**Javascript notes**

**1.How JavaScript works?**

Ans-Everything in JavaScript happens inside an "execution context".

Execution context has two components

1)memory component[variable environment] –

This is the place where all variables and functions are stored as key value pairs.

eg-{key: value || n:2;}

2)code component[Thread of execution] –

This is the place where code is executed one line at a time

Note-

\*\*JavaScript is a synchronous single-threaded language

–Single threaded means JavaScript can execute once command at a time

-Synchronous single-threaded that means JavaScript can execute one command at a time in a specific order.

**2. How javascript code is executed?**

1. When JavaScript code is executed, Execution Context is created and it is called Global Execution Context.

2. JavaScript program is executed in TWO PHASES inside Execution Context

a. MEMORY ALLOCATION PHASE - JavaScript program goes throughout the program and allocate memory of Variables and Functions declared in program.

b. CODE EXECUTION PHASE - JavaScript program now goes throughout the code line by line and execute the code.

3. A Function is invoked when it is called and it acts as another MINI PROGRAM and creates its own Execution Context.

4. Returns keyword return the Control back to the PREVIOUS Execution-Context where the Function is called and Execution Context of the Function is DELETED.

5. CALL STACK maintains the ORDER of execution of Execution Contexts.

It CREATES Execution Context whenever a Program starts or a Function is invoked and it pops out the Execution Context when a Function or Program ENDS.

**3 Hoisting in Javascript(Variables and Functions)**

1. In JS, before the code is executed, the variables get initialized to undefined.

2. Arrow functions enact as variables and get "undefined" during the memory creation phase while functions actually get run.

3. Hoisting: Mechanism in JS where the variable declarations are moved to the top of the scope before execution.

Therefore it is possible to call a function before initializing it.

4. Whenever a JS program is run, a global execution block is created, which comprises of 2: Memory creation and Code execution.

5. Variable declarations are scanned and are made undefined 6. Function declarations are scanned and are made available.

**4 How Functions works in Javascript & Variable Environment**

Summary:

1. We learnt how functions work in JS.
2. At first a global execution context is created, which consists of Memory and code and has 2 phases: Memory allocation phase and code execution phase.
3. In the first phase, the variables are assigned "undefined" while functions have their own code.
4. Whenever there is a function declaration in the code, a separate local execution context gets created having its own phases and is pushed into the call stack.
5. Once the function ends, the EC is removed from the call stack. 6. When the program ends, even the global EC is pulled out of the call stack.

**5 Shortest JS Program**

Summary:

1. Shortest Program in JS: Empty file. Still, browsers make global EC and global space along with Window object.

2. Global Space: Anything that is not in a function, is in the global space.

3. Variables present in a global space can be accessed by a "window" object. (like window.a) 4. In global space, (this === window) object.

# 6.undefined vs not defined in JS

1. Undefined is a Special Placeholder which is used to reserve memory for the variables in the memory creation phase.

Even before a single line of code is executed JS engine assigns undefined to the variables.

1. Not Defined means if we try to console or access any variable which is not declared in the code then we get Not Defined error.
2. JS is a loosely typed language or weakly typed language means it does not attaches its variables to specific data types like in C++ and java.
3. Remember undefined !== not defined.

# 7.The Scope Chain, Scope & Lexical Environment

1. Scope of a variable is directly dependent on the lexical environment.

2. Whenever an execution context is created, a lexical environment is created. Lexical environment is the local memory along with the lexical environment of its parent. Lexical as a term means in hierarchy or in sequence.

3. Having the reference of parent's lexical environment means, the child or the local function can access all the variables and functions defined in the memory space of its lexical parent.

4. The JS engine first searches for a variable in the current local memory space, if its not found here it searches for the variable in the lexical environment of its parent, and if its still not found, then it searches that variable in the subsequent lexical environments, and the sequence goes on until the variable is found in some lexical environment or the lexical environment becomes NULL.

5. The mechanism of searching variables in the subsequent lexical environments is known as Scope Chain. If a variable is not found anywhere, then we say that the variable is not present in the scope chain.

# 8.let & const in JS ,Temporal Dead Zone

Things learned:

1. let and const are hoisted but its memory is allocated at other place than window which cannot be accessed before initialisation.

2. Temporal Dead Zone exists until variable is declared and assigned a value.

3. window.variable OR this.variable will not give value of variable defined using let or const.

4. We cannot redeclare the same variable with let/const(even with using var the second time).

5. const variable declaration and initialisation must be done on the same line.

6. There are three types of error:

[1] referenceError {given where variable does not have memory allocation}

[2] typeError {given when we change type that is not supposed to be changed}

[3] syntaxError {when proper syntax(way of writing a statement) is not used}.

7. Use const wherever possible followed by let, Use var as little as possible(only if you have to). It helps avoid error.

8. Initialising variables at the top is good idea, helps shrinks TDZ to zero.

# 9.BLOCK SCOPE & Shadowing in JS

Things learned:

1. Code inside curly bracket is called block.

2. Multiple statements are grouped inside a block so it can be written where JS expects single statements like in if, else, loop, function etc.

