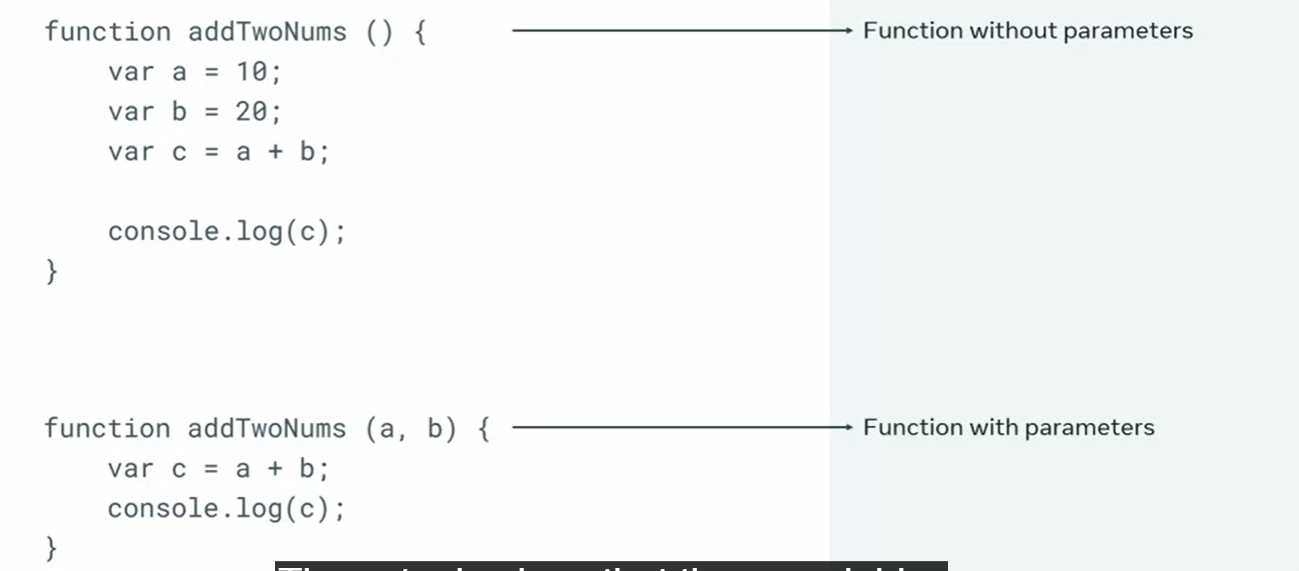
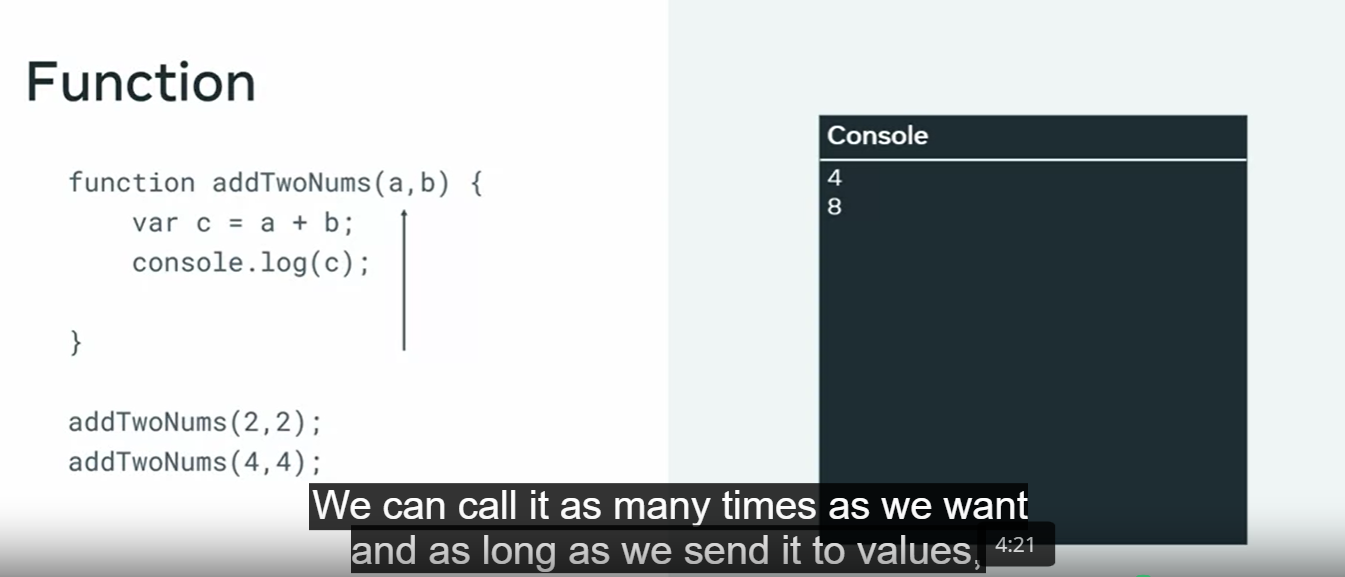
