

**Module (JAVASCRIPT BASIC & DOM) – 4**

##### (Basic logic Question)

* 1. What is JavaScript. How to use it?

JavaScript is a versatile programming language commonly used in web development to enhance the user experiences on the internet.

It is primarily utilized for implementing functionalities within our web applications. JavaScript is often hailed as a the "language of the web", is the most important part of modern web development.

*How to use it:*

JavasScript code is inserted between <script> and </script> tags when used in an HTML document.

Scripts can be placed inside the body or the head section of an HTML page or inside both the head and body.

We can also place JavaScript outside the HTML file which can be linked by specifying its source in the script tag.

* 1. How many types of Variable in JavaScript?

In JavaScript, there are primarily three types of variables:

var: var was traditionally used for variable declaration in JavaScript. Variables declared with var are function-scoped or globally-scoped, depending on whether they are declared inside or outside of a function, respectively.

let: Introduced in ECMAScript 2015 (ES6), let allows you to declare block-scoped variables. Variables declared with let are limited to the block

const: Also introduced in ECMAScript 2015 (ES6), const allows you to declare variables whose values are constant and cannot be reassigned. It behaves similarly to let in terms of block-scoping but prohibits reassignment of the variable after initialization.

* 1. Define a Data Types in js?

In JavaScript, variables hold values, and each value possesses a data type that indicates the nature of the stored information. Broadly, JavaScript classifies data types into two categories: Primitive data types and Non-primitive data types.

**Primitive Data Types:**

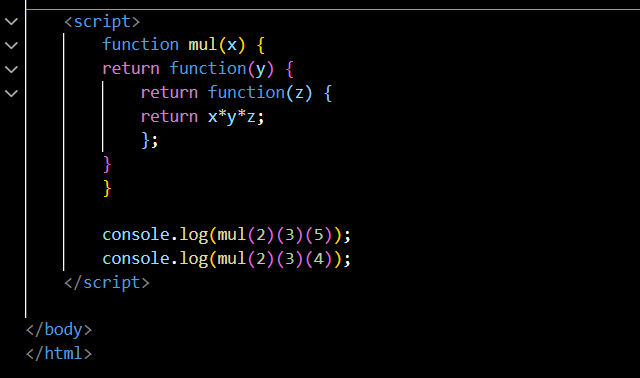
1. Number: Represents numeric values. It can be integers or floating-point numbers.
2. String: JavaScript Strings are similar to sentences. They are made up of a list of characters, which is essentially just an “array of characters, like “Hello ParthNamjshi” etc.
3. Boolean: Represents a logical value indicating true or false.
4. Undefined: Represents a variable that has been declared but has not been assigned a value.
5. Null: Represents the intentional absence of any object value.
6. Symbol: Symbols return unique identifiers that can be used to add unique property keys to an object that won’t collide with keys of any other code that might add to the object..
7. BigInt: Introduced in ECMAScript 2020, represents integers of arbitrary precision.

**Complex Data Types:**

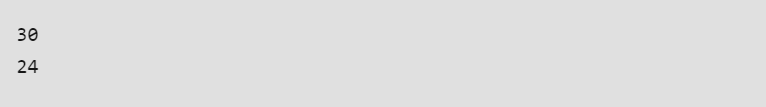
1. Object: Represents a collection of key-value pairs where values can be of any data type, including other objects, functions, and arrays.
2. Array: Represents a list-like collection of elements, which can be of any data type, accessed by numeric indices.
3. Function: Represents reusable blocks of code that can be called with different arguments.
4. Date: Represents dates and times.
   1. Write a mul Function Which will Work Properly When invoked With Following Syntax.

The MUL function is a miniature of the multiplication function. In this function, we call the function that required an argument as a first number, and that function calls another function that required another argument and this step goes on.

The first function’s argument is x, the second function`s argument is y and the third is z, so the return value will be xyz.



OUTPUT:

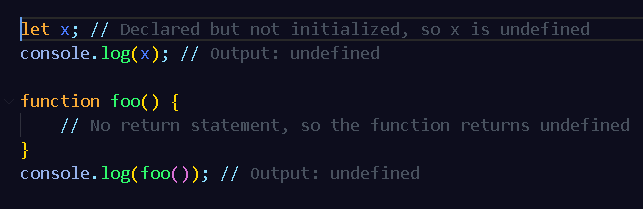


* 1. What the deference between undefined and undeclare in JavaScript?

**Undefined:**

In JavaScript, "undefined" is a primitive data type and also a global variable.

