common Pitfalls / Errors
`.send_keys()`	Text input karta hai element mein	`ElementNotInteractableException` agar element disabled ya hidden ho
`.clear()`	Input field ko empty karta hai	Same as above — agar element interactable nahi hai

`.submit()` Form ko submit karta hai (only if element is inside `<form>`) `JavascriptException` agar form structure galat ho

---

`.get_attribute()` Element ka attribute value deta hai Safe — even if element hidden ho

---

`.text` Element ke andar ka visible text deta hai Hidden text return nahi hota

---

`.is_displayed()` Check karta hai ki element screen pe visible hai ya nahi May return false even if element is in DOM

`.is_enabled()` Check karta hai ki element interactable hai ya nahi Useful before `.click()` or `.send_keys()`

`.is_selected()` Checkbox ya radio button selected hai ya nahi Only works for selectables

---

`.get_property()` JS property value deta hai (e.g. `value`, `checked`) More reliable than `get_attribute()` in some cases

---

`.location` / `.size` Element ka position aur dimensions deta hai Useful for visual debugging

---

`.screenshot()` Element ka screenshot le sakta hai Great for visual test logs

---

Common Gotchas to Watch Out For

- **Hidden Elements:** Agar element DOM mein hai but `display: none` ya `visibility: hidden` hai, toh `.click()`, `.send_keys()` fail karenge.
- **Disabled Inputs:** Agar input `disabled` hai, toh `.send_keys()` and `.clear()` fail karenge.
- **Dynamic DOM Updates:** Agar element reload ho raha ho (SPA apps), toh stale reference error aa sakta hai:

```
selenium.common.exceptions.StaleElementReferenceException
```

Pro Tips for Robust Testing

- Use `WebDriverWait` with `expected_conditions` for **every interaction**.
- Always check `is_displayed()` and `is_enabled()` before clicking or typing.
- Prefer `get_property("value")` over `get_attribute("value")` for input fields.
- Use `try-except` blocks to handle flaky UI behavior gracefully.

==========

Hidden Element ka Matlab Selenium ke Context Mein

Hidden ka matlab:

Element **DOM mein present hai**, lekin **screen pe visible nahi hai** — ya toh CSS ke through hide kiya gaya hai, ya wo kisi scroll ke neeche chhupa hua hai.

Hidden ke 2 Types:

1 CSS Hidden (True Hidden)

```
display: none;
visibility: hidden;
opacity: 0;
```

Is case mein Selenium ke liye wo element **interactable nahi hota**. Agar tum `.send_keys()` ya `.click()` karoge, toh error milega:

```
ElementNotInteractableException
```

2 Scroll ke neeche (Visible but out of view)

- Element screen pe nahi dikh raha, lekin **visible hai** aur **interactable hai**.
- Is case mein:
  - XPath usse locate kar lega ✓
  - But `.click()` ya `.send_keys()` tabhi kaam karega jab tum usse **scrollIntoView** kara lo.

Example:

Hidden Element (CSS se hide):

```
<input type="text" style="display: none;" id="email">
```

```
driver.find_element(By.ID, "email").send_keys("test@example.com")
#     ElementNotInteractableException
```

Scroll ke neeche element:

```
<input type="text" id="email">  <!-- visible but off-screen -->
```

```
email_input = driver.find_element(By.ID, "email")
driver.execute_script("arguments[0].scrollIntoView(true);", email_input)
email_input.send_keys("test@example.com")  #     Works now
```

How to Check if Element is Hidden

```
element.is_displayed()  # Returns False if hidden
```


Professional Explanation: Automation Testing Mindset + E-commerce Testing Flow + Reporting Anonymous August 07, 2025

58 Introduction

Bilkul! Yahaan maine aapka pura explanation **professionally**, **simple language**, aur **clear structure** me rewrite kiya hai, taaki kisi QA team, manager, ya documentation me easily use ho sake:

59 Automation Testing Overview

Automation Testing me sabse zaroori cheez hoti hai:

> **"Logic and Flow pe focus, phir proper Reporting."**

Is concept ko **2 main parts** me samajhte hain:

60 PART 1: E-commerce Website Test Flow (Step-by-Step)

Niche diya gaya flow ek **practical example** hai: login se lekar checkout tak ka automation sequence.