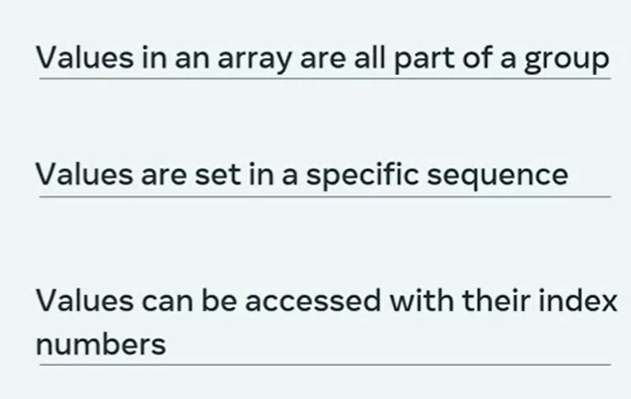
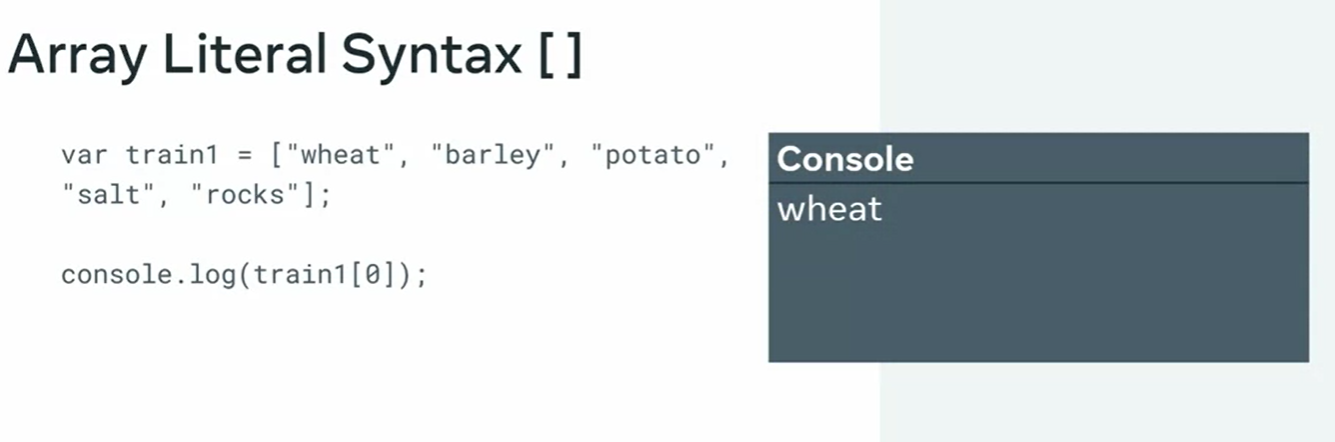
**Functions, Arrays and Objects**

