In this chapter you will learn the basics constructs of programming. These constructs are similar in a number of programming languages, and will be used in a number of our scripts in later chapters

3 Objectives

1. Declaring Variables
2. Using Operators
3. Creating Control Structures (Branches and Loops)
4. Functions (Built-in and programmer-created)

Declaring Your Variables

A variable is a name assigned to a location in a computer's memory to store data. Before you can use a variable in a JavaScript program, you must declare its name. Variables are declared with the **var** keyword, like this:

var x;

var y;

var sum;

You can also declare multiple variables with the same **var** keyword.

var x, y, sum;

To take it one step further you can combine variable declaration with initialization.

var x = 1, y = 3, sum;

If you don't specify an initial value for a variable when you declare it, the initial value is a special JavaScript undefined value.

Remember: JavaScript is case-sensitive so x and X are two different variable names.

Types of Variables

A big difference between JavaScript and other languages like JAVA and C is that JavaScript is ***untyped***. This means that a JavaScript variable can hold a value of any data type, and its data type does not have to be set when declaring the variable. This allows you to change the data type of a variable during the execution of your program, for example:

var x = 10;

x = "10";

In this example the variable x is first assigned the integer value of 10, and then the string value of the word ten.

Using Operators

Operators are the things that act on variables. We have already seen an operator used in the previous example, where we were assigning values to our variables. The example used one of the most common operators, "=" or the assignment operator. Another operator would be the addition operator "+".

```
var x = 1, y = 3, sum = 0;

sum = x + y;
```

This small chunk of JavaScript code will declare the variables x, y and sum and assign the number 1 to x, 3 to y and 0 to sum. The second line of the script will add x to y and assign it to sum. The value of the sum variable will be 4.

Other operators are used to compare things, i.e. "==" equality, ">" greater than. For example,

```
var x = 1, y = 3, sum = 0;

if ( sum == 0 ) {

sum = x + y;

}
```

This bit of code first checks to see if sum is equal to zero, and if so then it adds x and y together and assigns the result to sum. The "if" statement is an example of a control structure which we will examine shortly.

JavaScript Operators

Computational

These are probably the most common operators. They are used for common mathematical operations.

Unary negation (-)
Increment (++) --→ increment/increase by 1
Decrement (--) --→ decrement/decrease by 1
Multiplication (*)
Division (/)
Modulo arithmetic (%) ----→ remainder
Addition (+)
Subtraction (-)

Logical

These operators are very commonly used in conditional statements like "if" and "switch" that you will be learning about shortly.

Logical NOT (!)
Less than (<)
Greater than (>)
Less than or equal to (<=)
Greater than or equal to (>=)
Equality (==) ---→ equal to
Inequality (!=) ---→ not equal to
Logical AND (&&)
Logical OR (||)
Comma (,)

Assignment

It is important to note that the single equal sign is used for assignment and not for testing equality. The compound assignment operators can combine a couple of programming steps to make your code tighter, and more efficient.

Assignment (=)

Compound assignment operators

Addition (+=)   x = x+y
Subtraction (-=)
Multiplication (*=)
Division (/=)
Modulo Arithmetic (%=)

x += y	same as x = x+y
x -= y	same as x = x - y
x *= y	same as x = x*y
x /= y	same as x = x/y
x %= y	same as x = x%y

Control Structures (Loops and Branches)

Branches

if

The "if" statement is a fundamental control statement. It allows your program to perform a test, and act based on the results of that test. For example:

```
if ( (x == 1) && (y == 3) )
   {
    sum = y + x;
   }
```

In this statement the value of the x variable is compared to 1 to see if it is equal, and the value of the y variable is compared with 3 to see if it is equal. The use of the "&&", which is the "and" operator, adds an additional logical condition that says that the first comparison must be true **and** the second comparison must be true for the overall result of the test to be true. If the test results in an overall true condition then the statements that follow the if statement will be executed. If the test results are false nothing will occur.

An additional clause you can add to the "if" statement is the "else", an example:

```
if (sum == 0)
        {
        sum = x + y;
        }
else if (......)
        {
        .......
        }
 else
        {
        subtotal = sum;
        }
```

This statement is read as: if sum equals 0 then sum equals x plus y, or else subtotal equals sum.

Loops

A loop is a programming structure that forces the statements contained within its delimiters to execute over and over again until a condition is met at which point the loop ends.

while

While a condition is true, execute one or more statements. "While loops" are especially useful when you do not know how many times you have to loop, but you know you should stop when you meet the condition.

```
var x = 1;
while ( x <= 10 ) { // loop until x is greater than 10  until x is greaterthan 10

        x++; // add one to the value of x

     }
```

for

"For loops" are useful when you know exactly how many times you want the loop to execute.

```
        var x;
        for ( x = 1; x <= 10; x+=2 ) { // loop while x is <= 10}
        for ( x = 1; x <= 10; x=x+2 ) { // loop while x is <= 10}
```

// for (the initialize value of variable ; condition check ; increment/decrement by #)

```
// for (          ;  condition  ;   )

                do something ten times

        }
```

Functions

Functions are an important part of programming as they allow you to create chunks of code that **perform a specific task**. Functions in JavaScript are called subroutines or procedures in other programming languages. JavaScript has a number of built-in functions that are part of the language. JavaScript also gives you the ability to create your own functions. Your JavaScript programs may be made up of one function or several functions.

Built-in:

The built in functions are there to perform a number of actions that programmers expect to be part of a programming language. Some of the built in functions include: parseFloat(*string value*), parseInt(*string value*), isNaN(*value*), etc.. We will use these functions later in the course.

Programmer Created:

Functions that you create start with the command "function" and are followed by the name for the function. For example:

```
function function_name( argument1, argument2, … ) {
        .
        .
        JavaScript statements go here
        .
        .
        } // end of function
```

Usually the name is an indicator of the action the function is going to provide such as "checkForm". The name is followed by a set of parenthesis and inside the parenthesis there can be a number of arguments. Arguments are the variable names that will be used for values (data) being passed to the function.

All functions have the ability to pass back a value through the return statement. Another way of saying this is you can pass data into a function using its arguments, and get the results out with a returned value. Functions can only return **one** value. In the example below the sum of two numbers is returned by the add_two_num() function.

```
var globalVar = 1999;

function add_two_num( a, b) {
        var sum = a + b;
        return sum;
        } // end of function add_two_num

function mainProgram() {
        var x = 5, y = 3, total = 0;
        total = add_two_num( x , y );
        alert(total); // display the value of total
}
```

In the above example:

1. The function mainProgram declares variables and assigns their initial values as follows - "x" equal to 5, "y" equal to 3 and "total" equal to 0
2. Then it calls the add_two_num function and passes in the values of x and y.
3. In the add_two_num function the values are added together and stored in a variable named sum.
4. The value of sum is returned back to the mainProgram function and stored in the variable named total.
5. The value of total is then displayed to user in an alert box.

Variables declared in a function are only available while within that function's braces { }. These are commonly referred to as **local variables**.

Another type of variable is a global variable. Global variables are available to all functions, and should be used with caution, as it is easy to assign the wrong value to a global variable. In the above example globalVar is a **global variable** because it **is declared outside of all of the functions**.