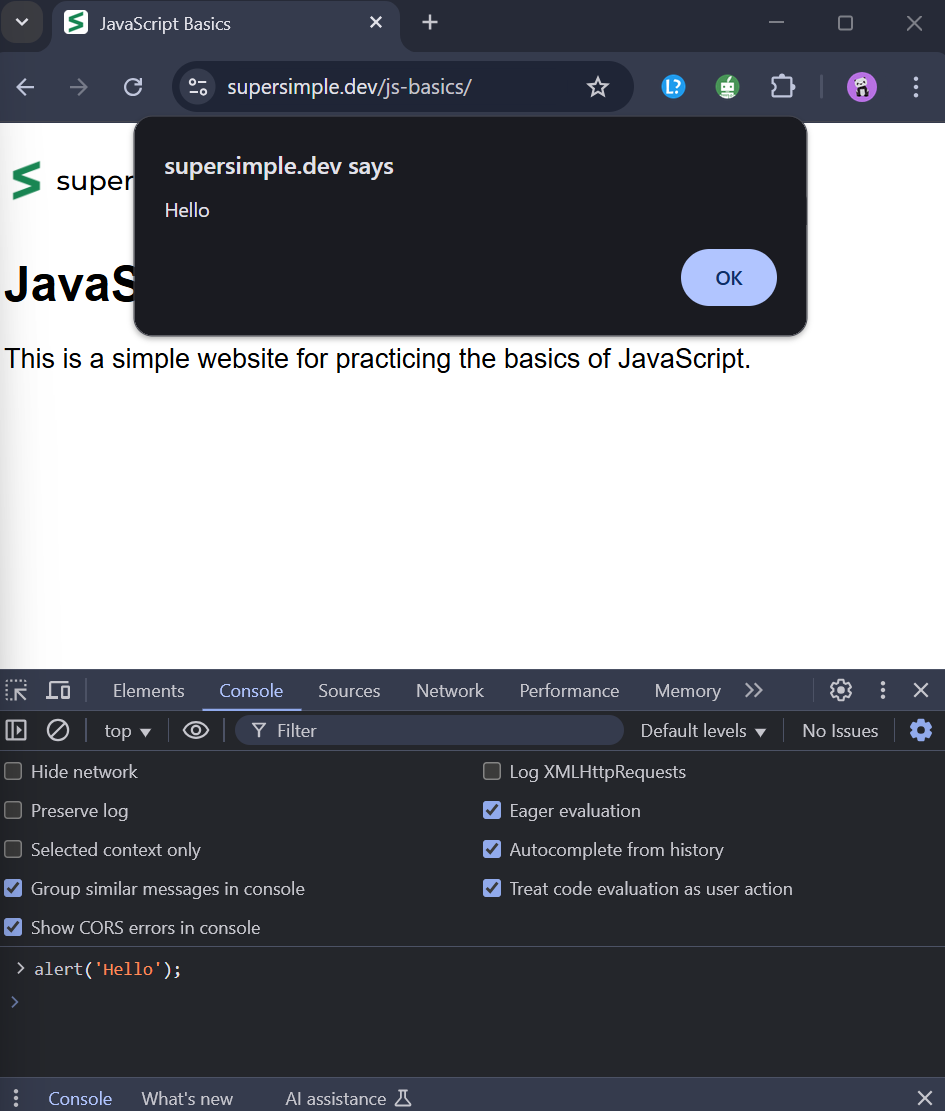
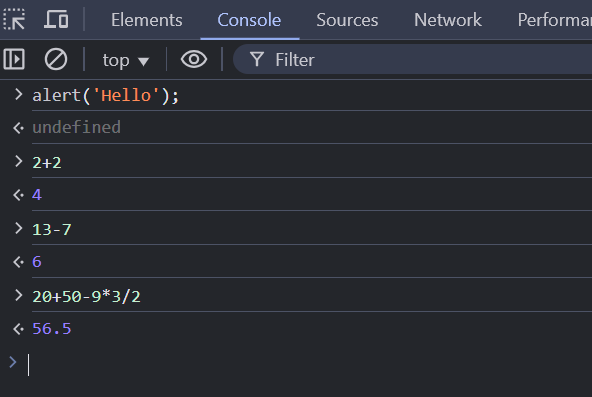
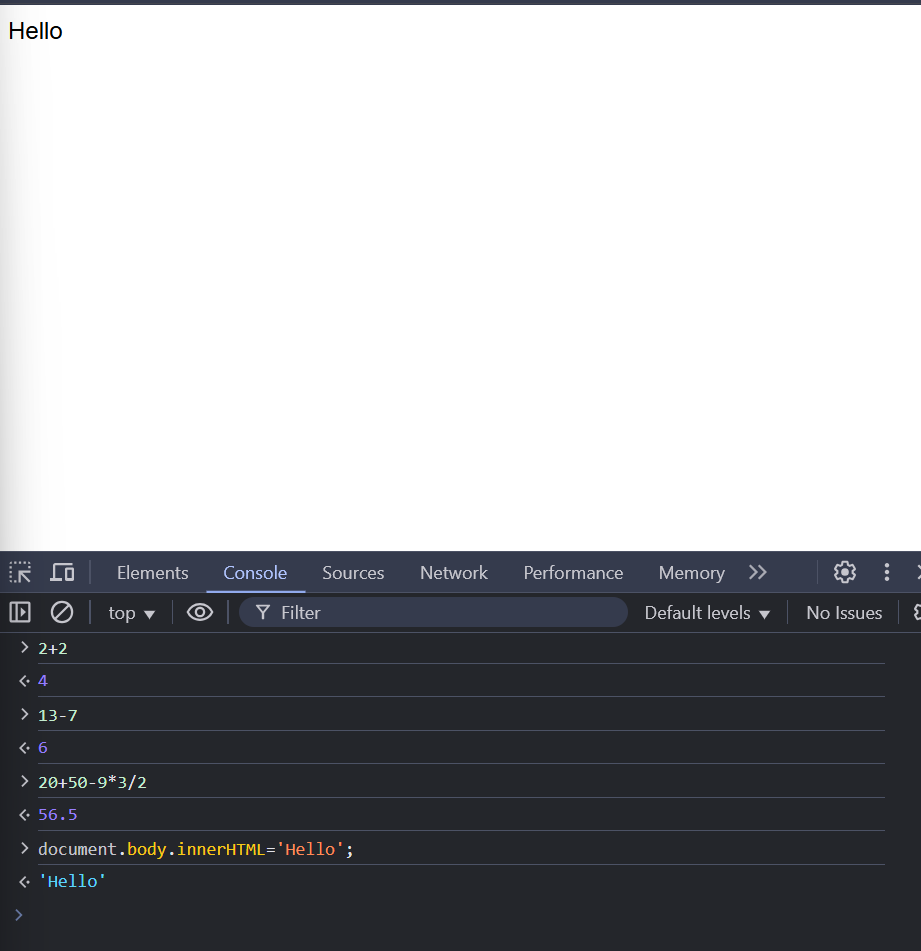
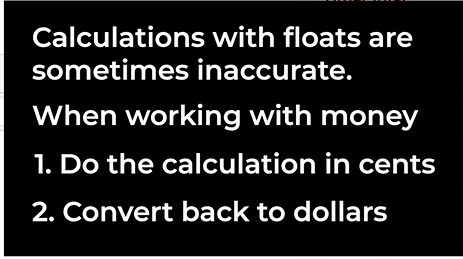
**JAVA SCRIPT**

# 1. Basics of JavaScript:

* What is JavaScript?
  1. A Technology that we use to create websites.
* JavaScript is case-sensitive.
* To create a pop-up with the text “Hello” inside it we use the following instruction:  
   alert(‘Hello’);  
   
* Math in JavaScript  
  
* In the below image, we have given an instruction to the browser to remove all the content in the webpage and replace it with the text inside the quotes (‘ ‘).  
   
