JavaScript Topics
---
variables
operators
control statements
  decision making control statements
  iterative control statements (loops)
JS pop-up boxes
Arrays
Functions in JS
  functions with parameters/without parameters
  functions with return value/without return value
  function expressions
  functions with default arguments
  using functions for handling events
  closures
  IIFE (Immediately Invoked Function Expression)
DOM (document Object Model)
 Handling one element at a time
 Handling multiple elements at a time
Object Oriented Prgoramming concepts
(OOP's) - ECMA6
How to write a class
How to write methods in a class
How to create objects and call methods
What is meant by inheritance
What are interfaces

In a HTML page, we can write <script></script> any where and that too for any number of times.

All that Javascript that we want to execute on behalf of a page must always between <script></script> tags only.
---

125 -7689 0 -5 integers

9.324 0.00005 -345.67 floating point values

'hello' 'a' '*' string (text type values)
"hello" "a" "*" "5"

(both single quotes and double quotes are allowed)

true/false boolean values

JavaScript uses an interpreter.
Every browser has JS interpreter present inside it as an extension.

```
<script>
  document.write(
    '<h1>John is ' + 9 + ' years young</h1>')
    document.write(
    '<h1>John is ' + 9 + 9 +' years young</h1>')
    document.write(
    '<h1>John is ' + (9 + 9) +' years young</h1>')
</script>
```

```
<script>
  //input
  m = 70
  p = 80
  c = 90

  //process
  total = m + p + c

  //show output

   document.write(
      '<h1> Total Marks are ' +
           total + '</h1>')

</script>
```

In JS every variable must be first defined and then only used in expressions.
Rules to remember while writing names of variables:
1) variable name can consist of alphabets, digits, underscores and $ symbols only. Special characters including space
2) a variable name can not start with a digit
3) we can not use key words or reserved words as names of variables.

In JS, optionally we can declare variables by using any one of the following 2 key words:
1) var
2) let

A variable is a named memory location which can contain one value (data) at a time. If a new value is assigned to a v
sed or overwritten)

How to concatenate values of variables with strings in JS?

In JS, the + operator can act both a arithmetic operator and string concatenation operator as well. When it appears b
n it appears between 2 strings or a string and a number then it will automatically act as string concatenation operato
cally convert numbers also into strings.

Example:

```
<script>
  var age = 9

  document.write(
    '<h1>John is ' + age + ' years young</h1>')

    document.write(
    '<h1>John is ' + age + 9 +' years                    young</h1>')

    document.write(
    '<h1>John is ' + (9 + 9) +' years                    young</h1>')
</script>
```
---
o/p:

John is 9 years young
John is 99 years young
John is 18 years young

Note:

In JS, ( ) called as parenthesis possess highest precedence.

In JS, we can optionally terminate statements with ;
(or)
in JS, we can show end of a statement using ; optionally

In JS, the alert pop-up box is used just to show a message to the user.
But, the prompt pop-up box is used to not only show a prompt but also to take input from the user.

Example:

```
<script>
   var yourname

   yourname = prompt('enter your name')

   document.write(
      '<h1>Hello ' + yourname +
         '! How are you?</h1>')
</script>
```

JS by default works in loosely typed mode. Loose typing sometimes can introduce bugs unknowingly. Therefore, we c
ict it becomes mandatory to declare every variable by using either the key word var or let.

The following example shows how to tell JS interpreter to act in strict typing mode:

```
<script>
   'use strict'

   var maths = 90
   var physics = 99
   var chemistry = 98

   var totalMarks = 0

   totolMarks = maths + physics + chemistry

   document.write(
      '<h1>Total marks are ' +
          totalMarks + '</h1>')

</script>
```

Flow control statements

There are 2 types of flow control statements in JS:

1) decision making flow control statements
2) iterative flow control statements (loops)
---
1) decision making flow control statements:

Statements which are written by using key words like if, if else, if else if , switch etc are all called as decision making f
---
syntax for writing if:

```
if(test-condition)
```

```
{
  statement/statements
}
```

```
10 > 9   true

10 <= 10  true

10 <= 9   false

10 == 10  true

var a = 10
var b = 5

a > b  true
```

```
<script>
    a = prompt('enter an integer number')
    a = parseInt(a)
    b = prompt('enter another integer number')
    b = parseInt(b)
    c = a + b
    document.write(
        '<h1>result of addition is ' + c + '</h1>')
</script>
```

```
<script>
    a = parseInt(prompt(
                'enter an integer number'))

    b = parseInt(prompt('enter another integer                    number'))
    c = a + b
    document.write(
        '<h1>result of addition is ' + c +              '</h1>')
</script>
```

The following example shows how to make a statement come under execution only when a test condition yields true

```
<script>
    var donation = parseInt(
                    prompt(
      'how much would you like to donate'))

    if(donation >= 1000)
    {
    document.write(
        '<h1>you are a very kind person</h1>')
    }

    document.write(
```

```
        '<h2>Thanks for donating ' +
                    donation + '</h2>')

</script>



if(test-condition)
{
  statement
}

if(test-condition)
 satement

-----------
if(test-condition)
{
  satement1;
  statement2;
  statement3;
  ....
}
------------
if else
-----------
syntax:
---------
if(test-condition)
{
  statement/statements
}
else
{
  statement/statements
}
----------------------
```
Write a script that can find whether an integer number entered by user is an even number or odd number

solution:
```
----------
<script>
    var n = parseInt(prompt('enter an integer'))

    var rem = n % 2

    if(rem == 0)
      document.write(
        '<h1>' + n + ' is even</h1>')
    else
       document.write(
          '<h1>' + n + ' is odd</h1>')
</script>

-------------------------
```
The above solution may also be written as shown below:

```
if(n % 2 == 0)
      document.write(
        '<h1>' + n + ' is even</h1>')
```

```
    else
      document.write(
         '<h1>' + n + ' is odd</h1>')
```

Note:

If an arithmetic expression and a relational expression (comparision) are specified together in one area then the JS in
.

