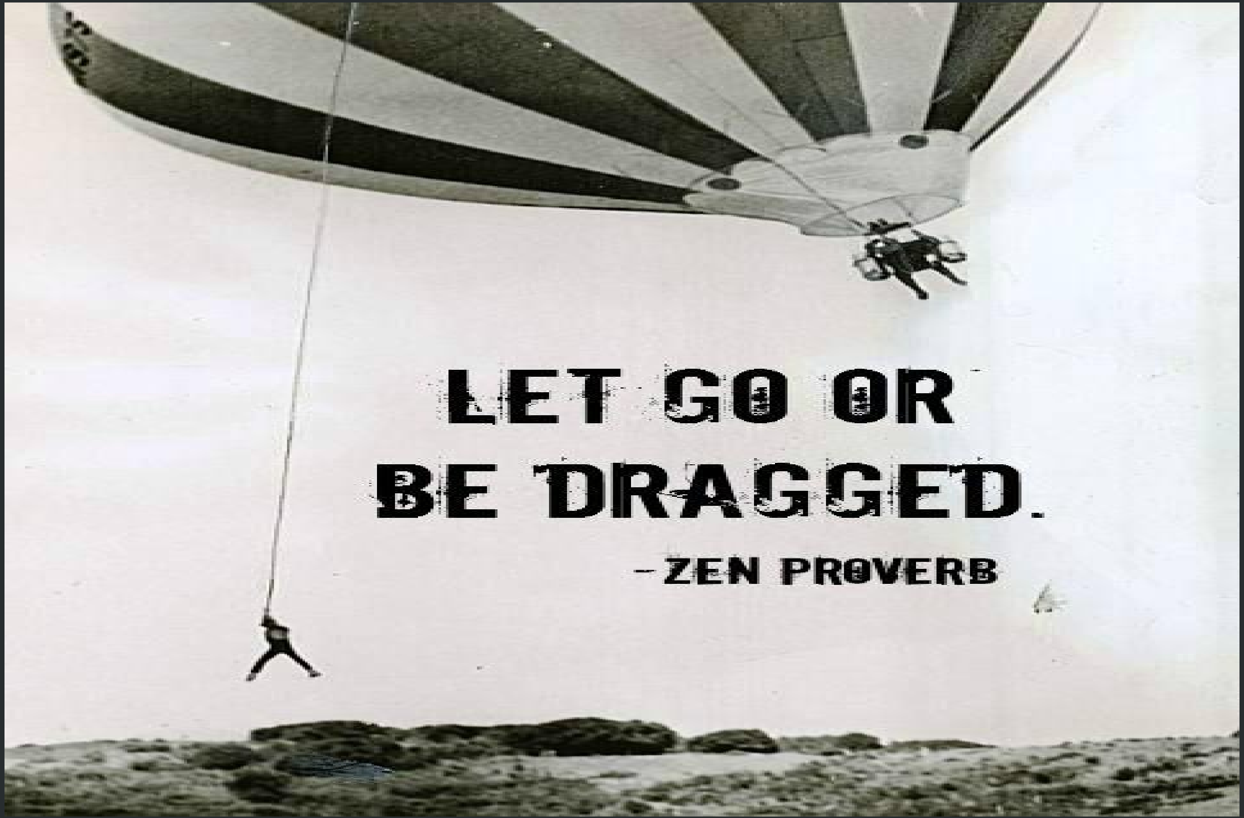
IRON
HACK

JavaScript Koans


LET GO OR
BE DRAGGED.
-ZEN PROVERB

What are Koans?

- Koans originate from Zen Buddhism, and are paradoxical riddles or stories used to test “students” on their path to enlightenment. They are designed to provoke thought or doubt in the student’s mind.

Example

THE STUDENT Doken was told to go on a long journey to another monastery. He was extremely upset, because he felt that this trip would interrupt his studies for many months. So he said to his friend, the advanced student Sogen:

"Please ask permission for me to to come on the trip with you. There are so many things I do not know; but if you come along we can discuss them - in this way I can learn as we travel."

"All right," said Sogen. "But let me ask you a question: If you are hungry, what satisfaction to you if I eat rice? If your feet are lame, what comfort to you if I go on merrily? If your bladder is full, what relief to you if I piss?"