**Functions**

* DRY in other words don't repeat yourself. And it's thanks to functions that you can avoid repetition
* function declaration (build) and function invocation (call)
* With functions you can take several lines of code that performs a set of related actions and then group them together under a single label. Then when you need to run the code that you've saved, you just invoke or call the function. You can run the code as many times as you want.
* 
* 
* Function addTwoNums(a,b) {} => a,b function parameters
* addTwoNums(2,2) => function arguments

**Arrays**

****

* 
* Array index starts with 0
* Array literal syntax: [ ]
* 

**Program to iterate through an elements in array using functions**

| function listArrayItems(arr) {  for (var i = 0; i < arr.length; i++) {  console.log(arr[i]);  console.log(i, arr[i]);  console.log(i+1, arr[i])  }  }  var colors = ['red', 'orange', 'yellow', 'green', 'blue', 'purple', 'pink'];  listArrayItems(colors); //display all items in the array at once |
| --- |

function listArrayItems(arr) {

for (var i = 0; i < arr.length; i++) {

if (arr[i] == 'red') {

console.log(i\*100, "tomato!")

} else {

console.log(i\*100, arr[i])

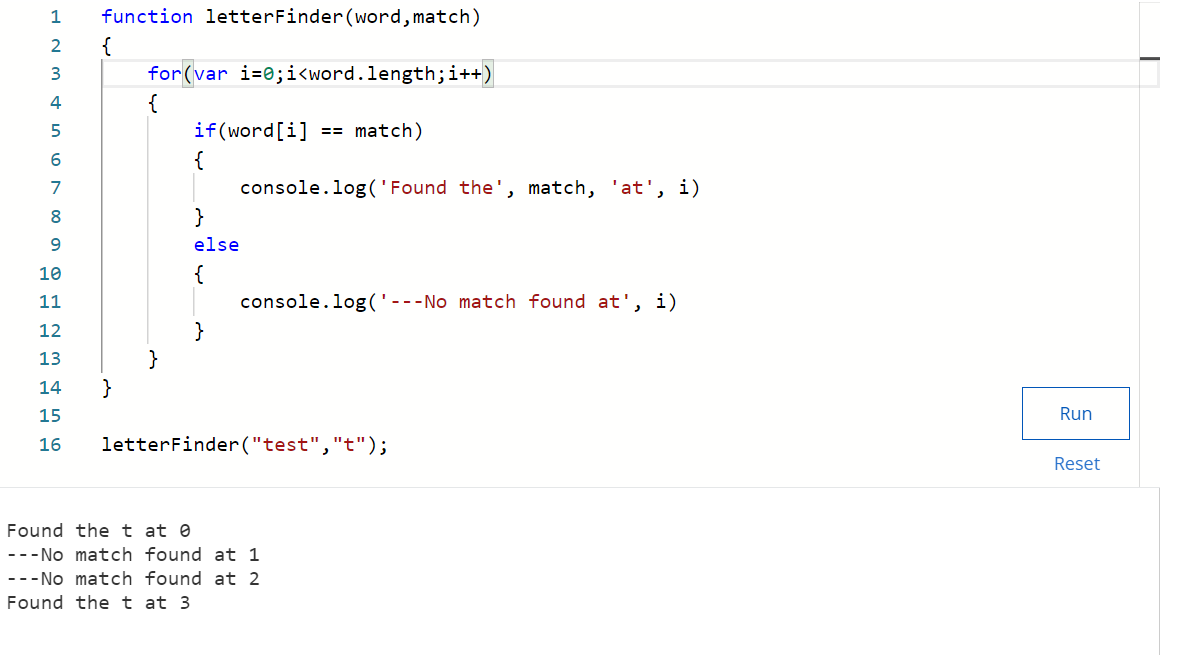
}

}

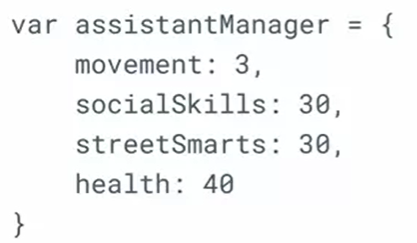
}

| red  orange  yellow  green  blue  purple  pink | 0 'red'  1 'orange'  2 'yellow'  3 'green'  4 'blue'  5 'purple'  6 'pink' | 1 'red'  2 'orange'  3 'yellow'  4 'green'  5 'blue'  6 'purple'  7 'pink' | 0 'tomato!'  100 'orange'  200 'yellow'  300 'green'  400 'blue'  500 'purple'  600 'pink' |
| --- | --- | --- | --- |

Exercise



**Objects**

* Objects can be described as collections of related properties where each property is represented as a key value pair.
* object literal syntax: **{}**.
* In programming, if you have groups of data that you would like to relate, you can assign them to something known as an object.
* Groups of variables related to each other
* 
* assistantManager.nextAchievement = “get promoted” => new property added to previous ones
* var user = {}; //create an object
* //creating an object with properties and their values

var assistantManager = {

rangeTilesPerTurn: 3,

socialSkills: 30,

streetSmarts: 30,

health: 40,

specialAbility: "young and ambitious",

greeting: "Let's make some money"

}

| var table = {  legs: 3,  color: "brown",  priceUSD: 100,  }  console.log(table);//display the object in the developer console  Output: {legs: 3, color: 'brown', priceUSD: 100}  console.log(table.color); // 'brown' |
| --- |

* An alternative approach of building objects is to first save an empty object literal to a variable, then use the dot notation to declare new properties on the fly, and use the assignment operator to add values to those properties;

var house2 = {};

house2.rooms = 4;

house2.color = "pink";

house2.priceUSD = 12345;

console.log(house); // {rooms: 3, color: "brown", priceUSD: 10000}

house.windows = 10;

console.log(house); // {rooms: 3, color: "brown", priceUSD: 10000, windows: 10}

house.windows = 11;

console.log(house); // {rooms: 3, color: "brown", priceUSD: 10000, windows: 11} update already existing properties

**Object literals and the brackets notation**

* There is an alternative syntax to the dot notation I used up until this point.
* This alternative syntax is known as *the brackets notation*.

var house2 = {};

house2["rooms"] = 4;

house2['color']= "pink";

house2["priceUSD"] = 12345;

console.log(house2); // {rooms: 4, color: 'pink', priceUSD: 12345}

var car = {};

car.color = "red";

car["color"] = "green";

car["speed"] = 200;

car.speed = 100;

console.log(car); // {color: "green", speed: 100}

* With the brackets notation, I can add space characters inside property names,

car["number of doors"] = 4;

console.log(car); // {color: 'green', speed: 100, number of doors: 5}

* I can add numbers (as the string data type) as property keys:

car["2022"] = 1901;

console.log(car); // {2022: 1901, color: 'green', speed: 100, number of doors: 5}

