***String***

*let* text = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

*let* length = text.length; *//? returns length of string*

*//! slice() extracts a part of a string and returns the extracted part in a new string.*

*let* slicetext = "Apple,Banana,Kiwi";

*let* part = slicetext.slice(7, 13);

*//! O/P=> "Banana"*

*let* lastpart = slicetext.slice(7);

*//! O/P=> "Banana, Kiwi"*

*let* negativepart = text.slice(-12);

*//! O/P=> "Banana, Kiwi" for negative index counting starts from last*

*let* newText = text.replace("Apple", "W3Schools");

*//! O/P=> W3Schools, Banana, Kiwi (will replace 1st occurance)*

*let* alltext = text.replaceAll("cats","dogs");

*//! replace all occurance*

*let* globalnewText = text.replace(/Microsoft/g, "W3Schools");

*//! will replace all occurance*

*let* smallnewText = text.replace(/MICROSOFT/i, "W3Schools");

*//! "/i" is used for case sensitive*

*let* upper = text.toUpperCase();

*let* lower = text.toLowerCase();

*let* text2 = text.trim();

*//! to trim space*

*let* char = text.charAt(0);

*//! returns character at that index.*

*let* charat = text[0]

*//! returns char at that index*

*let* splited = text.split(",")

*//! returns array splitted by separator*

**String search**

## JavaScript String indexOf()

The indexOf() method returns the **index** (position) the **first** occurrence of a string in a string:

let text = "Please locate where 'locate' occurs!";  
let index = text.indexOf("locate"); //op=> 7

## JavaScript String lastIndexOf()

The lastIndexOf() method returns the **index** of the **last** occurrence of a specified text in a string:

let text = "Please locate where 'locate' occurs!";  
let index = text.lastIndexOf("locate"); //op=>21

Both indexOf(), and lastIndexOf() return -1 if the text is not found:

Both methods accept a second parameter as the starting position for the search:

let text = "Please locate where 'locate' occurs!";  
let index = text.indexOf("locate", 15); //op=>21

## JavaScript String search()

The search() method searches a string for a string (or a regular expression) and returns the position of the match:

let text = "Please locate where 'locate' occurs!";  
text.search("locate");//op=>7

let text = "Please locate where 'locate' occurs!";  
text.search(/locate/);//op=>7

## JavaScript String match()

The match() method returns an array containing the results of matching a string against a string (or a regular expression).

let text = "The rain in SPAIN stays mainly in the plain";  
text.match("ain"); //=>

## JavaScript String includes()

The includes() method returns true if a string contains a specified value.

Otherwise it returns false.

let text = "Hello world, welcome to the universe.";  
text.includes("world");

Check if a string includes "world". Start at position 12:

let text = "Hello world, welcome to the universe.";  
text.includes("world", 12);

## JavaScript String startsWith()

The startsWith() method returns true if a string begins with a specified value.

Otherwise it returns false:

let text = "Hello world, welcome to the universe.";  
text.startsWith("Hello");

A start position for the search can be specified:

let text = "Hello world, welcome to the universe.";  
text.startsWith("world", 5)

## JavaScript String endsWith()

The endsWith() method returns true if a string ends with a specified value.

Otherwise it returns false:

let text = "John Doe";  
text.endsWith("Doe");

## The toString() Method

The toString() method returns a number as a string.

let x = 123;  
x.toString();  
(123).toString();  
(100 + 23).toString();

## The Number() Method

The Number() method can be used to convert JavaScript variables to numbers:

Number(true); op=>1   
Number(false); op=>  
Number("10"); op=>10  
Number("  10"); op=>10  
Number("10  "); op=>10  
Number(" 10  "); op=>10  
Number("10.33"); op=>10.33  
Number("10,33"); op=>NaN

## The parseInt() Method

parseInt() parses a string and returns a whole number. Spaces are allowed. Only the first number is returned:

parseInt("-10"); op=>-10  
parseInt("-10.33"); op=>-10.33  
parseInt("10"); op=>10  
parseInt("10.33"); op=>10.33

## How to Recognize an Array or any DS

A common question is: How do I know if a variable is an array?

