Week 6 Quiz

Reset Quiz

SHOW GOT IT! When is a JavaScript Error Object thrown?

The Error object is how JavaScript deals with runtime errors - so at code runtime!

SHOW GOT IT! How do you halt program execution with an instance of an error object in JavaScript?

Using the keyword throw you can throw your own runtime errors that will stop program execution.

SHOW GOT IT! What type of block will allow you to run an erroring function then continue the execution of code after that function is run?

We can use try...catch blocks with functions that might throw an error to catch that error and continue code execution after that error was thrown

SHOW GOT IT! When kind of error is thrown when the JavaScript engine attempts to parse code that does not conform to the syntax of the JavaScript language?

A SyntaxError is thrown when there is an error in the syntax of the executed code.

SHOW GOT IT! What kind of error is thrown when a variable or parameter is not of a valid type?

A TypeError is thrown when an operation cannot be performed because the operand is a value of the wrong type.

SHOW GOT IT! What type of error is thrown when a non-existent variable is referenced?

The ReferenceError object represents an error when a non-existent variable is referenced.

SHOW GOT IT! What kind of error will be thrown when the code below is executed?

```
function callPuppy() {
    const puppy = "puppy";
    console.log(pupy);
}

callPuppy();
```

ReferenceError: pupy is not defined

SHOW GOT IT! What kind of error will the code below throw when executed?

```
let dog;

dog();
```

TypeError: dog is not a function

SHOW GOT IT! What kind of error will the code below throw when executed?

```
const puppy = "puppy";

puppy = "apple";
```

TypeError: Assignment to constant variable.

SHOW GOT IT! What kind of error will be thrown when the code below is run?

```
function broken () {
    console.log("I'm broke");
}}
```

Syntax Error: Unexpected token '}'

SHOW GOT IT! What kind of error will be thrown when the code below is run?

```
function broken () {
    console.log("I'm brok"
```

SyntaxError: missing) after argument list

SHOW GOT IT! Identify at least two reasons why developers use TDD.

1. Writing tests before code ensures that the code written works.
2. Only required code is written.
3. TDD helps enforced code modularity

3. TDD helps enforced code modularity.
4. Better understanding of what the code should be doing

SHOW GOT IT! What does TDD stand for?

Test-driven development

SHOW GOT IT! What are three steps of the TDD workflow?

1. Red
2. Green
3. Refactor

SHOW GOT IT! What does the developer do in the Red step in the TDD workflow?

Write the tests and watch them fail (a failing test is red). It's important to ensure the tests initially fail so that you don't have false positives.

SHOW GOT IT! What does a developer do in the Green step in the TDD workflow?

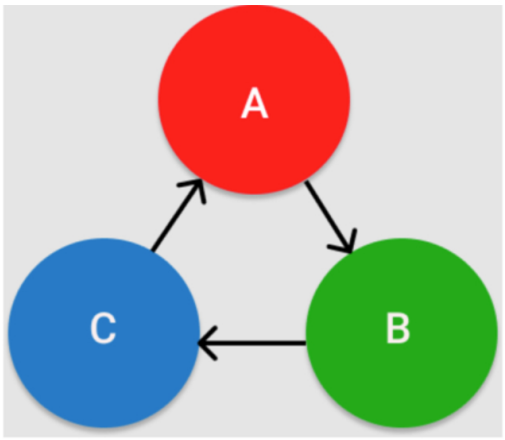
Write the minimum amount of code to ensure the tests pass (a passing test will be green).

SHOW GOT IT! What does a developer do in the Refactor step in the TDD workflow?

Refactor the code you just wrote. Your job is not over when the tests pass! One of the most important things you do as a software developer is to ensure the code you write is easy to maintain and read.

SHOW GOT IT!

The correct name of step C is:

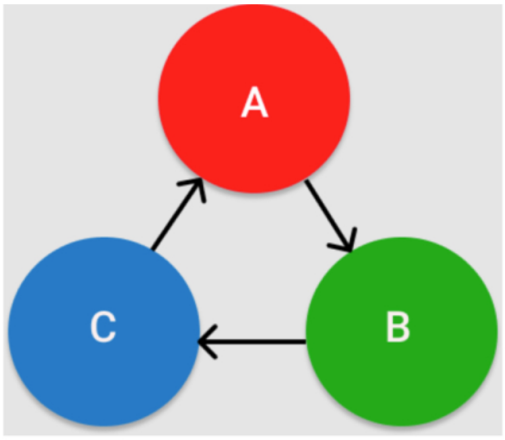


1. Green
2. Red
3. Refactor

#3: Refactor: The third step of the TDD workflow is Refactor where you refactor the code you wrote to ensure it is of the highest quality.

SHOW GOT IT!