When a variable is declared but not initialized, or when a function doesn't explicitly return a value, JavaScript assigns the value "undefined" to it by default. For example:

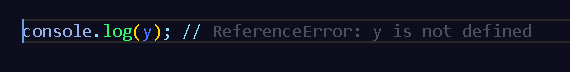


**Undeclared:**

An "undeclared" variable is one that has not been declared within the current scope using any of the variable declaration keywords (var, let, const).

Accessing an undeclared variable results in a ReferenceError.

For example



* 1. Using console.log() print out the following statement: The quote 'There is no exercise better for the heart than reaching down and lifting people up.' by John Holmes teaches us to help one another. Using console.log() print out the following quote by Mother Teresa:
  2. Check if typeof '10' is exactly equal to 10. If not make it exactly equal?
  3. Write a JavaScript Program to find the area of a triangle?
  4. Write a JavaScript program to calculate days left until next Christmas?

|  |
| --- |
| **Q No 6 to 9 Practicle Answer given in separate Practical Folder** |

* 1. What is Condition Statement?

JavaScript Conditional statements allow to execute specific blocks of code based on conditions. If the condition meets then a particular block of action will be executed otherwise it will execute another block of action that satisfies that particular condition.

There are several methods that can be used to perform Conditional Statements in JavaScript.

* if statement: Executes a block of code if a specified condition is true.
* else statement: Executes a block of code if the same condition of the preceding if statement is false.
* else if statement: Adds more conditions to the if statement, allowing for multiple alternative conditions to be tested.
* switch statement: Evaluates an expression, then executes the case statement that matches the expression’s value.
* ternary operator (conditional operator): Provides a concise way to write if-else statements in a single line.
* Nested if else statement: Allow for multiple conditions to be checked in a hierarchical manner.
  1. Find circumference of Rectangle formula : C = 4 \* a ?
  2. WAP to convert years into days and days into years?
  3. Convert temperature Fahrenheit to Celsius? (Conditional logic Question)
  4. Write a JavaScript exercise to get the extension of a filename.?

|  |
| --- |
| **Q No 11 to 14 Practical Answer given in separate Practical Folder** |

* 1. What is the result of the expression (5 > 3 && 2 < 4)?

The expression (5 > 3 && 2 < 4) is a logical AND operation. It checks whether both conditions are true:

- The condition 5 > 3 is true.

- The condition 2 < 4 is also true.

Since both conditions are true, the overall result of the expression is true.

* 1. What is the result of the expression (true && 1 && "hello")?

In JavaScript, the logical AND operator (&&) evaluates expressions from left to right and stops as soon as it encounters a falsy value. If all values are truthy, it returns the last truthy value.

In the expression (true && 1 && "hello"):

- true is truthy.

- 1 is truthy.

- "hello" is truthy.

Since all values are truthy, the result of the expression is the last truthy value, which is "hello".

* 1. What is the result of the expression true && false || false && true?

In JavaScript, the logical AND operator (&&) has higher precedence than the logical OR operator (||). The expression is evaluated from left to right.

Let's break down the expression true && false || false && true:

1. true && false evaluates to false.

2. false || false && true evaluates to false.

So, the overall result of the expression is false.

|  |
| --- |
| **Q No 18 to 29 Practical Answer given in separate Practical Folder** |

##### **(Conditional looping logic Question)**

Q.30 What are the looping structures in JavaScript? Any one Example?

In JavaScript, there are several looping structures commonly used to execute a block of code repeatedly:

1. for loop: This loop executes a block of code a specified number of times.

Example:

JavaScript:

for (let i = 0; i < 5; i++) {

console.log(i);

}

This loop will log the numbers 0 through 4 to the console.

2. while loop: This loop executes a block of code as long as a specified condition

evaluates to true.

Example:

javascript

let i = 0;

while (i < 5) {

console.log(i);

i++;

}

This loop will also log the numbers 0 through 4 to the console.

3. do...while loop: This loop is similar to the while loop, but it will always execute the block of code at least once before checking if the condition is true.

Example:

javascript

let i = 0;

do {

console.log(i);

i++;

} while (i < 5);

This loop will log the numbers 0 through 4 to the console, similar to the previous examples.These are the basic looping structures in JavaScript, each serving different purposes depending on the specific requirements of your code.

Q.31 Write a print 972 to 897 using for loop in JS?

To print numbers from 972 to 897 using a for loop in JavaScript, you can start the loop from 972 and decrement the loop variable until it reaches 897. Here's how you can do it:

javascript

for (let i = 972; i >= 897; i--) {

console.log(i);

}

This loop will start from 972, decrementing by 1 in each iteration, and will continue until it reaches 897, inclusive. It will print each number from 972 down to 897 on separate lines in the console.

