Java Script

18 July 2023 20:42

Introduction to programming:

Programming is a way to talk to computers. A language Hindi, English, Bengali can be used to talk to a human but for computers we need straight forward instructions

Computer is Dumb!

When was the last time you ordered same cereal and got DVDs of serial?

Programming is the act of constructing a program a set of precise instructions telling a computer what to do.

What is EcmaScript?

ECMA Script is a standard on which java script is based! It was created to ensure that different documents on java script are actually talking about the same language.

Java Script & ECMA Script can almost be used interchangeable Java Script is very liberal in what it allows.

How to execute Java Script?

Java Script can be created right inside one's browser you can open the java script console and start writing java script there

Another way to execute java script is a run time like Node.js which can be installed and used to run java Script code.

Yet another way to create a java script is by inserting it inside <script> tag of HTML document.

Alert
Console.log
Prompt
And all basics

Expressions and Conditional

A fragment of code that produces a value is called an expression. Every value written literally is an expression for example: 77; or "Avinash"

```
77;
"Avinash"
//runs without giving error
//but doesn't prints it.
```

Operators in Java Script:

1. Arithmetic Operators

+	Addition
-	Subtraction
*	Multiplication
**	Exponential
/	Division
%	Modulus
++	Increment
--	Decrement

```
console.log("Operators in JS")
let a = 45;
let b = 10;
console.log('a+b=',a+b)
console.log('a-b=',a-b)
console.log('a*b=',a*b)
console.log('a/b=',a/b)
console.log('a%b=',a%b)
console.log('a**b=',a**b)
console.log('a-b=',a-b)
console.log('a++=',a++)
console.log('a--',a--)
console.log('++a=',++a)
console.log('--a=',--a)
```

Input

```
Operators in JS
a+b= 55
a-b= 35
a*b= 450
a/b= 4.5
a%b= 5
a**b= 34050628916015624
a-b= 35
a++= 45
a-- 46
++a= 46
--a= 45
```

Output

2. Assignment Operator

=	x = y
+=	x= x + y
-=	x= x - y
*=	x= x * y
/=	x= x / y
%=	x= x % y
**=	x= x ** y

```
let a = 1
console.log(a += 5)
console.log(a += 5)
console.log(a -= 5)
console.log(a *= 5)
console.log(a /= 5)
console.log(a %= 5)
console.log(a **=5)
```

Input

```
6
11
6
30
6
1
1
```

Output

3. Comparison Operators

==	equal to
!=	not equal
===	equal value and type
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator

```
let comp1 = 6;
let comp2 = "6";
console.log("comp1 == comp2 is ", comp1 == comp2)
console.log("comp1 != comp2 is ", comp1 != comp2)
console.log("comp1 === comp2 is ", comp1 === comp2)
console.log("comp1 !== comp2 is ", comp1 !== comp2)
console.log("comp1 > comp2 is ", comp1 > comp2)
console.log("comp1 < comp2 is ", comp1 < comp2)
console.log("comp1 >= comp2 is ", comp1 >= comp2)
console.log("comp1 <= comp2 is ", comp1 <= comp2)
```

Input

```
comp1 == comp2 is  true
comp1 != comp2 is  false
comp1 === comp2 is  false
comp1 !== comp2 is  true
comp1 > comp2 is  false
comp1 < comp2 is  false
comp1 >= comp2 is  true
comp1 <= comp2 is  true
```

Output

4. Logical Operators:

&& and
|| or
! not

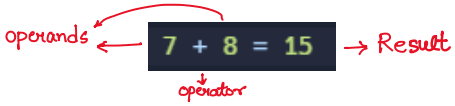
```
let x = 5;
let y = 6;
console.log(x>y && x==5)
console.log(x>y || x==5)
console.log(!false)
console.log(!true)
```

Input

```
false
true
true
false
```

Output

Apart from these we also have bitwise operators. Bitwise operators performs bit by bit operations on numbers.



Comments in Java Script:

Sometimes we want our programs to contains a text which is not executed by the JS engine
Such a text is called comment in Java Script
A comment in java script can be written as follows :

```
let a = 2; //this is  a single line comment
/* This is a
multiline comment */
```

Sometimes comments are used to prevent the execution of some lines of code.

Conditional Statements:

Sometimes we might have execute a block of code based of some condition
For example a prompt might ask for the age of the user and if… else statement.

1. if Statement
2. if… else statement
3. if… else if…else statement

- *Prompt takes input*
- *Alert give message in browser window*

```
let a = prompt("Hey what's your age?")
console.log(typeof a)
a = Number.parseInt(a) //Converting string into number
console.log(typeof a)
```

Input

```
Hey what's your age?> 45
string
number
```

Output

If statement

The if statements in Java Script looks like this :

```
//Syntax//
if (condition){
   //code//
}
```

The if statement evaluates the condition inside the () If the condition is evaluated to true the code inside the body of if is executed else the code is not executed

If-else statement

The if statement can have an optional else clause.
The syntax looks something like this:

```
//syntax//
if (condition) {
    //block of code if condition is true
}
else {
    //block of code if condition is false
}
```

If the condition is true, code inside if is executed else code inside if is executed else code inside else block is executed.

If-else if statement

Sometimes we might want to keep rechecking a set of conditions one until one matches. We use if else if for archiving this.
Syntax of if… else if looks like this

```
//syntax//
if  (age>0){
    console.log("A valid age");
}
else if (age>10 && age<15){
    console.log("not a kid");
}
else {
    console.log("Invalid Age")
}
```

EX:

```
let a = prompt("Hey what's your age?")
a = Number.parseInt(a)
if(a<0){
  alert("This is a invalid age")
}
else if(age<9){
  alert("You are a kid")
}
else if(age<18 && a>=9){
  alert("You are teen and cannot drive")
}
else{
  alert("You can drive")
}
```

```
Hey what's your age?> 56
You can drive
```

Switch case statement:

The `switch` statement is used to perform different actions based on different conditions.

```
switch(expression){
 case x:
    // code block
  break;
 case y:
    // code block
  break;
 default:
  // code block
}
```

Java Script ternary Operator

Evaluates a condition and executes a black of code based on the condition:

//syntax//
condition? exp1 : exp2

Example syntax of ternary operator looks like this:

(marks>10)? 'Yes' : 'No'

