Module 17) Javascript

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

JavaScript is a high-level, interpreted programming language mainly used to create interactive and dynamic web pages. It runs directly in the user's web browser.

Role of JavaScript in Web Development:

JavaScript plays a very important role in modern web development:

- **Makes web pages interactive (buttons, forms, pop-ups).**
- **Allows dynamic content updates without reloading the page.**
- **Validates form data on the client side.**
- **Creates animations, sliders, and effects.**
- **Communicates with servers using AJAX / APIs.**
- **Used in frontend (React, Vue, Angular) and backend (Node.js).**

JavaScript works together with HTML (structure) and CSS (design) to build complete websites.

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

Feature	JavaScript	Python	Java
Execution	Runs in browser	Runs on server	Runs on JVM
Typing	Dynamically typed	Dynamically typed	Statically typed
Main Use	Web development	Data science, backend	Enterprise applications
Compilation	Interpreted	Interpreted	Compiled
Platform	Browser + Server	Server	Platform independent
Syntax	Simple, event-based	Simple, readable	More complex

Key Differences:

- **JavaScript is event-driven and browser-based.**
- **Python focuses more on AI, ML, and backend development.**
- **Java is used for large-scale enterprise applications.**
- **JavaScript can run both client-side and server-side (Node.js).**

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:

The <script> tag is used to add JavaScript code inside an HTML document. It tells the browser to execute JavaScript.

It can be placed:

- **Inside the <head> section**
- **Before the closing </body> tag (recommended for performance)**

Example (Internal JavaScript):

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

---

2. Variables and Data Types

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

Variables in JavaScript:

A variable is used to store data values that can be used and changed in a program.

Ways to Declare Variables:

1. var

- **Old method of declaring variables**
- **Function-scoped**
- **Can be redeclared and updated**

```
var name = "Jay";
```

2. let

- **Introduced in ES6**
- **Block-scoped**
- **Can be updated but not redeclared in the same scope**

```
let age = 21;
```

3. const

- **Block-scoped**
- **Value cannot be changed after assignment**
- **Must be initialized at declaration**

```
const country = "India";
```

Difference Summary:

Keyword	Scope	Redeclare	Update
var	Function	Yes	Yes
let	Block	No	Yes
const	Block	No	No

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

JavaScript has two main categories of data types:

---

1. Primitive Data Types

Data Type	Description	Example
String	Text values	"Hello"
Number	Numbers	10, 3.14
Boolean	True or False	true
Undefined	Declared but not assigned	let x;
Null	Empty value	let y = null;
BigInt	Large integers	12345678901234567890n
Symbol	Unique identifiers	Symbol("id")

---

2. Non-Primitive (Reference) Data Types

Data Type	Example
Object	{ name: "Jay", age: 21 }
Array	[1, 2, 3]
Function	function hello(){}

Example Code:

```
let name = "Jay";        // String
let age = 21;            // Number
```

let isStudent = true; // Boolean

let city; // Undefined

let value = null; // Null

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

Feature	undefined	null
Meaning	Variable declared but no value assigned	Intentional empty value
Type	undefined	object
Assigned by	JavaScript	Programmer
Example	let x;	let y = null;

Example:

let a;

console.log(a); // undefined

let b = null;

console.log(b); // null

---

3. JavaScript Operators

Here are clear, simple, and exam-ready answers for your JavaScript Operators – Theory Assignment 👇

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

Operators are symbols used to perform operations on values or variables in JavaScript.

---

1. Arithmetic Operators

Used to perform mathematical calculations.

Operator	Description	Example
+	Addition	10 + 5 → 15
-	Subtraction	10 - 5 → 5

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

Example:

```
let a = 10;
let b = 5;
console.log(a + b);
```

---

2. Assignment Operators

Used to assign values to variables.

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

Example:

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

---

3. Comparison Operators

Used to compare two values and return true or false.

