

Test-Driven Development



Test-Driven Development

- ◉ TDD is a popular approach to writing code among professional engineers
 - Write tests that run your code and compare the behavior of your code against an expected outcome
 - Write code that passes the tests
 - Repeat

Test-Driven Development

- ◉ Some benefits:
 - Expectations of how code should behave defined before writing code
 - Automated tests easily verify that your code is working as expected
 - Well-written tests help others understand how your code is supposed to work
- ◉ Some perceived drawbacks
 - Tests take time to write
 - Tests can be difficult to read (that's why we're practicing now)

Test-Driven Development

- ◉ TDD plays a central role at Fullstack
 - Great learning tool
 - Very important skillset to develop before entering job market
- ◉ TDD plays a central role in the Bootcamp
 - You will not write tests yourself initially (we've done that for you)
 - The workshops over the next few weeks will use TDD

TDD Setup: Codepen

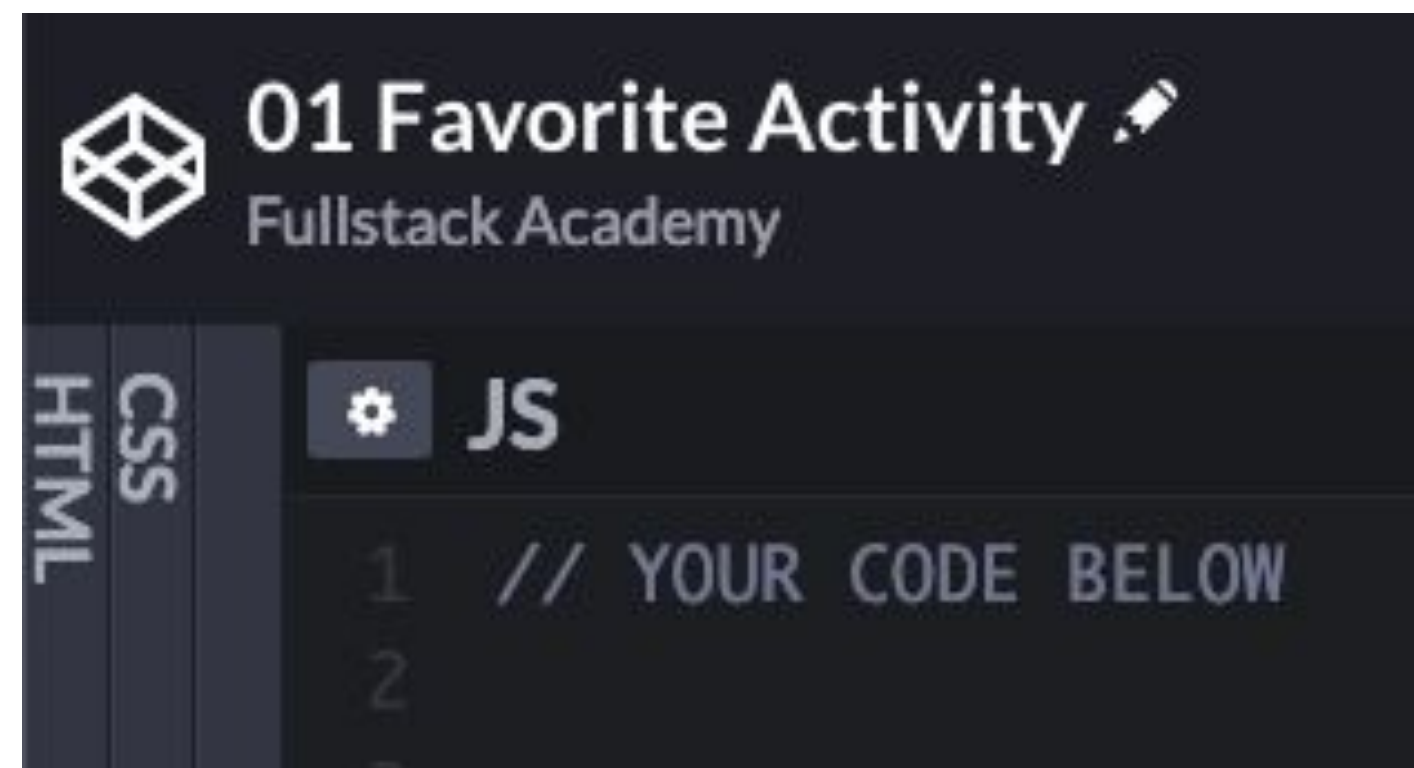
- ◉ Select the CodePen link for an individual problem from the “Practice Problems” section on LearnDot