* In the above example, we used JavaScript to modify the web page. Modifying the webpage is one of the most important features of JavaScript.

## 2. Numbers and Math:

* Order of operations:  
   Mathematical operations in JavaScript follows a priority order when found in a single expression.  
  For example, consider an expression –> 2+2\*3, This gives us the output as 8 (2+2\*3=> 2+6=8). But, we might usually think the answer would be 12 which is 2+2\*3=>4\*3=12. But JavaScript doesn’t work that way.   
  JavaScript follows a priority order (or) operator precedence when solving this kind of problem, and the order is: / (Division) and \* (Multiplication) have the same priority so if an expression has both the operators, the operation is done from left to right. Similarly, - (Subtraction) and + (Addition) have the same priority so if an expression has both the operators, the operation is done from left to right.
* Similar to some other programming languages, JavaScript also have a problem in storing some floating point values. This is due to the inaccuracy caused during the conversion of a decimal number to a binary number. For example, 0.2 in decimal system is stored as 0.199999972060322762   
   
* We can use Math.round(n.m) to round a number to the nearest integer in JavaScript where n.m is a float.

## 3. Strings:

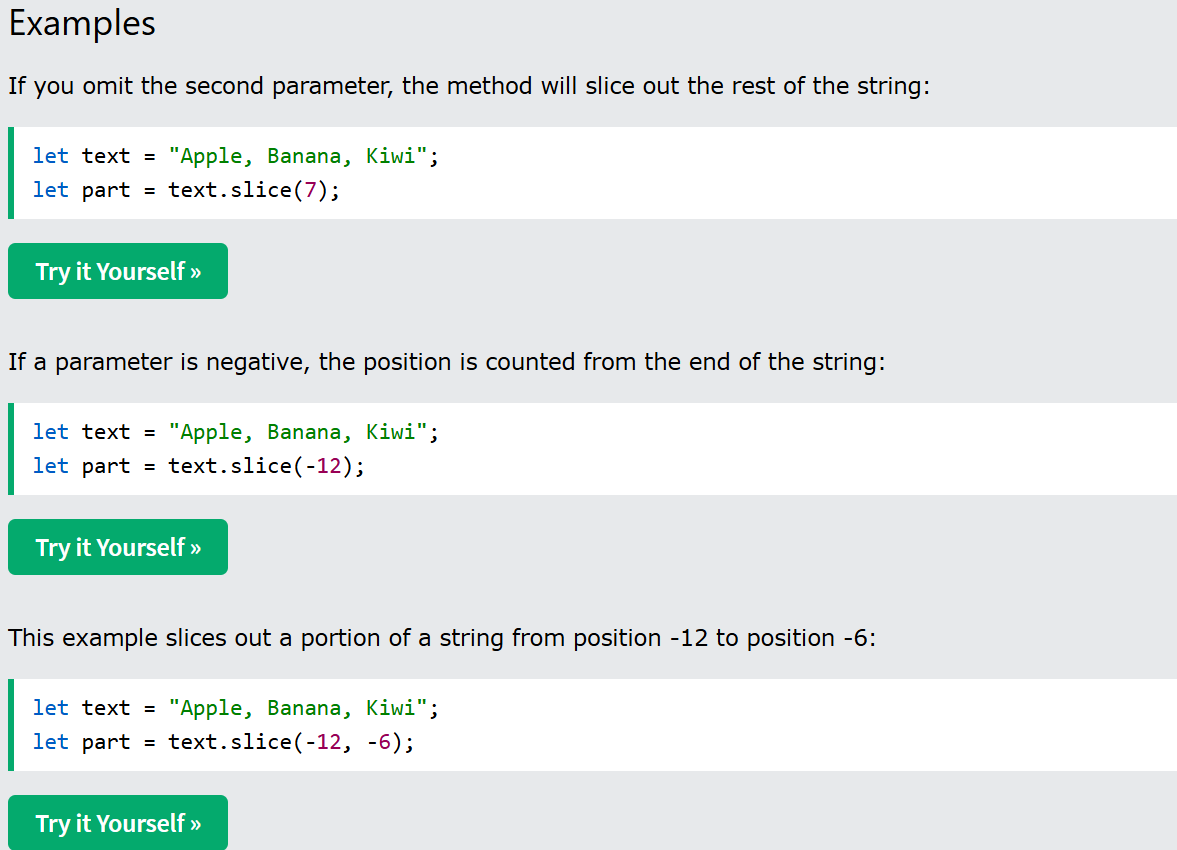
* A string in JavaScript is written by enclosing in a pair of single or double quotes.

Eg: ‘Hello’

* Adding multiple strings together: ‘some’ + ‘more’ + ‘time’ = somemoretime  
  This method of combining multiple strings is called as **Concatenation.**
* We use the ***typeof*** keyword to know the data type of a given value.  
  Eg: typeof 2 -> ‘number’  
   typeof ‘hi’ -> ‘string’
* ‘hello’+3 -> ‘hello3’ That is, when you add a string and an integer, it gives you the result as a string. This is called ***Type Coercion*** (Automatic type conversion).
* We can create strings in JavaScript in 3 ways:
  1. Using Single quotes (‘…’)
  2. Using Double quotes (“…”)
  3. Using Back ticks (`…`). The strings created by using back ticks is called as ***template strings***. Because they have some special features.  
     1. The first special feature of template string is called ***Interpolation.*** Interpolation lets us insert a value directly into a string.  
      The combination of characters **${}** is used to insert value directly into string  
      For eg, `Items (${1+1}): $${(2095+799)/100}` -> ‘Items (2): $28.94’

2. The second special feature of template string is Multi-line strings  
 For eg, `some  
 text` -> ‘some\ntext’ (next line escape character is returned)

* String Methods:
  + String length:  
     The length property returns the length of a string.  
     Eg: *let text = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";  
     let length = text.length; ----- > 26*
  + Extracting String Characters:  
     There are 4 methods for extracting string characters:
* The at(*position*) Method
* The charAt(*position*) Method
* The charCodeAt(*position*) Method
* Using property access [] like in arrays
  + - * **at():***const name = "W3Schools";  
        let letter = name.at(2); ------- > S*
      * **charAt():***let text = "HELLO WORLD";  
        let char = text.charAt(0); ------- > H*
      * **charCodeAt():***let text = "HELLO WORLD";  
        let char = text.charCodeAt(0); ------- > 72*
      * **[ ]:***const name = "W3Schools";  
        let letter = name[2]; ------- > S*
  + Extracting String Parts:  
     There are 3 methods for extracting a part of a string:
    - * **Slice(start,end):**slice() extracts a part of a string and returns the extracted part in a new string.  
        Eg: *let text = "Apple, Banana, Kiwi";  
         let part = text.slice(7, 13); ------- > Banana*

******

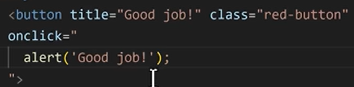
* **Substring():**substring() is similar to slice(). The difference is that, the start and end values less than 0 are treated as 0 in substring().  
  Eg: *let str = "Apple, Banana, Kiwi";  
   let part = str.substring(7, 13); ------ > Banana*
* Converting to Upper and Lower case:
  + **toUpperCase():***let text1 = "Hello World!";  
    let text2 = text1.toUpperCase(); --------------- > HELLO WORLD*
  + **toLowerCase():***let text1 = "Hello World!";          
    let text2 = text1.toLowerCase(); --------- > hello world*
* Concat():
  + **concat():**joins two or more strings.  
    Eg: *let text1 = "Hello";  
     let text2 = "World";  
     let text3 = text1.concat(" ", text2); -->Hello World*

**For more String Methods, refer ----** [**stringMethods**](https://www.w3schools.com/js/js_string_methods.asp)

# 4. HTML CSS Review, console.log

- We use *<script>* tag to run JavaScript code inside a HTML code. We add that tag inside the body of HTML code.