Koans in JavaScript

Life can be confusing, and the path to enlightenment is a difficult task that challenges ego, logic, and one’s own being.

Koans in JavaScript

While JavaScript may not be as heavy as life, your path to code enlightenment will be challenging, often times confusing.

Koans in JavaScript

Occasionally, your reaction to JavaScript may be that of a student of Zen, working on his master's Koans...

JavaScript Koans

- So, let’s begin our path to JavaScript “enlightenment”.
- We’ve created JavaScript Koans to guide you through many of JavaScript’s tricky pitfalls and quirks.

Testing

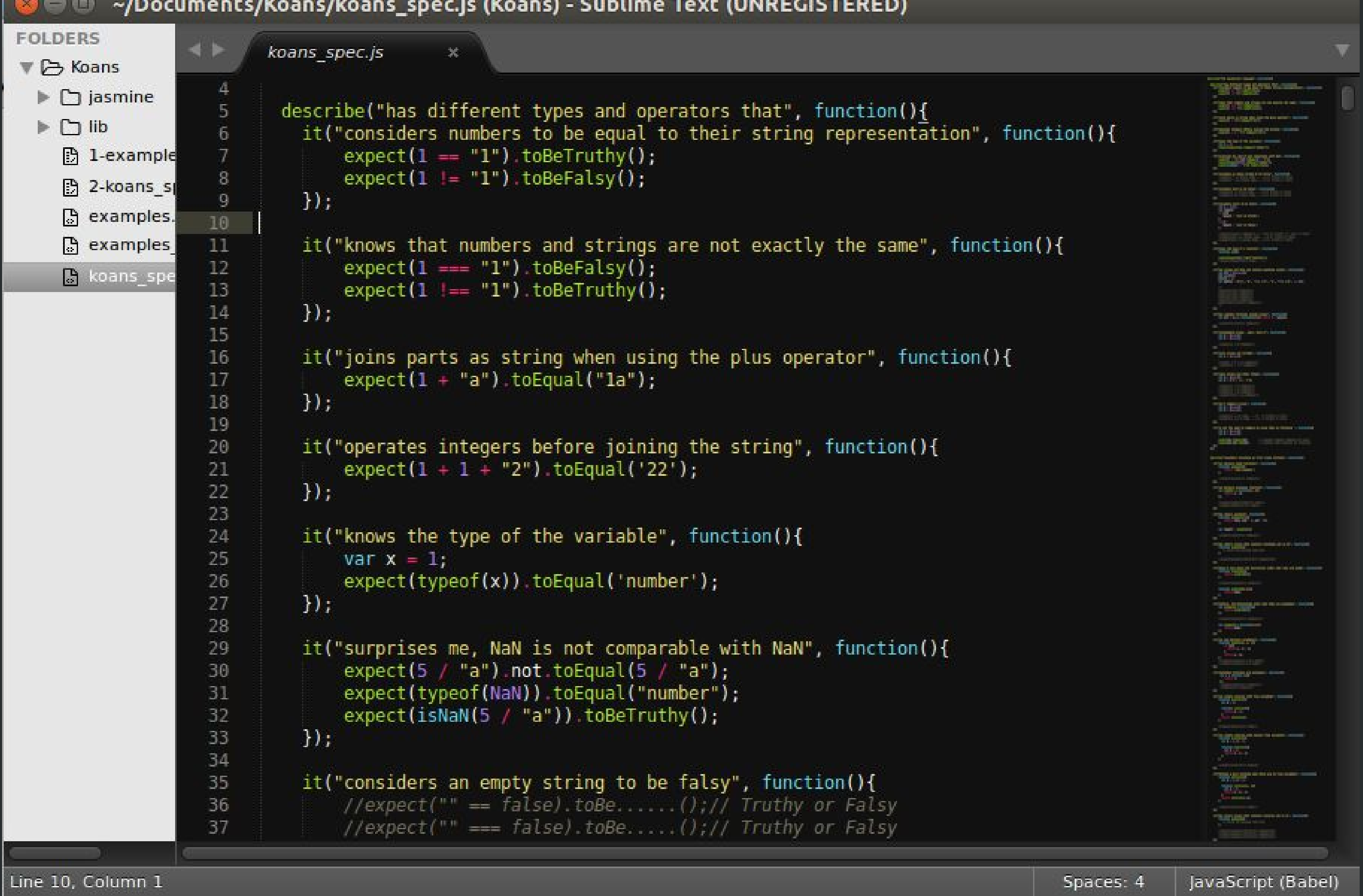
The JavaScript Koans are a series of assertions built on the Jasmine test kit that you must solve on your path to JavaScript enlightenment.

