Variables are fundamental building blocks of any programming language. For better understanding any programming language we first have to understand variables. In Javascript, we can define a variable using these three keyword **var**, **let**, **const**. These three keywords behave differently in a different scenario. In this blog, we will look into the lifecycle and scope of the variable.

#### **Prerequisite**

Basic Understanding of Javascript, hoisting , scope, scope chain in javascript

#### **Javascript Engine**

* Before getting into the variables scope and Lifecycle we need to first understand what is javascript engine and javascript engine lifecycle.
* Javascript engines are programs that convert JavaScript code into a lower level or machine code.
* Compilation and Interpretation are some general approaches used in code implementation by programming languages.
* JavaScript is usually categorized as interpreted although it is technically compiled. Modern JavaScript compilers actually perform Just-in-time Compilation which occurs during run-time.
* JavaScript engines are embedded in browsers and web servers, such as Node.js to allow run-time compilation and execution of JavaScript code.  
  these are some Popular JavaScript Engines  
  1)Google V8(used in chrome)  
  2)Spidermonkey(used in Mozilla )  
  3)Chakra(used in Internet Explorer)

#### **Javascript Engine Phases**

When a global code/function is executed it goes through the below phase,  
1)Creation Phase or Compilation  
2)Execution Phase  
**Creation Phase**  
The compiler runs through the entire code for 2 times before actually executing the code  
1)In the first run, It picks all function declarations and stores them in memory with their reference.  
2)In the second run, It picks all variables.  
**Execution Phase**  
In execution phase code is executed

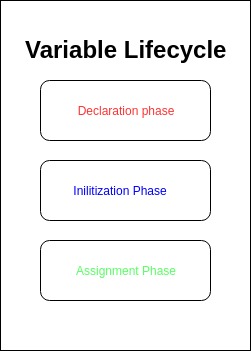
## **Javascript Variable Lifecycle**

Under the hood When the engine works with variables, their lifecycle consists of the following phases

1. **Declaration phase** is registering a variable in the scope.In declaration phase variable is not defined.
2. **Initialization phase** is allocating memory and creating a binding for the variable in the scope. At this step, the variable is automatically initialized with undefined.
3. **Assignment phase** is assigning a value to the initialized variable. For example, let’s see how this really works.

var x=2

1)In the declaration phase, variable **x** is registered. When any variable is registered it add that variable in scope chain of that scope.  
2)in the initialization phase, it’s initialized **x** with **undefined** in the scope. Now x will look like this  
**var x=undefined;**  
3)In Assignment phase 2 is assigned to variable **x** . Now x will look like this.  
**var x=2;**



we discussed the Javascript engine phase and variable lifecycle.if you haven’t understood this don’t worry we will discuss these things in detail.

let’s see some weird behavior of code

console.log(a);// variable is used before it is defined

var a=2;

Do you have any idea what output we will see?  
The output is **undefined**.  
The output is undefined because var is **hoisted**.

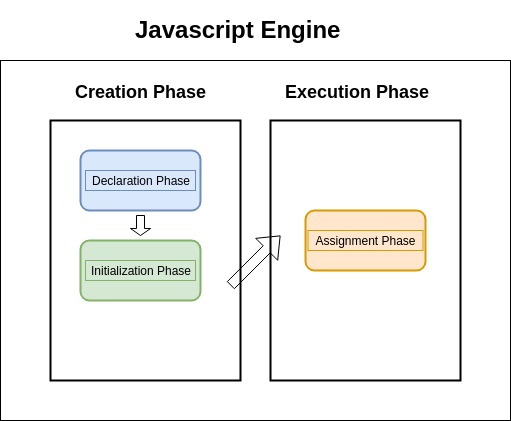
Now let’s take the above example again but this time we declare our variable using **let** instead of **Var**.

console.log(a);// variable is used before it is defined

let a=2;

Now if we run the above code we will get the following error  
**Uncaught ReferenceError: a is not defined**  
if we declare our variable using **const** we will get the same error, which we get in **let**. We are getting an error because let and const are not hoisted  
Why var is hoisted and let , const is not hoisted. Let’s understand this