- We also use *“onclick”* attribute to run JavaScript code, whenever we click that button.   
 onclick=””, Inside the double quotes we can write some JavaScript.  
 For Example,

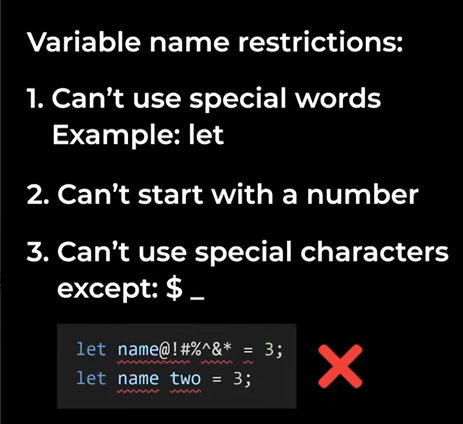
  
the output will be, when you click the red button, it gives the pop-up with the text inside the alert. i.e, in this example it is “Good Job!”.  
- **Comments** are the pieces of code that the computer ignores.

* Double slash (//) is used to write a comment in JavaScript.
* A multi-line comment is written between the pair of slash and asterisk (/\*\*/).
* console.log() method is used to display anything inside the closed brackets on to the console.

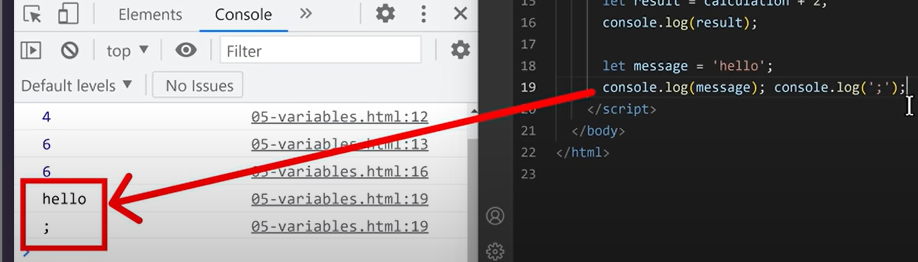
## 5. Variables:

- A variable is like a container. We can save a value or a string inside a variable and then use it later.

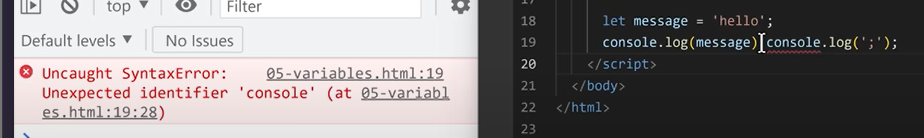
* The keyword ***let*** or ***var*** is used to create a new variable in JavaScript.  
  Eg: let num1=3;

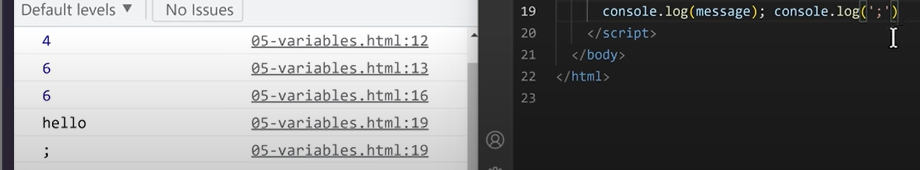


* Semi-colon in JavaScript is used to indicate the end of an instruction.
* The below example clearly demonstrates how semi-colon works in JavaScript.

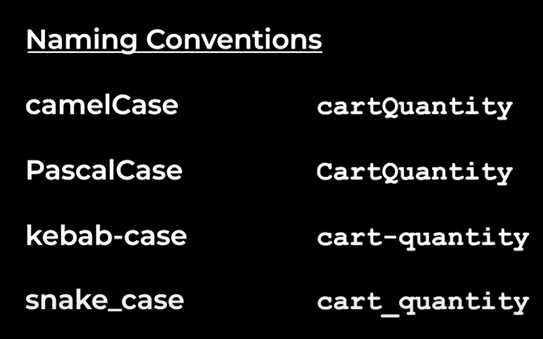


* We will be encountering the error as shown below, when we remove semi-colon after the first console.log() statement.



* JavaScript has a special feature called **Semi-Colon Insertion,** that means it will try to insert the last semi-colon in a line of code automatically.
* The below example demonstrates this special feature. In the below example (line:19), despite the missing semi-colon at the end of the instruction, yet we got the correct output. That is because, the JavaScript inserts the semi-colon at the end by itself. But, it is preferred to keep the semi-colon wherever it is required instead of depending on the semi-colon insertion. As the semi-colon insertion might be missed.
* We can re-assign a value to the variable in JavaScript. We can re-assign it as many times as we want.
* Syntax rules for re-assigning the value in a variable:

- Don’t use “let” keyword while re-assigning. That’s because ‘let’ creates a new variable.

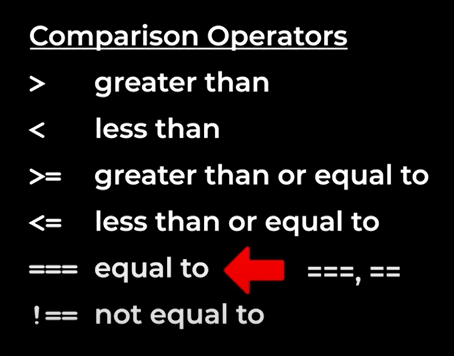


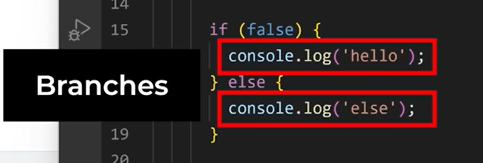
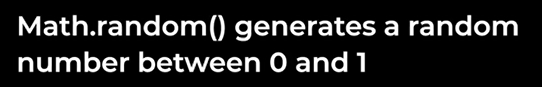
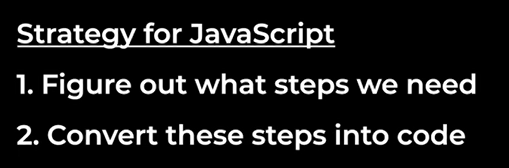
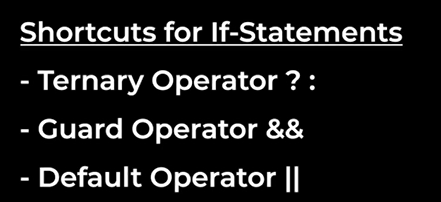
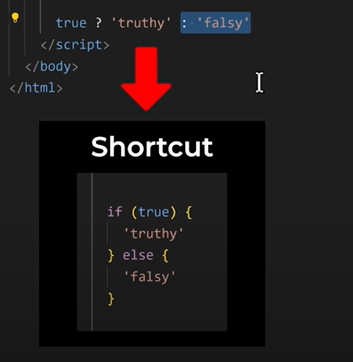
-Kebab Case doesn’t work in JavaScript as the symbol ‘- ‘is already reserved for the minus operation.

