**JAVASCRIPT**

JavaScript is an interpreted programming language.

slow and they had to execute all the instructions line by line.

Use to add animations and user interaction to our websites.

**alert(“hello”); -** pop-up the message inside the double quotes.

**alert** – keyword/function **hello** – message **;** - end.

**DATATYPES**

String, Number, Boolean

**typeof(“hello”); -** gives the datatype which is inside the parenthesis **-** string

**typeof(23); -** number

**typeof(true);** - Boolean.

**VARIABLES -**  containers

**var myName = “gukan”; -** here gukan is stored inside the myName.

**var –** keyword for variable **myName –** variable name **gukan –** value.

**var yourName = prompt(“what is your name?”); -**  here the prompt get the name and store it inside the yourName.

**NAMING CONVENTIONS**

1. There is no spaces between variable name – **my name( invalid).**
2. Variable name should not start with number or symbol – **1my, $my( invalid).**
3. Reserved keywords should not be used – **var, alert, prompt, etc(invalid).**
4. Should follow camel casing – **myName, yourName.**
5. Variable name can contain **letters, numbers, $, \_.**

**CONCATENATION**

“hello” **+** “ ” **+** “world” – hello world.

**STRING LENGTH**

**var written = prompt ("write a tweet");**

**alert ("you have written " + written.length + " words," + " you have " + 180-written.length + " characters left.");**

written.length – gives the length of the string.

**STRING SLICE**

var name = “gukan”;

name.**slice**(0,1); - g

name.**slice**(0,3); - guk

name**.slice**(2,3); - k

**alert( prompt("write a tweet").slice(0,140)); -** first gets a tweet from a user and then slice the first 140 characters and make a alert to user.

**EXAMPLE**

**var name = prompt("what is your name?");**

**var firstLetter = name.slice(0,1).toUpperCase();**

**var remLetter = name.slice(1,name.length).toLowerCase();**

**alert("Hello, "+ firstLetter + remLetter);**

**BASIC ARITHEMETIC**

**var dogAge = prompt("what is your dog's age?");**

**var humanAge = ((dogAge-2)\*4)+21; -** BODMAS rule.

**alert("dog's human age is " + humanAge);**

**INCREMENT AND DECREMENT OPERATOR**

x = x+1 => **x++**

x = x-1 => **x--**

x = x(+,-,\*,/) any number => **x(+,-,\*,/)=that number**

x = x\*2 => x\*=2

**FUNCTIONS**

**CREATING A FUNCTION:**

**Function function-name () {** code to be executed **}**

**CALLING A FUNCTION:**

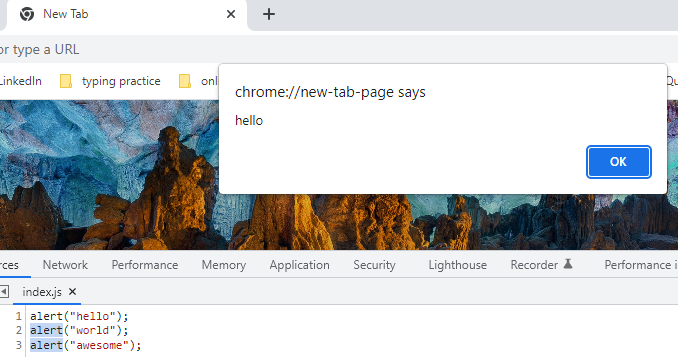
**Function-name ();**

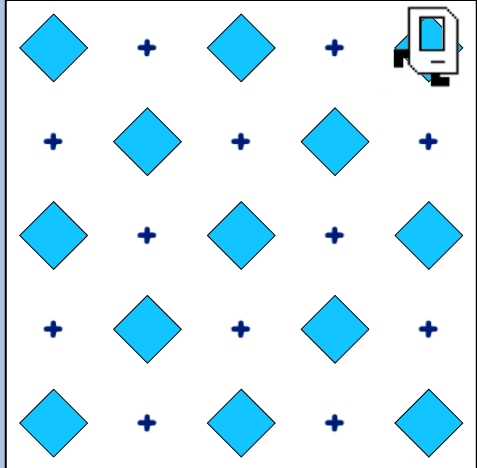
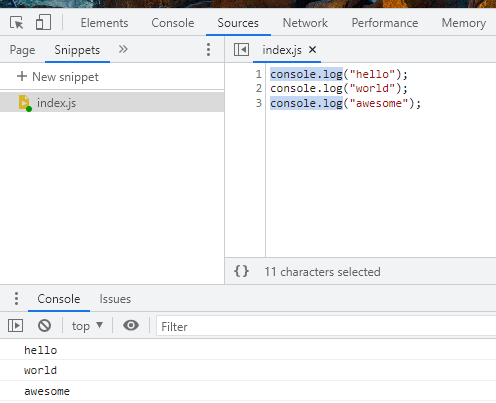
**ALERT AND CONSOLE LOG**

Important difference between the console log and the alert is that the alert is something that the user can see. So if you had **an alert** on your web site, then **any visitor will be able to see the alert**, but the **console logs** are only for the developer.

They **only show up in the console**, which is not intended for the user.

It's meant for the developer to debug your code, to find out if there were any problems, and to essentially print out parts of the code into the console.



