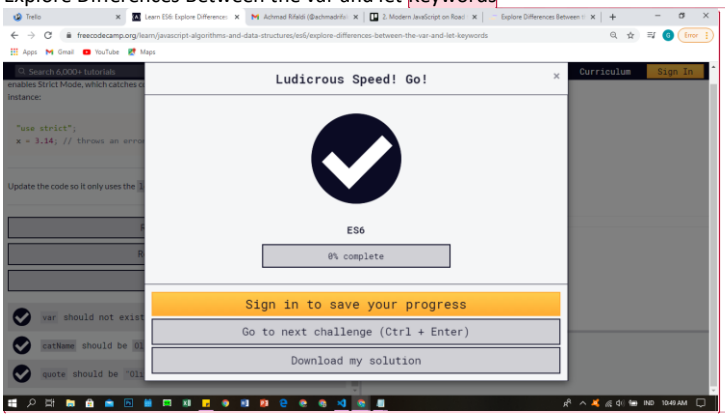
Date: 17/12/2020

https://www.freecodecamp.org/learn/javascript-algorithms-and-data-structures/es6/

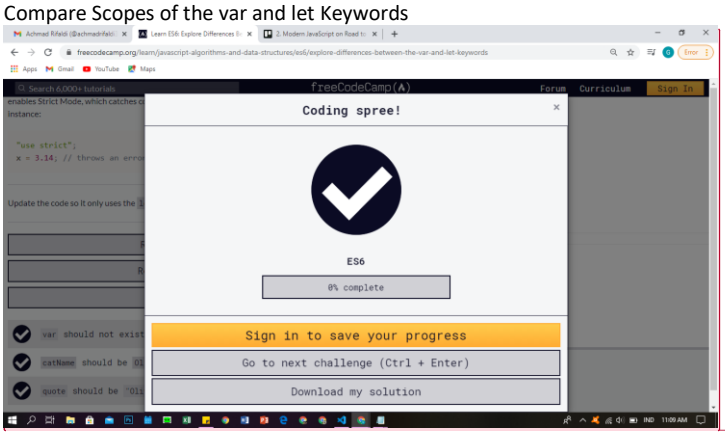
1. Explore Differences Between the var and let Keywords



Commented [WU1]:

Commented [WU2]:
```
let catName = "Jack";
let quote;

catName = "Queen";

function catTalk() {

  catName = "Oliver";
  quote = catName + " says Meow!";

}
catTalk();
```

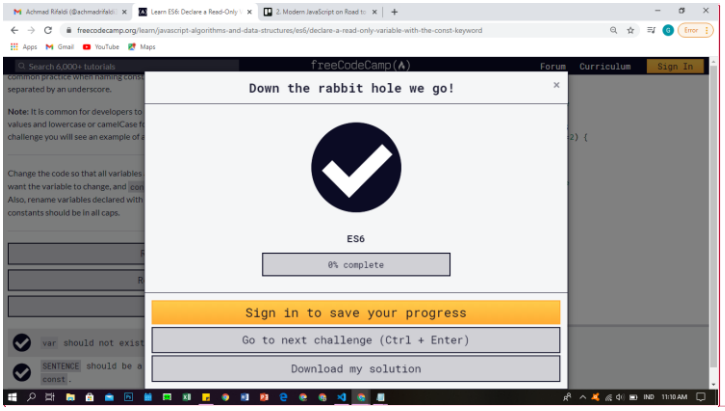
2. Compare Scopes of the var and let Keywords



Commented [WU3]:

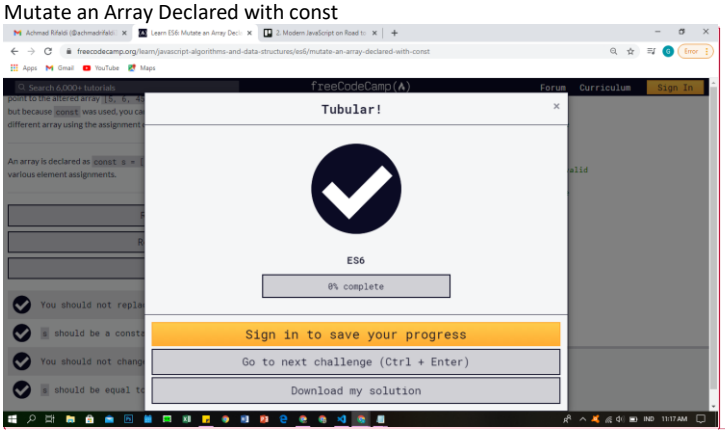
Commented [WU4]:
```
function checkScope() {

  let i = "function scope";
  if (true) {
    let i = "block scope";
    console.log("Block scope i is: ", i);
  }
  console.log("Function scope i is: ", i);
  return i;
}

checkScope();
```

