# Advance JavaScript

**Topics:**

* **Scope/ Closure/ Lexical Environment/ Currying/ IIFE.**
* **Object - Deep & Shallow Copy/ “This”/ Symbols.**
* **Functions- Call/ Apply/ Bind | Decorators- Debounce, Throttle, Memoization.**
* **Iterators- Iterables / Generators | Maps / Sets / Infinite Iterators.**
* **Prototypes- Inheritance, Constructor, Polyfills.**
* **Class- Derived class, Private / Protected members, Getter / Setters.**
* **Async-js- Callback / Promise/ Async-Await / Promise API.**

**SCOPES & CLOSURE:**

* Scope refers to the visibility and accessibility of variables, functions, and objects in a particular part of your code during runtime. Scopes in JavaScript can be global scope, function scope, or block scope.

**Block Scopes and Global Scopes:-**

* Block Scope :- The scope is the current context of execution in which values and expression are visible.
* Global Scope :- Any variable / expression which is written outside- i.e not inside any functions, blocks etc. This is shared across files.

**LET:-**

* This creates a block scope.
* Re-declaration is not allowed in same scope.
* Re-assignment is allowed.

**TDZ (Temporal Dead Zone):-**

* The area in which a variable is not accessible. Temporal because it depends on time of execution not position.

**CONST:-**

* This creates a block scope.
* Re-declaration is not allowed.
* Re-assignment is not allowed.
* Must be assigned at declaration.

**Variable Shadowing:-**

* The inner variable "shadows" or overrides the outer variable.

**Var:-**

* It does not have any block scope, and can be re-declared.
* It only had function scope.
* Var are Hoisted, so they can be used before the declaration.

**Module Scope:-**

* In modern JavaScript, a file can be considered as module, where we use export and import syntax to use variable across files.

**Global Object:-**

* The global object is the variable window in case of browser. This helps you to use variable across the scopes. Also, it is the this value for global functions.
  + Window.alert
  + Window.promise.
* In non-browser environment, window doesn’t exist. But other global objects exist.
* Var affects this global object, also function declerations.

**Function Scope:-**

* function scope refers to the visibility and accessibility of variables within a specific function. Variables declared inside a function using the var, let, or const keywords have function scope, meaning they are only accessible within that particular function and not outside of it.

**CLOSURE:**

* Closures in JavaScript are created when a function is defined inside another function and is returned or passed as a value. The inner function has access to the variables, parameters, and functions of the outer function, even after the outer function has completed.

**Lexical Environment:**

* Every variable in JavaScript (within global / block / or function ) has a reference to an object-like data called Lexical environment. This object ( kind of object ) serves as the basis of search for value of variable.

**Hoisting:**

* The movement of variable declaration to top of scope- before execution.
* Function declarations are properly hoisted ( value accessible ).
* Var is hoisted.

**Types of Hoisting:**

* There are two types of hoisting in JavaScript.

**Variable Hoisting:-**

* Variable declarations are hoisted to the top of their scope but not their assignments.
* When you declare a variable using the var keyword, the declaration is hoisted but the initialization remains in place.
* If you access a variable before it is declared, it will have the value undefined.

**Function Hoisting:-**

* Function declarations are hoisted in their entirety, including the function body.
* You can call a function before it is declared in your code.

**IIFE (Immediately invoked function expression):**

* This practice was popular due to var.
* Immediately invoking a function avoids-re-declaration of variable inside it.

**Currying:**

* Currying is a functional programming technique in JavaScript that allows you to transform a function with multiple arguments into a sequence of functions, each taking a single argument. It enables you to create new functions by partially applying arguments to an existing function.

**Object:**

* Variable value is not copied in case of object/arrays.
* But it will copy the reference of that object/array.
* Reference can be changed for a variable.
* It is better to use const always, and whenever you must need to re-assign change it to let.

**Shallow Copy:-**

* A shallow copy of an object refers to creating a new object that has the same properties as the original object, but the property values are copied by reference.
* Many methods can be used to copy object without old reference.
* But problem which these is they just create a copy of properties of that object, but not creating a copy of their references also.

**Deep Copy:-**

* This is a hard problem to solve in past as there can be multiple level of nested objects and there can be references to functions etc also. Few methods which are there:
* JSON.stringify and JSON.parse:- This method utilize the fact that every JSON can be converted to string value (exception of methods/function).
* structuredClone: Browser API which work even for circular references (but functions not supported).

**This:**