Note:

In JavaScript to check for equality there are 2 operators.

1) == !=
2) === !==

The == operator checks only if values are same or not (it does not bother about types)

The === operator checks if both types and values are same or not

Eg:
```
9 == '9'   //true because values are same

9 === '9'  //false because types are not same

'9' === '9' //true because both types and values
            //are same
```

Write a script which can decide grade of a student based on the average marks scored by him/her.

criteria:
---
avg marks >= 60 --> Grade A
avg marks >= 50 --> Grade B
avg marks < 50   --> Grade C

```
<script>
   var avg;
   avg = parseFloat(prompt('enter avg marks'))

   //decision making control structure
   if(avg >= 60)
     alert('Grade A')
  else if(avg >= 50)
     alert('Grade B')
   else
      alert('Grade C')
</script>
```

---
switch case statement
---
switch case is used for resolving menu driven interfaces. If there are large number of options from which user will ch
ter use switch case. Switch case runs faster than nested if else if, but, it has 2 limitations:

1) it can used against int or string values only
2) all comparisions must be == comparisions

syntax:

```
switch(expression)
{
 case expressionValue1:
   statements
 break;
 case expressionValue2:
   statements
 break;
 ....
 case expressionValueN:
   statements
 break;
 default:
   statement
 break;
}
```

Example:

```
<script>
   var choice;
   alert('let me decide your lucky day')
   choice = parseInt(
         prompt('enter any number from 1 to 7'))

switch(choice)
{
  case 1:
   alert('You like Sunday')
```

```
   alert('Happy Day')
  break;

  case 3:
   alert('You like Tuesday')
   alert('Very Boring Too...')
  break;

  case 2:
   alert('You like Monday')
   alert('Very Boring Day')
  break;

   case 4:
   alert('You like Wednesday')
   alert('so..so...')
  break;

  case 5:
   alert('You like Thursday')
   alert('Happy days are beginning')
  break;

  case 6:
   alert('You like Friday')
   alert('Happy days came back...')
  break;

  case 7:
   alert('You like Saturday')
   alert('A very joyful day')
  break;

  default:
   alert('Wrong choice - An unknown Day')
  break;
}
</script>
```

In JS ?: is called as conditional operator.
It is also called as ternary operator.
It acts as an aleternative to if else

syntax:

```
test-condition ? expr_true : expr_false
```

```
<script>
var a;
var b;
var big;

a = parseInt(prompt('enter an integer'))
b = parseInt(prompt('enter another integer'))

big = (a > b) ? a : b

alert('big = ' + big)
```

</script>

Iterative Control statements (Loops)

Increment/Decrement operators:

These operators are also called unary operators.

++ increments value of a variable by step 1
-- decrements value of a variable by step 1
==========
Example:

```
<script>
   var i = 10
   i++   //means i = i + 1
   document.write(
      '<h1>' + i + '</h1>')
</script>
```
o/p:
11
==========
If increment operation is combined with any other operation then the sequence following which increment and the o

PreIncrement:
if ++ appears before its operand then:

1) increment first
2) other operation next

PostIncrement:
If ++ appears after its operand then:

1) other operation first
2) increment next

PreIncrement Example:

```
<script>
   var i = 10
   var j

   j = ++i //increment first, assignment next

   document.write('<h1>i = ' + i +
                  ' j = ' + j + '</h1>')

</script>
```
o/p:
i = 11 j = 11

PostIncrement:

```
<script>
   var i = 10
   var j
```

```
    j = i++  //assignment first, increment next

    document.write('<h1>i = ' + i +
                    ' j = ' + j + '</h1>')

</script>
```

o/p:
i = 11 j = 10

Similarly in JS, Predecrement and Postdecrement operatoins alos exist.

Loops:
---
while
do
for

Loops are used to tell the computer to execute one or more statements repeatedly as long as a test condition is bec
s.

syntax for writing while loop:

```
intialization

while(test-condition)
{
  statement1;
  statement2;
  ....
  modifier expression/statement
  ...
 }
```

Example:

The following script prints a message for 3 times with the help of a while loop:

```
<script>
    var i = 1
    while(i <= 3)
    {
        document.write(
            '<h1>' + i + ' Welcome</h1>')
        i++
    }
</script>
```
o/p:
1 Welcome
2 Welcome
3 Welcome
==========
```
<script>
    var i = 3

    while(i >= 1)
    {
        document.write(
            '<h1>' + i + ' Welcome</h1>')
```

```
        i--
    }
</script>

o/p:
3 Welcome
2 Welcome
1 Welcome
----------------------
```
A do while loop is an inverted form of while loop. In a do while loop the test condition will be evaluated at the end of
nditionally. A do while loop is capable of running for at least once.

```
syntax:
----------
intialization

do
{
  statement1
  statement2
  ...
  modifier statement
}
while(test-condition);
--------------
Example:
---------
<script>
    var i = 1
    do
    {
        document.write(
            '<h1>' + i + ' Welcome</h1>')

            i++
    }
    while(i <= 3);

</script>
-------------------------
for loop:
------
```
A for loop offers best syntax for handling all those situations in which number of iterations is known in advance.

```
syntax:
-----------
for(intialization; test-condition; modifier expr)
{
  statements in the loop body
  ...
}
--------------
<script>
    for(var i = 1; i <= 3; i++)
    {
        document.write(
            '<h1>' + i + ' Welcome</h1>')
    }

</script>
```

