**How Website Front-end works?**

|  |  |  |
| --- | --- | --- |
| **HTML/HTML** | **CSS/CSS3** | **JavaScript** |
| HTML stands for Hyper Text Markup Language. | CSS stands for Cascading Style Sheets. | JavaScript is a text-based programming language used for both client side and server side that allows you to make interactive web pages. |
| HTML is the standard markup language for creating web pages. | CSS describes how HTML element will positioned and display on the screen. | JavaScript improves the user experience of the web pages by converting it from a static page into an interactive one. |
| HTML describes the structure of web page. | CSS can control the color, background-color, font, font-size and many more actions. |  |

**VALUES AND VARIABLES**

*var* **myName** = “Deeproshan”;

In above statement *var* is a variable (key) **myName** is variable name and **Deeproshan** is value.

**VARIABLE NAMING CONVENTION**

* The first character must be a letter or an underscore (\_) or a dollar sign ($). You can’t use a number as the first character.
* The rest of the variable name can include any letter, any number or the underscore (\_). You can’t use any other characters, including spaces.
* Variable names are case sensitive.
* No limit to the length of the variable name.
* You can’t use one of JavaScript reserved keywords as a variable name.

**Examples**

*var* \_myName = “Deeproshan”; // Valid

*var* \_my\_\_Name = “Deeproshan”; // Valid

*var* 1myname = “Deeproshan”; // Invalid

*var* $myName = “Deeproshan”; // Valid

*var* myName% = “Deeproshan”; // Invalid

**DATA TYPES IN JAVASCRIPT**

In JavaScript we have **six** primitive **data types** which are,

**undefined**

**boolean**

**number**

**string**

**bigInt**

**symbol**

**CHALLENGE**

10 + “20” = **1020**

9 – “5” = 4 (**Bug**)

“Java” + “Script” = **JavaScript**

“ “ + “ “ =

“ “ + 0 = **0**

“Deep” – “Roshan” = **NaN**

true + true = **2**

true + false = **1**

false + true = **1**

false – true = **-1**

**INTERVIEW QUESTION**

**1. What is the difference between null vs undefined?**

**Example**

*var* iAmUseless = null;

console.log (iAmUseless); // **null**

console.log (typeof(iAmUseless)); // **object (Bug)**

*var* iAmStandBy;

console.log (iAmStandBy); // **undefined**

console.log (typeof(iAmStandBy)); // **undefined**

**2. What is NaN?**

NaN is a property of global object.

In other words it is a variable in global scope.

The initial value of NaN is Not a Number.

**Example**

*var* myPhoneNumber = 727105\*\*\*\*;

*var* myName = “Deeproshan”;

console.log (myPhoneNumber); // **727105\*\*\*\***

console.log (myName); // **Deeproshan**

console.log (isNaN(myPhoneNumber)); // **false**

console.log (isNaN(myName)); // **true**

**CHALLENGE**

NaN === NaN // **false**

Number.NaN === NaN // **false**

isNaN(NaN) // **true**

isNaN(Number.NaN) // **true**

Number.isNaN(NaN) // **true**

**EXPRESSION AND OPERATORS**

5 + 25 = 30

In the above statement **5** and **25** is operand, + sign is operator, and **5 + 25** is expression.

In the JavaScript we have some types of operators which are listed below,

1. Assignment Operator
2. Arithmetic Operator
3. Increment and Decrement Operator
4. Comparison Operator
5. Logical Operator
6. String Operator
7. Conditional (Ternary) Operator

**ASSIGNMENT OPERATOR**

An assignment operator assigns a value to its left operand based on the value of its right operand.

The simple assignment operator is equal sign (=).