↳ if marks are greater than 10, you are passed else not

Chapter-2

1. Use logical operators to find whether the age of a person lies between 10 & 20?

2. Demonstrate the use of switch case statement in Java Script.

3. Write a Java Script program to find whether a number is Divisible by either 2 & 3.

4. Write a J.S program to find weather a number is divisible by either 2 or 3

5. Print "You can Drive" or "You cannot drive" based on age being greater than 18 using ternary operator.

```
JS ps_02.js > ...
1   // Chapter 2 Q1
2   const prompt = require("prompt-sync")({ sigint: true });
3
4   let age = prompt("Age")
5   if(age>=10 && age<=20){
6       console.log("Yes your age is ok")
7   }
8   else{
9       console.log("No you are not eligible")
10  }
11
```

```
Age13
Yes your age is ok
```

```
Age:34
No you are not eligible
```

```
let a = prompt("Enter the num")
if ((a%2==0) && (a%3==0)){
  console.log("Divisible by both 2 and 3")
}
else{
  console.log("No divisible by 2 and 3")
}
```

```
Enter the num> 12
Divisible by both 2 and 3
```

```
Enter the num> 14
No divisible by 2 and 3
```

```
let a = prompt("Enter the num")
if ((a % 2 == 0) || (a % 3 == 0)) {
  console.log("Divisible by both 2 or 3")
}
else {
  console.log("No divisible by 2 or 3")
}
```

```
Enter the num> 12
Divisible by both 2 or 3
```

```
Enter the num> 14
Divisible by both 2 or 3
```

```
Enter the num> 11
No divisible by 2 or 3
```

```
let a = prompt("Enter the age")
console.log("You can", a>18? 'drive': 'not drive')
```

```
>_ Console
Enter the age> 14
You can not drive
Hint: hit control+c anytime to enter REPL.
```

Loops & Functions

We use loops to perform repeated actions. For example - If you are designed a task of printing numbers from 1 to 100 , it will be very hectic to do it manually , Loops help us automatic such tasks.

Types of loops in Java Script

- **For loop** - loop a block of code no of times
- **For in loop** - loop through the keys of an object
- **For of loop** - loops through the value of an object
- **While loop** - loops a block based on a specific condition
- **Do while loop** - while loop variant which runs at least once

1. For loop

The syntax of a for loop looks something like this:

```
//syntax//
for (statement 1; statement 2; statement 3){
    //code to be executed
}
```

```
for (let i = 0; i < 5; i++) {
    console.log(i)
}
```

Input

```
0
1
2
3
4
```

Output

- Statement 1 is executed one time
- Statement 2 is the condition base on which the loops runs (loop body is executed)
- Statement 3 is executed every time the loop body is executed

Quick Quiz: Write a sample for loop of your choice

2. The for-in loop

The syntax of for in loop looks like this

```
//syntax//
for (key in object){
    //code to be executed
}
```

```
let obj = {
    harry: 90,
    shubh: 50,
    shivika: 56,
    ritika: 57,
}
for (let a in obj) {
    console.log("Marks of " + a + " are " + obj[a])
}
```

Input

```
Marks of harry are 90
Marks of shubh are 50
Marks of shivika are 56
Marks of ritika are 57
```

Output

Quick Quiz: Write a sample program demonstrating for-in loop of your choice

Note: for in loop also works with array which we will see later

3. The for of loop

The syntax of for of loop looks like this

```
//syntax//
for (variable of iterable){
    //code
}
```

for every iteration

Iterable means array or string so that for of loop works

4. The while loop

The syntax of while loop looks like this

```
//syntax//
for (condition){
    //code to be executed
}
```

```
let n = prompt("Enter the number ")
n = Number.parseInt(n)

let i = 0
while (i < n) {
  console.log(i)
  i++
}
```

Input

```
Enter the number > 5
0
1
2
3
4
```

Output

Note: If the condition never become false the loop will never end and this may crash the runtime.

5. Do While loop

The syntax of do while loop looks like this

```
//syntax//
do{
   //code to be executed
}
While(condition)
```

↳ Executed at least once

```
let n = prompt("Enter the number ")
n = Number.parseInt(n)

let i = 0
do {
  console.log(i)
  i++;
} while (i < n)
```

Input

Note: Do while loops executes at least one time

Examples

```
let sum = 0
let n = prompt("Enter the number ")
for (let i = 1; i <= n; i++) {
  sum += (i)
}
console.log("Sum of first "+ n + " natural numbers is "+ sum)
```

```
>_ Console
Enter the number > 5
Sum of first 5 natural numbers is 15
```

Sum of first n natural numbers using for loop

Functions in Java Script

A java Script function is a block of code designed to perform a particular task.
Syntax of a function looks something like this:-

```
//syntax//
function myfunc(){
   //code//
}
```

```
function binodFunc (parameter 1, parameter 2){
      //code
}
```

↳ Function with parameters

↳ Here the parameters behave as local variable

binodFunc (7,8) → Function invocation

Function invocation is a way to use the code inside the function

A function can also return a value. The value is "returned" back to the caller.

Another way to create & use the function

```
const sum = (a,b)=> {
      let c=a+b
      return C;
}
```

⇒ Return the sum

Example:

```
let y = sum ( 1,3 )
console.log(y)
```

→ Prints 4

```
function onePlusAvg(x, y) {
  return 1 + (x + y) / 2
}

let a = 1;
let b = 2;
let c = 3;

console.log("One plus Average of a and b is ", onePlusAvg(a,b))
console.log("One plus Average of a and b is ", onePlusAvg(b,c))
console.log("One plus Average of a and b is ", onePlusAvg(a,c))
```

Input

```
One plus Average of a and b is  2.5
One plus Average of a and b is  3.5
One plus Average of a and b is  3
```

Output

PS_03

20 July 2023 21:23

Chapter-3

1. Write a program to print the marks of a student in an object for loop.

obj = {harry: 98, rohan: 70, aakash: 7}

2. Write the program in Q1 using for in loop.

3. Write a program to print "try again" until the user enters the correct numbers.

4. Write a function to find mean of 5 numbers.

```
JS ps_03.js > ...
1  const marks = {
2      avinash: 74,
3      honey : 56,
4      krypton : 67
5    }
6    console.log(marks)
7  for (let i=0;i<Object.keys(marks).length;i++){
8      console.log("The marks of " + Object.keys(marks)[i] + " are " + marks[Object.keys(marks)[i]])
9  }
```

```
{ avinash: 74, honey: 56, krypton: 67 }
The marks of avinash are 74
The marks of honey are 56
The marks of krypton are 67
```

```
JS ps_03.js > ...
1  const marks = {
2      avinash: 73,
3      honey : 57,
4      krypton : 69
5    }
6    console.log(marks)
7  for( let key in marks ){
8      console.log("The marks of " + key + " are " + marks[key] )
9  }
```

```
{ avinash: 73, honey: 57, krypton: 69 }
The marks of avinash are 73
The marks of honey are 57
The marks of krypton are 69
```

```
1  let num = 5
2  let i = prompt("Enter a number")
3  while(i!=num){
4    i = prompt("Enter a number")
5  }
```

```
Enter a number> 3
Enter a number> 4
Enter a number> 5
```

```
1  function mean(a,b,c,d,e){
2    return (a+b+c+d+e)/5
3  }
4
5  let a = parseInt(prompt("Enter the num1"))
6  let b = parseInt(prompt("Enter the num2"))
7  let c = parseInt(prompt("Enter the num3"))
8  let d = parseInt(prompt("Enter the num4"))
9  let e = parseInt(prompt("Enter the num5"))
10
11 console.log(mean(a,b,c,d,e))
```

Input

```
Enter the num1> 10
Enter the num2> 3
Enter the num3> 5
Enter the num4> 2
Enter the num5> 20
8
```

Output

Strings

Strings are used to store and manipulate text. Strings can
Be created using the following syntax.

```
Let name = "Harry"
name.length
```

→ Create a string

↳ This property prints length of the string

Strings can also be created using single quotes

```
let name =‘Harry’
```

Template literals.

