JavaScript

1. JavaScript Introduction

Que 1:- What is JavaScript? Explain the role of JavaScript in web development.

- JavaScript is a high-level, interpreted programming language used mainly to make web pages interactive and dynamic
- It is one of the core technologies of web development, along with HTML and CSS.
- HTML defines the structure of a webpage.
- CSS controls the design and layout.
- JavaScript adds interactivity and behavior.

Role of JavaScript in Web Development:

- **Adds Interactivity:** Allows users to interact with elements like buttons, forms, and menus.
- **Controls Webpage Behavior**: You can show/hide content, validate forms, or create animations.
- **Dynamic Content Updates**: JavaScript can update webpage content without reloading the page using technologies like AJAX.
- **Backend Development:** With Node.js, JavaScript can also be used for server-side programming.

Que 2:- How is JavaScript different from other programming languages like Python or Java?

- JavaScript VS python vs java

1.Type

- JavaScript: Scripting language mainly for Web
- Python: General-purpose language (easy syntax)
- Java: Object-oriented, strongly typed language

2.Running Environment

- JavaScript: Runs in browsers & Node.js
- Python: Runs locally via interpreter
- Java: Runs on Java Virtual Machine (JVM)

3.Syntax

- JavaScript: Uses { } curly braces
- Python: Uses indentation for blocks
- Java: Uses { } curly braces & semicolon ;

4.speed

- JavaScript: Fast in browser engines
- Python: Moderate speed
- Java: Very fast (compiled

5.Use Case

- JavaScript: Web development (Frontend + Backend)
- Python: AI, Data Science, Scripting
- Java: Mobile apps, Desktop apps, Backend

6.File Extension

- JavaScript jJs
- Python .py

- Java .java

Question 3: Discuss the use of <script> tag in HTML. How can you link an external JavaScript file to an HTML document?

- The <script> tag in HTML is used to add JavaScript code to a webpage.
- It allows us to create dynamic, interactive features such as:
  - Form validation
  - Animations
  - Alerts
  - Event handling
  - DOM manipulation

1.inline JavaScript:

Ex. <script>

Alert(“Hello Worls!”)

</script>

2.external JavaScript:

Ex. <script src=”script.js”>

</script>

2.variable and Data Types

Que 1: What are variables in JavaScript? How do you declare a variable using var, let, and const?

- A variable in JavaScript is a container used to store data values. It acts as a symbolic name for a value that can be changed or accessed later in the program.

Declare Variable:

1.var

- Used in order JavaScript Versions.
- Has function scope
- Can be redeclared and updated

2.let

- Introduction in ES6
- Has block scope
- Can be updated but not redeclared in the same scope.

3.const

- Used for constant values that should not change.
- Has block scope.
- Cannot be redeclared or updated.

Question 2: Explain the different data types in JavaScript. Provide examples for each.

Two types of data

1.primitive datatype

2.Non – primitive datatype

1.number

- It represents numeric values (integer and decimal both).
Example: 10 , 3.14

2.string

- it represented a sequence.

Example: “Hello” , ‘world’

3.Boolean

- true or false value.

Example: true , false

4.Undefined

- a variable declared but not assigned a value

Example: let x; // undefined

5.Null

- represents “no value”

Example: let y = null;

6.symbol

- unique and immutable value.

Example: let id = symbol{“id”};

Ex.

let name = “Hetsi”;

let age = “20”;

let city ;

let salary = “null”;

let BigNumber = “123n”;

2.Non-Primitive Data Types

1.object: collection of key value

Ex. {name: “Hetsi” , age: 20}

2.Array: ordered list of values.

Ex. [10,20,30]

3.function: Block of reusable code.

Ex. Function greet() {

Alert(“hi”);

}

Question 3: What is the difference between undefined and null in JavaScript?

Undefine

- A variable has been declared but no value assigned.
- Types : undefined
- JavaScript (automatically)

- Ex. let x;

Console.log(y);

Null

- Represents “no value” or “empty value”.
- Types: object
- Developer(manually)
- Ex. let y = null;

  Console.log(y);

3.operators

Question 1: What are the different types of operators in JavaScript? Explain with examples.

