JavaScript-Theory Assignment

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

JavaScript is a programming language used to make web pages **interactive and dynamic**.

Role of JavaScript in web development:

- Adds interactivity (click, input, events)
- Updates content without page reload
- Validates forms
- Controls HTML & CSS
- Used in frontend and backend development

JavaScript makes websites **alive and user-friendly**.

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

JavaScript vs Python vs Java (Short Explanation):

- **JavaScript** runs mainly in the **browser** and is used for **web interactivity**.
- **Python** is mainly used for **data science, AI, automation, and backend**.
- **Java** is used for **enterprise applications, Android apps, and large systems**.

Key Differences:

- JavaScript runs in the **browser**, Python & Java usually do not.

- JavaScript is **interpreted**, Python is interpreted, Java is **compiled + interpreted**.
- JavaScript is **dynamically typed**, Java is **statically typed**.
- JavaScript is best for **frontend web development**.

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

Use of <script> tag in HTML

The <script> tag is used to **write or include JavaScript** in an HTML page.
It tells the browser to execute JavaScript code.

Uses:

- Add interactivity
- Handle events (click, submit, etc.)
- Manipulate HTML & CSS

Example (internal JavaScript):

```
<script>
  alert("Hello JavaScript");
</script>
```

Linking an External JavaScript File

You can link an external .js file using the **src attribute** of <script>.

Example:

```
<script src="script.js"></script>
```

Question 4: What are variables in JavaScript? How do you declare a variable using var, let, and const?

Variables in JavaScript

Variables are used to **store data values** (like numbers, text, etc.) in JavaScript.

Declaring Variables in JavaScript

JavaScript provides **three ways** to declare variables:

1 var

- Old method
- Function-scoped
- Can be re-declared and updated

var name = "Hasti";

---

2 let

- Block-scoped
- Can be updated but **not re-declared** in the same scope

let age = 20;

---

3 const

- Block-scoped
- Cannot be updated or re-declared
- Used for fixed values

const country = "India";

Question 5: Explain the different data types in JavaScript. Provide examples for each.

Data Types in JavaScript

JavaScript has **two main types** of data types: **Primitive** and **Non-Primitive**.

🔷 Primitive Data Types

1️⃣ **Number** – for numbers

let age = 20;

2️⃣ **String** – for text

let name = "Hasti";

3️⃣ **Boolean** – true or false

let isActive = true;

4️⃣ **Undefined** – variable declared but not assigned

let x;

5️⃣ **Null** – empty or no value

let y = null;

6️⃣ **BigInt** – very large integers

let bigNum = 123456789012345678 90n;

7️⃣ **Symbol** – unique values (advanced use)

let id = Symbol("id");

🔷 Non-Primitive Data Types

8️⃣ **Object** – stores data in key-value pairs

let student = { name: "Hasti", age: 20 };

9️⃣ **Array** – stores multiple values

let fruits = ["apple", "banana", "mango"];

1 0 **Function** – block of reusable code

function greet() {

return "Hello";

}

Question 6: What is the difference between undefined and null in JavaScript?

Point	**undefined**	**null**
Meaning	Variable **declared but value not assigned**	Variable **intentionally empty**
Who sets it	JavaScript automatically	Developer manually
Data type	undefined	object
Default value	Yes	No
Use case	Value not yet available	Value cleared / empty on purpose

Question 7: What are the different types of operators in JavaScript? Explain with examples. • Arithmetic operators • Assignment operators • Comparison operators • Logical operators

1 Arithmetic Operators

Operator	Meaning	Example
+	Addition	10 + 5 → 15
-	Subtraction	10 - 5 → 5
*	Multiplication	10 * 5 → 50
/	Division	10 / 5 → 2
%	Modulus (remainder)	10 % 3 → 1
++	Increment	a++
--	Decrement	a--

```
let a = 10, b = 3;
console.log(a + b); // 13
console.log(a % b); // 1
```

