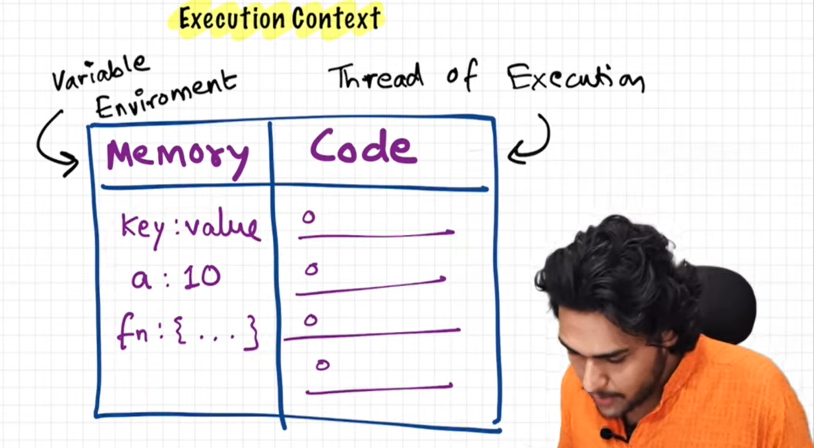
**Namaste Javascript notes**

*--By Adnan Qureshi*

# **Season 1**

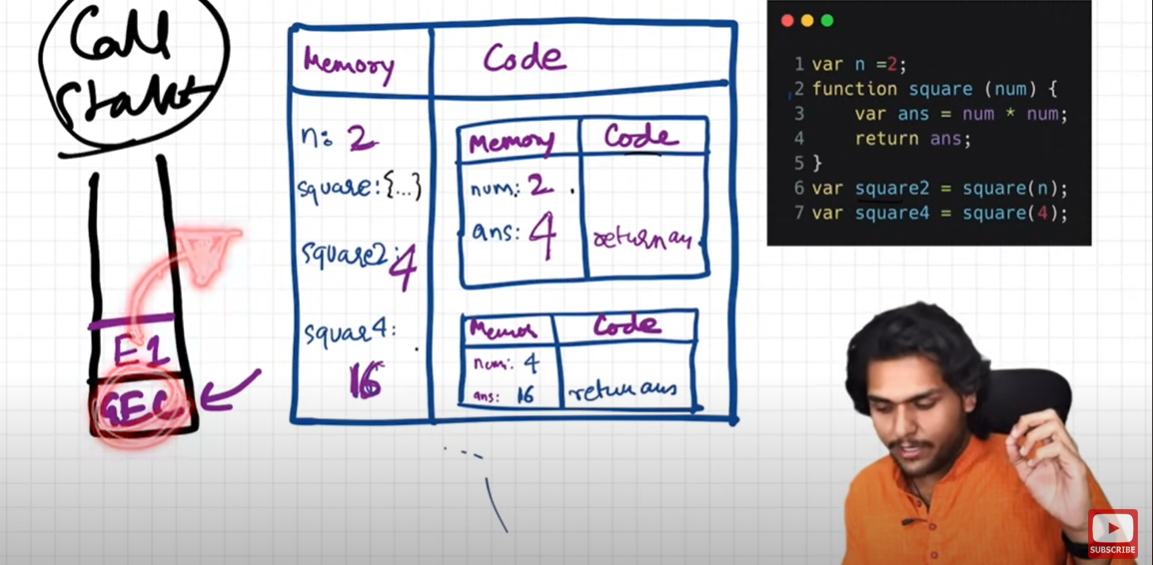
1. **Episode 1: how JavaScript works?**

* “Everything in JavaScript happens inside anExecution context**”**
* JavaScript is a synchronous single-threaded language.



