JavaScript Essentials for Playwright

A practical reference for mastering JavaScript fundamentals used in Playwright automation.

1. Variables

Use let for variables that may change, const for ones that don't.

Playwright use: Playwright use: Store test data, selectors, URLs, or results.

```
const url = "https://yourapp.com"; // URL won't change
let loginAttempts = 0;             // This value may change

await page.goto(url);
loginAttempts++;
```

2. Data Types

Use string, number, boolean, etc. to store information.

Playwright use: Playwright use: Pass data to forms, compare results, or assert conditions.

```
let expectedTitle = "Login";                        // string
let itemCount = 3;                                  // number
let isVisible = await page.isVisible("#login");  // boolean
```

3. Operators

Use arithmetic, comparison, and logical operators to work with values and conditions.

Playwright use: Playwright use: Validate results, write assertions, or make decisions in tests.

```
if (itemCount > 0 && isVisible) {
  console.log("Ready to proceed!");
}
```

4. Template Literals

Use backticks (`) and ${} to build strings with variables.

Playwright use: Playwright use: Build dynamic selectors or test data.

```
let userId = 123;
await page.click(`#user-row-${userId}`); // selector with variable
```

5. Control Structures

Use if, else, and switch to handle decision-making in code.

Playwright use: Playwright use: Handle conditional test flows or optional steps.

```
if (await page.isVisible("#promo")) {

  await page.click("#promo");

}
```

6. Loops

Use for, for...of, and .forEach() to repeat actions for multiple items.

Playwright use: Playwright use: Perform actions on lists of elements or data.

```
const items = await page.$$(".item");

for (const item of items) {

  await item.click();

}
```

7. Functions

Use regular or arrow functions for reusable logic.

Playwright use: Playwright use: Reuse steps and organize code for repeated actions.

```
async function login(page, username, password) {

  await page.fill("#user", username);

  await page.fill("#pass", password);

  await page.click("#login");

}

await login(page, "admin", "admin123");
```

8. Objects & Arrays

Use objects for related data, arrays for lists.

Playwright use: Playwright use: Organize test data, pass configs, or check API responses.

```
const user = { name: "Alex", email: "alex@email.com" };

const users = ["admin", "user1", "guest"];


for (const u of users) {
  await page.fill("#user", u);
}
```

9. Key Array Methods

Use .map(), .filter(), .forEach(), .find() to process lists.

Playwright use: Playwright use: Work with collections of elements or data.

```
const checkboxes = await page.$$("input[type=checkbox]");
await Promise.all(checkboxes.map(cb => cb.check()));
```

10. Error Handling

Use try/catch to handle errors and prevent test crashes.

Playwright use: Playwright use: Make tests resilient and manage failures gracefully.

```
try {
  await page.click("#dangerous-action");
} catch (err) {
  console.log("Button not found, skipping step.");
}
```

11. Async/Await

Use async/await to handle operations that take time.

Playwright use: Playwright use: Wait for navigation, network, or UI actions to complete.

```
await page.goto("https://playwright.dev");
const text = await page.textContent("h1");
```

12. ES6+ Features

Use destructuring, spread, and modern syntax for cleaner code.

Playwright use: Playwright use: Make your tests shorter and easier to read.

```
const { username, password } = userData;      // Destructuring
const allUsers = [...adminUsers, ...guests];  // Spread operator
```

13. Classes

Use class to organize code for reusable helpers or page objects.

Playwright use: Playwright use: Structure your test code using the Page Object Model for better *maintainability.*

```
class LoginPage {
  constructor(page) {
    this.page = page;
  }
  async login(user, pass) {
    await this.page.fill("#user", user);
    await this.page.fill("#pass", pass);
    await this.page.click("#submit");
  }
}

const loginPage = new LoginPage(page);
await loginPage.login("admin", "admin123");
```

14. Closures

Functions can remember variables from their outer scope.

Playwright use: Playwright use: Useful in advanced helpers or for encapsulating state in utilities.

```
function counter() {
  let count = 0;
  return function() {
    count++;
    return count;
  };
}

const next = counter();
console.log(next()); // 1
console.log(next()); // 2
```

15. Modules

Use import and export to organize your code across multiple files.

Playwright use: Playwright use: Split large test suites and helpers for clean project structure.

```
// loginHelper.js
export async function login(page, username, password) {
  await page.fill("#user", username);
  await page.fill("#pass", password);
  await page.click("#login");
}

// test.js
import { login } from "./loginHelper.js";
await login(page, "admin", "admin123");
```

16. Advanced Promises/Async Patterns

Use .then(), .catch(), and chaining for flexible async flows.

Playwright use: Playwright use: Handle multiple async steps or advanced error handling when needed.

```
fetch("https://api.com")
  .then(res => res.json())
  .then(data => console.log(data))
  .catch(err => console.log(err));
```

17. Functional Programming

Use .map(), .reduce(), and .filter() for concise data transformations.

Playwright use: Playwright use: Process or assert lists of elements or API data in tests.

```
// Get all visible labels' text
const labels = await page.$$("label");
const texts = await Promise.all(labels.map(l => l.textContent()));
const longLabels = texts.filter(text => text.length > 10);
```