Practice Problems

1) [Favorite Activity](#)

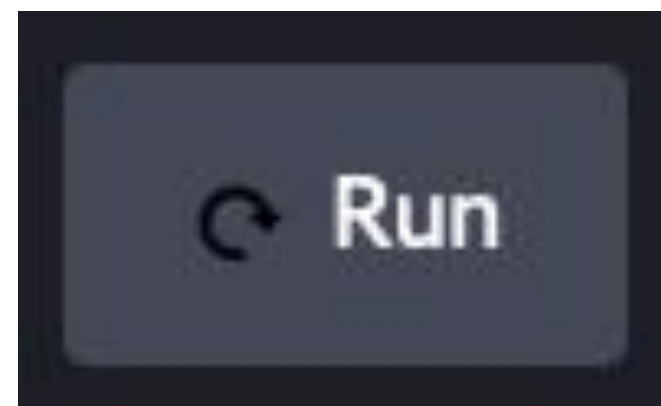
TDD Setup: Codepen

- ◉ Select the CodePen link for an individual problem from the “Practice Problems” section on LearnDot
- ◉ Write your code in the “JS” section of the CodePen, under the “YOU CODE BELOW” comment



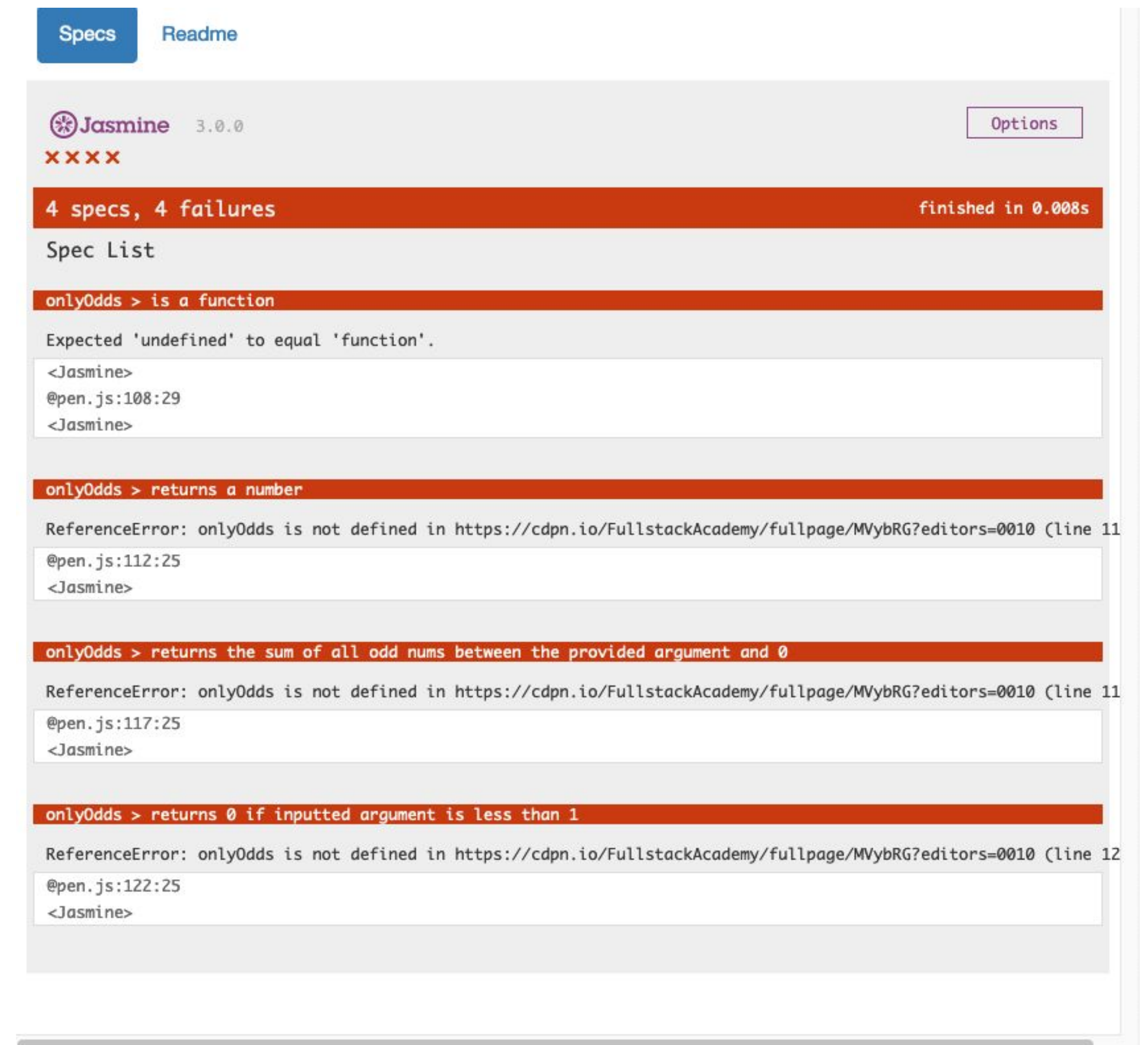
TDD Setup: Codepen

- ◉ Select the CodePen link for an individual problem from the “Practice Problems” section on LearnDot
- ◉ Write your code in the “JS” section of the CodePen, under the “YOU CODE BELOW” comment
- ◉ Select the “Run” button to run your code and the tests



TDD Setup: Codepen

- ◉ We will be using the Jasmine testing framework.
- ◉ All of the tests are initially failing because you haven't written any code yet!
- ◉ You can re-run the tests as many times as you want.
- ◉ When all the tests are passing, you've finished the workshop!



The screenshot shows a CodePen workspace with the 'Specs' tab selected. At the top, there are tabs for 'Specs' and 'Readme'. Below them, the Jasmine logo and version '3.0.0' are displayed, along with an 'Options' button. A red banner indicates '4 specs, 4 failures' and 'finished in 0.008s'. The 'Spec List' section shows four failing tests for the 'onlyOdds' function:

- onlyOdds > is a function**
Expected 'undefined' to equal 'function'.
@pen.js:108:29
- onlyOdds > returns a number**
ReferenceError: onlyOdds is not defined in https://cdpn.io/FullstackAcademy/fullpage/MVyBRG?editors=0010 (line 11)
