JAVASCRIPT

Inner HTML

Inside the <body>, just after you have created the element, use:

```
<script>document.getElementById("IdOfYourElement").innerText = 5 </script>
```

this will change the value of said element to 5.

this way you can use js while being on html.

How to link HTML file to JS file/ External JavaScript

External scripts are practical when the same code is used in many different web pages.

JavaScript files have the file extension **.js**.

To use an external script, put the name of the script file in the `src` (source) attribute of a `<script>` tag :

inside the <body> write this in html file

```
<script src="/index.js"></script>
```

then write this in js file

```
document.getElementById("count-el").innerText =5
```

this will do the same work as earlier.

JavaScript in <head> or <body>

You can place any number of scripts in an HTML document.

Scripts can be placed in the `<body>`, or in the `<head>` section of an HTML page, or in both.

JavaScript Functions and Events

A JavaScript `function` is a block of JavaScript code, that can be executed when "called" for.

For example, a function can be called when an **event** occurs, like when the user clicks a button.

External JavaScript Advantages

Placing scripts in external files has some advantages:

- It separates HTML and code
- It makes HTML and JavaScript easier to read and maintain
- Cached JavaScript files can speed up page loads

To add several script files to one page - use several script tags.

Semicolons ;

Semicolons separate JavaScript statements.

Add a semicolon at the end of each executable statement

When separated by semicolons, multiple statements on one line are allowed.

On the web, you might see examples without semicolons.

Ending statements with semicolon is not required, but highly recommended

JavaScript Code Blocks

JavaScript statements can be grouped together in code blocks, inside curly brackets {...}.

The purpose of code blocks is to define statements to be executed together.

One place you will find statements grouped together in blocks, is in JavaScript functions

JavaScript White Space

JavaScript ignores multiple spaces. You can add white space to your script to make it more readable.

JavaScript Line Length and Line Breaks

For best readability, programmers often like to avoid code lines longer than 80 characters.

If a JavaScript statement does not fit on one line, the best place to break it is after an operator

JavaScript Keywords

JavaScript statements often start with a **keyword** to identify the JavaScript action to be performed.

Our Reserved Words Reference lists all JavaScript keywords.

Here is a list of some of the keywords you will learn about in this tutorial:

Keyword	Description
var	Declares a variable
let	Declares a block variable
const	Declares a block constant
if	Marks a block of statements to be executed on a condition
switch	Marks a block of statements to be executed in different cases
for	Marks a block of statements to be executed in a loop
function	Declares a function
return	Exits a function
try	Implements error handling to a block of statements

JavaScript keywords are reserved words. Reserved words cannot be used as names for variables.

JavaScript Display Possibilities

JavaScript can "display" data in different ways:

- Writing into an HTML element, using `innerHTML`.
- Writing into the HTML output using `document.write()`.
- Writing into an alert box, using `window.alert()`.
- Writing into the browser console, using `console.log()`

Using `innerHTML`

To access an HTML element, JavaScript can use the `document.getElementById(id)` method.

The `id` attribute defines the HTML element. The `innerHTML` property defines the HTML content

This we have done earlier in previous page.

Using `document.write()`

For testing purposes, it is convenient to use `document.write()`

CAUTION: use it only for testing purposes.

Using `document.write()` after an HTML document is loaded, will delete all existing HTML

```
document.write("subroto");//prints directly into the webpage
```

Using `window.alert()`

You can use an alert box to display data

You can skip the `window` keyword.

In JavaScript, the window object is the global scope object. This means that variables, properties, and methods by default belong to the window object. This also means that specifying the `window` keyword is optional:

```
alert("subroto")// PRINTS WHAT THE PAGE SAYS
```

Using `console.log()`

For debugging purposes, you can call the `console.log()` method in the browser to display data.

```
var b = 5;
console.log(b);
```

Go to the web page, inspect and click on the console tab to see the output of console.log()

For more details, check index.js in people counter.

JavaScript Print

JavaScript does not have any print object or print methods.

You cannot access output devices from JavaScript.

The only exception is that you can call the `window.print()` method in the browser to print the content of the current window.

JavaScript Variables

In a programming language, **variables** are used to **store** data values.

JavaScript uses the keywords `var`, `let` and `const` to **declare** variables.