3. Declare a Read-Only Variable with the const Keyword



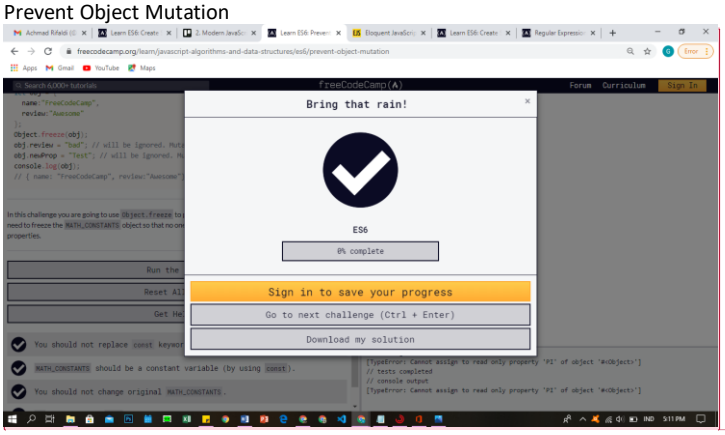
4. Mutate an Array Declared with const

Commented [WU5]:
```
function printManyTimes(str) {

  // Only change code below this line

  const SENTENCE = str + " is cool!";
  for (let i = 0; i < str.length; i+=2) {
    console.log(SENTENCE);
  }

  // Only change code above this line

}
printManyTimes("freeCodeCamp");
```



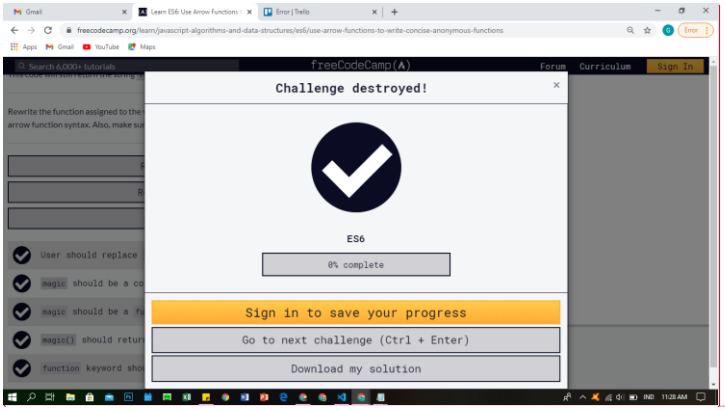
5. Prevent Object Mutation

Commented [WU6]:
```
const s = [5, 7, 2];
function editInPlace() {
  // Only change code below this line
s[0]=2;
s[1]=5;
s[2]=7;
  // Using s = [2, 5, 7] would be invalid

  // Only change code above this line
}
editInPlace();
console.log();
```



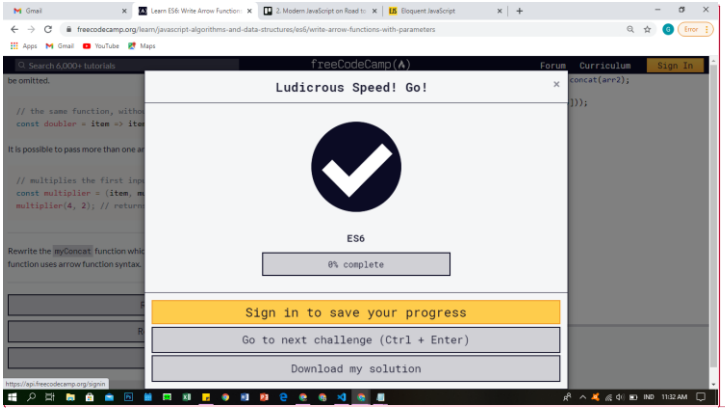
6. Use Arrow Functions to Write Concise Anonymous Functions

Commented [WU7]:
```
function freezeObj() {
  const MATH_CONSTANTS = {
    PI: 3.14
  };
  // Only change code below this line
Object.freeze(MATH_CONSTANTS);

  // Only change code above this line
  try {
    MATH_CONSTANTS.PI = 99;
  } catch(ex) {
    console.log(ex);
  }
  return MATH_CONSTANTS.PI;
}
const PI = freezeObj();
```



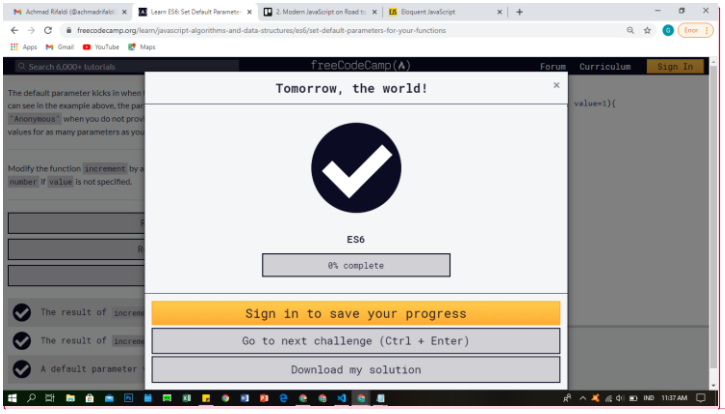
Commented [WU8]: `const magic = ()=>new Date( );`

