**Java script assignment**

**JavaScript Introduction**

**QUESTION: 1** **what is java script? Explain the role of java script in web development.**

**Answer:** JavaScript is a high-level, interpreted programming language that is primarily used to create dynamic and interactive effects within web browsers.

Here’s a closer look at the various roles JavaScript plays in web development:

* Client-side interactivity

1)dynamic updates

2)event handling

* Manipulating the DOM (document object model)
* Asynchronous communication
* Single page applications (SPAs)
* From validation
* Animation and visual effects
* Backend development with node.js
* Progressive web apps
* Cross browser compatibility
* Integrating external



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

**Answer:** JavaScript, Python, and Java are all popular programming languages, but they differ significantly in their design, use cases, and behaviour. the main differences between JavaScript and Python/Java:



* **Different for java script, java, or python**
* **Java script:** JavaScript is essential for web development and is primarily used for front-end scripting but can also be used on the server-side with Node.js.
* **Python:** python is known for its simplicity and is often used for scripting, data science, machine learning, and web development.
* **Java:** Java is a more structured, statically typed language used for large-scale, high-performance applications, especially in enterprise environments.



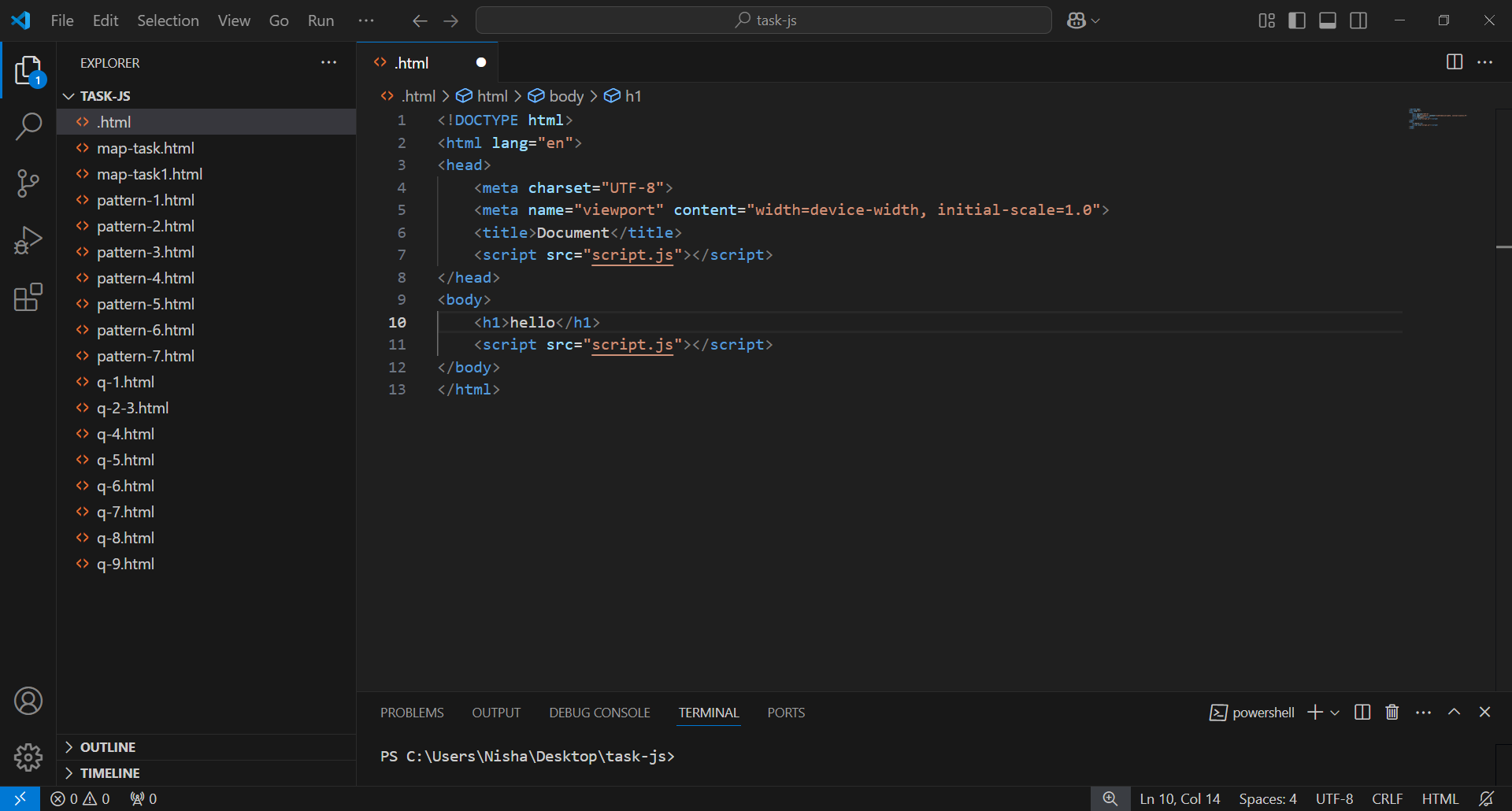
**QUESTION 3: Discuss the use of <script> tag in HTML. How can you link an external JavaScript file to an HTML document?**