1. **Episode 2: how JavaScript code is executed?**

* JavaScript program executed as a two phase.
* in 1st phase memory allocation (variable environment) is done all the variable and functions allocates the memory in 1st phase.
* Variable has the initial value “undefined”.
* Function contains whole function body as a value.
* 2nd phase is a code execution phase in this code runs once again line by line from start.
* In this phase all the undefined values change it to actual value of variable.
* When function invokes in the code a whole new execution context is created inside a global execution context .
* Then all the steps runs until function returns something.
* Return keyword gives control back to original execution context.
* Call stack maintains the order of execution of execution contexts. It holds global context in the bottom and new order is created whenever new function is invoked and after function ends the orders also gets deleted from the stack. and global execution also deleted after program ends and stack gets empty .



1. **Episode 3: Hoisting in JavaScript.**

* Hoisting is a phenomena in JavaScript by which we can access a variable or function before initialize it.
* in case of variable it gives undefined . Ex: console.log(X); it gives undefined before initialize x.
* In case of function a whole code is put in. and prints actual copy of a function.
* In case of arrow function it allocates undefined because it behaves more like variable and not function.

1. **Episode 4: How function works in JS.**

* Whenever a new function is invoked then a new execution context will also create and it very much limited to that function means function is end then execution context will also ends.
* All the variable under function body is independent and runs separately.
* As soon as new function is invoked it pushed into a call stack and after executed control return goes to the main execution.
* After function execution a whole function goes deleted including function variable memories and also from the call stack.

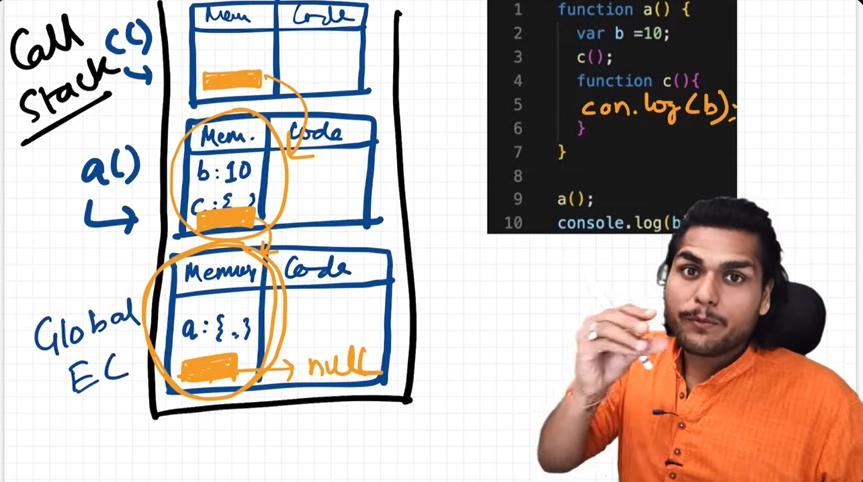
1. **Episode 5: Shortest JavaScript program.**

* JavaScript engine auto creates a global object named Window and we can easily access it anywhere in the program.
* At the global level this keyword refers to the window object (this === window).
* Whenever we creates a variable or function at the global level it gets attached to window object, and we can access it with window. Variable\_or\_function\_name. Eg : (window.x);
* (a || window.a || this.a) is same at the global level.

1. **Episode 6: Undefined VS not defined.**

* Undefined means allocates memories to variable.
* Not defined means variable is not present in memory allocation context.
* Undefined is not equal to empty it is just a placeholder before allocates actual value to the variable.
* JavaScript is loosely(weakly) typed language it is more flexible then other programming languages.

1. **Episode 7: Scope and lexical environment in JS.**

* Scope means where you can access a variable or a function inside the code.
* Scope is directly dependent in the lexical environment.
* Lexical means in order or in hierarchy. in terms of code where a function present in the another function.
* Lexical environment is created whenever a code execution is created, Lexical environment is the local memory along with the lexical environment with its parent. Execution context = local memory + lexical parent.
* In terms of global lexical environment points to null and in terms of another function it points to global.
* Scope chain is nothing but a chain between parent and its parent scope references.
* Depth level function has access to all of its parent (global) level function. But not vice versa.