@pen.js:112:25
- onlyOdds > returns the sum of all odd nums between the provided argument and 0**
ReferenceError: onlyOdds is not defined in https://cdpn.io/FullstackAcademy/fullpage/MVyBRG?editors=0010 (line 11)
@pen.js:117:25
- onlyOdds > returns 0 if inputted argument is less than 1**
ReferenceError: onlyOdds is not defined in https://cdpn.io/FullstackAcademy/fullpage/MVyBRG?editors=0010 (line 12)
@pen.js:122:25

Debugging a failing test

- ◉ Go back and set the variable `favoriteActivity` equal to the string `'not coding'`
- ◉ What does the output of the test say now?

```
favoriteActivity > should be coding  
Expected 'not coding' to equal 'coding'.
```

- ◉ The test is telling you, “I found the `favoriteActivity` variable, but the value stored in it is `'not coding'`, and it needs to be `'coding'` to pass this test”
- ◉ Use this information to help you fix your code until all of the tests pass

Reading a test

- ◉ It's often useful to read the tests so you can understand what they are testing
- ◉ Go back to the CodePen and look for the word expect:
`expect(favoriteActivity).toEqual('coding');`
- ◉ The test comes down to this line; the test will pass when the value stored in `favoriteActivity` equals the string 'coding'
- ◉ Look for lines that start with `expect` in the other test specs to better understand exactly how they're working

Recap

/*

- Benefits of Test-Driven Development
- TDD with Jasmine in CodePen
- Reading tests and passing specs

*/