Template literals use backtics. Instead of quotes to define a string

```
let name = harry
```

With template literals, it is possible to use both single as well as double quotes inside a string
Let sentence = 'The name is harry.

```
let sentence = `The name "is" Harry's`
```

↳ single quote

backtic

double quote

We can insert variables directly in template literal. This is culled string interpolation

```
let a= `this is ${name}`
```

→ Prints 'This is a harry'

name is a variable

Escape sequence characters:

If you try to print the following string, Java script will misunderstand it.

```
let name = 'Adam D' Angelo'
```

We can use single quote escape sequence to solve the problem

```
let name = 'Adam D\'Angelo'
```

Similarly we can use "inside a string with double quotes
Others escape sequence characters are as follows.

\n → new line
\t→ Tab
\r → carriage return

```
let name = "Harry"
console.log(name[0]) //Excess the character of the string
console.log(name[1])
let friend = 'Prakash'
console.log(friend)
let myfriend = 'Shubh' //Dont do this
console.log(myfriend)

// Template Literals
let boy1 = "Promod"
let boy2 = "Nikhil"
//Nikhil is a friend of promod
let sentence = `${boy2} is a friend of ${boy1}`
// This is also called string interpolation
console.log(sentence)

// Escape sequence character
let fruit = 'Bana\'na'
console.log(fruit)
```

```
H
a
Prakash
Shubh
Nikhil is a friend of Promod
Bana'na
```

String properties and methods.

1. let name = "Harry"
   name.length → prints 5

2. let name= "Harry"
   name.toUpperCase() → prints HARRY

3. let name="harry"
   name.toLowerCase() prints harry

4. let name= "Harry" (0 1 2 3 4) prints rr
   name.slice(2,4) → (from 2 to 4, 4 not included)

5. let name = "Harry"
   name.slice(2) → prints rry
   (from 2 to end)

6. let name="harry bhai"
   let newName = name.Replace("Bhai", "Bhau")

7. let -name1 = "Harry"
   let name2 = "Naman"
   let name 3 = name1.concat(name2, "yes")
   ↳ we can even use + operator

8. let name="harry"
   let newName= name.trim()
   ↳ Removes whitespaces.

```
let name = "  harry bhai  "
console.log(name.length)
console.log(name[0])
console.log(name.toUpperCase())
console.log(name.toLowerCase())
console.log(name.slice(2,4))
console.log(name.slice(2))
console.log(name.replace("bhai","bhau"))
console.log(name.trim())
```

```
14
  HARRY BHAI
  harry bhai
ha
harry bhai
  harry bhau
harry bhai
```

Strings are immutable. In order to access the character at an index we
We use the following syntax

let name="Harry"
name [0] → prints H
name [1] → prints a

```
let fr = "Raman" + "Shivika" + "Harry"
console.log(fr[0])
console.log(fr[1])
console.log(fr[2])
console.log(fr[3])
console.log(fr)
fr[4] = "o" //This is not possible
```

```
R
a
m
a
RamanShivikaHarry
```

Quick Quiz:

Includes

Chapter-4

1. What will the following print in Java Script?

```
console.log("her\"s".length)
```

2. Explore the includes, starts with & end with functions to lowercase.

3. Write a program to convert a given string to lowercase.

4. Extract the amount out of this string
"Please give Rs 1000"

5. Try to change 4th charater of a given story. Were you able to do it.

↓
Cannot do
(Strings are Immutable)

```
1  console.log("her\"s".length)

>_ Console  ×  +

5
```

```
let str = "Please give Rs 1000"
let amount = Number.parseInt(str.slice(15))
console.log(amount)
console.log(typeof amount)
```

```
1000
number
```

Arrays

Arrays are variables which com hold More than one value.

const fruits = ["banana", "Apple ", "grapes"]

const a1 = [7, "Harry", false]

↳ can be different types

Accessing values

let numbers = [1,2,7,9]

number [0]→ 1
number [1 → 2

```
let marks_class_12 = [91, 82, 63, 84, null, false, "Not present"]
//Accessing values of Array
console.log(marks_class_12)
console.log(marks_class_12[0])
console.log(marks_class_12[1])
console.log(marks_class_12[2])
console.log(marks_class_12[3])
console.log(marks_class_12[4])
console.log(marks_class_12[5])
console.log(marks_class_12[6])
//Finding length of Array
console.log("The length of marks_class_12 is", marks_class_12.length)
//Changing the values
marks_class_12[2] = "Avinash"
console.log(marks_class_12)
```

Input

```
[ 91, 82, 63, 84, null, false, 'Not present' ]
91
82
63
84
null
false
Not present
The length of marks_class_12 is 7
[ 91, 82, 'Avinash', 84, null, false, 'Not present' ]
```

Output

Finding the length

let numbers = [1,7,9,21]
0 1 2 3
number [0] →1
number.length →4

Changing the values

let number = [7, 2,40,9]

number [2] = 8 "numbers" now becomes [7, 2, 8, 9]
Arrays are mutable

In Java script arrays are objects. The type of operator on arrays return object

const n = [1 , 7 , 9]
typeof n → Returns object

Arrays can hold many values under a single name.

Array methods

There are some important array methods in Java script. Some of them are as :

```
let num = [1,2,3,34,4]
let b = num.toString()
console.log(b)
```

```
1,2,3,34,4
```

1- To strings() → converts an array to a string of comma separated values.

let n= [1,7,9]
n.tostring() ⟶ 1,7,9

2. join() joins all the array to a separator

```
let num = [1,2,3,34,4]
let b = num.join("")//Joins as string
console.log(b)
```

```
123344
```

Let n= [7,9,13]
n.join ("-") ⟶ 7-9-13

3. pop() removes last element from the array

```
let num = [1,2,3,34,4]
let b = num.pop()
console.log(num , b)
[ 1, 2, 3, 34 ] 4
```

Let n = [1,2,4]
n.pop() → updates the original array
returns the popped value

4. push() Adds a new element at the end of the array

```
let num = [1,2,3,34,4]
let b = num.push(56)
console.log(num , b)
//returns the new array length
[ 1, 2, 3, 34, 4, 56 ] 6
```

Let a = [7,1,2,8]
a.push (9) → modifies the original array
↳ returns the new array length

5. shift() Removes first element and returns it.

