1. Is Javascript Interpreted Language in its entirety ?

1.1. What is an Interpreted and compiled languages mean?

-Every program is a set of instructions, whether it’s very easy two line code or complex networking codes. Compilers and interpreters take human-readable code and convert it to computer-readable machine code.

-In a compiled language, the target machine directly translates the program. In an interpreted language, the source code is not directly translated by the target machine. Instead, a different program, also known as the interpreter, reads and executes the code.

1. **Compiled Languages**

Compiled languages are converted directly into machine code that the processor can execute. As a result, they tend to be faster and more efficient to execute than interpreted languages. They also give the developer more control over hardware aspects, like memory management and CPU usage.

Compiled languages need a build step – they need to be manually compiled first. One needs to rebuild the program every time he/she make a change.

Examples of pure compiled languages are C, C++, Erlang, Haskell, Rust, and Go.

1. **Interpreted Languages**

Interpreters run through a program line by line and execute each command.

Interpreted languages were once significantly slower than compiled languages. But, with the development of just-in-time-compilation that gap is shrinking.

Just-in-time compilation (JIT) also known as Dynamic compilation is a method for improving the performance of interpreted programs. During execution the program may be compiled into native code to improve its performance.

Dynamic compilation has some advantages over static compilation. When running Java or C# applications, the runtime environment can profile the application while it is being run. This allows for more optimized code to be generated. If the behavior of the application changes while it is running, the runtime environment can recompile the code.

Some of the disadvantages include startup delays and the overhead of compilation during runtime. To limit the overhead, many JIT compilers only compile the code paths that are frequently used.

Traditionally there are two methods for converting source code into a form that can be run on a platform. Static compilation converts the code into a language for a specific platform. An interpreter directly executes the source code.

JIT compilation attempts to use the benefits of both. While the interpreted program is being run, the JIT compiler determines the most frequently used code and compiles it to machine code. Depending on the compiler, this can be done on a method or smaller section of code.

Examples of common interpreted languages are PHP, Ruby, Python, and JavaScript.

-The difference between compiled and interpreted languages can be seen using a food recipe analogy scenario.

* Imagine we have a cake recipe that we want to make, but it's written in ancient Ge’ez. There are two ways we, a non-ancient-Ge’ez speaker, could follow its directions.

The first is if someone had already translated it into English or any other language we could speak. Then, we could read the English version of the recipe and make cake. Think of this translated recipe as the compiled version.

The second way is if we have someone who knows ancient Ge’ez. When we’re ready to make the cake, someone who knows both the ancient Ge’ez and English translates us each steps of the recipe line by line. In this case, that person is the interpreter for the interpretedversion of the recipe.

**1.2. Advantages and disadvantages**

1. **Advantages of compiled languages**

* Programs that are compiled into native machine code tend to be faster than interpreted code. This is because the process of translating code at run time adds to the overhead, and can cause the program to be slower overall.
* They are self-contained units that are ready to be executed. Because they are already compiled into machine language binaries, there is no second application or package that the user has to keep up-to-date. If a program is compiled for Windows on an x86 architecture, the end user needs only a Windows operating system running on an x86 architecture.
* Increase program performance. Users can send specific options to compilers regarding the details of the hardware the program will be running on. This allows the compiler to create machine language code that makes the most efficient use of the specified hardware, as opposed to more generic code. This also allows advanced users to optimize a program's performance on their computers

1. **Disadvantages of compiled languages**

-Additional time needed to complete the entire compilation step before testing

-Platform dependence of the generated binary code

- Because a compiler translates source code into a specific machine language, programs have to be specifically compiled for OS X, Windows or Linux, as well as specifically for 32-bit or 64-bit architectures. For a programmer or software company trying to get a product out to the widest possible audience, this means maintaining multiple versions of the source code for the same application.

1. **Advantages of interpreted languages**

-Interpreted languages tend to be more flexible, and often offer features like dynamic typing and smaller program size. Also, because interpreters execute the source program code themselves, the code itself is platform independent.  
-They offer dynamic typing as well as dynamic scoping  
- Provides an ease of debugging  
- They use the evaluator reflectively like in a first order evaluation function  
- They provide you with an automatic memory management.

1. **Disadvantages of interpreted languages**

-The most notable disadvantage is typical execution speed compared to compiled languages.

-  Another downside to the interpreted programs is the fact that the executables can only be run by an interpreter.

Most programming languages can have both compiled and interpreted implementations – the language itself is not necessarily compiled or interpreted. However, for simplicity’s sake, they’re typically referred to as such.

Python, for example, can be executed as either a compiled program or as an interpreted language in interactive mode. On the other hand, most command line tools, CLIs, and shells can theoretically be classified as interpreted languages.

1.3 Is JavaScript a Compiled or an Interpreted language?

The first thing to understand, Computer doesn't understand the programming languages directly. Every programming language got its own syntax, grammar, and structure. No matter what programming languages (JavaScript, Python, Java, etc.) we are writing the code with, it has to be translated into something that the machine (Computer) understands.

The most important fact here is, how does the JavaScript source code go through the journey of becoming a machine-understandable language? JavaScript Engine performs many of the steps (in fact, more cleaner and sophisticated ways) that a typical Compiler would perform in compiling source code.

* In JavaScript, the source code typically goes through the following phases it is executed.

I. **Tokenizing**: Breaking up a source code string into meaningful chunks called **Tokens.** For example, the source code var age 7=5; can be tokenize as var, age, =,and 7

II. **Parsing**: Parsing is a methodology to take the array of Tokens as input and, turn it into a tree of nested elements that are understood by the grammar of the programming language. This tree is called Abstract Syntax Tree (AST).