Q.32 Write to print factorial of given number?

JavaScript

function factorial(n) {

if (n === 0 || n === 1) {

return 1;

} else {

let result = 1;

for (let i = 2; i <= n; i++) {

result \*= i;

}

return result;

}

}

// Example usage:

console.log(factorial(5)); // Output: 120

Q.33 Write to print Fibonacci series up to given numbers?

javascript

function fibonacciSeries(limit) {

let fibArray = [0, 1];

while (fibArray[fibArray.length - 1] + fibArray[fibArray.length - 2] <= limit) {

fibArray.push(fibArray[fibArray.length - 1] + fibArray[fibArray.length - 2]);

}

return fibArray;

}

// Example usage:

const limit = 50;

const fibonacci = fibonacciSeries(limit);

console.log("Fibonacci series up to", limit, ":", fibonacci.join(', '));

Q.34 Write to print number in reverse order e.g.: number = 64728 ---> reverse

=82746 in JS?

javascript

function reverseNumber(number) {

const reversedNumber = parseInt(number.toString().split('').reverse().join(''));

return reversedNumber;

}

// Example usage:

const number = 64728;

const reversed = reverseNumber(number);

console.log("Original number:", number);

console.log("Reversed number:", reversed);

Q.35 Write a program make a summation of given number (E.g., 1523 Ans: - 11) in

JS?

javascript

function digitSum(number) {

let sum = 0;

const numStr = number.toString();

for (let i = 0; i < numStr.length; i++) {

sum += parseInt(numStr[i]);

}

return sum;

}

// Example usage:

const number = 1523;

const sum = digitSum(number);

console.log("Summation of digits of", number, "is:", sum);

Q.36 Write a program you have to make a summation of first and last Digit. (E.g.,

1234 Ans: - 5) in JS?

javascript

function sumFirstAndLastDigit(number) {

const numStr = number.toString();

const firstDigit = parseInt(numStr[0]);

const lastDigit = parseInt(numStr[numStr.length - 1]);

return firstDigit + lastDigit;

}

// Example usage:

const number = 1234;

const sum = sumFirstAndLastDigit(number);

console.log("Summation of the first and last digits of", number, "is:", sum);

|  |
| --- |
| **Q No 37 to 39 Practical Answer given in separate Practical Folder** |

**(Array and object Question)**

|  |
| --- |
| **Q No 40 to 46 Practical Answer given in separate Practical Folder** |

Q.47 What is the drawback of declaring methods directly in JavaScript objects?

One drawback of declaring methods directly in JavaScript objects is that it can lead to code duplication and increased memory consumption, especially if multiple instances of the object are created.

When methods are declared directly within an object, each instance of that object will have its own copy of those methods. This means that if you have multiple instances of the same object, each instance will consume memory to store its own copy of the methods, even though the logic of those methods is identical across all instances.

This can become inefficient, especially in scenarios where memory usage is a concern, such as in memory-intensive applications or applications running on devices with limited resources.

To mitigate this drawback, you can use prototypes in JavaScript to define methods for object constructors. By defining methods on the prototype, all instances of the object share the same copy of those methods, reducing memory consumption and eliminating code duplication.

|  |
| --- |
| **Q No 48 to 49 Practical Answer given in separate Practical Folder** |

Q.50 What is the drawback of declaring methods directly in JavaScript objects?

One drawback of declaring methods directly within JavaScript objects, especially when creating multiple instances of those objects, is that it can lead to increased memory consumption and reduced performance.

When methods are defined directly within an object, each instance of that object will have its own copy of those methods. This means that memory will be allocated for each instance to store the method definitions, even though the logic of those methods is identical across all instances.

Additionally, if you modify the method of one instance, it does not affect the method of other instances because each instance has its own separate copy of the method. This can lead to code duplication and maintenance issues if you need to update the method logic for all instances.

To mitigate these drawbacks, you can use prototypes in JavaScript. By defining methods on the prototype of an object constructor, you ensure that all instances of that object share the same copy of those methods. This reduces memory consumption and eliminates code duplication, leading to better performance and easier maintenance.

|  |
| --- |
| **Q No 51 to 56 Practical Answer given in separate Practical Folder** |

Q.57 how many type of JS Event? How to use it ?

JavaScript events are actions that occur as a result of user interactions or system events in a web page. There are various types of JavaScript events, including:

1. Mouse Events: These events occur when the user interacts with the mouse, such as clicking, hovering, or moving the mouse pointer.

- click: Occurs when the mouse button is clicked.