* there's one really useful thing that bracket notation has but is not available in the dot notation: It can evaluate expressions.

var arrOfKeys = ['speed', 'altitude', 'color'];

var drone = {

speed: 100,

altitude: 200,

color: "red"

}

for (var i = 0; i < arrOfKeys.length; i++) {

console.log(drone[arrOfKeys[i]])

}

Output:

100

200

red

**Arrays are objects**

* In JavaScript, arrays are objects. That means that arrays also have some built-in properties and methods.
* One of the most commonly used built-in methods on arrays are the **push()** and the **pop()** methods.

var fruits = [];

fruits.push("apple"); // ['apple']

fruits.push('pear'); // ['apple', 'pear']

To remove the last item from an array, I can use the **pop()** method:

fruits.pop();

console.log(fruits); // ['apple']

function arrayBuilder(one, two, three) {

var arr = [];

arr.push(one);

arr.push(two);

arr.push(three);

console.log(arr);

}

arrayBuilder('apple', 'pear', 'plum'); // ['apple', 'pear', 'plum']

-return array

function arrayBuilder(one, two, three) {

var arr = [];

arr.push(one);

arr.push(two);

arr.push(three);

return arr;

}

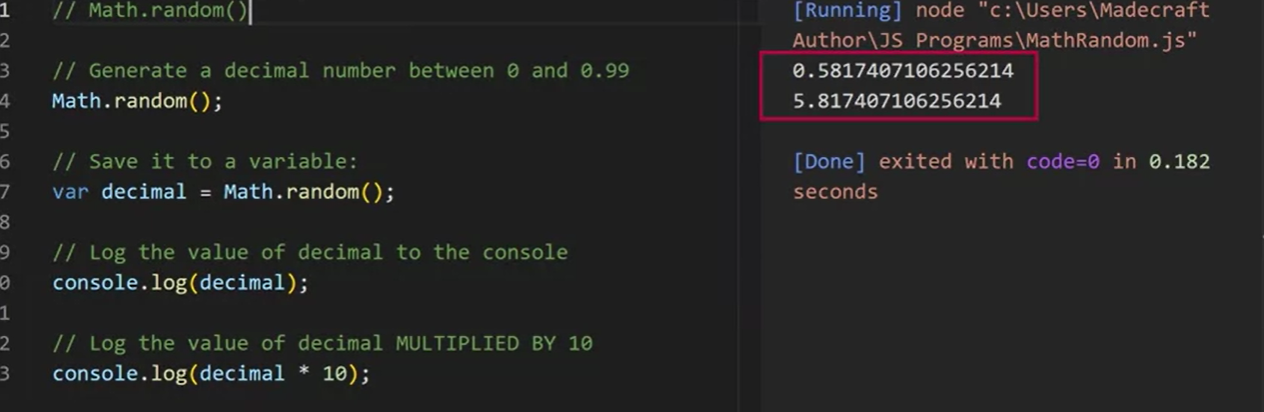
var simpleArr = arrayBuilder('apple', 'pear', 'plum');

console.log(simpleArr); // ['apple','pear','plum']

**popular built-in objects is the Math object**

* The PI number: Math.PI which is approximately 3.14159
* The Euler's constant: Math.E which is approximately 2.718
* The natural logarithm of 2: Math.LN2 which is approximately 0.693
* **Math.ceil()** - rounds up to the closest integer
* **Math.floor()** - rounds down to the closest integer
* **Math.round()** - rounds up to the closest integer if the decimal is **.5** or above; otherwise, rounds down to the closest integer
* **Math.trunc()** - trims the decimal, leaving only the integer
* **Math.pow(2,3)** - calculates the number **2** to the power of **3**, the result is **8**
* **Math.sqrt(16)** - calculates the square root of **16**, the result is **4**
* **Math.cbrt(8)** - finds the cube root of **8**, the result is **2**
* **Math.abs(-10)** - returns the absolute value, the result is **10**
* Logarithmic methods: **Math.log()**, **Math.log2()**, **Math.log10()**
* Return the minimum and maximum values of all the inputs: **Math.min(9,8,7)** returns **7**, **Math.max(9,8,7)** returns **9**.
* Trigonometric methods: **Math.sin()**, **Math.cos()**, **Math.tan()**, etc.