****

function main(){

movePutBeeper();

movePutBeeper();

moveLeft();

movePutBeeper();

moveRight();

movePutBeeper();

movePutBeeper();

moveLeft();

movePutBeeper();

moveRight();

movePutBeeper();

movePutBeeper();

putBeeper();

}

function movePutBeeper(){

putBeeper();

move();

move();

}

function moveLeft(){

putBeeper();

turnLeft();

move();

turnLeft();

move();

}

function moveRight(){

putBeeper();

move();

turnRight();

move();

turnRight();

}

**FUNCTIONS WITH PARAMETER AND ARGUMENTS**

function getMilk(**money**) {

console.log("buy " + **Math.floor(money/1.5)** + " bottles of milk"); - **here Math.floor is to round-off**

}

getMilk(10);

function lifeInWeeks(**age**) {

var x =((90\*365)-(**age**\*365));

var y =((90\*52)-(**age**\*52));

var z =((90\*12)-(**age**\*12));

console.log("You have "+ x + " days, " + y + " weeks, and " + z + " months left." );

}

lifeInWeeks(25);

**FUNCTIONS WITH OUTPUT AND RETURN VALUES**

function getMilk(**money**) {

console.log("buy " + **Math.floor(money/1.5)** + " bottles of milk");

**return** money % 1.5;

}

Var change = getMilk(10);

Console.log(change);

**BMI CHALLENGE**

function **bmiCalculator** (weight, height) {

var bmi1 = **Math.round**(weight/**Math.pow**(height,2));

return bmi1;

}

Var bmi = bmiCalculator(65,1.8);

**RANDOM GENERATION**

**Math.random() -**  static method returns a floating-point, pseudo-random number that's greater than or equal to 0 and less than 1 (0 – 0.9999999999999999(16 decimal places)).

**Math.floor() -** The **Math.floor()** static method always rounds down and returns the largest integer less than or equal to a given number.

console.log(Math.floor(5.95));

// expected output: 5

console.log(Math.floor(5.05));

// expected output: 5

console.log(Math.floor(5));

// expected output: 5

console.log(Math.floor(-5.05));

// expected output: -6

**LOVE CALCULATOR CHALLENGE**

var firstName = prompt("first person Name");

var secondName = prompt("second person Name");

var n= **Math.random();**

n = n\*100;

n = Math.floor(n)+1;

alert("love percentage is :" + n + " %");

**CONTROL STATEMENTS**

**IF - ELSE STATEMENTS**

If (n === 3) {code to executed}

Else {code to be executed }

**COMPARATORS**

**=== -** equal to.

!== - not equal to.

>= - greater than or equal to. > - greater than.

<= - lesser than or equal to. < - lesser than.

Var a = 1;

Var b =”1”;

If (a ===/== b) { console.log(“yes”);} –> **here == only checks the value whereas === checks the value and also its datatypes are matching.**

Else { console.log(“no”);}

**COMBINING COMPARATORS**

&& - **AND || - OR ! – NOT**

**if (n > 70)** {

alert("love percentage is :" + n + " %" + " love like flesh and skin");

}

**if (n > 30 && n <= 70)** {

alert("love percentage is :" + n + " %");

}

**if (n <=30)** {

alert("love percentage is :" + n + " %" + "love like oil and water");

}

**BMI CALCULATOR ADVANCED**

function bmiCalculator (weight, height) {

var bmi = weight/(Math.pow(height,2));

var interpretation;

if (bmi<18.5){

interpretation = "Your BMI is " + bmi +", so you are underweight."

}

if (bmi>=18.5 && bmi<=24.9){

interpretation = "Your BMI is " + bmi + ", so you have a normal weight."

}

if (bmi>24.9){

interpretation = "Your BMI is " + bmi +", so you are overweight."

}

return interpretation;

}

**LEAP YEAR CHALLENGE**

function isLeap(year) {

var statement;

if (year%4 ===0) {

if (year%100 === 0) {

if (year%400 === 0){

statement = "Leap year.";

}

else{

statement = "Not leap year.";

}

}

else {

statement = "Leap year.";

}

}

else {

statement = "Not leap year.";

}

return statement;

}

**ARRAYS**

It’s a collection of items that are related, and they can be stored together into the same container or the same variable.

**var name = [“gukan”, “vasu”, “devan”]; - initialization.**

var guestList = ["Angela", "Jack", "Pam", "James", "Lara", "Jason"];

var name = prompt("what is your name?");

if (guestList**.includes**(name)){

alert("Welcome!");

}

else {

alert("Sorry, next time please");

}

Here **guestList.includes(name) –** checks whether the name included in the guestlist array or not.

**FIZZBUZZ CHALLENGE**

var output = [];

var count = 1;

function fizzBuzz(){

if (count%3 === 0 && count%5 ===0){

output**.push**("FizzBuzz");  **-> here output.push() pushes the value into the array.**

} **similarly, output.pop() removes the last element of array.**

else if (count%3 === 0){

output.push("fizz");

}

else if (count%5 === 0){

output.push("Buzz");

}

else{

output.push(count);

}

count++;

console.log(output);

}

**Who buying the lunch challenge**

Var names = [“gukan”, “vasu”, “devan”, “mala”, “devi”];

function whosPaying(names) {

var n = Math.random();

n = Math.floor(n \* names.length );

return names[n] + " is going to buy lunch today!";

