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 clearly defined before writing code
  - Automated tests easily verify that your code is working as expected
  - Well-written tests help others understand how you code is supposed to work
- Some perceived drawbacks
  - Tests take time to write
  - Tests can be difficult to read (that’s why we start practicing now)

Test-Driven Development

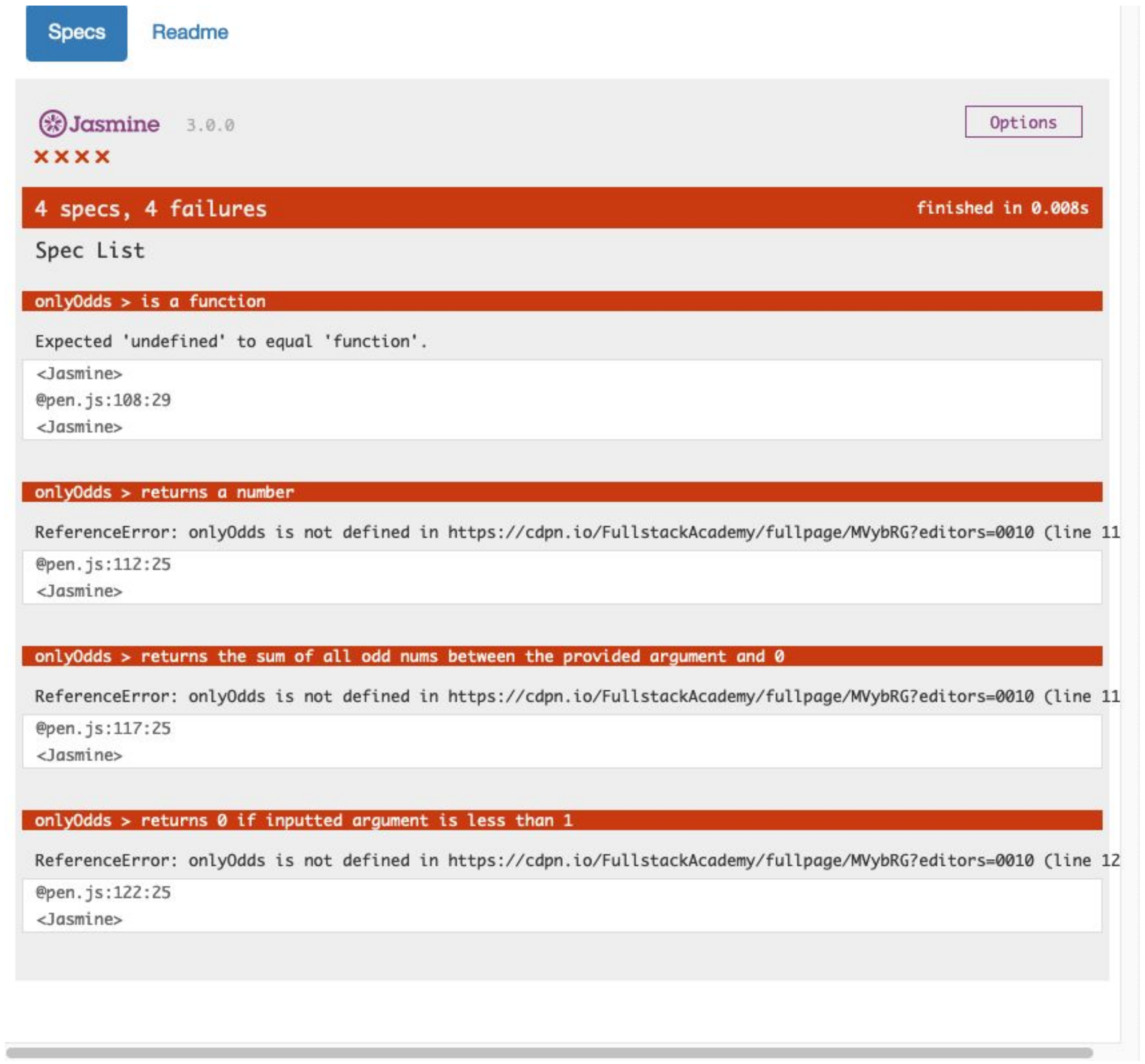
- TDD plays a central role at TEJ Fellowship
  - Great learning tool
  - Very important skillset to develop before entering job market
- TDD plays a central role in the Bootcamp
  - You will not write tests yourself initially (we’ve done that for you)
  - The workshops over the next few weeks will use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