Testing

The koans become increasingly more difficult as you continue, so don’t feel discouraged as you move forward through them.

How to

- Go to the dropbox link provided: Koans dropbox link
- Download and extract the zip into a new directory.
- Open the directory in sublime, move to the specs folder and select the koans.js, it should look something like this:

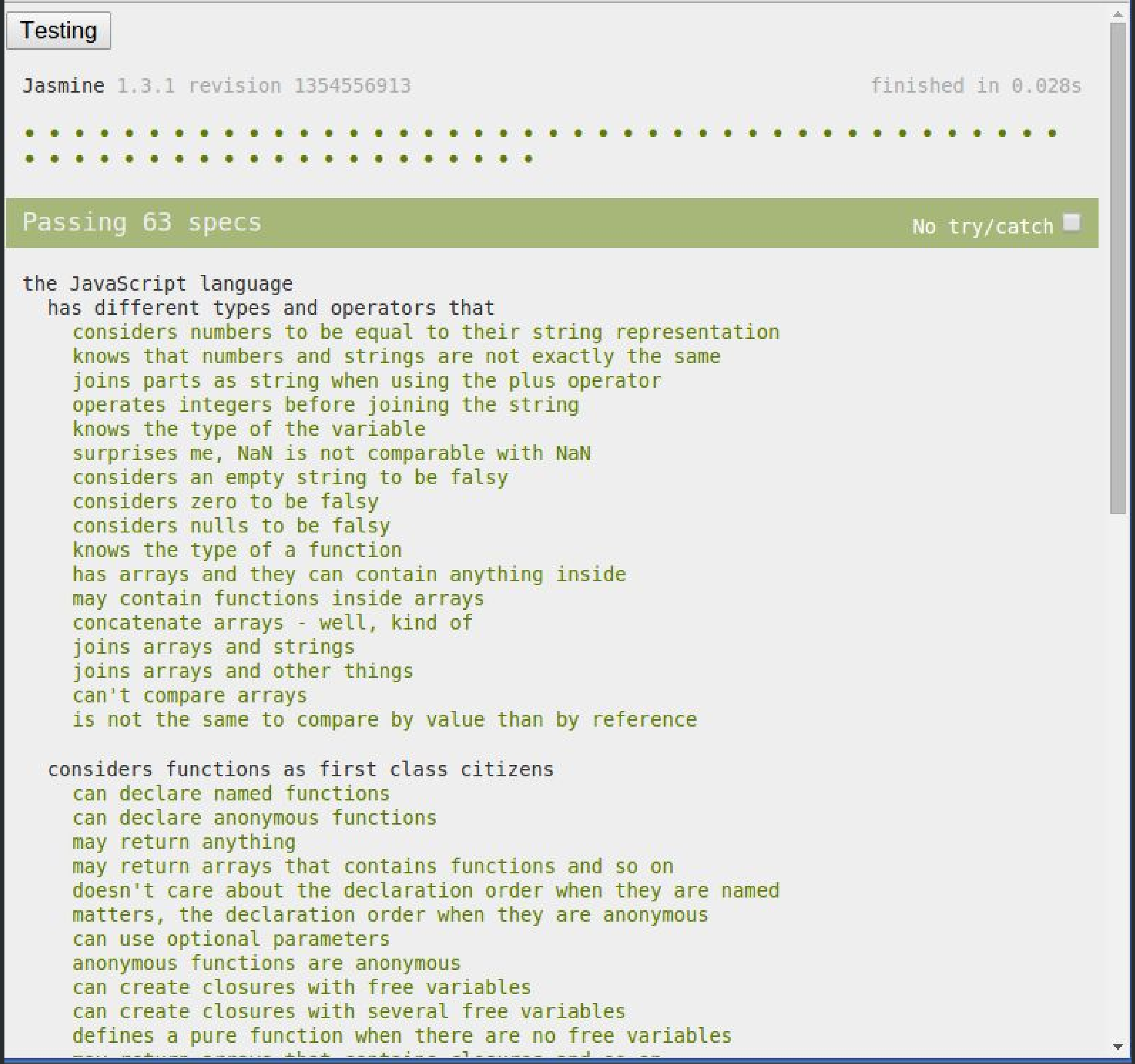


```

describe("has different types and operators that", function(){
  it("considers numbers to be equal to their string representation", function(){
      expect(1 == "1").toBeTruthy();
      expect(1 != "1").toBeFalsy();
  });

  it("knows that numbers and strings are not exactly the same", function(){
      expect(1 === "1").toBeFalsy();
      expect(1 !== "1").toBeTruthy();
  });

  it("joins parts as string when using the plus operator", function(){
      expect(1 + "a").toEqual("1a");
  });

  it("operates integers before joining the string", function(){
      expect(1 + 1 + "2").toEqual('22');
  });

  it("knows the type of the variable", function(){
      var x = 1;
      expect(typeof(x)).toEqual('number');
  });

  it("surprises me, NaN is not comparable with NaN", function(){
      expect(5 / "a").not.toEqual(5 / "a");
      expect(typeof(NaN)).toEqual("number");
      expect(isNaN(5 / "a")).toBeTruthy();
  });

  it("considers an empty string to be falsy", function(){
      //expect("" == false).toBe......();// Truthy or Falsy
      //expect("" === false).toBe.....();// Truthy or Falsy
```

