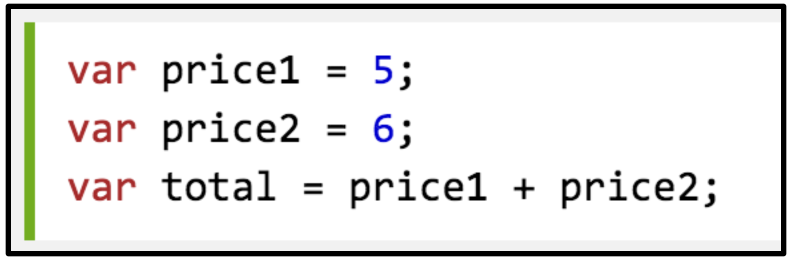
COMP30680
Web Application Development

JavaScript part 2 – Variables and functions

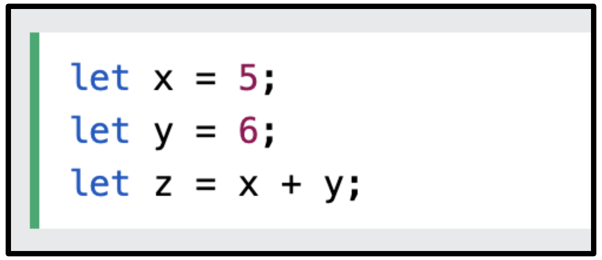
David Coyle
d.coyle@ucd.ie

Variables



```
var price1 = 5;
var price2 = 6;
var total = price1 + price2;
```

JavaScript variables are containers for storing data values.



```
let x = 5;
let y = 6;
let z = x + y;
```

Here z will be equal to 11.

In JavaScript the equal sign (=) is an assignment operator.

All JavaScript **variables** must be **identified** with **unique names**. These unique names are called **identifiers**.

The general rules for constructing identifiers:

- Identifiers can contain letters, digits, underscores, and dollar signs
- Identifiers must begin with a letter, a $ or an _
- Identifiers are case sensitive (y and Y are different variables)
- Reserved words (like JavaScript keywords) cannot be used as Identifiers

JavaScript Data Types

JavaScript variables can hold many **data types**: numbers, strings, booleans, arrays, objects and more.

```
let length = 16;                                // Number
let lastName = "Johnson";                       // String
let cars = ["Saab", "Volvo", "BMW"];            // Array
let x = {firstName:"John", lastName:"Doe"};     // Object
```

We will talk more about Objects later!

“Old” JavaScript has dynamic types. This means that the same variable can be used as different types.

```
var x;               // Now x is undefined
var x = 5;           // Now x is a Number
var x = "John";      // Now x is a String
```

```
let x;               // Now x is undefined
let x = 5;           // Now x is a Number
let x = "John";      // This is not allowed.
```

JavaScript Data Types: string, number, boolean

A string (or a text string) is a series of characters like "John Doe". Strings are written with quotes. You can use single or double quotes. For more details of strings see: http://www.w3schools.com/js/js_strings.asp.

```
let carName = "Volvo XC60";   // Using double quotes
let carName = 'Volvo XC60';   // Using single quotes
```

JavaScript has only one type of numbers. Numbers can be written with, or without decimals. For more details of numbers see: http://www.w3schools.com/js/js_numbers.asp.

```
let x1 = 34.00;      // Written with decimals
let x2 = 34;         // Written without decimals
```

Booleans can only have two values: true or false. For more details of Booleans see: http://www.w3schools.com/js/js_booleans.asp

```
let x = true;
let y = false;
```

Data Types and declarations

A variable declared without a value will have the value **undefined**.

```
let carName;
```

The variable carName will have the value undefined after the execution of this statement.

If you re-declare a JavaScript variable, it does not lose its value.

```
let carName = "Volvo";
let carName;
```

The variable carName will still have the value "Volvo" after the execution of these statements.

