**JavaScript Basics #1: Syntax and Program Structure**

Table of Contents

* [Introduction](https://www.ericsdevblog.com/index.php/2021/08/javascript-basics-1-syntax-and-program-structure/#Introduction)
  + [Setup the Environment](https://www.ericsdevblog.com/index.php/2021/08/javascript-basics-1-syntax-and-program-structure/#Setup_the_Environment)
* [Data Types](https://www.ericsdevblog.com/index.php/2021/08/javascript-basics-1-syntax-and-program-structure/#Data_Types)
  + [Numbers](https://www.ericsdevblog.com/index.php/2021/08/javascript-basics-1-syntax-and-program-structure/#Numbers)
  + [Strings](https://www.ericsdevblog.com/index.php/2021/08/javascript-basics-1-syntax-and-program-structure/#Strings)
  + [Boolean Values](https://www.ericsdevblog.com/index.php/2021/08/javascript-basics-1-syntax-and-program-structure/#Boolean_Values)
  + [Empty Values](https://www.ericsdevblog.com/index.php/2021/08/javascript-basics-1-syntax-and-program-structure/#Empty_Values)
  + [Data Type Conversion](https://www.ericsdevblog.com/index.php/2021/08/javascript-basics-1-syntax-and-program-structure/#Data_Type_Conversion)
* [Program Structure](https://www.ericsdevblog.com/index.php/2021/08/javascript-basics-1-syntax-and-program-structure/#Program_Structure)
  + [Statements and Bindings](https://www.ericsdevblog.com/index.php/2021/08/javascript-basics-1-syntax-and-program-structure/#Statements_and_Bindings)
  + [Functions](https://www.ericsdevblog.com/index.php/2021/08/javascript-basics-1-syntax-and-program-structure/#Functions)
  + [if Statements](https://www.ericsdevblog.com/index.php/2021/08/javascript-basics-1-syntax-and-program-structure/#if_Statements)
  + [for Loops](https://www.ericsdevblog.com/index.php/2021/08/javascript-basics-1-syntax-and-program-structure/#for_Loops)
  + [while Loops](https://www.ericsdevblog.com/index.php/2021/08/javascript-basics-1-syntax-and-program-structure/#while_Loops)
  + [do while Loop](https://www.ericsdevblog.com/index.php/2021/08/javascript-basics-1-syntax-and-program-structure/#do_while_Loop)
  + [Breaking Out of a Loop](https://www.ericsdevblog.com/index.php/2021/08/javascript-basics-1-syntax-and-program-structure/#Breaking_Out_of_a_Loop)

**INTRODUCTION**

JavaScript is the first programming language we are going to introduce in this web development course. It was created in 1995 as a way to add programs to web pages in the Netscape Navigator browser. Today the language has been adopted by all the other major web browsers.

**Setup the Environment**

In this tutorial, we are going to have a lot of example code snippets. To execute these code, we can simply open a browser, go to Developer Tools -> Console:

Graphical user interface, text, application, email

Description automatically generated

Or you can install [Node.js](https://nodejs.org/en/) on your computer, which allows you to run JavaScript programs using command terminals.

Text

Description automatically generated

**DATA TYPES**

In the computer world, it’s all about data. What a computer program does is essentially taking some input data, process them, and eventually return some output data. In this section, let’s talk about some different types of data that JacaScript can process.

**Numbers**

Numbers are the easiest because it works exactly like what you studied in your math class in elementary school.

// Integer

100

// