Chapter 0

Introduction to JavaScript

By Goinza

This guide covers the basics on how to use the programing language known as JavaScript. This language has several applications, but we are going to focus only in the things that are applicable to SRPG Studio plugins.

To use this guide, it is recommended that you have a text editor program, in order to be able to create code. While the basic Notepad can be used, it is not recommended. Instead, you should use more advanced programs like Notepad++ or Visual Studio Code.

Core Elements

Here we will talk about the main components used in a JavaScript program.

Variables

A variable is a space in the computer's memory where you can store information. Each variable has a name, and it can be modified and used along with other variables.

```
var x = 10; // This is a variable
```

Let's analyze each part of this line: every new variable starts with the word "var". After that, you write the name of the variable (in this case, it is called "x"). Finally, you write the "=" symbol and the value of the variable, ending with the ";" symbol. The "=" symbol means that the value is assigned to the variable, while the ";" means that the line of code has ended, it is used to separate the different instructions. Finally, an optional thing is using comments. These can be used in any place that starts with the characters "//". Comments don't affect the code, they are there to make the code easier to understand.
There is an alternative way of making variables. Instead of assigning a value while creating it, you can first create it without any value. After that, you can assign any value to the variable later. For example:

```
var x;
x = 10;
```

You can also declare more than one variable in the same line.

```
var x, y, z;
```

Data Types

Each value of a variable belongs to a specific data type. There are 6 data types: strings, numbers, booleans, functions, objects and undefined. Below are small descriptions of each one, along with examples.

- Strings: values that represent words or characters. This are written using the " or ' characters. For example: "word", 'string', "JavaScript". A string can be concatenated with another string using the '+' operator.
- Numbers: numerical values, either integer or decimal, like 4, -2 or 14.3. You can do numerical operations using variables holding number data types. The operators used are:
  - + for addition.
  - - for subtraction.
  - * for multiplication.
  - / for division.
  - % for modulus (division remainder).
  - ** for exponentiation
  - ++ for increment
  - -- for decrement.
- Booleans: true or false values. You can use the AND operator (&&) along with the OR operator (||) to make operations between one of more booleans. You can also use the ! operator to change the value of a boolean.
- Functions and Objects: these two are more complex than the others, so they will be explained in detail in other sections of this guide.
- Undefined: variables that haven't been defined with a value.

```
//Declarations of variables
var s1, s2, s3, n1, n2, n3, b1, b2, b3;
//String examples
s1 = "Hello World!";
s2 = 'This is a string';
s3 = s1 + " - " + s2; //This means: "Hello World! - This is a string"
//Number examples
n1 = 2;
n2 = -5.25;
n3 = n1 * 5; //Result is 10
n1++ //Increments by one the value. Now n1 is 3
//Boolean examples
b1 = true;
b2 = b1 && false;
b3 = !b1;
//Undefined
var u; //This is undefined because no value has been assigned to it
```

Functions

The data types mentioned before are not enough to make code. To do this, you need to create functions that, given some parameter, do something and can return a new value. This is what you are going to use and modify in order to create plugins.

For example, a function called add that has two numbers as parameter can do the add operation and return the sum of those numbers.

```
var add = function(x1, x2) {
    var result = x1 + x2;
    return result;
}

```

Now this function can be called from other functions. This is useful for making modular code. You always have to try to make your functions in a way that don't do much. For example, let's make new functions that adds three numbers.

```
var addThree = function(x1, x2, x3) {
    var aux = x1 + x2;
    var final = aux + x3;
    return final;
}

```

This is a simple example, so it may not seem like a problem. But what would happen if, instead of having to just add numbers, you would have to do a more complex task? You would have to do the same thing twice in order to complete this function. To avoid this, we use the add function that we made before.

```
var addThree = function(x1, x2, x3) {
    var aux = add(x1, x2);
    var final = add(aux, x3);
    return final;
}

```

Here, instead of having to do the add operations, we call the add function in order to make it easier to read and write.

<u>IMPORTANT NOTE</u>: This is just a very simple example to illustrate the importance of functions. While it may seem useless to divide this task into different functions, it is important to solve more complex tasks. Also, in this particular example, the easiest solution is to add three numbers at the same time, so in a real application you wouldn't use functions to solve this problem.

```
var result = x1 + x2 + x3;
```

Objects

An object is a container for all types of data: it can contain basic types, like numbers, strings, booleans, and it can contain functions. You can also add objects inside to an object.