7. Write Arrow Functions with Parameters



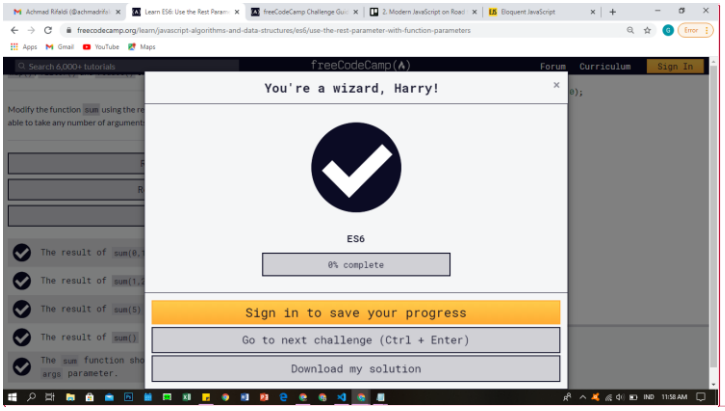
Commented [WU9]:
```
const myConcat = (arr1, arr2)=> arr1.concat(arr2);

console.log(myConcat([1, 2], [3, 4, 5]));
```

8. Set Default Parameters for Your Functions



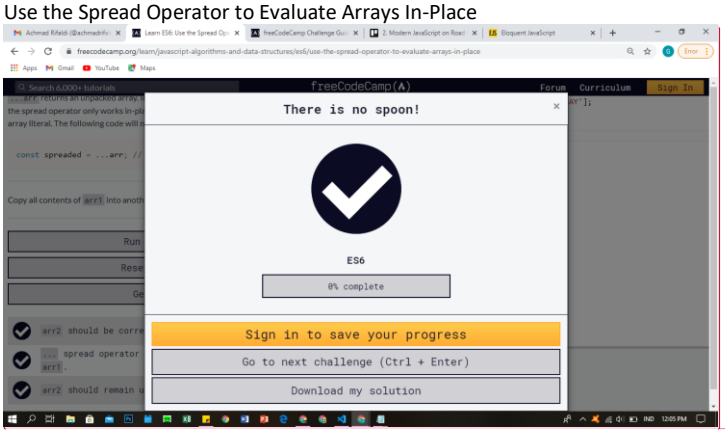
Commented [WU10]:
```
// Only change code below this line
const increment= (function(){
    return function increment(number, value=1){
        return number + value;

    };
})()

console.log(increment(5,2));
console.log(increment(5));
// Only change code above this line
```

9. Use the Rest Parameter with Function Parameters



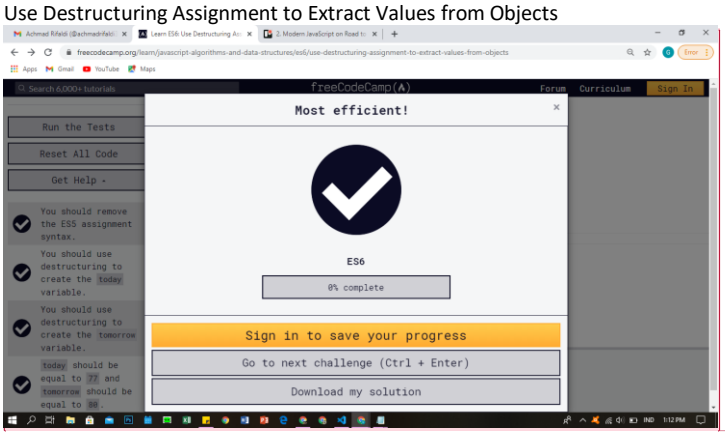
10. Use the Spread Operator to Evaluate Arrays In-Place

Commented [WU11]:
```
const sum = (...args)=>{
 return args.reduce((a, b) => a + b, 0);
}
console.log(sum(1, 2, 3)); // 6
```



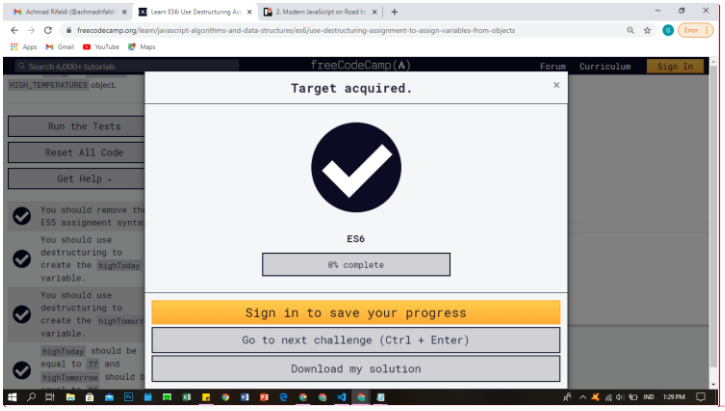
11. Use Destructuring Assignment to Extract Values from Objects

Commented [WU12]:
```
const arr1 = ['JAN', 'FEB', 'MAR', 'APR', 'MAY'];
let arr2;

arr2 = [...arr1];  // Change this line

console.log(arr2);
```