}

**CONTROL STATEMENTS**

**WHILE LOOPS**

**While ( something is true ) {**

**// do something**

**}**

**99 BOTTLES CHALLENGE**

var count = 99;

function beer() {

while (count >= 1) {

var less = count-1;

console.log(count + " bottles of beer on the wall, " + count + " bottles of beer. Take 1 down, pass it around, " + less + " bottles of beer on the wall.");

count--;

}

console.log("No more bottles of beer on the wall, no more bottles of beer.");

}

**FOR LOOP**

**For (i=1; i<2; i++) {code to be executed }**

var output = [];

function fizzBuzz(){

**for (count = 1; count<=100; count++){**

if (count%3 === 0 && count%5 ===0){

output.push("FizzBuzz");

}

else if (count%3 === 0){

output.push("fizz");

}

else if (count%5 === 0){

output.push("Buzz");

}

else{

output.push(count);

}

}

console.log(output);

}

**FIBONACCI SERIES**

function fibonacciGenerator (n) {

var output = [];

var a,b,c,x,y,z;

if (n===1) {

output.push(0);

return output;

}

else if (n===2) {

output.push(0,1);

return output;

}

else if (n>2) {

a = 0;

b = 1;

output.push(a,b);

for (var i=2; i<n; i++) {

z = a+b;

output.push(z);

c = a;

a = b;

b = c+a;

}

return output;

}

}

**OR**

function fibonacciGenerator (n) {

var output = [];

if (n===1) {

output.push(0);

}

else if (n===2) {

output.push(0,1);

}

else if (n>2) {

output.push(0,1);

for (var i=2; i<n; i++) {

output.push(output[output.length-2]+output[output.length-1]);

}

}

return output;

}

**ADDING JAVASCRIPT TO WEBSITES**

Similar to css styles, we can add javascript in three ways

**INLINE JAVASCRIPT**

<html lang="en" dir="ltr">

<head>

<meta charset="utf-8">

<title>My Website</title>

<link rel="stylesheet" href="css/styles.css">

</head>

**<body onload="alert('Hello');"> - here onload will execute the javascript codes inside**

<h1>Hello</h1> **the quotes when body loads up.**

</body>

</html>

**INTERNAL JAVASCRIPT**

Here we use **script tag** for internal, everything in between the script tag are consider as javascript codes.

<!DOCTYPE html>

<html lang="en" dir="ltr">

<head>

<meta charset="utf-8">

<title>My Website</title>

<link rel="stylesheet" href="css/styles.css">

</head>

<body>

<h1>Hello</h1>

**<script type="text/javascript">**

**alert("hello");**

**</script>**

</body>

</html>

**EXTERNAL JAVASCRIPT**

Here we use **script tag** that links source file**(index.js).**

<!DOCTYPE html>

<html lang="en" dir="ltr"> **index.js**

<head> alert("hello");

<meta charset="utf-8">

<title>My Website</title>

<link rel="stylesheet" href="css/styles.css">

</head>

<body>

<h1>Hello</h1>

**<script src="index.js" charset="utf-8"></script>**

</body>

</html>

Unlike css, javascript has different usage, because inside our script we tried to change an HTML element that does not yet exist then it will fail. So **always try to add external script at the bottom of body element to make sure the content loads first and then script.**

**DOCUMENT OBJECT MODEL (DOM)**

catalogues the web page into individual objects **that we can select and manipulate in live state.**

when you load up the web page, the browser turns each of these elements and their associated data into a **tree structure** with a whole bunch of objects that you can select and manipulate.

Objects inside the DOM can have

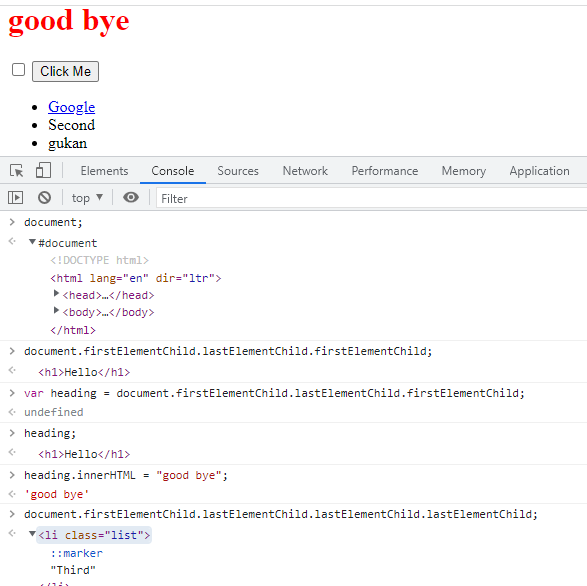
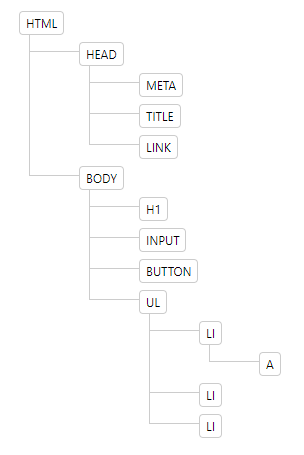
* **Properties(describe something about the object)**

Get property – **heading.colour;**

Set property – **heading.colour = “red”;**

* **Methods(something that object can do).**

**SELECTING HTML ELEMENT USING DOM**



**getElementsByTagName(“name of the tag”); -** gives the output in form of **array.**

used to select the **tag** mentioned for manipulation.