IRON HACK

How to

- Then, open `SpecRunner.html` in the browser, it should look like...

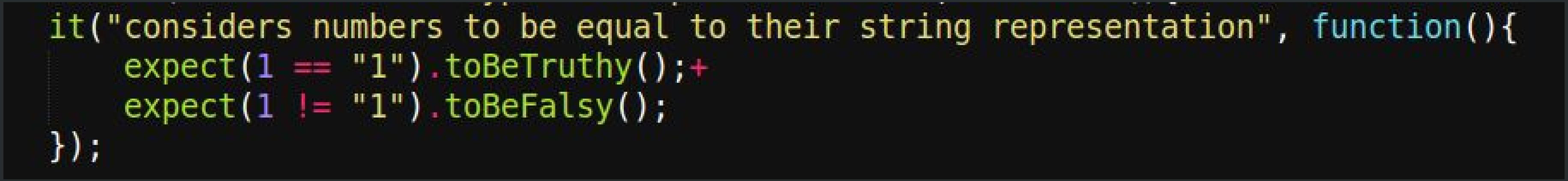


```
Testing

Jasmine 1.3.1 revision 1354556913                                finished in 0.028s

● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ●
● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ●

Passing 63 specs                                                    No try/catch

the JavaScript language
  has different types and operators that
    considers numbers to be equal to their string representation
    knows that numbers and strings are not exactly the same
    joins parts as string when using the plus operator
    operates integers before joining the string
    knows the type of the variable
    surprises me, NaN is not comparable with NaN
    considers an empty string to be falsy
    considers zero to be falsy
    considers nulls to be falsy
    knows the type of a function
    has arrays and they can contain anything inside
    may contain functions inside arrays
    concatenate arrays - well, kind of
    joins arrays and strings
    joins arrays and other things
    can't compare arrays
    is not the same to compare by value than by reference

  considers functions as first class citizens
    can declare named functions
    can declare anonymous functions
    may return anything
    may return arrays that contains functions and so on
    doesn't care about the declaration order when they are named
    matters, the declaration order when they are anonymous
    can use optional parameters
    anonymous functions are anonymous
    can create closures with free variables
    can create closures with several free variables
    defines a pure function when there are no free variables
```

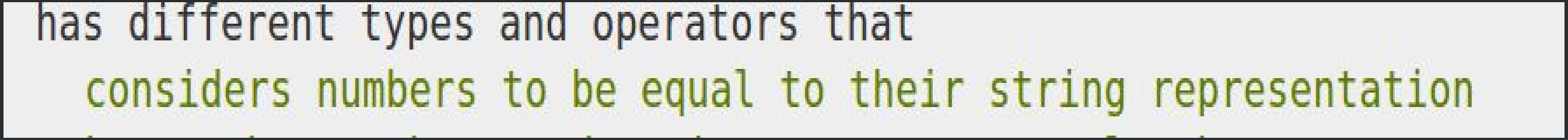
How to

- As you probably noticed, the test assertions match up with the passing tests in the browser right now.
- The jasmine test kit is testing the assertions we’ve made, and are displaying it in pretty green(passing) or red(failing) html.