**Example**

*var* x = 5;

*var* y = 5;

console.log (“Is the both x and y are equals or not?” + x === y); // **false**

**ARITHMETIC OPERATOR**

An arithmetic operator takes numerical values as their operands and return a single numerical value.

console.log (3 + 3); // **6**

console.log (10 – 5); // **5**

console.log (20 / 5); // **4**

console.log (5 \* 5); // **25**

console.log (20 % 6); // **2**

**INCREMENT AND DECREMENT OPERATOR**

**Operator** : x++( Postfix Increment ) or ++x ( Prefix Increment ) or x— (Postfix Decrement ) or –x (Prefix Decrement )

If used postfix, with operator after operands the increment operator increments and return the value before incrementing.

If used prefix, with operator after operands the increment operator increments and returns the value after incrementing.

**Example 1**

*var* num = 30;

*var* newNum = num++;

console.log (num); // **31**

console.log (newNum); // **30**

**Example 2**

*var* num = 30;

*var* newNum = ++num;

console.log (num); // **31**

console.log (newNum); // **31**

**COMPARISON OPERATOR**

A comparison operator compares its operands and returns a logical value based on whether the comparison is true.

**Example**

*var* a = 40;

*var* b = 10;

console.log (a == b); // **false**

console.log (a != b); // **true**

console.log (a > b); // **true**

console.log (a < b); // **false**

console.log (a >= b); // **true**

console.log (a <= b); // **false**

**LOGICAL OPERATOR**

Logical operators are typically used with Boolean (logical) values.

When they are, they return a boolean value.

**LOGICAL AND (&&)**

The logical AND operator for a set of operands it true if and only if all of its operands are true.

**Example**

*var* a = 30;

*var* b = -30;

console.log (a > b && b > 0); // **false**

**LOGICAL OR (||)**

The logical OR (||) operator (logical disjunction) for a set of operands is true if and only if one or more of its operands is true.

**Example**

*var* a = 10;

*var* b = 20;

console.log (a>b || b<a); // **false**

**LOGICAL NOT (!)**

The logical NOT operator (logical complement negation) takes truth to falsity and vice versa.

**Example**

*var* a = 10;

*var* b = 20;

console.log ((a>0) || (b<a)); // **true**

console.log (!(a>0) || (b<a)); // **false**

**STRING OPERATOR**

The concatenation operator (+) concatenates two string values together returning another string that is the union of the two operand string.

console.log (“Hello,” + “ JavaScript!”); // **Hello, JavaScript!**

**CONDITIONAL (TERNARY) OPERATOR**

The conditional ternary operator is the only JavaScript operator that takes three operands.

*var* name = (condition) ? value1 (true) : value2 (false)

**CHALLENGE**

**WAP to swap two numbers by using third variable and without third variable also.**

Without using third variable :

*var* a = 5;

*var* b = 10;

a = a + b;

b = a – b;

a = a – b;

console.log (a); // **10**

console.log (b); // **5**

**Using third variable**

*var* a = 5;

*var* b = 10;

*var* c = b;

b = a;

a = c;

console.log (a); // **10**

console.log (b); // **5**

**INTERVIEW QUESTION**

**What is the difference between == and === ?**

In JavaScript when we use == then its check only assigned value if they are equal then its return true. Whenever in the case of === if assigned value and data type both are equal then its return true.

**Example**

*var* num1 = 5;

*var* num2 = ‘5’;

console.log (num1 == num2); // **true**

console.log (num1 === num2); // **false**

**CONTROL STATEMENTS AND LOOPS**

**If - else**

The if statement executes a statement if a specified condition is true. If the condition is false another statement can be execute.

**Example**

*var* num = 404;

if (num==404){

console.log (“Oops, page not found !”); // **Oops, page not found !**

}else{

console.log (“Something went wrong…!”);

}

**OUTPUT**

Oops, page not found !

**CHALLENGE**

**WAP a program to check provided year is a leap year or not?**

*var* year = 2020;

if (year%4 === 0){

if (year%100 === 0){

if (year%400 === 0){

console.log (“The year” +year+ “is a leap year.”);

}else{

console.log (“The year” +year+ “is not a leap year.”);

}

}else{

console.log (“The year” +year” “is a leap year.”0;

}

}else{

console.log (“The year” +year” “is not a leap year.”);

}

**OUTPUT**

The year 2020 is a leap year.