The correct name of step A is:

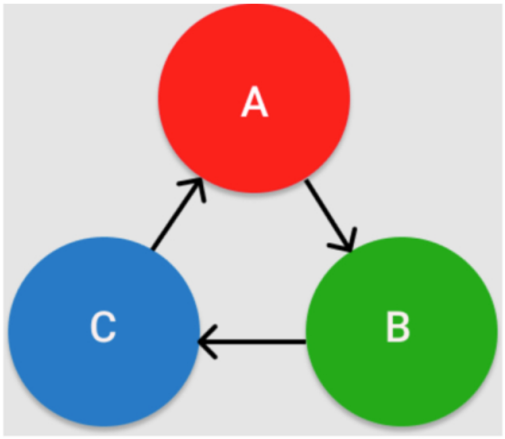


1. Refactor
2. Red
3. Green

#2: Red EXPLANATION: The first step of the TDD workflow is Red where you write a test and watch it fail.

SHOW GOT IT!

The correct name of step B is:



1. Red
2. Refactor
3. Green

#3: Green

SHOW GOT IT!

What is step 2 of the TDD workflow?

EXPLANATION: The second step of the TDD workflow is Green where you write the minimum amount of code required to pass the test.

SHOW GOT IT! Which of the following is true for the code below:

```
describe("sandwichMaker()", function() {
    context("given an invalid argument", function() {
        it("should throw a TypeError when given an invalid argument", function () {
            assert.throws(sandwichMaker(14), TypeError);
        });
    });
});
```

1. The `context` function's callback isn't being passed in a syntactically correct way.
2. The `sandwichMaker` function is being invoked with an argument that will throw an error - halting program execution.
3. The `assert.throws` method requires the error's message as well as the type of error.

#2: The `sandwichMaker` function is being invoked with an argument that will throw an error - halting program execution. To avoid this we could wrap the `sandwichMaker` function within another function.

SHOW GOT IT! Identify which function is invoked after every test within the same `describe` block?

1. `afterEach`
2. `before`
3. `after`
4. `beforeEach`

#1: The `afterEach` Mocha hook will be invoked AFTER each of the tests within the same block.

SHOW GOT IT! Identify which function is invoked before every test within the same `describe` block?

1. `beforeEach`
2. `after`
3. `before`
4. `afterEach`

The `beforeEach` Mocha hook will be invoked BEFORE EACH of the tests within the same block.

SHOW GOT IT!

How many times will the `before` hook be invoked?

```
describe("animalMakers", function() {
    before(function() {
      console.log("after");
    });

    describe("penguinMaker", function() {
      it("should make penguins", () => {});
    });

    describe("catMaker", function() {
      it("should make cats", () => {});
    });
  });
```

1. 1
2. 4
3. 2
4. 3

#1 EXPLANATION The `before` Mocha hook with be invoked once before all tests within the same block are run.

SHOW GOT IT!

What in the following code snippet is not according to Mocha writing best practices?

```
describe("makePopcorn()", function() {
    describe("when we have kernels", () => {
      it("should make popcorn", function() {
          // etc.
      });
    describe("when we don't have kernels", () => {
      it("should throw an error", function() {
          // etc.
      });
    });
  });
```

1. The above code snippet improperly nests `it` blocks withing `describe` blocks.
2. We are using `describe` blocks instead of `context` blocks when signifying the state of the function.
3. There should be an additional `describe` block wrapped around the entire test dictating how we are testing.

#2: EXPLANATION In the above code snippet we are setting two states (`have kernels` vs. `don't have kernels`) using two `describe` blocks. We should instead use the alias for `describe`, `context`, to signify we are testing two different contexts.

SHOW GOT IT!

Match the header fields of HTTP with the definitions

HTTP Headers:

1. Host	1. the address of the previous web page from which a link to the currently requested page was followed
2. User-Agent	2. specifies the domain name of the server
3. Referer	3. Indicates the media type found in the body of the HTTP message
4. Accept	4. a string that identifies the operating system, software vendor or version of the requester
5. Content-Type	5. informs the server about the types of data that can be sent back

1. Host: specifies the domain name of the server.
2. User-Agent: a string that identifies the operating system, software vendor or version of the requester
3. Referer: the address of the previous web page from which a link to the currently requested page was followed.
4. Accept: informs the server about the types of data that can be sent back.
5. Content-Type: Indicates the media type found in the body of the HTTP message

SHOW GOT IT!

Match HTTP verbs to their common uses

1. GET	1. similar to PUT but applies partial modifications to a resource
2. POST	2. sends data to the server creating a new resource.
3. PUT	3. deletes the specified resource.
4. PATCH	4. a request to retrieve data. It will never have a body.
5. DELETE	5. updates a resource on the server.