You can perform “additions” on both text and numbers, but be careful:

```
let x = 5 + 2 + 3;
```

gives number 10

```
let x = "John" + " " + "Doe";
```

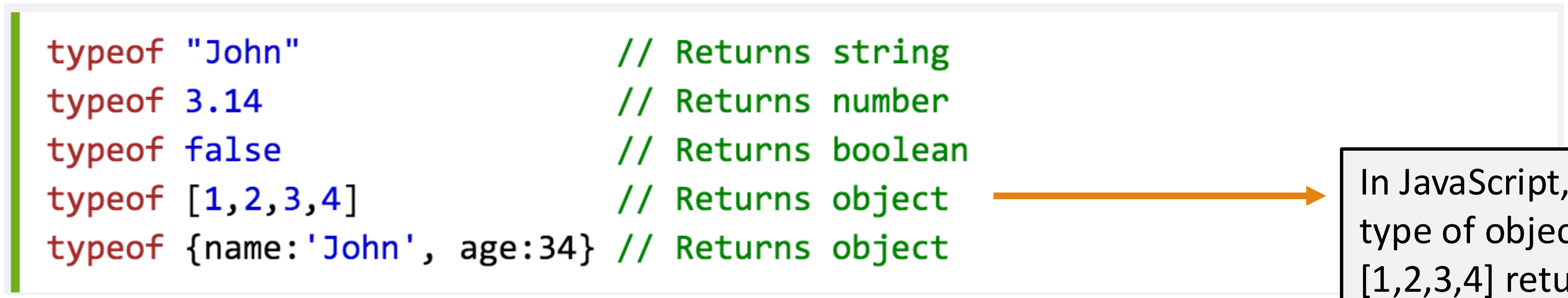
gives text “John Doe”

```
let x = "5" + 2 + 3;
```

gives text “523”

JavaScript Data Types

You can use the JavaScript **typeof** operator to find the type of a JavaScript variable:



```
typeof "John"                 // Returns string
typeof 3.14                   // Returns number
typeof false                  // Returns boolean
typeof [1,2,3,4]              // Returns object
typeof {name:'John', age:34} // Returns object
```

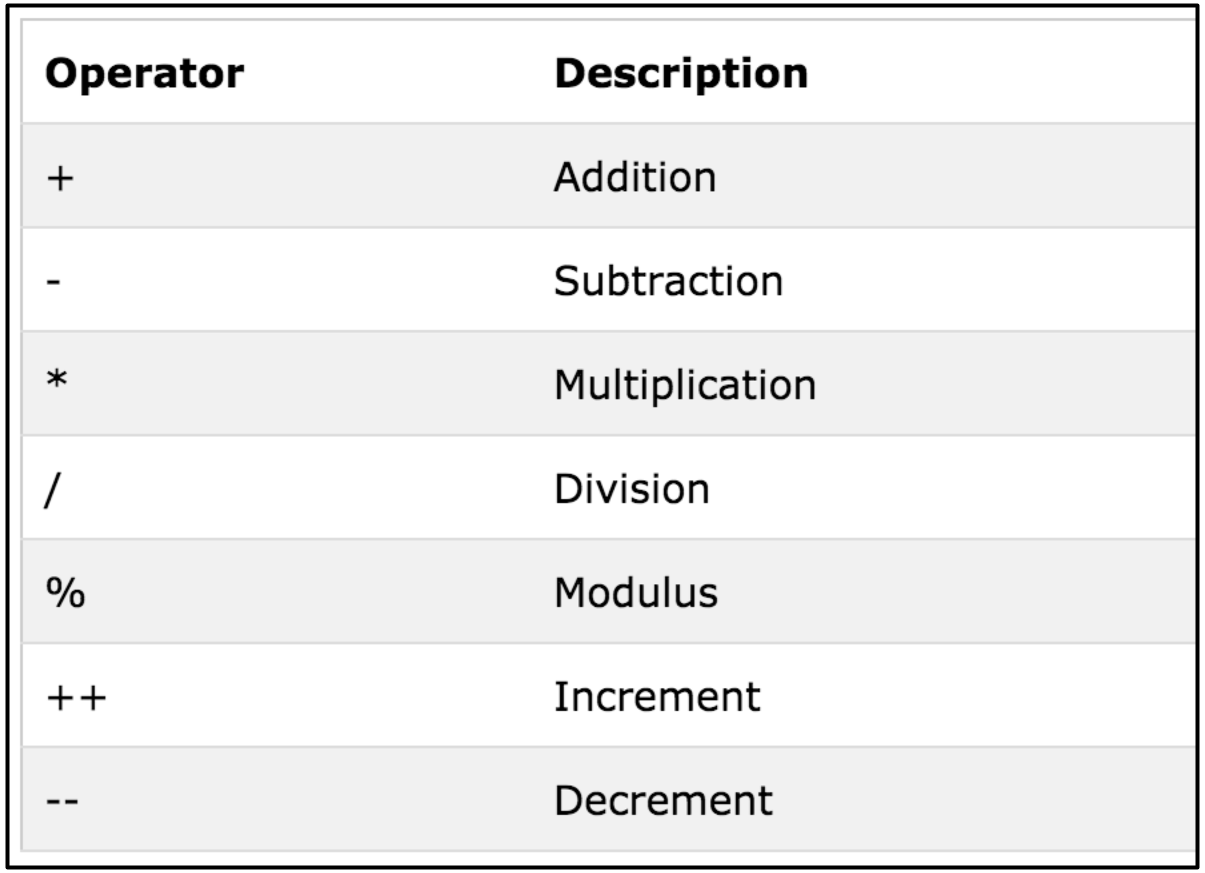
In JavaScript, an array is a special type of object. Therefore typeof [1,2,3,4] returns object.