**INTERVIEW QUESTION**

**What is Truthy and Falsy value in JavaScript?**

We have total five falsy value in JavaScript which are,

0, “”, undefined, null, NaN, false

If we assign these five value in any condition then they always executes else block.

**Example**

if (score = 0){

console.log (“OMG, we loss the game…!”);

}else{

console.log (“Yes, we won the game…!”);

}

**OUTPUT**

Yes, we won the game…!

if (score = “”){

console.log (“OMG, we loss the game…!”);

}else{

console.log (“Yes, we won the game…!”);

}

**OUTPUT**

Yes, we won the game…!

if (score = undefined){

console.log (“OMG, we loss the game…!”);

}else{

console.log (“Yes, we won the game…!”);

}

**OUTPUT**

Yes, we won the game…!

if (score = null){

console.log (“OMG, we loss the game…!”);

}else{

console.log (“Yes, we won the game…!”);

}

**OUTPUT**

Yes, we won the game…!

if (score = NaN){

console.log (“OMG, we loss the game…!”);

}else{

console.log (“Yes, we won the game…!”);

}

**OUTPUT**

Yes, we won the game…!

if (score = false){

console.log (“OMG, we loss the game…!”);

}else{

console.log (“Yes, we won the game…!”);

}

**OUTPUT**

Yes, we won the game…!

**Switch Statement**

Switch statement evaluates an expression, matching the expression’s value to a case. Clause and executes statements associated with that case.

**Example**

*var* area = “circle”;

*var* PI=3.142, l=5, b=4, r=3;

switch (area){

case ‘circle’:

console.log (“The area of circle is : “+PI\*r\*\*2);

break;

case ‘triangle’:

console.log (“The area of triangle is : “+(l+b)/2);

break;

case ‘rectangle’:

console.log (“The area of rectangle is :”+(l\*b));

break;

case default:

console.log (“Oops, Something went wrong!”);

}

**OUTPUT**

The area of circle is : 28.278

**While loop**

The while statement creates a loop that executes a specific statement as long as the test condition evaluates to true.

**Example**

*var* num = 1;

while (num<=10){

console.log (num);

num++;

}

**OUTPUT**

1

2

3

4

5

6

7

8

9

10

**Do-while loop**

The do...whilestatement creates a loop that executes a specified statement until the test condition evaluates to false. The condition is evaluated after executing the statement, resulting in the specified statement executing at least once.

**Example**

*let* result = '';

*let* i = 0;

do {

i = i + 1;

result = result + i;

} while (i < 5);

console.log (result);

**OUTPUT**

12345

**For loop**

The for statement creates a loop that consists of three optional expressions, enclosed in parentheses and separated by semicolons, followed by a statement (usually a [block statement](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/block)) to be executed in the loop.

**Syntax**

for (initializer; condition; iteration){

// **Code to be executed.**

}

**Example**

*let* str = '';

for (let i = 0; i < 9; i++) {

str = str + i;

}

console.log(str);

**OUTPUT**

012345678

**CHALLENGE**

**Print Table of a 21 using for loop.**

*var* num=21;

for (let i=1; i<=10; i++){

console.log (num + ”\*” + i + ”=” + num\*i);

}

**OUTPUT**

21 \* 1 = 21

21 \* 2 = 42

21 \* 3 = 63

21 \* 4 = 84

21 \* 5 = 105

21 \* 6 = 126

21 \* 7 = 147

21 \* 8 = 168

21 \* 9 = 189

21 \* 10 = 210

**For in loop**

The for...instatement iterates over all [enumerable properties](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Enumerability_and_ownership_of_properties) of an object that are keyed by strings (ignoring ones keyed by [Symbol](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Symbol)s), including inherited enumerable properties.