6. unshift() Adds element to the beginning Returns new array length

7. delete Array elements can be deleted using the delete operator
→ delete is an operator

8. concat() Used to join arrays to the given array

Let a1 = [1,2,3]
Let a2 = [4,5,6]
Let a3 = [9,8,7]

a1.concat(a2,a3) ⟶ Returns [1,2,3,4,5,6,9,8,7]
↓
Returns a new Array
Does not change existing arrays

9. Sort() sort method is used to sort an array alphabetically.

Let a = [7,9,8]
a.sort() → a change to [7,8,9]
[modifies the original array]

Sort() tales an optional compare function. If this is function is provided as the first argument the sort()
function will consider these values (the values returned from the compare function) as the basis of sorting

10. Splice() splice can be used to add new items to an array

Const numbers = [1,2,3,4,5]
number,splice(2,1,23,24)

Returns deleted items, modifies the Array

position to add
No. of elements to be removed
Elements to be added

11.slice () → slices out a piece from an array.

It creates a new array

const num = [1,2,3,4]

num.slice(2) → [3,4]

num.slice(1,3) → [2,3]

12. Reverse() Reverse the elements in the source array

```
// Removes first element and returns it
let num = [1,2,3,34,4]
let a = num.shift()
console.log(num , a)

//Adds new element to array at start
let b = num.unshift(12)
console.log(num, b)
```

```
[ 2, 3, 34, 4 ] 1
[ 12, 2, 3, 34, 4 ] 5
```

```

//delete is operator not method
delete num[0]
console.log(num)
console.log(num.length)
//Do not change the length of array

// Joins the arrays
let a1 = [1,2,3]
let a2 = [4,5,6]
let a3 = [9,8,7]
let c = a1.concat(a2,a3)
console.log(c)
```

```
[ <1 empty item>, 2, 3, 34, 4 ]
5
[
  1, 2, 3, 4, 5,
  6, 9, 8, 7
]
```

```

//Sort method sorts alphabetically
d = num.sort()
console.log(d)

//for sorting in ascending order
let compare = (a,b) => {
  return b - a
}
e = num.sort(compare)
console.log(e)
```

```
[ 2, 3, 34, 4, <1 empty item> ]
[ 34, 4, 3, 2, <1 empty item> ]
```

```
//Splice and Slice
let num1 = [551 ,345 ,42 , 66 ,4 ,5 ,229 ,65]
num1.splice(2,3,34,45,46)
console.log(num1)

let newNum = num1.slice(5)
console.log(newNum)
```

```
[
  551, 345,  34, 45,
   46,   5, 229, 65
]
[ 5, 229, 65 ]
```

Looping through Arrays

Arrays can be looped through using the classical JavaScript for loop or through some other methods discussed below

1. forEach loop calls a function, once for each array element

```
Const a = [1,2,3]
a.forEach ((value, index, array) => {
        //function , logic
        })
```

2. map() creates a new array by performing some operation on each array element.

```
Cons a = [1,2,3]
a.map((value, index, array) => {
        return value*value;
        })
```

3. filter () Filters an array with values that passes a test. Creates a new array

```
Const a = [1,2,3,4,5]
a.filter(greater_than_5)
```

4. reduce method Reduce an array to a single value

```
Const n = [1,8,7,11]
Let sum = number.reduce(add)
```

1 + 8 + 7 + 11 ↳ A function

5. Array.from Used to create an array from any other object

```
Array from ("Harry")
```

6. for-of For-of loop can be used to get the values from an array

7. for-in For-in loop can be used to get the keys from an array.

```js
let num = [3, 5, 1, 2, 4]

// for(let i = 0; i<num.length; i++){
//   console.log(num[i])
// }

num.forEach((main)=>{
  console.log(main*main)
})

// Array.from
let name = "Harry is"
let arr = Array.from(name)
console.log(arr)

// for...of(gives values)
for(let item of num){
  console.log(item)
}

// for...in(gives keys)
for(let i in num){
  console.log(i)
  console.log(num[i])
}
```

```
9
25
1
4
16
[
  'H', 'a', 'r',
  'r', 'y', ' ',
  'i', 's'
]
3
5
1
2
4
0
3
1
5
2
1
3
2
4
4
```

```js
let num = [43, 23, 32]
// Creates a new array while mapping
let a = num.map((value,index,array)=>{//These are parameters
  console.log(value,index,array)
  return value + 1
})
console.log(a)
// Difference between for each and map
// forEach is used to perform task without making array of the elements
// map creates a array of that array after performing funtion on it

let arr = [23,5,45,67,7,43,9,]
let a2 = arr.filter((value)=>{
  return value<10
})
console.log(a2)

let arr2 = [76,45,9,2,45]
let b = arr2.reduce((h1,h1){
  return h1 + h2
})
console.log(b)
```

```
43 0 [ 43, 23, 32 ]
23 1 [ 43, 23, 32 ]
32 2 [ 43, 23, 32 ]
[ 44, 24, 33 ]
[ 5, 7, 9 ]
```

Arr2 might not work

Array methods

Name	Description
at()	Returns an indexed element of an array
concat()	Joins arrays and returns an array with the joined arrays
constructor	Returns the function that created the Array object's prototype
copyWithin()	Copies array elements within the array, to and from specified positions
entries()	Returns a key/value pair Array Iteration Object
every()	Checks if every element in an array pass a test
fill()	Fill the elements in an array with a static value
filter()	Creates a new array with every element in an array that pass a test
find()	Returns the value of the first element in an array that pass a test
findIndex()	Returns the index of the first element in an array that pass a test
flat()	Concatenates sub-array elements
flatMap()	Maps all array elements and creates a new flat array
forEach()	Calls a function for each array element
from()	Creates an array from an object
includes()	Check if an array contains the specified element
indexOf()	Search the array for an element and returns its position
isArray()	Checks whether an object is an array
join()	Joins all elements of an array into a string
keys()	Returns a Array Iteration Object, containing the keys of the original array
lastIndexOf()	Search the array for an element, starting at the end, and returns its position
length	Sets or returns the number of elements in an array
map()	Creates a new array with the result of calling a function for each array element
pop()	Removes the last element of an array, and returns that element
prototype	Allows you to add properties and methods to an Array object
push()	Adds new elements to the end of an array, and returns the new length
reduce()	Reduce the values of an array to a single value (going left-to-right)
reduceRight()	Reduce the values of an array to a single value (going right-to-left)
reverse()	Reverses the order of the elements in an array
shift()	Removes the first element of an array, and returns that element
slice()	Selects a part of an array, and returns the new array
some()	Checks if any of the elements in an array pass a test
sort()	Sorts the elements of an array
splice()	Adds/Removes elements from an array
toString()	Converts an array to a string, and returns the result
unshift()	Adds new elements to the beginning of an array, and returns the new length
valueOf()	Returns the primitive value of an array

Chapter-5

1. Create an array of number and take input from the user to add number to this array.

2. Keep adding numbers to the array in (1) until 0 is added to the array.

3. Filter for numbers divisible by 10 from a given array

4. Create an array of square of given numbers.

5. Use reduce to calculate factorial of a given number from an array of first n natural number. (n being the number whose factorial needs to be calculated).