A variable without a value, has the value **undefined**. The typeof is also **undefined**.

Any variable can be emptied, by setting the value to **undefined**.

Arithmetic Operators

Arithmetic operators perform arithmetic on numbers:



Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
++	Increment
--	Decrement

Operator precedence describes the order in which operations are performed in an arithmetic expression.

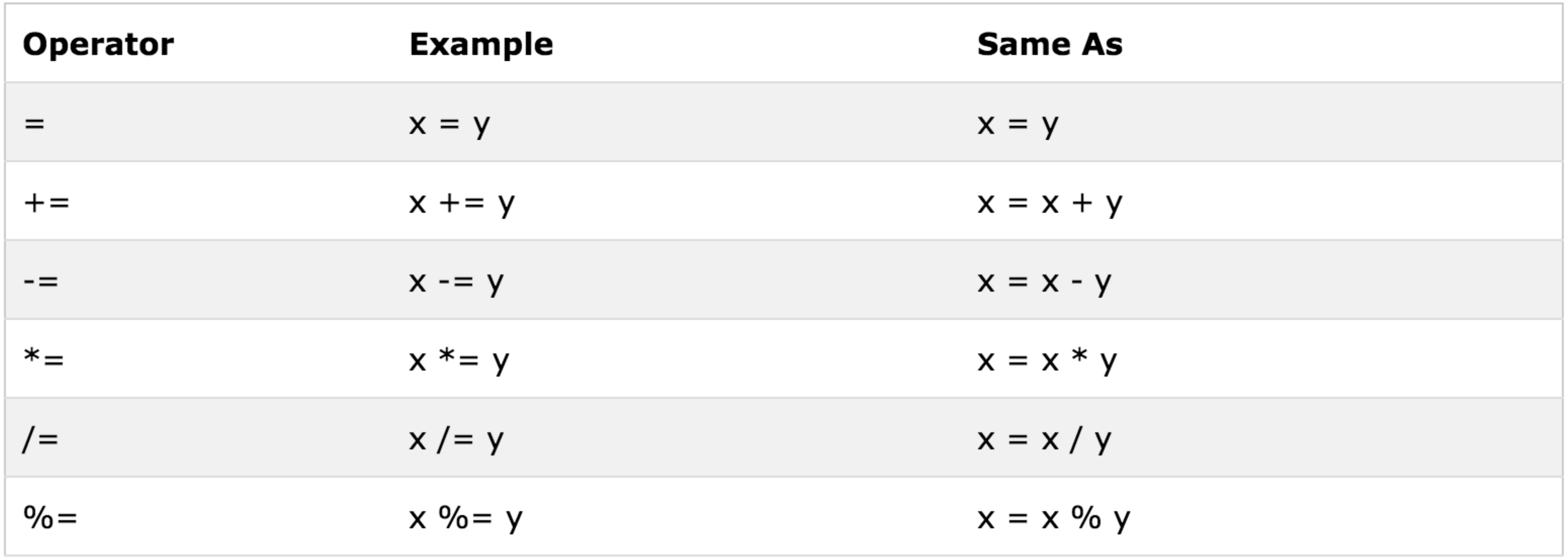
As in traditional mathematics for example, multiplication (*) and division (/) have higher **precedence** than addition (+) and subtraction (-).