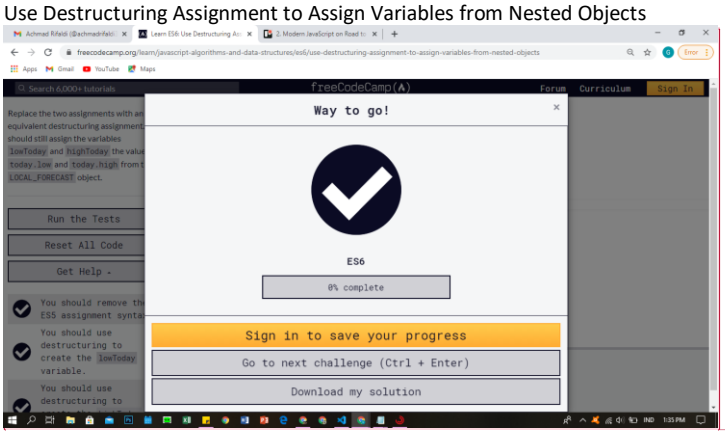
12. Use Destructuring Assignment to Assign Variables from Objects

Commented [WU13]:
```
const HIGH_TEMPERATURES = {
  yesterday: 75,
  today: 77,
  tomorrow: 80
};

// Only change code below this line
const {today,tomorrow} = HIGH_TEMPERATURES

// Only change code above this line

console.log(today);
console.log(tomorrow);
```



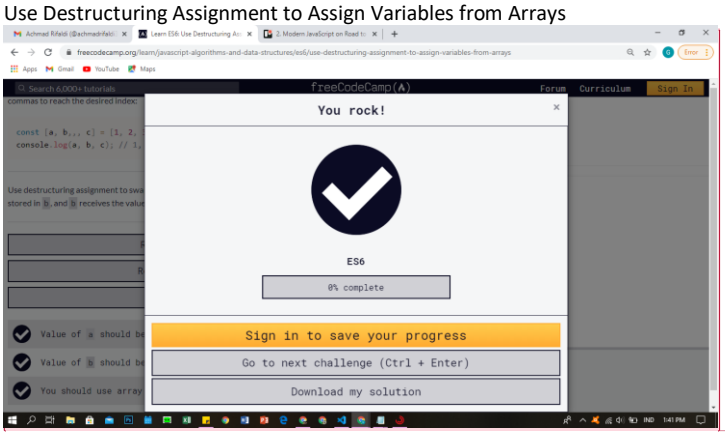
13. Use Destructuring Assignment to Assign Variables from Nested Objects

Commented [WU14]:
```
const HIGH_TEMPERATURES =
{
  yesterday: 75,
  today: 77,
  tomorrow: 80
};

// Only change code below this line

const{today:highToday, tomorrow: highTomorrow
 }= HIGH_TEMPERATURES;

// Only change code above this line
console.log(highToday);
```



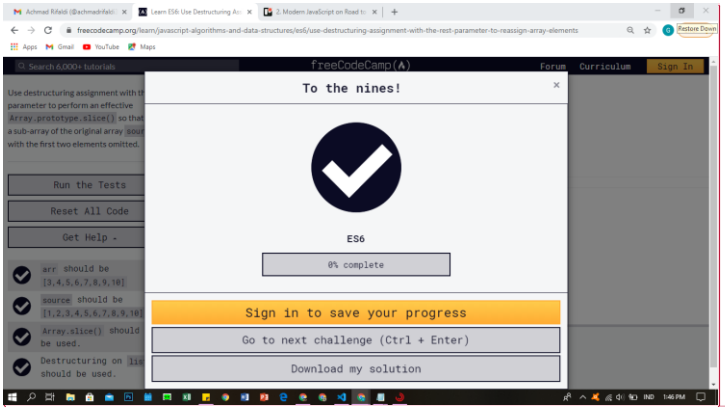
14. Use Destructuring Assignment to Assign Variables from Arrays

Commented [WU15]:
```
const LOCAL_FORECAST = {
  yesterday: { low: 61, high: 75 },
  today: { low: 64, high: 77 },
  tomorrow: { low: 68, high: 80 }
};

// Only change code below this line

const{today:{low:lowToday, high:highToday}}=
LOCAL_FORECAST;

// Only change code above this line
console.log(lowToday);
console.log(highToday);
```



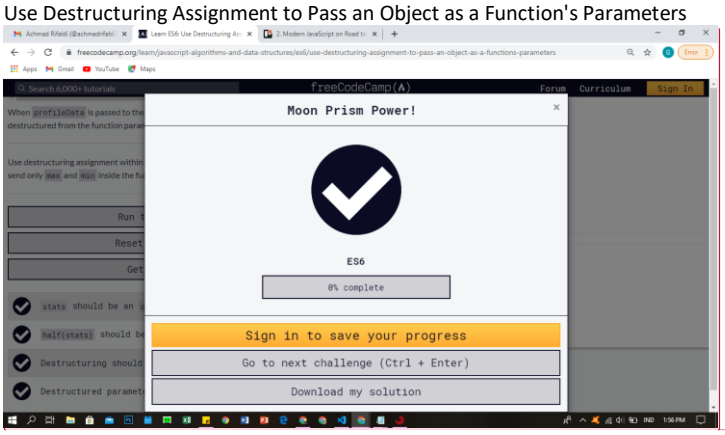
15. Use Destructuring Assignment with the Rest Parameter to Reassign Array Elements

Commented [WU16]:
```
let a = 8, b = 6;
// Only change code below this line
[a,b]=[b,a];

console.log(a);
console.log(b);
```