* There are 3 ways to create a variable in JavaScript:
  + Using the ***‘let’*** keyword.
  + Using the word **‘const’**. This keyword creates a variable similar to the keyword ***let***, except that the value cannot be re-assigned later.
  + Using the keyword ***‘var’***. We don’t use this keyword in new JavaScript code. Due to the issues faced because of using this.

## 6. Booleans:

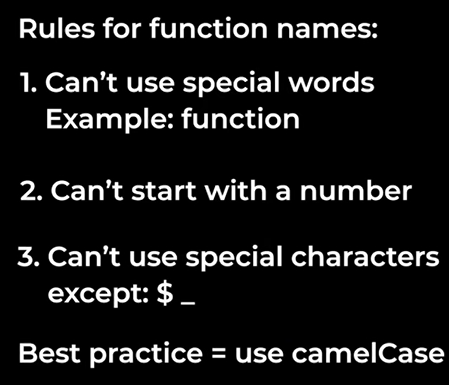
* Booleans are another type of values/ data types in JavaScript which represents whether something is true or false.
* There are only two types of Boolean values: True and False.

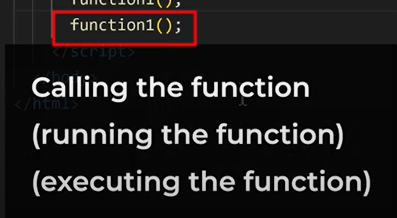


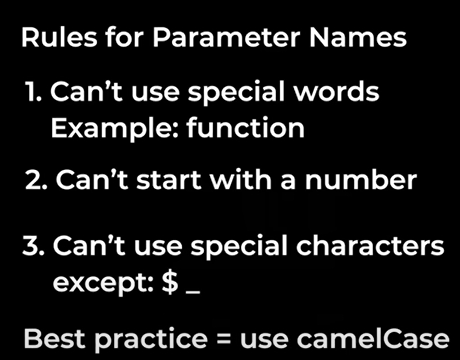
* **== (Equality Operator)**
* Compares values **after type conversion (type coercion)**.
* If the values are of different types, JavaScript converts them to a common type before comparison.
* **=== (Strict Equality Operator)**
* Compares **both value and type**.
* No type conversion occurs—values must be of the same type to be considered equal.
* If Statement:
* It checks whether the specified condition is true or not and executes the block of code inside it, only when the specified condition is true.
* **Syntax:**  
  if (condition){  
  block of code;  
  }
* We use else statement which is used to run the block of code inside it, when the if condition fails.
* Syntax:  
  if (condition){  
  block of code  
  }  
  else{  
  }
* 
* Else if statements are used to specify more than one condition.  
  
* 
* 
* **Ternary operator** is similar to if-else statement.
* Eg: true ? ‘truthy’ : ‘falsy’  
  the first value (i.e, true) is a condition.  
  For, the above example, if the condition is true/truthy, the result will be the value after the question mark. If it is false/falsy the result will be the value after the colon.
* In simple way, we can think of the question mark as the if-branch and colon as the else-branch.  
  
* The advantage of the ternary operator is that we can store the result in a variable.
* **Guard Operator  
  **

# 7. Functions

* A function lets us to reuse the code.
* We use the keyword *‘function’* to create a new function.
* Syntax: *function* function\_name (){  
  Function body}
* This code only creates a function, it doesn’t run the code inside the function. To run the code inside the function, we need to call the function.

****

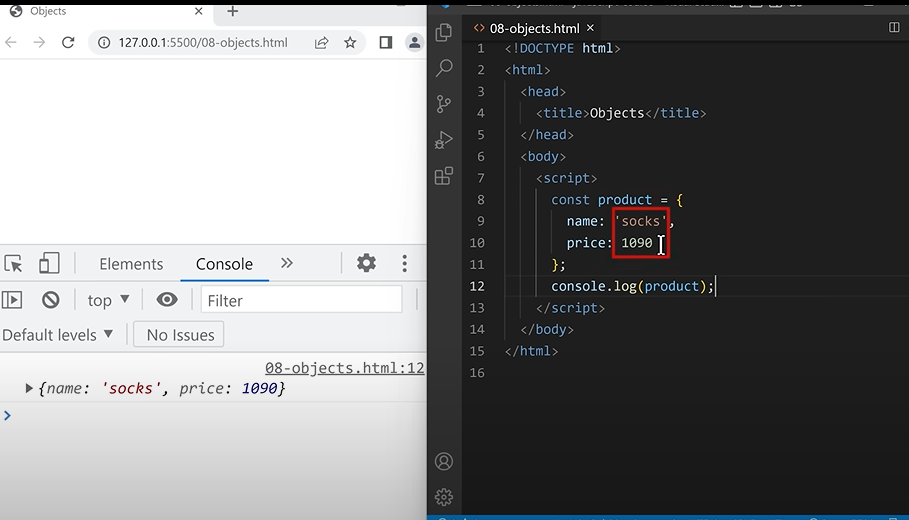
****

* A ***return*** statement lets us get a value out of a function.
* Instead of just a value, we can also return a calculation, a variable, a function and anything that results in a value.
* When we write a return statement, it is known as returning a value from a function, and the value which is returned is known as the ***return value.***
* If a function doesn’t have a return value or returns nothing, then an undefined will be returned.
* When we use a return statement, it ends the function immediately.
* A ***parameter*** puts a value into a function.
* Parameters work same as variables.
* 
* The value passed into the function at the time of function call is called an ***argument.***
* Eg: *function* calculateTax(cost);  
   calculateTax(2000);  
  Here, the ‘cost’ is a parameter. And the value 2000 is an argument.
* A function can have more than one parameter. We can have any number of parameters in a function, we just have to separate them with a comma.
* Eg: *function* calculateTax(cost, taxPercent);  
   calculateTax(2000, 0.2);
* We can also set a default value for the parameter. To set a default value for the parameter, we can just assign a value for it at the time of defining the function.
* Eg: *function* calculateTax(cost, taxPercent = 0.2);

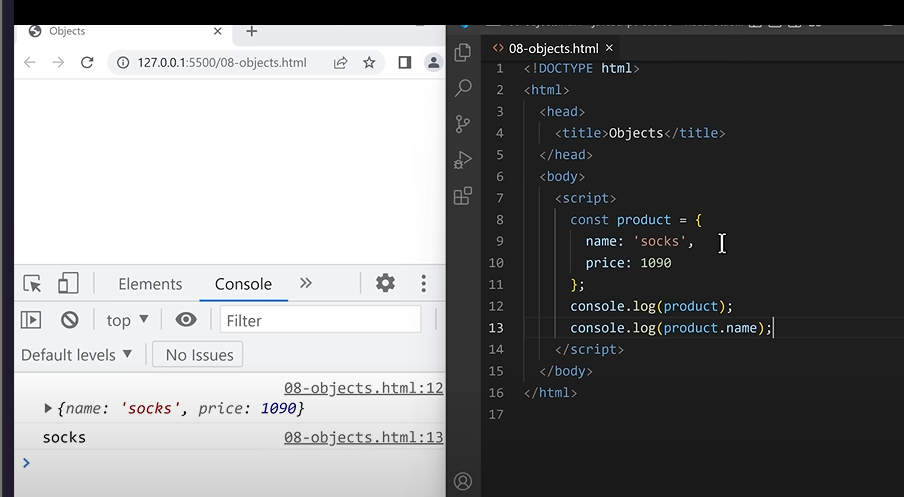
# 8. Objects

* An object groups multiple values together.

