

ADDIS ABABA UNIVERSITY

ADDIS ABABA INSTITUTE OF TECHNOLOGY

CENTER OF INFORMATION TECHNOLOGY AND SCIENTIFIC COMPUTING

**Web Design and Development Assignment**

**Name:** - Kaleb Fekadu

ID:- ATR/6065/11

Section:- IT

January 2021

Table of Content

# Compiled Vs. Interpreted Programming Languages ………….. 1

1. History Of “typeof null” ……………………………………………………. 2
2. Why hoisting is different with let and const? …………………… 3
3. Semicolon in JavaScript: To Use or Not to Use? ……………….. 4
4. Expression vs. Statement in JavaScript …………………………….. 5
5. References ………………………………………………………………………..8

# Compiled Vs. Interpreted Programming Languages

In order to be compiled and executed programming languages use different approaches. One of the common ways to differentiate them is to split them into two compiled and interpreted languages.

The main purpose of Compilation and Interpretation is to transform human readable source codes into machine codes so that it can be executed.

#### Compiled Languages

A compiled language is a programming language that is implemented using compilers rather than interpreters. A compiler is a program that translates the source code written in a particular programming language into a machine code.

#### Interpreted Languages

An interpreted language is a programming language that is implemented using interpreters. The source code is not directly compiled into machine code rather it will be read and directly executed, line by line.

**Is JavaScript Compiled or interpreted language?**

According to sources in the internet JavaScript used to be an interpreted language. But in modern approaches the fact that determines JavaScript as an interpreted language is under a question mark. Why?

In modern time JavaScript runtime environment JavaScript engines no longer just interpret JavaScript, they compile it.

For this we can take V8 engine as an example. V8 is JavaScript engine that provides runtime environment in which JavaScript executes while browsing with chrome.

JavaScript is internally compiled by V8 with just-in-time (JIT) compilation to speed up the execution. JIT or just-in-time compilation is a hybrid between compilation and interpretation. They work by analyzing the byte-code and decide which part should be fully compiled, and then the code will be compiled into machine code.

To conclude, JavaScript code indeed gets compiled. It is closer to be compiled than Interpreted. It is compiled every time.

**The History of “typeof null”**

In JavaScript “typeof” is an operator which provides information about the type of data contained inside a variable. These type of data are Undefined, Boolean, Number, Object, Function, and Null. “typeof null” returns object since null is considered as an empty value for an object. This operator is remnant from the first version of JavaScript in which values were stored in 32-bit units, and small type tags stored in the lower bits were consisted. There were five them:

* 000: object. The data is a reference to an object.
* 1: int. The data is a 31 bit signed integer.
* 010: double. The data is a reference to a double floating point number.
* 100: string. The data is a reference to a string.
* 110: Boolean. The data is a Boolean.

The type tags were three bits in length having two additional bits for four types.

Two values were special:

* Undefined (JSVAL\_VOID) was the integer −230 (a number outside the integer range).
* Null (JSVAL\_NULL) was the machine code NULL pointer. Or: an object type tag plus a reference that is zero.

From these the type of operator examines the type tag of NULL and understands NULL as an object, as “typeof null” returns an object type.

When defining a variable that is meant to later hold an object, it is advisable to initialize the variable to null as opposed to anything else.

**Why hoisting is different with let and const?**

**Hoisting** is **JavaScript's** default behavior of moving all declarations to the top of the current scope (to the top of the current script or the current function).

Variables declared with let and const are hoisted. Where they differ from other declarations in the hoisting process is in their initialization.

During the compilation phase, JavaScript variables declared with var and function are hoisted and automatically initialized to undefined. Contrastingly, variables declared with let, const, and class are hoisted but remain uninitialized, these variable declarations only become initialized when they are evaluated during runtime. The time between these variables being declared and being evaluated is referred to as the **temporal dead zone**.

So the difference between “let and const” other declaration is their initialization in hoisting process. Variable declared with let and const remain uninitialized while hoisted,

**Semicolons in JavaScript: To Use or Not to Use?**

When we come to using semicolons in JavaScript it is an optional thing as ASI (Automatic Semicolon Insertion) which is a kind of set of rule which determines whether or not a semicolon will be interpreted in a certain code.

There are three ASI rules to be aware of.

1. A semicolon will be inserted when it comes across a line terminator or a '}' that is not grammatically correct. So, if parsing a new line of code right after the previous line of code still results in valid JavaScript, ASI will not be triggered.
2. If the program gets to the end of the input and there were no errors, but it's not a complete program, a semicolon will be added to the end. Which basically means a semicolon will be added at the end of the file if it's missing one.
3. There are certain places in the grammar where, if a line break appears, it terminates the statement unconditionally and it will add a semicolon. One example of this is return statements.

To conclude even though ASI helps us replace a semicolon interpretation that would nice to use semicolons to avoid concatenation of codes or some errors that cannot be replaced by ASI .

**Expression vs. Statement in JavaScript**

Expression

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. As per the MDN documentation, JavaScript has the following expression categories.

## **Arithmetic Expressions**

Arithmetic expressions evaluate to a numeric value.

10; // Here 10 is an expression that is evaluated to the numeric value 10 by the JS interpreter10+13; // This is another expression that is evaluated to produce the numeric value 23

## **String Expressions**

String expressions are expressions that evaluate to a string.

'hello';  
'hello' + 'world'; // evaluates to the string 'hello world'

## **Logical Expressions**

Expressions that are evaluate to the Boolean value true or false are considered to be logical expressions.

10 > 9; // evaluates to boolean value true  
10 < 20; // evaluates to boolean value false  
true; //evaluates to boolean value true  
a===20 && b===30; // evaluates to true or false based on the values of a and b

## **Primary Expressions**

Primary expressions refer to stand alone expressions such as literal values, certain keywords and variable values.

'hello world'; // A string literal  
23; // A numeric literal  
true; // Boolean value true  
sum; // Value of variable sum  
this; // A keyword that evaluates to the current object

# Statements

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.

Statements in JavaScript can be classified into the following categories

## Declaration Statements

Such type of statements creates variables and functions by using the var and function statements respectively. Examples include

## Expression Statements

Wherever JavaScript expects a statement, you can also write an expression. Such statements are referred to as expression statements. But the reverse does not hold. You cannot use a statement in the place of an expression.

## Conditional Statements

Conditional statements execute statements based on the value of an expression. Examples of conditional statements includes the if…else and switch statements.

**References**

* <https://medium.com/@almog4130/javascript-is-it-compiled-or-interpreted-9779278468fc>
* <https://finematics.com/compiled-vs-interpreted-programming-languages/>
* <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>.
* <https://dev.to/adriennemiller/semicolons-in-javascript-to-use-or-not-to-use-2nli>
* <https://www.w3schools.com/js/js_hoisting.asp#:~:text=Hoisting%20is%20JavaScript's%20default%20behavior%20of%20moving%20all%20declarations%20to,script%20or%20the%20current%20function>).