**async ({ browser }) => { ... } vs async ({ page}) => { ... } vs async ({ context}) => { ... }**

In Playwright, the syntax async ({ browser }) => { ... }, async ({ page }) => { ... }, and async ({ context }) => { ... } refers to the use of **fixtures** provided by Playwright's built-in test runner, **Playwright Test** (@playwright/test). These fixtures (browser, context, page) represent different levels of browser-related objects, each providing a different type of resource for the test. Let's break down the difference between each.

**Understanding Each Fixture Syntax**

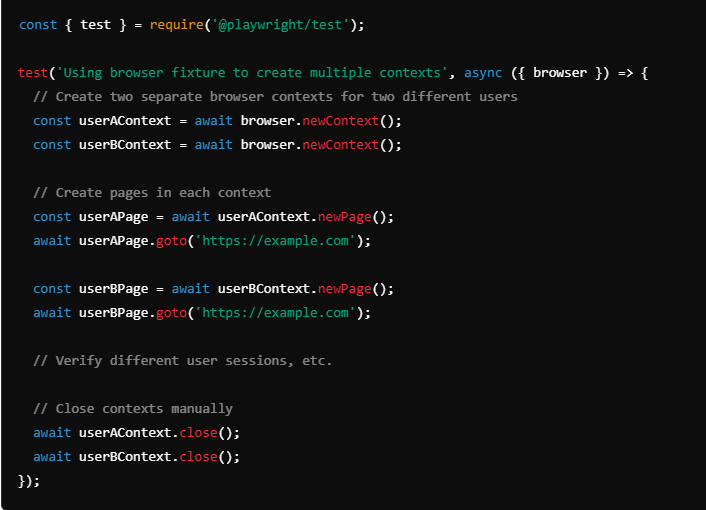
**1. async ({ browser }) => { ... }**

* **browser Fixture**:
  + The browser fixture represents the **browser instance** itself.
  + You use this fixture to create **contexts** manually when you need more control over the session.
  + You are responsible for creating **browser contexts** and managing them manually (including closing them).

**When to Use:**

* When you need to create **multiple contexts** manually.
* When you want to test **different user sessions** or **isolate** sessions between contexts.

**Example:**



* **Control Level**: Full control over the **browser context**.
* **Responsibility**: You need to **manually create** contexts and **close** them.

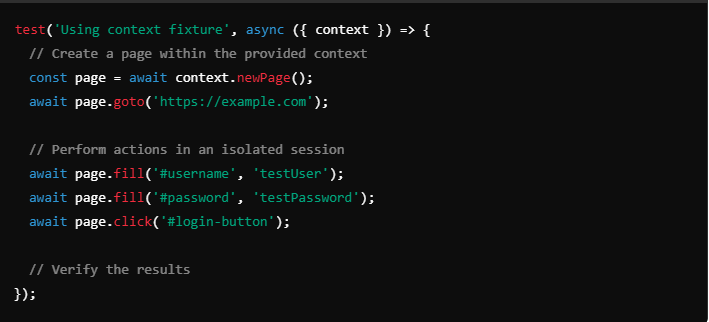
**2. async ({ context }) => { ... }**

* **context Fixture**:
  + The context fixture provides a **browser context** directly.
  + A **browser context** is like an isolated session in the browser, similar to an incognito window or separate user profile.
  + It is automatically managed by Playwright Test, which means the setup and teardown are handled for you.

**When to Use:**

* When you want to have an **isolated session** but don't need to manage multiple contexts.
* It’s useful for testing scenarios where each test should run in a completely isolated environment, but without the overhead of creating contexts manually.

**Example:**



* **Control Level**: Medium. You have a **browser context**, but Playwright manages setup and teardown.
* **Responsibility**: You do not need to **manually close** the context; Playwright takes care of it.

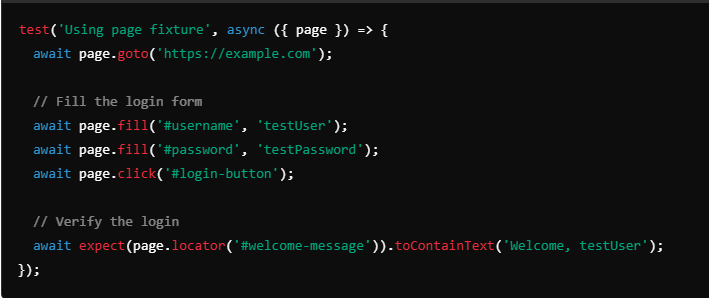
**3. async ({ page }) => { ... }**

* **page Fixture**:
  + The page fixture represents a **new page (tab)** within a **browser context**.
  + This fixture is the highest-level fixture, abstracting away both the **browser instance** and the **context** creation.
  + Playwright Test automatically creates a **browser**, **context**, and **page**, managing all the setup and teardown for you.