An **equal sign** is used to **assign values** to variables.

4 Ways to Declare a JavaScript Variable:

- Using `var`
- Using `let`
- Using `const`
- Using nothing

Difference between var, let and const

var can be used anywhere in the code

let can be used only in the scope of variable

const is constant throughout the code.

JavaScript Comments

Not all JavaScript statements are "executed".

Code after double slashes `//` or between `/*` and `*/` is treated as a **comment**.

Comments are ignored, and will not be executed

JavaScript Identifiers / Names

Identifiers are JavaScript names.

Identifiers are used to name variables and keywords, and functions.

The rules for legal names are the same in most programming languages.

A JavaScript name must begin with:

- A letter (A-Z or a-z)
- A dollar sign ($)
- Or an underscore (_)

Subsequent characters may be letters, digits, underscores, or dollar signs

All JavaScript identifiers are case sensitive.

Math in Javascript

```
var sum = 10 + 10; //sum
var subtract = b-pi; //minus
var multi = 5*2; //multiply
var div = 10/5; //divide
var modulo = 5%2; //modulus
b++; //increment
sum--; //decrement
console.log(sum,subtract,multi,div,modulo,b);
```

similarly you can use += , -= , *- , /= , %= for assignment.

Escape Sequences

\’ single quote

\” double quote

\\ backslash

\n newline

\r carriage return

\t tab

\b backspace

\f formfeed

Operations

Js allows us to perform normal calculations as other coding languages. it has the same arithmetic , assignment operators etc etc

Note: for finding length of string , use stringname.length

Features of strings

We can get characters of string, just like elements of array

eg.

```
var str = “subroto”
console.log(str[0])
```

this will print “s”

> 💡 Strings are immutable in js also means , you cannot change individual characters of the string, but rather the whole string can be changed

Arrays

arrays in js are similar to arrays in other languages, difference being, in js you can store elements of different data type in the same array. you can access elements of array in same way as in other languages.

declaration:

```
var arr = ["subroto",23,52.5]
```

Nested arrays:

```
var arr = [["subroto",23,52.5],["json",29,50.65]]
console.log(arr[0][1])
```

Note: Arrays are mutable

push()

appends elements to an array.

```
var arr = [["sub",1,3.65],["sexy",52]];
arr.push(["hero",123,65.45]);
console.log(arr[2][0]);
```

Console : hero

pop()

deletes an element from end of the array

```
var arr = [1,2,3];
arr.pop();
console.log(arr);
```

output : [1, 2]

shift()

deletes an element from starting of array

```
var arr = [1,2,3];
arr.shift();
```

```
console.log(arr);
```

output : [2, 3]

unshift()

appends an element at the start of the array

```
var arr = [1,2,3];
arr.unshift(4);
console.log(arr);
```

output: [4, 1, 2, 3]

FUNCTIONS IN JAVASCRIPT

```
function hlwrld(a,b){
    console.log(a+b);
}
console.log(hlwrld(10,5));
console.log(hlwrld("subroto ","banerjee"));
console.log(hlwrld("sub",5));
```

output:

15
subroto banerjee
sub5

NOTE :

we can declare a global as well as local variable with the same name, when we do so, the local variable takes precedence over the global one.

```
var s = "tshirt";
function dress(){
```

```
    var s = "books";
    return s;
}
console.log(dress());
console.log(s)
```

output : books

tshirt

if, else, else if STATEMENT

```
function truetha(s){
    if(s){
        return "HAAN BHAI SACH HAI";
    }
    return "JHUT BOLTI HAI YEH";
}
console.log(truetha(true));
console.log(truetha(false));
```

output:

HAAN BHAI SACH HAI
JHUT BOLTI HAI YEH

else can be used same as other coding languages

also else if can be used the same way.

EQUALITY AND INEQUALITY OPERATOR

== is used here also

this will convert both the operands to a common type and then evaluate

eg : 12 == ‘12’ and 12 == 12 both are true

!= is inequality operator, converts both operands to a common type then checks

STRICT EQUALITY AND INEQUALITY OPERATOR

=== is strict equality operator , returns true only when both side operands are strictly equal

eg: 12 === ‘12’ (false)

12 === 12 (true)

!== is strict inequality operator , does not convert both operands to a common type before checking.