Eg:

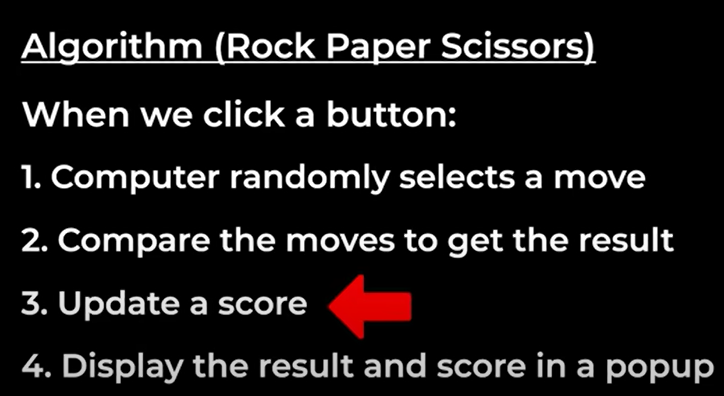


In the above example, we are grouping name and price properties into the product object. And when we call the object, we are able to access the properties inside it.

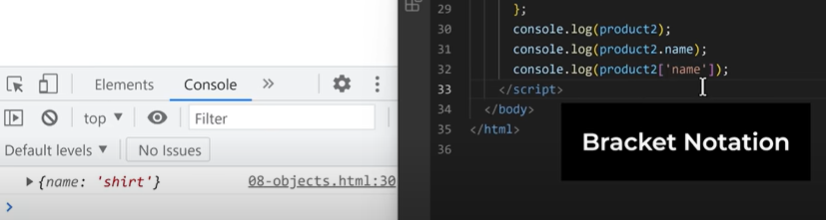
* Here, name and price are called **properties**. And using the property, we can access specific values in the object.
* Instead of accessing all the properties every time, you can also access a single property according to your necessity at that point of time. To access in this way, you’ll use the following syntax:  
   objectname.propertyname;  
  
* In the above example, we are accessing the value inside the property – ‘name’ of the object named – ‘product’, which is ‘socks’.
* We can also use the property to change the value inside the object. It can be done as shown below:  
  objectname.propertyname = newvalue;
* Eg: In the above example, The value inside the property ‘name’ is ‘socks’, to change it to a new value, we write the following instruction where the value ‘socks’ is changed to ‘cotton socks’.  
  product.name = ’cotton socks’;
* This is how an object works. We use it to group multiple values together and then we can access or modify the values using properties on the left.

Syntax Rules for Objects:

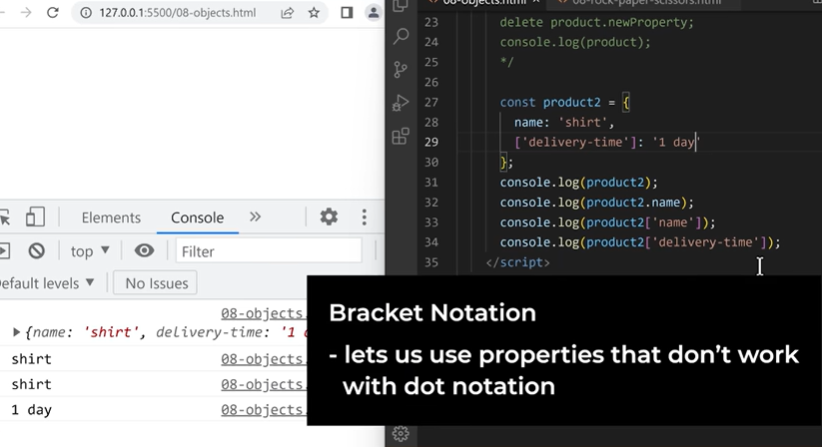
* + To create an object, we start with an open curly bracket and end with a closed curly bracket.
  + Inside an object, we can put multiple values, each of the value corresponding to a property which is on the left to the property. This is how we access the value inside the object.
  + We separate the value and the property with a colon and this is called a Property-Value pair. We can have any number of **Property-Value pairs** we want inside an object and separate them with a comma.
  + To access a value, we type the object’s name and dot and then the value we want to access like ‘name’ in our previous example. This will give us the value associated with the property. This syntax is called **Dot Notation.**
  + If we access a value that doesn’t exist, it will give us undefined.
  + To change a value in an object, we can use Dot Notation again, as shown in the above example.
  + We can also add a value to an object, we can type a property that doesn’t exist. For example, In the previous example let us add a new property named ‘NetQuantity’ as – product.NetQuantity= 3; (objectname.newpropertyname=value;).
  + We can also delete a property from an object by typing **delete** and then the objectname.propertyname; ---🡪 This code will delete the specified property.
  + Remember that object is just another type of value in JavaScript, which means it can be stored inside a variable.



* + We also have another notation named as **Bracket Notation**, which is shown below.

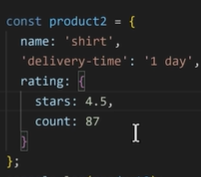
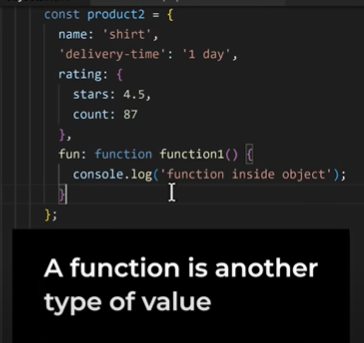
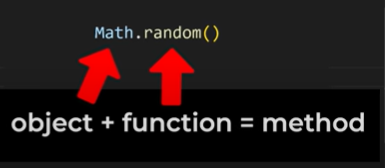
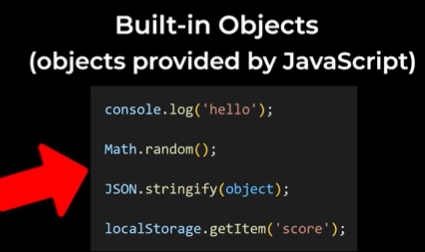
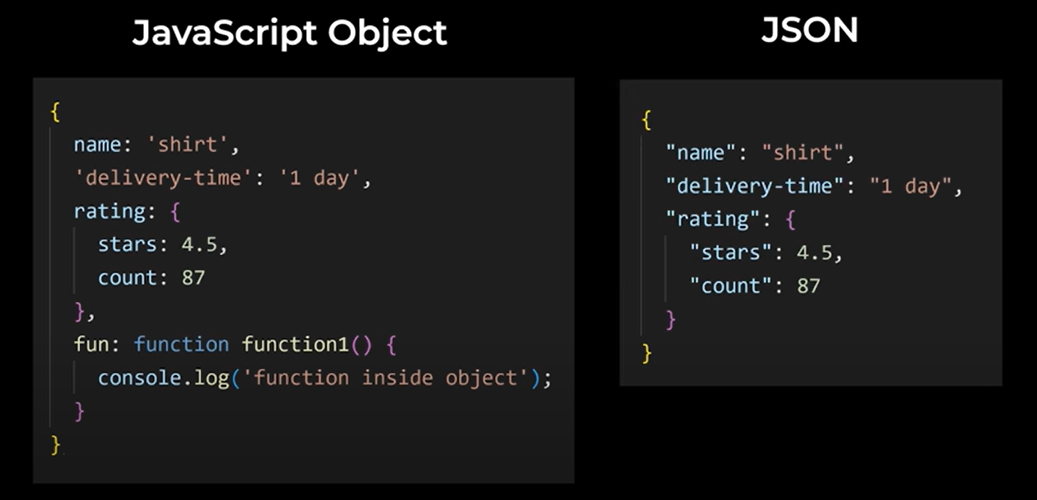