60.1 Test Steps:

1. **Launch Website**
   - Browser open karo aur website URL enter karo.
2. **Login**
   - Username & password fill karo, login button click karo.

- **Check:** Login successful? (e.g., dashboard ya welcome message dikh raha hai?)

3. **Product Search**
   - Search bar me product name daalo aur search button press karo.
   - **Check:** Relevant products dikh rahe hain?
4. **Select Product**
   - Search results me se ek product select karo.
   - **Check:** Product details page open hua?
5. **Add to Cart**
   - 'Add to Cart' button click karo.
   - **Check:** Product cart me successfully add hua?
6. **Open Cart**
   - Cart page open karo.
   - **Check:** Selected product cart me visible hai?
7. **Checkout Process**
   - Checkout button click karo.
   - Fill address, phone, etc.
   - Proceed to payment page.
8. **Payment**
   - Dummy payment details enter karo (test credentials).
   - Submit payment.
   - **Check:** Order placed confirmation aaya?
9. **Logout**
   - Logout button click karo.
   - **Check:** Logout successful?

10. **Step Logging**

    - Har step ka result log karo: **PASS** / **FAIL** / **ERROR**.

61 PART 2: Reporting Test Results to a File

61.1 Objective:

Test run hone ke baad ek file generate ho jisme **har step ka result** clearly likha ho (**PASS/FAIL** + message).

61.2 How to Do It:

1. **Print message after every step:**

    - e.g., `console.log("[STEP] Login - PASS")`

2. **Write all messages to a file:**

- Save all logs to a text file (e.g., `test_report.txt`).

3. **Include details:**

    - Add result status: `PASS`, `FAIL`, or `ERROR`, with short message if needed.

61.3 Example Report (test_report.txt)

```
[STEP] Open Website - PASS
[STEP] Login as testuser - PASS
[STEP] Search for product 'Shoes' - PASS
[STEP] Select first product - PASS
[STEP] Add product to cart - PASS
[STEP] Open cart and verify item - FAIL: Cart is empty.
[STEP] Checkout - SKIPPED (Previous step failed)
[STEP] Logout - SKIPPED
```

62 Summary for Professional Use:

- **Before test run:** Define each step and expected outcome.
- **During test:** Log step result with clear status.
- **After test:** Open the log file to review step-by-step results.

62.1 Industry Term:

This approach is called **"Stepwise Logging with Reporting"** — standard practice in QA automation.

63 Need Help With?

Let me know if you want:

- **A manual-style test case/report template**
- **Automation code logic or pseudocode for the above steps**
- **Tools/libraries suggestion for logging & reporting in automation frameworks (like Selenium, Playwright, etc.)**

64 Conclusion

This version is polished, clear, and suitable for professional documentation or knowledge-sharing. Let me know if you'd like it in PDF or formatted as a report.

==========

Python __init__.py, Package, Module Guide

__init__.py, Package, Module – Sab Doubts Clear (Hinglish Notes) Anonymous
August 12, 2025

65 Introduction

Yeh document Python ke **__init__.py**, **Package**, aur **Module** ke concepts ko Hinglish mein explain karta hai, har detail ke saath, bina kuch remove kiye. Selenium automation ke liye practical layout aur best practices bhi included hain.

66 Module vs Package – Seedhi si baat

- **Module:** ek single `.py` file jisme code hota hai.
  - **Example:** `cart_page.py` ek module hai.
- **Package:** ek folder jise Python import kar sakta hai (dotted path ke saath), jisme multiple modules ho sakte hain.
  - **Example:** `pages/` ek package, uske andar `user_panel_pages/` ek sub-package.

Think: Module = room, Package = ghar jisme multiple rooms.

67 __init__.py khali hoke bhi “special” kyun hai?

- Python interpreter file ka “**naam**” dekhta hai. Agar folder ke andar exact naam wali file `__init__.py` milti hai, to Python turant samajh jaata hai ki “yeh folder ek **package** hai.”

- Yeh **naming convention** aur language rule hai — content chahe empty ho, bas file ka naam magic hai.

- Jaise OS me `autorun.inf` ya Git me `.git` folder se special behavior trigger hota hai, waise hi Python me `__init__.py` se package behavior trigger hota hai.

Memory trick: “Nameplate” analogy.

