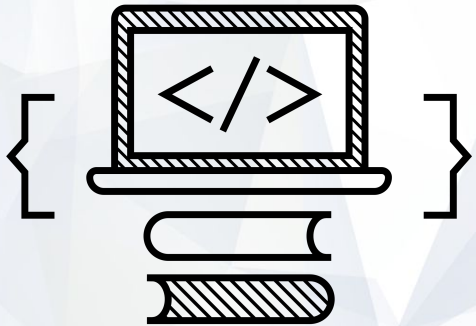
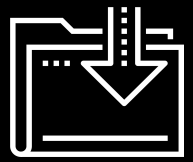


Behavior-Driven Development

Skills Bootcamp in Front-End Web Development

Lesson 14.2



© 2022 edX Boot Camps LLC. Confidential and Proprietary. All Rights Reserved.

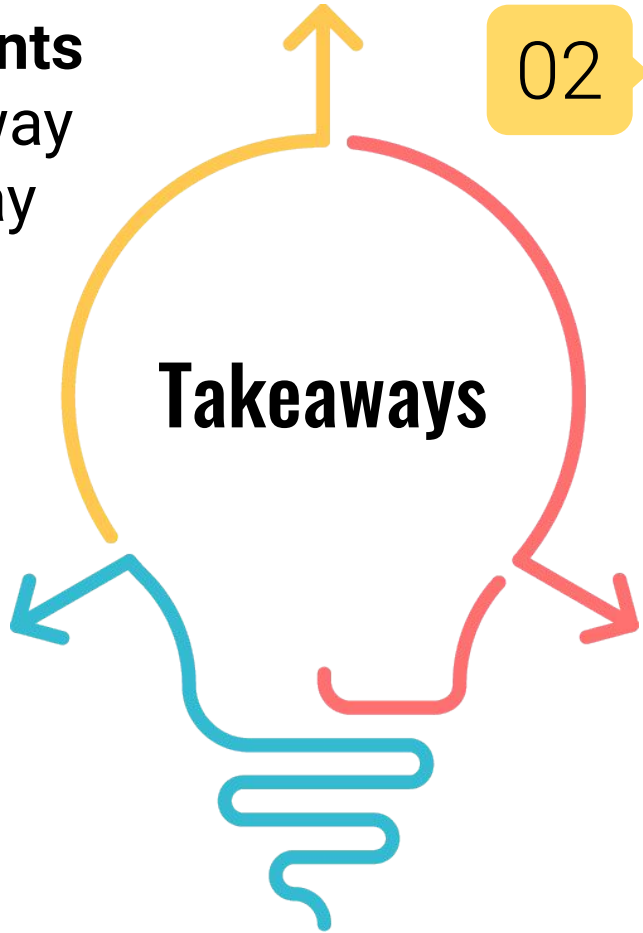
WELCOME

Learning Objectives

By the end of class, you will be able to:

- Compare and contrast BDD and TDD.
- Utilize fundamental concepts of BDD to approach testing React components with Enzyme and Jest.
- Explain the difference between Jest and Enzyme.
-
-

The three main points you should walk away understanding today are:



01

BDD is the better option in comparison to TDD.

02

How to configure Enzyme within a React app.