**Example**

*var* myFriends = [‘Shubham’, ‘Ajeet’, ‘Sandeep’, Dheerendra’, ‘Deepak’];

for ( let elements in myFriends ){

console.log ( elements );

}

**OUTPUT**

0

1

2

3

4

*const* object = { a: 1, b: 2, c: 3 };

for (const property in object) {

console.log(`${property}: ${object[property]}`);

}

**OUTPUT**

"a: 1"

"b: 2"

"c: 3"

**for of loop**

The for...ofstatement creates a loop iterating over [iterable objects](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Iteration_protocols" \l "the_iterable_protocol), including: built-in [String](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String), [Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array), array-like objects (e.g., [arguments](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/arguments) or [NodeList](https://developer.mozilla.org/en-US/docs/Web/API/NodeList)), [TypedArray](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/TypedArray), [Map](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map), [Set](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set), and user-defined iterables. It invokes a custom iteration hook with statements to be executed for the value of each distinct property of the object.

**Example**

*var* myFavColors = [‘Black’, ‘Cyan’, Hotpink’, ‘Blue’, ‘Red’];

for ( let elements of myFavColors ){

console.log ( myFavColors );

}

**OUTPUT**

Black

Cyan

Hotpink

Blue

Red

**forEach**

The **forEach()** method executes a provided function once for each array element.

Doesn’t support this argument.

**Example**

*const* array1 = ['a', 'b', 'c'];

array1.forEach(element => console.log(element));

**OUTPUT**

a

b

c

**FUNCTION**

Functions are one of the fundamental building blocks in JavaScript. A function in JavaScript is similar to a procedure - a set of statements that performs a task or calculates a value, but for a procedure to qualify as a function, it should take some input and return an output where there is some obvious relationship between the input and the output. To use a function, you must define it somewhere in the scope from which you wish to call it.

**FUNCTION DEFINITION**

A function definition (also called a function declaration, or function statement) consists of the [function](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/function) keyword, followed by:

The name of the function.

A list of parameters to the function, enclosed in parentheses and separated by commas.

The JavaScript statements that define the function, enclosed in curly brackets, {...}.

**Syntax**

*function* functionName( ){

// **Code to be exected**

}

**Example**

*var* a = 10;

*var* b = 20;

*var* sum = a + b;

console.log (sum);

**OUTPUT**

30

*function* sumOfTwoNumbers( ){

*var* a = 10, b = 20;

*var* total = a + b;

}

**CALLING FUNCTION**

Defining a function does not execute it. Defining it names the function and specifies what to do when the function is called.

**Calling** the function actually performs the specified actions with the indicated parameters. For example, if you define the function sumOfTwoNumbers, you could call it as follows:

**sumOfTwoNumbers( );**

**FUNCTION PARAMETER VS FUNCTION ARGUMENT**

Function parameters are the name listed in the function’s definitions.

Function arguments are the real values passed to the function.

*function* sumOfTwoNumbers(a,b){ // **a and b are parameters.**

*var* sum = a+b;

console.log(sum);

}

sumOfTwoNumbers(10,20); // **10 and 20 are arguments.**

**INTERVIEW QUESTION**

**Why function?**

We can reuse code write once use many times.

We can use same code many times with different arguments.

OR,

A function is a group of reusable code which can be called anywhere in your program. This eliminates the need of writing the same code again and again.

**FUNCTION EXPRESSION**

While the function declaration above is syntactically a statement, functions can also be created by a function expression. Such a function can be **anonymous**; it does not have to have a name. For example, the function sum could have been defined as:

OR

Function expression simply means create a function and put it into the variable.

**Example**

*function* sum (a,b){

*var* total = a+b;

console.log (total);

}

*var* funExp = sum (10,40); // **50**

**RETURN KEYWORD**

When JavaScript reaches on a return statement, then that function will stop execution.

Function often computes a return value.

The return value is returned back to the caller.