* + It does the same thing as dot notation, it It gets te value with the specified property.  
    Bracket Notation uses:
  1. Bracket Notation lets us use properties that don’t work with dot notation.
     + For example, when you write the following line of code, it gives us an error that ‘time’ is not defined.  
       *console.log(product.delivery-time);*Because, it treats ‘-‘(hypen) as a minus or subtraction operation, So to access a variable with that kind of special characters in it, we use Bracket notation.  
       *console.log(product[‘delivery-time’]);*
  2. Bracket Notation lets us add values to the object when we create it.



* 1. Between the brackets, we don’t just have to use a string, we can also use a variable, a calculation or anything that results in a value.
* *Which notation should we use?*
  + Use dot notation by default.
  + For properties that don’t work with dot notation, use bracket notation.



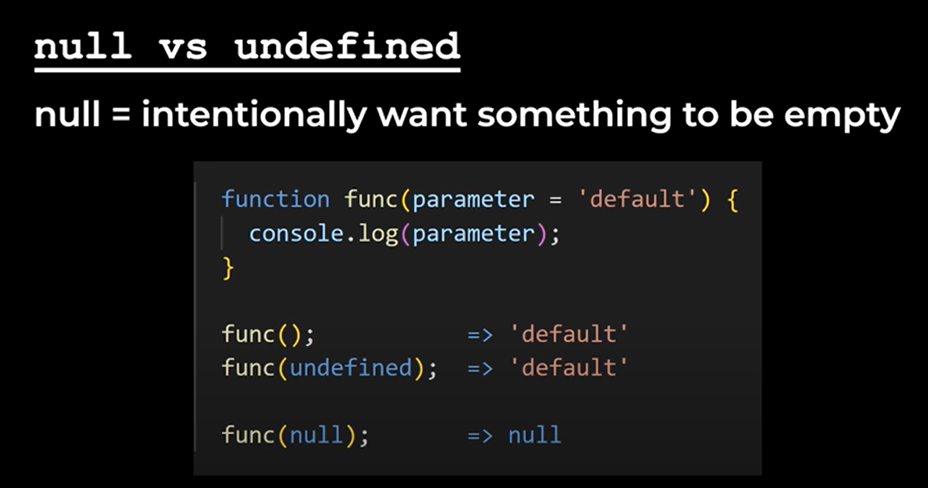
* Inside an object, we can save any type of value. i.e, It can be a string, num, object etc.
* 
* To access the values inside the inner object, we just use the dot notation or Bracket notation multiple times.  
  Eg: In the above example, if we want to access the value inside the inner object, we write an instruction as below:  
   *console.log(product2.rating.count); ---------- > 87*
* Another type of value we can put in an object is – *A function.*
* It is done as shown below:  
  **
* To access this function, we can use the dot or bracket notation as usual.
* For the above example, we can use the below line of code to call the function.  
  *product2.fun();*
* When we save a function inside an object, it is called a **Method.**
* Everything in JavaScript is an object, console is an object, Math is an object, log() is a function, So console.log() is a method.
* 
* **Built-in Objects:  
  -** These are the methods that are built in the language. Eg, console, Math, JSON, localStorage.
* ****
  1. **JSON Built-in Object:  
     JSON:** - Helps us work with JSON.  
      - JSON = JavaScript Object Notation.  
      - JSON is basically a syntax, similar to a JavaScript object but has less features.   
     ****
     + From the above example, we can observe the following differences:  
       - JSON does not support Single quotes.  
       - JSON does not support functions.

- JSON is similar to JS object but with less features. So why do we use JSON..?  
 - JavaScript code makes sense only in JS. But, where as JSON can be understood by almost every language. So, JSON syntax is more universal.   
 - For this reason, we use JSON when we send data between two computers that might use different programming languages.  
**JS ---- > JSON**  
**-** A JSON object helps us convert a JS object to JSON.

- To change our product2 object in the previous example of JS to a JSON object, we write the below line of instruction.  
 *console.log(JSON.stringify(product2));*

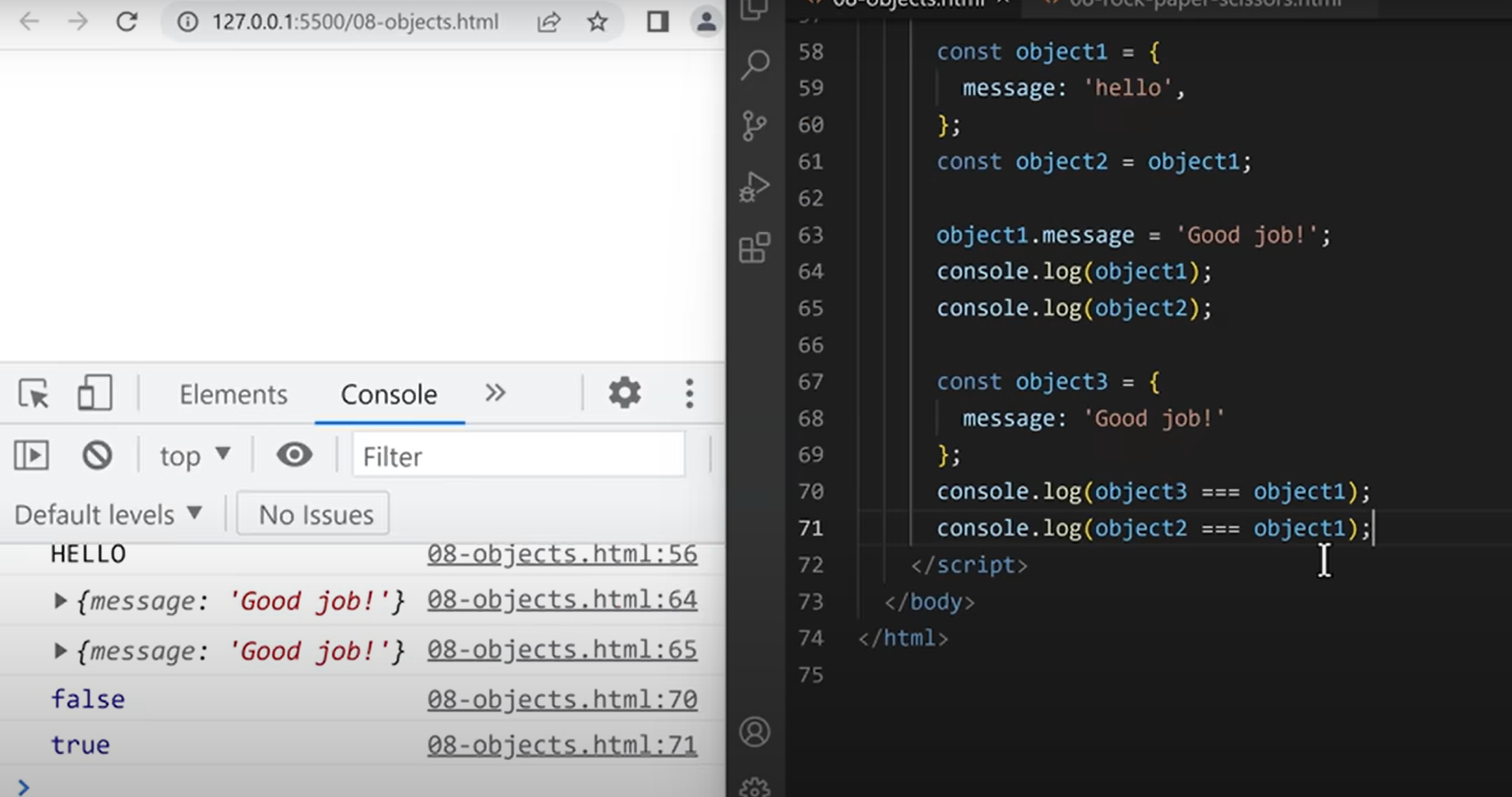
- stringify function returns a string.  
**JSON ----- > JS Object**- We use *JSON.parse()***.**- JSON.parse(string\_you\_want\_to\_convert)