Java Script in the browser

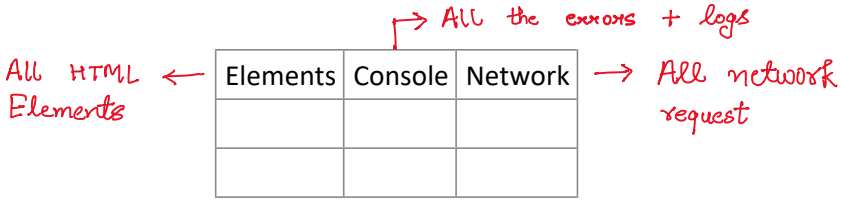
Java Script was initially created to make web pages alive. JS can be written in a web pages HTML to make it interactive

The browser has an embedded engine called the protect the user's safety. For example a webpage on https://google.com cannot access https: and steak information from there

Developer tools

Every browser has some developer tools which makes a developer's life a lot easier.

F_{12} on keyboard opens dev tools



We can also write Java Script commands in the console

The script tag

The script tag is used to insert Java Script into an HTML page

The script tag is used to insert external or internal scripts

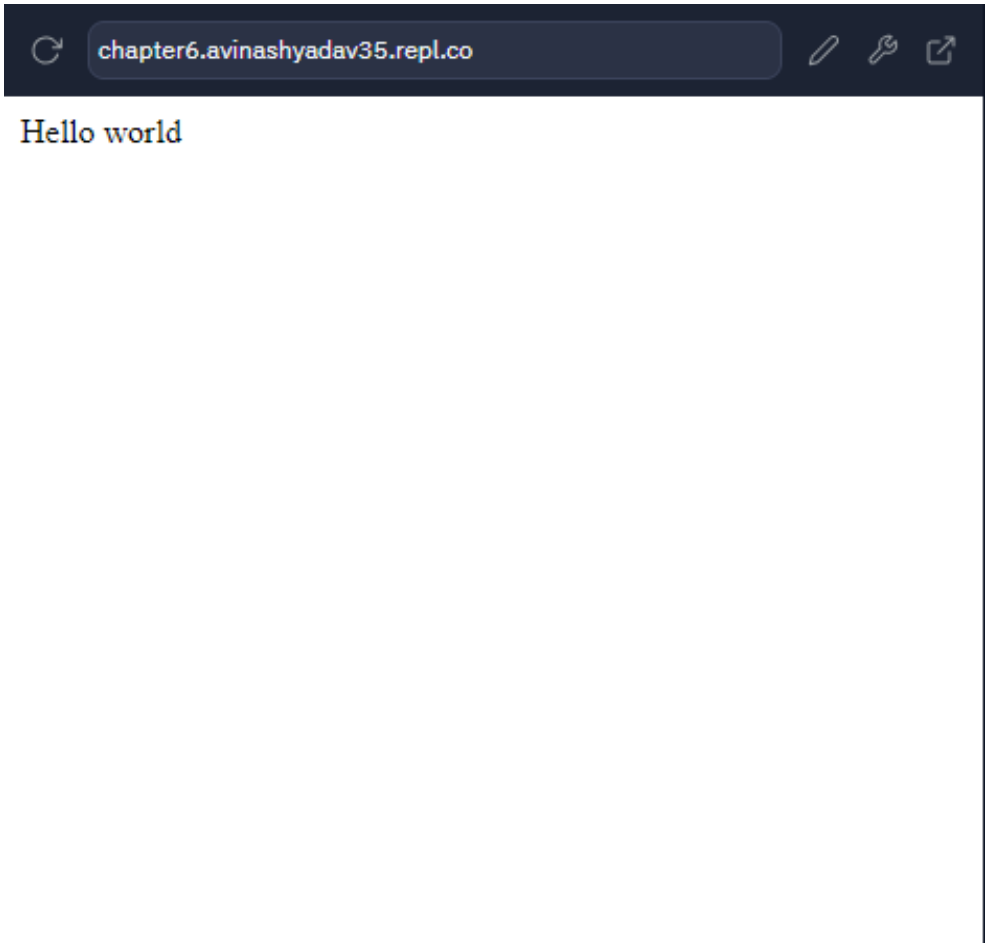
```
<script>
  alert("Hello")
</script>
//or…
<script src="Js/thisone.js"></script>
```

The benefit of a separate java script file is that the browser will download it and store it in its cache

index.html

```
1  <!DOCTYPE html>
2  <html>
3
4  <head>
5    <meta charset="utf-8">
6    <meta name="viewport" content="width=device-width">
7    <title>replit</title>
8    <link href="style.css" rel="stylesheet" type="text/css" />
9  </head>
10
11 <body>
12   Hello world
13   <script src="script.js"></script>
14
15   <script src="https://replit.com/public/js/replit-badge-v2.js"
   theme="dark" position="bottom-right"></script>
16 </body>
17
18 </html>
```

Ln 14, Col 1 History



Inline script or url script

Console Object Methods

The console object has several methods log being one of them. Some of them are as follows:

assert() used to asset a condition
clear() clears the console
log() outputs a message to the console
table() Displays a tabular data
warn() Used for warnings
time() & timeEnd()
error() Used for errors
info() Used for special information

```
> console.log("Hi Avi")
  Hi Avi
< undefined
> console.error("Hey this is an error")
✖ ▶ Hey this is an error
< undefined

> console.assert(5>52)
✖ ▶ Assertion failed: console.assert
< undefined
> console.assert(444>56)
< undefined
```

```
Console was cleared                                    VM1048:1
undefined
obj = {a: 1, b: 2, c:3}
▶ {a: 1, b: 2, c: 3}
console.table(obj)
                                                       VM1163:1
```

(index)	Value
a	1
b	2
c	3

▶ Object

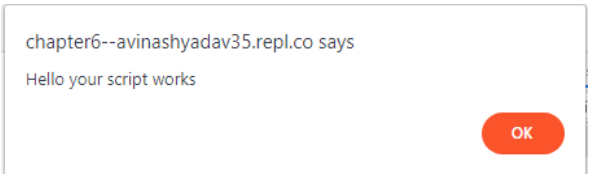
```
> console.time("Avi")
< undefined
> console.timeEnd("Avi")
  Avi: 17056.02294921875 ms
< undefined
>
```

You will naturally remember some or all of these with time
Comprehensive list can be looked up on MDN

Interaction: alert, prompt and confirm

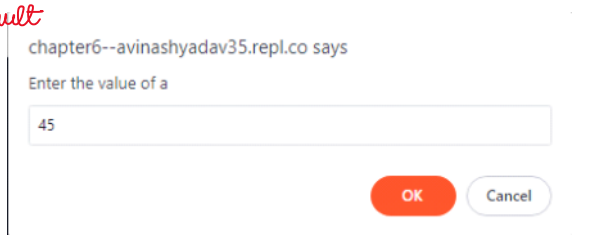
alert : Used to invoke a mini window with a message
alert("hello")

```
1  alert("Hello your script works")
```



prompt: Used to take user input as string
inp = prompt ("Hi", "No")
↳ optional default value

```
2  let a = prompt("Enter the value of a")
3  a = Number.parseInt(a)
4  document.write(a)
```



confirm: shows a message and waits for the user to press ok or cancel. Returns true for ok and false for cancel

```
let a = prompt("Enter the value of a")
a = Number.parseInt(a)
alert("You entered a of type " + (typeof a))
let write = confirm("Do you want to write it to the page")
if (write) {
  document.write(a)
}
else {
  document.write("Please allow me to write")
}
```