2 Assignment Operators

Operator	Example	Meaning
=	x = 10	Assign value
+=	x += 5	x = x + 5
-=	x -= 2	x = x - 2
*=	x *= 3	x = x * 3

```
let x = 10;
x += 5;
console.log(x); // 15
```

3 Comparison Operators

Operator	Meaning	Example
==	Equal value	5 == "5" → true
===	Equal value & type	5 === "5" → false
!=	Not equal	5 != 3 → true
!==	Not equal value or type	5 !== "5" → true
>	Greater than	10 > 5 → true
<	Less than	3 < 5 → true

```
console.log(5 === "5"); // false
```

4 Logical Operators

Operator	Meaning	Example
&&	AND	true && false → false
`		`
!	NOT	!true → false

```
let age = 20;
console.log(age > 18 && age < 60); // true
```

Question 8: What is the difference between == and === in JavaScript?

Point	== (Loose Equality)	=== (Strict Equality)
Comparison	**Only value** compare	**Value + data type** compare
Type conversion	Automatically	No type conversion
Result reliability	Less reliable	More reliable (recommended)

Question 9: What is control flow in JavaScript? Explain how if-else statements work with an example.

What is Control Flow in JavaScript?

Control flow refers to the **order in which the statements of a program are executed** in JavaScript.
Normally, code runs from **top to bottom**, but control flow statements allow the program to **make decisions** and execute different blocks of code based on conditions.

How do if–else statements work?

The if–else statement is used to **check a condition** and execute code accordingly:

- If the condition is **true**, the if block executes.
- If the condition is **false**, the else block executes.
- **Syntax**

```
if (condition) {
// code executes if condition is true
 } else {
// code executes if condition is false
   }
```

Question 10: Describe how switch statements work in JavaScript. When should you use a switch statement instead of if-else?

How do switch statements work in JavaScript?

A **switch statement** is used to execute **one block of code out of many possible options** based on the value of an expression.

The expression is evaluated **once**, and its value is compared with each case.
When a matching case is found, the corresponding code block runs.

Example

```
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");
}
```

Question 11: Explain the different types of loops in JavaScript (for, while, do-while). Provide a basic example of each.

What is a Loop?

A **loop** is used to **repeat a block of code multiple times** until a specified condition is met.

---

1 for Loop

Use

- When the **number of iterations is known**

Syntax

```
for (initialization; condition; increment/decrement) {
  // code to repeat
}
```

2 while Loop

Use

- When the **number of iterations is not known in advance**

Syntax

```
while (condition) {
  // code to repeat
}
```

3 do–while Loop

Use

- When the loop must run **at least once**

Syntax

```
do {
  // code to repeat
} while (condition);
```

Question 12: What is the difference between a while loop and a do-while loop?

Point	while Loop	do–while Loop
Condition check	Before loop	After loop
Minimum execution	0 times	1 time
Use case	When loop may not run	When loop must run at least once

Question 13: What are functions in JavaScript? Explain the syntax for declaring and calling a function.

What are Functions in JavaScript?

A **function** in JavaScript is a **reusable block of code** that performs a specific task.
Functions help to **reduce code repetition**, improve readability, and make programs easier to maintain.

Syntax for Declaring a Function

Function Declaration

```
function functionName(parameters) {
  // code to be executed
}
```

Calling (Invoking) a Function

To execute a function, you **call it using its name followed by parentheses**.

```
greet("John");
```

Question 14: What is the difference between a function declaration and a function expression?

Feature	Function Declaration	Function Expression
Hoisting	Yes, can be called before definition	No, must define first
Name	Must have a name	Can be anonymous
Usage	Standard reusable functions	Often used for callbacks, inline functions

Question 15: Discuss the concept of parameters and return values in functions.

Question 15: Discuss the concept of parameters and return values in functions.

1 Parameters

Parameters are variables that you **pass into a function** to provide input.
They allow functions to work with **dynamic values** rather than fixed ones.