TDD Setup: Running Code Locally

- Download the practice problems for the course
- Open the downloaded folder in VSCode (VSC)
- In VSC, open the terminal (View > Debug Terminal > Terminal tab)
- Type testem
- Copy the url (similar to http://localhost:7357/) and paste it into your browser

TDD Setup: Running Code Locally

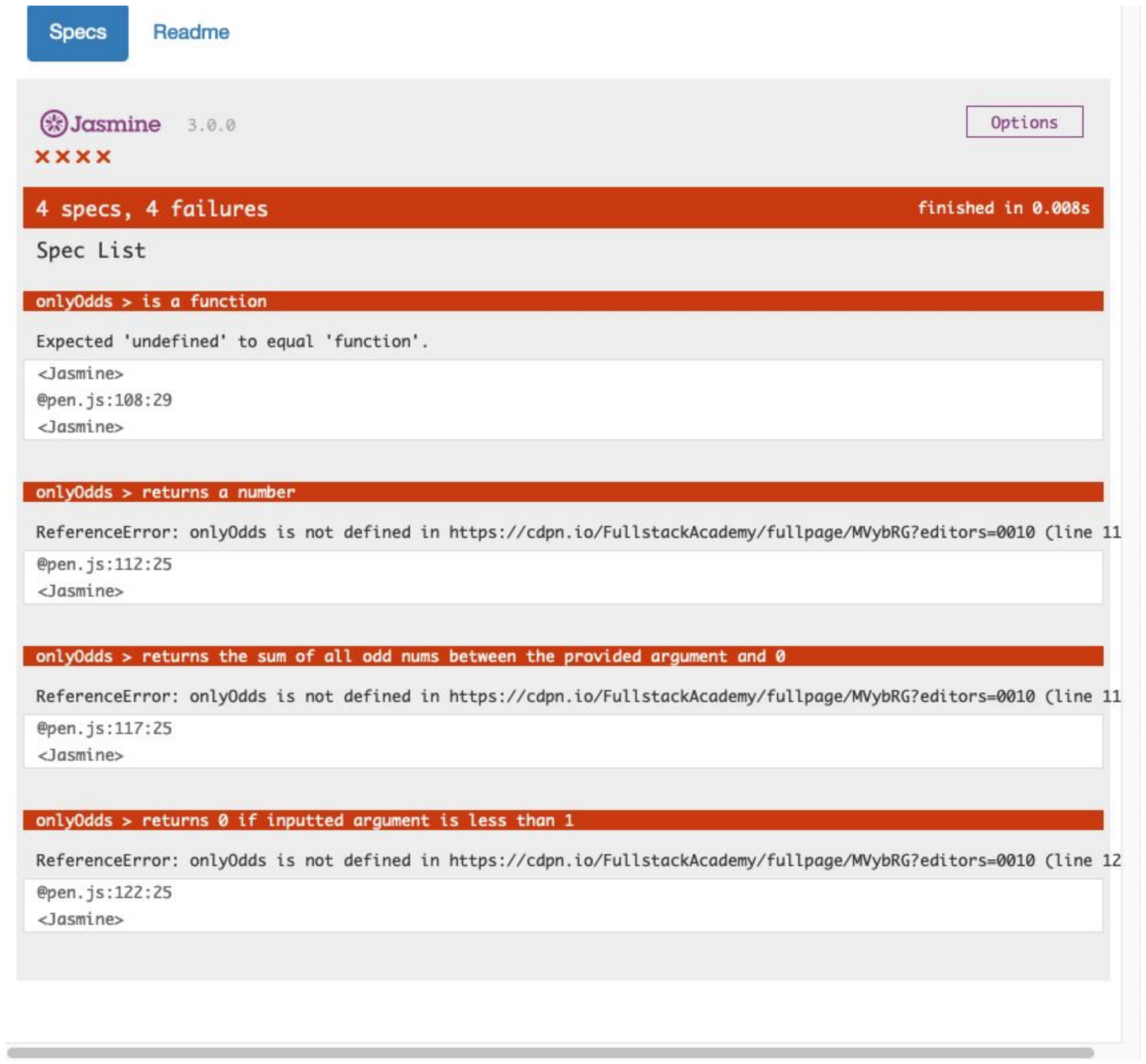
- The site you've navigated to is not from the internet; testem is using your own computer to create this site.
- This site is showing you the current status of the tests (or specs) being run against your code.



```
××××
4 specs, 4 failures                                                    finished in 0.008s
Spec List
onlyOdds > is a function
Expected 'undefined' to equal 'function'.
<Jasmine>
@pen.js:108:29
<Jasmine>
onlyOdds > returns a number
ReferenceError: onlyOdds is not defined in https://cdpn.io/FullstackAcademy/fullpage/MVybRG?editors=0010 (line 11
@pen.js:112:25
<Jasmine>
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
<Jasmine>
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
<Jasmine>
```