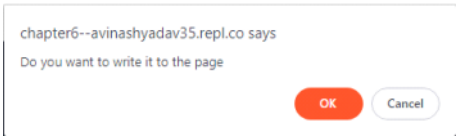
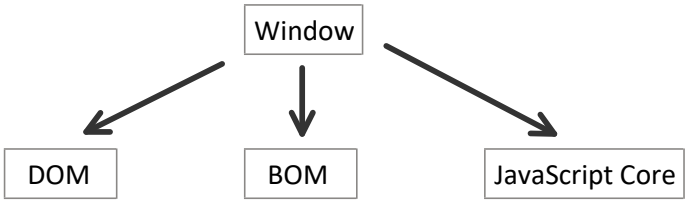


Fig: It ask for confirmation

The exact location & look is determined by the browser which is a limitation

Window object, BOM & DOM

We have the following when JavaScript runs in a browser



Window object represents browser window and provides methods to control it. It is a global object

Document Object Model (DOM)

DOM represent the page content as HTML

document.body → Page body as J.S object
document.body.style.background="green"
↳ changes page background to green

Browser Object Model (BOM)

The Browser Object Model (BOM) represent additional object provided by the browser (host environment for working with everything except the document.

The function alert/confirm/prompt are also a part of the BOM

location href= "https://codewithharry.com"
↳ Redirect to another URL

```
▶ assert: ƒ assert()
▶ clear: ƒ clear()
▶ context: ƒ context()
▶ count: ƒ count()
▶ countReset: ƒ countReset()
▶ createTask: ƒ createTask()
▶ debug: ƒ debug()
▶ dir: ƒ dir()
▶ dirxml: ƒ dirxml()
▶ error: ƒ error()
▶ group: ƒ group()
▶ groupCollapsed: ƒ groupCollapsed()
▶ groupEnd: ƒ groupEnd()
▶ info: ƒ info()
▶ log: ƒ log()
▶ memory: MemoryInfo {totalJSHeapSize:
▶ profile: ƒ profile()
▶ profileEnd: ƒ profileEnd()
▶ table: ƒ table()
▶ time: ƒ time()
▶ timeEnd: ƒ timeEnd()
▶ timeLog: ƒ timeLog()
▶ timeStamp: ƒ timeStamp()
▶ trace: ƒ trace()
▶ warn: ƒ warn()
```

PS_06

Saturday, 22 July 2023 03:30

Chapter-6

1. Write a program using prompt function to take input of age as a value from the user and use alert to tell him if he can drive!

2. In Q1 use confirm to ask the user if he wants to see the prompt again.

3. In the previous question, use console.error to log the error if the age entered is negative.

4. Write a program to change the url to google.com (Redirection) if user enters a number greater than 4.

5. Change the background of the page to yellow, red or any other color based on user input through prompt.

Walking the DOM

DOM tree refers to the HTML page where all the nodes are objects.
There can be 3 types of nodes in the DOM tree:

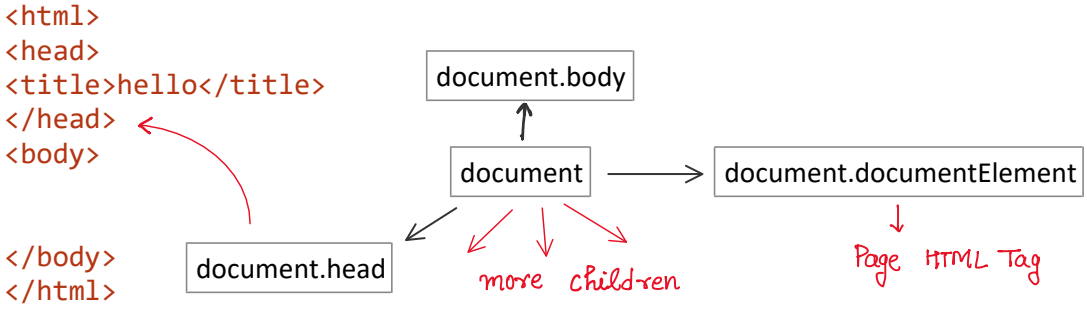
1. text nodes
2. element nodes
3. comment nodes

In an HTML page <html> is at the root and <head> and <body> are its children , etc.
An text node is always a leaf of the tree

Auto Correction

If an erroneous HTML is encountered by the browser, it tends to correct it for example, if we put something after the body. Another example is <table> tag which must contain <tbody>

Walking the DOM



Note : document.body can sometimes be null if the JavaScript is written before the body tag.

Children of an element

Direct as well as deeply nested elements of an element are called its children

Child nodes Elements that are direct children for example head & body are children of <html>

Descendant nodes All nested elements, children, their, children and so on…

firstchild, lastchild & childNodes

element.firstchild	first child element
element.lastChild	last child element
element.childNodes	All child nodes

```
1  console.log(document.body.firstChild)
2  console.log(document.body.lastChild)
3  console.log(document.body.childNodes)
```

```
▶ #text                                          script.js:1
                                                 script.js:2
<script src="script.js"></script>
                                                 script.js:3
▼ NodeList(4) [text, div, text, script]
    ▶ 0: text
    ▶ 1: div
    ▶ 2: text
    ▶ 3: script
    ▶ 4: text
    ▶ 5: script
    ▶ 6: a
    ▶ 7: text
    ▶ 8: iframe
    ▶ 9: div#GOOGLE_INPUT_CHEXT_FLAG
      length: 10
    ▶ [[Prototype]]: NodeList
```

Following is always true:

elem.childNodes[0] === elem.firstChild

elem.childNodes[elem.childNodes.length -1] === elem.lostChild

There is also a method elem.hasChildNodes() to check whether there are any child nodes.

Note: childNodes looks like ab array . But it's not actually an array but a collection. We can use Array.from(collection) to convert it into an Array.

Array methods won't work untill we convert it into array.

Notes on DOM collections

- They are read-only → Cannot edit document
- They are live elem.childNodes variable (reference) will automatically update if childNodes of elem is changed.
- They are iterable using for … of loop

$0 and $1 ke baare me

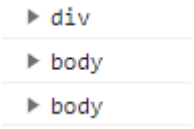
Siblings and the parent

Siblings are nodes that are children of the same parent.