- **Arithmetic operators**
- **Assignment operators**
- **Comparison operators**
- **Logical operators**

1. Arithmetic Operators

- These operators are used to perform mathematical calculations.
- + add , - subtract , * multiply , / divide , % modulus

Ex.

```
let a = 10;

let b = 3;

console.log(a + b); // 13

console.log(a - b); // 7

console.log(a * b); // 30

console.log(a / b); // 3.33
```

```
console.log(a % b); // 1  (remainder)
```

2. Assignment Operators

- These operators are used to assign values to variables.

- = , += , -= , *= , /= , %=

Ex.

```
let x = 5;   //
x += 3;  // x = x + 3 → 8
x -= 2;  // x = x - 2 → 6
x *= 2;  // x = x * 2 → 12
x /= 3;  // x = x / 3 → 4
```

3. Comparison Operators

- These operators are used to compare two values.
- They return either true or false.

- == , === , != , !== , > , < , >= , <=

Ex.

```
let a = 10;
let b = 5;

console.log(a > b);   // true
console.log(a < b);   // false
```

```
console.log(a == "10");  // true (only value checks)
console.log(a === "10"); // false (value + type check)
console.log(a != b);  // true
console.log(a !== 10); // false
```

4. Logical Operators

- These operators are used to combine conditions.
- They return true or false.

&& (AND), || (OR), ! (NOT)

Ex.

```
let age = 20;
let isStudent = true;

// AND: both conditions must be true
console.log(age > 18 && isStudent); // true

// OR: at least one condition true
console.log(age < 18 || isStudent); // true

// NOT: reverses the value
console.log(!isStudent); // false
```

Question 2: What is the difference between == and === in JavaScript?

== : check only value

Equality operators

5 == ‘5’

Output-true

=== : value and data type

Strict equality operators

5 === ‘5’

Output-false

Ex.

```
let a=5;
let b=’5’;
console.log(a == b);
console.log(a === b);
```

4.control flow(if-else ,switch)

Question 1: What is control flow in JavaScript? Explain how if-else statements work with an example.

- Control flow in JavaScript refers to the order in which statements are executed in a program.

- code runs from top to bottom, but using control flow statements like if else, switch, loops -, etc.
- we can make decisions and change the flow based on conditions.

If-else Statement:

- It checks a condition.
- If the condition is true, the code inside if block runs.
- If the condition is false, the code inside else block runs.

Ex.

```
let age = 18;
if (age >= 18) {
   console.log("You are eligible to vote!");
} else {
   console.log("You are not eligible to vote!");
}
```

Question 2: Describe how switch statements work in JavaScript. When should you use a switch statement instead of if-else?

- A switch statement is used to compare a single value with different possible cases.
- It executes the matching case's code and stops when it finds a break statement.

Ex.

```
let day = 3;
switch(day) {
   case 1:
      console.log("Monday");
      break;
   case 2:
      console.log("Tuesday");
      break;
   case 3:
      console.log("Wednesday");
      break;
   default:
      console.log("Invalid day");
}
```

5.Loops (for , while , do-while)

Question 1: Explain the different types of loops in JavaScript (for, while, do-while). Provide a basic example of each.

- Loops are used in JavaScript to repeat a block of code multiple times until a condition becomes false.

For loop

- Used when the number of iterations is known.

Ex.

```
for (let i = 1; i <= 5; i++) {
   console.log(i);
}
```

While Loop

- Executes code while the condition remains true.

Ex.

```
let i = 1;
while (i <= 5) {
   console.log(i);
   i++;
}
```

Do-While Loop

- Executes the block at least once even if the condition is false, because the condition is checked after execution.

Ex.

```
let i = 1;
do {
   console.log(i);
   i++;
```

} while (i <= 5);

Question 2: What is the difference between a while loop and a do-while loop?