- dblclick: Occurs when the mouse button is double-clicked.

- mouseover: Occurs when the mouse pointer moves over an element.

- mouseout: Occurs when the mouse pointer moves out of an element.

- mousemove: Occurs when the mouse pointer is moved while it is over an element.

- mousedown: Occurs when the mouse button is pressed down on an element.

- mouseup: Occurs when the mouse button is released over an element.

2. Keyboard Events: These events occur when the user interacts with the keyboard.

- keydown: Occurs when a key is pressed down.

- keyup: Occurs when a key is released.

- keypress: Occurs when a key is pressed and released.

3. Form Events: These events occur when the user interacts with HTML form elements.

- submit: Occurs when a form is submitted.

- change: Occurs when the value of an input element changes (e.g., input field, select box).

- focus: Occurs when an element receives focus.

- blur: Occurs when an element loses focus.

4. Window Events: These events occur when the window or document is loaded, resized, or scrolled.

- load: Occurs when the window or document is loaded.

- resize: Occurs when the window is resized.

- scroll: Occurs when the document is scrolled.

5. Media Events: These events occur when media elements (such as <audio> or <video>) are played, paused, or ended.

- play: Occurs when the media playback starts.

- pause: Occurs when the media playback is paused.

- ended: Occurs when the media playback has ended.

You can use JavaScript event handlers to respond to these events. Event handlers are functions that are executed when an event occurs. You can attach event handlers to HTML elements using the addEventListener() method, or directly in HTML attributes. Here's an example of attaching an event handler using addEventListener():

javascript

// Get the element

var element = document.getElementById('myElement');

// Attach an event handler for the click event

element.addEventListener('click', function(event) {

// Handle the click event

console.log('Element clicked');

});

Alternatively, you can directly specify event handlers in HTML attributes like this:

html

<button onclick="handleClick()">Click me</button>

In this case, the handleClick() function will be called when the button is clicked.

Q.59 What is Bom vs Dom in JS?

BOM (Browser Object Model) and DOM (Document Object Model) are two important concepts in JavaScript related to manipulating web pages and interacting with the browser environment. Here's a brief overview of each:

1. DOM (Document Object Model):

- The DOM represents the structure of an HTML document as a tree-like structure, where each node represents an element, attribute, or text in the document.

- It provides a way for JavaScript to access, manipulate, and update the content and structure of a web page dynamically.

- With the DOM, you can traverse the document tree, select specific elements, modify their content or attributes, add or remove elements, and handle events.

- The DOM is standardized by the World Wide Web Consortium (W3C), and it's supported by all modern web browsers.

2. BOM (Browser Object Model):

- The BOM represents the browser itself as an object, providing access to browser-specific features and functionalities.

- It includes objects such as window, document, navigator, location, history, screen, etc., which allow JavaScript to interact with various aspects of the browser and the user's environment.

- Unlike the DOM, which deals with the structure and content of web pages, the BOM deals with browser-related tasks such as controlling browser behavior, managing windows and frames, handling cookies, manipulating the browser's history, and accessing information about the user's system and environment.

- The BOM is not standardized by any official organization, and its features may vary between different browsers. As a result, developers often need to write browser-specific code or use feature detection to ensure cross-browser compatibility.

In summary, while the DOM provides a standardized way to interact with the structure and content of HTML documents, the BOM provides browser-specific functionalities and allows JavaScript to interact with the browser environment beyond just the document.

Q.60 Array vs object defences in JS?

In JavaScript, both arrays and objects are commonly used data structures, but they serve different purposes and have distinct characteristics. Here's a comparison of arrays and objects in terms of their usage and characteristics:

1. Arrays:

- Arrays are ordered collections of values, typically used when you have a list of items that need to be accessed by their index.

- They can contain elements of different data types, including numbers, strings, objects, or even other arrays.

- Arrays have numerical indices, starting from 0, which allow for fast random access to elements.

- You can use various built-in array methods for manipulating and iterating over arrays, such as push(), pop(), splice(), forEach(), map(), filter(), etc.

- Arrays are useful for storing lists of related items, such as a list of names, numbers, or objects.

2. Objects:

- Objects are collections of key-value pairs, where each key is a unique string (or symbol) and each value can be of any data type.

- They are typically used to represent entities or complex data structures with named properties.

- Objects allow for easy access to values using their keys (property names), rather than numerical indices.

- You can use dot notation (object.property) or bracket notation (object['property']) to access or modify properties of an object.

- Objects can contain methods (functions) as values, allowing for encapsulation of behavior related to the object.

- They are commonly used for representing real-world entities, such as users, products, or any other structured data.

