1) Introduction

1) what is JS?

JS is a programing language which is used to develop dynamic web and mobile apps.

By using JS we can manipulate (add, update, delete) DOM.

By using JS we can perform validations.

2) What is ES?

Ecma Script is a specification for JS.

3) What is nodejs?

Node js is a run time environment for JS. By using nodejs we can create APIs.

Node js is not a programing lanaguage. Node js is not a library.

Node js is not a framework.

2) Setup Development environment?

2) Front-end

- We need a Browser (Chrome).
- We need an IDE to write code (vs code).
- We need a HTML file to run js code in front-end.
- Every browser has js engine in it to run js code.

3) Back-end

- We need to install Nodejs to run js code in back-end.
- We need an IDE to write code.
- Nodejs is a runtime environment for js.
- By using nodejs js we can create APIs.
- How to run js file in nodejs

=> node filename

3) Basics

=> When to store data?

We need to store data in memory, when there is need in app.

=> How to store data in memory?

By using var, let, const, we can store data in memory.

We can perform some operations on the data which is stored in memory

=> What to store in memory?

We have to store values (data) by using data types. These are the real values to do any functionality.

1) variables (we need variables to manipulate/perform some operation/update/delete/save)

We will declare variables by using var, let, const keywords.

2) data types

Primitive data types: we use primitives data types to store single value in memory.

Note: All primitive data types will store value directly in memory.

1) string: String is collection of characters to be stored in memory.

By using '' "" `` we can store string in memory.

2) number: We can store numbers in memory by using number data type.

whether it is integer or float number.

3) boolean:

We can store true or false in memory by using boolean datatype.

4) undefined:undefined means a variable has been declared but it's value has not been assigned.

5) null: Null means an empty value. The variable which has been assigned as null contains no value.

Reference data types: we use reference data types to store multiple values in memory.

Note: All reference data types will store value somewhere in memory location. The stored memory location address will be stored in main memory.

1) Object:

By using object we can store multiple values in single memory location in the form of

key & value pair. By using dot operator we can access the object values in app.

Objects are often used to model real-world entities such as a person, car, or any other entity that has properties and behaviors.

2) Array:

By using array we can store collection of values in single memory location. It stores only values. Internally js attaches index numbers to the values in array. By using index numbers we can access array values in app.

3) Function:

- Funtion is block of code. By using a function we can do some task and return some value.

- For every function call seperate excution context will be created.

- For every excution context, there are memory creation phase and code excution phase.

- We can stores multiple values in functional scope.

ES-6

4) Map

5) WeakMap

6) Set

7) WeakSet

3) typeOf operator

- By using typeof operator we can find data type for the value which is stored in memory.

value : datatype

'sachin': string

40: number

true/false: boolean

undefined: undefined

null: object

{}: object

[]: object

function(){}: function

4) operators:

why operators?

we use operators to develop some logic or expression in combination with variables.

1) Arithmatic

We use arithmatic operators to perform some mathematical operations.

+ add

- subtraction

* multification

/ devision

% remainder

** exponential

++ incremental : It increases 1 at a time.

__ decremental : It decreases 1 at a time.

2) Assignment (=)

By using assignment operator we can assign/store value (data) to a variable.

3) Comparision

It compares the two values, the result from this operators will be true/false.

1) Rational/Relational

>

>=

<

<=

2) Equality

1) Loose equality (==) It compares only value of variables

2) Strict equality (===) It compares value and data type of variables

undefined == null

undefined === null

3) Not equality

1) Loose inequality (!=)

The != operator is the inequality operator. It checks whether two values are not equal, regardless of their types. If the values are different, it returns true. If the values are the same, it returns false.

2) Strict inequality (!==)

The !== operator is the strict inequality operator. It checks whether two values are not equal and whether they are of the same type. If the value or the type are different, it returns true; otherwise, it returns false.

4) Ternary operator

We use ternary operator to render content conditionally.

```
let age = 15;

let vote = age >= 18 ? 'Having vote' : 'Not having vote';

console.log(vote);
```

5) Logical operator

In JavaScript, logical AND (&&) and logical OR (||) are operators used to perform logical operations on boolean values or expressions.

1) logical and &&

2) logical or ||

3) ! Operator

The exclamation mark (!) is the logical NOT operator.

When used, it converts a true value to false and vice versa.

! is the logical NOT operator, used for negating boolean values.

6) Control statements

We use control statements to develop some logic or functionality when we have multiple conditions.

1) if else - In JavaScript, the if...else statement is used for conditional execution of code. It allows you to perform different actions based on a specific condition.

The syntax of the if...else statement is as follows:

```
if (condition) {
 // Code block to be executed if the condition is true
 } else {
 // Code block to be executed if the condition is false
 }
```

```
const num = 10;
if (num > 0) {
 console.log("Positive");
} else if (num < 0) {
 console.log("Negative");
} else {
 console.log("Zero");
}
```

2) switch case (It does not work for step value)

- In JavaScript, the switch statement is another way to perform conditional excution of code based on the value of an expression. It is often used as an alternative to multiple if...else statements.