**getElementsByClassName(“name of the class”);** - gives the o/p in form of **array**.

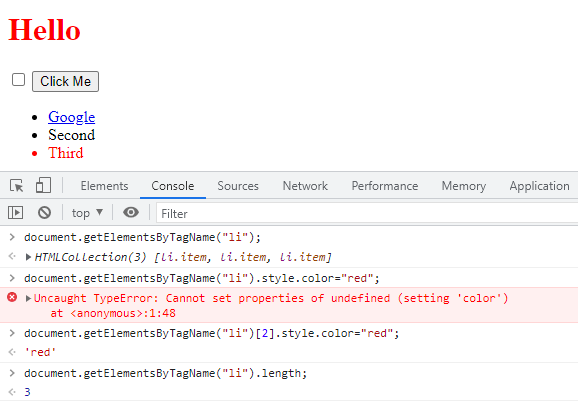
Used to select the **class** mentioned for manipulation.

**getElementById(“id name”);** - gives the **single** element.

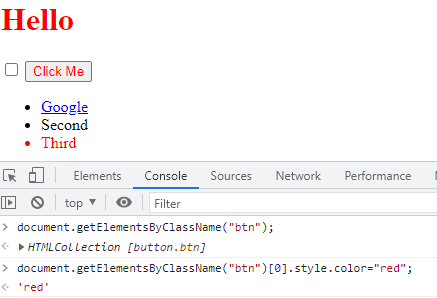
Used to select the **id** mentioned for manipulation.

**querySelector(“tag/.class/#id/combined selectors”); -**gives only single element which is first.

**querySelectorAll(“tag/.class/#id/combined selectors”); -** gives the all elements in **array.**



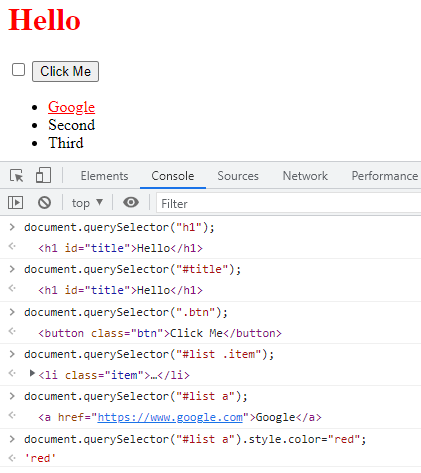
**getElementsByTagName(“tag”);**

****

**getElementsByClassName(“class”);**



**getElementById(“id”);**



**querySelector(“tag/class/id/combined selectors”);**



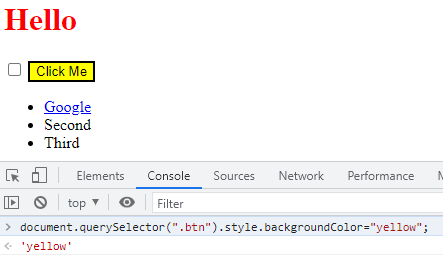
**querySelectorAll(“tag/class/id/combined selectors”);**

**MANIPULATING AND CHANGING STYLES OF HTML ELEMENTS USING JAVASCRIPT**

Unlike css, we have different usage for property like

**CSS JAVASCRIPT**

1. **font-size: 10rem; fontSize = ”10rem”;**
2. **background-color: red; backgroundColor = ”red”;**



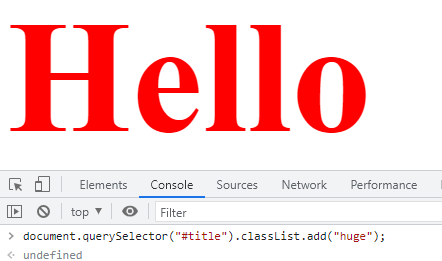
**SEPERATION OF CONCERNS**

**HTML –** only for content.

**CSS –** only for styles.

**JAVASCRIPT –** only for behaviour.

At the same time we can change the style of HTML element using javascript methods like

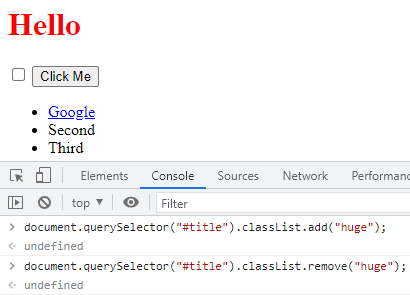


**css/styles.css**

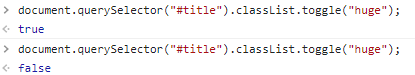
.huge {

Font-size:10 rem;

}



**.toggle(“class name”); -** this **method add the class if not added and remove the class if not removed**



**MANIPULATION OF TEXT AND ITS PROPERTY**

**<h1 id = "title"><strong>Hello</strong></h1>**

****

Here **.innerHTML** gives the entire

Text(**<strong>Hello</strong>)** b between the **h1** tags whereas

**.textcontent** gives only the text(**hello)**

****

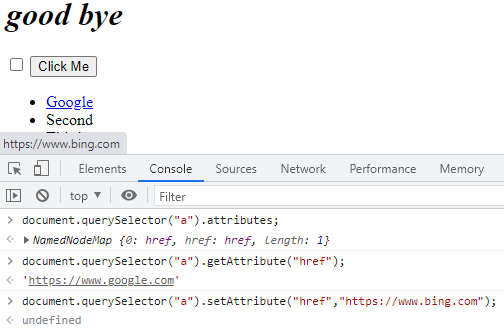
Here we can change **inner text only** using **.textcontent** whereas using **.innerHTML** we can change the **text property** also.