1. **Episode 8: Let & Const in JS.**

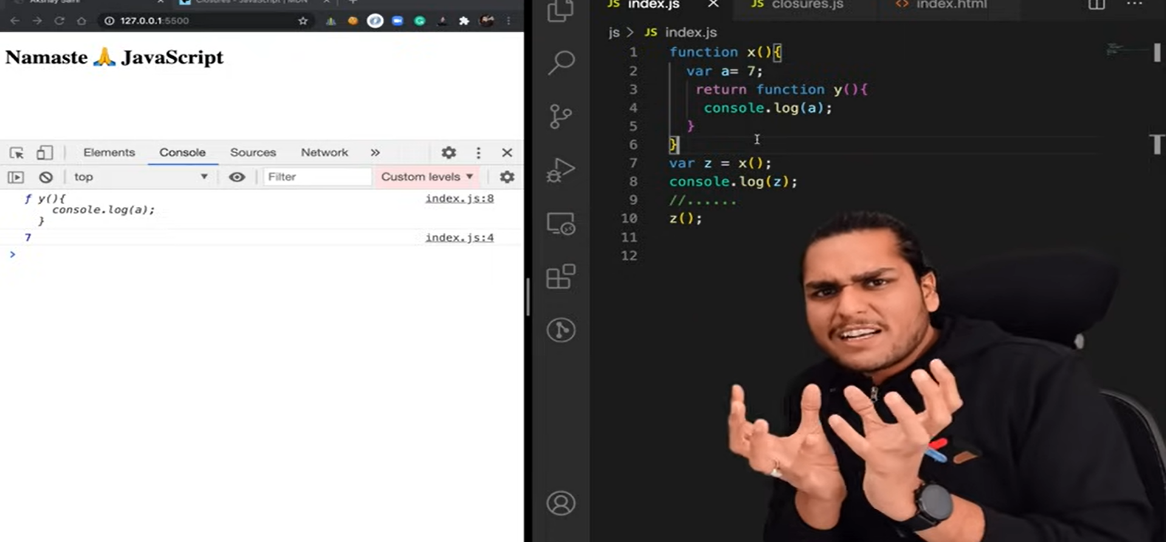
* Both the let & Const is hoisted in JS but in different way.
* From hoisting or memory allocation to declaration some actual value to identifier phase known as temporal dead zone.
* We can not access variable when it is in temporal dead zone. if it do it shows an error of ReferenceError(you can not access before initialization).
* in case of var it gives value undefined, in case of let it shows ReferenceError
* we can’t access let and const with Window.object because it stores in a separate memory space and not in the global memory
* Hence let is slightly strict version of var. we can’t redeclare same variable in let.
* In the case of cost we can’t assign value after declaration. it gives an error. If it is declared it have to initialize right there, and we can’t reassign it after.

1. **Episode 9: Block Scope & Shadowing in JS.**

* Block is something binding under {} it holds some group of variables. We can use some statement together in the block.
* In the block scope we can access variable and function inside the block.
* If a variable present inside the block we can not access it from outside the block.
* Let and const are block scope and var is a globe scope.
* In the case of var if two variable has the same name and pointing to same value then second one shadowed the first one (modified its value to new value). In case of let & const it works opposite.
* Function and arrow function has same scope.

1. **Episode 10: Closures in JS.**

* Function along with its lexical scope forms a closure.
* A closure is the combination of a function bundled together (enclosed) with reference to its surrounding state (the lexical environment.)
* Closure gives you access to an outer function’s scope from an inner function, closure are created every time a function is created.