1. GET: a request to retrieve data. It will never have a body.
2. POST: sends data to the server creating a new resource.
3. PUT: updates a resource on the server.
4. PATCH: similar to PUT, but it applies partial modifications to a resource.
5. DELETE: deletes the specified resource.

SHOW GOT IT!

Match common HTTP status codes (200, 302, 400, 401, 402, 403, 404, 500) to their meanings.

HTTP response status codes indicate whether a specific HTTP request has been successfully completed.

1. 200:	1. Forbidden. The client does not have access rights to the content.
2. 302:	2. Bad Request. The server could not understand the request due to invalid syntax.
3. 400:	3. Payment Required.

4. 401:
5. 402:
6. 403:
7. 404:
8. 500:

4. Not Found. The server cannot find the requested resource.
5. Unauthorized. The client must authenticate itself to get the requested response.
6. Internal Server Error. The range from 500-599 indicate server errors.
7. OK. The request has succeeded
8. Found. The URI of requested resource has been changed temporarily.

1. 200: OK. The request has succeeded.
2. 302: Found. The URI of requested resource has been changed temporarily.
3. 400: Bad Request. The server could not understand the request due to invalid syntax.
4. 401: Unauthorized. The client must authenticate itself to get the requested response.
5. 402: Payment Required.
6. 403: Forbidden. The client does not have access rights to the content.
7. 404: Not Found. The server cannot find the requested resource.
8. 500: Internal Server Error. The range from 500-599 indicate server errors.

SHOW GOT IT! How do you open a direct connection with a URL to manually send HTTP requests?

Request:

```
nc -v google.com 80
GET / HTTP/1.1
```

Response:
```
HTTP/1.1 200 OK
Date: Thu, 28 May 2020 20:50:17 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
<!doctype html>
<html>
</html>
```

SHOW GOT IT! What is the module needed for http requests?

```
const http = require('http');
```

SHOW GOT IT! What is the method to create an http request using the http module?

```
http.createServer(function (request, response) { }).listen(8080, function() {});
```

SHOW GOT IT! Write a very simple HTTP server using http in node with paths that will result in the common HTTP status codes:

```
    http.createServer(function(request, response) {
        if (request.url === '/') {
            response.writeHead(
                200,
                { 'Content-Type': 'text/html' }
            );
            response.write('<h1>OK</h1>');
            response.end();
        } else {
            response.writeHead(404);
            response.end();
        }
    }).listen(8080, function() {
        console.log(
            'listening for requests on port 8080...'
        );
    });
```

SHOW GOT IT! How do you instantiate a promise?

```
const myPromise = new Promise((resolve, reject) => {});
```

SHOW GOT IT! How would we cause sleep to execute in a synchronous fashion with other functions?

```
    const sleep = (millisenconds) => {
        return new Promise((resolve) => {
            setTimeout(resolve, milliseconds);
        });
    }
```

Now that setTimeout is wrapped in a promise we can use it in the synchronous fashion:

```
    sleep(1000)
        .then(() => doAThing())
        .then(() => sleep(1000))
        .then(() => doADifferentThing())
        .then(() => sleep(3000))
        .then(() => doAFinalThing());
        .catch((reject) => fixIt());
```

SHOW GOT IT! How would you read 3 files, if you don't care about the order you read them?

```
Promise.all([
        fs.readFile("d1.md", "utf-8"),
        fs.readFile("d2.md", "utf-8"),
        fs.readFile("d3.md","utf-8")
    ])
    .then((contents1, contents2, contents3) => {
        return contents1 + contents2 + contents3;
    })
    .then((concatted) => {
        console.log(concattted);
    });
```

SHOW GOT IT! How would you use promises with the async fs?

```
const fs = require('fs').promises
```

SHOW GOT IT! What module do you need to import to use fetch?

```
const fetch = require('node-fetch');
```

SHOW GOT IT! How would you use fetch API to make a Promise-based call to https://ifconfig.me/all.json, then console log the output?

```
    const fetch = require('node-fetch');

    fetch("https://ifconfig.me/all.json")
        .then((response) => {
            return response.json(); // convert from json
        })
        .then((data) => {
            console.log(data);
        });
```

SHOW GOT IT! Given 2 functions that return promises, how would you call them to execute synchronously?

```
    function slow() {
        return new Promise((resolve, reject) => {
            setTimeout(() =>
                resolve('That was slow.');
            }, 2000);
        });
    }

    function fast() {
        return new Promise((resolve, reject) => {
            setTimeout(() => {
                resolve('That was fast.');
            }, 1000);
        });
    }




    async function syncLike() {
        console.log(await slow());
        console.log(await fast());
    }
```

SHOW GOT IT! Use async/await to write a fetch call to https://ifconfig.me/all.json. Hint: you also need to convert the json into a POJO.

```
    const fetch = require('node-fetch');

    async function getIpAddress() {
        const response = await fetch("https://ifconfig.me/all.json");
        const ipInfo = await response.json();
        return ipInfo;
    }

    (async() => {
        const ipInfo = await getIpAddress();
    })();
```

SHOW GOT IT! How do you transform an IIFE into an async function?

```
(async() => {})();
```

SHOW GOT IT! When using the node-fetch module, what two steps must you take?