**MANIPULATION OF HTML ELEMENT ATTRIBUTES**

**.attributes –** gives the list of attributes which is associated with the HTML element.

**.getAttribute(“name of the attribute”); –** get the value of that specified attribute.

**.setAttribute(“attribute name”, “value”); –** set the value of the specified attribute.

****

**ADDING EVENT LISTENER TO THE BUTTON**

**EventTarget.addEventListener()**

method of the EventTarget interface sets up a function that will be called whenever the specified event is delivered to the target.

**.addEventListener(type, listener)**

**type -** A case-sensitive string representing the [event type](https://developer.mozilla.org/en-US/docs/Web/Events) to listen for.

**Listener -** The object that receives a notification (an object that implements the [Event](https://developer.mozilla.org/en-US/docs/Web/API/Event) interface) when an event of the specified type occurs. ( or simply a **javascript function**)

**document.querySelector("button").addEventListener("click",handleClick); -** here when the button got clicked then EventListener will call the function handleClick whereas the alert message shown.

**function handleClick(){**

**alert("i got clicked");**

**}**

EventTarget - **document.querySelector("button") – button.**

Event type – **click –** whenever the button clicked.

Listener – **handleClick** – javascript function – here the function call didn’t include parenthesis because **handleClick()** will call the function as soon as line gets loaded without waiting for the **click** event type.

**document.querySelector("button").addEventListener("click",function (){**

**alert("i got clicked");**

**});**

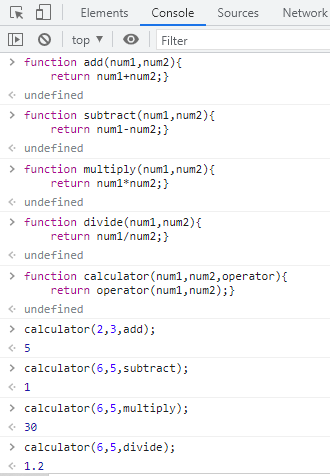
here the **listener – anonymous function.**

**HIGHER OREDER FUNCTIONS**

Functions which can take other functions as inputs are called higher order functions.

For debugging:

1. type - **debugger;** - then **hold shift + enter.**
2. **Code for debugging. –** press **enter.**



**PLAYING SOUNDS ON WEBSITES**

**HTML AUDIO ELEMENT**

var audio = **new Audio**("sounds/tom-1.mp3");

audio**.play();**

here **new Audio** method used to identify and load the sound files

**play()** method used to play the sound.

**ADDING IMAGE TO THE BUTTON**

<button class="w drum">w</button> **css/styles.css**

.w {

 **background-image: url(“images/tom1.png”);**

}

**JAVASCRIPT OBJECT WITH PROPERTIES**

Similar to **structure** in C, here we have **javascript objects – storing different datatype in single variable.**

var houseKeeper1 = {

name:"angela",

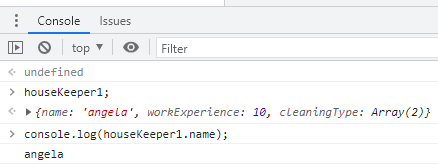
workExperience: 10,

cleaningType: ["bathroom","kitchen"]

}

Here **houseKeeper1** is javascript object whereas **name, workExperience, cleaningType** are its properties.

Alert(“hello, my name is “+”**houseKeeper1.name”**);



here the problem is, if we want to create another housekeeper object means we want to write whole bunch of properties (name, workExperience, cleaningType) again. So this can be overcome by constructor function.

**CONSTRUCTOR FUNCTION**

**function HouseKeeper** (name, experience, cleaningType) {

**this.name**=name;

**this.workExperience** = experience;

**this.cleaningType** = cleaningType

}

Here the **function name (HouseKeeper)** should be **capitalized not camel casing.**

**var houseKeeper1 = new HouseKeeper("angela",10, ["bathroom","kitchen"]);** - creating objects with constructor function.

**var houseKeeper2 = new HouseKeeper("vennila",12,["kitchen","bedroom"]);**

**SWITCH STATEMENT IN JAVASCRIPT**

Switch (condition) {

Case (expression):

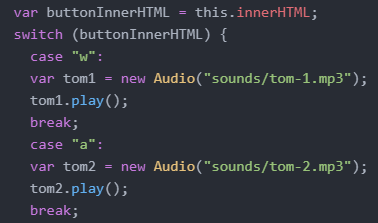
Break;

Case (expression):

Break;

………

Default:





**JAVASCRIPT OBJECTS WITH PROPERTIES AND METHODS**

function HouseKeeper (name, experience, cleaningType) {

this.name=name;

this.workExperience = experience;

this.cleaningType = cleaningType;

