ADDIS ABABA UNIVERSITY

ADDIS ABABA INSTITUTE OF TECHNOLOGY

CENTER OF INFORMATION TECHNOLOGY AND SCIENTIFIC COMPUTING

DEPARTMENT OF SOFTWARE ENGINEERING

More On JavaScript

Prepared by: BekenAdugna – ETR/2532/11

Submitted To: Mr. Fitsum Alemu

Table of Contents

Is JavaScript Interpreted Language In Its Entirety

Before answering that we need to know what it means by interpreted and compiled programing language, we need to convert our source code to machine code so we can run it and there are two ways of doing that which are,

- **Compiled programing** is when the source code file is typically complied to the machine code before its being executed.

For example if I were to give you some code done by a compiled programing language a program called compiler goes through the source code and creates a separate file that contains the machine code , so now it will be an executable file which means you only receive the end so you just execute it on your pc.

- **Interpreted programing** in this case the source code will be read line by line and been directly executed. In the above example the compiled one you never get to see the source code but when interpreted you send the source code and the machine on the other end is responsible to change it to a machine language (executable file).

When coming to the question it JavaScript entirely interpreted I would go with no and why I think that is the JavaScript engine (V8 engine) has its own JIT Compiler within which compiles the source code before it run but this compiler is not as advanced in other compiled programing language like C++, but still the fact that this compiler exist and does the compilation before execution won't make JavaScript all in all an interpreted programing language.

The Evolution of internet History of Typeof Null

TypeOf null was there starting from the first version of JavaScript and was stored as 32 bit units which has small type tags (1-3) the actual data value to the type tag when stored in 5 lower bit units which was

- 000 Object : which refers to Objects
- 010 Double : date Double
- 100 Strings: data String
- 110 Boolean: data refers to Boolean

- 1 Int : Integers

And so when the type tag was three bit in length providing two additional bits for four type two values got special the undefined and null, so when the type tag is examined the type tag said its an Object.

Detail Why Hoisting is Different in Let

When declaring a value with **var** it automatically has an undefined value attached to it by our engine until we assign a value to it, but when it comes to **let**and **const** our engine works differently these variable we just declared with **let and const** are hoisted in the top of the block but not initialized these means that the block of code knows of the variables but it can't be used until it's been declared it will result in Reference Error. The variables in this known as the Temporary Dead zone.

Examples

Console.log(x);

Var x;

This can be done because of hoisting and has an default undefined value in case of var

Let x; //declaration

Console.log(x);

Works because its automatic assignation of default value is being done after declaration in Let and const case but if we do like this

Console.log(x); // Reference Error- because we are in the Temporal Dead Zone

Let x; // default value is assigned at this time

Semicolons in JavaScript

Semicolons are sometimes optional in JavaScript because of ASI(Automatic Semicolon Insertion) but they are not always optional it has its own rule, these doesn't mean actual

semicolons are being inserted into your code its more of a set of rules used in JavaScript will determine whether or not semicolons will be interpreted as center spots

These Are The Following Rules

A. If a program is passed until the end of the input and its not yet a complete program a semicolon is inserted this is to say short for saying its going to insert a semicolon at the end of a file

B. When reading a token from left to right that doesn't match the grammar rule has a semicolon inserted before it if either of the conditions are met.
   - The error token if separated from the previous by at least one terminator.
   - The error token is "}".

Example

Var x Var x;

Var y will Var y;

But watch out when line begins with "{"or "["

Example

A= b + c

(d + c).print() will be A= b + c (d + c).print not A= b + c;

(d + c).print();

C. If a line terminates in encounter with a restrained production ASI will try to save you by adding a semicolon before the line terminates

Example

Return statements

```
Function calc(){
    Return{
        Name="beken"
```

}in this case it becomes undefined

But if we make then on the same line it will work like this

```
Function calc(){
    Return{Name=”beken”}
```

And we have exception where we our self-need to put semicolons the ASI won’t help us in such case like a heard of a For loop

For(i=0 i>10 i++){

Return i} this is unacceptable

So when we come to should we use semicolons or not I prefer to use then even though the ASI will fill some gaps because

- If we are going to write a JavaScript without optionalsemicolons we need to make sure we really know and understand the rule governing it
  - ASI can make our JavaScript invalid and make it hard to debug if we rely up on it totally
  - Plus it’s really a bad programing habit that might be hard for beginner programmers to stop we get used to not using these semicolons by our own since not all programing languages support ASI

Expression Vs Statements

Expressions are pieces of code that result to have a value they become value

For example

Const x =10;

A function call is also an expression

Const y= add(); this is an expression to because the add function brings some value or at least is assigned to a value

- Most things often in our code are expressions.
- **Statements** is an instruction they perform action and control actions but don’t become values

For example

```
If (x>0){
    Return x;


for(let i=0;i>0;i++){
    return i;}
```

those are statements when we said earlier that a statement cant be a value it means this

```
console.log(const x) // these results in error
```

or

```
let b = if (x>0){
    Return x;} // will result in error
```