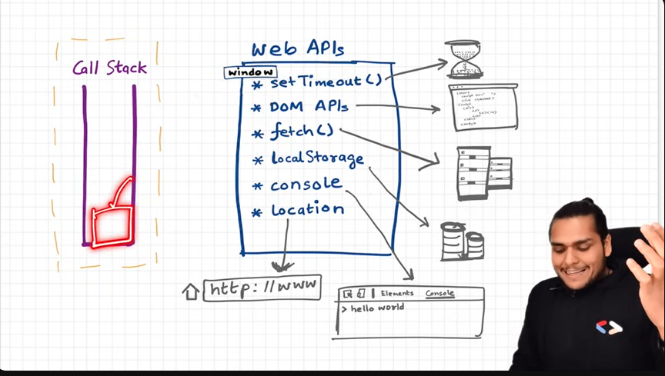
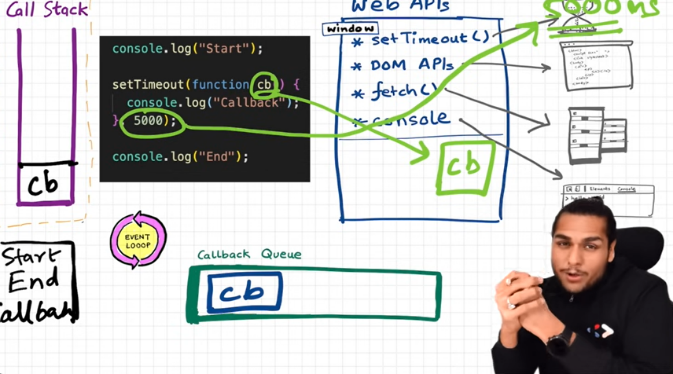
1. **Episode 11: Function in JS?**

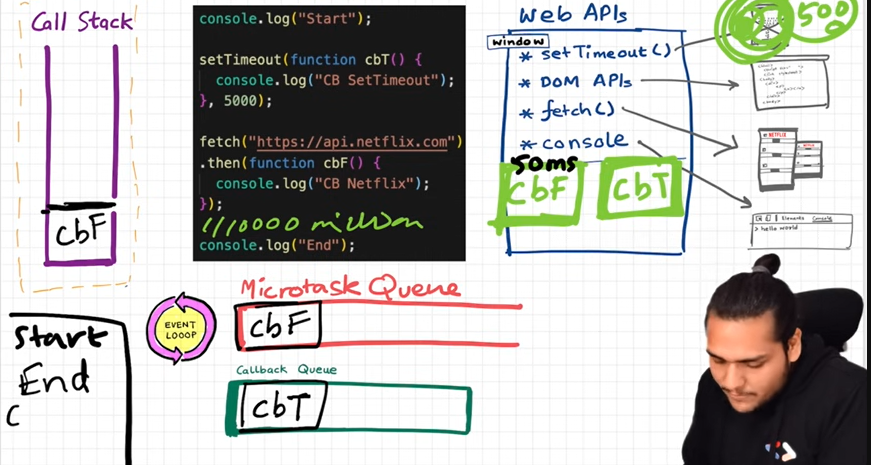
* Function also acts as a value in JS, also known as (function expression).
* Function statement and function expression both works differently in hoisting concept.
* Function without name is known as anonymous function.
* whenever you created a function and put some value() inside it, it known as parameter ex: (@params1, @params2);
* And we call the function we pass argument to it.
* We can pass function as an argument to another function, and can also return a function.
* The ability to use function as value, argument and return a function is known as first class function.

