What are Variables?

Variables are containers that are used when we want to store some data in them. We can also imagine a variable as a placeholder for a value like when we go shopping, we carry a bag that contains some fruits inside it, so the bag is the variable and the fruits are the value of the variable.

Just like fruits in a bag, the values of variables may or may not change.

Let's see an example,

```
let name = “Shrikant"
```

In the above code snippet, `name` is a variable which is storing the value `Shrikant.`
It should be noted that a variable can only contain a value at a time, and that value can be of any type.

Declaring variables in JavaScript

In JavaScript, there are different ways with which we can declare variables. These are:

- Using `let` keyword
- Using `const` keyword
- Using `var` keyword

Now let's make use of all these keywords to understand them better.

Using the let keyword

To declare a variable using the `let` keyword, we can consider the example code snippet shown below.

```
let name;
```

```
let isCoding;
```

In the above code, we are creating two variables called `name` and `isCoding`.

The let keyword was introduced in the ES6 version of JavaScript. It is the most popular way of creating variables because it is less error-prone and more stable than the var keyword.

Both the variables that we created in the above code are defined and not assigned any values, and hence their values will be `undefined` for now.

Using the var keyword

Before 2015, there was only one way of creating variables in JavaScript, using the var keyword. Later after the introduction of let and const in 2015, let has been the preferred way of creating variables.

However, people are still using var, so you will find tons of examples where the var is used, although it is more prone to errors.

In JavaScript, we use the var keyword to declare variables as shown below.

```
var name;
var message;
```

In the above code snippet, we are making use of var keywords to create two different variables, namely, `name` and `message`.

One more important thing to note about the var keyword is that if you are making use of older web browsers, then you might have to consider using `var` only, as chances are that the let keyword won't work there at all.

Using the const keyword

The const keyword provides another way of creating variables in JavaScript. If we use const, we are telling our interpreter that the value stored in the variable will remain constant and never change.

It is a very powerful keyword, and you will always notice that it is mainly used when we want to store configuration values of our applications or values that will never change.

It is always recommended that you use const first, and when you explore a scenario in which you need to change the value of the variable, then only consider making use of the let keyword.

Below is a code snippet showing how you can use the const keyword.

```
const age;
const totalAttempts;
```

In the above code snippet, we are making use of the const keyword to declare two constants,
namely, `age` and `totalAttempts`.

Now that we have explored the variable declaration using

the three keywords, it's time we learn how we can assign values to the variables as well.

Assigning values to the variables

Let's create a simple example in which we will make use of all three keywords: var, let, and const, to declare and assign values at the same time.

Consider the code shown below.

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Variable Assignment</title>
</head>
<body>
```

```
    <script>
        let name = "Shrikant Kokate";
        console.log(name);
        const isCoding = "Yes";
        console.log(isCoding);
        var designation = "Senior Software Developer";
        console.log(designation);
    </script>
</body>
</html>
```

In the above code, we have declared three variables, namely, `name`, `isCoding`, and `designation`. At the time of declaration, we are assigning a value to them as well.

Note that we have only assigned text values to the above variables. Instead, we can assign values of any type.

Now let's assign numeric values to the variables as well. We will replace some of the variables that were there in the above example, but the rest of the code will be the same.

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Variable Assignment</title>
</head>
<body>
    <script>
        let age = 25;
        console.log(age);
        var  year = 2022;
        console.log(year);
        const graduateYear = 2019;
        console.log(graduateYear);
    </script>
```

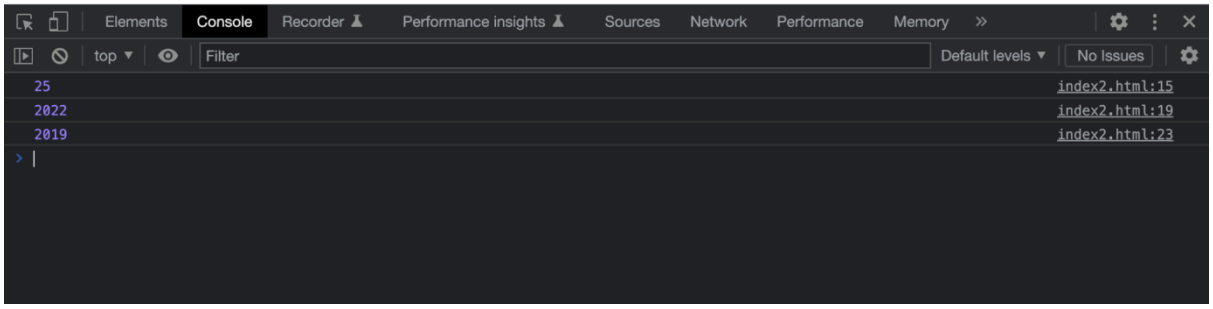
```
</body>
```

```
</html>
```

In the above code example, We assigned numeric values to the variables instead of the text values that we had in the previous example.