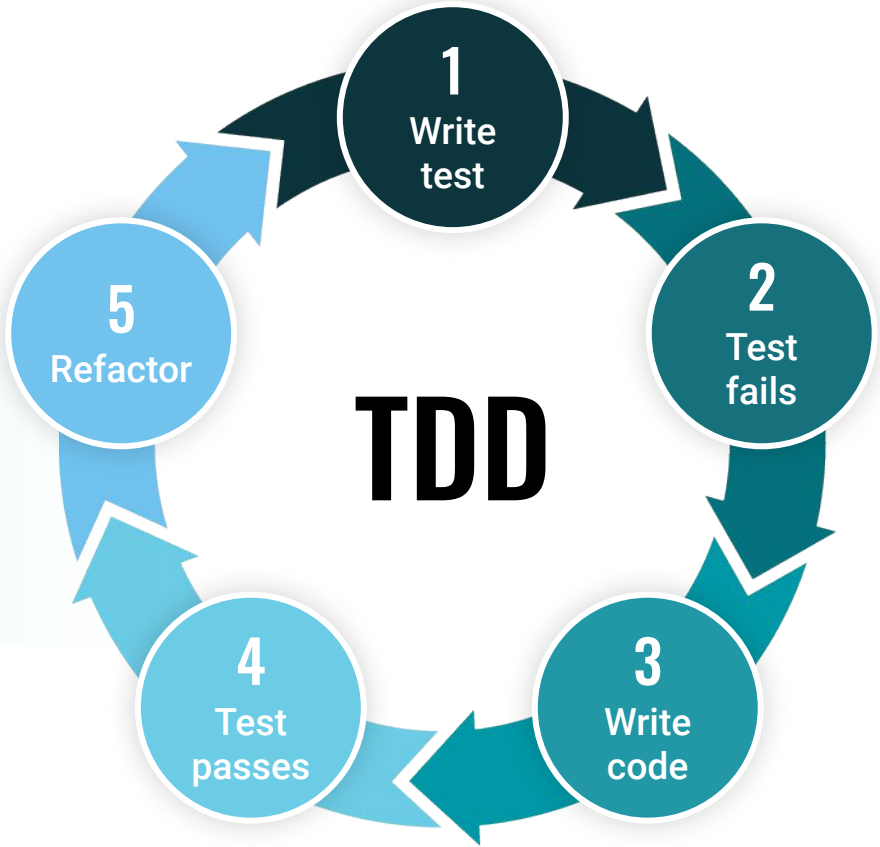
03

How to write a basic Enzyme test for React components.

TDD: Test-Driven Development

What Is Test-Driven Development (TDD)?

TDD is the practice of developing software by first writing a test and then writing code that passes said test.



Test-Driven Development (TDD) in Action

Steps of TDD:

01 Write the failing test.

02 Write the minimum amount of code to have for the test to pass.

03 Further enhance the tests for the code you just wrote.

04 Refactor the code to make the new tests pass.

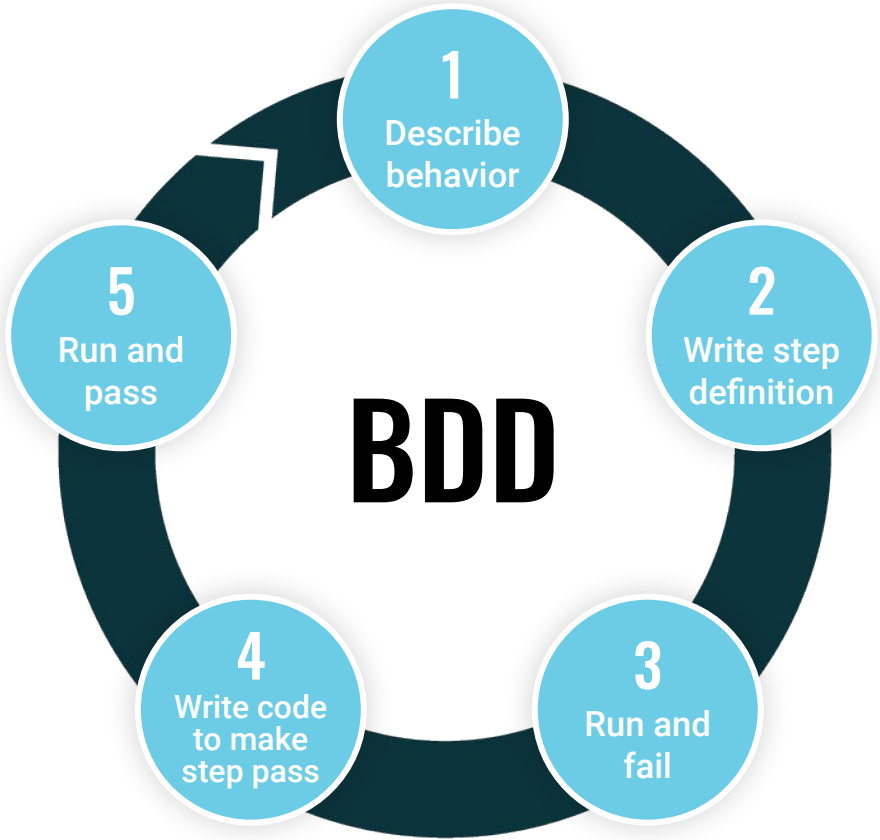
05 Iterate between adding tests and refactoring code until the desired functionally and code quality is achieved.