- For example: <head> and <body> are siblings. Siblings have same parent. In the above example its html
- <body> is said to be the next or "right" sibling of <head>, <head> is said to be the previous or "left" sibling of <body>
- The next sibling is in nextSibling property ,and the previous one in previousSibling.
- The parent is available as parentNode.

alert(document.documentElement.parentNode); //document
alert(document.documentElement.parentNode); //null

```
console.log(document.body.firstChild)
a = document.body.firstChild
console.log(a.parentNode)
console.log(a.parentElement)
// Difference is Node returns text ,comments, elements etc.
// Whereas Element returns only element tags
```




```
<!-- First child of body is div -->
<body><div>
    <div class="first">first</div>
    <div class="second">second</div>
</div>
```

Element only Navigation

Sometimes, we don't want text or comment nodes. Some links only tale Elements nodes into account. For example

document.previousElementSibling
document.nextElementSibling
documet.firstElementChild
document.lastElementChild

```
<nav>
  <ul>
    <li>Home</li>
    <li>About Me</li>
    <li>Hire Me</li>
  </ul>
</nav>
<script src="script.js"></script>
```

```
const bgred = ()=>{
  document.body.firstElementChild.style.background = "red"
}//Run this function in console to get bg red

let b = document.body
console.log("First child of b is:", b.firstChild)
console.log("First element child of b is: ", b.firstElementChild)
```

Table links

Certain DOM elements may provide additional properties specific to their type for convenience.
Table element supports the following properties:

table.rows
table.caption
table.tHead
table.tFoot
table.tBodies
tbody.rows

tr.cells
tr.sectionRowIndex
tr.rowIndex

td.cellIndex

```
<body>
  <table class="table">
    <thead>
      <tr>
        <th scope="col">#</th>
        <th scope="col">First</th>
        <th scope="col">Last</th>
        <th scope="col">Handle</th>
      </tr>
    </thead>
    <tbody>
      <tr>
        <th scope="row">1</th>
        <td>Mark</td>
        <td>Otto</td>
        <td>@mdo</td>
      </tr>
      <tr>
        <th scope="row">2</th>
        <td>Jacob</td>
        <td>Thornton</td>
        <td>@fat</td>
      </tr>
      <tr>
        <th scope="row">3</th>
        <td colspan="2">Larry the Bird</td>
        <td>@twitter</td>
      </tr>
    </tbody>
  </table>
```

#	First	Last	Handle
1	Mark	Otto	@mdo
2	Jacob	Thornton	@fat
3	Larry the Bird		@twitter

```
let t = document.body.firstElementChild.firstElementChild
console.log(t)
console.log(t.rows)
console.log(t.caption)
console.log(t.tHead.firstChildElements)
console.log(t.tFoot)
console.log(t.Bodies)

```

```
▶ table.table
▶ HTMLCollection(4) [tr, tr, tr, tr]
null
undefined
null
undefined
```

Quick Quiz: Print typeof document and typeof window int the console & see what it prints.

Searching the DOM

DOM navigation properties are helpful when the elements are close to each other. If they are not close to each other, we have some more methods to search the DOM

→document.getElementById
This method is used to get the element with a given "id" attribute

```
Let span = document.getElementById('span')
    span.style.color = "red"
```

→document.querySelectorAll
Return all elements inside an element matching the given CSS selector

→document.getElementsByTagName
Returns elements with the given tag name

→document.getElementsByClassName
Returns elements that have the given CSS class.

→document. getElementsByName
Searches elements by the name attribute.

matches, closest & contain methods.

There are three important methods to search the dot.

1. elem.matches(CSS)→ To check if elementmatches thegiven CSS selector
2. elem.closet (CSS) → To look for the nearest ancestor that matches the given CSS selector. The element itself is also checked.
3. elem.contains (elem) → returns true if elemA (a descendant of elemA) or when elem A == elem B

PS-07

Saturday, 22 July 2023 03:48

Chapter-7

1. Create a navbar and change the color of its first elements to red.

2. Create a table without tbody Now use "View page source" button to check whether it has a tbody or not.

3. Create an elements with 3 children. Now change the color of first and last element to green

4. Write a Java Script code to change background of all <li> tags to cyan.

5. Which of the following is used to look for the farthest ancestor that matches a given CSS selector

a) matches b) closet c) contains d) none of these.

Events & other DOM

console.dir function

console.log shows the element dom tree
console.dir shows the element as an object with its properties.

```
> console.log(document.getElementsByTagName('span')[0])
  <span>Hey I am span</span>
< undefined
> console.dir(document.getElementsByTagName('span')[0])
  ▶ span
< undefined
```

console.dir gives directory of element

tag name/ node name

Used to read tag name of an element
Tag name → only exists for element nodes
Node name → defined for any node (text, comment etc.)

```
> document.body.firstChild.nodeName
< '#text'
> document.body.firstElementChild.nodeName
< 'SPAN'
```

example

Inner HTML and outer HTML

The inner HTML property allows to get the HTML inside the element as a string.

The outer HTML property contains the full HTML.
Inner HTML + the element itself.
Inner HTML is valid only for element modes. For other mode types we can use no de value or data.

```
> first.innerHTML
< 'Hey I am span'
> first.innerHTML = "<i> hey I am italic </i>"
< '<i> hey I am italic </i>'
> first.outerHTML
< '<span id="first"><i> hey I am italic </i></span>'
> first.outerHTML = "<div>Hey</div>"
< '<div>Hey</div>'
```

works only when first is id
Doesn't works of first is class

Text context

Provides access to the text inside the element: Only text , minus all tags

```
> document.body.firstChild
<   " Hello World "
> document.body.firstChild.data
< '\n  Hello World\n  '
> document.body.firstChild.nodeValue
< '\n  Hello World\n  '
```

Screen clipping taken: 16-08-2023 22:39

The hidden property

The hidden attribute and the DOM property specifies whether the element is visible or not.

```
<div hidden> I am hidden </div>
<div id = "element"> I can be hidden </div>
<script>
  element.hidden = true;
</script>
```

Attribute methods

1. elem.hasAttribute(name) Method to check for existence of an attribute
2. elem.getAttribute(name) Method used to get the value of and attribute
3. elem.setAttribute(name) Method used to set the value of an attribute
4. elem.removeAttribute(name) Method to remove the attribute from elem
5. elem.attributes Method to get the collection of all attributes

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-wi
    <title>Document</title>
    <link rel="stylesheet" href="index.css">
</head>
<body>
    <div id="first" class="hey">
        Hey I am First Container
    </div>
    Hello World
</body>
<script>
    let a = first.getAttribute("class")
    console.log(a)
    console.log(first.  any  tribute("class"))
    console.log(first.hasAttribute("style"))
    first.setAttribute("hidden","true") //This hides the element
    first.setAttribute("class","true sachin") //Adds classes named true and sachin in id (first)
    first.removeAttribute("class") //Remove class ans its attributes in id (first)
</script>
```

```
hey
true
false
```

Data attributes

We can always create custom attributes but the ones starting with data are reserved for programmers use. They are available in a property named dataset.

If an element has an attribute named "data-one" it's available as element.dataset.one