In terms of defense, both arrays and objects can have certain vulnerabilities, such as:

- Access Control: It's important to control access to properties or elements in both arrays and objects, especially in shared or public code. Exposing sensitive information or allowing unauthorized modifications can lead to security risks.

- Input Validation: Always validate input data when working with arrays or objects, especially when dealing with data from external sources (such as user input or API responses). Failure to validate input can lead to vulnerabilities like injection attacks or unexpected behavior.

- Data Integrity: Ensure data integrity by validating and sanitizing data before performing operations on arrays or objects. This helps prevent errors, inconsistencies, or unexpected behavior in your code.

In summary, while arrays and objects are versatile and powerful data structures in JavaScript, it's important to implement proper defensive practices to ensure the security and reliability of your code. This includes controlling access, validating input, and maintaining data integrity.

javascript

const str = "Hello, world!";

const arr = str.split(','); // Split the string at commas

console.log(arr); // Output: ["Hello", " world!"]

Q.60 Array vs Object defences in JS?

In JavaScript, both arrays and objects are commonly used data structures, but they serve different purposes and have distinct characteristics. Here's a comparison of arrays and objects in terms of their usage and characteristics:

1. Arrays:

- Arrays are ordered collections of values, typically used when you have a list of items that need to be accessed by their index.

- They can contain elements of different data types, including numbers, strings, objects, or even other arrays.

- Arrays have numerical indices, starting from 0, which allow for fast random access to elements.

- You can use various built-in array methods for manipulating and iterating over arrays, such as push(), pop(), splice(), forEach(), map(), filter(), etc.

- Arrays are useful for storing lists of related items, such as a list of names, numbers, or objects.

2. Objects:

- Objects are collections of key-value pairs, where each key is a unique string (or symbol) and each value can be of any data type.

- They are typically used to represent entities or complex data structures with named properties.

- Objects allow for easy access to values using their keys (property names), rather than numerical indices.

- You can use dot notation (object.property) or bracket notation (object['property']) to access or modify properties of an object.

- Objects can contain methods (functions) as values, allowing for encapsulation of behavior related to the object.

- They are commonly used for representing real-world entities, such as users, products, or any other structured data.

In terms of defense, both arrays and objects can have certain vulnerabilities, such as:

- Access Control: It's important to control access to properties or elements in both arrays and objects, especially in shared or public code. Exposing sensitive information or allowing unauthorized modifications can lead to security risks.

- Input Validation: Always validate input data when working with arrays or objects, especially when dealing with data from external sources (such as user input or API responses). Failure to validate input can lead to vulnerabilities like injection attacks or unexpected behavior.

- Data Integrity: Ensure data integrity by validating and sanitizing data before performing operations on arrays or objects. This helps prevent errors, inconsistencies, or unexpected behavior in your code.

In summary, while arrays and objects are versatile and powerful data structures in JavaScript, it's important to implement proper defensive practices to ensure the security and reliability of your code. This includes controlling access, validating input, and maintaining data integrity.

javascript

const str = "Hello, world!";

const arr = str.split(','); // Split the string at commas

console.log(arr); // Output: ["Hello", " world!"]

|  |
| --- |
| **Q No 61 to 66 Practical Answer given in separate Practical Folder** |

* What is JavaScript?

JavaScript is a versatile programming language commonly used in web development to enhance the user experiences on the internet.

It is primarily utilized for implementing functionalities within our web applications. JavaScript is often hailed as a the "language of the web", is the most important part of modern web development.

* What is the use of isNaN function?

The JavaScript isNaN() Function is used to check whether a given value is an illegal number or not. It returns true if the value is NaN else returns false.

* What is negative Infinity?

The negative infinity in JavaScript is a constant value that is used to represent a value that is the lowest available.

This means that no other number is lesser than this value. It can be generated using a self-made function or by an arithmetic operation.

Syntax: Number.NEGATIVE\_INFINITY;

* Which company developed JavaScript?

JavaScript was created at Netscape Communications by Brendan Eich in 1995.

Netscape and Eich designed JavaScript as a scripting language for use with the company's flagship web browser, Netscape Navigator.

* What are undeclared and undefined variables?

Simply undeclared is a variable that is not declared.

And undefined is one that is declared without any value

Forexample:

let x;

console.log(x)

output: undefined

Because the value of x is not defined

console.log(y);

output:Undeclare

Because y is not declared or defined

We can say every undeclared variable is undefined

but every undefined variable is not undeclared.

