**Slide 1: Title Slide**

* **Title**: "Session 1: Introduction to Playwright"
* Subtitle: "Kickstarting Playwright Test Automation"
* Your name, date, and session details

**Slide 2: What is Playwright?**

* **Definition**: An open-source, cross-platform, end-to-end testing tool.
* **Highlights**:
  + Fast, reliable, and robust test automation for modern web apps.
  + Supports all major browsers: Chromium, WebKit, and Firefox.
  + Cross-platform: Runs on Windows, macOS, and Linux.
* Visual: Diagram of Playwright’s ecosystem.

**Slide 3: Why Playwright?**

* Key advantages over other tools:
  + **Parallel Testing**: Built-in test execution parallelism.
  + **Auto-Wait Mechanism**: Handles dynamic content seamlessly.
  + **Test Generators**: Record and generate tests easily.
  + **Built-in Reporting**: Rich test reporting capabilities.
  + Supports mobile emulation, video recordings, and screenshots.

**Slide 4: Playwright vs Competitors**

* **Comparison Table**:

| **Feature** | **Playwright** | **Selenium** | **Cypress** | **Puppeteer** |
| --- | --- | --- | --- | --- |
| Browser Support | Chromium, WebKit, Firefox | Major Browsers | Only Chromium-based | Chromium only |
| Language Support | JavaScript, Python, C#, Java | Many | JavaScript only | JavaScript |
| Auto-wait | Yes | No | No | No |
| Mobile Emulation | Yes | Limited | No | Limited |

* Conclusion: "Playwright is a modern alternative designed for the future of testing."

**Slide 5: Course Overview**

* **Objective**: Master Playwright for comprehensive web automation.
* **What to Expect**:
  + Fundamentals of Playwright.
  + Writing test scripts and creating test suites.
  + Advanced features: mocking, tracing, and auto-healing.
  + Hands-on practice sessions.
* **Structure**:
  + 12 sessions (10 theory, 2 hands-on).
  + Duration: 45 minutes per session.

**Slide 6: Setting Up Playwright**

* **Prerequisites**:
  + Node.js (version XX or higher)
  + Basic knowledge of JavaScript or TypeScript.
* **Installation Steps**:
  + Install Node.js.
  + Run npm init playwright@latest.
  + Set up the testing directory structure.
  + Verify installation: Run the sample script.
* Visual: Screenshot of terminal commands and expected outputs.

**Slide 7: First Look at Playwright**

* Running the sample test:

bash

CopyEdit

npx playwright test

* Overview of the default folder structure:
  + tests/
  + playwright.config.js
  + Example script: Open a website, perform a search, validate results.
* Visual: Highlight key parts of the sample test script.

**Slide 8: Key Takeaways**

* Playwright is fast, reliable, and feature-rich.
* Installation and setup are straightforward.
* Course structure designed for theoretical and practical learning.

Here’s a comprehensive guide for setting up the Playwright environment from scratch for **Windows**, **Mac**, and **Linux**. For Mac and Linux users, I’ve included the necessary Bash commands.

**Step 1: Install Node.js**

Playwright requires Node.js (minimum version 14). You need to install Node.js on your system first.

**For Windows:**

1. Visit the [Node.js official website](https://nodejs.org/).
2. Download the **LTS version** (recommended for most users).
3. Run the installer and follow the on-screen instructions.
4. During installation, check the option to add Node.js to the system PATH.
5. Verify the installation:
   * Open **Command Prompt** or **PowerShell**.
   * Run:
   * node -v
   * npm -v
   * This should return the Node.js and npm versions.

**For Mac:**

1. Open the **Terminal**.
2. Install Node.js using **Homebrew**:
3. brew install node
4. Verify the installation:
5. node -v
6. npm -v

If you don't have Homebrew installed, first install it:

/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"

Alternatively, you can download Node.js directly from the [Node.js website](https://nodejs.org/) and follow the installer instructions.

**For Linux:**

1. Open the **Terminal**.
2. Update your system packages:
3. sudo apt update && sudo apt upgrade -y
4. Install Node.js (using NodeSource):
5. curl -fsSL https://deb.nodesource.com/setup\_16.x | sudo -E bash -
6. sudo apt install -y nodejs
7. Verify the installation:
8. node -v
9. npm -v

Alternatively, use a package manager specific to your distribution.

**Step 2: Install a Code Editor (Optional)**

* Download and install [Visual Studio Code](https://code.visualstudio.com/) as it’s widely used for Playwright projects.

**Step 3: Set Up a Playwright Project**

1. Open a terminal (Command Prompt, PowerShell, or Bash).
2. Create a new project directory:
3. mkdir playwright-project
4. cd playwright-project
5. Initialise a new Node.js project:
6. npm init -y

This will create a package.json file in your project directory.

1. Install Playwright:
2. npm install playwright

For browsers (if you want to ensure all necessary browsers are downloaded):

npx playwright install

**Step 4: Verify Playwright Installation**

1. Check if Playwright is installed:
2. npx playwright --version
3. Create a test script:
4. npx playwright codegen

This will open a browser window where you can record interactions to generate Playwright code.