Objects are used to represent more complex objects than other normal types. You use basic data types to store information, and use functions to add behavior to that object. For example, we can create the concept of a “person”, which is defined by their first name and last name. In that case, we need to store two different strings for each one. Then, we want to add a function that returns both names together to form the full name of that person.

```
var Person = {
    firstName: "John",
    lastName: "Doe",

    fullName: function() {
        return this.firstName + " " + this.lastName; // "John Doe"
    }
}
```

As with any other variable, an object starts with the “var” word and the object’s name. Inside the object, we don’t use the “var” word to create new variables. Note that for the variables inside an object, we use different characters. Instead of using ‘=’, we use ‘:’, and instead of ‘;’, we use ‘,’. Also, the ‘,’ character is not used to finalize an instruction. You use it to *separate* instructions. That means that the last element doesn’t have that character. This is important, because an extra ‘,’ can cause an error in the script.

If you look at the function in the example above, you will notice that to call a value from the object, we use “this”. The word “this” always means the object which we currently are inside. You always need to use in order to use the values of that object. If you want to use the values while you are not in the object, you need to use the object’s name instead of the word “this”. For example, use Person.firstName instead of this.firstName.

Note that the dot (.) is used to go to access a property of the object, and it is a separation of the names of the object and property.

In this example, we already gave the person a name, but you can use it in a different way, changing the values dynamically at any moment.

```
//Object representing a person
var Person = {
    firstName: "", //Emtpy string
    lastName: "", //Empty string

    fullName: function() {
        return this.firstName + " " + this.lastName;
    },

    setFirstName: function(firstName) {
        this.firstName = firstName;
    },

    setLastName: function(lastName) {
        this.lastName = lastName;
    }
}

//Given a first name and a last name, return a string of the full name.
var getFullName = function(firstName, lastName) {
    Person.firstName = firstName;
    Person.setLastName(lastName);
    return Person.fullName();
}
```

In this example, the function getFullName modified the Person object, changing its values in two different ways. First, it access the property directly and replaces the value with a new one. Note that both the function's parameter and the object's property share the same name, but it is not a problem because they are both at different "levels". One is inside the function, while the other is inside an object that is used by the function. The other way it modifies a value is by using a function of the object. In this case, it uses the function setLastName to change the value of lastName.
While both methods are valid and do the same, it is recommended to use functions instead of changing a property directly. That will make it easier in the long term to handle scripts that have objects with lot of code. Finally, once both values are changed, it uses the fullName function to get the full name of that person.

Arrays

An array is a special type of object. It can hold more than one value at the same type. Each value will have its own index assigned, so with only one variable you can access all of them easily.

```
var fib = [1, 1, 2, 3, 5, 8, 13, 21];
var n1 = fib[2]; //2
var n2 = fib[0]; //1
var n3 = fib.length; //8
```

As you can see in this example, an array can be created as any other variable. The main difference is that it holds a set of elements, each of them separated by a comma (,) and the array itself is surrounded by brackets “[ ]”.
Once the array is created, you can access any value by the order they were added. For example, fib[0] is the first element of the array, fib[1] is the second, and so on.
The “length” property is something unique to arrays. The value is updated automatically and it always return the amount of elements the array has.
Given an array, there are ways to add and remove elements from it. You can access a specific value of the array using the index, which allows you to add, remove and change the elements on the array. There are also functions specific to the array object. In the context of creating plugins, you won’t need to delve deep into this, so we will only see the simplest examples on how to add elements to an array.

```
var f = function() {
    var arr = []; //Emtpy array,
    arr[0] = 10; //New element at index 0
    arr.push(4); //New element at end of the array - Index 1
    arr[2] = arr[0] + arr[1] ; //New element at index 2


    return arr;
}
```

Conditionals and Loops

Now that we explained the main elements present in the code, let’s talk about some tools that are a bit more complex, but give you a lot more flexibility and power to create code.
Until now, the code we written inside functions is completely linear. Once a instructions is executed, it goes to the next line and executes that instruction. But there are ways to change that: a conditional block executes one block of code if a specific condition is met; a loop block executes the same code several times.