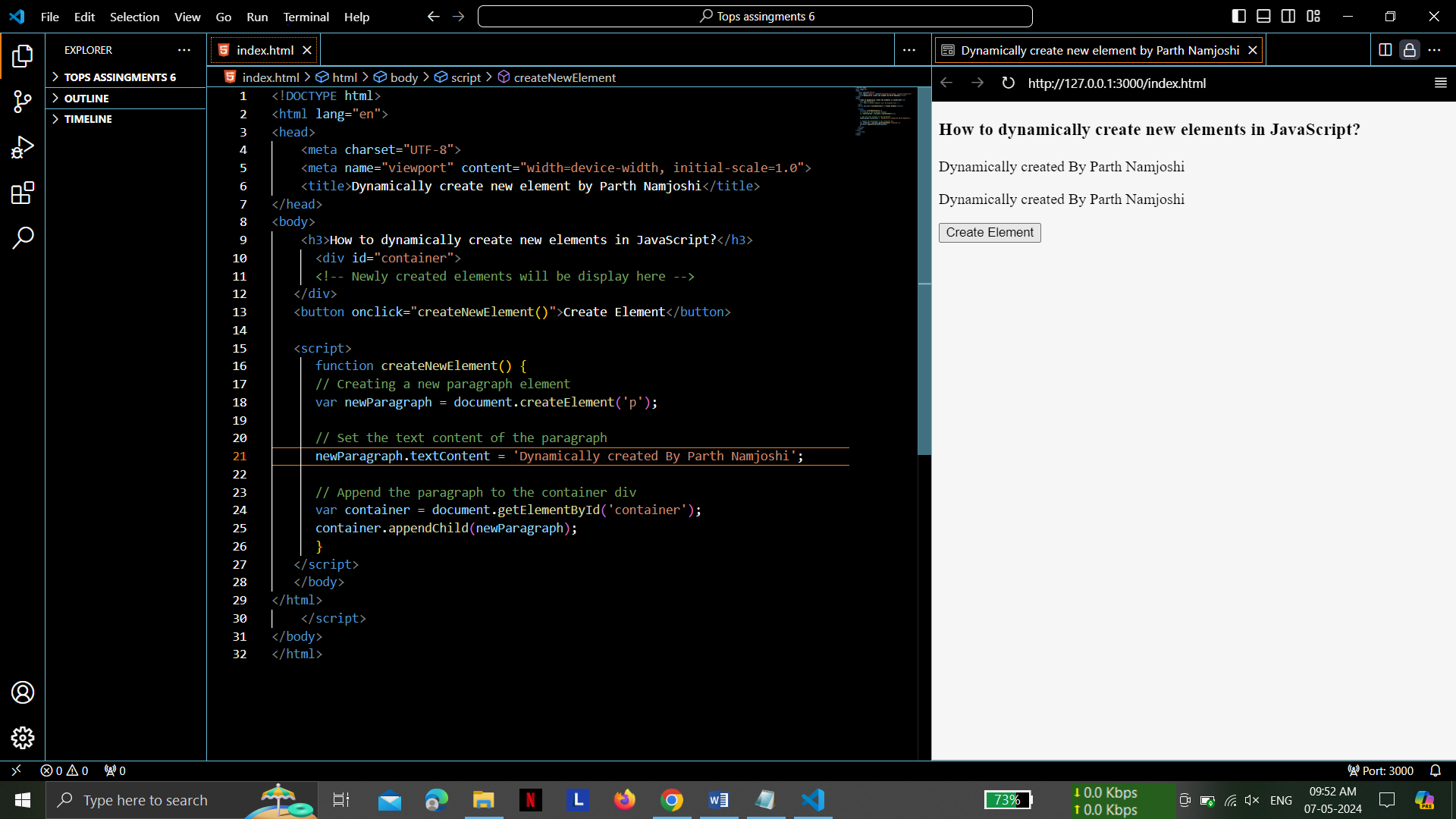
It is possible to declare variable and still it remains undefined, until we give it some value.

* Write the code for adding new elements dynamically?

Dynamic element creation in JavaScript allows to generate new HTML elements on the get-go.

Whether we need to add content to a web page based on user interactions or dynamically generate elements for a specific task, it can help enhance the flexibility and interactivity of web applications.

For example



* What is the difference between ViewState and SessionState?

ViewState and SessionState are used for client-side state management and server-side

state management respectively.

The basic difference between these two is that the

ViewState is to manage state at the client’s end, making state management easy for end-

User.

While SessionState manages state at the server’s end, making it easy to manage

content from this end too.

**ViewState:** It is maintained at only one level that is page-level. Changes made on a single page is not visible on other pages.