How to



```
it("considers numbers to be equal to their string representation", function(){
    expect(1 == "1").toBeTruthy();+
    expect(1 != "1").toBeFalsy();
});
```

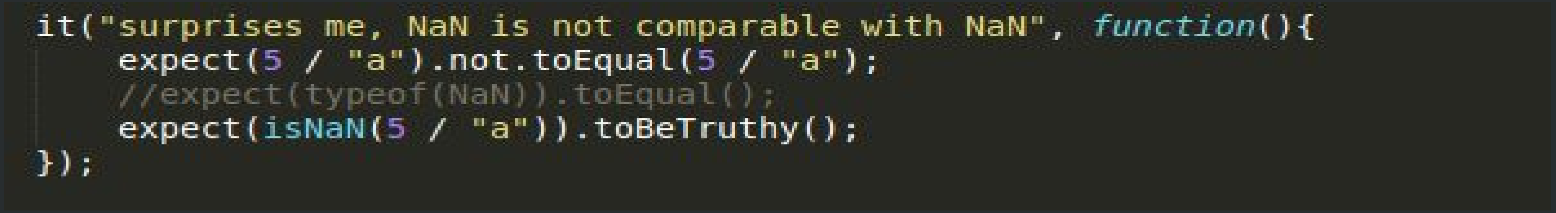


```
has different types and operators that
  considers numbers to be equal to their string representation
```

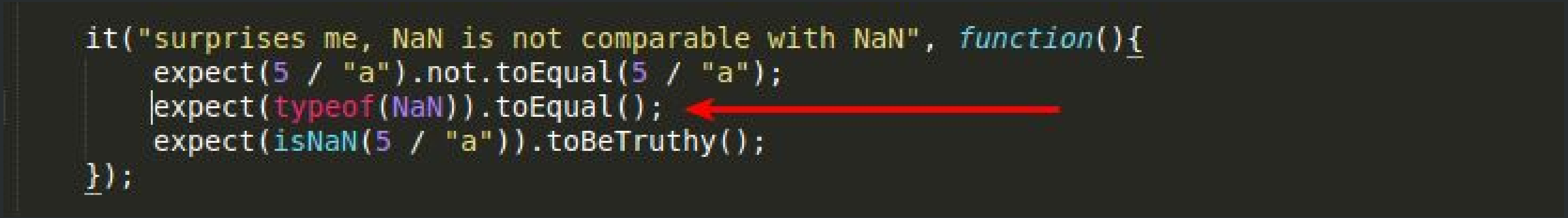
How to

- As you notice, the assertion is already filled in with the correct answer and the test is passing, a “green” test.
- Lets make one fail:

How to

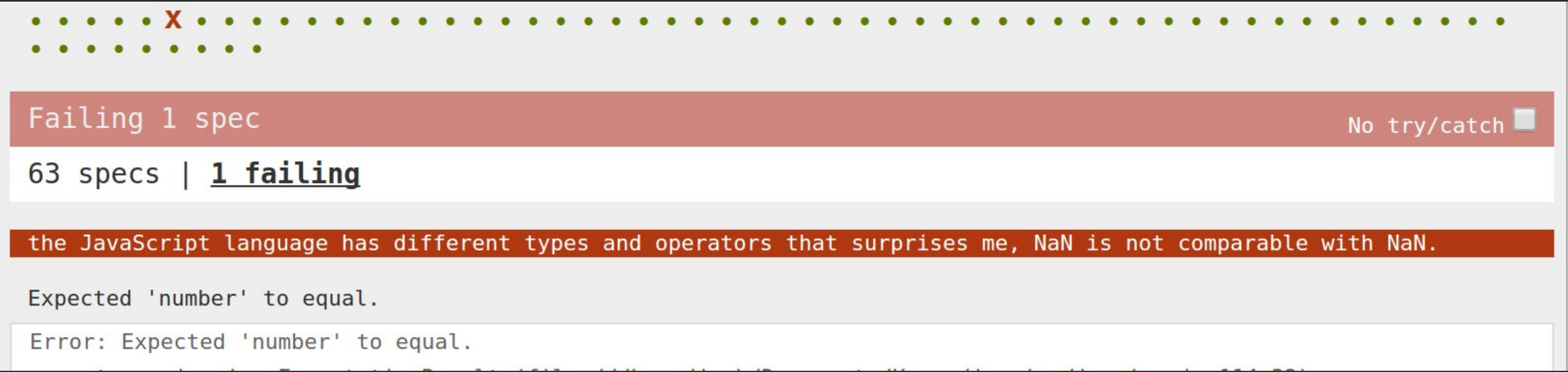


```
it("surprises me, NaN is not comparable with NaN", function(){
    expect(5 / "a").not.toEqual(5 / "a");
    //expect(typeof(NaN)).toEqual();
    expect(isNaN(5 / "a")).toBeTruthy();
});
```



```
it("surprises me, NaN is not comparable with NaN", function(){
    expect(5 / "a").not.toEqual(5 / "a");
    expect(typeof(NaN)).toEqual();
    expect(isNaN(5 / "a")).toBeTruthy();
});
```

