JS Interview Prep

**Q. Define box model?**

A. The Box Model is a concept in HTML/CSS that defines how elements are laid out and sized on a web page.

In this model, every HTML element is treated as a rectangular box, which is comprised of four parts:

1. Content: This is where the text, images, or other content of an element appears.
2. Padding: This is the space between the content and the border of an element.
3. Border: This is the line that surrounds the padding of an element.
4. Margin: This is the space between the border of an element and its neighboring elements.

Together, these four parts make up the total size of an element on the web page. In CSS, you can adjust the size and spacing of each of these parts to control the appearance of an element. By default, the content area will be sized to fit its content, while the padding, border, and margin areas will be set to zero.

**Q. What is CSS Selector?**

A. CSS selectors are like element identifiers that you can use to find or select the HTML element that you want to style.

**Q. Difference between display none and visibility hidden?**

A. Display none hides the html element from the viewport and the proceeding element takes its place whereas visibility hidden, hides the element but the element occupies its space.

**Q. What is flexbox in CSS?**

A. Flexbox is a one-dimensional layout model. It allows responsive elements within a container to be automatically arranged depending upon screen size. It is useful in aligning the elements and distributing the spaces between them.

**Q. Diff b/w flexbox and CSS grid?**

CSS grid is a 2-dimensional layout model as it can handle both columns and rows whereas flexbox is one dimensional layout model, it can arrange the elements either in row or column.

**Q. JS is interpreted or compiled?**

A. JS is said to be a single threaded interpreted language however this is not entirely correct as some JS engines provide just in time compilation.

**Q. What do you mean by JIT?**

A. JIT compilation is compiling the bytecode to machine code at run time. For example, the V8 JS engine does not compile the JS code before execution rather it does it while executing the code so the interpretation and the compilation goes hand in hand. JIT compilation works by analyzing the JavaScript code as it runs and then generating optimized machine code on the fly. This can result in faster execution times for frequently used code.

In summary, JavaScript is an interpreted language, but some modern JavaScript engines use JIT compilation to improve performance.

**Q. JS is single-threaded meaning?**

A. Yes, JS engines have one call stack that manages the execution context where the code is executed line by line, one piece at a time.

**Q. Diff between undefined and not defined?**

A. When something has been defined in a scope but its value is not provided it will have a default value which is undefined.

When something has not even been defined that is when it will throw a reference error saying that this variable is not defined.

**Q. How does “hoisting” in JavaScript works?**

A. JS program runs in two phases,

i) Creation of execution context within the call stack and assigning memory space to all the variables with values as undefined and functions as their declaration.

ii) Line by line code execution.

**Q. Do let variables get hoisted?**

Yes, they are hoisted but they get into TDZ or (Temporal Dead zone) which means they are inaccessible till the code execution reaches the line where they have been declared and any attempt to access these variables till, they are in TDZ will throw a reference error.

=> Const variables are needed to be assigned a value at the time of initialization and will throw a syntax error if not done so.

**Q. Why do we call JS dynamic language?**

=> We call JS as a dynamic language coz the datatype of the variables can change at run-time.

for ex: =

let a = true //currently a is of type boolean

a = "true" // now the type of a is a string

so on and so forth...

**Q. How does JS determine the data type?**

=> In JavaScript, the type of variable is determined dynamically at runtime, based on the value that is assigned to it. This means that the same variable can hold different types of values at different points in time, and that the type of a variable can change during the execution of a program.

**Q. What is typeOf operator or how can we determine type of a variable in JS?**

=> typeOf operator helps us in getting the datatype of any variable that is supplied to it and with help of this typeOf operator we can determine the datatype.

**Q. What are different datatypes in JS?**

=> Major ones are SNNUBO

String Null Number Undefined Boolean Object other two are big int and Symbol.

**Q. Explain undefined datatype?**

=> undefined is the default data type given to every variable when it is declared but not assigned or initialized with a value. And it generally defines the absence of data.

**Q. Explain Null or diff b/w undefined and null?**

=> Null is same as undefined as far as the value prospect goes but the key difference is it is the intentional absence of data so I can say.

for ex:=>

let a; // absence of data but undefined

let b = null; // intentional absence of data and null

**Q. Hoisting?**

=> The JavaScript program runs in two phases:

i) Creation of execution context within the call stack and assigning memory space to all the variables with values as undefined and functions as their declaration.

ii) line by line execution

Hoisting is a mechanism where variables and function declaration are moved to the top of the scope.

**Q. Explain the two phases of JavaScript program execution.**

JavaScript program execution happens in two phases: **the creation phase and the execution phase.**

**Creation phase:** In the creation phase, JavaScript creates the execution context for the current code. During this phase, the following things happen:

I) The memory space is allocated for variables and functions declared in the code. This is called hoisting.

ii) The global object (window object in a browser) is created, and this keyword is assigned a value.