1. npm install node-fetch
2. `const fetch = require('node-fetch');`

SHOW GOT IT! What is the HTML syntax for including a javascript file?

```
<script async type="module" src="index.js"></script>
```

SHOW GOT IT! How do you include a style sheet in your HTML file?

```
<link rel="stylesheet" href="style.css">
```

SHOW GOT IT! What is the syntax for the first line in your html file?

<!DOCTYPE html>

SHOW GOT IT! How do you create a link on your HTML page?

```
<a href="https://marylark.com/work-samples">Quizzes</a>
```

SHOW GOT IT! What is the red factor for TDD

Red: We begin by writing a test (or tests) that specify what we expect our code to do. We run the test, to see it fail, and in doing so we ensure that our test won't be a false positive.

SHOW GOT IT! What is the Green factor for TDD?

We write the minimum amount of code to get our test to pass. This step may take just a few moments, or a longer time depending on the complexity of the task.

SHOW GOT IT! What is the Refactor step in TDD?

The big advantage of test driven development is that it means we always have a set of tests that cover our codebase. This means that we can safely make changes to our code, and as long as our tests still pass, we can be confident that the changes did not break any functionality. This gives us the confidence to be able to constantly refactor and simplify our code - we include a refactor step after each time we add a passing test, but it isn't always necessary to make changes.

SHOW GOT IT!

Match the definitions with their names below

- Syntax Error
- Reference Error
- TypeError

- These errors refer to times in our code where we reference a variable that is not available in the current scope.
- These errors refer to times in our code where we reference a variable of the wrong type. Modifying a value that cannot be changed, using a value in an inappropriate way, or an argument of an unexpected type being passed to a function, are all causes of TypeErrors.
- These errors refer to problems with the syntax of our code, they usually refer to either missing or rogue characters that cause the compiler to be able to understand the code we are feeding it.

- Syntax Error: These errors refer to problems with the syntax of our code, they usually refer to either missing or rogue characters that cause the compiler to be able to understand the code we are feeding it.
- Reference Error: These errors refer to times in our code where we reference a variable that is not available in the current scope.
- TypeError: These errors refer to times in our code where we reference a variable of the wrong type. Modifying a value that cannot be changed, using a value in an inappropriate way, or an argument of an unexpected type being passed to a function, are all causes of TypeErrors.

SHOW GOT IT! What module do you need to include to use Mocha's "assert" method?

```
const assert = require("assert");
```

SHOW GOT IT! When testing a file in Mocha, what step do you need to take so Mocha knows about the file?

```
const reverseString = require('../lib/reverse-string').reverseString;
```

SHOW GOT IT! How would you determine if a string was reversed in Mocha?

```
const assert = require("assert");
const reverseString = require('../lib/reverse-string').reverseString


    describe("reverseString", () => {
        it("should reverse simple strings", () => {
            assert.equal(reverseString("fun"), "nuf");
        });
        it("should throw a TypeError if it doesn't receive a string", () => {
            assert.throws(() => reverseString(0));
        });
    });
```

SHOW GOT IT! How do you check if a TypeError is thrown in Mocha?

```
assert.throws(() => reverseString());
```

SHOW GOT IT! What does BDD stand for?

Behavior Driven Development

SHOW GOT IT! What framework provides a more BDD style?

Chai, where rather than `assert.equal(foo, 'bar');` from the strict Mocha TDD-style, chai provides more human readable code `expect(foo).to.be.a('string');` or `foo.should.equal('bar')`

SHOW GOT IT! What does the "before" hook in Mocha provide?

SHOW GOT IT! What does the "before" hook in Mocha provide?

Runs once before the first test in this block.

```
describe('hooks', function() {
    before(function() {
        //to run once before the first test in this block.
    }
});
```

SHOW GOT IT! What does the "after" hook in Mocha provide?

Runs once after the last test in this block.

```
describe('hooks', function() {
    after(function() {
        //to run once after the last test in this block.
    }
});
```

SHOW GOT IT! What does the "beforeEach" hook in Mocha provide?

Runs before each test in this block.

```
describe('hooks', function() {
    beforeEach(function() {
        //to run before EACH test in this block.
    }
});
```

SHOW GOT IT! What does the "afterEach" hook in Mocha provide?

Runs after each test in this block

```
describe('hooks', function() {
    afterEach(function() {
        //to run after EACH test in this block.
    }
});
```