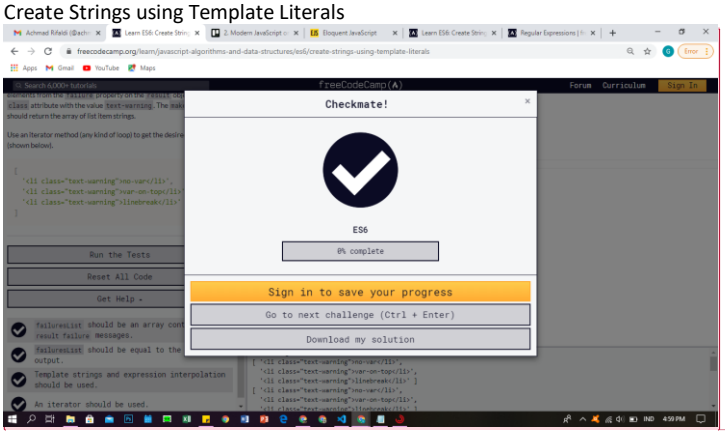
16. Use Destructuring Assignment to Pass an Object as a Function's Parameters

Commented [WU17]:
```
const source = [1,2,3,4,5,6,7,8,9,10];
function removeFirstTwo(list) {
  // Only change code below this line
  const [a, b, ...arr] = list; // Change this line
  // Only change code above this line
  return arr;
}
const arr = removeFirstTwo(source);
console.log(arr);
console.log(source);
```



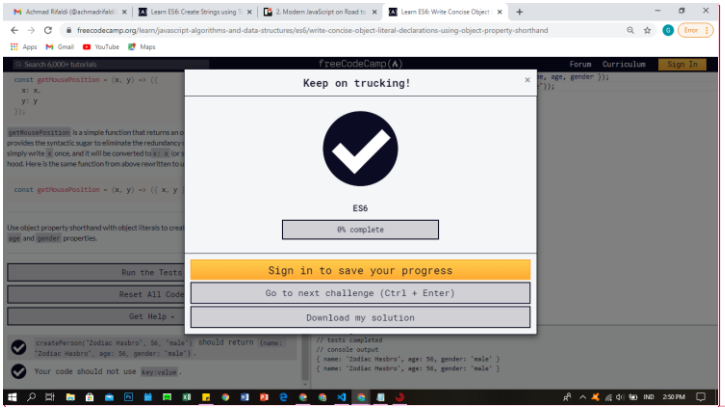
17. Create Strings using Template Literals

Commented [WU18]:
```
const stats = {
  max: 56.78,
  standard_deviation: 4.34,
  median: 34.54,
  mode: 23.87,
  min: -0.75,
  average: 35.85
};

// Only change code below this line
const half = ({max,min})=> {
  return (max + min)/2.0;
}

// Only change code above this line
console.log(stats);
console.log(half(stats));
```



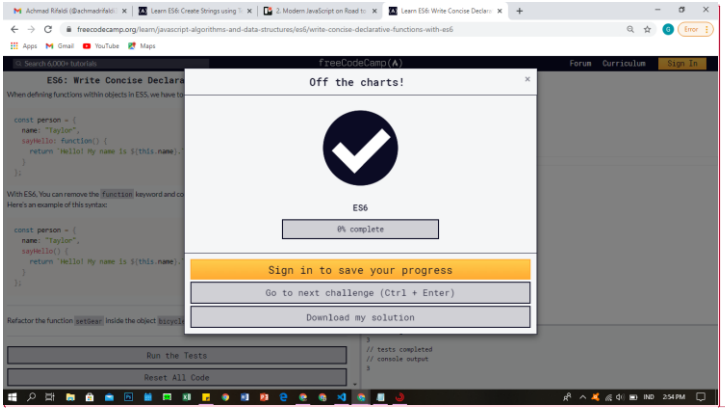
18. Write Concise Object Literal Declarations Using Object Property Shorthand

Commented [WU19]:
```
const result = {
  success: ["max-length", "no-amd", "prefer-arrow-functions"],
  failure: ["no-var", "var-on-top", "linebreak"],
  skipped: ["no-extra-semi", "no-dup-keys"]
};
function makeList(arr) {
  // Only change code below this line
  const failureItems = [];
 for (let i=0; i < result.failure.length; i++){
 failureItems.push(`<li class="text-warning">${arr[i]}</li>`);
 }


  // Only change code above this line
console.log(failureItems)
  return failureItems;
}

const failuresList = makeList(result.failure);
console.log(failuresList);
```



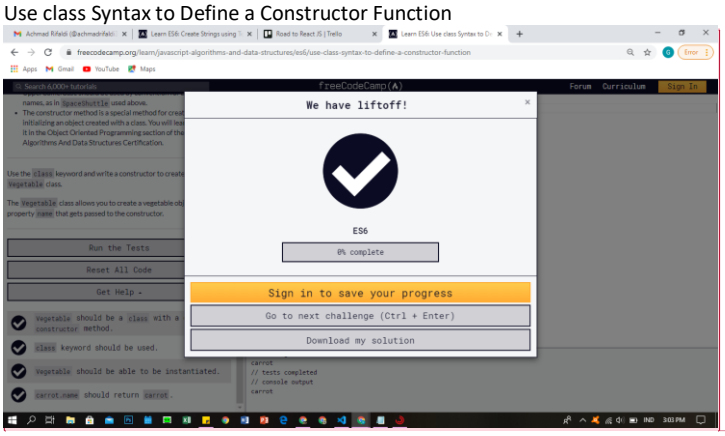
19. Write Concise Declarative Functions with ES6