**Answer:** The <script> tag in HTML is used to define and embed client-side JavaScript code in an HTML document.

**How can you link an external JavaScript file to an HTML document**

Steps to Link the JavaScript File:

1. Create the JavaScript File: Save your JavaScript code in a file with a .js extension, e.g., script.js.
2. Link the JavaScript File to Your HTML:
   * Use the <script> tag in your HTML document.
   * Place it either inside the <head> or at the bottom of the <body> tag.

**Example: **

**Variables and Data Types**

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

**Answer:**

What are variables in JavaScript?

In JavaScript, variables are used to store data that can be referenced and manipulated throughout your code.

How do you declare a variable using var, let, and const?

* 1)var keyword: var keyword is a redeclare and reassign both possible.
* 2)let keyword: let keyword is redeclare is not possible but reassign is possible.
* 3)const keyword: const means fix not change reassign and redeclare both not possible.

**Example of all three:**

1. var variable:
2. var p = 20
3. document.Write('<br> p is ',p)
4. var p = "hello"
5. document.write('<br> p is ',p)

output:

p is 20  
p is hello

1. const variable:
2. const pi = 3.14
3. document.write('<br> PI is ',pi)

output:

PI is 3.14

1. let variable:
2. let name = "text"
3. document.write('<br> name is ',name)
4. name=2400
5. document.write('<br> name is ',name)

output:

name is text  
 name is 2400



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

**Answer:**

Explain the different data types in JavaScript.

JavaScript supports various data types that are used to store values. These data types can be divided into two main categories:

1. Primitive Data Types (immutable)
2. Reference Data Types (mutable)

* **Primitive data types**

**a. Number:**

* Represents both integer and floating-point numbers.
* Example:
* let a = 10
* document.write('<br> a=10 is ',typeof(a))

output:

a=10 is – number

**b. string:**

Represents a sequence of characters enclosed in single quotes ('), double quotes (").

* Example:
* a = "hello"
* document.write('<br> a=hello is ',typeof(a))
* a = 'h'
* document.write('<br> a=h is ',typeof(a))

output:

a=hello is string

a=h is string

**c. Boolean:**

* Represents a logical entity that can have only two values: true or false.
* Example:
* a = true
* document.write('<br> a=true is ',typeof(a))
* a = false
* document.write('<br> a=false is ',a)

output:

a=true is Boolean

a=false is Boolean

**d. Undefined:**

* Represents a variable that has been declared but not yet assigned a value.
* Example:
* a;
* console.log(a)
* document.write('<br> undefind is ',a)

