06-02-2025 to 07-02-2025

**JavaScript: 2nd and 3rd day notes**

**1. Exception Handling in JavaScript**

JavaScript provides built-in mechanisms to handle runtime errors, preventing script crashes.

**Error Types in JavaScript**

1. **RangeError:** Occurs when a number is out of range.
   * Example: Trying to create an array with an invalid length.
2. **ReferenceError:** Happens when trying to access an undeclared variable.
   * Example: Using x before declaring it.
3. **SyntaxError:** Raised when JavaScript code has an invalid syntax.
   * Example: Missing a closing bracket or writing incorrect expressions.
4. **TypeError:** Happens when an operation is performed on a value of an unexpected type.
   * Example: Trying to call a non-function as a function.
5. **URIError:** Thrown when an invalid URI function is used.
   * Example: Passing an improperly formatted URL to decodeURIComponent().

**Handling Errors using try...catch**

* try block: Contains the code that might throw an error.
* catch block: Catches the error and handles it gracefully.
* finally block: Executes code after try and catch, whether an error occurs or not.
* throw statement: Allows custom errors to be created.

**Example Workflow**

* Try executing a risky operation.
* If an error occurs, catch it and execute fallback logic.
* Ensure cleanup actions run regardless of error occurrence.

**2. The this Keyword in JavaScript**

The this keyword refers to the object that executes the function.

**Behavior of this in Different Contexts**

1. **Global Context:**
   * In browsers, this refers to the window object.
   * In Node.js, this refers to the global object.
2. **Inside Object Methods:**
   * this refers to the object the method belongs to.
3. **Arrow Functions:**
   * Arrow functions do not have their own this; they inherit it from the surrounding scope.
4. **Inside Constructors & Classes:**
   * this refers to the instance of the class being created.
5. **Event Listeners:**
   * In DOM event handlers, this refers to the element that triggered the event.
6. **Explicit Binding (call, apply, bind):**
   * call() and apply() invoke functions with a specified this value.
   * bind() creates a new function with a permanently set this.

**3. Destructuring in JavaScript**

Destructuring makes it easy to extract values from arrays and objects.

**Array Destructuring**

* Allows unpacking array elements into individual variables.
* Useful for swapping values without using temporary variables.

**Object Destructuring**

* Extracts properties from an object and assigns them to variables.
* Supports renaming variables during extraction.

**Default Values in Destructuring**

* Default values can be assigned if the extracted variable is undefined.

**Rest Operator in Destructuring**

* Captures remaining elements of an array or object properties into a new variable.

**4. Callbacks in JavaScript**

A callback function is a function passed as an argument to another function.

**Uses of Callbacks**

* Handling **asynchronous** operations like API requests and event listeners.
* Customizing behavior in **higher-order functions** like map(), filter(), and reduce().

**Types of Callbacks**

1. **Synchronous Callbacks:** Executed immediately within the function that calls them.
   * Example: Callbacks in array methods like forEach().
2. **Asynchronous Callbacks:** Used in operations that complete later.
   * Example: setTimeout(), API calls, database queries.

**Challenges with Callbacks (Callback Hell)**

* Nesting multiple callbacks can lead to complex, unreadable code.
* This problem is solved using **Promises** and **async/await**.

**5. Filtering Even/Odd Numbers using Callbacks**

Functions can be passed as arguments to filter numbers based on conditions.

**Concept of Filtering**

* A function checks if a number is even by verifying divisibility by 2.
* Another function filters numbers using the callback.

**Key Concepts:**

* **Modulus Operator (%)**: Used to check divisibility.
* **Higher-Order Functions**: Functions that accept other functions as arguments.

**6. NaN (Not-a-Number) in JavaScript**

NaN is a special value that represents an uncomputable number.

**When does NaN occur?**

* Performing mathematical operations on non-numeric values.
* Trying to convert invalid strings to numbers.
* Operations like 0/0 result in NaN.

**Properties of NaN**

* NaN is not equal to itself (NaN !== NaN).
* Use isNaN() to check if a value is NaN.
* Number.isNaN() is a more reliable method.

**Conclusion**

Understanding these JavaScript concepts is crucial for writing efficient and error-free code. By handling exceptions, managing this, using destructuring, applying callbacks, filtering numbers, and working with NaN, developers can write clean and scalable JavaScript programs.