While loop

- Before executing the loop
- May execute 0 times if condition is false.
- When you want to run loop only if condition is true first
- If the condition is false at the beginning, the loop will not run even once.
- Syntax: while()

Do while loop

- After executing the loop.
- Execute at least 1 time
- When the code must run once before checking condition.
- Even if the condition is false, the loop will run at least once.
- Syntax: do { } while()

6.functions

Question 1: What are functions in JavaScript? Explain the syntax for declaring and calling a function.

- A function in JavaScript is a reusable block of code designed to perform a specific task.
- Instead of writing the same code again and again, we define a function once and call it whenever needed.

Syntax:

- function functionName() {
  // code to execute
  }

Ex.

```
function greet() {
console.log("Hello, welcome to JavaScript!");
}
greet();
```

Question 2: What is the difference between a function declaration and a function expression?

Function declaration

- Created using the function keyword with a name.
- Fully hoisted (can be called before declaration).
- function myFunc() {}

Function expression

- Assigned to a variable, can be anonymous.
- Not fully hoisted (cannot be called before assignment).
- let myFunc = function() {};

Ex.

```
Declaration: function add(a, b) {
                return a + b;
                }
Expression:  let add = function(a, b) {
```

```
return a + b;
};
```

Question 3: Discuss the concept of parameters and return values in functions.

Parameters:

- The values listed inside the function parentheses when defining a function.
- Used to **receive** data from function calls.

Return Value:

- A function can send back a value using the return statement.
- After return executes, the function stops running and sends output.

Ex.

```
function multiply(x, y) {   // x and y are parameters
   return x * y;        // returning the result
}
let result = multiply(5, 4); // 5 and 4 are arguments
console.log(result);
```

7.Arrays

Question 1: What is an array in JavaScript? How do you declare and initialize an array?

- An array in JavaScript is a special type of object used to store multiple values in a single variable.
- Each value in an array has an index, starting from 0.

Declaration & Initialization:

```
let fruits = ["Apple", "Banana", "Mango"];

let numbers = [];

numbers[0] = 10;

numbers[1] = 20;

let data = ["Sahil", 22, true];
```

Question 2: Explain the methods push(), pop(), shift(), and unshift() used in arrays.

push()

- push() adds a new element to the end of an array.

Ex.

```
let arr = [1, 2, 3];

arr.push(4);

 console.log(arr);
```

pop()

- pop() removes the element from the end of an array.
- It also returns the removed value.

Ex.

```
let arr = [10, 20, 30];
arr.pop();
console.log(arr);
```

shift()

- shift() removes the first element of an array.
- It returns the removed value as well.

ex.

```
let arr = ['a', 'b', 'c'];
arr.shift();
console.log(arr);
```

unshift()

- unshift() adds a new element to the beginning of an array.

Ex.

```
let arr = [5, 6, 7];
arr.unshift(4);
console.log(arr);
```

8.objects

Question 1: What is an object in JavaScript? How are objects different from arrays?

- A JavaScript object is a collection of key–value pairs where each key (also called a property name) stores a specific value.
- Objects are used to represent real-world entities like a person, product, or car with multiple properties.

Ex.

```
let student = {
   name: "Sahil",
   age: 20,
   city: "Surat"
};
```

Difference between object and arrays

Object

- Key value pairs
- By property name (key)
- For describing properties of an entity.
- String/identifier

Arrays

- Indexed values
- By index number
- For ordered list of items
- Numeric index(0,1,2..)

Question 2: Explain how to access and update object properties using dot notation and bracket notation.

1.Dot Notation

- Used when property name has no spaces or special characters.

Ex:

```
let person = { name: "Amit", age: 25 };

console.log(person.name);

person.age = 26;

console.log(person.age);
```

2.bracket Notation

- Property names contain spaces
- Property names are dynamic (variable-based)

Ex:

```
let person = { "first name": "Amit", age: 25 };

console.log(person["first name"]);

person["age"] = 30;

console.log(person.age);
```

9.javaScript Events

Question 1: What are JavaScript events? Explain the role of event listeners.

- JavaScript events are actions or occurrences that happen in the browser, which JavaScript can respond to.