The problem is that the JavaScript operator typeof returns "object":

const fruits = ["Banana", "Orange", "Apple"];  
let type = typeof fruits;

**Array**

## JavaScript Array shift()

The shift() method removes the first array element and "shifts" all other elements to a lower index.

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.shift(); op=>["Orange", "Apple", "Mango"];

## JavaScript Array unshift()

The unshift() method adds a new element to an array (at the beginning), and "unshifts" older elements:

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.unshift("Lemon"); op=>["Lemon","Banana", "Orange", "Apple", "Mango"];

## Changing Elements

Array elements are accessed using their **index number**:

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits[0] = "Kiwi";

## Flattening an Array

Flattening an array is the process of reducing the dimensionality of an array.

The flat() method creates a new array with sub-array elements concatenated to a specified depth.

const myArr = [[1,2],[3,4],[5,6]];  
const newArr = myArr.flat(); [1,2,3,4,5,6]

## JavaScript Array splice()

The splice() method can be used to add new items to an array:

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.splice(2, 0, "Lemon", "Kiwi");

The first parameter (2) defines the position **where** new elements should be **added** (spliced in).

The second parameter (0) defines **how many** elements should be **removed**.

The rest of the parameters ("Lemon" , "Kiwi") define the new elements to be **added**.

The splice() method returns an array with the deleted items:

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.splice(2, 2, "Lemon", "Kiwi");

## Using splice() to Remove Elements

With clever parameter setting, you can use splice() to remove elements without leaving "holes" in the array:

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.splice(0, 1);

## JavaScript Array slice()

The slice() method slices out a piece of an array into a new array.

This example slices out a part of an array starting from array element 1 ("Orange"):

const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];  
const citrus = fruits.slice(1); op=>["Orange", "Lemon", "Apple", "Mango"];

const citrus = fruits.slice(3); op=>["Apple", "Mango"];

const citrus = fruits.slice(1, 3); op=>["Orange", "Lemon"]

**sorting array**

## Sorting an Array

The sort() method sorts an array alphabetically:

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.sort(); op=>[Apple,Banana,Mango,Orange]

## Reversing an Array

The reverse() method reverses the elements in an array.

You can use it to sort an array in descending order:

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.sort();  
fruits.reverse(); op=>[Orange,Mango,Banana,Apple]

## Numeric Sort

By default, the sort() function sorts values as **strings**.

const points = [40, 100, 1, 5, 25, 10];  
points.sort(function(a, b){return a - b});

Use the same trick to sort an array descending:

const points = [40, 100, 1, 5, 25, 10];  
points.sort(function(a, b){return b - a});

## Using Math.max() on an Array

Math.max(1, 2, 3) op=>3

## Using Math.min() on an Array

Math.min(1, 2, 3) op=>1

## Sorting Object Arrays

JavaScript arrays often contain objects:

const cars = [  
  {type:"Volvo", year:2016},  
  {type:"Saab", year:2001},  
  {type:"BMW", year:2010}  
];

cars.sort(function(a, b){return a.year - b.year});

## JavaScript Array forEach()

The forEach() method calls a function (a callback function) once for each array element.

const numbers = [45, 4, 9, 16, 25];  
let txt = "";  
numbers.forEach(myFunction);  
  
function myFunction(value) {  
  txt += value + "<br>";  
}

## JavaScript Array map()

The map() method creates a new array by performing a function on each array element.

const numbers1 = [45, 4, 9, 16, 25];  
const numbers2 = numbers1.map((value, index, array) {  
  return value \* 2;  
});

## JavaScript Array filter()

The filter() method creates a new array with array elements that pass a test.

const numbers = [45, 4, 9, 16, 25];  
const over18 = numbers.filter((value, index, array)=> {  
  return value > 18;  
});

## JavaScript Array reduce()

The reduce() method runs a function on each array element to produce (reduce it to) a single value.

const numbers = [45, 4, 9, 16, 25];  
let sum = numbers.reduce((total, value) {  
  return total + value;  
});

## JavaScript Array find()

The find() method returns the value of the first array element that passes a test function.

const numbers = [4, 9, 16, 25, 29];  
let first = numbers.find((value, index, array) {  
  return value > 18;  
});

## JavaScript Array includes()

ECMAScript 2016 introduced Array.includes() to arrays. This allows us to check if an element is present in an array (including NaN, unlike indexOf).