## **Var Lifecycle**



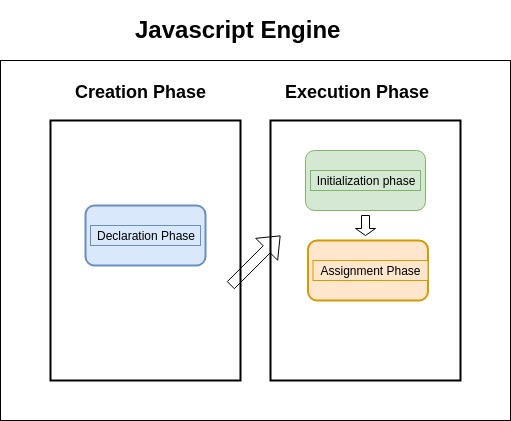
Above diagram will demonstrate the life cycle of var. For understanding this , let’s use the same example which we used previously

console.log(a);//variable is used before it is defined

var a=2;

1. When we give this code to the javascript engine, its creation phase is started. In the creation phase , only the variable is created. In this phase is only sees variable or function declaration. So in the first line when it sees **console.log(a)** it skips this.
2. In line 2 it sees variable declaration for the **variable a**.
3. variable a declared with var is declared and initialized in the creation phase.
4. After the initialization phase variable, **a** is initialized with **undefined** in the scope.
5. After completion of the Creation phase, the Execution phase starts. In the execution phase, it executes code line by line.
6. In line 1, it sees console.log(a). current value of **a** in the scope is **undefined**, so it logs undefined. That’s why Var is hoisted
7. After line 1 it goes to line 2 and value 2 is assigned to that variable.

## **Let Lifecycle**



Now understand this diagram

1. when a variable is declared with let, in the creation phase only Declaration phase of the variable run. In the Declaration phase variable a is registered in scope but not defined.
2. In the execution phase initially, Initialization phase of variable executes. After Initialization phase variable is defined in memory with **undefined**.
3. After Initialization, Assignment phase of variable **a** executes and value is assigned to variable.

//Now lets understand why blow code throw error

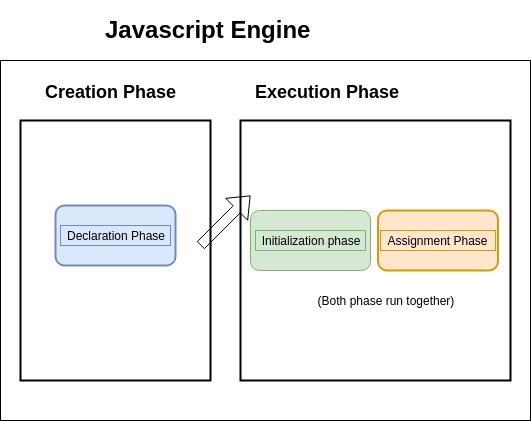
console.log(a);//throw error

let a=2;

1. a is defined with let so, in the creation phase, only variable Declaration phase runs. Here a is registered but not defined
2. When Execution phase of javascript engine runs, it executes code line by line and goes to line 1 and executes line.
3. Javascript engine starts finding in the scope where **a** is defined.but it doesn’t find **a**, so it throws an error. That’s why let and const are not Hoisted  
   **Uncaught ReferenceError: a is not defined**

## **Const Lifecycle**

In the case of const, everything happens the same as let except one thing.



In the case of let after Initialization phase, assignment phase executes.  
But in the case of const Initialization and assignment phase happen together. And this makes sense because we can assign single value time to const. If both phases don’t happen together then initially undefined in assigned after that 2 will assign, which will break const basic definition

## **Difference Between Var, Let, Const**

**Scope**

| **var** | **let** | **const** |
| --- | --- | --- |
| Variables declared with var are in the function scope. | Variables declared as let are in the block scope. | Variables declared as const are in the block scope. |

**Hoisting**

Hoisting, refers to the**default behavior of Javascript** to **process and put all variables and functions declarations into memory first** during compile phase of **its execution context**, regardless where they are written in code.

**Hoisting** means that you can define a variable before its declaration.

| **var** | **let** | **const** |
| --- | --- | --- |
| Allowed | Not allowed | Not allowed |

**Reassign the value**

To reassign a value is to reassign the value of a variable.

| **var** | **let** | **const** |
| --- | --- | --- |
| Allowed | Allowed | Not allowed |

**Redeclaration of the variable**

The redeclaration of a variable means that you can declare the variable again.

| **var** | **let** | **const** |
| --- | --- | --- |
| Allowed | Not allowed | Not allowed |

the differences are:

* var declarations are globally scoped or function scoped while let and const are block scoped.
* var variables can be updated and re-declared within its scope; 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. But while var variables are initialized with undefined, let and const variables are not initialized.
* While var and let can be declared without being initialized, const must be initialized during declaration.

Ref: https://excellencetechnologies.in/blog/javascript-variable-scope-and-lifecycle/