If we run the above code in the browser, we will get the following output in the developer console.

Output:



Use of Variables in JavaScript

In JavaScript programs, variables are used everywhere to store a variety of data, and they are an integral part of any code.

It is important to create and store the data using variables. If we directly start working with raw data, there is a high chance that the data will be lost at some point, so storing the data in a variable will help us prevent that.

If the data value is too long or too complicated, it's hard to remember the whole data (something like id number, passport number). Instead, we can store them in a variable and provide an easy-to-remember name that we can use later.

These are some of the benefits of using variables in JavaScript.

Types of Variables

Before we learn about the types of variables, let's first understand about scope. A scope for reference can be defined as the space in which a variable can be accessed within a program in JavaScript.

Based on the scope, we can classify variables into two types:

Global Variables - A global variable is one that has a global scope which means that we are able to use it anywhere in the program.

Local Variables - A local variable is one that can only be accessed within a function inside which it is defined.

Now let's understand what a global scope and local scope variable actually mean.

Consider the code shown below.

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Scope Example</title>
</head>
<body>
    <h1>Global Scope and Local Scope.</h1>
    <script>
```

```
        let isCoding = true;
        function normalFunction() {
            let running = false;
            console.log("Running 1:", running);
            console.log("isCoding 1:", isCoding);
        }
        normalFunction();
        console.log("isCoding 2:", isCoding);
        console.log("isRunning 2:", running);
    </script>
</body>
</html>
```

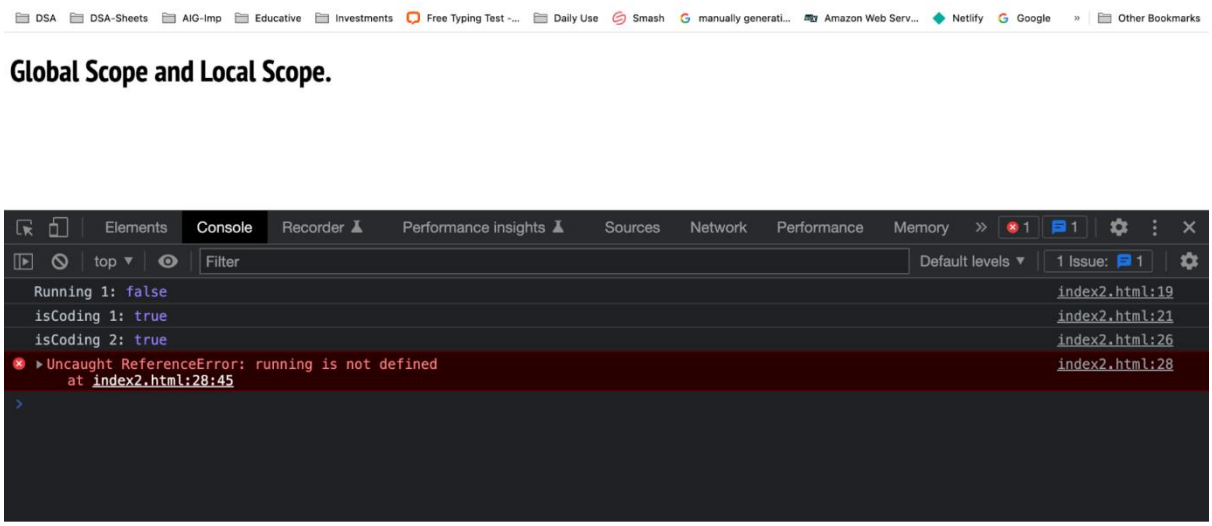
Explanation - In the above code, we need to focus on the code that is written inside the script tag. The variable isCoding is a globally scoped variable, as it is not defined inside any function, and we are able to access it anywhere in the program, and then we created a function named normalFunction.

In normalFunction, a variable named running is initialized and that variable is only available inside that function, if we

try to access that variable outside this function, we will get a reference error.

If we run the above code in the browser, we will get the following output in the developer console.

Output:



The reference error in the above output is because we are trying to access a locally scoped variable outside the function in which it is defined.

It should be noted that since ES2015, there's something called block scope as well, which is the scope in which a variable, once defined, is only available and if we try to access the variable outside that block, it won't be available.

Only the let and const defined variables are blocked scope variables, and the var keyword is either local scoped or global scoped.

Now let's consider an example of block scope as well.

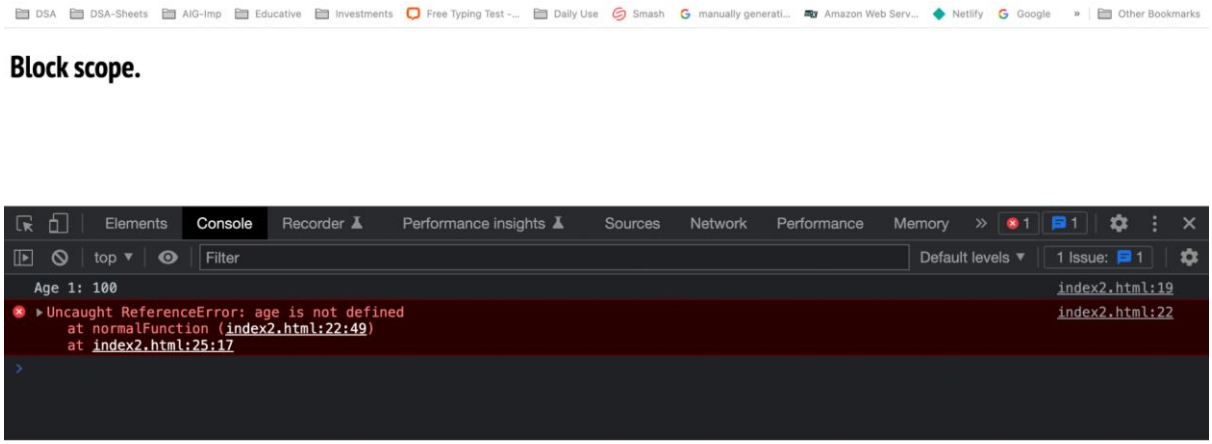
index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Block Scope Example</title>
</head>
<body>
    <h1>Block scope.</h1>
    <script>
        function normalFunction() {
            let running = false;
```

```
        {
            let age = 100;
            console.log("Age 1: " + age);
        }
        console.log("Age 2: " + age);
    }
    normalFunction();
</script>
</body>
</html>
```

