1. Is JavaScript Interpreted Language in its entirety?

Computer programming languages are languages that express a set of instructions for the digital computers. These languages can be of two types, the compiled and the interpreted. The former one converts source code to machine 9byte) code before execution while the latter one read and executes the source code directly line by line. The complied languages use compiler to convert high-level-languages to machine language while the interpreted languages use interpreter, which runs at the runtime, to convert in to machine languages.

So, in which category does JavaScript classified?

Before diving into the details, let us look the overview of how the JavaScript works. JavaScript requires a browser or a web server to run its code. In the browser is the search engine which takes the code and executes it. Such as, V8 of Google Chrome, Spider Monkey of Firefox and so on.

Inside the engines are two ways AOT (Ahead of Time) and JIT (Just in Time). The former one is to refer executions to be done ahead of execution time while the latter is to refer executions to be done during the execution time. Based on these two ways the engines take some steps to help the codes run. The first step is to parse the code by a parser; it will read the code line by line and will check if the syntax is correct. The parser will know rules for JS to ensure this. If it finds any mistakes it will throw an error and execution will stop. If everything is ok the parser will produce a data structure which is called Abstract Syntax Tree (AST) and it is then turned into machine code, machine code is executed directly by the computer's processor and that is where our code runs.

Most agree on that the JavaScript is interpreted language while few argue that it is not necessarily only that. The advocators of the latter idea raise the issues of difference in the engines where the JavaScript runs like the V8 of Google Chrome which compiles the source code to a byte or machine code. Even though JavaScript seems a compiled language the case is not that. Every language is compiled but the difference is in compiled languages the file is compiled before the execution time but in the case of JavaScript the source code is compiled Just-In-Time manner.

So far, I have tried to explain in short how JavaScript works and whether it is interpreted or compiled. Even though it is still now not solved, from my point of view, JavaScript is an interpreted language.

2. The History of “typeof null”

   Null and undefined in JavaScript are actually values and types created to simulate errors and keywords common in other programming languages. When a variable is `undefined`, or uninitialized, in most programming languages it means that a space in memory has been assigned to a variable name, but the programmer has not yet done anything with that space in memory. This usually results in a compile time error.

   Unlike other programming languages, null is not a keyword to indicate the space in a memory; rather it is an object which incorrectly suggests that null is an object. It is a remnant from the first version of a JavaScript. It is an object that points to zero. Null is supposed to signal that a variable has no value because the programmer purposefully cleared the value and set it to `null`. All reference types (pointers) in JavaScript are objects. In early JavaScript, null was meant to simulate a null pointer (reference), ergo it was hard coded to return 'object' for its type.

3. Explain in detail why hoisting is different with let and const?

   In the literal meaning, dictionary meaning, hoisting is raising something higher. Based on this meaning many think hoisting in programming languages is raising the codes that they write to the top which is not right. The JavaScript engine interprets the JavaScript written within this Global Execution Context in two separate phases; compilation and execution. In the former phase it parses and creates a space for the declared variables. This process of lifting the variables up and giving them a space in memory is called hoisting. Hoisting is a JavaScript mechanism where variables and function declarations are moved to the top of their scope before code execution.

   In variable declarations in JavaScript, var was used before the recent version of JavaScript (ES6) came which introduced newest ways of declaration, let and const. These are assumed to improve some issues with the var, re-declaring it is the main problem with the var.

Unlike the var, let come with the block scope, {}, meaning it can be accessible in that block which means it is neither a function or global scoped rather a block scoped. You can update let but not re-declare it unlike the var. it is also possible to define the same variable in different scope with let. Just like var, let declarations are hoisted to the top.
Unlike var which is initialized as undefined, the let keyword is not initialized. So if you try to use a let variable before declaration, you'll get a Reference Error.

Variables declared with the const maintain constant values. Like let, they are accessed in the block level. Const cannot be updated not re-declared. Const declarations must be initialized at the time of declarations which is not mandatory with the let.

4. Semicolons in JavaScript: To Use or Not To Use?

Semicolons in JavaScript divide the community. Some prefer to use them always, no matter what. Others like to avoid them.

The semicolon in JavaScript is used to separate statements, a piece of code that tells the computer to do something. Using semicolon is an option in JavaScript because JavaScript does not strictly require semicolons. When there is a place where a semicolon was needed, it adds it behind the scenes. Automatic Semicolon Insertion (ASI) is what does these.

The rules of JavaScript Automatic Semicolon Insertion

The JavaScript parser will automatically add a semicolon when, during the parsing of the source code, it finds these particular situations:

1. when the next line starts with code that breaks the current one (code can spawn on multiple lines)
2. when the next line starts with a }, closing the current block
3. when the end of the source code file is reached
4. when there is a return statement on its own line
5. when there is a break statement on its own line
6. when there is a throw statement on its own line

7. when there is a continue statement on its own line

5. Expression Vs Statement in JavaScript?

Statements and expressions are two very important terms in JavaScript. While an expression produces a value, a statement performs an action. Any unit of code that can be evaluated to a value is an expression. Since expressions produce values, they can appear anywhere in a program where JavaScript expects a value such as the arguments of a function invocation. A statement is an instruction to perform a specific action. Such actions include creating a variable or a function, looping through an array of elements, evaluating code based on a specific condition etc. JavaScript programs are actually a sequence of statements.

There are different types of expression like Arithmetic expression, Logical expression, String expression and so on… There are also different statements such as Declaration statement, conditional statements and so on.