BDD: Behavior-Driven Development

What Is Behavior-Driven Development?

BDD is an evolution of TDD.

Whereas in TDD we write a test based on user requirements, in BDD we test to make sure that it is specific.



Behavior-Driven Development (BDD) in Action

Steps of BDD:

01 Describe behavior.

02 Write step definition.

03 Run and fail.

04 Write code to make step pass.

05 Run and pass.

TDD vs. BDD

TDD vs. BDD

While BDD and TDD sound very similar, there are a few defining factors that make them different:

01 TDD is a development practice, while BDD is a team methodology.

02 In TDD, developers write the tests.

03 In BDD, the automated specifications are created by users or testers.

04 For small, co-located, developer-centric teams, TDD and BDD are effectively the same. For a much more detailed discussion, InfoQ sponsored a virtual panel on the topic.

TDD vs. BDD

TDD	BDD
TDD is a development practice that focuses on **implementation of a feature**.	BDD is a team methodology that focuses on the **application behavior**.
In TDD, developers write the tests.	In BDD, the test are written by developers, customers, and/or QAs.
TDD's main focus is **unit testing**.	BDD's main focus is **testing application requirements**.

For small development teams, TDD and BDD are effectively the same thing.

Jest

Jest is a **JavaScript test runner and library** that can be utilized for creating, running, and structuring tests.

What Is Jest?

We will be using it mostly for its test “runner” functionality.

Jest has many benefits, including:

- It is pre-configured by CRA out of the box.
- It has great documentation.
- It is very easy to get up and running.

What Is Jest?

Jest's `test runner` allows you to have a dynamic environment for watching your test as you develop your code.

The Jest methods we will be using in this lesson are as follows:

`it():`

A method that runs a test. It is an alias for the test() method.

`expect():`

Allows you to test a value based on providing arguments.