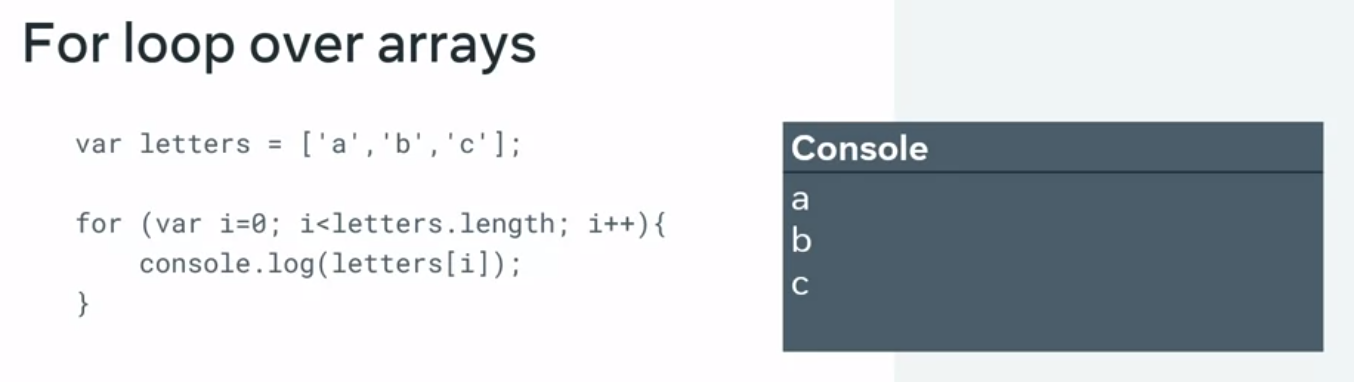
**Math.random()**

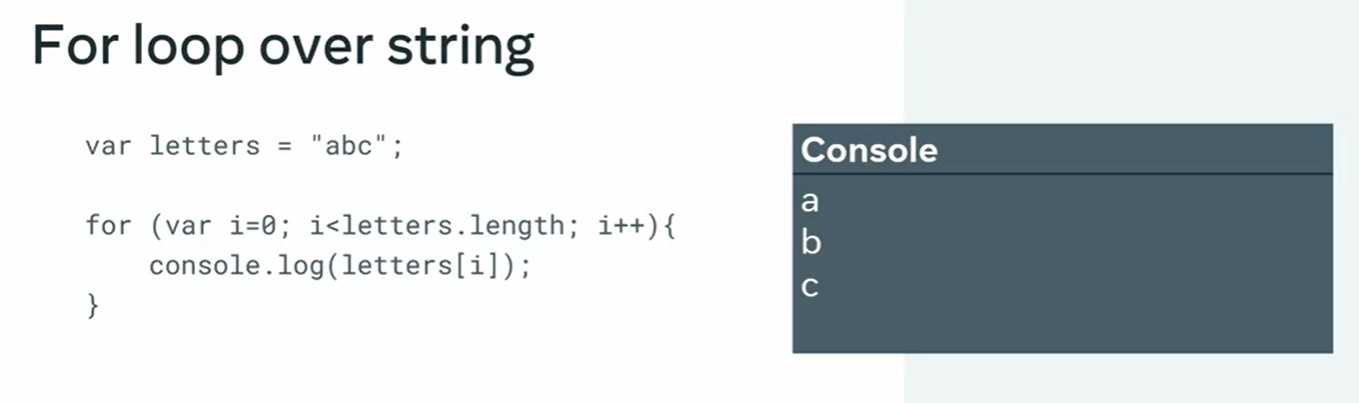
* A part of the Math object that can generate a number between 0 and 0.99
* If you want to output a number larger than 1, you must multiply the variable by 10
* 

**Ceil method**

* A part of the math object that rounds a decimal upto the nearest integer
* var rounded = Math.ceil(0.0001) => 1 - ceil method can round only upwards
* var rounded = Math.ceil(0.5) => 1 - ceil method can round only upwards
* var rounded = Math.ceil(0.99) => 1 - ceil method can round only upwards
* var rounded = Math.ceil(1.01) => 2
* var rounded = Math.ceil(1.5) => 2
* var rounded = Math.ceil(2.99) => 3

**String**

****

****

var greet = "Hello, ";

greet.length; // 7

greet.charAt(0); // 'H'

“Wo".concat("rl").concat("d"); // 'World'

The **indexOf** returns the location of the first position that matches a character:

"ho-ho-ho".indexOf('h'); // 0

"ho-ho-ho".indexOf('o'); // 1

"ho-ho-ho".indexOf('-'); // 2

The **lastIndexOf** finds the last match, otherwise it works the same as **indexOf**.

The **split** method chops up the string into an array of sub-strings:

"ho-ho-ho".split("-"); // ['ho', 'ho', 'ho']

greet.toUpperCase(); // "HELLO, "

greet.toLowerCase(); // "hello, "

**Object Method**

* Variable and methods in object

var car = {};

car.mileage = 98765;

car.color = "red";

console.log(car);

car.turnTheKey = function() {

console.log("The engine is running")

}

car.lightsOn = function() {

console.log("The lights are on.")

