Variables and Types

JavaScript is a loosely typed language. This means that you can use the same variable for different types of information, but you may also have to check what type a variable is yourself, if the differences matter. For example, if you wanted to add two numbers, but one variable turned out to be a string, the result wouldn't necessarily be what you expected.

Variable declaration

Variables are commonly explicitly declared by the `var` statement, as shown below:

```
var c;
```

Doing so is obligatory, if the code includes the string comment `"use strict;"`. The above variable is created, but has the default value of `undefined`. To be of value, the variable needs to be initialized:

```
var c = 0;
```

After being declared, a variable may be assigned a new value that will replace the old one:

```
c = 1;
```

But make sure to declare a variable with `var` before (or while) assigning to it; otherwise you will create a "scope bug."

Primitive types

Primitive types are types provided by the system, in this case by JavaScript. Primitive type for JavaScript are Booleans, numbers and text. In addition to the primitive types, users may define their own classes.

The primitive types are treated by JavaScript as value types and when passed to a function, they are passed as values. Some types, such as string, allow method calls.

Boolean type

Boolean variables can only have two possible values, true or false.

```
var mayday = false;
var birthday = true;
```

Numeric types

You can use an integer and double types on your variables, but they are treated as a numeric type.

```
var sal = 20;
var pal = 12.1;
```

In the ECMA JavaScript specification, your number literals can go from 0 to -+1.79769e+308. And because 5e-324 is the smallest infinitesimal you can get, anything smaller is rounded to 0.

String types

The `String` and `char` types are all strings, so you can build any string literal that you wished for.

```
var myName = "Some Name";
var myChar = 'f';
```

Complex types

A complex type is an object, be it either standard or custom made. Its home is the heap and is always passed by reference.

Array type

In JavaScript, all Arrays are untyped, so you can put everything you want in an Array and worry about that later. Arrays are objects, they have methods and properties you can invoke at will. For example, the `.length` property indicates how many items are currently in the array. If you add more items to the array, the value of the `.length` gets larger. You can build yourself an array by using the statement `new` followed by `Array`, as shown below.

```
var myArray = new Array(0, 2, 4);
var myOtherArray = new Array();
```

Arrays can also be created with the array notation, which uses square brackets:

```
var myArray = [0, 2, 4];
var myOtherArray = [];
```

Arrays are accessed using the square brackets:

```
myArray[2] = "Hello";
var text = myArray[2];
```

There is no limit to the number of items that can be stored in an array.

Object types

An object within JavaScript is created using the new operator:

```
var myObject = new Object();
```

Objects can also be created with the object notation, which uses curly braces:

```
var myObject = {};
```

JavaScript objects can implement inheritance and support overriding, and you can use polymorphism. There are no scope modifiers, with all properties and methods having public access..

You can access browser built-in objects and objects provided through browser JavaScript extensions.

Scope

In JavaScript, the scope is the current context of the code. It refers to the accessibility of functions and variables, and their context. There exists a *global* and a *local* scope. The understanding of scope is crucial to writing good code. Whenever the code accesses `this`, it accesses the object that "owns" the current scope.

Global scope[edit]

An entity like a function or a variable has **global scope**, if it is accessible from everywhere in the code.

```
var a = 99;

function hello() {
  alert("Hello, " + a + "!");
}
hello();    // prints the string "Hello, 99!"
alert(a);   // prints the number 99
console.log("a = " + a); // prints "a = 99" to the console of the browser
```

Here, the variable `a` is in global scope and accessible both in the main code part and the function `hello()` itself. If you want to debug your code, you may use the `console.log(...)` command that outputs to the console window in your browser. This can be opened under the Windows OS with the F12 key.

Local scope[edit]

A **local scope** exists when an entity is defined in a certain code part, like a function.

```
var a = 99;

function hello() {
  var x = 5;
  alert("Hello, " + (a + x) + "!");
}
hello();    // prints the string "Hello, 104!"
alert(a);   // prints the number 99
alert(x);   // throws an exception
```

If you watch the code on a browser (on Google Chrome, this is achieved by pressing F12), you will see an **`Uncaught ReferenceError: x is not defined`** for the last line above. This is because `x` is defined in the local scope of the function hello and is not accessible from the outer part of the code.