**Advanced Javascript**

* this keyword
* call, bind, apply
* Arrows functions
* OOPs using Prototype
* Closures
* Ajax vs Fetch vs Promises vs Observables vs async/await
* Event Loop
  + Memory Heap
  + Call Stack
  + Web APIs
  + Async vs Sync
  + Callback Queue
  + Nano tasks vs Micro tasks vs Macro tasks vs NodeJS Tasks vs Browser tasks (queues)
* Code Execution Context
  + Creation Phase
    - Execution Context Object
    - Creation of Scope Chain
      * What is a Scope
      * Lexical Scoping
      * Global Scope
      * Parent Scope
      * Local Scope
    - Setting value of *this* keyword
  + Execution Phase
    - Execution Context Object Variables Are Set Before Code Execution
    - Code is executed
      * Compiled
      * Byte Code Generation
      * Executed
  + Global Execution Context
  + Function Execution Context
  + Global Execution Context vs Function Execution Context
* Execution Stack
* Check Execution Stack in Browser
* Errors
  + Type Error: ‘f’ is not a function
  + Maximum call stack size exceeded
  + Error stack trace
* OOPS using Prototype
* Module
  + Object Literal ‘{ }’ as Module
  + Closure
  + Modules
  + Module Export
  + Revealing Module
  + Augmented Module
  + Loose Augmentation
  + Tight Augmentation
  + Script Loaders
    - AMD
    - CommonJS
    - UMD
  + Es6 Modules
* Advanced Functions
  + High Order Functions
  + Generator\* Functions
  + Functional Programming
  + Memoization
  + Mixins
  + Composition
  + Currying
  + Math.max
  + forEach, map, filter, reduce, debounce, throttle
  + Hacking above functions
* Developer Tools
  + Tabs
    - Elements
    - Console (REPL)
    - Sources
    - Network
    - Debugging
    - Performance
    - Application
* Code Optimization
* JS Runtime
  + Compile Time vs Runtime
  + Different Engines of Browser
    - CSS Engine
    - HTML Engine
    - JS Engine
* Critical Rendering Path
  + What is the limitation of Browser in terms of loading multiple Resources
  + The role of HTTP protocol in optimization and behavior of HTML resources loading
  + What is Rendering
    - How Browser renders a page
  + What is Blocking vs Non-Blocking Resources
  + What is Above the Fold Resources
  + How to make blocking as non-blocking
    - Async vs Defer
  + Different ways of Placing CSS and Scripts and change in their behavior
  + Bundling, Minification and Obfuscation
    - Why we need it
    - Webpack
    - SystemJS
    - Babel
* CSS as Javascript
* Node JS
  + Event Loop comparison with Browser Event Loop
  + Capabilities of Node JS vs Browser JS
    - Access to the Environment (OS) it is running on
    - Blocking vs Non-Blocking Operations
    - Libuv Library
    - Thread Pool
    - Event Queues
    - What is a ‘Tick’
    - What is a Module
      * Require function and how it searches
        + Index.js
      * What are global modules
      * What happens when a module is created
  + Running parallel processing
    - Spawn vs Fork
    - Monitoring
      * NodeMon
      * Cluster
  + NPM Packages
    - How to create your own package
    - Deploy it to the NPM Registry
    - Use the new package in your own project
    - Package.json file
* Philosophy of SPA
  + Why we need SPA
  + AJAX
    - XMLHttpRequest
    - ActiveX
  + MV\* Pattern
    - MVC
    - MVP
    - MVVM
    - Mix of All the above
  + Templating
  + Bindings
  + Shadow DOM, CSS
  + Web Component
  + Components
    - Every component is a module
    - UI
    - Code Behind
    - Isolation
    - Corresponding HTML and CSS in browser
  + Views
  + Pages
  + State Management
  + Communication b/w Views/Components
  + Refresh will destroy current state and everything and start afresh
  + How the Refresh gives us back the same route when it is not on server
  + Create your own SPA framework (a simple one)
  + Routing
    - History API
    - Swapping Views/Components
  + Gulp vs Grunt vs WebPack vs Latest Module loaders
* SPA frameworks
  + Angular
    - Modules
    - Services
    - Components
      * Angular has its own Compiler apart from JS Engine
      * Life Cycle Events
    - Directives
    - Pipes
    - Template Binding
    - Change Detection
      * How it works
      * Zone.js
      * Monkey Patching
    - State Management
  + React
    - Not Opiniated
    - JSX
      * React has its own compiler along with V8 Engine
    - Class vs Functional Components
      * Stateless
      * StateFull
      * Container
      * High Order Components
    - State vs Props
      * Immutability
      * Why it is efficient
      * Why new state every time
    - Hooks
      * Shortcut of many life-cycle event handlers
    - State Management
      * What is a state
      * How many ways a state can be changed
      * Redux
    - Change Detection
      * What is it
      * Change Detection means State is changed
      * Re-rendering of UI dependent on the changed state
    - Virtual DOM
      * When is Virtual DOM get rendered and re-rendered
    - Routing
* SOLID Principles
  + SRP
  + Open/Close Principle
  + LISKOV Principle
  + Interface Segregation
  + Dependency Inversion
* Inheritance vs composition
  + Downside of Inheritance
  + Association vs Aggregation
  + Specialization
  + Generalization
  + Testable Code
* Interface vs Abstract Class
  + Understand how we break things and when we choose either of the two
* Design Patterns
  + MV\*
  + FlyWeight pattern
  + Decorator
  + Façade
  + Factory
  + Composition
  + Observables
  + Provider
  + Strategy
  + Singleton
* Cross Cutting Concerns
  + Security
    - Authentication
    - Authorization
  + Logging
  + HTTP Requests
  + Threading
  + Error Handling
  + Transactions Handling
  + Data Validation
  + Techniques
    - Decorators
    - Attributes
    - Meta Programming
  + Using Third Party libraries
* Observe famous Libraries and their use of the JS Concepts && Patterns to solve many problems
  + Browser DOM (Flyweight Pattern)
  + Backbone (MVC Pattern)
  + Knockout (MVVM Pattern)
  + Angular (MVC, Strategy, Visitor, Composition, Lazy, Singleton, Factory, Command Patterns )
  + React (HOF, Provider, Module)
  + JQuery (Decorator Pattern, Lazy Initialization Pattern, Composition Pattern, Plugin Pattern)
* Service Workers
* Progress Web Applications
  + Service Workers
  + Life Cycle of Service Workers
  + Application Shell
  + Offline mode
  + Caching
  + Manifest
  + Notifications
  + Updating Service Workers
  + Sync Offline content
  + Limited Resources
  + Storage Mechanism
    - Cookies
    - Caching
    - Cache Busting
    - ETag
    - Local Storage
    - Session Storage
    - Indexed DB
    - Application Cache
* Isomorphic Javascript
  + Factoring out common JS
    - NodeJS specific JS
    - Browser specific JS
  + One time Validations logic and reusing it
* Game Development using Javascript
  + Frame
  + requireAnimationFrame function
* Advanced Browser Rendering
* Server-Side Rendering
* JS Tasks Automation
  + Linting
  + Testing
  + File Watchers
  + Data Seeders
  + Debuggers
  + Comet Technologies
  + Sockets to Browser communication
    - In React
    - In Angular
  + Script Loaders
  + Tasks Runners
  + Module Loaders
  + Building
  + Bundling
  + Tree Shaking
  + Hot Reloading
  + Code Splitting
  + Lazy Loading Modules
* Babel
  + Transpilers
  + Plugins
  + Presets
  + Future JS
  + Shim
  + Polyfills
* Code Smells
* Anti-Patterns