1. **Episode 12: Callback Function in JS?**

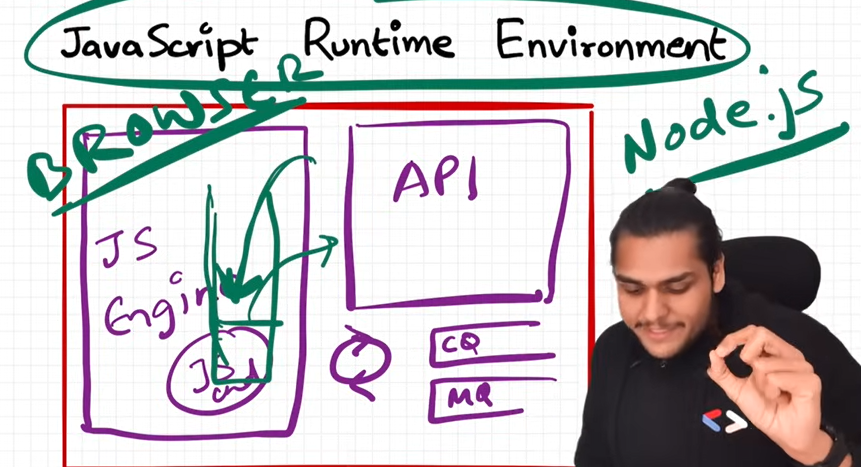
* Callback is nothing but a function inside the function.

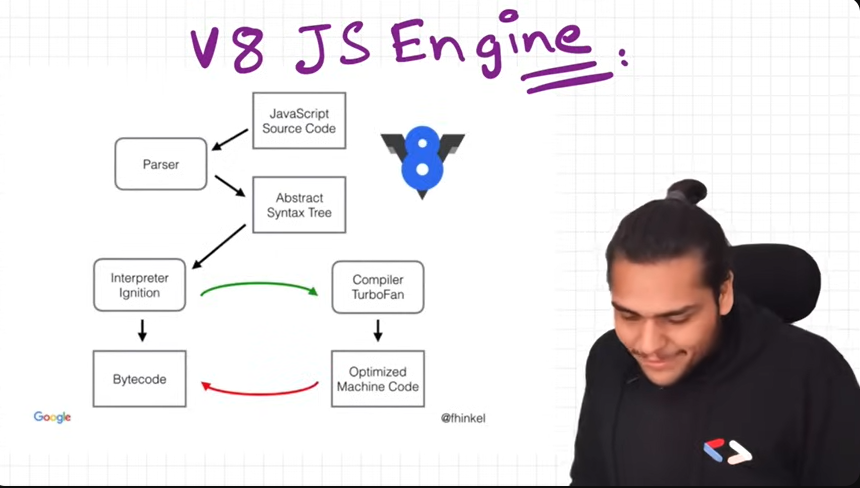
1. **Episode 13: Event loop in JS?**
   * Set Timeout, DOM APIs, fetch(), console(), Its all are the web apis and it is not the part ofJS.
   * fetch uses the microTask Queue it has a higher priority against callback Queue. if any action present in this queue will be execute first and other execute later.
   * Event loop schedules the task of both the queues after global context is empty.
   * all the call back function uses promises and the mutation observer will go inside the microTask Queue.

****



1. **Episode 14: JavaScript Engine?**
   * Browser has javascript run time environment.
   * Node.js is one of the open source javascript run time environment.
   * javascript machine is not a hardware or marchine, it is nothing but a so called program which is written in low level languages.
   * javascript is compiled and interpreted language.
   * there is many js engine is present such as google v8, charkha etc.

****

****

1. **Episode 15: Higher order function in js?**
   * a function which takes another function as a argument or return a function from it is known as higher order function.
   * a function which is passed into the function is known as callback function.
   * functions are first class citizens in the javascript.
2. **Episode 16: map, Filter and reduce in js?**
   * Map function is used to transform an array.
   * Filter function is used to filter values from the array.
   * Reduce function works as a iterate over the array and give a one single value like sum or avg.
   * Reduce take two argument one is accumulator and second is current.
   * current represents the actual value of a array, and accumulator accumulates the value like sum, avg.
   * accumulator takes 2 argument one is function (acc,curr) and second is some initial value 0 or something.
   * we can use arrow function, outside functions as a argument inside this methods.
   * we can chain all this function at one place.