3. Block values are stored inside separate memory than global. They are stored in block. (the reason let and const are called block scope)

4. Shadowing of variables using var, let and const.

5. The shadow should not cross the scope of original otherwise it will give error.

6. shadowing let with var is illegal shadowing and gives error.

7. var value is stored in nearest outer function or global scope and hence can be accessed outside block as well whereas same is not the case with let and const.

# 10.Closures in JS

**Closure :**Function bundled with its lexical environment is known as a closure. Whenever function is returned, even if its vanished in execution context but still it remembers the reference it was pointing to. Its not just that function alone it returns but the entire closure and that's where it becomes interesting !!

# 11.setTimeout + Closures Interview Question

1. setTimeout stores the function in a different place and attached a timer to it, when the timer is finished it rejoins the call stack and executed.

2. Without closure the var reference gives the latest value as it does not retain the original value but rather has the reference so any update in value after timeout will be shown.

3. If we use let/const because they have block scope, every time a new copy of variable is attached, thus this can be done without closure.

# 12.FIRST CLASS FUNCTIONS 🔥ft. Anonymous Functions

Things learned:

1. What is Function Statement ?

A. A normal function that we create using Naming convention. & By this we can do the Hoisting.

For Ex - function xyz(){ console.log("Function Statement"); }

2. What is Function Expression ?

A. When we assign a function into a variable that is Function Expression. & We can not do Hoisting by this becz it acts like variable.

For Ex - var a = function(){ console.log("Function Expression"); }

3. What is Anonymous Function ?

A. A Function without the name is known as Anonymous Function. & It is used in a place where function are treated as value.

For Ex - function(){ }

4. What is Named Function Expression ?

A. A function with a name is known as Named Function Expression.

For Ex - var a = function xyx(){ console.log("Names Function Expression"); }

5. Difference b/w Parameters and Arguments ?

A. When we creating a function & put some variabels in this ( ) that is our Parameters.

For Ex - function ab( param1, param2 ){ console.log(" } & When we call this function & pass a variabel in this ( ) that is our Arguments

For Ex - ab( 4, 5 );

6. What is First Class Function Or First class citizens?

A. The Ability of use function as value,

\* Can be passed as an Argument,

\* Can be executed inside a closured function &

\* Can be taken as return form.

For Ex - var b = function(param){ return function xyz(){ console.log(" F C F "); } }

7. Function are heart of JS. They are called first class citizens or first class functions because they have the ability to be stored in the variables, passed as parameters and arguments. They can also be returned in the function.

# 13.Callback Functions in JS ft. Event Listeners

Things learned:

1. Function that is passed on as argument to another function is called callback function.

2. setTimeout helps turn JS which is sinhlethreaded and synchronous into asynchronous.

3. Event listeners can also invoke closures with scope.

4. Event listeners consume a lot of memory which can potentially slow down the website therefore it is good practice to remove if it is not used.

# 14. Asynchronous JavaScript & EVENT LOOP from scratch

Things learned:

1. Browser has superpowers that are lent to JS engine to execute some tasks, these superpowers include web API's such as console, location, DOM API, setTimeout, fetch, local storage.

2. Callback functions and event handers are first stored in Web API environment and then transferred to callback queue.

3. Promises and mutation observer are stored in API environment and then transferred to microtask queue.

4. Event loop continuously observes call stack and when it is empty it transfers task to call stack.

5. Micro task is given priority over callback tasks.

6. Too many micro tasks generated can cause Starvation (nit giving time to callback tasks to execute).

# 15. JS Engine EXPOSED 🔥 Google's V8 Architecture

Things learned:

1. JS runtime environment contains all elements required to run JS.

2. It contains JS engine, set of API's, callback queue, microtask queue, event loop.

3. JS engine is a piece of code.

4. Process includes Parsing ---> Compilation -----> Execution.

5. Parsing breaks code into tokens and converts it into AST(Abstract Syntax Tree).

6. Modern JS engine follows JIT compilation, it interprets while it optimises code as much as it can. 7. Execution and Compilation are done together. 8. Execution has Garbage collector and other optimisation such as inlining, copy elusion, inline caching etc.

# 16. TRUST ISSUES with setTimeout()

Things learned:

1. The setTimeout function stores it in the callback queue which is executed only after call stack is empty, even if setTimeout is set to 0ms.

2. setTimeout ensures that minimum it will take the time mentioned because it may be paused due to call stack not empty.

# 17. Higher-Order Functions ft. Functional Programming

Things learned:

1. Follow DRY(Don't Repeat Yourself) principle while coding.

2. Use function to stop writing repeating line of codes.

3. Function that takes another function as argument(callback function) is known as Higher order functions.

4. It is this ability that function can be stored, passed and returned, they are called first class citizens.

5. If we use Array.property.function-name. This function is accessible to any array in your code.

# 18.map, filter & reduce

Things learned:

1. map method is used when we want transformation of whole array.

2. filter is used when we want to filter the arrar to obtain required value.

3. reduce is used when we want to reduce the array to single value eg (max, min, avg, sum, difference etc).

4. reduce passes two arguments one function(which includes accumulator and initial value as argument itself) and another initial value of accumulator.

5. Homework: const output = user.reduce(function(acc, curr){ if (curr.age < 30){ acc.push(curr.firstName); } return acc; }, [ ]) console.log(output);