- To convert the previous example into JSON string:  
 const jsonString = JSON.stringify(product2);  
- *console.log(JSON.parse(jsonString));*

* 1. **localStorage**- It is used to save values more permanently.  
     - So far, we’ve been saving our values in variables, variables are temporary. If we refresh the page or close the page, all the variables are deleted.   
     - To store a value in localStorage, It has a method called localStorage.setItem().  
     - localStorage only supports strings.  
      *localStorage.setItem(‘message’ , ’hello’); ---------- > inside the string- message, the value ‘hello’.*  
      *console.log(localStorage.getItem(‘message’);* ----- > *hello*

**Auto-Boxing:**

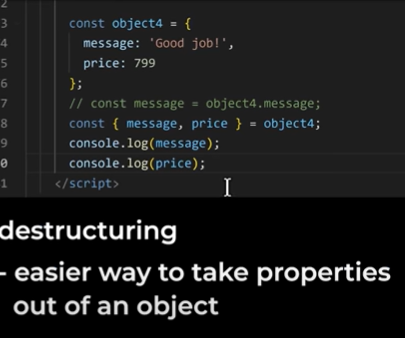
* For example, consider *console.log(‘hello’,toUpperCase());*, JavaScript automatically wraps the string ‘hello’ in a special object first and this object has the method length, and the property -> toUpperCase. This is done automatically. This feature is called **Auto-Boxing.**
* JavaScript automatically wraps a string in a special object like a box. This doesn’t only work with string, it works the same for any type of value. But, Auto-Boxing doesn’t work with null or undefined.
* Objects are references.
* When you create an object name *obj1*, the value inside the object is stored somewhere in the computer’s memory, and a reference to this memory is stored in the variable named obj1.
* In simple words, it’s like a shortcut in your desktop.
* Copy by reference.
* We can’t compare objects directly. Because, when we compare an object, it compares the references, not the values inside.



* Shortcuts of Objects:

1. Destructuring:   
    Eg: const message = object4.message;

The above line of code is same is a the below:  
const {message} = object4;

* + - In the above example destructuring shortcut takes the property *‘message’* out of the object and assigned it to a *‘message’* variable.
    - We can use this shortcut for multiple properties.  
      

1. Shorthand Properties
2. Shorthand Methods

# 9. Document Object Model (DOM)

* DOM is another built-in object, called the Document Object.
* Eg: document.body.innerHTML=’Hello’;

document.title=’Good job!’;

* The document object represents/ models the webpage. That is why we call this object, Document Object Model (DOM).
* DOM combines JavaScript and HTML together. It gives JavaScript full control of the page.
* **Syntax Rules for the DOM:**
  + This document object works like any other object, it has properties like body, title etc. But it’s speciality is that, it is connected to web-page directly. Which means if we change certain properties in this object, it will also change the web-page.
  + In addition to properties, DOM also has methods like querySelector etc.
* **Properties and Methods:** 
  + document.title (property) – To update or access the title of a web-page.

Eg: *console.log(document.title); -> It prints the title of the web-page on to the console.*

*document.title=’changed’; -> This changes the title of the web-page to ‘changed’.*

* + document.body (property) – Lets us get the body element and ut it inside our JavaScript code.
    - document.body.innerHTML – Controls all the HTML inside the body.
    - When an HTML element is inside JavaScript, it is converted to a JavaScript object.

Eg: *console.log(document.body.innerHTML); -> Prints what is inside the HTML body.*

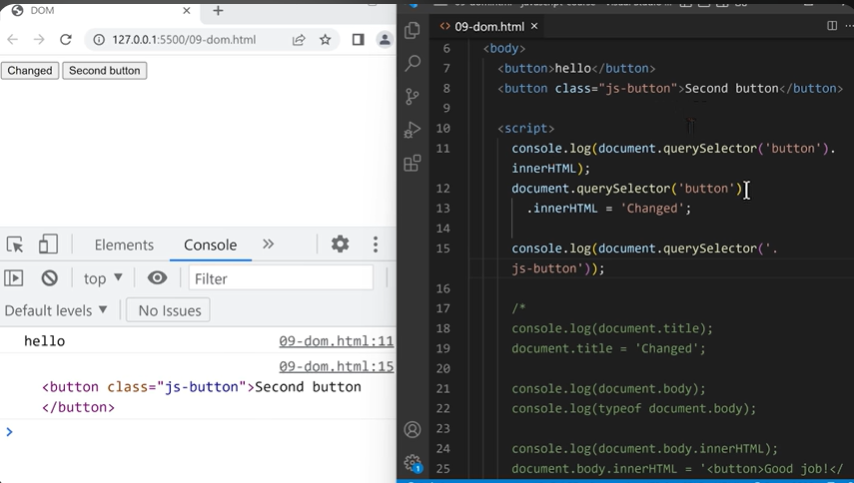
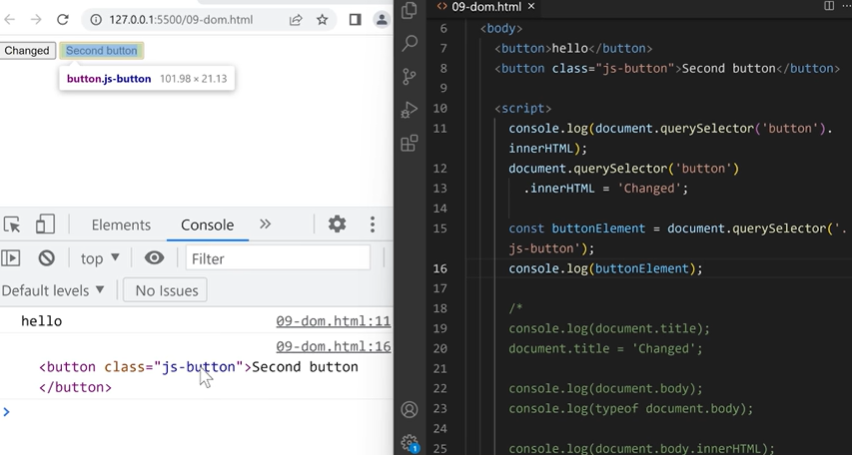
*document.body.innerHTML = ‘Changed’ -> Replaces the complete code content inside HTML body to the string ‘Changed’. Not only a string, but also we can replace it with a HTML code.  
 document.body.innerHTML = ‘<button>Changed</button>’; -> This code will replace all the HTML code inside the body with the button specified.*

* + document.querySelector() – Lets us get any element from the page and put it inside the JavaScript.