Syntax

```
function greet(name) {
  console.log("Hello " + name);
}
```

2 Return Values

Return value is the **output a function gives back** after execution.
Use the return keyword to send a value back to the caller.

Syntax

```
function add(a, b) {
  return a + b;
}
```

Question 16: What is an array in JavaScript? How do you declare and initialize an array?

An **array** in JavaScript is a **data structure** used to store **multiple values in a single variable**.
The values are stored in an **ordered list** and can be accessed using **index numbers** (starting from 0).

🔷 Declaring an Array

Method 1: Using square brackets (Most common)

```
let fruits = [];
```

Method 2: Using Array constructor

```
let fruits = new Array();
```

🔷 Initializing an Array

Example

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

- "Apple" → index 0
- "Banana" → index 1

- "Mango" → index 2

🔷 Accessing Array Elements

console.log(fruits[0]); // Apple

console.log(fruits[2]); // Mango

Question 17: Explain the methods push(), pop(), shift(), and unshift() used in arrays.

These methods are used to **add or remove elements** from an array.

🔷 push()

👉 Adds one or more elements to the end of an array

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

fruits.push("Mango");

console.log(fruits);

Output:

["Apple", "Banana", "Mango"]

🔷 pop()

👉 Removes the last element from an array

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

fruits.pop();

```
console.log(fruits);
```

Output:

```
["Apple", "Banana"]
```

🔷 shift()

👉 Removes the first element from an array

```
let fruits = ["Apple", "Banana", "Mango"];
fruits.shift();

console.log(fruits);
```

Output:

```
["Banana", "Mango"]
```

🔷 unshift()

👉 Adds one or more elements to the beginning of an array

```
let fruits = ["Banana", "Mango"];
fruits.unshift("Apple");

console.log(fruits);
```

Output:

```
["Apple", "Banana", "Mango"]
```

Question 18: What is an object in JavaScript? How are objects different from arrays?

What is an Object in JavaScript?

An **object** in JavaScript is a **collection of properties**, where each property is a **key–value pair**.
Objects are used to represent **real-world entities** with characteristics and behaviors.

Example

```
let person = {
  name: "John",
  age: 25,
  city: "Delhi"
};
```

- name, age, city → **keys (properties)**
- "John", 25, "Delhi" → **values**

🔷 What is an Array?

An **array** is a special type of object used to store **multiple values in an ordered list**, accessed using **index numbers**.

Example

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

🔁 Differences Between Objects and Arrays

Feature	Object	Array
Structure	Key–value pairs	Indexed list
Access	Using keys	Using index (0,1,2…)
Order	No guaranteed order	Ordered
Use case	Real-world entities	Lists of similar items
Data type	object	object (special type)

Question 19: Explain how to access and update object properties using dot notation and bracket notation.

Example Object

```
let student = {
  name: "Amit",
  age: 20,
  course: "JavaScript"
};
```

🔷 1️⃣ Dot Notation

👉 Access property

```
console.log(student.name);   // Amit
console.log(student.age);    // 20
```

👉 Update property

```
student.age = 21;
console.log(student.age);    // 21
```

📌 Use when:

- Property name is **known and fixed**
- Property name is a **valid identifier**

🔷 2️⃣ Bracket Notation

👉 Access property

```
console.log(student["course"]);  // JavaScript
```

👉 Update property

```
student["course"] = "React";

console.log(student.course);     // React
```

Question 20: What are JavaScript events? Explain the role of event listeners.

JavaScript events are actions or occurrences that happen in the browser, such as **clicking a button, typing on the keyboard, or loading a page**.

An **event listener** is used to **detect these events** and execute a function when the event occurs.

Example:

```
button.addEventListener("click", myFunction);
```

Question 21: How does the addEventListener() method work in JavaScript? Provide an example.

The **addEventListener()** method is used to **attach an event handler to an element**.
It listens for a specific event and **executes a function when that event occurs**.