Output:

undefind is null

**e. Null:**

* It's a special object type that is used to indicate "no value" or "empty".
* Example:
* a = null
* document.write('<br> a=null is ',typeof(a))

Output:

a=null is object

* **Reference Data Types**

**a. Object:**

* In JavaScript, an object is a collection of data organized into key-value pairs.
* Example:
* let person = {
* name: "Alice",
* age: 25,
* };
* document.write('<br> ',(person.name))
* document.write('<br> ',person.age)

output:

Alice  
25

**b. Array:**

* An ordered collection of values, which can be of different types. Arrays are special types of objects in JavaScript.
* Example:
* let arr = ["nisha","minal","mansi","zarana","shani","gopi","prasatnta"]
* document.write('<br> ',arr)

output:

nisha, minal, mansi, zarana, shani, gopi, prasatnta



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

**Answer:**

**difference between undefined and null in JavaScript?**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **undefined** | **null** |
| **Meaning** | Variable not initialized or missing value | Explicitly assigned "no value" or "empty" |
| **Type** | undefined | object (a JavaScript bug) |
| |  | | --- | | **Default value** |  |  | | --- | |  | | Automatically assigned by JavaScript | Assigned explicitly by the developer |
| **Use Case** | For uninitialized variables or parameters, return value of a function without return | To indicate an intentional empty or null reference |

**JavaScript Operators**

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

**Answer:**

**1.Arithmetic Operators**

Arithmetic operators perform basic mathematical operations such as addition, subtraction, multiplication, and division.

**Operators:**

* **addition (+)**

Example:

 let a = 10

 let b = 20

        document.write('<br> addition is ',(a+b))

output:

addition is 30

* **minus (-)**

Example:

 let a = 10

 let b = 20

        document.write('<br> minus is ',(a-b))

output:

minus is -10

* **multiplication (\*)**

Example:

let a = 10

let b = 20

        document.write('<br> multiplication is ',(a\*b))

output:

multiplication is 200

* **division (/)**

Example:

let a = 10

let b = 20

        document.write('<br> division is ',(a/b))

output:

division is 0.5

* **modules (%)**

Example:

let a = 10

let b = 20

        document.write('<br> module is ',(a%b))

output:

module is 10

**increment (++)**

Example:

\* Postfix---old value return

let ch = 25

document.write('<br> Ch is ',ch)

let xy = ch++ //postfix --> old value

document.write('<br> Ch is ',ch)

document.write('<br> xy is ',xy)

output:

Ch is 25  
Ch is 26  
xy is 25

\* Prefix---new value return

let ch = 25

        document.write('<br> Ch is ',ch)

let xy = ++ch

         document.write('<br> Ch is ',ch)

         document.write('<br> xy is ',xy)

output:

Ch is 25  
Ch is 26  
xy is 26

* **decrement (--)**

Example:

\* Prefix---new value return

 <script>

    let ch = 25

        document.write('<br> Ch is ',ch)

 let xy = --ch

         document.write('<br> Ch is ',ch)

         document.write('<br> xy is ',xy)

output:

Ch is 25  
Ch is 24  
xy is 24

\* Postfix---old value return

let ch = 25

        document.write('<br> Ch is ',ch)

    let xy = ch--

         document.write('<br> Ch is ',ch)

         document.write('<br> xy is ',xy)

output:

Ch is 25  
Ch is 24  
xy is 25

**2.Assignment Operators:**

Assignment operators are used to assign values to variables. They can also combine operations while assigning values.