Precedence can be changed by using parentheses

For a more complete precedence list see: http://www.w3schools.com/js/js_arithmetic.asp

Assignment Operators

Assignment operators assign values to JavaScript variables.



Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

JavaScript Data Types: Arrays

An array is a special variable, which can hold more than one value at a time.

JavaScript arrays are written with square brackets.

Array items are separated by commas.

The following code declares (creates) an array called cars, containing three items (car names):

```
let cars = ["Saab", "Volvo", "BMW"];
```

Spaces and line breaks are not important. A declaration can span multiple lines:

```
let cars = [
    "Saab",
    "Volvo",
    "BMW"
];
```

The following also creates an Array, and assigns values to it:

```
let cars = new Array("Saab", "Volvo", "BMW");
```

The two examples above do exactly the same. There is no need to use new Array().
For simplicity, readability and execution speed, use the first one (the array literal method).

Arrays: Accessing the elements of an Array

Array indexes are zero-based, which means the first item is [0], second is [1], and so on.

You refer to an array element by referring to the **index number**.

This statement accesses the value of the first element in cars:

```
let name = cars[0];
```

This statement modifies the first element in cars:

```
cars[0] = "Opel";
```

The best way to loop through an array, is using a "for" loop.

```
let index;
let fruits = ["Banana", "Orange", "Apple", "Mango"];
for (index = 0; index < fruits.length; index++) {
    text += fruits[index];
}
```

Arrays: properties and methods

A real strength of JavaScript arrays are the built-in array properties and methods.

E.g. the **length** property of an array returns the length of an array (the number of array elements). On the previous slide we saw the **length** property used in a for loop.

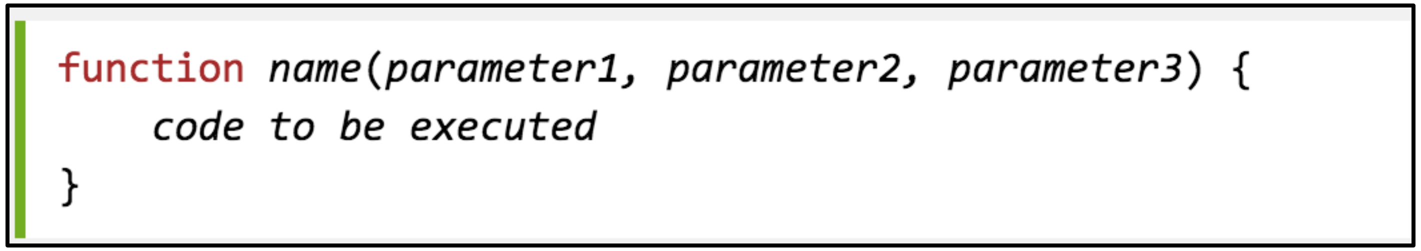
Common array methods include:

toString()	- converts an array to a string of (comma separated) array values.
pop()	- removes the last element from an array.
push()	- adds a new element to an array (at the end).
shift()	- removes the first array element and "shifts" all other elements to a lower index.
unshift()	**-** adds a new element to an array (at the beginning), and "unshifts" older elements.
sort()	- sorts an array alphabetically.
reverse()	- reverses the elements in an array.