SWITCH STATEMENT

```
function s(val){
    switch(val){
        case 1 : return "one"; break;
        case 2 : return "two"; break;
        case 3 : return "three"; break;
        default: return "ghanta"; break;
    }
}
console.log(s(4));
```

output : ghanta

SPECIAL FEATURE :

```
function s(val){
    switch(val){
        case 1 :
        case 2 :
        case 3 :
        return "ghanta"; break;
        default: return "kuch nahi"; break;
    }
}
console.log(s(1));
console.log(s(2));
console.log(s(3));
console.log(s(4));
```

output :

ghanta

ghanta

ghanta

kuch nahi

OBJECTS IN JAVASCRIPT

Objects are somewhat similar to arrays , just they have properties and its defined values.

```
var sub = {
    "name": "subroto",
    "age":19,
    "sex":"male",
    "education": "10+2"
};
//ACCESSING VALUES USING DOT NOTATION
var s = sub.name;
console.log(s);
//ANOTHER WAY OF ACCESSING USING BRACKET NOTATION
var ss = sub["age"];
console.log(ss);
var sb = sub['education'];
console.log(sb);
```

output:

subroto

19

10+2

we can use objects in place of switch statement

ALL FEATURES OF OBJECTS:

```
var sub = {
    "name": "subroto",
    "age":19,
    "sex":"male",
    "education": "10+2",
    "nested obj": {
        1: "tu hai dumdar"
    }
};
//UPDATING THROUGH DOT
sub.name = "subroto banerjee"
console.log(sub.name);
//ADDING PROPERTIES TO OBJECTS
sub.style = "hip hop";
console.log(sub.style);
//DELETING PROPERTIES
delete sub.style ;
console.log(sub.style);
//TESTING IF OBJECT HAS CERTAIN PROPERTIES
function check(a){
    if(sub.hasOwnProperty(a)){ //THIS FUNCTION CHECKS IF THE
        //SAID PROPERTY IS PRESENT OR NOT
        return sub[a];
    }
    else{
        return " nai mila";
    }
}
console.log(check("class"));
//ACCESSING NESTED OBJECTS
console.log(sub["nested obj"][1])
```

output:

subroto banerjee
hip hop
undefined
nai mila

tu hai dumdar

while AND for LOOPS

```
var i = 0;
var arr = [] ;
//WHILE LOOP
while(i < 5){
    arr.push(i);
    i++
}
console.log(arr);
var ar = [];
//FOR LOOP
for(var j =0 ; j < 10 ;j++){
    ar.push(j);
}
console.log(ar);
```

output:

[0, 1, 2, 3, 4]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

do-while LOOP

it exists as same as other languages

RANDOM MATH FUNCTIONS

```
// MATH RANDOM FUNCTIONS
//GENERATING RANDOM NUMBER BETWEEN 0 AND 1 BUT NOT 1
console.log(Math.random());

//GENERATING RANDOM WHOLE NUMBER
console.log(Math.floor(Math.random()*200));

//GENERATING RANDOM WHOLE NUMBERS BETWEEN A RANGE
var ourMin = 2;
var ourMax = 10;
console.log(Math.floor(Math.random()*(ourMax-ourMin +1))+ourMin);
```

parseInt()

```
console.log(parseInt("100",2));
console.log(parseInt("100"));
```

output :

4

100

ternary operator

```
var c = 22;
var d = 25;
c%2==0 ? console.log('even') : console.log('odd');
d%2==0 ? console.log('even') : console.log('odd');
```

output: even

odd

multiple ternary operators

```
var num = 'ss';
var result = num > 0 ? "positive": num < 0 ? "negative" : num === 0 ?
"zero" : "yeh kya gobar kiye ho be? :)";
console.log(result);
```

output: yeh kya gobar kiye ho be? :)

let keyword

it does not let you declare a variable more than once

```
const arr = [1,2,3];
arr[0] = 3
arr[1] = 5
arr[2] = 56
console.log(arr);
```

this will not return error.

var keyword

lets you create a global variable

const keyword

creates a constant , read only variable, usually it is named in capital letter.