Jest Test Runner

The file(s) from which the tests are being run

Number of passing and failing tests and test suites

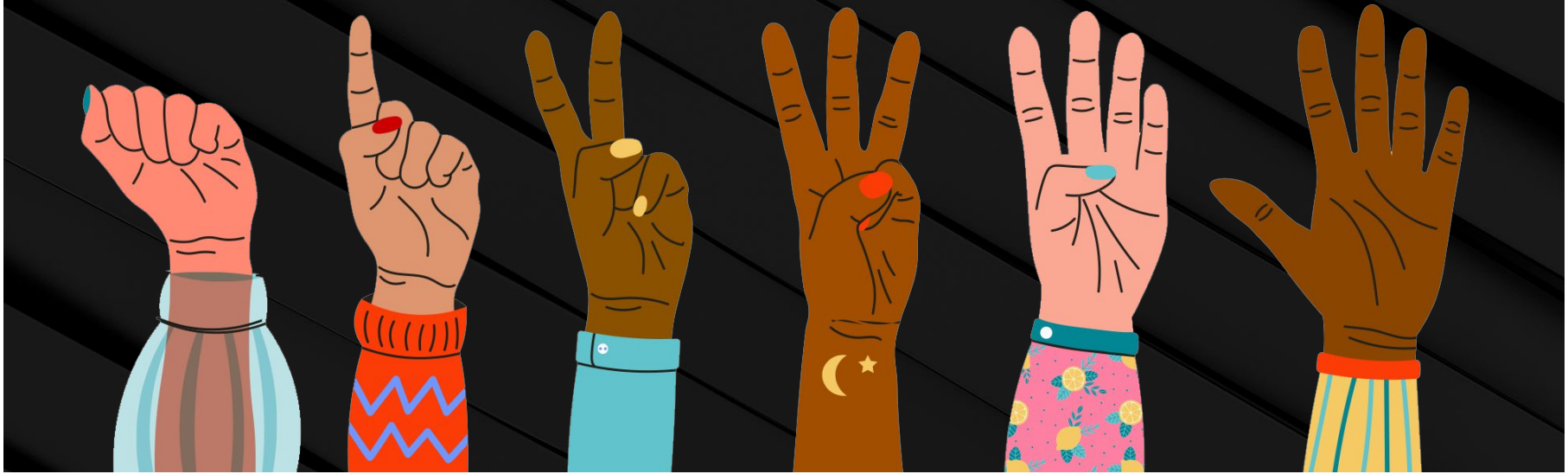
```
 PASS  src/App.test.js
   ✓ renders learn react link (31 ms)

Test Suites:  1 passed, 1 total
Tests:        1 passed, 1 total
Snapshots:    0 total
Time:         2.508 s
Ran all test suites.

Watch Usage: Press w to show more.
```

FIST TO FIVE:

How comfortable do you feel with the concepts we covered for Jest?



Enzyme

Enzyme is a **testing utility for React** created by Airbnb.

What Is Enzyme?

In short, it is a utility that we can use to write a test for a React component.
We use `Enzyme` to write our tests and `Jest` to run them and display their results.

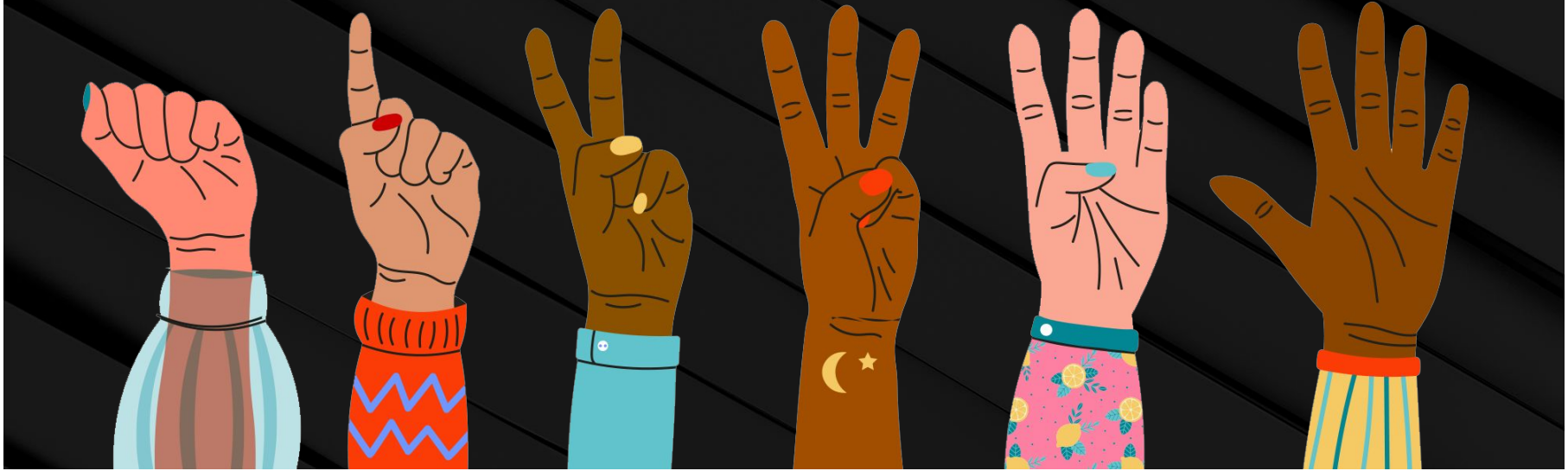
Enzyme Methods

The Enzyme methods we will be using in this lesson are as follows:

`find()`	Finds all nodes in the rendered wrapper that match the selector passed as an argument.
`text()`	Returns a string of the rendered text of the tree on which it is being invoked.
`shallow()`	Is a actually an `Enzyme API` that we utilize like a method. It accepts a single component as an argument and renders it “one level deep.”
`simulate()`	Simulates a event on the wrapper upon which it has been invoked.

FIST TO FIVE:

How comfortable do you feel with the concepts we covered for Enzyme?



Partner Activity: TDD vs. BDD

With a partner, you will discuss the differences between BDD and TDD.

Suggested Time:

15 minutes

Questions?

Time to Code

Demo Install React 16 with CRA

Suggested Time:

10 Minutes

Instructions: Demo Install React 16 with CRA

Follow the instructions in the file provided to:

Create a React project with `Create React App`.

Downgrade the project to a React version 16 project.

Install `Enzyme Adapter 16`.

Questions?

Configure Enzyme Within a React Project

Suggested Time:

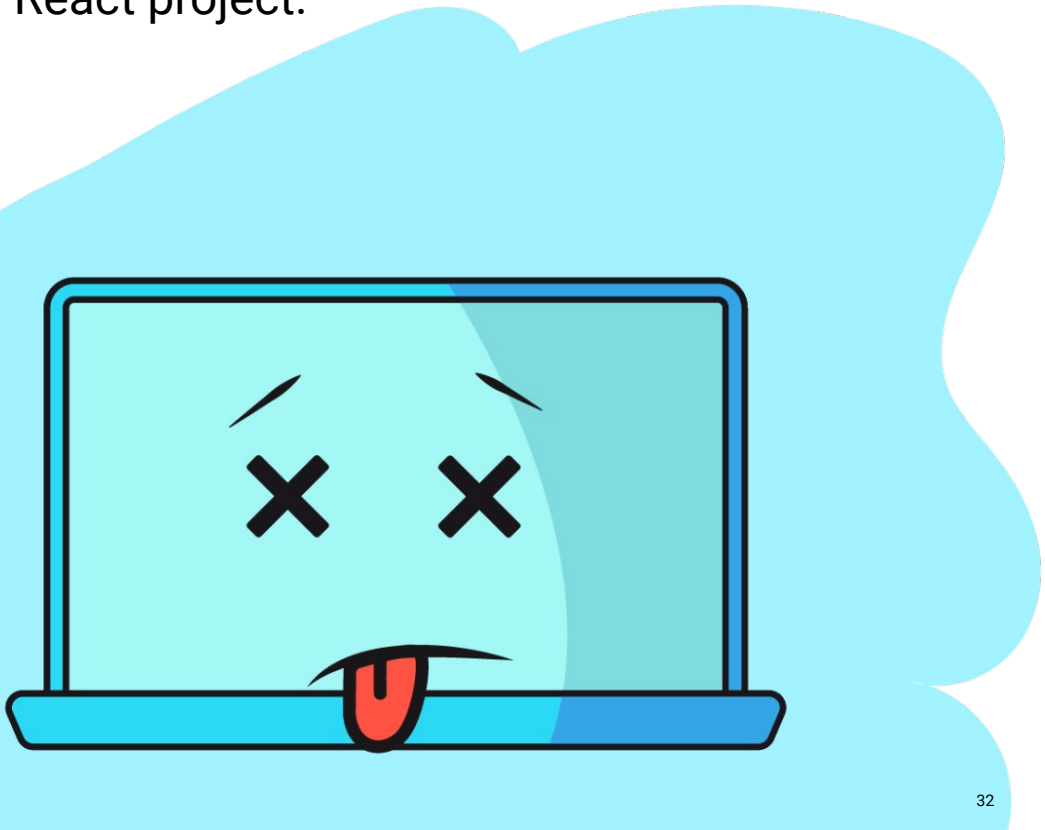
15 Minutes

Instructions: Creating a React V16 Project with CRA

Follow the instructions in the file provided to configure `Enzyme` and its subdependencies within your React project.