Write a script which can find result of the following series:

1 + 2 + 3 10

```
<script>
   var sum = 0

   for(i = 0; i <= 10; i++)
     sum = sum + i;

   document.write('<h1>sum = ' + sum + '</h1>')
</script>
```

Write a script which can find factorial of an integer number entered by the user

4!
1 X 2 X 3 X 4

5!
1 X 2 X 3 X 4 X 5

n!
1 X 2 X 3 X .....n

```
<script>
  var product = 1
  var n = parseInt(
     prompt('enter an integer number'))

  for(i = 1; i <= n; i++)
    product = product * i;


   document.write('<h1>Factorial of ' + n +
                  ' is ' + product + '</h1>')

</script>
```

Logical operators

&& logical AND operator
!! logical OR operator
! logical NOT operator

A student has 3 subjects. Write a script that will ask the user to enter marks obtained in each one of those 3 subjects

criteria:
result can be shown as passed only if the student scores a minimum of 35 marks in each and every subject, otherwis

---

m
p
c

```
if(m >= 35 && p >= 35 && c >= 35)
   alert('passed')
else
   alert('failed')
```

```
------------------------------
m <-- 90
p <-- 12
c <-- 99

if(m < 35 || p < 35 || c < 35)
 alert('failed')
else
 alert('passed')
```

```
T && T --> T
T && F --> F
F && T --> F
F && F --> F
==============
T || T --> T
T || F --> T
F || T --> T
F || F --> F
==============
!T  --> F
!F  --> T
```

Functions ---------- A function is a block of code which can be written once but can be called for many number of times. Advantages of Functions: 1) They provide modularity which makes debugging process easy 2) They help in avoiding redundency.

To achieve modularity, we should ensure that each function is doing only one task.

---------- Javascript has several inbuilt functions and also allows programmers to write user defined functions.

syntax for writing definition of a function:

function name_of_the_function(parameters if any) { statements in the body of the function .... return statement if any }

depending upon the requirement or the type of operation which a function has to do, we can write a function as:

a function with parameters and with return value (or) a function without parameters and with return value (or) a function with parameters but without return value (or) a function without parameters and without return value ---------- Every function must be first defined and only then it can be called. When we call a function, control goes to the function defintion. During this time the control can carry data in the form of parameters also. After branching into the defintion of the function, the control the executes all the statements which are present in the body and in the end returns back to the next statement which immediately appears after the function call. During the time it returns back, it can also bring back a value (return value). ---------- The following example shows how to write functions which does not take parameters and does not return any value:

<script> function dollarline() { var line=''; for(i = 1; i <= 40; i++) line = line + '$ ' document.write('<h1>'+line+'</h1>'); }

function starline() { var line=''; for(i = 1; i <= 40; i++) line = line + '* ' document.write('<h1>'+line+'</h1>'); } starline() //call the function

document.write('<h2>Student<---------->Course</h2>')

dollarline()

document.write('<h3>John <----------> Java</h3>')

document.write('<h3>Allen <----------> Python</h3>') starline() </script> ----------

The following example shows how to write a function such that it takes a parameter and does not return any value.

<script> function charline(symb) { var line=''; for(i = 1; i <= 40; i++) line = line + symb + ' ' document.write('<h1>'+line+'</h1>'); }

charline('*') //specify argument

document.write('<h2>Student<---------->Course</h2>')

var x = '$' charline(x)

document.write('<h3>John <----------> Java</h3>')

document.write('<h3>Allen <----------> Python</h3>')

charline('*')

</script>

---------- The following example shows how to write a function that takes parameters and does not return any value.

<script> //your code function simpleInterest(pamt,ty,ri) { var si = pamt * ty * ri / 100 document.write('<h1>simple interest = ' + si + '</h1>') }

//your friend's code var p = parseInt(prompt('enter principle amount'))

var t = parseInt(prompt('enter time in years'))

var r = parseFloat(prompt('enter rate of interest'))

//calculate simple interest and show the result simpleInterest(p,t,r); </script> ---------- In JavaScript, a function can also return a result after generating it. the following example shows how to define a return type function and also shows how to call it.

<script> //function definition

function simpleInterest(pamt,ty,ri) { var si = pamt * ty * ri / 100 return si }

var p = parseInt(prompt('enter principle amount'))

var t = parseInt(prompt('enter time in years'))

var r = parseFloat(prompt('enter rate of interest'))

//call the above function var res = simpleInterest(p,t,r)

document.write('<h1 style="color:green">simple interest is ' + res + '</h1>') </script> ---------- A function can call another function. The following example show how functions communicate with each other:

<script> function getArea(len,bre) { return len * bre }

function getvolume(len,bre,ht) { return getArea(len,bre) * ht }

var len = 25 var bre = 10 var ht = 12

var vol = getvolume(len,bre,ht)

alert('volume = ' + vol)

</script> ---------- Functions with default parameters ---------- In JavaScript, at the time of writing definition of a function we can also assign default values to its parameters. Default values will be automatically used when ever the interpreter finds that actual arguments are missing in the function call.

sytnax: ---------- function fun_name(param1=value,param2=value,....) {

}

Note: --- Missing arguments must always appear as trailing arguments only.