Mutating array declared with const

since const declares a constant variable, a const array cannot be reassigned:

```
const s = [1,2,3];
s= [2,5,8];
console.log(s);
```

the above code will result in error.

but we can do this:

```
const s = [1,2,3];
s[0] = 2;
s[1] = 5;
```

```
s[2] = 8;
console.log(s);
```

the above code will give output : [2 , 5 , 8]

Prevention of data mutation

since const keyword cannot actually protect your data, we have to use another way to actually protect it.

Object.freeze(nameofobject)

use this to totally freeze the constants

```
function freezeObj(){
    "use strict";
    const MATH_CONSTANTS = {
        PI : 3.14
    };

    Object.freeze(MATH_CONSTANTS);//BY USING THIS WE FROZE THE DATA WITHIN
    //THE OBJECT MATH_CONSTANTS AND HAVE ACTUALLY PROTECTED IT

    try{
        MATH_CONSTANTS.PI = 99;
    }catch(ex){
        console.log("kya chull hai be ? ");
    }
    return MATH_CONSTANTS.PI;
}
const PI = freezeObj();
```

output : kya chull hai be ?

ARROW FUNCTIONS, REST AND SPREAD OPERATORS

Anonymous functions are those functions that have no name, so whenever we write anonymous function, we can convert it into an arrow function.

Arrow functions{()=>} are a clear and concise method of writing normal/regular Javascript functions in a more accurate and shorter way. **Arrow functions** were introduced in the ES6 version. They make our code more structured and readable.

Arrow functions are anonymous functions i.e. they are functions without a name and are
not bound by an identifier. Arrow functions do not return any value and
can be declared without the function keyword. They are also called **Lambda Functions**.

Advantages of Arrow functions:

- Arrow functions reduce the size of the code
- The return statement and functional braces are optional for single-line functions.
- It increases the readability of the code.

```
//Arrow function returning only one value
var magic = () => new Date();
console.log(magic());

//Arrow function with parameters
var myc = (arr1,arr2) => arr1.concat(arr2);
console.log(myc("sub","roto"));


//SETTING UP A DEFAULT PARAMETER
const increment = (function(){
    return function increment(number, value = 1){
        return number + value ;
    };
}) ();
console.log(increment(5,2)); //OUTPUT 7
console.log(increment(5)); // OUTPUT 6

//VARIABLE ARGUEMENTS
const sum = (function(){
    return function sum(...args){ // here ... is rest operator,
// used to take variable number of paramters
        return args.reduce((a,b)=>a+b,0);
    };
})();
console.log(sum(1,2,3,4))
console.log(sum(1,2,3,4,5))


//CALL BY VALUE
```

```
const arr1= ["jan","feb","march","april"];
let arr2;
(function(){
    arr2 = [...arr1]; // ... is spread operator
//used to take call by value
    arr1[0] = "potato";
}) ();
console.log(arr1);
console.log(arr2);

//CALL BY REFERENCE
const arr1= ["jan","feb","march","april"];
let arr2;
(function(){
    arr2 = arr1; //not using the spread op, results in call by value and hence
//the output will be the same array for both as both arr1 and arr2
//point to the same array.
    arr1[0] = "potato";
}) ();
console.log(arr1);
console.log(arr2);
```

output:

Sat Mar 25 2023 16:51:49 GMT+0530 (India Standard Time)
subroto
7
6
10
15
['potato', 'feb', 'march', 'april']
['jan', 'feb', 'march', 'april']

['potato', 'feb', 'march', 'april']

['potato', 'feb', 'march', 'april']

Limitations of Arrow functions:

- Arrow functions do not have the prototype property
- Arrow functions cannot be used with the new keyword.
- Arrow functions cannot be used as constructors.

- These functions are anonymous and it is hard to debug the code.

DESTRUCTURING ASSIGNEMENT - check link for more

Destructuring Assignment is a JavaScript expression that allows to **unpack values** from arrays, or properties from objects, into distinct variables data can be extracted from *arrays, objects, nested objects* and **assigning to variables** . In Destructuring Assignment on the left-hand side defined that which value should be unpacked from the sourced variable.