Examples of events include:

- Clicking a button
- Hovering the mouse
- Typing on the keyboard
- Loading a webpage
- Submitting a form

Question 2: How does the addEventListener() method work in JavaScript? Provide an example.

The addEventiListener() method attaches an event to an HTML element without overwriting other events.
It takes two main arguments**:**

- The event name (like “click”)
- The function to execute when the event happens

Ex.

```
<button id="myBtn">Click Me</button>

<script>

  let button = document.getElementById("myBtn");

  button.addEventListener("click", function() {

    alert("Button clicked!");
```

```
});
</script>
```

10.DOM Manipulation

Question 1: What is the DOM (Document Object Model) in JavaScript? How does JavaScript interact with the DOM?

- The DOM (Document Object Model) is a programming interface that represents a webpage structure as a tree of objects.

With the help of DOM, JavaScript can:

- Access HTML elements
- Change content / text
- Change CSS styles
- Add or remove elements dynamically
- Handle user interactions (events)

Question 2: Explain the methods getElementById(), getElementsByClassName(), and querySelector() used to select elements from the DOM.

1. getElementById()

Purpose:
Used to select one single element from the DOM using its unique id.

How it works:

- Every id in HTML is unique.
- This method directly finds that element and returns it.

Ex.

```
<p id="title">Hello</p>

<script>

  let a = document.getElementById("title");

  console.log(a);

</script>
```

2. getElementsByClassName()

Purpose:
Used to select multiple elements that have the same class name.

How it works:

- Finds all elements with the given class.
- Returns an HTMLCollection (a list that looks like an array).
- You can access elements using index numbers.

Ex.

```
<p class="demo">A</p>
```

```
<p class="demo">B</p>


<script>
   let items =
document.getElementsByClassName("demo");
   console.log(items[0]);
   console.log(items[1]);
</script>
```

3. querySelector()

Purpose:
Selects the first element that matches a given CSS selector.

How it works:

- Very powerful because it supports all CSS selectors: id, class, tag, attributes, etc.
- Returns only the first matching element.

Ex.

```
<p id="one" class="demo">Hello</p>
<script>
   document.querySelector("#one");
   document.querySelector(".demo");
   document.querySelector("p");
</script>
```

11. JavaScript Timing Events (setTimeout, setInterval)

Question 1: Explain the setTimeout() and setInterval() functions in JavaScript. How are they used for timing events?

- JavaScript provides two important timing functions to schedule code execution after a delay or repeatedly over time.

1.setTimeout()

- Executes a function **once** after a specified time delay
- Delay is given in **milliseconds** (1000 ms = 1 second)

Example behavior:

- Run code after 3 seconds → setTimeout(function, 3000)

2. setInterval()

- Executes a function repeatedly at fixed time intervals
- Continues running unless stopped using clearInterval()

Example behavior:

- Run code every 1 second → setInterval(function, 1000)

Question 2: Provide an example of how to use setTimeout() to delay an action by 2 seconds.

Example Code:

setTimeout(function() { console.log("This message appears after 2 seconds!"); }, 2000);

Explanation:

- The function inside setTimeout() will run **after 2000 milliseconds** (2 seconds)
- A message will show in the console after the delay

12.JavaScript Error Handling

Question 1: What is error handling in JavaScript? Explain the try, catch, and finally blocks with an example.

- Error handling in JavaScript allows us to detect and manage errors in a program without stopping its execution.
- JavaScript uses the try...catch mechanism to handle runtime errors safely.

Try - Code that may cause an error

Catch - Runs if an error occurs in try block

Finally - Always runs (optional) whether error happens or not

Ex.

```
try {
  let num = 10;
  console.log(num.toUpperCase());
```

```
}
catch (error) {
    console.log("An error occurred: " + error.message);
}
finally {
    console.log("This block always runs!");
}
```

Question 2: Why is error handling important in JavaScript applications?

- Error handling is important because:
- It prevents the application from crashing
- Helps in debugging by showing clear error messages
- Improves user experience by handling failures smoothly

Useful for handling errors like:

- Network failures
- Invalid user input
- Missing data or undefined variables