Explanation - In the above code, inside the function, we created a block where we declared and initialized a variable named age. We are trying to access the age variable outside the block, that's why we will get the same reference error.

Output:



Variable Names in JavaScript

There are certain rules that we should follow when we are declaring variables in JavaScript. These mainly are:

- Variables must start with a letter, an underscore(_), or the dollar symbol ($).
- Variables cannot be named with numbers at the starting index.
- Variables in JavaScript are case-sensitive.
- Keywords cannot be used as variable names.

Let's understand all of these points with the help of a single example.

Consider the code shown below.

index.html

```
<!DOCTYPE html>
```

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Variable Convention Example</title>
</head>
<body>
    <h1>Variable Convention.</h1>
    <script>
        // Valid Variable Declarations
        let _name = “Shrikant”;
        console.log('_name:' + _name);
        let $age = 25;
        console.log('Age: ' + $age);
        let isStudent = false;
        console.log('isStudent:' + isStudent);
        // InValid Variable Declarations
        let 1age = 25;
```

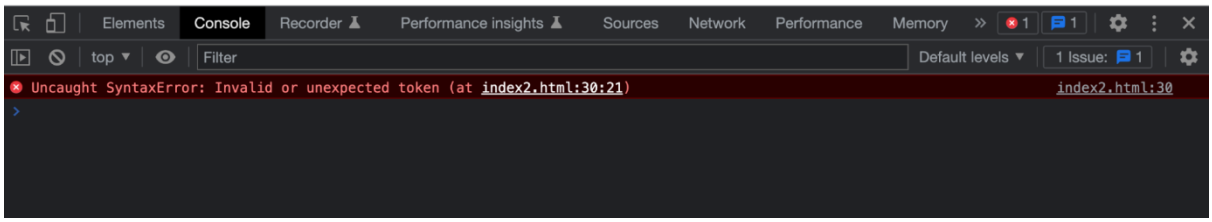
```
        console.log('1age:' + 1age);
        // Cannot use keywords as variable names
        let var = 23;
        console.log('var:' + var);
    </script>
</body>
</html>
```

Explanation - In the above code, we explored valid variable names: _name, $age, and isStudent and invalid variable names: 1age and var.

The above code will generate a runtime error.

Output:

Variable Convention.



Below is the list of keywords that we can't use as variable names in JavaScript.

Keywords

JavaScript Reserved Words

A list of all the reserved words in JavaScript are given in the following table. They cannot be used as JavaScript variables, functions, methods, loop labels, orany object names.

await	break	case	catch	class	const
continue	debugger	default	delete	do	else
enum	export	extends	false	finally	for
function	if	implements	import	in	instanceof
interface	let	new	null	package	private
protected	public	return	super	switch	static
this	throw	try	true	typeof	var
void	while	with	yield		