SOFTWARE GUILD

JavaScript Operators, Statements, and Functions

.NET Cohort

SOFTWARE GUILD

Lesson Goals

1. Learn about the JavaScript language, how it is similar and different from other C-variants like C#
2. Learn the common operators in JavaScript
3. Learn how JavaScript handles code statements
4. Learn how JavaScript defines functions (methods)

Operators

JavaScript has many of the same operators as C# and most of them behave the same way.

Unary Operators

Increment/Decrement

Like C#, JavaScript supports ++ and -- operators to add or subtract one from a numeric type.

When used on a string that can be converted to a number, it converts it to a number and applies the change.

For a string that cannot be converted to a number, the value is changed to NaN.

A Boolean will convert to 0 (false) or 1(true), then apply the change.

An object will call valueOf() then toString().

```
var s1 = '2';
var s2 = 'z';
var b = false;
var f = 1.1;
var o = {
    valueOf: function() {
        return -1;
    }
};

s1++; //value becomes numeric 3
s2++; //value becomes NaN
b++; //value becomes numeric 1
f--; //value becomes 0.10000000000000009
o--; //value becomes numeric -2
```

Unary Plus and Minus

The plus operator will add two numbers, or convert a different type to a number.

The minus operator (left) will convert a positive number to negative or a negative number to positive.

```
var s1 = '01';
var s2 = '1.1';
var s3 = 'z';
var b = false;
var f = 1.1;
var o = {
    valueOf: function() {
        return -1;
    }
};

s1 = -s1; //value becomes numeric -1
s2 = -s2; //value becomes numeric -1.1
s3 = -s3; //value becomes NaN
b = -b; //value becomes numeric 0
f = -f; //change to -1.1
o = -o; //value becomes numeric 1
```

Logical NOT

The logical NOT operator is much like C# and is represented by an exclamation point (!).

This operator always returns a Boolean value, regardless of what data type it's used on.

It is often used to check whether strings have any characters or whether something is null or NaN.

Sometimes, developers chain two NOTs to get the inverse of the value; e.g. !!'blue' returns true because there is data in the string.

```
alert(!false); //true
alert(!'blue'); //false
alert(!0); //true
alert(!NaN); //true
alert(!''); //true
alert(!null); // true
alert(!12345); //false
```

Logical AND

Logical AND is represented by double ampersand (&&) and can be used on any type of operand.

If the first operand is an object, then the second is always returned.

If the second operand is an object, then the object is only returned if the first operand is true.

If both operands are objects, the second is returned.

If either is null, then null is returned.
If either is NaN, then NaN is returned.
If either is undefined, then undefined is returned.

Logical OR

Logical OR is represented by double pipe (||) and can be used on any type of operand.

If the first operand is an object, then the first is always returned.

If the first operand evaluates to false, then the second is returned.

If both operands are objects, the first is returned.

If both are null, then null is returned
If both are NaN, then NaN is returned
If both are undefined, then undefined is returned.

Other Operators

We can also do multiplication (*), division (/), and modulus (%).

For extra strangeness, there is a keyword called *Infinity* which represents an infinite number.

Testing for Equality

In C#, we use == to check for equality, and it’s the only operator we need because of the type system. However, JavaScript has two operators:

1. == converts the value and checks for equality. For example, (‘10’ == 10) would be true because of type conversion.
2. === returns true only if the data types are equal without conversion. So, (‘10’ === 10) would be false.

Conditional Operator

The conditional operator is the same as in C#:

```
variable = boolean_expression ? true_value : false_value
```

It’s basically a short cut if/else:

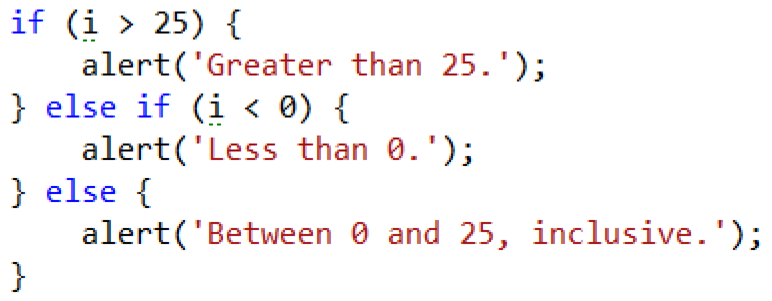
```
var max = (num1>num2) ? num1 : num2;
```

Is the same as:

```
if(num1 > num2) { max = num1; } else {max = num2 };
```

if Statement

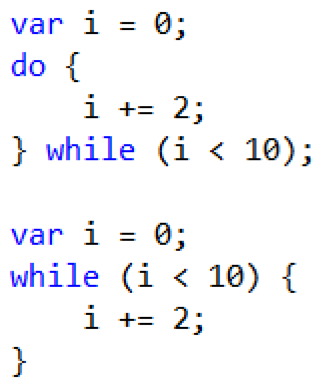
No surprises here, it’s just like C#.



```
if (i > 25) {
    alert('Greater than 25.');
} else if (i < 0) {
    alert('Less than 0.');
} else {
    alert('Between 0 and 25, inclusive.');
}
```

while and do-while

No surprises here either:

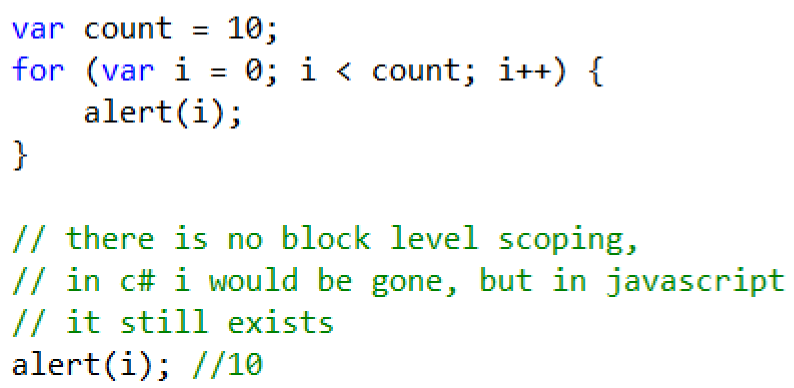


```
var i = 0;
do {
    i += 2;
} while (i < 10);

var i = 0;
while (i < 10) {
    i += 2;
}
```

for Loops

These are mostly the same, with one caveat. In JavaScript, variables declared in statements (even if statements), do not have block level scoping.



```
var count = 10;
for (var i = 0; i < count; i++) {
    alert(i);
}

// there is no block level scoping,
// in c# i would be gone, but in javascript
// it still exists
alert(i); //10
```

for-in

The for-in statement iterates through all the properties of an object. Note that for-in does not do any ordering, so don't depend on things being returned in a particular order.

```
for (var propName in window) {
    document.write(propName);
}
```

break and continue

These statements behave just as they do in C#. Break will leave a loop and continue will stop execution and go back to the top of a loop.

switch Statements

While the structure of switch statements is the same as C#, they have some additional capabilities in JavaScript, mainly that case values can be expressions instead of constants.

As long as the case evaluates to true, the code will be executed.

```
switch (i) {
    case 25:
        /* falls through */
    case 35:
        alert('25 or 35');
        break;
    case 45:
        alert('45');
        break;
    default:
        alert('Other');
}

switch ('hello world') {
    case 'hello' + ' world':
        alert('Greeting was found.');
        break;
    case 'goodbye':
        alert('Closing was found.');
        break;
    default:
        alert('Unexpected message was found.');
}
```

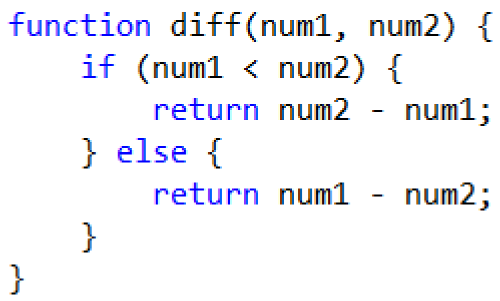
Functions

Functions are what we call Methods in C#. They encapsulate blocks of code for reuse. There are some key differences in how functions are defined in JavaScript. In particular, overloading is not allowed, and a return type is not required.

```
function sayHi(name, message) {
    alert('Hello ' + name + ', ' + message);
}
```

return Statements

We can return values from a function using the return statement. If there is no return statement, the function is considered the equivalent of void in C#.



```
function diff(num1, num2) {
    if (num1 < num2) {
        return num2 - num1;
    } else {
        return num1 - num2;
    }
}
```

arguments

arguments is a special variable. It is an array of all the arguments passed to a method. Unlike in C#, you can pass as few or as many arguments as you like to a function, whether they are defined parameters or not.

This is weird to people coming from a static typed language.

This also means we can't overload a method because JavaScript doesn't care about types.

```
function doAdd() {
    if (arguments.length == 1) {
        alert(arguments[0] + 10);
    } else if (arguments.length == 2) {
        alert(arguments[0] + arguments[1]);
    }
}

doAdd(10); //20
doAdd(30, 20); //50

function doAdd(num1, num2) {
    if (arguments.length == 1) {
        alert(num1 + 10);
    } else if (arguments.length == 2) {
        alert(arguments[0] + num2);
    }
}

doAdd(10); // 20
doAdd(30, 20); // 50
```

No Overloading

If you try to declare the same named function twice with the same parameters, the second will replace the first.

```
function addSomeNumber(num) {
    return num + 100;
}

function addSomeNumber(num) {
    return num + 200;
}

var result = addSomeNumber(100); //300
```

SOFTWARE GUILD

Conclusion

As you can see, JavaScript, being rooted in C, has a lot in common with C# and many of the common structures we're already used to, like statements and loops, translate over.

All languages have these concepts, so learning new languages after you master one is not all that difficult.