* **Assignment (=):**

Example:

let a = 5

        document.write('<br> A is ',a)

output:

A is 5

* **Addition Assignment (+=):**

Example:

 a += 10

        document.write('<br> A is ',a)

output:

A is 15

* Minus Assignment (\*=):

Example:

a \*= 3

        document.write('<br> A is ',a)

output:

A is 13

* **Multiplication Assignment (\*=):**

Example:

a \*= 3

        document.write('<br> A is ',a)

output:

A is 39

* **Division Assignment (/=):**

a /= 1

        document.write('<br> A is ',a)

output:

A is 39

Example:

* **Modulus Assignment (%=):**

a %= 2

        document.write('<br> A is ',a)

output:

A is 1

**3.Comparison Operators:**

Comparison operators are used to compare two values. These operators return a boolean value (true or false).

* **Greater than (>):**

Example:

let a = 10

let b = 20

        document.write('<br> a > b is ',(a>b))

output:

a > b is false

* **Less than (<):**

Example:

let a = 10

let b = 20

        document.write('<br> a < b is ',(a<b))

output:

a < b is true

* **Greater than or equal to (>=):**

Example:

let a = 10

let b = 20

        document.write('<br> a >= b is ',(a>=b))

output:

a >= b is false

* **Less than or equal to (<=):**

Example:

 let a = 10

 let b = 20

        document.write('<br> a <= b is ',(a<=b))

output:

a <= b is true

* **Equal to (==): - check same value**

Example:

let a = 10

let b = 20

        document.write('<br> a == b is ',(a==b))

output:

a == b is false

* **Not equal to (! =): - check not same value**

Example:

let a = 10

let b = 20

        document.write('<br> a != b is ',(a!=b))

output:

a! = b is true

* **Strict Equal to (===): - check same value and datatype**

Example:

 let a = 10

        let b = 20

        document.write('<br> a === b is ',(a===b))

output:

a === b is false

* **Strict Not equal to (! ==): - check not same value**

Example:

 let a = 10

        let b = 20

        document.write('<br> a!== b is ',(a!==b))

output:

a! == b is true

**4.Logical Operators**

Logical operators are used to combine multiple boolean expressions or values. They return a boolean value (true or false).

* **Logical AND (&&):**

Example:

 let a = 24

        let b = 35

        let c = 22

        document.write('<br> a>b && a>c ----',(a>b&&a>c))

output:

a>b && a>c ----false

* **Logical OR (||):**

Example:

 let a = 24

        let b = 35

        let c = 22

        document.write('<br> a>b || a>c --- ',(a>b||a>c))

output:

a>b || a>c --- true

* **Logical NOT (!):**

Example:

let a = 24

        let b = 35

        let c = 22

        document.write('<br> ! a>b || a>c ------',!(a>b||a>c))

output:  
! a>b || a>c ------false



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

**Answer:**

In JavaScript, == and === are both comparison operators used to compare values, but they behave differently when it comes to how they evaluate the values:

1. **Equality operators(==): - check same value**

The == operator compares two values for equality after performing type coercion. JavaScript will try to convert the values to the same type before making the comparison.

Example:

* **==(** **Equal to): - check same value**

Example:

let a = 10

let b = 20

        document.write('<br> a == b is ',(a==b))

output:

a == b is false

1. **=== (Strict Equality Operator): - check same value & datatype**

Example:

 let a = 10

        let b = 20

        document.write('<br> a === b is ',(a===b))

output:

a === b is false

**Control Flow (If-Else, Switch)**

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

**Answer:**

**Control flow** in programming refers to the order in which individual statements, instructions, or function calls are executed or evaluated in a program.

**Explain how if-else statements work with an example.**

An if-else statement in programming allows you to execute certain blocks of code based on whether a condition is true or false. It is a fundamental control flow structure that helps to make decisions in your code.

Example:

et age = 29

        //if\_else

        if(age>=28){

             document.write('<br> you are eligible for vote')

        }else{

            document.write('<br> you are not eligible for vote')

        }

Output:

Eligible for vote



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

**Answer:**

how switch statements work in JavaScript

A switch statement in JavaScript allows you to evaluate an expression and match it against multiple cases. It provides a more readable and efficient way of handling multiple possible conditions compared to using multiple if-else statements.