Array Destructuring:

```
//SIMPLEST WAY
const [x,y] = [1,2];
console.log(x);
console.log(y);
console.log([x,y]);   //output will be:
//1
//2
//[1,2]
```

using spread operator

```
const arr = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15];
const [...args] = arr;
console.log(args);
console.log(arr);

//small manipulations
const arr = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15];
const [ , , , , , , , , , , ,...args] = arr;
console.log(args);

//another
const arr = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15];
const [ a,b ,...args] = arr;
console.log(args);
```

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

output:

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]

[11, 12, 13, 14, 15]

[3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
1
2

Object Destructuring:

```
const s = {
    "name" : "subroto",
    "age" : 19,
    "gender" : "M"
};
const {"name":a,"age":b,"gender":c} = s;
console.log(a)
console.log(b)
console.log(c)
```

output: subroto

19

M

another way using spread operator

```
const s = {
    "name" : "subroto",
    "age" : 19,
    "gender" : "M"
};
```

```
const {"name":x,...restof} = s;
console.log(x)
console.log(restof)
```

output :

subroto

{ age : 19 , gender : ' M ' }

Template Strings/Template Literals

Template Literal in ES6 provides new features to create a string that gives more control over dynamic strings. Traditionally, String is created using **single quotes (')** or **double quotes (")** quotes. Template literal is created using the **backtick (`)** character.

Multiline Strings: In-order to create a multiline string an escape sequence **\n** was used to give new line character. However, Template Literals there is no need to add **\n** string ends only when it gets **backtick (`)** character.

Expressions: To dynamically add values into new Template Literals expressions are used. The ${} syntax allows an expression in it that produces the value. This value can be a string stored in a variable or a computation operation.

```
const person = {
    "name" : "slim shady",
    "age" : 69
};

const s = `Hi my name is chick chick ${person.name}
I am ${person.age} years old` ;
console.log(s);
```

output:

Hi my name is chick chick slim shady
I am 69 years old

Tagged Templates: One of the features of Template Literals is its ability to create Tagged Template Literals. Tagged Literal is written like a function definition, but the difference is when this literal is called. There is no parenthesis() to a literal call. An array of Strings are passed as a parameter to a literal.

```
function TaggedLiteral(strings){
    console.log(strings);
}

TaggedLiteral `subroto banerjee` ;
```

output: subroto banerjee

Raw String: Raw method of template literal allows access of raw strings as they were entered, without processing escape sequences. In addition, the String.raw() method exists to create raw
strings just like the default template function, and string concatenation would create.

```
var s = String.raw`hi my name is ${60+9}`;
console.log(s);
```

TIP:

you can shorten the assignment of values to keys of objects like this:-

```
const createPerson = (name , age, gender ) => ({name,age,gender});
console.log(createPerson("albert sins",69,"unknown"));
```

OUTPUT:

{name: 'albert sins', age: 69, gender: 'unknown'}

CLASSES IN JAVASCRIPT

similar in fashion to Java

```
class Spaceshuttle {
    constructor(target){
        this.target = target;
    }
}
var v = new Spaceshuttle('earthwa');
console.log(v.target);
```

output : earthwa

GETTERS AND SETTERS

```
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;
}
const thermostat = makeClass();
const thermos = new thermostat(76);
let temp = thermos.temperature;
thermos.temperature = 26;
temp = thermos.temperature;
console.log(temp);
```

output : 26

NOT A NUMBER (Nan) Property

In JavaScript, **NaN** stands for **N**ot **a N**umber. It represents a value that is not a valid number. It can be used to check whether a number entered is a valid number or not a number. To assign a variable to NaN value, we can use one of the two following ways.

Syntax:

```
var a = NaN
    // OR
var a = Number.NaN
```

JavaScript Nullish Coalescing(??) Operator

Nullish Coalescing Operator returns the right-hand value if the left-hand value is **null** or **undefined**. If not null or undefined then it will return left-hand value.

There are values in JavaScript like 0 and an empty string that are logically false by nature. These values may change the expected behavior of the programs written in JavaScript. All the reoccurring problems led to the development of the **Nullish Coalescing Operator**. The Nullish Coalescing Operator is defined by two adjacent question marks **??** :

When the passed parameter is less than the number of parameters defined in the function prototype, it is assigned the value of **undefined**.
To set default values for the parameters not passed during the function call, or to set default values for fields not present in a JSON object, the above method is popular.