- The switch statement evaluates an expression once and then matches the value of the expression to a case label. If a matching case label is found, the corresponding block of code is executed.

```
switch (expression) {
  case value1:
  // Code to be executed if the expression matches value1
  break;
```

```
  case value2:
  // Code to be executed if the expression matches value2
  break;
  // Add more cases as needed
  default:
  // Code to be executed if none of the cases match the expression
  break;
 }
Ex:  const dayOfWeek = 3;

switch (dayOfWeek) {
 case 1:
  console.log("Monday");
  break;
 case 2:
  console.log("Tuesday");
  break;
 case 3:
  console.log("Wednesday");
  break;
 case 4:
  console.log("Thursday");
  break;
 case 5:
  console.log("Friday");
  break;
 case 6:
  console.log("Saturday");
  break;
 case 7:
  console.log("Sunday");
```

```
   break;
  default:
   console.log("Invalid day");
 }
```

7) Loops

- We use loops to do same task agin and again simply.

- We use loops to access memory value multiple times in an application.

Conditinal Loops

1) for loop:

In JavaScript, a for loop is used to execute a block of code repeatedly for a specified number of times.

2) while loop:

In JavaScript, a while loop is used to execute a block of code repeatedly as long as a specified condition is true. The loop continues until the condition evaluates to false.

3) do while loop:

In JavaScript, a do-while loop is similar to a while loop, but with a slight difference. The primary difference is that in a do-while loop, the loop body is executed at least once before the loop condition is checked.

This ensures that the loop body is executed at least once, regardless of whether the condition is initially true or false. The do-while

4) infinity loop:

An infinite loop in JavaScript is a loop that runs infinitely, continuously executing the same code block without ever stopping. This usually happens when the loop condition always evaluates to true

**break=> by using Break keyword we can break the loop and make exit the loop.

**continue=> by using continue keyword we can make jump the loop.
Non conditional loops

5) for in loop:

We use this loop to iterate keys in object.

In JavaScript, the for...in loop is used to iterate over the enumerable properties of an object. It allows you to loop through the keys (property names) of an object and access their corresponding values.

6) for of loop:

In JavaScript, the for...of loop is used to iterate over the values of iterable objects(array/string object/set/map/arguments object/generator object/).. It provides an easy and concise way to loop through arrays, strings, sets, maps, and other objects that are iterable.

Note:

- It's important to note that the for...in loop should be used for iterating over object properties. If you want to loop through elements of an array, it is recommended to use the for...of loop or a simple for loop with an index.

- keep in mind that the for...of loop is not suitable for iterating over regular objects (objects created with {}) since they are not iterable by default. For iterating through object properties, you should use the for...in loop.

8) Funtions

1) What is function?

=>Function is a block of code which is used to do some task and return value.

=>It stores multiple values in functional scope.

=>For every function call seperate excution context will be created.

For every excution context there are memory creation phase and code excution phase.

2) How can we define function?

1) Function declaration

2) Function expression

1) named fucntion expression

2) anonymous function expression

3) arrow function (ES-6)

3) parameters vs arguments

Parameters

Parameters are the placeholders or variables defined in the function's declaration.

- They are like local variables that store the values passed to the function when it is called.

- They are defined within the function's parentheses in the function declaration and serve as placeholders for the arguments that will be provided when the function is called.

Arguments

Arguments are the actual values or expressions passed to a function when it is called.

- They represent the data that you want to work with within the function.

- The number of arguments should match the number of parameters defined in the function.

- Arguments are provided in the function call and are placed inside the parentheses.

4) what is default parameter

- Default parameters were introduced in ECMAScript 6 (ES6) and have since become a common feature in modern JavaScript.

- To define default parameters in a function, you assign a default value to a parameter in the function's declaration.

- Default parameters in JavaScript allow you to specify default values for function parameters.

5) Varying no of parameters or arguments

If we have varying no of params and arguments then we have to handle with below concepts.

=> arguments object (ES-5)

It takes all values at a time and stores in memory.

This is available in all functions except arrow function.

arguments object is an iterable object, it has symbol.iterator() method.

=> rest parameter (ES-6)

It starts with ...

It takes all values and stores in array.

It should be last parameter in parameter list.

6) Scope

1) Global scope

2) Function / local scope

3) Block scope

Note:

1) var is functionl scope

2) let, const is block scope

-In Js, "scope" refers to the context in which variables, functions are stored and can be accessed.

-The scope determines the accessability and lifetime of these variables and functions.

-Every execution context will create a new scope.

-Every function call will create new execution context.

JavaScript has two main types of scope:

Global Scope:

Variables declared or stored outside of any function or block have global scope.

They can be accessed from any part of the code, including inside functions and blocks.

Local Scope:

Variables declared or stored within a function or a block have local scope. They are only accessible within that specific function or block.

Note:

Local variables take precedence over global variables if they share the same name.

Function scope, which means that variables declared with var are only scoped within the function where they are defined.

However, with the introduction of let and const in ES6, block scope was introduced. Variables declared with let or const are scoped to the nearest enclosing block (defined by curly braces {}).