ADDIS ABABA INSTITUTE OF TECHNOLOGY

Center Of Information Technology and Scentific computing

Fundamentals of Web Design And Development

Reading Assignment

Name: Bezawit Nigussie

ID: ATR/4920/08

Date: Jan/25/2021

JavaScript

- **Is it Compiled or Interpreted?**

JavaScript is primarily a client-side language.The source code is passed through a program called a compiler, which translates it into bytecode that the machine understands and can execute. In contrast, JavaScript has no compilation step. Instead, an interpreter in the browser reads over the JavaScript code, interprets each line, and runs it. More modern browsers use a technology known as Just-In-Time (JIT) compilation, which compiles JavaScript to executable bytecode just as it is about to run.

- The difference between an interpreted and a compiled language lies in the result of the process of interpreting or compiling.
- An interpreter produces a result from a program, while a compiler produces a program written in assembly language.
- The assembler of architecture then turns the resulting program into binary code. Assembly language varies for each individual computer, depending upon its architecture.
- Consequently, compiled programs can only run on computers that have the same architecture as the computer on which they were compiled.

JavaScript is an interpreted language, not a compiled language.

The History of "Type Of NUll"

In JavaScript are actually values and types created to simulate errors and keywords common in other programming languages. ... These are used in JavaScript to act as placeholders to let the programmer know when a variable has no value.

When a variable is `null` in other programming languages, null is typically a keyword to indicate the space in memory is a pointer (reference), and that pointer is pointing to an invalid memory address (usually 0x0). This is usually used when a programmer

is done using the value of a variable and wants to purposefully clear it by literally pointing it to nothing.

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

he “typeof null” bug is a remnant from the first version of JavaScript. In this version, values were stored in 32 bit units, which consisted of a small type tag (1–3 bits) and the actual data of the value.

The type tags were stored in the lower bits ofthe units. There were five of 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.
- That is, the lowest bit was either one, then the type tag was only one bit long.

  Or it was zero, then the type tag was three bits in length, providing two additional bits, for four types

they are used in JavaScript to act as placeholders to let the programmer know when a variable has no value.

Explain In Detail Why Hosting Is Different With Let And Const?

Let

let is now preferred for variable declaration.

let is block scoped

A block is a chunk of code bounded by {}. A block lives in curly braces. Anything within curly braces is a block. So a variable declared in a block with let is only available for use within that block.

let can be updated but not re-declared, a variable declared with let can be updated within its scope.

if the same variable is defined in different scopes, there will be no error. This is because both instances are treated as different variables since they have different scopes. This fact makes let a better choice than var. When using let, you don't have to bother if you have used a name for a variable before as a variable exists only within its scope.

Hoisting of 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.

Const

Variables declared with the const maintain constant values. const declarations share some similarities with let declarations. const declarations are block scoped. Like let declarations, const declarations can only be accessed within the block they were declared. const cannot be updated or re-declared.

This means that the value of a variable declared with const remains the same within its scope. It cannot be updated or re-declared. So if we declare a variable with const, we can neither do this:

Every const declaration, therefore, must be initialized at the time of declaration. This behavior is somehow different when it comes to objects declared with const. While a const object cannot be updated, the properties of this objects can be updated.

Hosting of const

Just like let, const declarations are hoisted to the top but are not initialized. So just in case you missed the differences, here they are:

- let and const are block scoped.
- let variables can be updated but not re-declared; const variables can neither be updated nor re-declared.
- They are all hoisted to the top of their scope. let and const variables are not initialized.
- const must be initialized during declaration.

Automatic Semicolon Insertion (ASI)

The reason semicolons are sometimes optional in JavaScript is because of automatic semicolon insertion, or ASI.

Here are 3 main points to be aware of when it comes to ASI.

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.

There is also a time we should not to use Semicolons

if (...) {...} else {...}

for (...) {...}

while (...) {...}

Note: You do need one after: do{...} while (...);

If you're going to write your JavaScript without optional semicolons, it's probably good to at least know what ASI is doing.

Also, it can be harder to debug without semicolons since your code may be concatenating together without you realizing it. If you put a line break where there shouldn't be one, ASI may jump in and assume a semicolon even if there shouldn't be one.

Yes, we should always use semicolons, Because if you end up using a JavaScript compressor, all your code will be on one line, which will break your code.

Expression vs Statment

Statements and expressions are two very important terms in JavaScript. Given how frequently these two terms are used to describe JavaScript code, it is important to understand what they mean and the distinction between the two.

“Wherever JavaScript expects a statement, you can also write an expression. Such a statement is called an expression statement. The reverse does not hold: you cannot write a statement where JavaScript expects an expression. For example, an if statement cannot become the argument of a function.”

Expressions

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.

JavaScript has the following expression categories.

- Arithmetic Expressions: evaluate to a numeric value. Examples include the following
- String Expressions: are expressions that evaluate to a string.
- Logical Expressions: Expressions that evaluate to the boolean value true or false are considered to be logical expressions. This set of expressions often involve the usage of logical operators && (AND), ||(OR) and !(NOT).
- Primary Expressions: refer to stand alone expressions such as literal values, certain keywords and variable values.
- Left-hand-side Expressions: Also known as lvalues, left-hand-side expressions are those that can appear on the left side of an assignment expression.
- Assignment Expressions: When expressions use the = operator to assign a value to a variable, it is called an assignment expression.

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 create variables and

functions by using the var and function statements respectively.

- 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. Stand alone primary expressions such as variable values can also pass off as statements depending on the context.

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

- Loops and Jumps : Looping statements includes the following statements: while, do/while, for and for/in. Jump statements are used to make the JavaScript interpreter jump to a specific location within the program.

A function expression, particularly a named function expression, and a function declaration may look the same but their behavior is very different.

A function expression is part of a variable assignment expression and may or may not contain a name.

Function expressions are typically used to assign a function to a variable. Function expressions are evaluated only when the interpreter reaches the line of code where function expressions are located.

On the other hand, function declarations are statements as they perform the action of creating a variable whose value is that of the function. Function declaration falls under the category of declaration statements. Also, function declarations are hoisted to the top of the code unlike function expressions. Function declarations must always be named and cannot be anonymous.