Testing Exercises

why do we write tests?

- To save time doing the manual labor
- Coverage ensures that all code paths are tested
    - It saves us from making mistakes
- If tests run on CI, then no bad build will be delivered to the end-user.
- It helps us refactor or add features with confidence.
- Manually testing, you would test less. But automated tests can run on every change and make sure things work.
- Tests serve as the best documentation as you don't have to work extra to keep them up to date. In summary, they don't go stale.
- You'll end up writing better code. Mostly when you do Test Driven Development. RGR (Red Green Refactor)
    - Modularity - module
    - Pure Functions

why did this change happen?

WATERFALL was what we used to do previously. It would take months to get the software out.

AGILE

- DevOps - you push your code to source control —> Take it, package it, and deliver it to the user completely automated
    - This increased the shipping velocity
        - You needed more confidence
            - automated tests

some testing terms

RGR - Red Green Refactor

TDD - Test Driven Development

how should we think about testing?

- A lot of devs think testing is HARD.
- A lot of devs think testing is way too EASY.
- FOR USER ←— Program ←— Tests
- Program —> Makes User’s life easy
- Tests —> Makes Programmer’s life easy

Deploy on FRIDAYS! YAY!

Okay, I’m joking.

let’s code some tests

ex01: a simple test

challenge

Create a basic testing framework in JavaScript.

1. Start by writing an `add` function that adds two numbers.
2. Next, build a `test` function that takes a test name and a callback. The callback should return a boolean value to indicate if the test passed or failed. Log the test name and the test result.
3. Finally, write a test case using the `test` function to check if the `add` function correctly adds two numbers. Test if the sum of 5 and 7 is equal to 12.

solution

https://codesandbox.io/s/jolly-tree-7pjvcw

ex02: first test in jest

challenge

https://codesandbox.io/s/first-jest-test-challenge-9fxynz

solution

https://codesandbox.io/s/first-jest-test-solution-znxrp7

ex02.a let's write some more tests

documentation

Find the appropriate matcher for the current use case https://jestjs.io/docs/using-matchers

challenge

https://codesandbox.io/s/utils-test-challenge-qdzslz

test cases

solution

https://codesandbox.io/s/utils-test-solution-zgz2zh

02.1 capitalize

Description: Write tests for a `capitalize` utility function that takes a string as input and returns the same string with the first letter capitalized.

```
function capitalize(str) {
  return str.charAt(0).toUpperCase() + str.slice(1);
}
```

COPY

02.2 filterEven

Create tests for a `filterEven` utility function that filters out even numbers from an array.

```
function filterEven(arr) {
  return arr.filter(num => num % 2 === 0);
}
```

COPY

02.3 double

Write tests for a `double` utility function that doubles each number in an array.

```
function double(arr) {
  return arr.map(num => num * 2);
}
```

COPY

02.4 Reducer (sum)

Description: Create tests for a `sum` utility function that calculates the sum of all numbers in an array.

```
function sum(arr) {
  return arr.reduce((acc, curr) => acc + curr, 0);
}
```

COPY

Test Case	Input	Expected Output
Test 1	[1, 2, 3, 4, 5]	15

02.5 Reducer (average)

Write tests for an `average` utility function that calculates the average of all numbers in an array.

```
function average(arr) {
  const sum = arr.reduce((acc, curr) => acc + curr, 0);
  return sum / arr.length;
}
```

COPY

ex03: testing reducer

challenge

Write a test for this reducer.

- Add two products and test that the output is as expected times

https://codesandbox.io/s/testing-reducer-challenge-j9cfwy

solution

https://codesandbox.io/s/testing-reducer-solution-9442j4

ex04: let's try TDD

Do these by writing tests first. Your test will fail and then you need to write the code to make the test pass.

ex04.a

In the previous reducer implement the functionality to remove an item

challenge

https://codesandbox.io/s/tdd-remove-item-challenge-f9lt2y

solution

https://codesandbox.io/s/tdd-remove-item-solution-cm7yxw

ex04.b

Change the quantity of individual items: just do INCREMENT

```
const action = {
      type: "INCREMENT_QUANTITY",
      payload: {
        itemId: "1236"
      }
    };
```

COPY

- `individualQuantity` increases
- `totalQuantity` is untouched
- but the `overallPrice` increases

challenge

https://codesandbox.io/s/tdd-add-quantity-challenge-m70t0u

solution

https://codesandbox.io/s/tdd-add-quantity-solution-d10x1

h/w: practice setup/teardown

documentation

https://jestjs.io/docs/setup-teardown

challenge

Try each setup/teardown.

Just print `console.log("trying beforeEach")` and similar things to know that these exist.

The best practice is to use `beforeEach()` and `beforeAll()` and avoid `after` calls.

ex06: talk about folder structure

- Some people favor `__tests__`
- I favor, colocation. Easy to move the entire feature together.

ex07: best practices for testing

- Fast
  - easier to run
- The order shouldn't matter
- Independent
- Automatic (obviously)
- Readable
  - AAA → Arrange Act Assert
- One test should test one thing

ex08: code coverage

- Supported by tool
- Few words on coverage
  - Don't chase 100%
  - Instead, make sure your P0 scenarios are tested
  - Anything above 85% is GOLD!

ex09: Debugging

- `console.log()`
- `test.only`

ex10: the way forward

- testing-library/react
  - Integration Testing

- https://testing-library.com/docs/react-testing-library/intro
- already included with CRA
- Read KCD blogs on testing

- Cypress
  - e2e testing
  - Most used tool
  - Should explore one of the projects once you're in the job and you have enough time on your hand

test cases

capitalize

Test Case	Input	Expected Output
capitalizes the first letter of a string	"hello"	"Hello"
capitalizes the first letter of a string	"world"	"World"
returns an empty string if input is empty	""	""

double

Test Case	Input	Expected Output
doubles each number in an array	[1, 2, 3, 4, 5]	[2, 4, 6, 8, 10]
returns an empty array if input is empty	[]	[]

filterEven

Test Case	Input	Expected Output
filters out even numbers from an array	[1, 2, 3, 4, 5]	[2, 4]
returns an empty array if no even numbers are found	[1, 3, 5, 7, 9]	[]

sum

Test Case	Input	Expected Output
calculates the sum of numbers in an array	[1, 2, 3, 4, 5]	15
returns 0 for an empty array	[]	0

average

Test Case	Input	Expected Output
calculates the average of numbers in an array	[1, 2, 3, 4, 5]	3
returns NaN for an empty array	[]	NaN