For a complete array reference see: http://www.w3schools.com/jsref/jsref_obj_array.asp. For examples see: http://www.w3schools.com/js/js_array_methods.asp.

Functions

A JavaScript function is a block of code designed to perform a particular task.



```
function name(parameter1, parameter2, parameter3) {
    code to be executed
}
```

A JavaScript function is defined with the **function** keyword, followed by a **name**, followed by parentheses **()**.

Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).

The parentheses may include parameter names separated by commas: **(*parameter1, parameter2, ...*)**

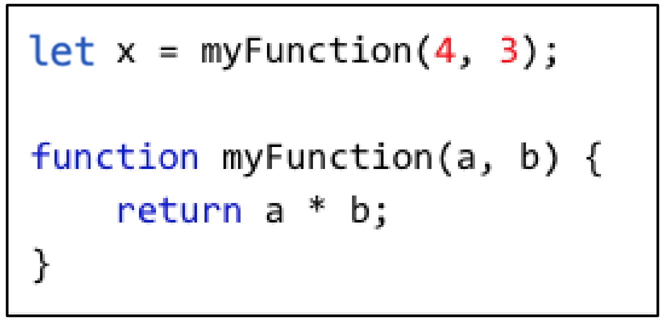
The code to be executed, by the function, is placed inside curly brackets: **{}**

The code in a function is not executed when the function is **defined**. It is executed when the function is executed when "something" **invokes** (calls) the function.

Function invocation and return

The code inside the function will execute when "something" **invokes** (calls) the function:

- When an event occurs (when a user clicks a button)
- When it is invoked (called) from JavaScript code
- Automatically (self invoked)



```
let x = myFunction(4, 3);

function myFunction(a, b) {
    return a * b;
}
```

Note the syntax here. The function is invoked using the ( ) operator.

When JavaScript reaches a **return statement**, the function will stop executing.

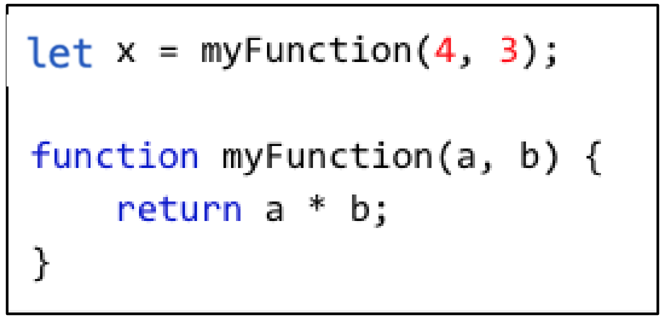
If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.

Functions often compute a **return value**. The return value is "returned" back to the "caller":

Function Parameters and Arguments

Function **parameters** are the **names** listed in the function definition.

Function **arguments** are the real **values** passed to (and received by) the function.

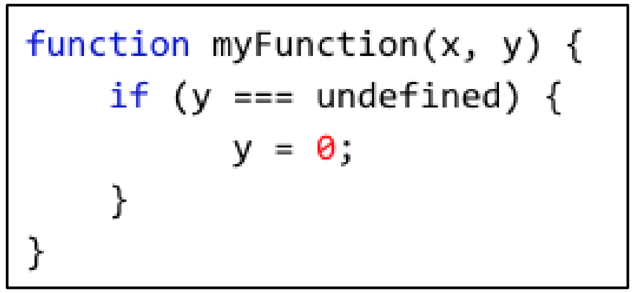


```
let x = myFunction(4, 3);

function myFunction(a, b) {
    return a * b;
}
```

