1. What is the basic difference between an array and a tuple in TypeScript?

An array holds multiple values of the same type with flexible length, while a tuple enforces a fixed structure where each position has a predefined type.

Side-by-side comparison

Feature	Array	Tuple
Length	Dynamic	Fixed
Types	Usually same type	Different types per index
Order matters	Less strict	Very strict
Type safety	Lower	Higher

2. How is any different from unknown, and when would you use each?

any disables TypeScript’s type checking completely, so you can perform any operation on the variable without compiler errors. This is risky because issues are only caught at runtime and can lead to flaky Playwright tests.

unknown is a type-safe alternative to any. It allows assigning any value, but forces explicit type checks before usage. This ensures better validation and safer code.

In Playwright, I prefer unknown when handling API responses, JSON data, utility functions, or error handling, because the data shape may not be guaranteed. I use any only in rare cases like legacy code migration or poorly typed third-party libraries.

This approach improves test stability, readability, and long-term maintainability.

any disables TypeScript’s type checking completely, so you can perform any operation on the variable without compiler errors. This is risky because issues are only caught at runtime and can lead to flaky Playwright tests.

unknown is a type-safe alternative to any. It allows assigning any value, but forces explicit type checks before usage. This ensures better validation and safer code.

In Playwright, I prefer unknown when handling API responses, JSON data, utility functions, or error handling, because the data shape may not be guaranteed. I use any only in rare cases like legacy code migration or poorly typed third-party libraries.

This approach improves test stability, readability, and long-term maintainability.

In playwright, we can reuse common methods across multiple test files by organizing reusable logic into Page Objects, Utility Classes, Custom Fixtures, or Helper Functions. The choice depends on what you're reusing.

3. How do you reuse common methods across multiple test files?

1. Page Object Model

Used for reusable UI actions.

2. Utility / Helper Functions

Used for reusable generic functions like waits, data generation, API calls.

3. Custom Fixtures (Best for shared setup like login)

Used when multiple tests need same setup.

4. Base Class Approach (For advanced frameworks)

Create a base class with common methods.

4. What happens when one API endpoint redirects to another?

When one API endpoint redirects to another in Playwright, the request automatically follows the redirect, and you can capture both the redirect response and the final response depending on how you handle it.

This applies to both:

page (UI network requests)

APIRequestContext (API testing using request fixture)

5. What kind of values do you usually keep in environment config files in playwright?

In Playwright, environment config files are used to store **environment-specific and sensitive values** so tests can run across different environments (dev, QA, staging, prod) without code changes.

I use .env files along with Playwright config to manage environment-specific values like baseURL, credentials, and API tokens. This allows easy switching between environments and supports CI/CD pipelines."

Here’s what we typically store:

1. Base URLs (Most Common)

Different environments have different URLs.

2. Login Credentials

Avoid hardcoding credentials in test files.

3. API Endpoints

Different environments may use different API URLs.

4. Environment Name

Useful for switching configs dynamically.

5. Timeouts

Different environments may need different timeout values.

6. Browser Configuration

7. Test Data / Test Users

8. Tokens / API Keys (secure way via .env)