How to

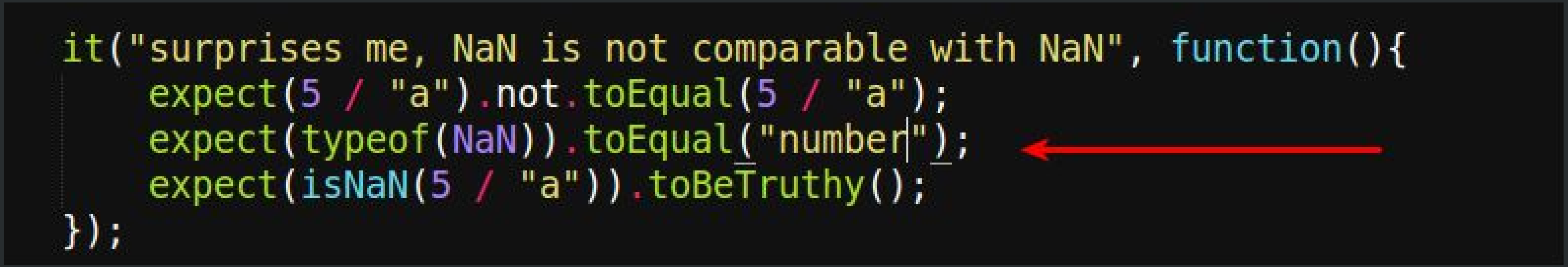


```
• • • • • X • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • •
• • • • • • • • •

Failing 1 spec                                                                     No try/catch

63 specs | 1 failing

the JavaScript language has different types and operators that surprises me, NaN is not comparable with NaN.

Expected 'number' to equal.

Error: Expected 'number' to equal.
```