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
  
fruits.includes("Mango"); // is true

**Forin loop**

## The For In Loop

The JavaScript for in statement loops through the properties of an Object:

const person = {fname:"John", lname:"Doe", age:25};  
  
let text = "";  
for (let x in person) {  
  text += person[x];  
}

## For In Over Arrays

The JavaScript for in statement can also loop over the properties of an Array:

const numbers = [45, 4, 9, 16, 25];  
  
let txt = "";  
for (let x in numbers) {  
  txt += numbers[x];  
}

## Array.forEach()

The forEach() method calls a function (a callback function) once for each array element.

## const numbers = [45, 4, 9, 16, 25]; let txt = ""; numbers.forEach((value, index, array)=>{   txt += value; }); Looping over a String

let language = "JavaScript";  
  
let text = "";  
for (let x of language) {  
text += x;  
}

**set**

## The new Set() Method

Pass an Array to the new Set() constructor:

const letters = new Set(["a","b","c"]); // removes duplicate

# **JavaScript Regular Expressions**

## Using String search() With a String

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

let text = "Visit W3Schools!";  
let n = text.search("W3Schools");

## Using String replace() With a String

The replace() method replaces a specified value with another value in a string:

let text = "Visit Microsoft!";  
let result = text.replace("Microsoft", "W3Schools");

**call bind apply**

## Function Bind()

With the bind() method, an object can borrow a method from another object.

const person = {  
  firstName:"John",  
  lastName: "Doe",  
  fullName: function () {  
    return this.firstName + " " + this.lastName;  
  }  
}  
  
const member = {  
  firstName:"Hege",  
  lastName: "Nilsen",  
}  
  
let fullName = person.fullName.bind(member);

**call, apply, bind:**

## 

## Call method are used to call a method directly by passing the referance of the object which point to “this” inside the method.

## Apply is same as call method except it recives the seconf parameter in the form of a array.

## Bind method doesn’t call the method but provides the copy of the method which can be invoked later.

## Js curring:

function add(a, b){

return a+b

}

function multiply(a, b){

return a\*b

}

function op(operation){

return function(a){

return function(b){

return operation(a,b)

}

}

}

const op1 = operation=>a=>b=>operation(a,b)

console.log(op(add)(4)(6))

console.log(op1(multiply)(4)(6))

## Currying is a process of converting a function with multiple argument to multipl function with single argument in a sequence.

## curring function are used for creating function that are already configured.

## It can be used to create higher order function.

## Kind of code reusability as well.

## JavaScript Promises

A Promise is a JavaScript object that links "Producing Code" and "Consuming Code".

"Producing Code" can take some time and "Consuming Code" must wait for the result.

const myPromise = new Promise(function(myResolve, myReject) {  
// "Producing Code" (May take some time)  
  
  myResolve(); // when successful  
  myReject();  // when error  
});  
  
// "Consuming Code" (Must wait for a fulfilled Promise).  
myPromise.then(  
  function(value) { /\* code if successful \*/ },  
  function(error) { /\* code if some error \*/ }  
);

## JavaScript Object Entries

ECMAScript 2017 added the Object.entries() method to objects.

Object.entries() returns an array of the key/value pairs in an object:

const person = {  
  firstName : "John",  
  lastName : "Doe",  
  age : 50,  
  eyeColor : "blue"  
};  
  
let text = Object.entries(person); op=>firstName,John,lastName,Doe,age,50,eyeColor,blue

## JavaScript Threads

In JavaScript you use the Web Workers API to create threads.

Worker threads are used to execute code in the background so that the main program can continue execution.

Worker threads run simultaneously with the main program. Simultaneous execution of different parts of a program can be time-saving.

