The Learning House

427 S. 4th Street #300

Louisville KY 40202

02 – JavaScript Data Types

.NET Cohort

Coding Bootcamp

SOFTWARE GUILD

Lesson Goals

1. Learn about the JavaScript language and how it is similar and different from other C-variants like C#
2. Learn about some downright weird (to us, C# types) things about JavaScript Data Types

Syntax

- JavaScript, like C#, is case-sensitive; e.g. myVar is a different variable than myvar.
- For variables, functions, and properties, the name (identifier) may start with a-z, underscore _, or a dollar sign $.
  - After the first character, you can also use numbers
- It's not a requirement, but convention in JavaScript is that all identifiers are Camel Case, so the first letter is lowercase and each additional word has a capital letter:
  - doSomething
  - firstNumber
  - savePersonToDatabase
- Code comments are just like C#.
  - // for a single line comment
  - /* */ for multiple line comments

Our Friend, the Semi-Colon

In JavaScript, like C#, the end of statement is denoted by a semi-colon. There are some cases where the semi-colon is not required, but using it just as you would in C# is considered best practice.

Variables

Unlike in C#, variables in JavaScript are *loosely typed*, meaning that a variable can hold any type of data and can even change data type during execution. The preferred method of defining a variable is to use the *var* keyword:

```
var message;
message = "hello world";
```

Notice that we can change the type of message during execution:

```
var message;
message = "hello world";
message = 100;
```

This is generally frowned upon, because it makes debugging very difficult.

Omitting the var Keyword

In JavaScript, a variable's scope in a function (recall that scope means where we can use it) is limited to the current function if the var keyword is used. If we omit the var keyword, the variable is global.

```
function sayHi() {
    var message;
    message = "hello world";
}

sayHi();
console.log(message); // error
```

```
function sayHi() {
    message = "hello world"; // global
}

sayHi();
console.log(message); // prints hello world
```

Omitting var, in this case, is not best practice!

Multiple Declarations

We can also declare multiple variables at a time, if we like, by comma-separating them.

```
var message = "hello world",
    found = false,
    age = 30;
```

Note: it's not required to put each variable on a new line, but I find this easier to read.

Data Types

While there are many data types in a strictly-typed language like C#, in a loosely-typed language like JavaScript, there are significantly fewer.

Because we can store any type in variables, JavaScript provides a typeof() operator which returns a string of what type a variable currently holds.

The typeof(variable) statement will return one of six values:

- “undefined” if the value is undefined (null)
- “boolean” if the value is true/false
- “string” if it is a string
- “number” if it is any number
- “object” if it is an object (or null)
- “function” if the variable contains a function

Undefined

The undefined type is a special value in JavaScript that is the value returned when a variable is declared using var, but never assigned a value.

```
var myVar;
alert(myVar == undefined);  // true
```

Undefined is mainly used for comparison situations, to make sure something has a value. It's somewhat frowned upon to explicitly set a variable to undefined.

Also note that if you call typeof() on a variable that hasn't been declared, it will return undefined, but using the variable will cause an error.

Null

Null is a special data type that represents an empty object. It is considered best practice when declaring a variable that will store an object later to set it to null.

```
var person = null;
if (person != null) {
    // do something with person
}
```

Interestingly enough, if we compare null to undefined, it will return true even though they have different uses.

Boolean

Much like in C#, Booleans in JavaScript can only have values of true and false. Unlike some languages, true is not equal to 1 and false is not equal to 0 in JavaScript.

true and false are case sensitive and must be in lower case to be used literally.

We can use the Boolean() casting function to convert a value to a Boolean:

```
var message = "hello world";
var messageBoolean = Boolean(message);
```

Boolean() Conversions

Data Type	True Value	False Value
String	Any non-empty string	“” (empty string)
Number	Any non-zero number	0, NaN (not a number)
Object	Any object	null
Undefined		Always false

Numbers

JavaScript can store numbers as integers, floats, octal (base 8), and hexadecimal (base 16).

Numbers can range from 5e-324 to 1.7976931348623157e+308

Any number outside those bounds will be set to a literal *infinity*.

NaN is a special keyword which indicates that a number setting operation has failed. A good example is dividing by zero: in most languages, that would throw an error; but in JavaScript, it simply passes back a result of NaN.

We can use the *isNaN(variable)* to determine true/false whether a number setting was successful.

Testing for NaN

Here are some examples of isNaN in action:

```
alert(isNaN(NaN));       // true
alert(isNaN(10));        // false, it is numeric
alert(isNaN("10"));      // false, converted to 10
alert(isNaN("blue"));    // true
alert(isNaN(true));      // false, converted to 1
```

Converting Numbers

JavaScript provides three functions to convert values to numbers:

- Number() can be used on any data type
- parseInt and parseFloat can only be used on strings

Number() Conversions

Applied to	Returns
Boolean	0 or 1
Numbers	the number
null	0
undefined	NaN
empty string	0
string that isn't numeric	NaN

parseInt

parseInt is the preferred method for converting strings to integers. This function will scan the string from left to right. If the string doesn't start with a number, it will return NaN; otherwise, it will start parsing and stop when it reaches a non-numeric character. This is a bit odd because "1234blue" will convert to 1234.

```
var num1 = parseInt("1234blue");    //1234
var num2 = parseInt("");            //NaN
var num3 = parseInt("0xA");         //10 - hexadecimal
var num4 = parseInt(22.5);          //22
var num5 = parseInt("70");          //70 - decimal
var num6 = parseInt("0xf");         //15 - hexadecimal
```

Notice that it also handles hexadecimal values and truncates decimal remainders, instead of rounding up.

parseFloat

parseFloat is used similarly to parseInt, but it preserves the decimal precision. Because hexadecimal variables are stored as integers, note that parseFloat will convert them to 0.

Also note that if the number is an integer, the function returns it as an integer (to save space).

```
var num1 = parseFloat("1234blue");     //1234 - integer
var num2 = parseFloat("0xA");          //0
var num3 = parseFloat("22.5");         //22.5
var num4 = parseFloat("22.34.5");      //22.34
var num5 = parseFloat("0908.5");       //908.5
var num6 = parseFloat("3.125e7");      //31250000
```

Strings

Strings can be contained by either single or double quotes, and both are exactly the same (unlike some other languages):

```
var name = 'Jimbo Jones';
var city = "Atlanta";
```

Just like in C#, we can embed special characters in a string using the \ syntax. Our familiar \n, \t,]r, \\, \', and \" are all there.

Also like in C#, most values have a toString() method to convert it to a string.

```
var age = 11;
var ageAsString = age.toString();    //the string "11"
var found = true;
var foundAsString = found.toString(); //the string "true"
```

String() Function

Calling toString() on a null or undefined value will cause an error, so if you aren't sure, use the String() function.

```
var value1 = 10;
var value2 = true;
var value3 = null;
var value4;

alert(String(value1));      //"10"
alert(String(value2));      //"true"
alert(String(value3));      //"null"
alert(String(value4));      //"undefined"
```

Objects

Similar to C#, but not exactly the same, objects in JavaScript are created by using the new operator followed by the name of an object to create.

We will cover objects later, but here is the basic syntax for creating a new object:

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

Conclusion

There are fewer data types to keep track of in JavaScript, but where you will get tripped up as a C# developer is the differences in conversions.