```
console.log(first.dataset)
console.log(first.dataset.game)
console.log(first.dataset.player)
```

```
▶ DOMStringMap {game: 'forza', player: 'Avinash'}
forza
Avinash
```

Screen clipping taken: 02-01-2024 10:16

Insertion methods

We look at some ways to insert elements in the DOM. Here is another way :

```
Let div = document.createElement('div') //create
div.className = "alert" //set class
div.innerHTML = "<span>hello</span>
document.body.append(div)
```

Here are some more insertion methods:

node.append(e) append at the end of node.
node.prepend(e) Insert at the beginning of node.
node.before(e) Insert before node.
node.after(e) Insert after node.
node.replaceWith(e) replaces node with the given node.

Quick Quiz: Try out all these methods with your own webpage.

insertAdjacentHTML/Text/Element

This method is used to insert HTML. The first parameter is a Code word, specifying where to insert. Must be one of the following:

Beforebegin" - Inset HTML immediately before element
Afterbegin" - Insert HTML into element at the beginning
Beforeend" - Insert HTML into element at the end
Afterend - Insert HTML immediately after element

The second parameter is an HTML string

Example:

```
<div id="div"></div>
<script>
	div.insertAdjacentHTML('beforebegin','<p>Hello</p>');
	div.insertAdjacentHTML('afterend',<p>Bye</p>);
</script>
```

The output would be:

Chapter - 8

1. Write a program to show different alerts when different button are clicked.

2. Create a website which is capable of storage bookmarks of your favourite websites using href.

3. Repeat Q-2 using event listeners.

4. Write a javascript program to keep fetching contents of a website (every 5 seconds).

5. Create a glowing bulb effect using classlist toggle method in Java Script.

Asynchronous actions are the actions that we inititate now and they finish later. Eg setTimeout Synchronous actions are the actions that initiate and finish ine-by-one

Call back functions
Acallback function is a function passed into another function as an argument, which is then involved inside the outer fucntion to complete an action.

Here is an example of a callback:

```
Function laoadScript (src, callback){
        Let script = document.createElement('script')
        Script.src =src
        Script.onload = ()=> callback(script)
        Document.head.append(script)
}
```

Now we can do something like this:

```
loadScript('https://cdn.harry.com',(script)=>{
        Alert('script is loaded')
        Alert('script.src')
        });
```

This is called 'Calledback-based' style of async programming. A function that does something asynchronously should provide a callback argument where we put the function to run after its complete

Handling errors
We can handle callback error by supplying error argument like this:

```
function loadScript(src,callback){
……
……
        script.onload =() => callback(null, Script);
        script.onerror =()=> callback(new Error('failed'));
……
}
```

Then inside of loadScript call:

```
    loadScript('cdn/harry', function(error,script){
    …
    If (error){
    //handle error
    }
    Else{
    //script loaded
    }
});
```

Pyramid of Doom

When we have callback inside callback, the code gets difficult to manage

```
loadScript((…){
        loadScript..
                loadScript…
                        loadScript…
```

As calls become more nested, the code becomes deeper and increasingly more difficult to manage, especially if we have real code instead of…

This is sometimes called "callback hell" or "pyramid of doom"

The "pyramid" pf these calls grows to words the right with every asynchronous action. Soon it spirals out of control. So this way of coding isn't very good!

Introduction to promises

The solution to the callback hell is promises. A promise is a "promise of code execution". The code either executes or fail, in both the cases the subscriber will be notified.

The syntax of a Promise looks like this:

```
Let promise = new Promise(function(resolve, reject){
        //executor
});
```

Resolve and reject are two callback provided by javaScript itself. They are called like this:

Resolve(value) If the job is finished successfully
Reject(error) If the job fails

The promise object returned by the new Promise constructor has these properties

1. State: Initially pending, then changes to either "fulfilled when resolve is called or "rejected" when reject is called
2. Result : Initially undefined, then changes to value if resolved pr error when rejected

Consumers: then & catch
The consuming code can receive the final result of a promise through then & catch

The most fundamental one is then

```
        promise.then(function(result){*/handle*/}
                        Function(error){/*handleerror*/}
                        );
```

If we are interested only in succesful complitions, we can provide only one function argument to then();

```
Let  promise = new Promise(resolve=>{
        setTimeout(()=> resolve ("done"),1000);
        });
promise.then(alert);
```

Promise.finally = (()=>{}) is used to perform general cleanups

Quick Quiz: Rewrite the loadScript function we wrote in the beginning og=f this chapter of the chapter using promises.

Promises Chaining
We can chain promises and make then pass the resolved values to one another like this

```
a.then(function(result)=> {
        Alert(result);return 2;
        }).then
        …
```

The idea is to to pass the result through the claim of then handlers.
Here is the flow of execution

1. The initial promise reslove in second(Assumption)
2. The next .then() handlers is then called, which returns a new promise(resolved with 2 value)
3. The next.then() gets the result of previous one and this keeps on going

Every call to then() return a new promise whose value is passed to the next one and so on. We can even create promises inside.then()

Attaching multiple handlers
We can attach multiple handlers to one promise. They don't pass the result to each other; instead they process it independently.

Let p is a promise

```
p.then(handlers1)
p.then(handlers2)
p.then(handlers3)
```

Promise API
There are 6 static methods of Promise class:

1. Promise.all(promise) -- Waits for all promises to resolve and returns the array to their results. If any one fails, it becomes the error & all other results are ignored
2. Promise.allSettled(promises) -- Waits for all the promises to settle and resturn their results as an array of objects with status and value.
3. Promise.race(promises) -- Waits for the first promises to settle and its result/error becomes the outcome.
4. Promise.any (promise) -- Waits for the first promise to fulfill (& not rejected), and its result becomes the outcome. Throws AggregateError if all the promises are rejected.
5. Promise.resolve(value) -- Makes a resolves promise with the given value.
6. Promise.reject(error) -- Makes rejceted promise with the given error.

Quick quiz: Try all these promise APIs on your custom promises.

Async/Await
There is a special syntax to work with promises in javaScript
A function can be made async by using async keyword like this:

```
async function harry (){
        return 7;
}
```

An async function always returns a promise, other values are wrapped in a promise automatically
We can do simething like this:

```
Harry().then(alert)
```

So, async another keyword called await that works only inside async functions

The await keyword called await that works only inside async functions

Let value = await promise;

The await keyword makes javaScript wait until the promise settle and returns its value.

Its just a more elegant syntax of getting the promise result than promise, then + its easier to read & write

Error Handling
We all make mistakes. Also sometimes our script can have errors. Usually a program halts when an error occurs.

The try.. Catch syntax allows us to catch errors so that the script instead of dying can do some thing more reasonable

The try… catch syntax
The try catch syntax has two main blocks:
Try and then catch

```
Try{
        //try the code
}
Catch(err){
        //error handling
        }
```

It works like this

1. First the code in try is executed
2. If there is no error catch is ignored else catch is executed