Debounce and throttling:

* Debouncing is used to wait for a certain time period before execuitng a function again.
* Throttling limits the number of call of a function in a time period.

const fn =(e)=>{

setDisplay(e.target.value)

}

const deBounce = (fn,timeout)=>{

let timer;

return function(...args){

clearTimeout(timer)

timer=setTimeout(()=>{

fn(...args)

},timeout)

}

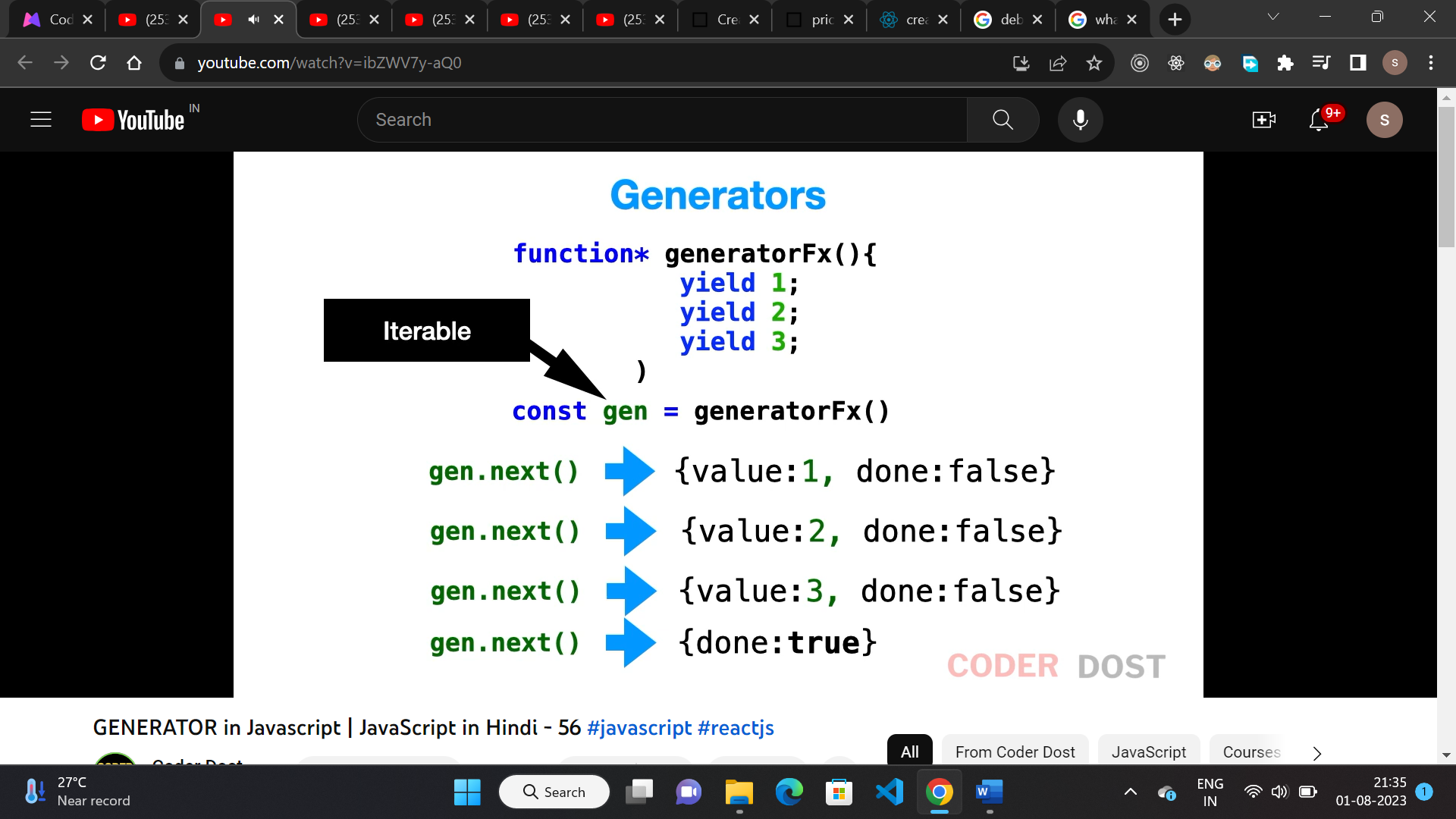
}

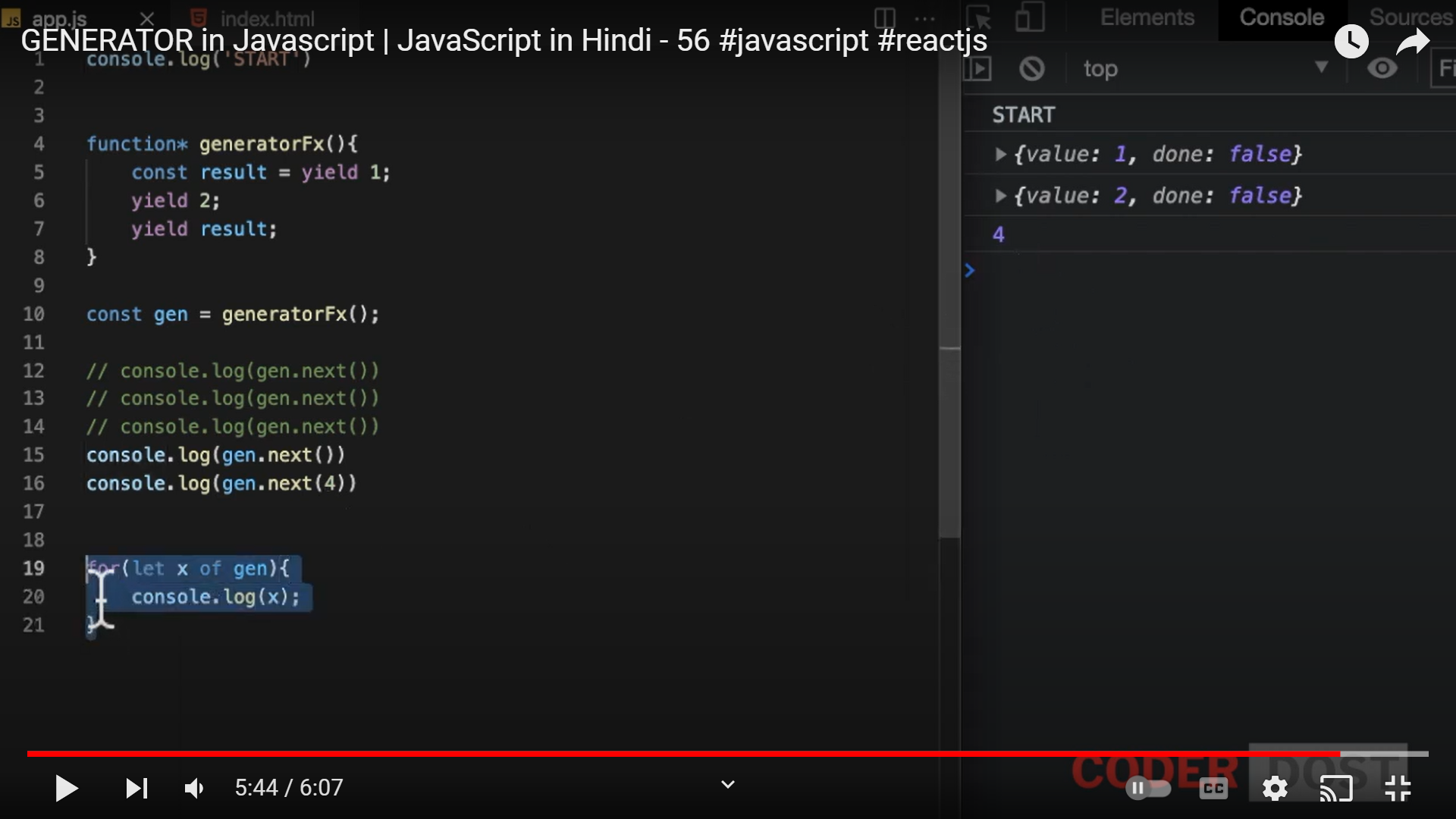
const handleChange = deBounce(fn, 500)

**event bubbling:**

* When we run a event on the child element, then all parent element having same event will run as well and we call that event bubbling.
* Only same event will get called not different event.
* To stop event bubbling we call “event.stopPropagation”

**Generator function:**





* Generator function is used to generate vales.
* It returns a generator object.

**Parsing deeply nested JSON javascript:**

* This can be achived using recursive function in javascript as mentioned below.