Iii) The outer environment (i.e., the scope chain) is set up by creating references to variables and functions in the parent scope.

iv) Function declarations are added to the memory and are available for use. Function expressions, on the other hand, are not hoisted and cannot be accessed until the execution phase.

v) Variables are initialized to the undefined value.

**Execution phase:** In the execution phase, JavaScript executes the code in the current execution context. During this phase, the following things happen:

I) The JavaScript engine executes the code line by line, following the order of execution.

ii) When the engine encounters a function call, it creates a new execution context for that function and runs its code. This is called function invocation.

Iii) When a function returns, the execution context for that function is destroyed, and control returns to the parent context.

iv) Values are assigned to variables, and their values are updated during the execution of the code.

v) Objects and arrays are created and updated with new properties or elements.

vi) Conditional statements and loops are executed, based on the conditions specified in the code.

vii) Events are triggered, and event handlers are executed.

viii) Timers are set up and their callbacks are executed when the timer expires.

By understanding these two phases of JavaScript program execution, you can better understand how the code is executed and how variables and functions are created and accessed during the program's lifetime.

**Q. Are JavaScript initialization hoisted?**

=> No, only the declarations are hoisted.

**Q. What are global variables?**

=> Global variables are those which are available/ visible throughout the context of the program. That is, they can be accessed anywhere in the program.

**Q. What are the issues with Global variables?**

=> It can make an application very hard to debug and buggy

**Q. What happens when you declare a variable without var?**

=> The variable becomes global.

**Q. What is the use of "use strict"?**

=> "use strict" checks if the variable is declared with var keyword or not and if not, it throws an exception of Reference error.

**Q. How can we avoid global variable issues?**

=> We can put them in namespace. Or we can use Module with closures.

**Q. What are closures in JavaScript?**

=> In JavaScript, a closure is created when a function is defined inside another function, and the inner function has access to the outer function's variables and parameters, even after the outer function has returned.

A closure allows a function to maintain access to the variables and parameters of the parent function, even after the parent function has completed execution. This is achieved by creating a "closure" around the inner function, which captures a reference to the parent function's variables and parameters.

Closures are commonly used to create private variables and functions, which are not accessible from outside the function. This can be useful for encapsulating functionality and preventing naming conflicts.

**Q. Why do we need closures?**

=> To create Self-contained modules. It results in self-contained states and in avoiding global variables issues. It helps you to expose only the properties that you want.

**Q. Explain IIFE?**

=> Immediately invoked function expression, these are anonymous functions which invoke or call themselves immediately.

**Q. What is the use of IIFE?**

=> It solves the name collision problems. Let's say we have one function named Init() but later we defined a variable as Init. We can use an anonymous function and immediately invoke it to solve this problem.

**Q. What is the “name collision” in global scope?**

=> Name collision happens when same name function names and variable names are declared in same context.

**Q. What are design patterns?**

=> Design patterns are time tested solutions.

**Q. What is the most used design pattern in JS?**

=> Module design pattern or the Module revealing pattern.

**Q. What is module revealing pattern?**

=> Module pattern or revealing module pattern has 2 big advantages:

-- Self-contained independent components.

-- Provides Encapsulation and Abstraction.

Module revealing pattern is a combination of IIFE and closures

=> better code management because of IIFE and good encapsulation because of closures.

**Q. IIFE vs normal function?**

=> A normal function simply includes a declaration and a definition, but an IIFE is declaration, definition and invocation at the same time.

**Q. What are the different ways of declaring objects in js?**

=>

i) Using literals that means {key: value} giving a key value pair.

ii) Using Object.create();

iii) Constructor way that means by using functions

iv) ES6 class method

**Q. How can we do inheritance in JS/ What is protoype in JS/ Explain Prototype chaining?**

=> In JavaScript, prototype chaining is the mechanism by which objects inherit properties and methods from their prototype chain. Every object in JavaScript has a prototype, which is an object that serves as a template for the properties and methods of that object. When a property or method is accessed on an object, JavaScript first looks for that property or method on the object itself. If it is not found, it then looks for it on the object's prototype. This process continues up the prototype chain until the property or method is found or until the root of the prototype chain is reached (which is usually the Object.prototype object).

**Q. What is the let keyword in js?**

=> Let is an ES6 feature, and it helps to create immediate block level local scope variables.

let variables do get hoisted but they get into TDZ Temporal dead zone and they do not get initialized even with an undefined.

if tried to access

Uncaught ReferenceError: Cannot access 'x' before initialization

**Q. What is the temporal dead zone?**

=> It's a period or it's a state of a variable where variables are named in memory, but they are not initialized with any value.

**Q. let vs var?**

=> 2 differences:

i) Scoping rules: var are scoped to immediate function body, whereas let are scoped to immediate enclosing block.

ii) Initialized value during the hoisting phase: var are initialized with undefined, let are initialized with nothing

**Q. String concatenation and arithmetic puzzle?**

=> "10"+"10" = "1010"

10+10 = 20

1 + 1 + "4" = "24"