Example: --------- <script> function charline(symb='*',n = 40) { for(i = 1; i <= n; i++) document.write(symb)

document.write('
') } //call the above function charline() //takes both default values charline('$') //takes second default value only charline('#',60) //default values will not be used

</script> ---------- Function expression in JavaScript ---------- syntax: ----------

var some_expr = function(param1,param2,...){ code inside the body }

syntax for calling a function through expression:

some_expr(arg1,arg2,....) ---------- Example: --------- <script> var x = function(n){ var f = 1 for(i = 1; i <= n; i++) f = f * i

return f }

//call the above function through //its function expression document.write('<h1>' + x(5) + '</h1>') </script> ---------- Handling HTML events using JS functions Closures In JS IIFE Arrays Array comprehensions DOM (Document Object Model) Objects in JS ----------

---------- Handling HTML events using JS functions Closures In JS IIFE (Immediately Invoked Function Expression) Function Lambdas Arrays Array comprehensions DOM (Document Object Model) Objects in JS -------- ---------- The following example shows how to use a javascript function as an event handler: ---------- <html> <head> <script> function show() { var x = document.getElementById("msg")

x.innerHTML = "Welcome To JavaScript"

} </script> </head> <body> <button onclick="show()"> Show Message</button>   </body> </html> ---------- In the above example, document.getElementById() method searches for the element whose id is "msg" and returns its reference into JS code. document.getElementById() acts as a bridge between HTML and JS.

In browser's memory every html page will be first loaded in the form of DOM tree as shown below:

---------- In JS, the innerHTML property of a div, p, span etc represents the text that appears between the corresponding opening tag and the closing tag. -------- What is the meaning of "this" key word in JS? -------- In JS, the this key word always represents current object. ---------- In the following example, we are going to send this as an argument to event handlers so as to avoid further search in the DOM tree by using document.getElementById() method. ---------- html> <head> <script> function fun_focus(x) { x.style.backgroundColor="yellow" }

function fun_blur(x) { x.style.backgroundColor = "red" } </script> </head> <body> Enter Your First Name <input type="text" id="t1" onfocus="fun_focus(this)" onblur="fun_blur(this)">
  Enter Your Last Name <input type="text" id="t2" onfocus="fun_focus(this)" onblur="fun_blur(this)">
 <input type="submit" value="submit"> </body> </html> ---------- How to write JavaScript in an external file and make it the source for an HTML page? ---------- In VSCode --> to the folder in which we are storing all html files --> add a new file --> myscript.js

(make sure that the extension in the file name appears as .js only)

now write code in myscript.js as shown below: ---------- function fun_focus(x) { x.style.backgroundColor="green" }

function fun_blur(x) { x.style.backgroundColor = "red" } ---------- Now add a file called functions2.html to the same folder and write the following html code in it: ---------- <html> <head> <script src="myscript.js"> </script> </head> <body> Enter Your First Name <input type="text" id="t1" onfocus="fun_focus(this)" onblur="fun_blur(this)">
  Enter Your Last Name <input type="text" id="t2" onfocus="fun_focus(this)" onblur="fun_blur(this)">

 <input type=”submit” value=”submit”> </body> </html> ---------- ---------- The following example demonstrates mouseenter and mouseleave events. In order to build this example:

1) download a free image (preferably gif file) from internet and copy it into the same folder in which you are saving you html files also.

2) develop the html file in the same folder.

html and JS code to be developed in .html file: ---------- <html> <head> <script> function bigImg(x) { x.style.height=”128px” x.style.width=”128px” }

function normalImg(x) { x.style.height=”64px” x.style.width=”64px” } </script> </head> <body> <h1>HTMl DOM Events</h1> </body> </html> ----------

Closure in JavaScript ---------- In javascript to build a closure we have to write function inside another function. If there is an inner function inside an outer function then the outer function acts as a closure and will preserve values of its variables during successive calls without destryoing them. ---------- Example: --------- <script>

function outerfun() { var i = 0

function innerfun() { i++ alert(i) }

return innerfun }

//call the outerfun var x = outerfun()

//by now x becomes an expression to innerfun

x() //call the inner function through x x() x()

</script> o/p: 1 2 3 ---------- IIFE -------- Immediately invoked function expression calls an anonymous function immediately as soon as the interpreter reads the function definition.

1) IIFE’s are generally used to perform page initialization activities 2) An IIFE can call only one function that too for only once during the life time of a page.

syntax: ------ (function(){ //write code inside //the body of the //function })() ---------- Example: -------- <script> (function(){ alert(’I was just born’) })() </script> ---------- completed: ---------- Handling HTML events using JS functions Closures In JS IIFE (Immediately Invoked Function Expression) DOM (Document Object Model) ---------- yet to be studied: ---------- Function Lambdas Arrays Array comprehensions Objects in JS

---------- Handling HTML events using JS functions Closures In JS IIFE (Immediately Invoked Function Expression) Function Lambdas Arrays Array comprehensions DOM (Document Object Model) Objects in JS -------- ---------- The following example shows how to use a javascript function as an event handler: ---------- <html> <head> <script> function show() { var x = document.getElementById("msg")

x.innerHTML = "Welcome To JavaScript"

} </script> </head> <body> <button onclick="show()"> Show Message</button>   </body> </html> ---------- In the above example, document.getElementById() method searches for the element whose id is "msg" and returns its reference into JS code. document.getElementById() acts as a bridge between HTML and JS.