Operator	Description	Example
==	Equal (value only)	5 == "5" → true

Operator	Description	Example
===	Strict equal (value + type)	5 === "5" → false
!=	Not equal	5 != 3
!==	Strict not equal	5 !== "5"
>	Greater than	10 > 5
<	Less than	5 < 10
>=	Greater than or equal	5 >= 5
<=	Less than or equal	5 <= 10

Example:

```
console.log(10 > 5);
```

---

4. Logical Operators

Used to combine multiple conditions.

Operator	Description	Example
&&	Logical AND	true && false → false
`		`
!	Logical NOT	!true → false

Example:

```
let age = 20;
console.log(age > 18 && age < 25);
```

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

Feature	== (Loose Equality)	=== (Strict Equality)
Comparison	Value only	Value and data type
Type conversion	Yes	No
Recommended	❌ No	✅ Yes

Example:

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

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:

Control flow refers to the order in which statements are executed in a JavaScript program.
It allows the program to make decisions and execute different blocks of code based on conditions.

---

if-else Statement:

The if-else statement is used to execute code based on a condition.

Syntax:

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

Example:

```
let age = 20;

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

Explanation:

- **The condition age >= 18 is checked.**
- **If it is true, the if block runs.**
- **Otherwise, the else block runs.**

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

switch Statement:

A switch statement is used to compare a value with multiple possible cases and execute the matching block.

Syntax:

```
switch(expression) {
  case value1:
    // code
    break;
  case value2:
    // code
    break;
  default:
    // code if no case matches
}
```

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");
}
```

When to Use switch Instead of if-else:

Use if-else	Use switch
Complex conditions	Single variable comparison
Range checks	Fixed values
Logical expressions	Multiple exact matches
Few conditions	Many conditions

---

5. Loops (For, While, Do-While)

Question 1: Explain the different types of loops in JavaScript. Provide a basic example of each.

Loops are used to execute a block of code repeatedly as long as a specified condition is true.

1. for Loop

Used when the number of iterations is known in advance.

Syntax:

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

Example:

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

---

2. while Loop

Used when the number of iterations is not fixed and depends on a condition.

Syntax:

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

Example:

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

---

3. do-while Loop

Similar to while, but the loop executes at least once, even if the condition is false.

Syntax:

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

Example:

```
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?

Feature	while Loop	do-while Loop
Condition Check	Before loop execution	After loop execution
Minimum Execution	May execute zero times	Executes at least once
Use Case	When condition must be checked first	When code must run at least once

Example Difference:

```
let x = 10;
```

```
while (x < 5) {
   console.log("While Loop");
}

do {
   console.log("Do-While Loop");
} while (x < 5);
```

Output:

```
Do-While Loop
```

---

6. Functions

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

Functions in JavaScript:

A function is a block of reusable code that performs a specific task. Functions help reduce repetition and make code organized and modular.

Syntax for Function Declaration:

```
function functionName(parameters) {
   // code to execute
}
```

Example:

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

// Calling the function
greet();
```

Explanation:

- **functionName is the name of the function.**
- **parameters are optional values passed to the function.**

- **The function is called using its name followed by parentheses: greet();.**

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

Feature	Function Declaration	Function Expression
Syntax	function name(){}	const name = function(){}
Hoisting	Hoisted (can be called before declaration)	Not hoisted (must be defined before calling)
Name	Required	Optional (anonymous functions possible)
Example	function add(a,b){ return a+b; }	const add = function(a,b){ return a+b; };

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

Parameters:

- **Parameters are inputs to a function.**
- **They allow functions to work with different data each time they are called.**

Example:

```
function add(a, b) {
   console.log(a + b);
}

add(5, 3); // Output: 8
add(10, 20); // Output: 30
```

Return Values:

- **A function can return a value to the caller using the return keyword.**
- **The returned value can be stored in a variable or used directly.**

Example:

```
function multiply(a, b) {
   return a * b;
}

let result = multiply(4, 5);
console.log(result); // Output: 20
```

7. Arrays

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

Array in JavaScript:

An array is a special variable that can store multiple values in a single variable.
Each value in an array is called an element, and it has an index starting from 0.