# **Season 2**

1. **Episode 01: Callback and Callback hell in JS?**
   * callback function means function inside the function, it is mostly used to achieve asynchronous in js.
   * sometimes there is many function inside one another may creates a dependeny issues its also knows as callback hell.
   * when we have many api and it is dependent one after one it is very complex to manage it through callback and this situation known as callback hell.
   * the structure is also known as pyramid of doom.
   * callback also generates a problem of inversion control.

1. **Episode 02: Promise in JS?**
   * Promises is a placeholder for a certain period of time until we receive a value from asynchronous operation.
   * Promises is an object that represents the eventual completion of a asynchronous operation.
   * it is a container for a future value.
   * It is more safe and secure because we only attach the function in it not passing the whole function inside another function.
   * Function methods .then() it will run after promise was fulfilled.
   * .catch() method rejects the promise.
   * Promise has 3 states pending, fulfilled, rejected.
   * we can create a promise chaining one after one when many process are dependent each other.
   * We can create a promise with the help of new Promise keyword it holds tow value resolve and reject inside it.
   * always write catch statement to avoid the error.
2. **Episode 03: Promise APIs in JS?**
   * Promise.all() handles all the promises together. it takes array (iterables) of promises it will three parallel api calls and get the results. It will return an array of results of values. it will wait for all of them to finish.
   * if any value is rejected the output will be error as soon as error happened it will not even wait for other values result.
   * If any single value failed the whole result consider as failed.
   * prmise.allSettled() : it same in the success case. but in the failure case if any value get failed it will wait for other values result and then send response according to this.
   * promise.race() : in case of race it will give you first settled promise weather it is failed or success doesn’t matter.
   * promise.any() : it is very much similar to .race but it will wait for the first settled success.in case of all are failed it will give to the list of all the errors (aggregate error).
3. **Episode 04: Async-await in JS?**
   * async is a keyword used to create async function async function is a differnct type of function which always returns a promise.
   * if you return a function then it is ok but if you will return some other value then this function will wrapped it into a promise and then return it, means it always return a promise.
   * async-await together used to handles a promise.
   * await is a keyword that can only be used inside an async function. it is used to handling promises.
   * in the case of await keyword js engine waits to promise resolved and after only that will proceed next lines code.it will only give to the next line once promise will resolved.
   * but in case of promise its just the opposite all the code are proceed except promise and after resolved the code of promise will be execute.
   * whenever a await promise are comes in call stack then the call stack will empty and waits to resolve a promise.
   * fetch keyword returns a promise and we can handle it with await keyword.fetch gives you a response object we have to convert it into json for human redable.
   * when we do this data.json() it will again a promise so we have to use again await keyword before data.json()
   * we will use try-catch method for error handling inside async-await function . coz we don’t have .catch() method.
   * when we have async await code wrapped it inside a try and then write catch block for handling errors.
   * when we use async await javascript will use promises behind the scene. it is just a syntactical sugar for us.
4. **Episode 05: this keyword in JS?**
   * at the global level this keyword refers to the global object, it can be window or global depending on where we runs our javascript.
   * this keyword works differently in both strict and non strict mode.
   * in case of function scope value of this keyword depends on strict or non strict mode in case of strict it is undefined and non strict it refers to the window.
   * (this substitution) if the value of this keyword is undefined or null this will replace with global object only in non strict mode.
   * this keyword value is depends on how the function is called. if the function is called without any reference it will be undefined and if it is called with window.function\_name() it referce to same calling object (window) object.
   * when we create a function inside the object is known as method.
   * and if we call it using obj.key() it refer to the object (calling object).
   * the value of this can be modified using call, apply and bind method.
   * Arrow function does not have their own this binding associated to it, but it will behave exactly its enclosing lexical context, where the object is lexically present in the code (how it is written inside the code).if arrow function present in the global space it gives you a window object. arrow function don’t provide their own this binding instead of it returns this value of its lexical context.
   * Inside the DOM element this keyword refers to the perticular html element.