In browser's memory every html page will be first loaded in the form of DOM tree as shown below:

---------- In JS, the innerHTML property of a div, p, span etc represents the text that appears between the corresponding opening tag and the closing tag. -------- What is the meaning of "this" key word in JS? -------- In JS, the this key word always represents current object. ---------- In the following example, we are going to send this as an argument to event handlers so as to avoid further search in the DOM tree by using document.getElementById() method. ---------- html> <head> <script> function fun_focus(x) { x.style.backgroundColor="yellow" }

function fun_blur(x) { x.style.backgroundColor = "red" } </script> </head> <body> Enter Your First Name <input type="text" id="t1" onfocus="fun_focus(this)" onblur="fun_blur(this)">
  Enter Your Last Name <input type="text" id="t2" onfocus="fun_focus(this)" onblur="fun_blur(this)">
 <input type="submit" value="submit"> </body> </html> ---------- How to write JavaScript in an external file and make it the source for an HTML page? ---------- In VSCode --> to the folder in which we are storing all html files --> add a new file --> myscript.js

(make sure that the extension in the file name appears as .js only)

now write code in myscript.js as shown below: ---------- function fun_focus(x) { x.style.backgroundColor="green" }

function fun_blur(x) { x.style.backgroundColor = "red" } ---------- Now add a file called functions2.html to the same folder and write the following html code in it: ---------- <html> <head> <script src="myscript.js"> </script> </head> <body> Enter Your First Name <input type="text" id="t1" onfocus="fun_focus(this)" onblur="fun_blur(this)">
  Enter Your Last Name <input type="text" id="t2" onfocus="fun_focus(this)" onblur="fun_blur(this)">

 <input type="submit" value="submit"> </body> </html> ---------- ---------- The following example demonstrates mouseenter and mouseleave events. In order to build this example:

1) download a free image (preferably gif file) from internet and copy it into the same folder in which you are saving you html files also.

2) develop the html file in the same folder.

html and JS code to be developed in .html file: ---------- <html> <head> <script> function bigImg(x) { x.style.height="128px" x.style.width="128px" }

function normalImg(x) { x.style.height="64px" x.style.width="64px" } </script> </head> <body> <h1>HTMl DOM Events</h1> </body> </html> ----------

Closure in JavaScript ---------- In javascript to build a closure we have to write function inside another function. If there is an inner function inside an outer function then the outer function acts as a closure and will preserve values of its variables during successive calls without destryoing them. ---------- Example: --------- <script>

function outerfun() { var i = 0

function innerfun() { i++ alert(i) }

return innerfun }

//call the outerfun var x = outerfun()

//by now x becomes an expression to innerfun

x() //call the inner function through x x() x()

</script> o/p: 1 2 3 ---------- IIFE -------- Immediately invoked function expression calls an anonymous function immediately as soon as the interpreter reads the function definition.

1) IIFE's are generally used to perform page initialization activities 2) An IIFE can call only one function that too for only once during the life time of a page.

syntax: ------ (function(){ //write code inside //the body of the //function })() ---------- Example: -------- <script> (function(){ alert('I was just born') })() </script> ---------- completed: ---------- Handling HTML events using JS functions Closures In JS IIFE (Immediately Invoked Function Expression) DOM (Document Object Model) ---------- yet to be studied: ---------- Function Lambdas Arrays Array comprehensions Objects in JS ---------- Function Lambdas ---------- A function lambda works like a inline function. Function lambdas run faster than

their normal counter parts. Function lambdas can be easily specified as arguments to other functions. Note: ------- A function lambda should preferably contain one or two statements in its body only then it runs fast.

syntax: -------- var some_expr = (arg1,arg2...) => some statement

---------- Following example show a lambda that takes one argument and returns result of an arithmetic expression.

<script> var x = (n) => n * n

alert(x(5)) </script> ---------- (or) ---------- <script> var x = n => n * n alert(x(6)) </script> ---------- The following example shows how to write a function lambda that takes multiple arguments ---- <script> var x = (a,b) => a > b

alert(x(10,15)) </script> ---------- The following example show a function lambda with 0 arguments ---------- <script> var x = () => alert('hello hello hello.....') x() </script> ---------- Note: ------- ( ) must be specified mandatorily if the function lambda take 0 or 2 or more arguments. If it takes only 1 argument then ( ) are optional. ---------- Arrays ----- if(a > b) then a is big otherwise b is big ---------- if(a >b && a > c) then a is big else if(b > c) then b is big else c is big ---------- Arrays --------- An array is a collection of data items. Arrays provide us the ability to access large number of data items sequentially.

Array acts like a data structure.

In an array elements or data items will be stored in adjacent memory locations. That means, memory for an array will be always allocated as one single block.

Elements of an array automatically receive indexes and those indexes start from 0 (and hence the index of the last element will be equal to length - 1 where length denotes number of elements present in the array)

To know the lenght of an array at runtime we can use a predefined property called length.