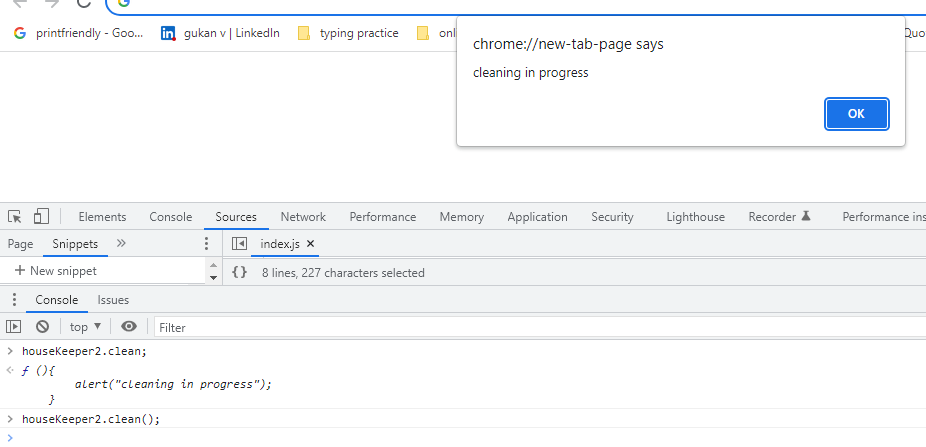
**this.clean = function(){**

**alert("cleaning in progress");**

**}**

}

Here the constructor function with both properties(name, workExperience and cleaningType) and methods(**clean**)



**KEYBOARD EVENT LISTENER**

**document.addEventListener("keydown",function(event){**

**drumSound(event.key);**

**});**

Here the entire webpage look for the event (**key pressed)** to be occurred. At the time when event occurs, addEventListener will execute the anonymous function.

Here **EventTarget** – document – entire web page.

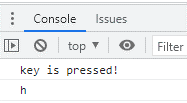
**EventType** - **Element:” keydown” event -**

The keydown event is fired when a key is pressed.

Unlike the deprecated [keypress](https://developer.mozilla.org/en-US/docs/Web/API/Element/keypress_event) event, the keydown event is fired for all keys, regardless of whether they produce a character value.

**LISTENER –** anonymous function executes whenever the **event(key press in keyboard)** occurs.

**drumSound(event.key); -** custom function created in atom editor. Where **event.key** gives the key which was pressed.



document.addEventListener("**keydown**",function(event){

console.log("key is pressed!");

console.log(**event.key**);

});  
here .addEventListener – **higher order function**  and function(event) – **callback function.**

**EXPLANATION**

function **anotherAddEventListener**( **typeOfEvent, callback** ){

//event occured

var **eventThatHappened** = {

eventType : "keydown",

key : "p",

duration : 2

}

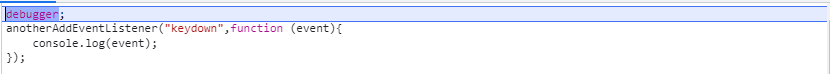
if (**eventThatHappened.eventType === typeOfEvent**){

**callback**(eventThatHappened);

}

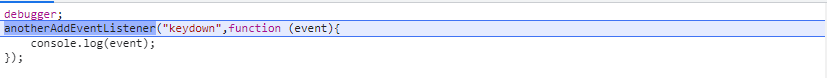
}

**EXECUTION SEQUENCE**

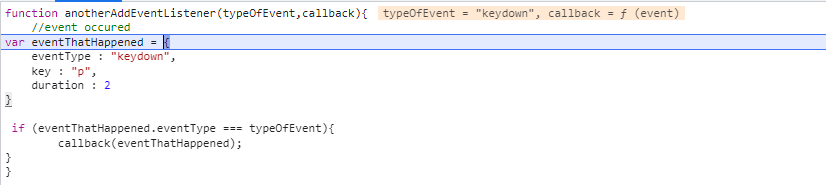


1. we begin by calling **anotherAddEventListener**, passing in these two inputs.

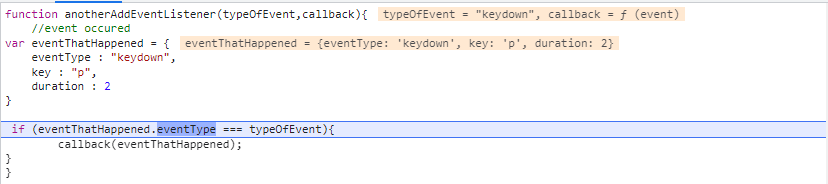
So that takes us to where that function was declared.



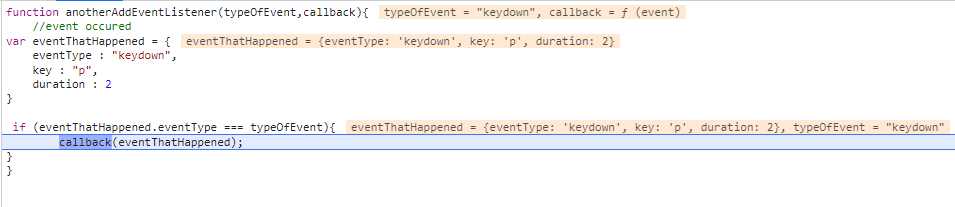
1. The first thing that happens is we're **waiting to detect an event**. And this part of the code waits to detect an event. Let's say that **I press the P key on the keyboard.** Now at that point then this **eventThatHappened** **object gets created** and it captures a lot of information about my key press.



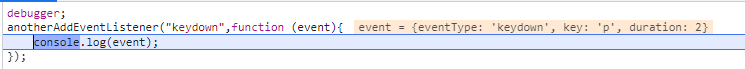
1. this event object created after I triggered the event then our code will **check if the type of the event that happened is equal to the type** that the program was looking for then it will go ahead and trigger the callback function.



1. then the **callback function** (the function that we used as an input) is **triggered**.



