TDD in JavaScript

Section 2: Test Driven Development

2.1 What is TDD

There may be variations on the concept of Test Driven Development (TDD) depending on the article(s) you may read on this topic but this document will introduced you to a very simple and practical explanation on TDD.

There is generally a workflow to TDD whereby you first write your test based on a use case. This step is done before you write any kind of production code. Normally you will have use cases that you want the software to go through and they can be isolated down to particular areas and those areas is where you start writing your test.

An example would be writing a calculator app. We would create a test and in that test we would create an object for a calculator (which we will not have yet because we are writing the test first) — we may call this calculator object e.g. calc.

On that calc object we may then create our first test e.g. calc.sum(a, b) which takes in two numbers as parameters. This will write the test and the test would fail because there is no calculator object and this is perfectly fine as this is the first step. This will now force us to write the associated code and we can then see the test will pass once we have the bare minimum amount of logic to make the test pass.
For example, we can pass two 5 as the two parameters which will equal 10 and test the .sum(a, b) would return 10 back because that will pass our test as the bare minimum.

Once we have the code working we can then refactor the code and do whatever manipulation we want to the production code and ensure the test will continue passing at that point forward.

Therefore, the test should be very self-explanatory as to what we want to test i.e. it displays the use case which is very easy to read. Initially it will fail because there is nothing behind the test which will force us to write the code which is why it is called Test Driven Development. We write enough to make the test initially pass. Finally, we then go back to the code and do any refactoring i.e. put in more logic, clean the syntax or re-write the code for performance — building out the production code al the while ensuring the test is passing against that particular use case.