Declaring and Initializing an Array:

1. Using square brackets (Recommended):

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

2. Using the Array constructor:

let numbers = new Array(10, 20, 30);

Accessing elements:

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

console.log(numbers[2]); // Output: 30

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

These are commonly used array methods to add or remove elements.

Method	Description	Example
push()	Adds one or more elements to the end of the array	fruits.push("Mango");
pop()	Removes the last element from the array	fruits.pop();
shift()	Removes the first element from the array	fruits.shift();
unshift()	Adds one or more elements to the beginning of the array	fruits.unshift("Grapes");

Example:

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


fruits.push("Mango");     // ["Apple", "Banana", "Orange", "Mango"]

fruits.pop();             // ["Apple", "Banana", "Orange"]
```

fruits.shift(); // ["Banana", "Orange"]

fruits.unshift("Grapes"); // ["Grapes", "Banana", "Orange"]

console.log(fruits);

Output:

["Grapes", "Banana", "Orange"]

---

8. Objects

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

Object in JavaScript:

An object is a collection of key-value pairs, where keys (properties) are strings (or symbols) and values can be any data type including arrays, functions, or other objects.
Objects are used to store and organize data in a structured way.

Example:

```
let student = {
  name: "John",
  age: 21,
  isEnrolled: true
};
```

Difference Between Objects and Arrays:

Feature	Object	Array
Structure	Key-value pairs	Indexed list of elements
Indexing	Access via keys	Access via numeric index
Use Case	Store related properties	Store ordered collection of items
Example	{name: "John", age: 21}	["apple", "banana", "cherry"]

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

1. Dot Notation:

- **Access or update property using a dot . followed by the property name.**

```
let student = {name: "John", age: 21};


// Access property

console.log(student.name); // Output: John


// Update property

student.age = 22;

console.log(student.age); // Output: 22
```

---

2. Bracket Notation:

- **Access or update property using square brackets [] with the property name as a string.**
- **Useful when the property name contains spaces or special characters or is dynamic.**

```
let student = {name: "John", age: 21};


// Access property

console.log(student["name"]); // Output: John


// Update property

student["age"] = 23;

console.log(student["age"]); // Output: 23


// Dynamic property

let prop = "name";

console.log(student[prop]); // Output: John
```

---

9. JavaScript Events

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

JavaScript Events:

A JavaScript event is an action that occurs in the browser, usually as a result of user interaction.
Events allow you to make web pages interactive and responsive.

Common events include:

- **click – when a user clicks a button or link**
- **mouseover – when the mouse pointer moves over an element**
- **keydown – when a key is pressed**
- **load – when the page finishes loading**

Role of Event Listeners:

- **An event listener is a function that waits for an event to occur and then executes code in response.**
- **Event listeners separate HTML from JavaScript, making code cleaner and easier to maintain.**

Example:

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


<script>
document.getElementById("myButton").addEventListener("click", function() {
  alert("Button clicked!");
});
</script>
```

- **Here, the event listener waits for the click event on the button and executes the function.**

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

addEventListener() Method:

The addEventListener() method attaches an event handler to an element without overwriting existing event handlers.