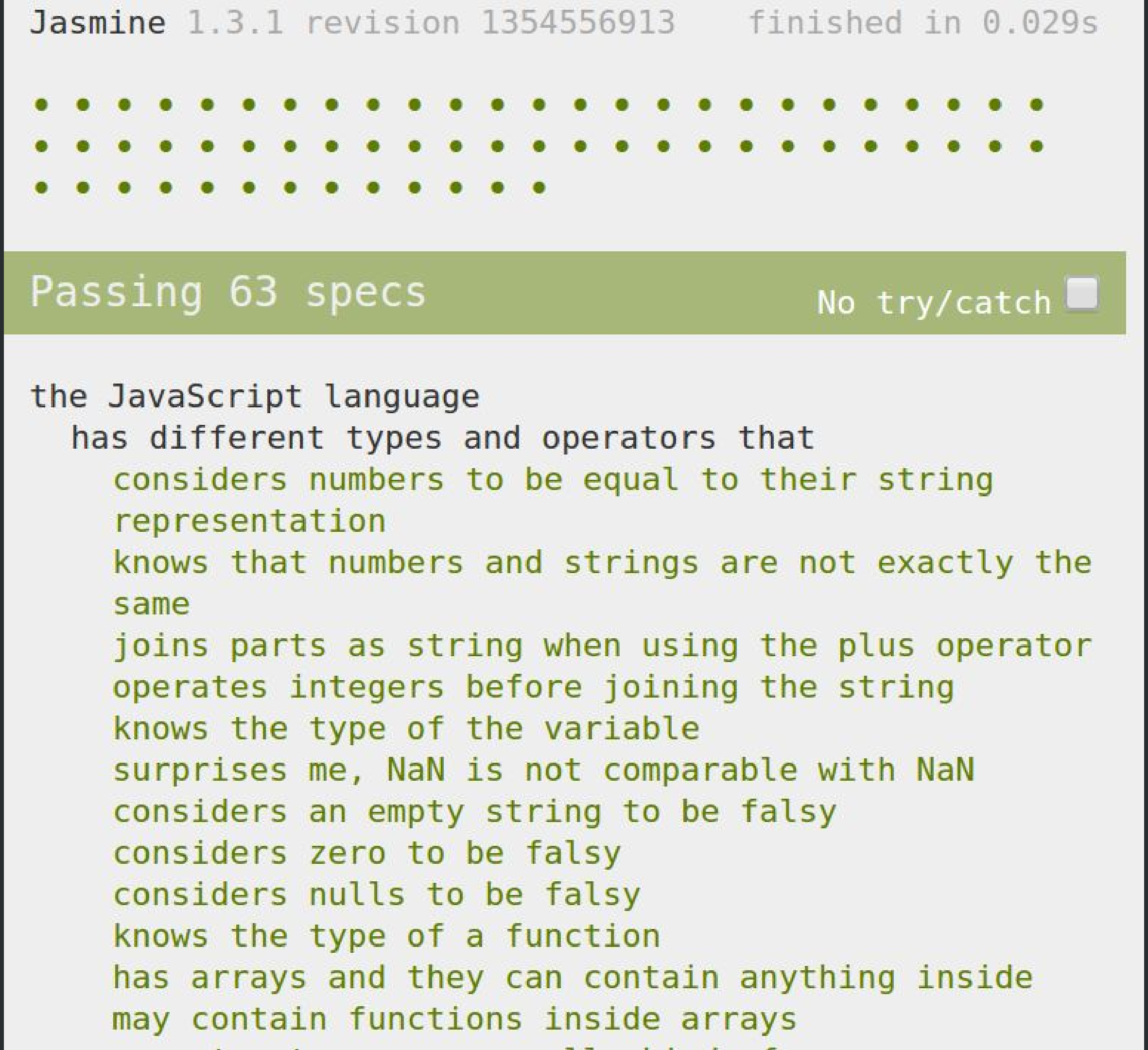
How to

- Assertions left to be filled in by you are currently commented out, now it is your job to make them pass.
- So what is the type of NaN(Not a Number), and will in turn make the test pass...?

How to



```
it("surprises me, NaN is not comparable with NaN", function(){
    expect(5 / "a").not.toEqual(5 / "a");
    expect(typeof(NaN)).toEqual("number");
    expect(isNaN(5 / "a")).toBeTruthy();
});
```



```
Jasmine 1.3.1 revision 1354556913    finished in 0.029s

• • • • • • • • • • • • • • • • • • • • • • • • •
• • • • • • • • • • • • • • • • • • • • • • • • •
• • • • • • • • • • • • •

Passing 63 specs                        No try/catch ☐

the JavaScript language
  has different types and operators that
    considers numbers to be equal to their string
    representation
    knows that numbers and strings are not exactly the
    same
    joins parts as string when using the plus operator
    operates integers before joining the string
    knows the type of the variable
    surprises me, NaN is not comparable with NaN
    considers an empty string to be falsy
    considers zero to be falsy
    considers nulls to be falsy
    knows the type of a function
    has arrays and they can contain anything inside
    may contain functions inside arrays
```

How to

- Common sense, right?
- Like I said, JavaScript is a bit tricky at times, and these koans are a way to get well acquainted with them.
- So continue on as we just did, uncomment any commented assertions, fill them in, refresh your browser and see if they pass.

How to

- Assertions with `toEqual()` are going to be filled in with values, for example: `toEqual("number").`
- Assertions with toBe...are going to be filled in with `toBeTruthy` or `toBeFalsy`.

How to

Some assertions may not pass no matter the value you place in them. Watch out for those!

JavaScript Koans

Now, go! Seek enlightenment.