JavaScript Basics #02

JavaScript is a programming language that drives the web: from front-end user interface design to server-side backend programming, you'll find JavaScript at every stage of a web site and web application. In this course, you'll learn the fundamental programming concepts and syntax of the JavaScript programming language.

Storing and Tracking Information with Variables

Learn how to use variables to store information that changes during a program, like a score in a game, or a sales total. You'll also learn about declaring variables with the keywords let and const.

Introducing Variables

Learn how to use variables to store and track information, as well as use and manipulate information.

JavaScript variables are containers for storing data values.

JavaScript uses reserved keyword **var** to declare a variable. A variable must have a unique name. You can assign a value to a variable using **equal to** (=) operator when you declare it or before using it.

Example: Variable Declaration & Initialization

```
var one = 1; // variable stores numeric value

var two = 'two';  // variable stores string value

var three;  // declared a variable without assigning a value
```

Example: Multiple Variables in a Single Line

```
var one = 1, two = 'two', three;
```

Rules For Naming Variables

Here are six rules you must always follow when giving a variable a name:

1. The name must begin with a letter, dollar sign or underscore. It must not start with a number.
2. The name can contain letters, numbers, dollar sign or an underscore.
3. You cannot use **keywords** or **reserved** words**.**

4. All variables are case sensitive, so score and Score would be different variable names.
5. Use a name that describes the kind of information that the variable stores. For example, fisrtName might be used to store a person's name first name, lastName for their last name and age for their age.
6. If your variable name is made up of more than one word, use a capital letter for the first letter of every word after the first word. For example, firstName rather than firstname.

Challenge 01

Create a variable named myName. Assign it your name between quotes (a string value).

```
let myName = "Marlon";
```

Change the Value in a Variable

Once a variable has been initialized with a value, you can change (or update) that value by giving it a different value.

```
// var variables can be re-declared and updated
// This means that we can do this within the same scope and won't get an error.

var favourite_fruit = "Apple";
console.log(favourite_fruit); //Apple

// re-declared
var favourite_fruit = "Pear";
console.log(favourite_fruit); //Pear

// updated
favourite_fruit = "Kiwi";
console.log(favourite_fruit); //Kiwi
```

Hoisting of var

Hoisting is a JavaScript mechanism where variables and function declarations are moved to the top of their scope before code execution. This means that if we do this:

```
console.log (greeter);
var greeter = "say hello";
```

it is interpreted as this:

```
var greeter;
console.log(greeter); // greeter is undefined
greeter = "say hello"
```

So `var` variables are hoisted to the top of their scope and initialized with a value of `undefined`.

Define Variables with let and const

Learn how `const` and `let` work, and why you might want to use them instead of `var` when declaring variables.

Let

`let` is now preferred for variable declaration. It's no surprise as it comes as an improvement to `var` declarations. It also solves the problem with `var` that we just covered. Let's consider why this is so.

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 me explain this with an example:

```
let greeting = "say Hi";
   let times = 4;

   if (times > 3) {
        let hello = "say Hello instead";
        console.log(hello);// "say Hello instead"
    }
   console.log(hello) // hello is not defined
// since hello is declared inside the block scoped, so cannot be used outside
```

We see that using `hello` outside its block (the curly braces where it was defined) returns an error. This is because `let` variables are block scoped .

let can be updated but not re-declared.

Just like `var`, a variable declared with `let` can be updated within its scope. Unlike `var`, a `let` variable cannot be re-declared within its scope. So while this will work:

```
let greeting = "say Hi";
    greeting = "say Hello instead";
```

this will return an error:

```
    let greeting = "say Hi";
    let greeting = "say Hello instead"; // error: Identifier 'greeting' has already been d
eclared
```

However, if the same variable is defined in different scopes, there will be no error:

```
    let greeting = "say Hi";
    if (true) {
        let greeting = "say Hello instead";
        console.log(greeting); // "say Hello instead"
    }
    console.log(greeting); // "say Hi"
```

Why is there 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.

Also, since a variable cannot be declared more than once within a scope, then the problem discussed earlier that occurs with `var` does not happen.

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`.

```
console.log (name);
let name = "Peter Pan";
// VM91:1 Uncaught ReferenceError: name is not defined at <anonymous>:1:14
```

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:

```
const greeting = "say Hi";
    greeting = "say Hello instead";// error: Assignment to constant variable.

// or this

const greeting = "say Hi";
    const greeting = "say Hello instead";// error: Identifier 'greeting' has already been
 declared
```

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. Therefore, if we declare a `const` object as this:

```
const greeting = {
        message: "say Hi",
        times: 4
    }

// while we cannot do this:

greeting = {
        words: "Hello",
        number: "five"
    } // error:  Assignment to constant variable.


// we can do this:
greeting.message = "say Hello instead";

// This will update the value of greeting.message without returning errors.
```

Hoisting of const

Just like `let`, `const` declarations are hoisted to the top but are not initialized.

```
// For example, this code works fine:
function greetings() {
  console.log(`Hello, ${name}`);
}

let name = "Hannah";

greetings();
// "Hello Hannah"
```

However, this will result in a reference error:

```
function greetings() {
 console.log(`Hello, ${name}`);
}

greetings();

let name = "Hannah";
// Uncaught ReferenceError: name is not defined
```

Recap

- `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.

```
// var is function scope.

// let and const are block scope.Function scope is within the function.
// Block scope is within curly brackets.
```

```
var age = 100; // function scope
  if (age > 12){
    let dogYears = age * 7; // block scope
    console.log(`You are ${dogYears} dog years old!`);
  }
```

Notes

https://www.freecodecamp.org/news/var-let-and-const-whats-the-difference/