**What is an Abstract syntax tree (AST)?**

Abstract syntax trees are data structure widely use in compilers, due to their property of representing the structure of program code. An AST is usually the result of the syntax analysis phase of a compiler. AST is not only used strictly with Javascript environments such **nodejs** or the **browser,** everything started on Java world long time ago, with **Netscape**.

1. **Code Generation**: In this phase, the AST used as input and an executable byte-code is generated that is understood by the environment (or platform) where the executable code will be running. The executable bite code then refined/converted even further by the optimizing JIT (Just in Time) compiler.

1.4 Conclusion

To conclude, JavaScript code indeed gets compiled. It is more closer to be Compiled than Interpreted. It is compiled every time. After the compilation process produces a binary byte code, by which the JS Virtual Machine executes. But, unlike other compiled programming languages, JavaScript the compilation doesn’t take place at the build time.

* So, seeing all the above properties and explanations, we can conclude that even though JavaScript is not strictly complied or interpreted language, it exhibits the properties of both types. But it takes much of the properties of the Compiled language. So, JavaScript is rather a **Compiled language** if it is not in between.

2. The history of “typeof null”

2.1 What is typeof operator?

-Before seeing about the history of “typeof null”, it is important to discover about the JavaScript **typeof** operator.

-The typeof operator in JavaScript evaluates and returns a string with the data type of the operand. For example, to find the type of 123, we would write as

typeof 123

This will return a string with a type of 123, which, in this case will be “number”. In addition to “number”, the type of operator, can return one of the 6 potential results, which namely are “number”, ”string”, ”boolean”, “object”, “function”, “undefined” and “symbol”.

-Apart from the above types that the typeof operator could return, there are some unexpected results. Some of them are:-

* What’s the type of [1,2,3]

-the type of the above array is actually an object. In JavaScript arrays are technically objects, just with special abilities and behaviors.

* What’s the type of null

-Here we come to our point. **typeof null** would actually return an **object** value.

-the null value is technically a primitive, the way object or number are primitives. This would typically means that the type of **null** should also be **null**. However, this is not the case because of the peculiarity with the way JavaScript was first defined

-In the first implementation of JavaScript, values were represented in two parts- a type tag and the actual value. There were 5 type tags that could be used, and the tag for referencing an object was 0. The null value, however, was represented as NULL pointer, which was **0x00**  for most platforms. As a result of this similarity, null has the 0 type tag, which corresponds to an object.

2.2 Nature of “typeof null” object

-In JavaScript, typeof null is ‘object’, which incorrectly suggests that null is an object (it isn’t). This is a bug and one that unfortunately can’t be fixed, because it would break existing code.

- The reason to call it a bug is that the bug came about because in the initial implementation of JavaScript all values were stored in 32 bits units, where:

* The first 1–3 bits contain the type tag
* The remaining 29–31 bits contained the actual data

-The type tag for objects was **0**. While **null** was represented with a **NULL** pointer. Note that a **NULL** pointer does not point to an address — it simply points to nothing (hence why it is called null in the first place). But in the C standard, **NULL** is considered equal to **0.**Consequently, **null** had **0** as its type tag, and since **0** is for objects, then **typeof** **null**fraudulently gives us “object”.

It should now be obvious why typeof thought that null was an object: it examined its type tag and the type tag said “object”.

- <https://medium.com/@almog4130/javascript-is-it-compiled-or-interpreted-9779278468fc>--medium.com

- <https://www.freecodecamp.org/news/compiled-versus-interpreted-languages/> --free code camp.org

- <https://www.freecodecamp.org/news/just-in-time-compilation-explained/> --free code camp.org

- [https://www.techwalla.com/articles/disadvantages-advantages-of-compilers --techwalla.com](https://www.techwalla.com/articles/disadvantages-advantages-of-compilers%20--techwalla.com)

- <https://itinterviewguide.com/interpreted-language/> --it interview gide.com

- [https://medium.com/jspoint/how-javascript-works-in-browser-and-node-ab7d0d09ac2f --medium.com](https://medium.com/jspoint/how-javascript-works-in-browser-and-node-ab7d0d09ac2f%20--medium.com)

- [https://blog.greenroots.info/javascript-interpreted-or-compiled-the-debate-is-over-ckb092cv302mtl6s17t14hq1j --blog.greenroots.com](https://blog.greenroots.info/javascript-interpreted-or-compiled-the-debate-is-over-ckb092cv302mtl6s17t14hq1j%20--blog.greenroots.com)

- [https://medium.com/@jotadeveloper/abstract-syntax-trees-on-javascript-534e33361fc7 --medium.com](https://medium.com/@jotadeveloper/abstract-syntax-trees-on-javascript-534e33361fc7%20--medium.com)

- [https://javascriptrefined.io/null-and-typeof-9330e475d272 --javascriptrefind.io](https://javascriptrefined.io/null-and-typeof-9330e475d272%20--javascriptrefind.io)

- <https://bitsofco.de/javascript-typeof/> -- bitsofco.de

- <https://stackoverflow.com/questions/18808226/why-is-typeof-null-object> -- stackoverflow.com

- <https://2ality.com/2013/10/typeof-null.html#:~:text=In%20JavaScript%2C%20typeof%20null%20is,it%20would%20break%20existing%20code.&text=The%20data%20is%20a%20reference%20to%20an%20object>. –2ality.com