Parameter Rules:

- JavaScript function definitions do not specify data types for parameters.
- JavaScript functions do not perform type checking on the passed arguments.
- JavaScript functions do not check the number of arguments received.



```
function myFunction(x, y) {
    if (y === undefined) {
        y = 0;
    }
}
```

If a function is called with **missing arguments** (less than declared), the missing values are set to **undefined**

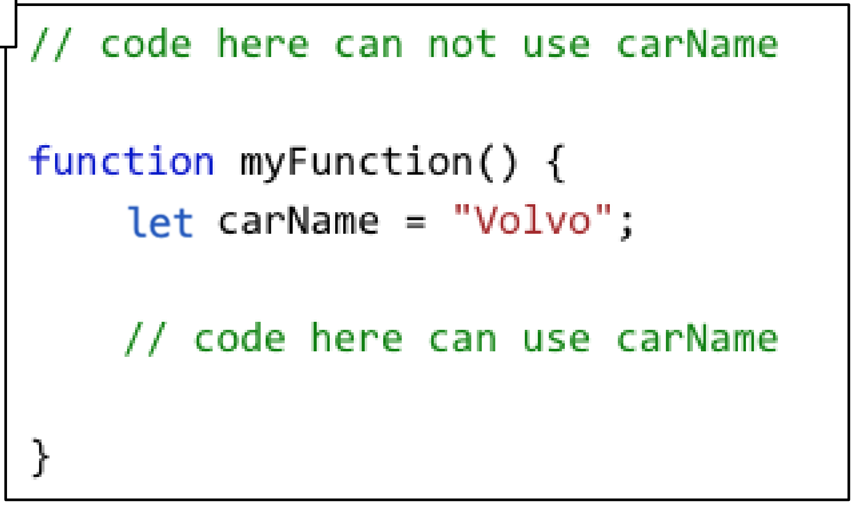
Scope

In JavaScript, scope is the set of variables, objects, and functions you have access to.

The key distinction is between **local** and **global** variables. JavaScript has function scope: The scope changes inside functions:

- A variables defined inside a function only have local scope, i.e. it local to that function.
- A variable defined outside a function has global scope.

Local



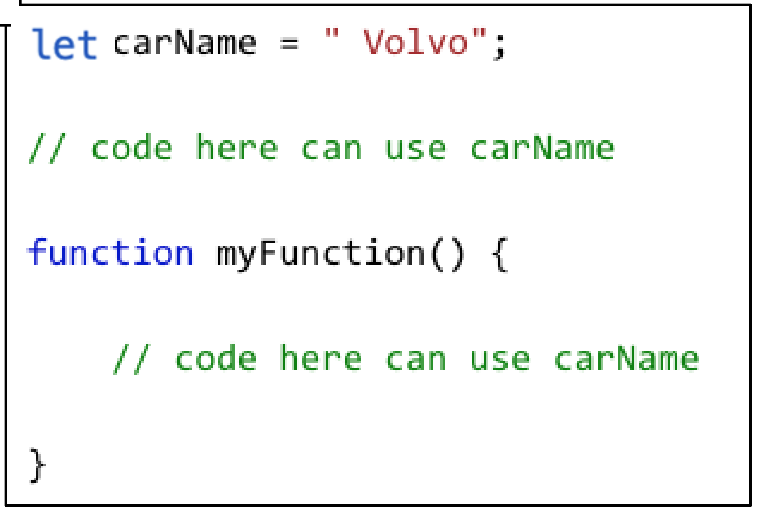
```
// code here can not use carName

function myFunction() {
    let carName = "Volvo";

    // code here can use carName

}
```

Global



```
let carName = " Volvo";

// code here can use carName

function myFunction() {

    // code here can use carName

}
```

If you assign a value to a variable that has not been declared, it will automatically become a **GLOBAL** variable.

Questions, Suggestions?

Next class:

JavaScript part 3 – Conditional statements and loops