Commented [WU20]:
```
const createPerson = (name, age, gender) => ( { name, age, gender });
console.log(createPerson("Zodiac Hasbro", 56, "male"));
```



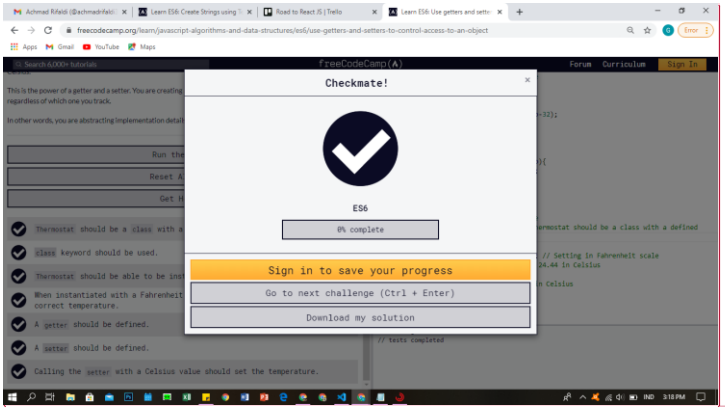
20. Use class Syntax to Define a Constructor Function

Commented [WU21]:
```
// Only change code below this line
const bicycle = {
  gear: 2,
  setGear(newGear) {
    this.gear = newGear;
  }
};
// Only change code above this line
bicycle.setGear(3);
console.log(bicycle.gear);
```



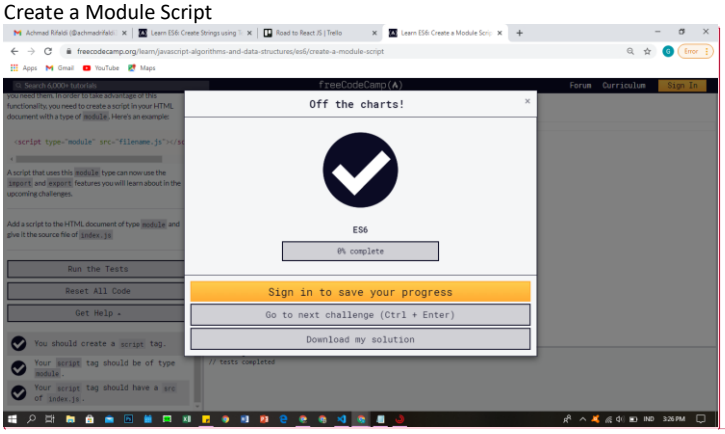
21. Use getters and setters to Control Access to an Object

Commented [WU22]:
```
// Only change code below this line
const Vegetable=makeClass()
function makeClass(){
    class Vegetable {
        constructor(name){
            this.name=name;

        }
    }
    return Vegetable;
}

// Only change code above this line

const carrot = new Vegetable('carrot');
console.log (carrot.name); // Should display 'carrot'
```



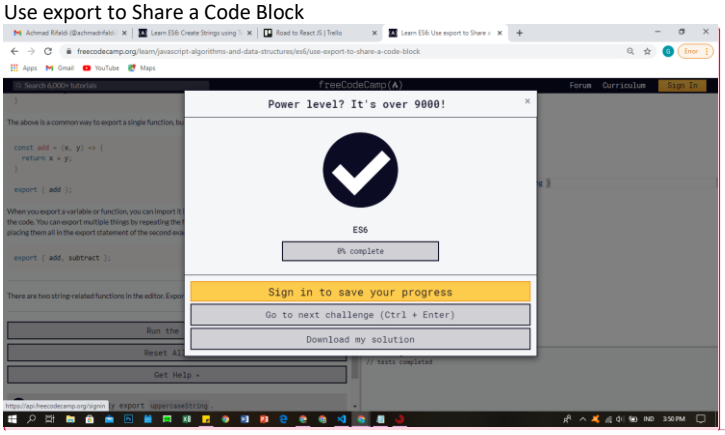
22. Create a Module Script

Commented [WU23]:
```
// Only change code below this line
function makeClass(){
    class Thermostat{
        constructor(temp){
            this._temp = 5/9 *(temp-32);
        }
        get temperature(){
            return this._temp;
        }
        set temperature(updatedTemp){
            this._temp=updatedTemp;
        }
    }
    return Thermostat;
}
// Only change code above this line
const Thermostat = makeClass();//Thermostat should be a class with a defined constructor method.

const thermos = new Thermostat(76); // Setting in Fahrenheit scale
let temp = thermos.temperature; // 24.44 in Celsius
thermos.temperature = 26;
temp = thermos.temperature; // 26 in Celsius
```