In JS, at the time of declaring an array we need not specify its size. An array can grow to any extent as we keep on adding elements to it.

In JS, an array can be created by using any one of the following 3 syntaxes:

var somevar = [ele1,ele2,ele3,.....]

var somevar = new Array()

var somevar = new Array(ele1,ele2,ele3,....)

While accessing any element of an array we must write its index in between [ ] (square brackets)

---------- The following script shows how we can access all the elements of an array sequentially: ----------

<script> var arr = [12,15,11,26,98,16,5]; for(i = 0; i < arr.length; i++) document.write('<h1>' + arr[i] + '</h1>'); </script>

---------- There are 2 special forms of for loop which can run faster than the traditional for loop:

1) for..... of loop 2) for .... in loop

syntax for writing for....of loop

for(loop-variable-name of name-of-the-array) { //statements that use loop-variable's data }

In a for....of loop, the loop variable attains value of the next element present in the array.

Example: ---------- <script> var countries = ['india','china','usa','uk']

for(x of countries) document.write('<h2>'+ x +'</h2>')

</script> ---------- The for...in loop on the other hand generates indexes through its loop variable and allows us to access elements through their indexes.

syntax: ------- for(loop-variable-name in name-of-the-array) { code that used the loop variable value as index } ---------- The following example shows how to create an array as an object of Array class: ---------- <script> var nums = new Array() nums[0] = 10 nums[1] = 25 nums[2] = 12 nums[3] = 100

for(i in nums) document.write('<h1>' + i + '--->'+ nums[i] + '</h1>')

document.write('

')

//print only the 3rd element document.write('<h1>' + nums[2] + '</h1>') </script>

---------- Handling HTML events using JS functions Closures In JS IIFE (Immediately Invoked Function Expression) Function Lambdas Arrays Array comprehensions DOM (Document Object Model) Objects in JS -------- ---------- The following example shows how to use a javascript function as an event handler: ---------- <html> <head> <script> function show() { var x = document.getElementById("msg")

x.innerHTML = "Welcome To JavaScript"

} </script> </head> <body> <button onclick="show()"> Show Message</button>   </body> </html> ---------- In the above example, document.getElementById() method searches for the element whose id is "msg" and returns its reference into JS code. document.getElementById() acts as a bridge between HTML and JS.

In browser's memory every html page will be first loaded in the form of DOM tree as shown below:

---------- In JS, the innerHTML property of a div, p, span etc represents the text that appears between the corresponding opening tag and the closing tag. -------- What is the meaning of "this" key word in JS? -------- In JS, the this key word always represents current object. ---------- In the following example, we are going to send this as an argument to event handlers so as to avoid further search in the DOM tree by using document.getElementById() method. ---------- html> <head> <script> function fun_focus(x) { x.style.backgroundColor="yellow" }

function fun_blur(x) { x.style.backgroundColor = "red" } </script> </head> <body> Enter Your First Name <input type="text" id="t1" onfocus="fun_focus(this)" onblur="fun_blur(this)">
  Enter Your Last Name <input type="text" id="t2" onfocus="fun_focus(this)" onblur="fun_blur(this)">
 <input type="submit" value="submit"> </body> </html> ---------- How to write JavaScript in an external file and make it the source for an HTML page? ---------- In VSCode --> to the folder in which we are storing all html files --> add a new file --> myscript.js

(make sure that the extension in the file name appears as .js only)

now write code in myscript.js as shown below: ---------- function fun_focus(x) { x.style.backgroundColor="green" }

function fun_blur(x) { x.style.backgroundColor = "red" } ---------- Now add a file called functions2.html to the same folder and write the following html code in it: ---------- <html> <head> <script src="myscript.js"> </script> </head> <body> Enter Your First Name <input type="text" id="t1" onfocus="fun_focus(this)" onblur="fun_blur(this)">
  Enter Your Last Name <input type="text" id="t2" onfocus="fun_focus(this)" onblur="fun_blur(this)">

 <input type="submit" value="submit"> </body> </html> ---------- ---------- The following example demonstrates mouseenter and mouseleave events. In order to build this example:

1) download a free image (preferably gif file) from internet and copy it into the same folder in which you are saving you html files also.

2) develop the html file in the same folder.

html and JS code to be developed in .html file: ---------- <html> <head> <script> function bigImg(x) { x.style.height="128px" x.style.width="128px" }

function normalImg(x) { x.style.height="64px" x.style.width="64px" } </script> </head> <body> <h1>HTMl DOM Events</h1> </body> </html> ----------

Closure in JavaScript ---------- In javascript to build a closure we have to write function inside another function. If there is an inner function inside an outer function then the outer function acts as a closure and will preserve values of its variables during successive calls without destryoing them. ---------- Example: --------- <script>

function outerfun() { var i = 0

function innerfun() { i++ alert(i) }

return innerfun }

//call the outerfun var x = outerfun()

//by now x becomes an expression to innerfun

x() //call the inner function through x x() x()

</script> o/p: 1 2 3 ---------- IIFE -------- Immediately invoked function expression calls an anonymous function immediately as soon as the interpreter reads the function definition.