}

console.log(car);

car.turnTheKey();

car.lightsOn()

**Output:**

{ mileage: 98765, color: 'red' }

{

mileage: 98765,

color: 'red',

turnTheKey: [Function (anonymous)],

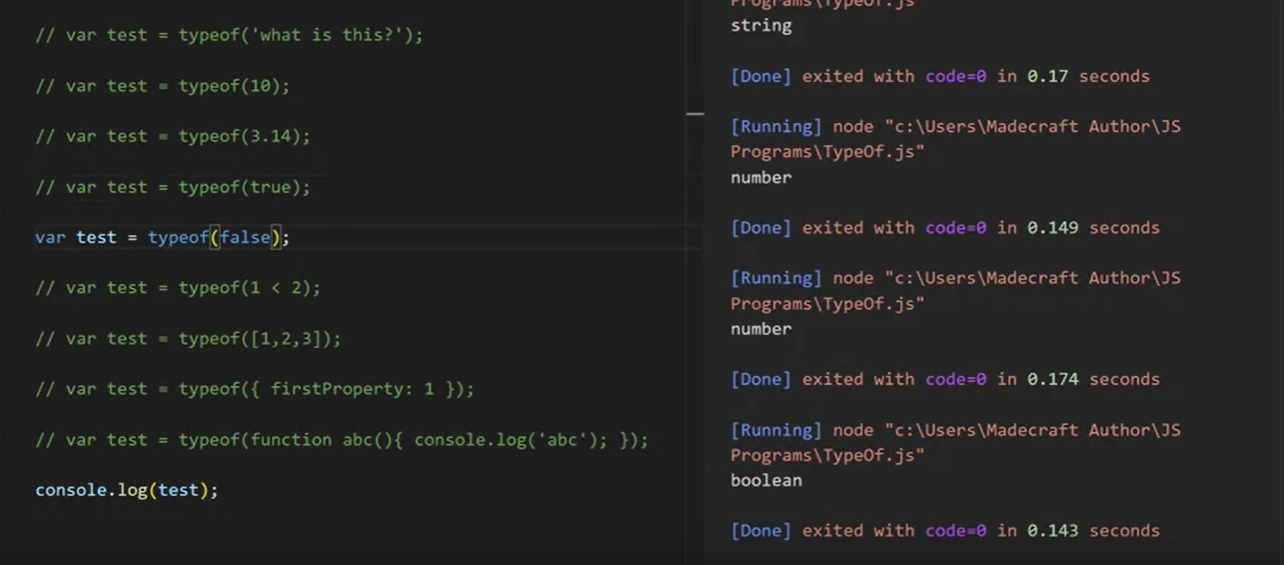
lightsOn: [Function (anonymous)]

}

The engine is running

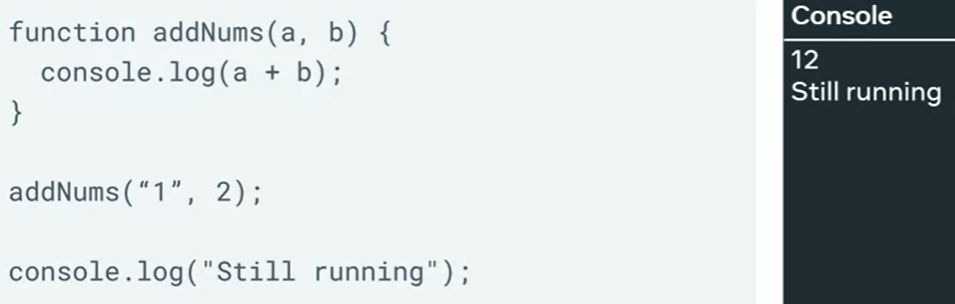
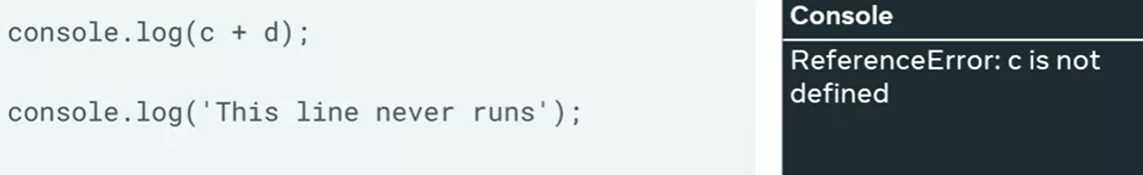
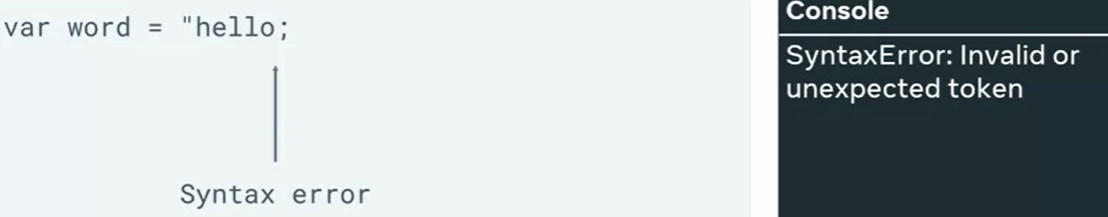
The lights are on.

