**Basic Concepts (crucial for web3)**

* Variables
* Datatypes
* Data Structures
* Equality Comparisons
* Loops and Iterations
* Control Flow
* Functions
* Asynchronous
* Working with APIs
* Classes
* Modules

**Functions**

* Function definition has parameters and function call has arguments.
* For every function call, there is a “call stack”

**Real-Life Applications:**

1. Web Development

* Named/Anonymous Functions: Handling events, form validations, and API calls.
* Arrow Functions: Simplifying syntax for concise code, especially in callbacks and promises.
* IIFE: Encapsulating module code to prevent global namespace pollution.
* Async Functions: Fetching data from APIs, handling asynchronous operations seamlessly.
* Callback Functions: Event handling, asynchronous operations with older APIs.

1. Libraries/Frameworks

* Higher-Order Functions: Used in libraries like Lodash and frameworks like React for composing functions and handling side effects.
* Generator Functions: Managing asynchronous flow with libraries like Redux-Saga.

1. Object-Oriented Programming

* Constructor Functions: Creating and initializing objects in traditional OOP style.

**Arrow Functions**

* **this :** this keyword tells about the current context
* to understand arrow functions, we need to understand “this” keyword and objects
* arrow functions has explicit and implicit return property

**Immediately Invoked Function Expression (IIFE)**

* executes immediately. website start hote hi DB ka connection start hojae – in jese kamon k lye iife use krte
* also used to prevent pollution from Global scope
* arrow functions has explicit and implicit return property

**Memories in JS**

2 types:

1. Heap (non primitives use heap memory) – get reference of original value
2. Stack (primitives datatypes)- jo b variable declare kra h uski copy milti h

**Objects**

2 methods for declaration:

1. Literal li trha -> const obj={ }

* Multiple instances bnjate heyn

1. Constructor ki trha -> Object.create

* Jb constructor se bnate heyn to “singleton” ka object bnta h (means ye apni trha ka ek hi object h)

**Real World Applications of JS Objects:**

1. **Web Development:**

* DOM Manipulation: JavaScript objects are used to represent and manipulate HTML elements on a webpage.
* Event Handling: Objects can be used to manage event listeners and handlers for user interactions.
* AJAX Calls: Objects facilitate asynchronous HTTP requests to fetch or send data to a server without reloading the page.

1. Data Storage and Management:

* Configuration Settings: Objects store configuration settings for applications and libraries.
* State Management: In frameworks like React, objects are used to manage the state of components and applications.
* Data Models: Objects represent data models in applications, allowing for easy manipulation and retrieval of data.

1. APIs and Backend Services:

* Request and Response Objects: In server-side environments like Node.js, objects represent HTTP request and response data.
* Middleware Configuration: Objects configure middleware functions for processing HTTP requests in web servers like Express.js.
* Database Interactions: Objects represent and manipulate data retrieved from databases.

1. User Interfaces:

* Component-Based Frameworks: In frameworks like Vue.js and Angular, objects define components and their properties, methods, and lifecycle hooks.
* Form Handling: Objects manage form data and validation logic.

1. Libraries and Frameworks:

* Utility Libraries: Libraries like Lodash and Moment.js use objects to provide utility functions for data manipulation and date handling.
* Charting Libraries: Objects configure and render charts and graphs in libraries like Chart.js and D3.js.

1. Game Development:

* Game Entities: Objects represent game entities, such as players, enemies, and items, with properties and methods to define their behavior.
* Physics Engines: Objects manage physical properties and interactions between game elements.

1. Machine Learning and Data Analysis:

* Data Structures: Objects store and manipulate datasets, feature sets, and model parameters.
* Library Configurations: Libraries like TensorFlow.js use objects to configure and run machine learning models in the browser.

1. Automation and Scripting:

* Task Runners: Objects define tasks and workflows in automation tools like Gulp and Grunt.
* Scripts: Objects manage the configuration and execution of scripts for various automation tasks.