When should you use a switch statement instead of if-else?

A **switch statement** is used instead of if-else when you need to compare a single variable or expression against multiple distinct values.

Example:

 let day = 3;

switch(day) {

    case 1:

     document.write("Monday");

        break;

    case 2:

    document.write("Tuesday");

        break;

    case 3:

    document.write("Wednesday");

        break;

    case 4:

    document.write("Thursday");

        break;

    case 5:

    document.write("Friday");

        break;

    case 6:

    document.write("Saturday");

        break;

    case 7:

    document.write("Sunday");

        break;

    default:

    document.write("Invalid day");

}

Output:

Wednesday

**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.**

**Answer:**

In JavaScript, loops allow you to repeat a block of code multiple times based on certain conditions. There are three main types of loops: **for loop**, **while loop**, and **do-while loop**. Each serves a specific purpose and is used in different scenarios.

**1. For Loop**

The for loop is used when you know in advance how many times you want to execute a block of code. It is commonly used for iterating over arrays or executing code a specific number of times.

Example:

for(let i=1;i<=10;i++){

            document.write('<br> '+i)

  }

Output:

1

2

3

4

5

6

7

8

9

10

**2. While Loop – Entry controlled loop**

The while loop executes as long as a specified condition evaluates to true. It's useful when the number of iterations is not known in advance, and you want to continue looping until a condition is met.

Example:

 let a = 1

        while(a<=5){

            document.write('<br> ',a)

            a++

        }

Output:

1

2

3

4

5

**3. Do-While Loop :- exit controlled loop**

The do-while loop is similar to the while loop, but with one key difference: the block of code is executed at least once before checking the condition. This is useful when you want the code to run at least once, regardless of whether the condition is initially true or false.

Example:

do{

     document.write('<br> '+a)

     a+=2

    }while(a<=10)

Output:

1

3

5

9



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

**Answer:**

**the difference between a while loop and a do-while loop:**

|  |  |  |
| --- | --- | --- |
| **Feature** | **while loop** | **do-while loop** |
| Condition Evaluation | Before executing the code block | After executing the code block |
| Guaranteed Execution | May not execute if condition is false initially | Executes **at least once** |
| Common Use Case | When you want to repeat code while a condition is true, and it's possible that the code may not run at all if the condition is false. | When you want to ensure the code executes at least once, even if the condition is initially false. |

* **while loop**: Condition is checked first, so if it's false, the loop will **not** run.
* **do-while loop**: The code runs at least once, and then the condition is checked.

**Functions**

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

**Answer:**

A **function** is a named block of code that performs a specific task, can accept input (parameters), and may return an output. Functions allow for efficient, organized, and reusable code.

**Declaring :**

* Defines a function, specifying what it will do.
* Informs the compiler about a function's name, parameters, data types, and return type.
* The function declaration doesn't allocate storage for the function, but it does help the compiler check the syntax and ensure the type and name are correct.

Example:

 function functionName(parameter1, parameter2, ...) {

    // Code block to be executed

}

**Calling:**

* Calls a function to be executed by the compiler.
* The function call operator, (), is used after the function name.
* Executes the code inside the function by referring to its name and passing any required arguments.

Example:

functionName(arguments);



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

**Answer:**

Function Declaration: A function defined with the function keyword and has a name. It is hoisted, so you can call it before its declaration in the code.

Function Expression: A function defined and assigned to a variable. It is not hoisted, meaning you can only call it after it is defined.