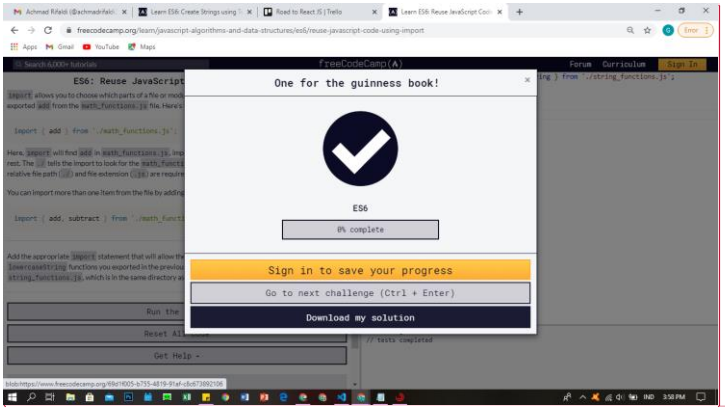
23. Use export to Share a Code Block

Commented [WU24]:
```
<html>
  <body>
    <!-- Only change code below this line -->
  <script type="module" src="index.js"></script>

    <!-- Only change code above this line -->
  </body>
</html>
```



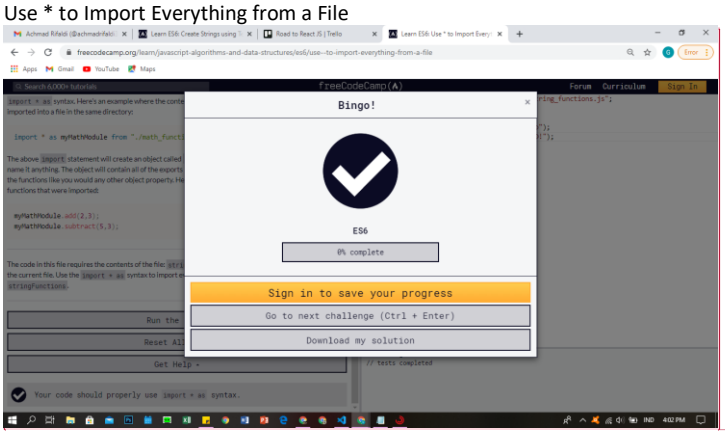
24. Reuse JavaScript Code Using import

Commented [WU25]:
```
const uppercaseString = (string) => {
  return string.toUpperCase();
}

const lowercaseString = (string) => {
  return string.toLowerCase()
}

export{uppercaseString, lowercaseString };
```



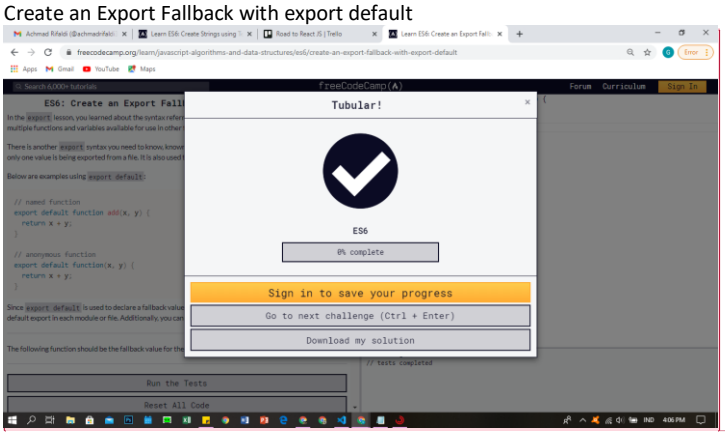
25. Use * to Import Everything from a File

Commented [WU26]:
```
import { uppercaseString, lowercaseString } from './string_functions.js';
// Only change code above this line

uppercaseString("hello");
lowercaseString("WORLD!");
```



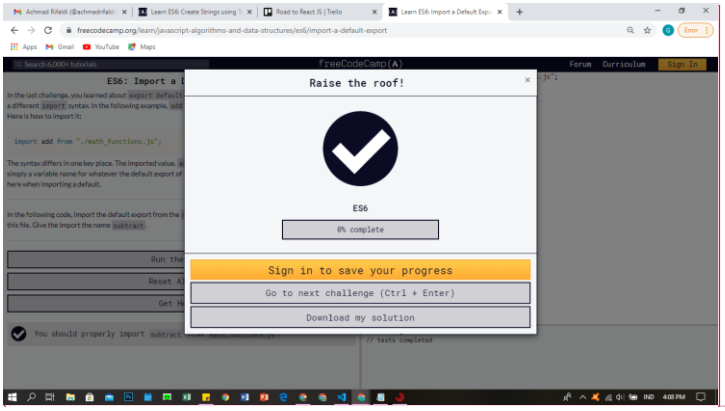
26. Create an Export Fallback with export default

Commented [WU27]:
```
import * as stringFunctions from "./string_functions.js";
// Only change code above this line

stringFunctions.uppercaseString("hello");
stringFunctions.lowercaseString("WORLD!");
```