**Typeof**

* The type of operator accepts and evaluates a parameter and returns the name of the data type represented as a string.
* 

Output: boolean, typeof(1<2) boolean, **typeof([1,2,3])is object because array in Javascript is an object,** object, function

**Error Handling**

* When a bug happens, our program keeps running, but it behaves in a way we didn't intend.
* When an error happens, our program stops running as the result of an error.
* Example for bug: program continue to run
* 
* Example for Reference Error as no values exists for variables c and d and program stops running and next console log line never going to be executed.
* ReferenceError can occur when a value is not defined, but you attempt to use it in your code.
* 
* SyntaxError occurs when you write a piece of code that JavaScript cannot read.
* 
* TypeError can occur when you try to use a value in an inappropriate way.
* Array method on number cause TypeError
* 

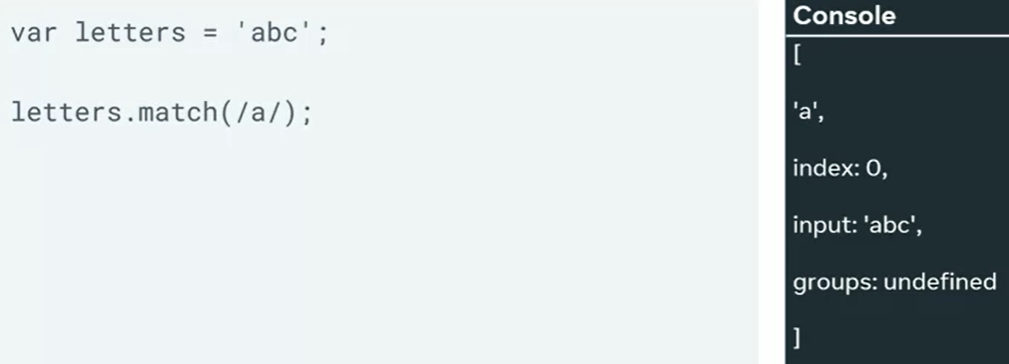
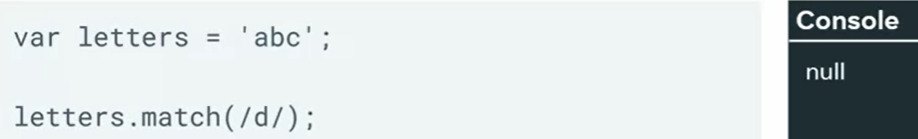
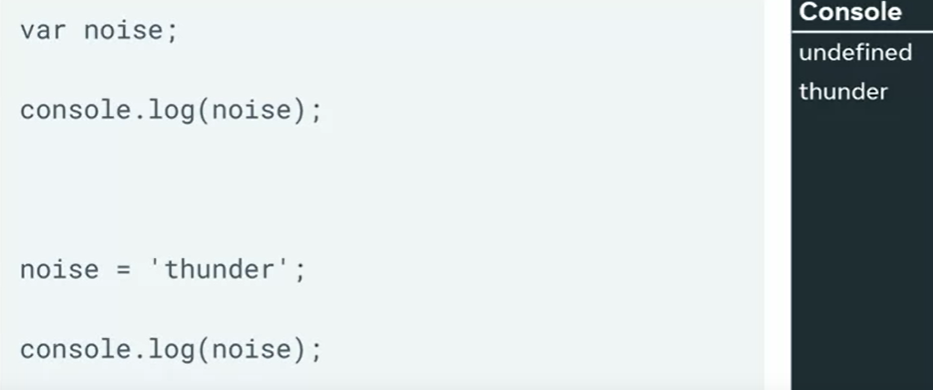
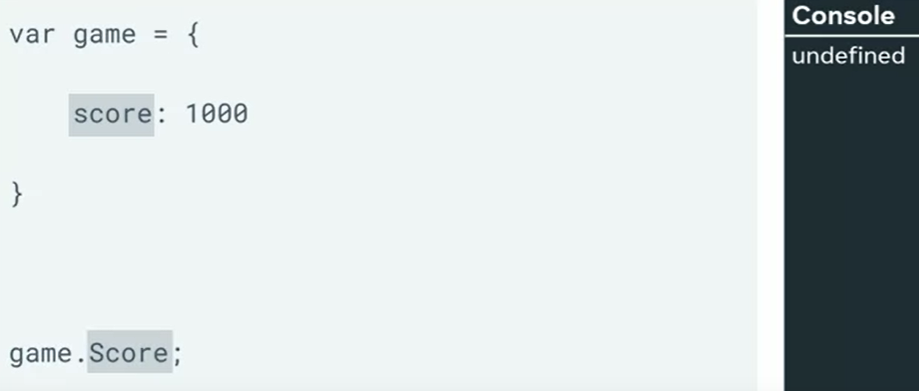
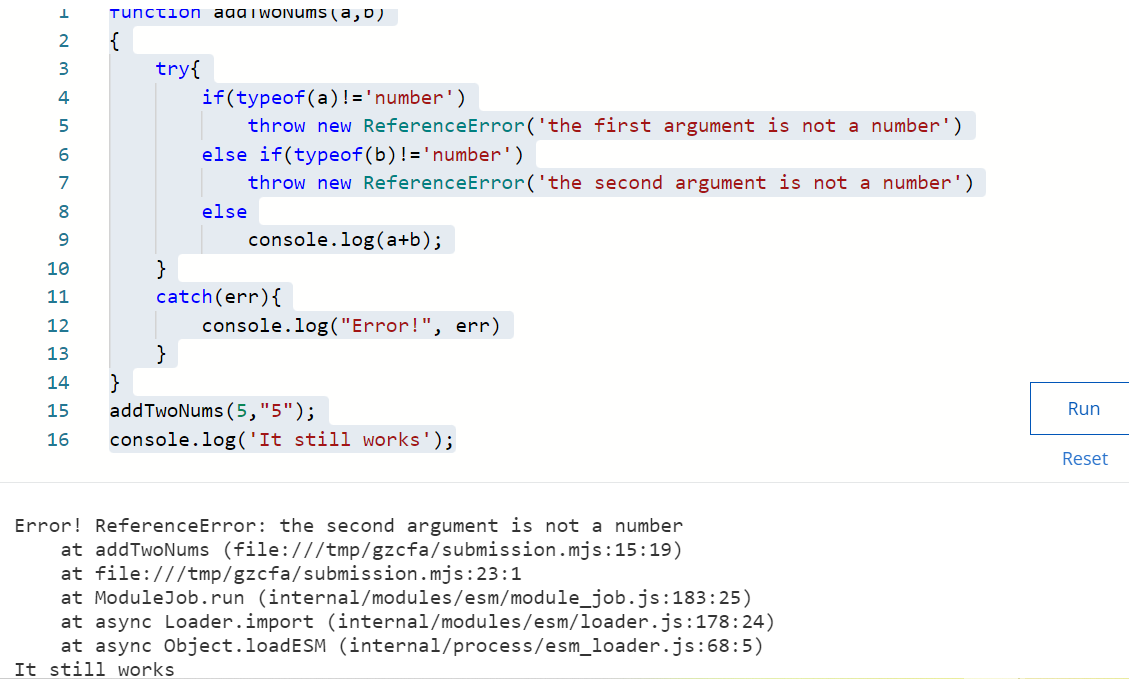
**Throw, Try & Catch**