**6. Episode 06: Call, apply and blind in JS?**

* + Every function has its own call, apply and bind method.
  + With the help of call method we can function borrowing we can borrow a function from another function.
  + Syntax: **object.method.call(another\_object that needs to borrow).**
  + We can also pass more than one argument but first argument should always be reference object.
  + in .apply() method we can pass all the other argument wrapping into an array as a second argument instead of passing individually in .call() function.
  + .bind() method exactly look as the apply method but with the smaller difference, the only difference is instead of calling method directly it binds a method and return a function so we can call it later, it gives you copy which can be invoke later.

# **Some important concept of JS.**

1. **Arrays and objects destructuring** 
   * We can use destructuring in arrays, objects to avoid unnecessary written statement.
   * we can destructure the elements of an array like const [a,b,c, , d] = arr\_name, it extracts the value of an array elements and also skip the elements by using empty comma (,) , also set some default vaule
   * the rest parameter extract all the rest elements of an array.
   * we can also extract the values of function by destructuring.
   * We can also use …other to access other elements of array it is knows as spread operator.
   * we can also use destructirng on objects, also Renaming variable dusring destructuring, default values. also use nested objects destructuring.
2. **Optional chaining in JS.**
   * optional chaining feature (?.) prevents errors when accessing deeply nested properties that might be null or undefined. instead of throwing an error it returns undefined.
   * first it checks if property exist in the object with ? if it returns undefined or null it simply returns undefined and avoid error.
   * ex: (user.properties?.age)
   * we can use optional chaining in functions , array, objects etc.
   * we can also use ?? (nullish coalscing) to set default value if optional chaining returns undefined or null. ex(user.profile?.age??18);
3. **Polyfill for bind method.**
   * We have to create our own bind method (polyfill).
   * **CODE :**  Function.prototype,mybind = function(…args){

let obj = this, params = args.slice(1);

return function (…args2){

obj.apply (args[0], […params, …args2])}}

1. **Function curry using bind.**
   * Function currying is a technique in js where a function with multiple parameters transformed into a sequence of function, each taking single argument, this means instead of calling a function with all arguments at once, you call it step by step.
   * **CODE :** function curryAdd(a) {

return function (b) {

return function (c) {

return a + b + c;

};

};

}

console.log(curryAdd(2)(3)(5));

1. **Session storage and local storage.**
   * Local storage has maximum storage capacity and we can save our data in long time (permanent).
   * When the port changes, host changes we can not access local storage data.
   * In session storage (cookies) data saved as temporary with minimum storing capacity.
2. **Event bubbling and capturing in JS.**
   * In case of event bubbling when we click on the child it goes up at the root of dom tree (click propagates from down to up).
   * Event capturing is opposite to it (click propagates from up to down).
   * we can manage event bubbling and capturing in addeventlistners in this 3rd argument is useCapture which is Boolean type we can true for event capturing and false for event bubbling.
   * Event bubbling is used by default we don’t pass any 3rd argument in eventlistners. (as false by default).
   * **Event Bubbling:** child -> parent -> grandparent.
   * **Event capturing:** grandparent -> parent -> child.
   * We can stop propagation by using e.stopPropagation() in our event listeners. and end the whole propagation cycle.
3. **Event Delegation in JS.**
   * Event Delegation helps us to manage events.
   * suppose if we have thousands of elements so instead of attaching event handlers to all we can attach event handler to its parent element and when we click on any element the event will bubble up and parent event listener will executes this is called event delegation.
   * we can target any specific event from the parent element by e.target method.
4. **Throttling and debouncing in JS.**
   * It is used for performance optimization.
   * Deboucning is also used for limiting the rate of function call.
   * When we have to call a function so it will make a call after some certain of time so it known as throttling.
   * if you type fast and when after key pauses when api call made it known as debouncing.