1) IIFE's are generally used to perform page initialization activities 2) An IIFE can call only one function that too for only once during the life time of a page.

syntax: ------ (function(){ //write code inside //the body of the //function })() ---------- Example: -------- <script> (function(){ alert('I was just born') })() </script> ---------- completed: ---------- Handling HTML events using JS functions Closures In JS IIFE (Immediately Invoked Function Expression) DOM (Document Object Model) ---------- yet to be studied: ---------- Function Lambdas Arrays Array comprehensions Objects in JS ---------- Function Lambdas ---------- A function lambda works like a inline function. Function lambdas run faster than

their normal counter parts. Function lambdas can be easily specified as arguments to other functions. Note: ------- A function lambda should preferably contain one or two statements in its body only then it runs fast.

syntax: -------- var some_expr = (arg1,arg2...) => some statement

---------- Following example show a lambda that takes one argument and returns result of an arithmetic expression.

<script> var x = (n) => n * n

alert(x(5)) </script> ---------- (or) ---------- <script> var x = n => n * n alert(x(6)) </script> ---------- The following example shows how to write a function lambda that takes multiple arguments ---- <script> var x = (a,b) => a > b

alert(x(10,15)) </script> ---------- The following example show a function lambda with 0 arguments ---------- <script> var x = () => alert('hello hello hello.....') x() </script> ---------- Note: ------- ( ) must be specified mandatorily if the function lambda take 0 or 2 or more arguments. If it takes only 1 argument then ( ) are optional. ---------- Arrays ----- if(a > b) then a is big otherwise b is big ---------- if(a >b && a > c) then a is big else if(b > c) then b is big else c is big ---------- Arrays --------- An array is a collection of data items. Arrays provide us the ability to access large number of data items sequentially.

Array acts like a data structure.

In an array elements or data items will be stored in adjacent memory locations. That means, memory for an array will be always allocated as one single block.

Elements of an array automatically receive indexes and those indexes start from 0 (and hence the index of the last element will be equal to length - 1 where length denotes number of elements present in the array)

To know the lenght of an array at runtime we can use a predefined property called length.

In JS, at the time of declaring an array we need not specify its size. An array can grow to any extent as we keep on adding elements to it.

In JS, an array can be created by using any one of the following 3 syntaxes:

var somevar = [ele1,ele2,ele3,.....]

var somevar = new Array()

var somevar = new Array(ele1,ele2,ele3,....)

While accessing any element of an array we must write its index in between [ ] (square brackets)

---------- The following script shows how we can access all the elements of an array sequentially: ----------

<script> var arr = [12,15,11,26,98,16,5]; for(i = 0; i < arr.length; i++) document.write('<h1>' + arr[i] + '</h1>'); </script>

---------- There are 2 special forms of for loop which can run faster than the traditional for loop:

1) for..... of loop 2) for .... in loop

syntax for writing for....of loop

for(loop-variable-name of name-of-the-array) { //statements that use loop-variable's data }

In a for....of loop, the loop variable attains value of the next element present in the array.

Example: ---------- <script> var countries = ['india','china','usa','uk']

for(x of countries) document.write('<h2>'+ x +'</h2>')

</script> ---------- The for...in loop on the other hand generates indexes through its loop variable and allows us to access elements through their indexes.

syntax: ------- for(loop-variable-name in name-of-the-array) { code that used the loop variable value as index } ---------- The following example shows how to create an array as an object of Array class: ---------- <script> var nums = new Array() nums[0] = 10 nums[1] = 25 nums[2] = 12 nums[3] = 100

for(i in nums) document.write('<h1>' + i + '--->'+ nums[i] + '</h1>')

document.write('

')

//print only the 3rd element document.write('<h1>' + nums[2] + '</h1>') </script>

---------- What is meant by 'undefined' in JS?

In JS, the iterpreter returns undefined if we try to access value of a variable which was not initialized.

Eg: ------ <script> var x document.write('<h1>' + x + '</h1>') </script> o/p: undefined ---------- Difference between for loop and for....in loop ---------- for loop: -------- <script> var nums = new Array() nums[0] = 10 nums[1] = 25 nums[2] = 12 nums[4] = 100

for(i = 0; i < nums.length; i++) document.write('<h1>' + i + '--->'+ nums[i] + '</h1>') </script>

o/p: ---- 0--->10 1--->25 2--->12 3--->undefined 4--->100

In the above example, since index 3 was missing originally in the array, JS returned the state of nums[3] as undefined. ---------- The for....in loop simply treats each index like one key with the help of which the associated value can be indentified. So, it does not rely upon sequential indexes.

Example: ------- <script> var nums = new Array() nums[0] = 10 nums[1] = 25 nums[2] = 12 nums[4] = 100

for(i in nums) document.write('<h1>' + i + '--->'+ nums[i] + '</h1>')

</script>

o/p: 0--->10 1--->25 2--->12 4--->100 ---------- How can we treat an array in JS as a QUEUE?

Ans) QUEUE is a First-in-First-Out (FIFO) Data structure. In order make an array behave like a QUEUE we have to make sure that every new element is always added to the end of the array and also we have to use shift() method to retrieve every next available element as shown below: ---- <script> var nums = new Array(100,200,300) nums[3] = 400 console.log(nums.shift()) console.log(nums.shift()) console.log(nums.shift()) console.log(nums.shift()) </script> o/p: 100 200 300 400 ---------- IN JS, we can use an array like a stack by using pop() method. While using an array in the form of a stack we must follow Last-in-First-out (LIFO) style of data access. -------- <script> var nums = new Array(100,200,300) nums[3] = 400 console.log(nums.pop()) console.log(nums.pop()) console.log(nums.pop()) console.log(nums.pop()) </script>

o/p: 400 300 200 100 ---------- Note: ------- In VSCode, we must always ensure that the terminal window is appearing as the normal command prompt window (cmd window). If there appears the PS prefix before the command prompt then it means that the terminal is running in admin's power shell mode. In order to change it to the normal cmd mode, follow these steps:

1) Kill the present terminal by clicking on the delete icon in the terminal window.