**differences between a function declaration and a function expression:**

|  |  |  |
| --- | --- | --- |
| Feature | Function Declaration | Function Expression |
| Syntax | he function keyword followed by a name, parentheses, and a block of code. | A function is defined and assigned to a variable |
| Hoisting | Function declarations are **hoisted**, meaning the function can be called before the line | Function expressions are **not hoisted**. The function can only be called **after** the line |
| Function Name | A function has a **name** (e.g., greet) as part of its declaration. | A function can be **anonymous** (without a name) or **named**. |
| Assigned to a Variable | It is **not assigned** to a variable. It is directly declared. | It is **assigned to a variable** or a constant. |
| Availability | t is available **throughout the entire scope.** | It is available only after the line where the function is assigned to the variable. |
| Scope | The function is added to the current scope when the code is hoisted. | The function is **available only after the assignment** to the variable. |
| Use in Conditional | Cannot be used inside a conditional statement as it would result in syntax issues. | Can be used inside conditional statements because it is assigned to a variable. |
| Example | if (true) {      function greet() {}  // SyntaxError  } | if (true) {  const greet = function() {};  } |



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

**Answer:**

1) Parameters:

* Definition: Placeholders for values that are passed into the function when called.
* Usage: Allow functions to be dynamic and flexible by accepting different inputs.

Example:

function add(a, b) {

    return a + b;  // a and b are parameters

}

let sum = add(5, 10);  // 5 and 10 are arguments

document.write('<br> ',sum);

output:

15

2) Return Values:

* Definition: The result of a function's operation, which is sent back to the caller.
* Usage: Allows the function to provide a result that can be used elsewhere in the program.
* Implicit return: If no return statement is provided, the function implicitly returns undefined.

Example:

function add(a, b) {

return a + b;  // Returning the sum of a and b

}

let result = add(5, 10);  // result stores the return value of the add function

document.write('<br> ',result);

output:

15

**Arrays**

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

**Answer:**

in array in JavaScript is a list-like object used to store multiple values in a single variable.

**Key features of array:**

* Indexed: Arrays in JavaScript are indexed by numbers starting from 0
* Dynamic: JavaScript arrays can grow or shrink in size as needed.
* Heterogeneous: Arrays can store elements of different data types (e.g., strings, numbers, objects).

How do you declare and initialize an array

* **Array Literal Syntax (Preferred):**

This is the most common way to declare and initialize an array.

Example:

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

console.log(fruits);

output:

[“apple”, “banana”, “cherry”];

**Array Constructor:**

This method allows you to declare and initialize an array using the Array constructor.

Example:

let numbers = new Array(1, 2, 3, 4, 5);

console.log(numbers);

output:

[1,2,3,4,5]



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

**Answer:**

push (): Adds one or more elements to the end of the array.

Example:

let a = [1,2,3,4,5]

        document.write('<br> Array is '+a)

a.push(25)

        document.write('<br> Array is '+a)

Output:

Array is [1,2,3,4,5,25]

pop (): Removes the last element from the array.

Example:

let a = [1,2,3,4,5,25]

        document.write('<br> Array is '+a)

a.pop(25)

        document.write('<br> Array is ',a)

Output:

Array is [1,2,3,4,5]

shift (): Removes the first element from the array.

Example:

let a = [1,2,3,4,5]

        document.write('<br> Array is '+a)

 a.shift(1)

        document.write('<br> ',a)

Output:

Array is [2,3,4,5]

unshift (): Adds one or more elements to the beginning of the arra

Example:

let a = [1,2,3,4,5]

        document.write('<br> Array is '+a)

 a.unshift(33)

        document.write('<br> ',a)

Output:

Array is [33,1,2,3,4,5]

**Objects**

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

**Answer:**

An object in JavaScript is a data structure that stores data in key-value pairs.

Both **objects** and **arrays** are used to store collections of data in JavaScript, but they differ in their structure, use cases, and how data is accessed.