TDD Setup: Running Code Locally

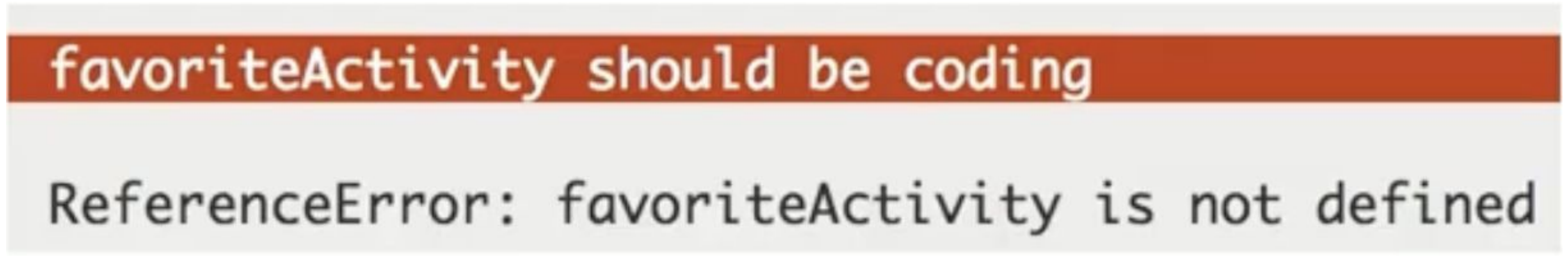
- All of the tests are initially failing because you haven't written any code yet!
- Every time you save one of the solution files, the tests will be automatically run again.
- When all the tests are passing, you're finished with the workshop!



```
××××
4 specs, 4 failures                                                    finished in 0.008s
Spec List

onlyOdds > is a function
Expected 'undefined' to equal 'function'.
<Jasmine>
@pen.js:108:29
<Jasmine>

onlyOdds > returns a number
ReferenceError: onlyOdds is not defined in https://cdpn.io/FullstackAcademy/fullpage/MVybRG?editors=0010 (line 11
@pen.js:112:25
<Jasmine>

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
<Jasmine>

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
<Jasmine>
```

Passing the first test

- Let’s look at the prompt for the first practice problem:
  - “Create a variable called favoriteActivity. Assign the value ‘coding’ to favoriteActivity.”
- Now let’s look at the output for the first test:

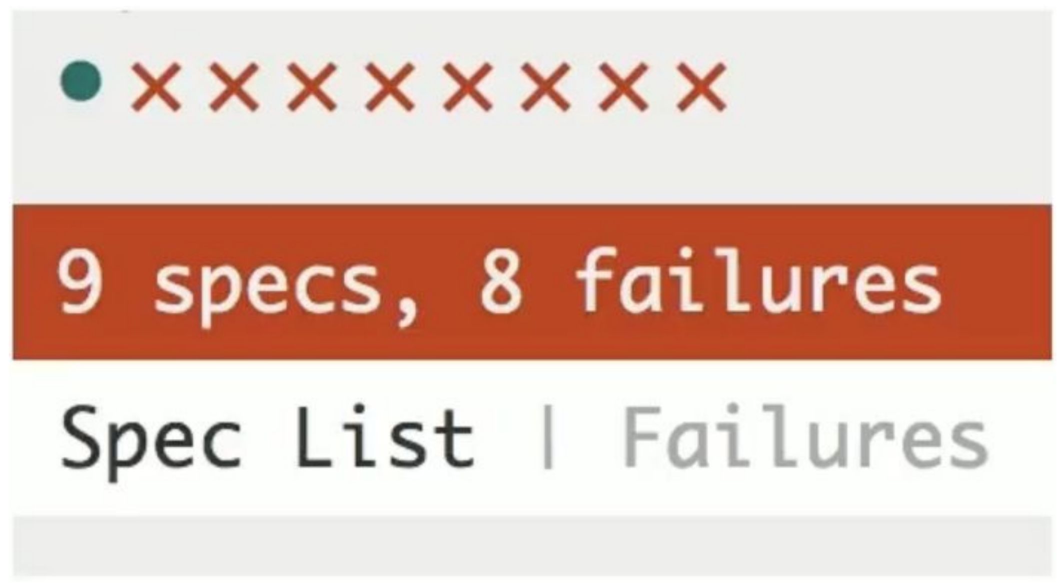


```
favoriteActivity should be coding

ReferenceError: favoriteActivity is not defined
```

- The test is looking through your code, trying to find a variable called `favoriteActivity`.

Passing the first test

- Open the Favorite Activity CodePen (or favorite-activity.js in the 01-favorite-activity folder) and create a `favoriteActivity` variable and set it equal to 'coding'.
- As soon as you run or save your code, the test site should update

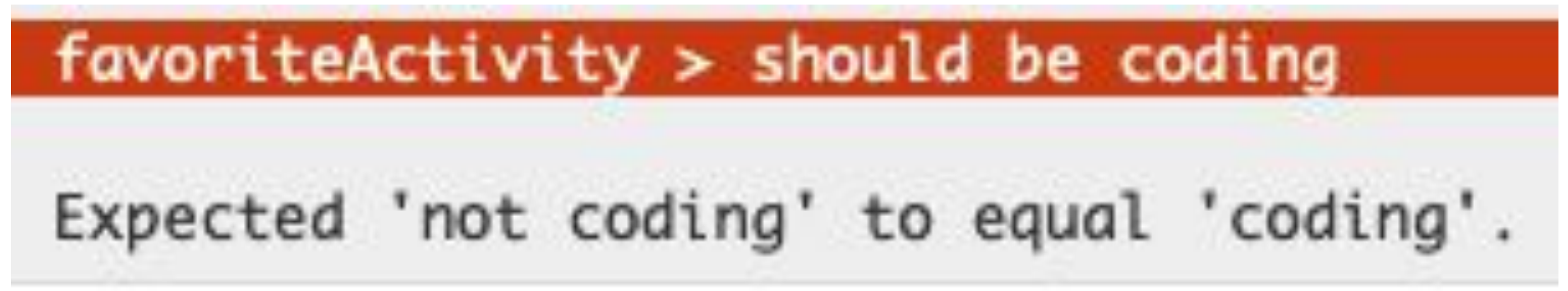


```
•×××××××××
9 specs, 8 failures
Spec List | Failures
```

- One of the red Xs turned into a green dot! You're passing a test!

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
- open favorite-activity-spec.js.
- 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
  - Using CodePen is fine!
  - Reading tests and passing specs
*/
```