Information that is gathered in view state is stored for the clients only and cannot be transferred to any other place.

When view state is used, the values posted of a particular page persist in the browser that the client is using and post back only when the entire operation is done.

The data of the previous page is no longer available when another page is loaded.

Also, Data is not secure in this case because it is exposed to clients.

Encryption can be used for data security.

**SessionState:** It is maintained at session-level and data can be accessed across all pages in the web application.

The information is stored within the server and can be accessed by any person that has access to the server where the information is stored.

It can be used to store information that you wish to access on different web pages.

* What is === operator?

=== Is Strict Equality Operator.

It used to compare two operands and return true if both the value and type of operands are the same.

Since type conversion is not done, so even if the value stored in operands is the same but their type is different the operation will return false.

* How can the style/class of an element be changed?

To change the style or class of an element in JavaScript,

We have a few different options

depending on what exactly we want to modify.

We can change both the style

properties and the class of an HTML element using JavaScript:

***Changing Style Property***

We can directly manipulate the style property of an element to change specific CSS

styles.

By Style Property Directly:

element.style.color = "red";

        element.style.fontSize = "20px";

Using setAttribute:

element.setAttribute("style", "color: red; font-size: 20px;");

Adding or removing Classes

// Adding a class

element.classList.add("new-class");

// Removing a class

element.classList.remove("old-class");

// Toggling a class (adds if not present, removes if present)

element.classList.toggle("active");

***Changing Class***

We can directly manipulate the class of an element using the className or classList property.

Using `className`

// Set the class attribute directly

element.className = "new-class";

Using `classList`

// Adding a class

element.classList.add("new-class");

// Removing a class

element.classList.remove("old-class");

// Toggling a class (adds if not present, removes if present)

element.classList.toggle("active");

* How to read and write a file using JavaScript?

The fs.readFile() and rs.writeFile() methods are used to read and write of a file using javascript.

*Using fs.readFile() function*

The file is read using the function, which is an inbuilt method.

This technique reads the full file into memory and stores it in a buffer.

Syntax:

fs.readFile( file\_name, encoding, callback\_function )

filename:

It contains the filename to be read, or the whole path if the file is saved elsewhere.

encoding:

It stores the file’s encoding. ‘utf8’ is the default setting.

callback function:

This is a function that is invoked after the file has been read.

err:

If there was an error.

data:

The file’s content.

*Using rs.writeFile() function*

The fs.writeFile() function is used to write data to a file in an asynchronous manner. If the file already exists, it will be replaced.

Syntax:

fs.writeFile( file\_name, data, options, callback )

* What are all the looping structures in JavaScript?
* JavaScript Loops are powerful tools for performing repetitive tasks efficiently.
* Loops in JavaScript execute a block of code again and again while the condition is true.
* Loops are integral to maintaining the readability of our code and reducing pressure on the system by minimizing the number of lines needed to accomplish a given task.

This directly enhances the quality of the code, improves runtime efficiency, and saves

considerable time

that would otherwise be spent writing extensive code.

For example,

suppose we want to print “Hello Parth Namjoshi!” 5 times. This can be done using JS Loop easily. In Loop, the statement needs to be written only once and the loop will be executed 5 times as shown below:

for (let i = 0; i < 5; i++) {

    console.log("Hello Parth Namjoshi!");

}

Output:

Hello Parth Namjoshi!

Hello Parth Namjoshi!

Hello Parth Namjoshi!

Hello Parth Namjoshi!

Hello Parth Namjoshi!

There are primarily three types commonly used.

1. For loop

The most fundamental and widely used loop is 'for' loop. The "for" loop in JavaScript allows us to execute a block of code repeatedly until a specified condition is met. It's particularly useful when we know how many times you want to iterate.

A for loop is constituted with four parts: the initialization, the condition, increment/decrement and the code which has to be executed if the conditions are met true.

Syntax:

for (initialization; condition; increment/decrement) {

    // code block to be executed

}

1. While loop

The "while" loop in JavaScript provides another method to execute a block of code repeatedly based on a specific condition.

The primary difference between the 'for' loop and the 'while' loop lies in their syntax, also we use 'while' loop when the number of iterations is not predetermined or when we want to keep looping until a specific condition is met.

Syntax:

while (condition) {

    // code to be executed

}

1. Do-while loop

The 'do-while' loop in JavaScript is similar to the 'while' loop but, with one key difference: the condition is checked after the loop body executes. Due to this property if we use 'do-while' loop, the code inside the loop will execute at least once, regardless of the initial condition.

Syntax:

do {

    // code to be executed

} while (condition);

JavaScript encompasses two additional loops within its domain.