1. it goes ahead and tries **to log the event object.**



**SET TIMEOUT FUNCTION**

The setTimeout() method calls a function after a number of milliseconds.

1 second = 1000 milliseconds.

**setTimeout(myGreeting, 5000); -** myGreeting is function, 5000- waiting time(milliseconds).

function buttonAnimation(currentKey){

var btnPressed = document.querySelector("." + currentKey);

btnPressed.classList.add("pressed");

**setTimeout(function(){**

**btnPressed.classList.remove("pressed");**

**},100); - (after 0.1 second the function will be executed.)**

}

**ALTERNATE FORLOOP – forEach()**

The forEach() method calls a function for each element in an array. It is not executed for empty elements.

Here forEach() method will execute the function three(no. of items in fruits array) times until all the elements in the array gets logged in the console.

Const fruits = [“apple”,”mango”,”lemon”]; **output**

fruits.**forEach**(function(fruit){ apple

console.log(fruit); mango

}); lemon

**MAP/FILTER/REDUCE**

Same with **map** and **forEach.**

var numbers = [3, 56, 2, 48, 5];

const newNumbers = numbers.map(function (x) {

return x \* 2;

});

////////////////////////forEach//////////////////////////////////

let newNumbers1 = [];

numbers.forEach(function (x) {

newNumbers.push(x \* 2);

});

console.log(newNumbers);

**CONSOLE **

**FILTER -** filter function is going to look through each of the numbers inside this numbers array and for each of these numbers, it's going to return only the ones that meet a particular criterion.

var numbers = [3, 56, 2, 48, 5];

const newNumbers = numbers.filter(function(num){

return num >10

});

////////////////////////forEach//////////////////////////////////

const newNumbers = [];

numbers.forEach(function (num) {

if (num > 10) {

newNumbers.push(num);

}

});

console.log(newNumbers); 

**REDUCE** - Accumulate a value by doing something to each item in an array.

var numbers = [3, 56, 2, 48, 5];

const number = numbers.reduce(function (accumulator, currentvalue) {

return accumulator + currentvalue;

});

console.log(number);

//////////////////////forEach///////////////////////////////////

let newNumber = 0;

numbers.forEach(function(currentNumber){

newNumber += currentNumber;

});

console.log(newNumber); 

**FIND -** The **find()** method returns the first element in the provided array that satisfies the provided testing function. If no values satisfy the testing function, [undefined](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/undefined) is returned.

var numbers = [3, 56, 2, 48, 5];

const number = numbers.find(function(num){

return num > 10

});

console.log(number); 

**FIND INDEX -**The **findIndex()** method returns the index of the first element in an array that satisfies the provided testing function. If no elements satisfy the testing function, -1 is returned.

var numbers = [3, 56, 2, 48, 5];

const number = numbers.findIndex(function(num){

return num > 10

});

console.log(number); 

**ARROW FUNCTION IN ES6**

Simplifying the code for writing a function.

var numbers = [3, 56, 2, 48, 5];

const newNumbers = numbers.map(function square(x){

return x \* x;

});

var numbers = [3, 56, 2, 48, 5];

const newNumbers = numbers.map(function (x){

return x \* x;

});

Here we simplify by omit the function name 🡪anonymous function

var numbers = [3, 56, 2, 48, 5];

const newNumbers = numbers.map((x)=>{

return x \* x;

});

Here we replace the **function** keyword with **=>**(arrow function)

var numbers = [3, 56, 2, 48, 5];

const newNumbers = numbers.map( x => x \* x);

here we omit the **parenthesis, curly braces, return** keyword.

If function with two parameters, then parenthesis must be included

const newNumbers = numbers.map( (x, y) => x \* y);

if function with two parameters and multi-line expressions then parenthesis, curly braces, return keyword must be included

const newNumbers = numbers.map( (x, y) => { let z = 0;

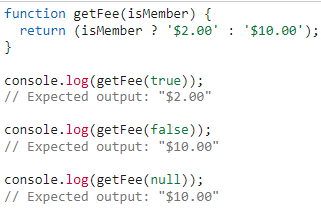
z = x \* y;

return(z\*z);});

# Conditional (ternary) operator

The **conditional (ternary) operator** is the only JavaScript operator that takes three operands: a **condition** followed by a question mark **(?)**, then an **expression** to execute if the condition is **true** followed by a colon (:), and finally the **expression** to execute if the condition is **false** This operator is frequently used as an alternative to an **if…else** statement.

**condition ? exprIfTrue : exprIfFalse**



if we want expression to be executed only the condition is true then

**condition ? exprIfTrue : null**

else use conditional AND operator as below.

**Conditional AND(&&) operator**

It works because in JavaScript, **true && expression** always evaluates to expression, and **false && expression** always evaluates to false.

Therefore, if the condition is true, the element right after && will appear in the output. If it is false, React will ignore and skip it.