Syntax:

```
element.addEventListener(event, function, useCapture);
```

- **event – the type of event (e.g., "click", "mouseover")**
- **function – the function to execute when the event occurs**
- **useCapture (optional) – determines event propagation**

Example:

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

<script>

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

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

alert("Hello! You clicked the button.");

});

</script>

Explanation:

- **When the button with id "btn" is clicked, the anonymous function runs and shows an alert.**
- **Using addEventListener allows multiple events to be attached to the same element, unlike onclick which can be overwritten.**

---

10. DOM Manipulation

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

DOM (Document Object Model):

- **The DOM is a programming interface for HTML and XML documents.**
- **It represents the page structure as a tree of objects, where each element, attribute, and text is a node.**
- **Using the DOM, JavaScript can read, modify, add, or delete elements and content dynamically on a web page.**

How JavaScript Interacts with the DOM:

- **JavaScript can access HTML elements using selectors.**
- **It can change content, styles, and attributes.**
- **It can add event listeners to make the page interactive.**

Example:

document.getElementById("demo").innerHTML = "Hello, DOM!";

- **This changes the content of an element with id="demo".**

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

1. getElementById()

- **Selects a single element by its unique id.**
- **Returns the element object.**

Syntax:

```
let element = document.getElementById("myId");
```

Example:

```
document.getElementById("demo").innerHTML = "Hello World!";
```

---

2. getElementsByClassName()

- **Selects all elements with a specific class.**
- **Returns an HTMLCollection (array-like object).**

Syntax:

```
let elements = document.getElementsByClassName("myClass");
```

Example:

```
let items = document.getElementsByClassName("item");
items[0].innerHTML = "First Item";
```

---

3. querySelector()

- **Selects the first element that matches a CSS selector.**
- **Very flexible; can select by id, class, tag, or combinations.**

Syntax:

```
let element = document.querySelector("selector");
```

Example:

```
document.querySelector(".item").style.color = "red";  // First element with class "item"
document.querySelector("#demo").innerHTML = "Updated!";
```

---

11. JavaScript Timing Events (setTimeout, setInterval)

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

setTimeout()

- **setTimeout() is used to execute a function once after a specified delay.**
- **The delay is measured in milliseconds (1000 ms = 1 second).**

Syntax:

setTimeout(function, delay);

Use cases:

- **Show a message after some time**
- **Delay animations or actions**
- **Auto-hide notifications**

---

setInterval()

- **setInterval() is used to execute a function repeatedly at fixed time intervals.**
- **The function keeps running until it is stopped using clearInterval().**

Syntax:

setInterval(function, interval);

Use cases:

- **Digital clocks**
- **Counters and timers**
- **Automatic image sliders**

Example:

```
setTimeout(() => {
   console.log("This runs after 2 seconds");
}, 2000);


setInterval(() => {
   console.log("This runs every 1 second");
```

```
}, 1000);
```

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

Example:

```
<!DOCTYPE html>
<html>
<head>
  <title>setTimeout Example</title>
</head>
<body>
  <h2>Wait for 2 Seconds</h2>

  <script>
    setTimeout(function() {
      alert("This message appears after 2 seconds");
    }, 2000);
  </script>
</body>
</html>
```

Explanation:

- **The function inside setTimeout() runs after 2000 milliseconds.**
- **The alert is delayed by 2 seconds.**

---

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:

Error handling is the process of detecting, managing, and responding to errors that occur while a JavaScript program is running.
It prevents the program from crashing and allows graceful handling of unexpected situations.

---

try, catch, and finally Blocks:

try

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

catch

- **Executes when an error occurs in the try block.**
- **Receives the error object.**

finally

- **Executes whether an error occurs or not.**
- **Used for cleanup tasks.**

Syntax:

```
try {
   // code that may cause an error
} catch (error) {
   // code to handle the error
} finally {
   // code that always runs
}
```

Example:

```
try {
   let x = 10;
   console.log(x / y); // y is not defined
} catch (error) {
   console.log("An error occurred:", error.message);
} finally {
   console.log("Program execution completed.");
}
```

Output:

```
An error occurred: y is not defined
```

Program execution completed.

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

Importance of Error Handling:

- **Prevents application crashes.**
- **Improves user experience by showing friendly error messages.**
- **Helps developers debug issues easily.**
- **Maintains application stability.**
- **Ensures important code (like closing resources) always executes.**