Boolean expressions

Before we deal with conditionals and loops, it is important to explain in further detail the capacity of a boolean value. While it can only hold true or false as results, it can be used as an expression, instead of a simple value. Think of how

number variables work: you can assign a number to a variable, or you can assign an expression like 2 + 2, and it will store the value 4 in that variable. You can do the same with boolean variables.
For boolean expressions, there are several operators that can be used. Here are the more commonly used:

- “Equal” operator (==): a boolean expression is true if both sides of the equal operator represent the same value. For example, (2 + 2 == 4). This works for several types of data, like numbers, strings, objects, booleans, etc.
- “Greater” (>) and “greater or equal” (>=): used by numbers. The expression is true if the first number is greater (or equal) to the second. For example (6 > 2), (5 >= 5).
- “Lesser” (<) and “lesser or equal” (<=): used by numbers. The expression is true if the first number is lesser (or equal) to the second. For example (-1 > 2), (3 <= 3).

```
var n1 = 10;
var s1 = "word";
var b1 = n1 >= 5; //true
var b2 = "example" == s1; //false
var b3 = (n1 + 5 == 15); //true
```

Conditional

“If – then – else” block

This conditional block is divided into two parts. One part is executed if a condition is met, while the other executes if the condition is not met. The condition is a boolean expression. For example:

```
var conditionalExample = function(number) {
    var s;
    if (number >= 0) {
        s = "Number is postive or zero"
    }
    else {
        s = "Number is negative";
    }

    return s;
}
```

There is another way of using the “if” block. You can use it without the “else” part, so if the condition is not met, the block doesn’t do anything.

```
//Given an number, it returns its absolute value
//Absolute value: non-negative value of the number without regard to its
sign
var absoluteNumber = function(number) {
    if (number < 0) {
        //Number is negative, multiply it for -1 to make it positive
        number = number * -1;
    }
    return number;
}
```

Loops

Loop blocks repeat some code until a condition is met. There are two types of loop blocks.

“For” block

A “for” loop is meant to be used to repeat something a set amount of times. Generally, the amount of times the code is repeated is a fixed number. For example, given an array, the “for” block will be repeated as many times as the amount of elements on that array. For example, let’s look at this function that, given an array of numbers, returns the sum of all the numbers of the array.

```
var sumArray = function(arr) { //arr is an array of numbers
    var total = 0; //Total to return
    var i; //Index

    for (i = 0; i < arr.length; i++) {
        total += arr[i];
    }

    return total;
}
```

Let’s analyze the components of the first “for” line. There are three parts: the first element is the index used to determine the number of iterations. This part determines the initial value of the index, which usually starts with zero. Then, the second part is the condition that must be met before executing the “for” code. This condition is checked each time the code is repeated. Finally, the third part changes the value of the index every time an iteration ends. Most of the time, you want to increase the index by 1, so you use the increment operator.
In this particular example, we have a variable called “i” that starts with the value 0. The condition that must be met is that the variable is less that the amount of elements on the array, and finally, each time an iteration ends, the index is increased by 1.

Note how the “total” variable is used. Before the loop starts, it is initialized with the value zero, but during the for loop, it uses the operator “+=”. This is equivalent to say `total = total + arr[i]`
This can be applied to other operators like multiplication (*=), subtraction (-=), etc. Finally, it is important to clarify that, because the index of arrays starts with 0, the last index is just below the amount of elements on that array. That mean that array with a length of 5 has an index range from 0 to 4. That is why the condition specifies “index lesser than length” instead of using “lesser or equal”.

<u>“While” block</u>
The “while” block is similar to the “for” loop, but it is meant to be used when the amount of repetitions is unknown. Generally, this is used when you want to find a specific element on an array, so once you found it, you don’t need to keep repeating the loops. For example, a function that finds a specific number on an array and returns the index assigned to that number.

```
var findNumber = function(num, arr) {
    var index = 0;
    var found = false;

    while (index < arr.length && !found) {
        if (arr[index] == num) {
            found = true;
        }
        else {
            index++;
        }
    }

    if (!found) {
        index = -1; //It didn't found the number
    }

    return index;
}
```

“While” blocks are a bit easier to write than “for” blocks. You only need to specify the boolean expression which will act as the condition that must be met before executing the code. Note that we added two conditions, joined by the AND operator (&&). This means that both conditions must be true. The first condition is the same one used in the “for” loop, the index must be lesser than the length of the index. The second condition is the negation of a variable that starts at false, so the condition starts with true. This boolean is set to true if the number that we are looking for is found.

Note that, after the “while” block, we added a last conditional, which changes the index to -1 if the number has not been found. This lets the use know easily that it has not been found, because a negative number can never be an index of an array.

Final Notes

It is important to clarify that this is a very basic guide to the JavaScript language. Some of the more complex or less used tools are not mentioned here.
The goal of this guide is to give an introduction to the main things used in the creation of plugins for SRPG Studio. If you want to delve deeper into some of the things that this language can do, I recommend that you look for more comprehensive tutorials and documentations.
Hopefully, this guide helped you understand the basics of programming. Now, if you want to learn more about how to make plugins for the engine, you may want to check my other guides, which explain things specific to SRPG Studio.