<u>**Brief recap of JEST**</u>

Jest is a library you can use from npm, let's us run tests and includes an assertion library. That means we can run comparisons and shows us what is failing and passing.

Remember to use your basic knowledge when reading docs – anything with () is a callable (normally function), anything with .dot notation is some sort of object with properties (or methods), what is being imported or not.

Creating a *filename*.test.js file tells Jest where the tests are

Human-readable message describing what the test is – **Given, When, Then** (e.g. Given a variable, when called then the function returns something)

Three A – **Arrange, Act, Assert** (do all this inside the inline function inside the overall test function, after composing your human-readable message saying what the test does)

Basic plan for any testing:

- Import function that you want to test (export from original file and import in test file)
- Optionally: explicitly import globals from Jest that we're going to use in this test:
  - test
  - expect
  - describe
  - e.g. import { test, expect, describe } from "@jest/globals";
- Write your test:
  - Call test function
  - Write human-readable message
  - Declare inline function
  - ARRANGE – set variables needed
  - ACT – call function passing in variables
  - ASSERT – expect(actual).toBe(expected) (or whatever you want actual and expected to be)

<u>**Test Driven Development (TDD)**</u>

https://www.browserstack.com/guide/what-is-test-driven-development

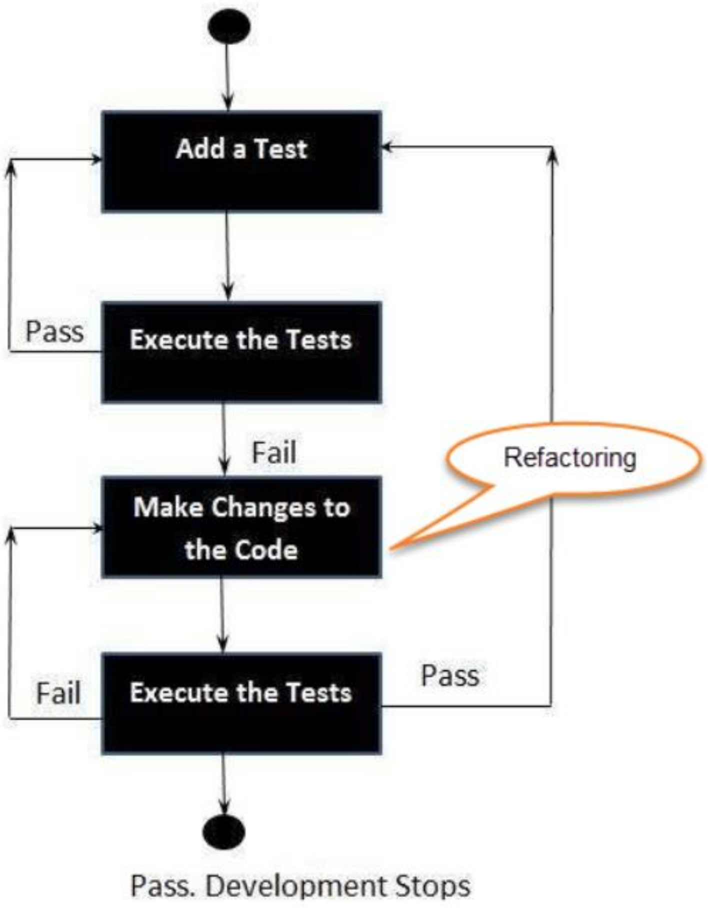
Try to get in to the mindset of writing testable code.

Basic process:
- Write the tests before you write the actual code
- So the test should always fail first time (a.k.a. Red Stage)
- Then write the minimum amount of code to pass the test (a.k.a. Green Stage)
- Then you can refactor (either your code or the test) (a.k.a. Refactor Stage)

- Add more tests and repeat!

Three phases of Test Driven Development:

- Create precise tests: Developers need to create precise unit tests to verify the functionality of specific features. They must ensure that the test compiles so that it can execute. In most cases, the test is bound to fail. This is a meaningful failure as developers are creating compact tests based on their assumptions of how the feature will behave.
- Correcting the Code: Once a test fails, developers need to make the minimal changes required to correct the code so that it can run successfully when re-executed.
- Refactor the Code: Once the test runs successfully, check for redundancy or any possible code optimizations to enhance overall performance. Ensure that refactoring does not affect the external behaviour of the program.



https://www.ibm.com/garage/method/practices/code/practice_test_driven_development/

https://www.youtube.com/watch?v=Jv2uxzhPFI4

Wallaby – paid extension with free trial in VS Code that tells you whether your tests are passing directly without needing to run it and checking the console

PM sessions were on Cypress – see Guest Talk