1. for/in loop

JS for/in loop is used to iterate over the properties of an object. The for-in loop iterates only over those keys of an object which have their enumerable property set to “true”.

Syntax

for(let variable\_name in object\_name) {

    // Statement

}

1. for/of loop:

JS for/of loop is used to iterate the iterable objects for example – array, object, set and map. It directly iterate the value of the given iterable object and has more concise syntax than for loop.

for(let variable\_name of  object\_name) {

    // Statement

}

* How can you convert the string of any base to an integer in JavaScript?

Converting a string to an integer in JavaScript means transforming a string that represents a numeric value into an actual integer data type.

This is useful when we need to perform arithmetic operations or comparisons on numeric values stored as strings.

We have different methods to convert the string to integer

* parseInt() Method in JavaScript :

The parseInt() method accepts the string and radix parameter and converts it into an integer.

Syntax: parseInt( Value, radix )

* Number() Method in JavaScript

The number() method is used to convert primitive data type to a number, if it is not convertible it returns NAN.

Syntax: Number( value )

* Unary Operator in JavaScript

The Unary operator(+) is used to convert a string, boolean, and non-string to a number.

* Math.floor() Method in JavaScript

The Math.floor() method is used to convert a string into number.

* What is the function of the delete operator?

The JavaScript delete operator deletes/ removes a property from an object. It removes the property as well as value of the property from the object. It works only with the objects not with the variables or functions.

* What are all the types of Pop up boxes available in JavaScript?

JavaScript provides various popup boxes to notify, warn, or to get input from the user. Popup boxes prevent the user from accessing other aspects of a program until the popup is closed, so they should not be overused.

There are three different kinds of popup methods used in JavaScript.

* Alert box

An alert dialog box is mostly used to inform or alert the user by displaying some messages in a small dialogue box.

* Confirm box

A confirmation box is used to let the user make a choice. When Javascript pops up a confirm box, the user will have to click either "OK" or "Cancel" to proceed to the next step.

* Prompt box

Javascript Prompt Box can be used when we want to get some user input. When Javascript displays a prompt box, the user will see a popup box with an input field and buttons "OK" or "Cancel" to proceed after entering an input value.

* What is the use of Void (0)?

In English, void means nothing. In a programming language, void means return nothing. “javascript: void(0)” is similar to void.

javascript: void(0) means return undefined as a primitive value. We use this to prevent any negative effects on a webpage when we insert some expression.

For example,

in the case of URL hyperlinks. Hyperlinks open by reloading the page when the user clicks on the link. When you need to run some other code in such cases, you can use javascript: void(0).

* How can a page be forced to load another page in JavaScript?

In JavaScript, we can force a page to load another page by using the window.location object.

There are a few methods to achieve this. To force a page to load another page in JavaScript, we have multiple approaches:

Below are the approaches used to force a page to load another page in JavaScript:

* Using window.location.replace:

The replace function is used to navigate to a new URL without adding a new record to the history.

* Using window.location.assign Property

The assign function is similar to the href property as it is also used to navigate to a new URL.

The assign method, however, does not show the current location, it is only used to go to a new location.

Unlike the replace method, the assign method adds a new record to history (so that when the user clicks the “Back” button, he/she can return to the current page).

* What are the disadvantages of using innerHTML in JavaScript?
* **The use of innerHTML very slow:** The process of using innerHTML is much slower as its contents as slowly built, also already parsed contents and elements are also re-parsed which takes time.
* **Preserves event handlers attached to any DOM elements:** The event handlers do not get attached to the new elements created by setting innerHTML automatically. To do so one has to keep track of the event handlers and attach it to new elements manually. This may cause a memory leak on some browsers.
* **Content is replaced everywhere:** Either you add, append, delete or modify contents on a webpage using innerHTML, all contents is replaced, also all the DOM nodes inside that element are reparsed and recreated.
* **Appending to innerHTML is not supported:** Usually, += is used for appending in JavaScript. But on appending to an Html tag using innerHTML, the whole tag is re-parsed.
* **Old content replaced issue:** The old content is replaced even if object.innerHTML = object.innerHTML + ‘html’ is used instead of object.innerHTML += ‘html’. There is no way of appending without reparsing the whole innerHTML. Therefore, working with innerHTML becomes very slow. String concatenation just does not scale when dynamic DOM elements need to be created as the plus’ and quote openings and closings becomes difficult to track.
* **Can break the document:** There is no proper validation provided by innerHTML, so any valid HTML code can be used. This may break the document of JavaScript. Even broken HTML can be used, which may lead to unexpected problems.