🔷 Syntax

element.addEventListener("event", function);

🔷 Example

<button id="btn">Click Me</button>

<script>

document.getElementById("btn").addEventListener("click", function () {

alert("Button clicked!");

});

</script>

Question 22: What is the DOM (Document Object Model) in JavaScript? How does JavaScript interact with the DOM?

What is the DOM?

The **DOM (Document Object Model)** is a **tree-like representation of an HTML document**.
It represents all HTML elements as **objects**, allowing JavaScript to access and manipulate them.

Each HTML element becomes a **node** in the DOM tree.

🔷 How does JavaScript interact with the DOM?

JavaScript interacts with the DOM by:

- **Accessing elements**
- **Changing content**

- **Modifying styles**
- **Adding or removing elements**
- **Handling events**

Question 23: Explain the methods getElementById(), getElementsByClassName(),and querySelector() used to select elements from the DOM.

getElementById()

- Selects **one element** using its **unique id**
- Returns a **single element**

document.getElementById("title");

Use when the element has a unique id.

🔷 **getElementsByClassName()**

- Selects **all elements** with a given class name
- Returns an **HTMLCollection**

document.getElementsByClassName("box");

Access elements using index:

document.getElementsByClassName("box")[0];

🔷 **querySelector()**

- Selects the **first element** that matches a **CSS selector**
- Very flexible

document.querySelector(".box"); // class

document.querySelector("#title"); // id

document.querySelector("p"); // tag

Question 24: Explain the setTimeout() and setInterval() functions in JavaScript. Howare they used for timing events?

setTimeout()

- Executes a function **once** after a specified delay
- Delay is given in **milliseconds**

Syntax

setTimeout(function, delay);

Example

```
setTimeout(function () {
  console.log("Hello after 2 seconds");
}, 2000);
```

. Used when you want to **delay an action** (e.g., show a message after some time)

🔷 **setInterval()**

- Executes a function **repeatedly** at fixed time intervals
- Interval is also given in **milliseconds**

Syntax

setInterval(function, interval);

Example

```
setInterval(function () {
  console.log("Runs every 1 second");
}, 1000);
```

. Used when you want to **repeat an action** (e.g., digital clock)

Question 25: Provide an example of how to use setTimeout() to delay an action by 2 seconds.

```
<!DOCTYPE html>
<html>
<body>

<button onclick="showMessage()">Click Me</button>
<p id="msg"></p>

<script>
  function showMessage() {
    setTimeout(function () {
      document.getElementById("msg").innerHTML =
        "This message appears after 2 seconds";
    }, 2000);
  }
</script>

</body>
</html>
```

Question 26: What is error handling in JavaScript? Explain the try, catch, and finally blocks with an example.

What is Error Handling in JavaScript?

Error handling in JavaScript is the process of **detecting and managing runtime errors** so that the program does not crash and can continue running smoothly.

JavaScript uses **try, catch, and finally** blocks to handle errors.

🔷 try Block

- Contains code that **may cause an error**

```
try {
  let result = x / 10; // x is not defined
}
```

🔷 catch Block

- Executes **if an error occurs** in the try block
- Handles the error gracefully

```
catch (error) {
  console.log("Error occurred:", error.message);
}
```

🔷 finally Block

- Executes **whether an error occurs or not**

- Used for cleanup code

```
finally {
  console.log("Program execution completed");
}
```

Question 27: Why is error handling important in JavaScript applications?

Why is error handling important in JavaScript applications?

Error handling is important in JavaScript applications because it helps manage runtime errors and prevents the application from crashing.
It ensures a better **user experience**, makes debugging easier, and keeps the application **stable and secure**.

🔷 **Key Reasons (Short Points)**

- Prevents application crashes
- Shows **user-friendly error messages**
- Helps in **debugging and maintenance**
- Keeps program flow under control
- Improves **reliability and stability**