**When to Use:**

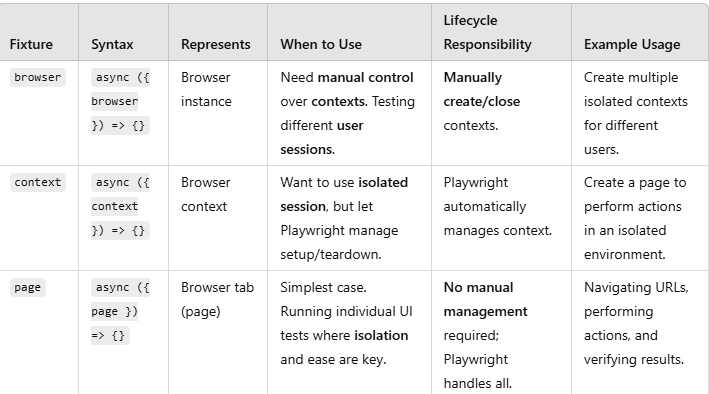
* When you want to keep your tests simple.
* For **most UI tests**, where each test requires a clean slate—a new **browser context** and **page**.
* It is the most convenient way to perform operations on a page without worrying about creating or managing contexts or browsers.

**Example:**



* **Control Level**: Low. You work directly with a **page** and don’t need to worry about browser or context creation.
* **Responsibility**: None for setup or teardown; Playwright automatically handles everything.

**Key Differences & Use Cases**



**Summary**

* **browser Fixture**:
  + Full control over the **browser** and **contexts**.
  + Suitable for creating multiple **isolated user sessions**.
  + **You manage the lifecycle**.
* **context Fixture**:
  + Provides a **browser context** that is automatically managed.
  + Suitable when you want each test to run in a **separate isolated environment**.
  + **Playwright manages lifecycle**.
* **page Fixture**:
  + Provides a **browser page** for testing.
  + Most commonly used fixture for simplicity and isolation.
  + **Playwright manages everything** from browser to context to page creation.

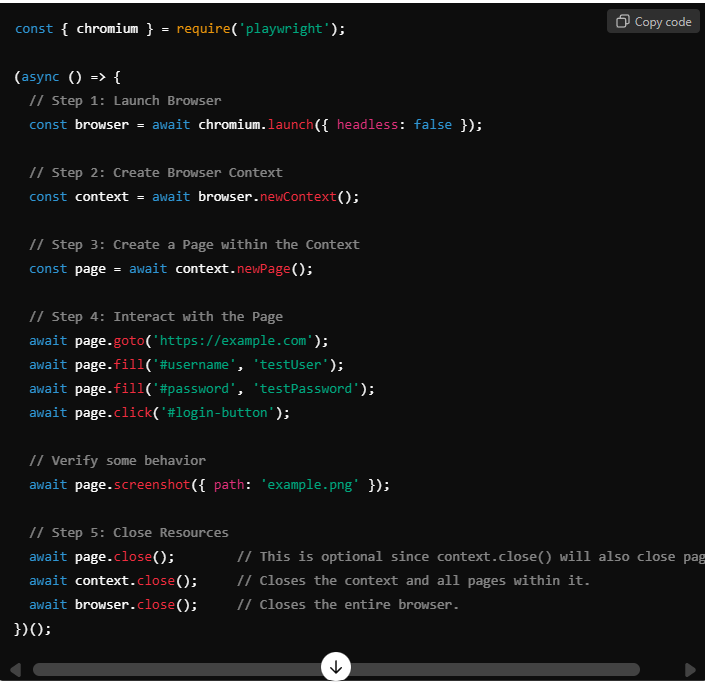
The sequence and hierarchy of **Browser**, **Context**, and **Page** in Playwright:

**Sequence of Browser, Context, and Page**

1. **Browser** (browser):
   * The **browser** is the highest-level instance.
   * It represents a physical browser application like Chromium, Firefox, or WebKit.
   * You first **launch** a **browser** using chromium.launch(), firefox.launch(), or webkit.launch().
2. **Context** (context):
   * A **browser context** is like an isolated session (similar to an incognito or private window).
   * You create a **context** from the **browser** instance.
   * Each **context** provides a completely isolated environment where cookies, cache, and storage are not shared.
   * **Multiple contexts** within a single **browser** instance help you simulate different users or sessions without interference.
3. **Page** (page):
   * A **page** represents a tab within the **browser context**.
   * You create a **page** from a **context**.
   * Multiple **pages** can be created within a single **context**. They will share cookies, cache, and storage as they belong to the same session.

**Example of the Sequence**

Here is a code example that shows the sequence in which you create and use the **browser**, **context**, and **page**:



**Sequence Hierarchy (Summary):**

1. **Browser** (browser)
   * Top-level instance representing the browser.
   * **Launch the browser** (await chromium.launch()) to get started.
2. **Browser Context** (context)
   * Created from a **browser**.
   * Represents an **isolated session**.
   * **Multiple contexts** can be created from the same **browser**.
   * **Create context** (await browser.newContext()) for isolated environments (e.g., different user profiles).
3. **Page** (page)
   * Created from a **context**.
   * Represents a **tab** in the browser.
   * **Multiple pages** can be created from the same **context**.
   * **Create page** (await context.newPage()) to interact with a website