**Step 5: Optional - Install Playwright Test Framework**

If you plan to use Playwright for testing, install the Playwright test runner:

npm install @playwright/test

Create a configuration file:

npx playwright test init

**Step 6: Run a Sample Test**

1. Create a new test file, e.g., example.spec.js:
2. const { test, expect } = require('@playwright/test');
3. test('basic test', async ({ page }) => {
4. await page.goto('https://example.com');
5. const title = await page.title();
6. expect(title).toBe('Example Domain');
7. });
8. Run the test:
9. npx playwright test

**Step 7: Update PATH for Global Commands (Optional for Linux/Mac)**

If you want to use playwright globally:

echo 'export PATH=$PATH:$(npm bin -g)' >> ~/.bashrc

source ~/.bashrc

Let me know if you’d like me to expand on any part!

**1. What is Playwright?**

**Talking Points**

* "Playwright is an open-source automation testing tool designed to test modern web applications. It’s built by Microsoft and supports end-to-end testing of web apps."
* "One of its standout features is its cross-browser support. You can test your application on Chromium, WebKit, and Firefox without additional plugins or tools."
* "It’s designed to be fast and reliable, with built-in features like auto-waiting, parallel test execution, and detailed tracing for debugging."

**Expand with Examples**

* "For example, Playwright can handle dynamic web pages where content loads asynchronously. Unlike older tools, you don’t have to add explicit waits—Playwright handles that for you."

**2. Why Playwright?**

**Talking Points**

* "Playwright is gaining popularity because it addresses limitations of traditional tools like Selenium and newer ones like Cypress."
* Highlight these advantages:
  1. **Parallel Testing:** "Playwright can execute tests across multiple browsers in parallel, saving significant time during test execution."
  2. **Auto-Wait:** "For dynamic content, Playwright intelligently waits for elements to be visible or actionable, reducing flaky tests."
  3. **Test Generators:** "With the Playwright CLI, you can record and generate tests by interacting with your app in a browser."
  4. **Built-in Reporting:** "Generate video recordings, screenshots, and detailed trace logs for debugging failing tests."

**Example:**

* "Imagine testing a single-page application with Selenium. You might run into issues with elements loading asynchronously. Playwright avoids this problem by waiting automatically, saving time and effort."

**3. Playwright vs Competitors**

**Talking Points**

* Create a table to explain the comparison. Walk through each feature and explain its significance:
  + **Browser Support:** "Playwright supports WebKit, which powers Safari, along with Chromium and Firefox. Selenium also supports all browsers, but Cypress and Puppeteer are limited."
  + **Auto-Wait:** "Playwright’s built-in wait mechanism is unique. Unlike Selenium, you don’t have to add explicit waits for elements."
  + **Language Support:** "Playwright supports multiple languages like JavaScript, TypeScript, Python, C#, and Java. Cypress only supports JavaScript."
  + **Mobile Emulation:** "Playwright can emulate mobile devices out of the box, a feature lacking in Cypress."

**Audience Engagement:**

* Ask: "Has anyone faced challenges with tools like Selenium or Cypress? How did you solve them?"
* Discuss how Playwright overcomes those issues.

**4. Course Overview**

**Talking Points**

* "Over the next 12 sessions, we’ll cover Playwright from the ground up. By the end, you’ll be able to write robust automation scripts for modern web apps."
* Briefly introduce the structure:
  + "The first few sessions will focus on basics like setup and writing simple tests."
  + "Later, we’ll dive into advanced topics like mocking, tracing, and handling flaky tests."
  + "Finally, you’ll have two hands-on sessions to practice what you’ve learned."
* Highlight the importance of practical learning: "We’ve included hands-on sessions because the best way to learn is by doing!"

**5. Setting Up Playwright**

**Talking Points**

1. **Prerequisites:**
   * "Before we start, ensure you have Node.js installed on your system. Playwright is a Node.js-based tool, so it’s essential."
   * "A basic understanding of JavaScript or TypeScript will also help you follow along."
2. **Installation Steps (Live Demo):**
   * "Let’s set up Playwright together. Open your terminal and run the following commands:

bash

CopyEdit

npm init playwright@latest

```"

* + Explain the options: "This command creates a new Playwright project with a ready-to-use folder structure."
  + Show how to navigate the structure:
    - tests/: Where your test scripts live.
    - playwright.config.js: Configuration for browsers, timeouts, etc.

1. **Run a Sample Test (Demo):**
   * Run:

bash

CopyEdit

npx playwright test

* + "This command runs the default test included with Playwright. You’ll see browser windows open and tests execute automatically."
  + Point out the output: "The results show whether each test passed or failed, along with detailed logs."

**6. First Look at Playwright’s Capabilities**

**Talking Points**

* "Now that we’ve set it up, let’s explore Playwright’s features."
* Walk through an example script (on slides or live):

javascript

CopyEdit

const { test, expect } = require('@playwright/test');

test('Visit Google and search', async ({ page }) => {

await page.goto('https://google.com');

await page.fill('input[name="q"]', 'Playwright test automation');

await page.press('input[name="q"]', 'Enter');

await expect(page).toHaveTitle(/Playwright test automation/);

});

* Explain key commands:
  + page.goto: Navigates to a URL.
  + page.fill: Enters text into a field.
  + expect: Verifies test assertions.

**7. Audience Engagement**

**Activity:**

* Ask attendees to set up Playwright on their own machines. Guide them through any issues.
* Challenge: "Modify the sample test to visit another website and validate its title."

**8. Recap and Key Takeaways**

**Talking Points**

* "Today, we covered an introduction to Playwright, its unique features, and how it compares to other tools."
* "We also walked through the setup process and ran a sample test."
* "In the next session, we’ll dive deeper into writing scripts and exploring Playwright’s core APIs."

**Audience Interaction:**

* Ask: "Do you feel confident setting up Playwright on your own? Any questions before we wrap up?"