2) ctrl + shift + p --> invoke the command palette

3) In the command pallette, type Terminal: select default profile --> select cmd.exe

4) go to view menu --> select terminal ---------- Note: ------- In the abvoe two examples the shift() method and the pop() method were used to remove elements from QUEUE and STACK respectively. ---------- In JS, the unshift() method always adds an element at the beginning of an array as shown below: ----- <script> var nums = new Array(100,200,300) nums.unshift(10) nums.unshift(20)

for(x of nums) document.write('<h1>' + x + '</h1>') </script> o/p: 20 10 100 200 300 ---------- Array comprehensions in JS ---------- JS provides 2 array comprehensions called as: map() filter()

A array comprehension can avoid explicit usage of a loop and can run faster than any looping mechanism developed by us.

The following example creates a new array by copying data from an existing array. At the time of copying each element from source array into destination

array, element of source array is truncated.

Method1: (basic approach) ---------- Without using the array comprehension called map() ----------

<script> const a = [12.5,45.34,12.8,9.5]

const b = new Array()

for(i = 0; i < a.length; i++) { var x = a[i] x = Math.trunc(x) b[i] = x }

for(t of b) document.write('<h1>' + t + '</h1>')

</script> ========== Method2: (advanced and more efficient approach) ---------- By using Array comprehension called map(): ---------- <script> const a = [12.5,45.34,13.8,9.5]

const b = a.map(x => Math.trunc(x))

for(t of b) document.write('<h1>' + t + '</h1>')

</script>

---------- The filter comprehension can produce a sub array from a main array by extracting only those elements from the main array which satisfy a condition. ---------- The following example creates a sub array without using the filter() comprehension: ---------- <script> const a = [10,20,12,98,35,48,22,17]

const b = new Array()

var i = 0; for(x of a) { if(x >= 20) { b[i] = x i++ } }

for(t of b) document.write('<h1>' + t + '</h1>')

</script> ---------- Below, we are writing solution for the same problem but this time by using the comprehension called filter() ---------- <script> const a = [10,20,12,15,35,48,22,17]

const b = a.filter(x => x >= 20)

for(t of b) document.write('<h1>' + t + '</h1>')

</script>

---------- const in JS: In JS, the const key word is used to define a constant whose value can not be changed after once it is initialized. ---------- <script> var radius = parseFloat(prompt('enter radius'))

const PI = 3.14

var area = PI * Math.pow(radius,2)

document.write('<h1>Area = ' + area + '</h1>')

</script>

========== Objects in JavaScript ---------- An object is a block of fields. The purpose of creating an object is to store data belonging to an entity in the form of one block. By storing data in the form of objects and then by carrying out operations on objects we can achieve abstraction. Abstraction means "hiding complexity or hiding internal details" ---------- ---------- In JS, we can create and store data inside an object by following any one of the following 3 approaches: ---------- 1) An object can be created as a new instance of Object class as shown below: ------- <script> var x = new Object() x.id = 1212 x.name = 'john' x.salary = 5000

var y = new Object() y.id = 1313 y.name = 'donald' y.salary = 4500

console.log(x.id + ' ' + x.name + ' ' + x.salary)

console.log(y.id + ' ' + y.name + ' ' + y.salary) </script>

o/p:

1212 john 5000 1313 donald 4500

---------- 2) Using constructor: A constructor is a function which can not only create object but can also initialize it. All the Objects which are created using same constructor will have same structure and there by promoting consistency.

Example: ---------- <script> //constructor function Employee(id,name,salary) { this.id = id this.name = name this.salary = salary }

var x = new Employee(1212,'allen',7200)

var y = new Employee(1515,'king',9999)

console.log(x.id + ' ' + x.name + ' ' + x.salary)

console.log(y.id + ' ' + y.name + ' ' + y.salary)

</script> ---------- 3) Creating object as a literal.

syntax: ---------- var some_var = { field1 : value, field2 : value, ... } ---------- Example: ------- <script> // creating object as a literal var x = { id:1212, name:'allen', salary:4800 }

var y = {id:1717,name:'blake',salary:5300}

console.log(x.id + ' ' + x.name + ' ' + x.salary)

console.log(y.id + ' ' + y.name + ' ' + y.salary)

</script> ------ o/p 1212 allen 4800 1717 blake 5300 ========== Scopes of variables in JS ----------

There are 3 scopes in javascript.

1) Global scope

In order to place a variable in global scope we have to declare it outside of all functions.

var some_var

function fun1() { // this function can access some_var } function fun2() { //this function can also access some_var } ========== 2) function scope: ---------- For declaring a variable inside a function scope also we have to use the var key word again.

function fun1() { var some_var

//this variable called some_var will become //visible only to this function } ---------- 3) Block scope ---------- In order to declare a variable inside a block scope we must use let key word in javascript. A block scope can appear under if, else or any loop etc

funtion fun1() { if(condition) { let some_var //above variable will become visible //only to this if block }

//we cannot access some_var from //outside of its block } ----------