Eg: document.querySelector(‘button’); -> This line method got the first button element inside the HTML code and put it inside the JavaScript code. The string which we specify inside the method will be bought into JavaScript i.e, For example, If we consider the string inside that method is ‘body’, then the first body element will be selected – In short, it will then do the same thing as document.body.

* Every HTML element has a property **‘*.innerHTML’*** *Which lets us control the HTML inside the element.*Eg: *<body>   
   <button>hello</button>  
   <script>*  
   *console.log(document.querySelector(‘button’).innerHTML);*

</script>  
 </body>  
The above code will give the output as ‘hello’. Because, the html inside the first button element is ‘hello’ and it is being accessed using .innerHTML.

* Consider we are having two buttons inside the HTML code and we want to select the second button element, But the querySelector method only selects the first button element. So, to solve this, we are going to add a class attribute to that button element. When we are using class selector, we specify the class name inside the querySelector method with a “.”.  
  
* It is a common practice to start the class name with “js-classname”, to indicate that it is used in the JavaScript.
* HTML elements are JavaScript objects. That means, this class attribute is also a js object. All the js objects are nothing but values, i.e, it can also be stored inside a variable. When we save that, it will show us that, the value is stored inside that particular element as shown below.  
  
* Using the DOM, we can get HTML elements from the page, put it inside the JavaScript, and then save it in a variable.
* 
* When you run the above code, if you give a space or new-line with the button label- ‘Subscribe’, the code doesn’t work. It doesn’t change anything.
* So, we are using innerText instead of innerHTML. It gets us the text without the actual spaces.
* Whenever we get a value from the DOM, the value will be a string. To convert it into an integer, we use Number().
* There is a HTML attribute which runs JavaScript when we press a key down on our keyboard, that attribute is onkeydown=’…’ .
* Clicks and keydowns are known as ***events*.**

Onclick and onkeydown are known as ***event listeners*.** They check for events and runs JavaScript when those events happen.



* The *document* object represents a webpage, the *window* object represents a browser. So, everything inside the browser, is inside the window i.e, a web-page is inside the browser which means that the document object is inside the window. Similarly, console and alerts are also a part of the browser.

# 10. HTML, CSS, and JavaScript Together

* classList
  + classList is an object. Similar to any other object, classList also have methods for it. One such method is *.add()*.
  + ***.add()*** – adds a class to an element.

**Eg:** *buttonElement.classList.add(‘is-subscribed’);*

The above line adds a class called – ‘is-subscribed’ to the button, whenever the button is pressed.

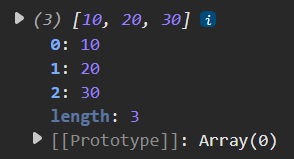
* + ***.remove()*** – removes a class to an element.

# 11. Arrays and Loops

- An array is another type of value in JavaScript which represents a list of values.

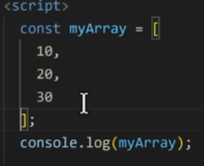
**Eg:** *let arr1=[10,20,30];*

*console.log(arr1);*



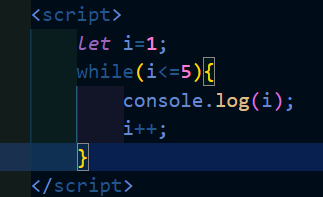
* To get a specific value in an array, we can use square brackets -> ‘[]’.

Eg: arr1[0] -> Access the value at the index 0 in the array named arr1 i.e, the first value in the array.

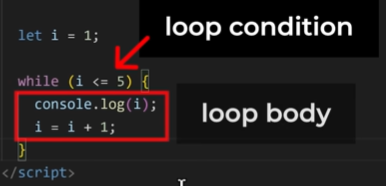
* We can also change the value in an array using this method i.e, square braces -> ‘[]’.
* 
* In the above example, the value in the first index is changed to ‘99’.
* **Syntax rules for Arrays:**
  + To create an array, we start with an open square bracket and end with a closed square bracket and we separate each value with a comma.
  + We can also write an array in multiple lines.  
    
  + We can use indexes to get a value inside an array, which represents the position in the array that we want to access. Index of an array starts from zero.
  + Inside an array, we can put any type of value.  
    Eg: arr1= [1, ’hello’, true, {name:’socks’}, [1,2]];
  + An array is also a value.
  + Array is a special type of object.
  + As an array is also a type of object, it also has it’s own properties and methods.
  + If we want to check something is specifically an array or not, we use *.isArray()* method. This method returns a Boolean value.
  + *.length()* – Gives the length of the array.
  + *.push() –* Adds a value to the end of the array.
  + *.splice() –* Removes a value from an array. Splice method takes two parameters, which are – The index which we want to remove, and the other is the number of values we want to remove.   
    Eg: *arr1.splice(0,2); ->* It removes two values starting from the index ‘0’.

## Loops:

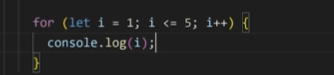
* + Loops let us run some code over and over.
  + A loop runs over and over till the condition fails (or) whenever the code is true, the loop will keep going.
  + There are different types of loops in JavaScript:
    - While loop
    - For loop
  + **While loop:**



* + In the above example we are using a ***while-loop*** *(while the mentioned condition is true, the loop will run through out that time)***,** which is a type of loop that runs as long as the condition specified inside it is false. Here, it prints the values from 1 to 5 and when the value of ‘i’ is updated to 6, inside the loop and runs again while checking the condition, it returns false – loop ends here.
  + **Syntax rules for WHILE loop:**
    - *while(loop condition) {loop body}*

****

* + - The variable *‘i’* here, is called the loop variable.
    - Every time we loop, we are going to update the loop variable. This step will ensure that, the loop ends at some point, else the loop will go forever (infinite loop).
    - Loop creates a scope; that is, any variable we create between the curly brackets will only exist inside them.
  + **For loop:**
    - The keyword we use to define a for-loop is ‘*for*’.
    - After starting with the keyword ‘*for*’, we write three pieces of code inside the round brackets ‘()’ which are separated by a semi-colon ‘;’ each, those are –
      1. Loop variable
      2. Loop condition
      3. Updation
    - Eg: *for (let i=1; i<=5; i++) {loop body}*



* + - The above for-loop also does the same thing which was done by the previous while-loop. i.e, printing the first first 5 natural numbers.  
        
      