As with many new versions, they tend to break things. React V17 has broken the Enzyme adapter.

In short, we need to create our React projects as version 16.



Questions?

Instructor Demonstration

Initial Testing

Activity: Initial Testing

In this activity, you will complete each prompt listed in the README.

Suggested Time:

15 minutes

Before You Get Started...

`Name`

Is the method that should be the name that is used as an argument for the test.

`Test cases`

Are the requirements for that specific test.

Time's Up! Let's Review.

Break

Activity: Component Buildout Part 1

In this activity, you will will develop code in your `App.js` file so that the Enzyme tests written in `App.test.js` all pass.

Suggested Time:

10 minutes

Time's Up! Let's Review.

Questions?

Activity: Component Buildout Part 2

In this activity, you will continue to develop code in your `App.js` file so that the Enzyme tests written in `App.test.js` all pass.

Suggested Time:

15 minutes

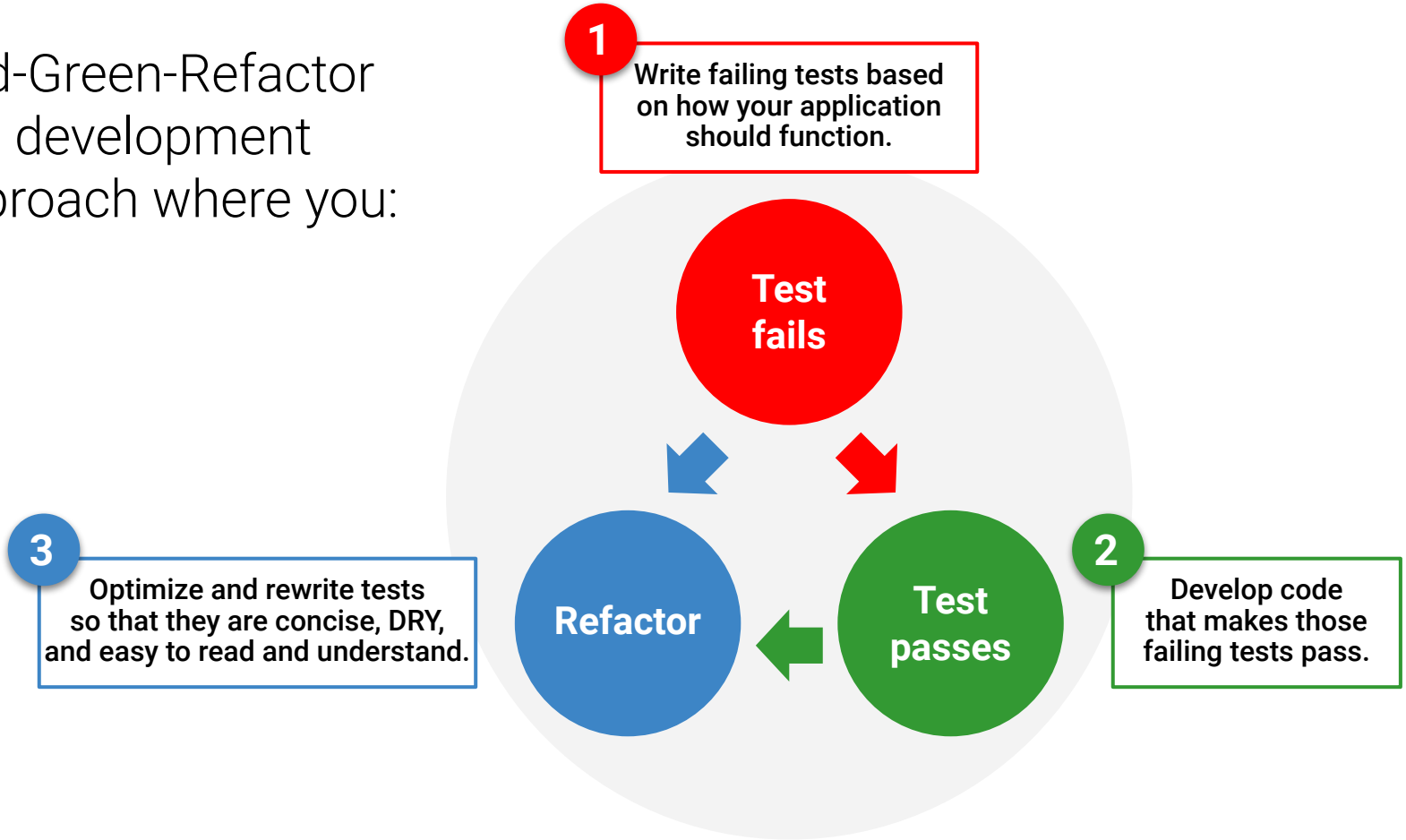
Time's Up! Let's Review.