* **How Are Objects Different from Arrays?**

|  |  |  |
| --- | --- | --- |
| Feature | Object | Array |
| Purpose | Store collections of data in **key-value pairs** | Store **ordered lists** of data. |
| Indexing | Uses **keys** which are typically strings or Symbols. | Uses **numeric indexes** starting from “0”. |
| Accessing Values | Values are accessed using **keys**: object. Key or object["key"]. | Values are accessed using **numeric indexes**: array [0], array [1], etc. |
| Type of Data Stored | Can store any data type as values, including objects, functions, etc. | Typically stores **ordered lists** of items, which can be of any data type numbers, strings, objects, etc.. |
| |  | | --- | | **Use Case** |  |  | | --- | |  | | Suitable for representing real-world entities with properties e.g., a person object with name, age, etc. | Suitable for representing lists or collections of items e.g., a list of numbers or a list of strings. |
| Methods | |  | | --- | | Objects can have **methods** functions as properties. |  |  | | --- | |  | | Arrays come with built-in methods for managing data, like. push (), .pop(), .map(), .filter(), etc. |

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

**Answer:**

* **Dot notation:**

Dot notation is the most used way to access or modify object properties. You use a dot **(.)** followed by the property name.

* **Bracket Notation:**

Bracket notation allows you to access or modify object properties by using square brackets **([])** and passing the property name as a string (either directly or as a variable).

**JavaScript Events**

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

**Answer:**

In JavaScript, events are actions or occurrences that happen in the system you are programming, often as a result of the user interacting with webpage. These actions could include things like clicking a button, moving the mouse, pressing a key on the keyboard, or loading a webpage.

The Role of Event Listeners:

An **event listener** is a function that waits for a specific event to occur on a particular HTML element, and when that event occurs, the listener responds by executing a block of code (a callback function).

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

**Answer:**

The addEventListener() method in JavaScript is used to attach an event handler (a function) to a specific HTML element. This method listens for a specified event (such as a click, keypress, or mouse movement) and triggers the associated callback function when the event occurs.

**DOM manipulation**

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

**Answer:**

* The Document Object Model (DOM) is a programming interface provided by web browsers that allows JavaScript (or other programming languages) to interact with the content and structure of HTML or XML documents.
* The DOM allows JavaScript to access and manipulate HTML documents by representing the HTML elements as objects that can be modified, deleted, or created.
* JavaScript interacts with the DOM by manipulating the document's structure, content, and style. This is done using the built-in DOM methods and properties available in JavaScript. Some of the primary tasks JavaScript can perform on the DOM include:

1. Accessing Elements
2. Modifying Elements
3. Creating Elements
4. Event Handling

**• Question 2: Explain the methods getElementById (), getElementsByClassName (), and query Selector () used to select elements from the DOM.**

**Answer:**

1. getElementById ()

* The getElementById () method is used to select an element by its unique id attribute.

1. getElementsByClassName ()

* The getElementsByClassName () method is used to select all elements that have a specific class name.

1. query Selector ()

* The query Selector () method is a more flexible method for selecting elements using **CSS selectors**.

**JavaScript Timing Events (setTimeout, setInterval)**

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

setTimeout () and setInterval () are key JavaScript functions for handling **timing events**, which allows you to schedule actions to happen after a certain delay or at repeated intervals.

**1) setTimeout () for Delayed Events**

setTimeout () is used when you want to execute a function or event after a **specific delay**.

How it works for timing events:

* Delayed Action:

You can use setTimeout () to delay an event for a specific period. For example, if you want to show a message after 3 seconds, or make a background change after a delay.

* One-time Event:

It is typically used when you want a single action to occur after waiting for a certain amount of time.

**2) setInterval () for Repeated Events**

setInterval () is used to execute a function or event **repeatedly at fixed intervals**.

How it works for timing events:

* Repeating Action:

You can use setInterval () to repeatedly run code at regular intervals. This is useful for actions that need to happen periodically, such as updating a countdown timer, refreshing data, or performing animations.

* Repeated Event:

The event happens at the interval you specify until you stop it, using clearInterval().

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

<script>

        setTimeout (function () {

  console.log ("This message appears after 5 seconds.");

}, 2000); // 2000 milliseconds = 2 seconds

</script>

**JavaScript Error Handling**

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