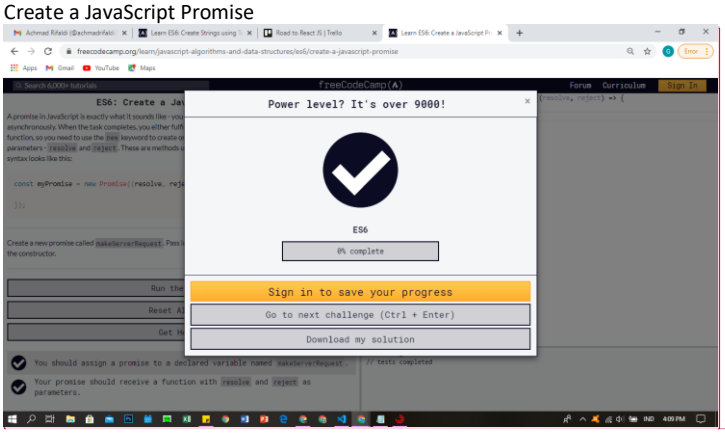
27. Import a Default Export

Commented [WU28]:
```
export default function subtract(x, y) {
  return x - y;
}
```



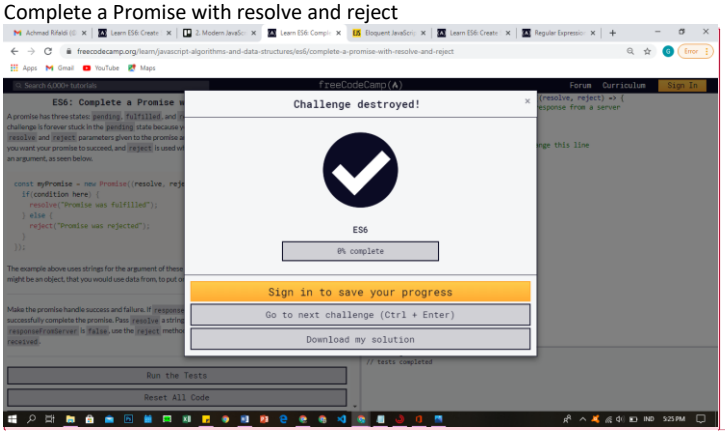
28. Create a JavaScript Promise

Commented [WU29]:
```
import subtract from "./math_functions.js";
// Only change code above this line

subtract(7,4);
```



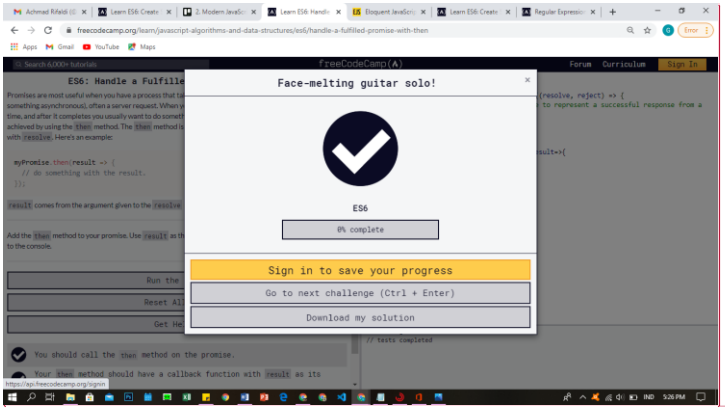
29. Complete a Promise with resolve and reject

Commented [WU30]:
```
const makeServerRequest = new Promise((resolve, reject) => {

});
```



30. Handle a Fulfilled Promise with then

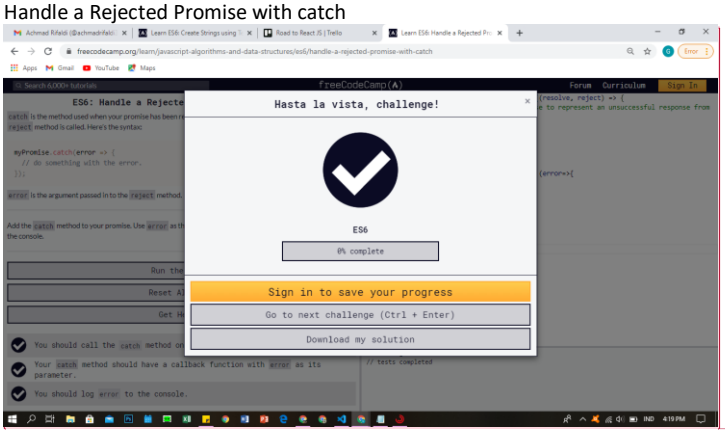
Commented [WU31]:
```
const makeServerRequest = new Promise((resolve, reject) => {
  // responseFromServer represents a response from a server
  let responseFromServer;

  if(responseFromServer) {
    resolve("We got the data"); // Change this line
  } else {
    reject("Data not received");
  }
});
```



Commented [WU32]:

```
const makeServerRequest = new Promise((resolve, reject) => {
  // responseFromServer is set to true to represent a successful response from a server
  let responseFromServer = true;

  if(responseFromServer) {
    resolve("We got the data").then(result=>{
      console.log(result);
    });
  } else {
    reject("Data not received");
  }
});
```

31. Handle a Rejected Promise with catch



Commented [WU33]:

```
const makeServerRequest = new Promise((resolve, reject) => {
  // responseFromServer is set to false to represent an unsuccessful response from a server
  let responseFromServer = false;

  if(responseFromServer) {
    resolve("We got the data");
  } else {
    reject("Data not received").catch (error=>{
      console.log(error);
    });
  }
});

makeServerRequest.then(result => {
  console.log(result);
});
```