* Program wont stop with reference error using try and catch
* 
* Throw, try and catch
* 
* **Using the throw keyword, you can force an error to be thrown from the try block to the catch block**. It's important to remember that **you can use the throw keyword outside the try block, but it will not be possible to catch it.**
* The catch block accepts something called an **error which is an object**. This is the actual error that is thrown from the try block.
* throw new Error();
* console.log("Hello"); //output: nothing will printout
* Throwing an error will stop the execution of the code.
* Correct code
* try {
* throw new Error();
* console.log('Hello');
* } catch(err) {
* console.log('Goodbye');
* }
* output:Goodbye

**Working with errors in Javascript**

* Test a block of code for errors using the TRY CATCH statement
* the most common errors in JavaScript:
* ReferenceError
* SyntaxError
* TypeError
* RangeError
* There are some other errors in JavaScript. These other errors include:
* AggregateError
* Error
* InternalError
* URIError
* **RangeError** A RangeError is thrown when we're giving a value to a function, but that value is out of the allowed range of acceptable input values.
* (10).toString(2); // '1010' The value of 2 when passed to the toString() method, is like saying to JavaScript: "convert the value of 10 of the Base 10 number system, to its counter-part in the Base 2 number system".
* (10).toString(100); // Uncaught RangeError: toString() radix argument must be between 2 and 36

**Undefined, null and empty values**

* Types of empty values: null, undefined and empty strings
* **Null** - intentional absence of any object value. Return value of match method is an array which is object in JS
* 
* 
* **Undefined** - In JavaScript, there may also be times when you are building something that hasn't been clearly defined yet and so you can't assign a value to it. Fortunately, there is a way to store it so that you can assign it later using the undefined datatype
* Undefined data type hold only one value ie undefined
* String, boolean
* By default all functions will return undefined, After console log u will always see undefined this is because we are not returning any value in console.log by default it is returning undefined after displaying the output
* 
* Unassigned variable have undefined value
* 
* Didn’t declare variable noise2
* 
* **undefined datatype acts like a placeholder for a value that the JavaScript engine knows to exist.**
* Another scenario of undefined is when you try to **access an object property that doesn't exist.** Score and score are different variables
* 
* var obj ={
* 'key1':"hi"
* }
* console.log(obj1.name);
* console.log(obj1.name);
* ^
* ReferenceError: obj1 is not defined
* 

**Defensive programming**

* Defensive programming is all about *assuming* that all the arguments a function will receive are of the wrong type, the wrong value or both.

**Exercise**