**Example**

*function* sum (a,b){

return total = a+b;

}

var funExp = sum(20,30);

console.log (funExp); // **50**

**ANONYMOUS FUNCTION**

MDN …

**ARRAY IN JAVASCRIPT**

**Example**

**Lower index lower boundary Upper index upper boundary**

*var* names = [“Deep”, “Roshan”, “Ruchi”, “Aruhi”];

**Index**  *0 1 2 3*

console.log (names.length); // **4**

Array Methods

MDN

How to insert, add, replace and delete elements in array

MDN

**CHALLENGES**

**How to add December at the end of an array?**

*const* months = [‘January’, ‘February’, ‘March’, ‘April’];

**What is the return type of the splice method?**

[]// **Empty array**

**How do we can update march by March using splice method?**

*const* months = [‘January’, ‘February’, ‘march’, ‘April’];

**How to we can delete April from an array using splice method?**

**MAP, REDUCE AND FILTER**

**MAP**

**MDN**

**REDUCE**

**How to flat an array… Can’t flat nested array.**

**MDN**

**FILTER**

**MDN**

**CHALLENGE**

Find the Square root of each element in array.

*let* arr = [20, 40, 60, 80, 100];

*let* arrSqrt = arr.map ((currElem)=>{

return Math.sqrt (currElem);

});

console.log (arrSqrt);

**OUTPUT**

[

4.47213595499958,

6.324555320336759,

7.745966692414834,

8.94427190999916,

10

]

**Multiply each elements by 2 and return elements if they are greater than 10.**

*let* arr = [2, 3, 4, 5, 6, 7, 8];

*let* newArr = arr.map ((currElem)=>{

return currElem\*2;

}).filter ((currElem)=>{

return currElem>10;

});

console.log (newArr);

**OUTPUT**

[ 12, 14, 16 ]

**Compressed Solution**

*let* arr = [2, 3, 4, 5, 6, 7, 8];

*let* newArr = arr.map (currElem=>currElem\*2).filter((currElem)=>currElem>10);

console.log (newArr);

**OUTPUT**

[ 12, 14, 16 ]

**STRINGS**

**What is string?**

A JavaScript string is zero or more characters written inside quotes.

JavaScript strings are used for storing and manipulating text.

We can use both single or double quotes to represent any string.

String can be created as primitive from string literals, or as objects using the string( ) constructor.

**Example**

*let* myName = “Deeproshan”;

*let* myFullName = “Deeproshan Kumar”;

console.log (myName);

console.log (myFullName);

**OUTPUT**

Deeproshan

Deeproshan Kumar

**STRING METHODS**

MDN

**ESCAPE CHARACTER**

Example

*let* sentence = “I am \”Deep\” from India.”;

*let* sentence2 = ‘I am \’Deep\’ from India.’;

*let* sentence3 = ‘I am “Deep” from India.”;

*let* sentence4 = “I am ‘Deep’ form India.”;

console.log (sentence);

console.log (sentence2);

console.log (sentence3);

console.log (sentence4);

**OUTPUT**

I am "Deep" form India.

I am 'Deep' form India.

I am “Deep” from India.

I am ‘Deep’ form India.

**FINDING A STRING IN STRING**

**String.prototype.indexOf (searchValue[, fromIndex]) MDN**

The indexOf( ) method returns the index of (the position of) the first occurrence of a specified text in a string.

**Example**

*const* myBio = ‘I am programmer.’;

console.log (myBio.indexOf(“programmer”)); // **5**

**String.prototype.lastIndexOf (searchValue[, fromIndex]) MDN**

The lastIndexOf( ) method returns the index within the calling string object of the last occurrence of search value or -1 if index is not found.

Example

*const* myBio = ‘I am programmer.’;

console.log (mybio.lastIndexOf(“p”, 5); // **5**

**SEARCHING FOR A STRING IN A STRING**

**String.prototype.search(regexp) MDN**

The search( ) method searches a string for a specified value and returns the position of the match.

The search( ) method can’t take a second start position argument.

**Example**

*const* myBio = ‘I am programmer.’;

*let* res = myBio.search(“programmer”);

console.log (res); // **5**

**EXTRACTING STRING PARTS**