{time >= 12 ? <h1>Why are you still working</h1> : null}

Both these lines do same functionality

{time >= 12 && <h1>Why are you still working</h1> }



**JQUERY**

Javascript library used to minimize our code such as

**Document.querySelector(“h1”);** can be replaced by **jQuery(“h1”); OR $(“h1”);**

**ADDING JQUERY TO WEBSITES**

1. Simply add the script(CDN) before the end of the body element and external javascript also.

**<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.3/jquery.min.js"></script>**

<script src="index.js" charset="utf-8"></script>

</body>

**Index.js**

$(“h1”).css(“color”,”red”);

1. Use **ready function** to ensure that jquery script fully loaded before it was used in the website.

**Index.js**

$(document).**ready**(function(){

$("h1").css("color","red");

})

**SELECTING ELEMENTS WITH JQUERY**

Document.querySelector(“h1”); replaced by **$(“h1”);**

Document.querySelectorAll(“button”); replaced by **$(“button”);**

**MANIPULATING STYLES WITH JQUERY**

Get property -



Set property -

By keeping in mind the **separation of concerns,** the styling is only done by CSS.

**Styles.css**

.big-title{

font-size:5rem;

font-family:cursive;

color:yellow;

}

.margin-50{

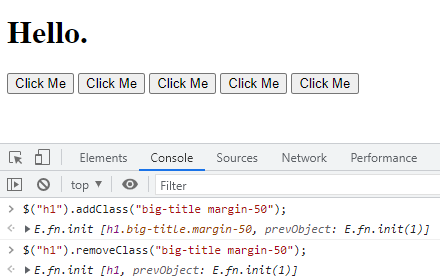
margin:50px;

}

1. **addClass –** used to add classes using jquery



1. **removeClass –** used to remove classes using jquery.

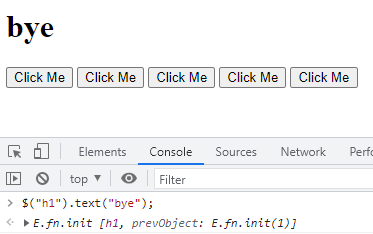


1. **hasClass –** gives true/false whether the specified class is applied or not.



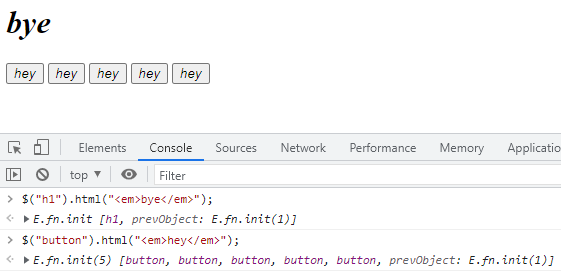
**MANIPULATING TEXT WITH JQUERY**

**.text(“content”); -** method to manipulate the text only using jquery.

****

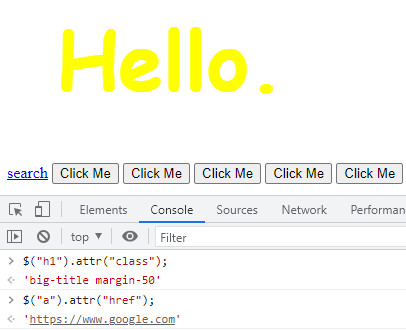
**MANIPULATING TEXT PROPERTY**

**.html(“content”); -** method to manipulate the text and its property also using jquery.

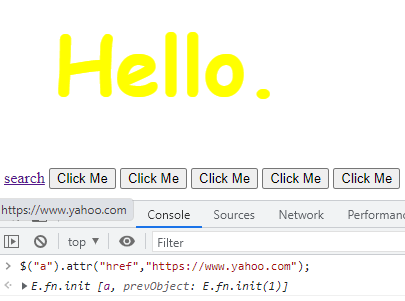
****

**MANIPULATING ATTRIBUTES WITH JQUERY**

**.attr(“attribute name”); -** to get the attribute value.



**.attr(“attribute name”, “value to be changed for that attribute”); -** to set attribute value.



**ADDING EVENT LISTENERS WITH JQUERY**

1. $(document).**keydown**(function(){

$("h1").text(event.key);

});

$("button").**click**(function(){

$("h1").css("color","purple");

});

1. $("button").**on**("**click**",function(){

$("h1").css("color","purple");

});

$(document).**on**("**keydown**",function(){

$("h1").text(event.key);

});

**ADDING ELEMENTS WITH JQUERY**

**Before method –**

$("h1").**before**("<button>new</button>"); 🡪

**<button>new</button>**<h1>hello.</h1>

****

**After method –**

$("h1").**after**("<button>new</button>"); 🡪

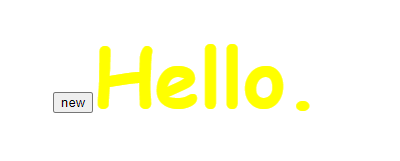
<h1>hello.</h1**><button>new</button>**

****

**Prepend method –**

$("h1").**prepend**("<button>new</button>"); 🡪

<h1>**<button>new</button>**hello.</h1>

****

**Append method –**

$("h1").**append**("<button>new</button>"); 🡪

<h1> hello**.<button>new</button>** </h1>

****

**REMOVING ELEMENTS WITH JQUERY**

**$("button").remove(); -** removes all the specified elements(button).

**WEBSITE ANIMATIONS USING JQUERY**

$("button").on(“click”, function(){

$("h1").[**hide, show, toggle, slideUp, slideDown, slideToggle, fadeIn, fadeOut, fadeToggle]**();

});

$("button").on(“click”, function(){

$("h1").**animate({opacity:0.5, margin:20},300); -** here the **animate** method only acceptproperty with **numeric value.**

});

**CHAINING THE ANIMATIONS**

$("button").on(“click”, function(){

$("h1").**slideUp(300).slideDown(300).animate({opacity:0.5, margin:20},2000);**

});