Questions?

Red-Green-Refactor

What is Red-Green-Refactor?

Red-Green-Refactor is a development approach where you:



Refactor App.test.js:
Create Custom Helper Functions

Refactor App.test.js: Create Custom Helper Functions

Inside of `src/App.test.js`, create the following functions that meet the criteria provided:

01

`setup()`:

This function should accept no arguments and return a shallow render of the app component each time it is invoked.

02

`findByTestAttr()`:

This function should accept two arguments, wrapper and value, and perform a find lookup within the wrapper based on the data-test value provided.

Wrapper:

The component wrapper within which we want to look.

Value:

The test attribute value for which we are searching.

Refactor App.test.js: Create Custom Helper Functions

Prompt 1:

`setup()`

This function should accept no arguments and return a shallow render of the app component each time it is invoked.

```
const setup = () => shallow(<App />)
```

Refactor App.test.js: Create Custom Helper Functions

Prompt 1: major takeaways

Our custom setup function returns a single instance of a shallow rendered app component.

It allows us to optimize our code so that we do not have to instantiate an instance of the app component inside of every test, which in turn speeds up how much time it takes to run our test.

We are concerned about the time it takes to run our test because, in a large application, you can have hundreds, if not thousands, of tests. The more time we can save in running our test, the faster we can deploy our code.

Refactor App.test.js: Create Custom Helper Functions

Prompt 2:

`findByTestAttr()`

This function should accept two arguments, `wrapper` and `value`, and perform a find lookup within the wrapper based on the `data-test` value provided.

* Wrapper: The component wrapper within which we want to look.

* Value: The test attribute value for which we are searching.

```
const findByTestAttr = (wrapper, value) => wrapper.find(`[data-test='${value}']`)
```

Prompt 2: Major Takeaways

This function is meant to speed up our coding process by giving us a reusable function to quickly transverse and search any wrapper (i.e., component) for a specific data-test attribute value.

We could even rewrite this function so that it searches for a class or id value instead of a data-test attribute.

Prompt 2: Major Takeaways

This functions allows us to provide two arguments:

01 **Wrapper**

Such as the app component on which we perform a find to locate the value passed as the second argument within the said wrapper.

02 **Value**

Is the data-test attribute value for which we are searching within the wrapper provided as our first argument.

Activity: Refactor App.test.js: Create Custom Helper Function

In this activity, you will reinforce and build upon the testing skills that you have learned so far by refactoring the code inside of `App.test.js` to contain the `setup` and findByTestAttr function.

Suggested Time:

15 Minutes

Time's Up! Let's Review.

Questions?

The End