- Ghar par nameplate lagate hi postman ko pata chal jaata hai yeh registered address hai.

- `__init__.py` = package ki nameplate. Khali ho sakti hai, par identity de deti hai.

68 __init__.py ka kaam exactly kya?

- **Package declare karta hai:** dotted imports enable.
  - `from pages.user_panel_pages.cart_page import CartPage`

- **Relative imports** allow karta hai package ke andar:
  - `from .login_page import LoginPage`

- **Public API** expose karne ka point:
  - `__init__.py` me `from .cart_page import CartPage` likh do, phir bahar se short import:
    * `from pages.user_panel_pages import CartPage`

- Tools/IDEs/linters ko structure clearly samajh aata hai.

69 "Namespace packages" kya hote hain? (Why sometimes it works without __init__.py)

- Python 3 ne ek feature diya: bina `__init__.py` ke bhi kuch folders "**namespace package**" jaisa behave kar sakte hain.
- Par yeh **unreliable** ho sakta hai:
  - Different machines/IDEs/CI me kabhi import ho, kabhi na ho.
  - Same-name folders merge ho sakte hain (confusing behavior).
  - Relative imports toot sakte hain.

Conclusion: Production/automation me predictable behavior chahiye. Isliye har importable folder me `__init__.py` lagana best practice.

70 Agar __init__.py NA ho to kya problems?

- `ModuleNotFoundError: No module named 'pages'`
- `Relative import error: Attempted relative import beyond top-level package`
- Local run ok, par CI/virtualenv/VS Code debug se run karoge to fail.
- "**Works on my machine**" type flaky issues.

71 Kab lagana chahiye?

- Jis folder ko import path ka hissa banana hai (jahan se aap dotted import karoge), wahan `__init__.py` rakho.
- **Typical automation project:**
  - `pages/`, `pages/user_panel_pages/`, `pages/admin_panel_pages/`
  - `utilities/`, `helpers/`, `config/`, `data/` (agar inhen import karte ho)
  - `tests/` me zaroori nahi, par subfolders me rakh doge to bhi theek; imports stable rehte hain.

72 Tumhare Selenium POM ke liye ready-to-use layout

```
pages/
  __init__.py
  user_panel_pages/
    __init__.py
    login_page.py
    cart_page.py
    main_dashboard_page.py
    main_zone_page.py
    buy_now_page.py
    my_order_page.py
  admin_panel_pages/
    __init__.py
    admin_login_page.py
    admin_dashboard_page.py
    orders_mgmt_page.py
tests/
  user_panel_tests/
    test_login.py
    test_cart_page.py
  admin_panel_tests/
    test_admin_login.py
conftest.py (common fixtures)
Optional: pytest.ini
```

Imports after split:

- `from pages.user_panel_pages.cart_page import CartPage`
- `from pages.user_panel_pages.login_page import LoginPage`
- `from pages.admin_panel_pages.admin_login_page import AdminLoginPage`

Shortcut via __init__.py export:

- `pages/user_panel_pages/__init__.py`:
  - `from .cart_page import CartPage`
  - `from .login_page import LoginPage`
- Ab tests me:

- `from pages.user_panel_pages import CartPage, LoginPage`

73 Pytest discovery quick rules

- **Test files:** `test_*.py` ya `*_test.py`.
- `__init__.py` not required for discovery, par allowed hai.
- Hamesha project root se run karo:
  - `python -m pytest -q`
- `sys.path.append` hacks avoid karo; packages sahi banane se zarurat nahi padti.

74 Mini FAQ (rapid fire)

- **Q:** `__init__.py` ka size? Content?
  - **A:** Empty chalega. Chahe to public API expose karne wali imports likh do.
- **Q:** Kya har folder me chahiye?
  - **A:** Sirf un folders me jisse import karna hai.
- **Q:** Django me kyon notice nahi hota?
  - **A:** `startproject/startapp` automatically `__init__.py` bana deta hai.

75 60-second checklist (note me chipka lo)

- Importable folder? Put `__init__.py`.
- Use dotted imports from project root package.
- Relative imports sirf package ke andar (`.something`).
- Run tests from root (`python -m pytest`).
- Avoid `sys.path.append`.
- Split user/admin pages into sub-packages with their own `__init__.py`.