**Object Destructuring**

Mostly used in React.js to pass “props”

**API- JSON**

Before that, values from backend API comes in XML structure and that was complex, but now in JSON

1. What is JSON?

* JSON is an object with no name: {“keys and values here”}
* Some online tools are available to understand the JSON data including: JSON formatter

**How JS Executes code? + call stack (important for interviews)**

**JavaScript Execution Context**

JavaScript runs file in 2 phases:

1. Global Execution/Environment Context

* Sbse pehle “this” variable k andr Global Context ko rkh dia jata h/ allocate kia jata h. eg; browser k andr this ki value windows object ati h.
* Ye “thread” k upr execute hota h

Note: JavaScript is single threaded

1. Functional/Function Execution Context
2. Eval EC

* We find this one while working with mongoose etc

**Phases:**

1. Memory Creation Phase:
2. Execution Phase

**forEach() vs filter()**

* Foreach value return nhi krta h filter return krta h
* Dono apne andr call back functions lete heyn

**Chaining method**

* We can use method after methods continuously
* Such as variableName.map().filter().reduce()

**DOM**

* We can manipulate dom by usinf document.querySelector() or document.
* Almost All Dom selectors are:

1. **By ID**

document.getElementById('id');

1. **By Class Name**

document.getElementsByClassName('className');

1. **By Tag Name**

document.getElementsByTagName('tagName');

1. **By Name Attribute**

document.getElementsByName('name');

1. **By CSS Selector (single element)**

document.querySelector('selector');

1. **By CSS Selector (all elements)**

document.querySelectorAll('selector');

1. **Select First Child Element**

element.firstElementChild;

1. **Select Last Child Element**

element.lastElementChild;

1. **Select All Child Elements**

element.children;

1. **Select Parent Element**

element.parentElement;

1. **Select Next Sibling Element**

element.nextElementSibling;

1. **Select Previous Sibling Element**

element.previousElementSibling;

1. **Select by Attribute**

document.querySelectorAll('[attribute]');

1. **Select by Attribute with Specific Value**

document.querySelectorAll('[attribute="value"]');

1. **Select by Pseudo-class**

document.querySelectorAll('selector:pseudo-class');

1. **Select Form Elements by Type**

document.querySelectorAll('input[type="text"]');

1. **Select Elements by Data Attribute**

document.querySelectorAll('[data-attribute]');

1. **Select All Descendant Elements**

element.querySelectorAll('selector');

1. **Select Document Root Element**

document.documentElement;

1. **Select Body Element**

document.body;

**Convert HTMLCollection or anything into Array:**

Array.from(‘variable name’)

**Events**

* Events runs sequentially in js.
* Js meyn directly html tag k andr onclick event nhi lgate scaling k time masla hota h. iski jga js ki file m getElementById(‘id).onclick krte heyn

**Purpose of event propagation(third parameter in addEventListener( , , false/true)):**

1. Event capturing (true) ->from top to bottom (used in some scenarios)
2. Event bubbling (false) -> from bottom to top (mostly used)

**To stop event propagation:**

* e.stopPropagation()

**Async JS**

* JS is a Synchronous language by default
* And runs on a single thread
* The above behavior is the default behavior of js

**Execution Control:**

Each execution waits for the last one to complete before executing

**setTimeout():**

* a method which calls an api too
* not a part of core JS. Available through browser

**API**

* fetch naya aya h us se pehle kis trha api fetch krte the?
* Us se pehle xml http request bhejte the
* Console.log() is not a part of core js. Its just a dev tool from browser

**PROMISES**

* The **Promise** object represents the eventual completion (or failure) of an asynchronous operation and its resulting value.
* **Promises have 3 states:**
* pending: initial state, neither fulfilled nor rejected
* fulfilled: meaning that the operation was completed successfully
* *rejected*: meaning that the operation failed
* **There are 2 parts of promises:**
* Create promise
* Consume promise

**fetch()**

* Network based request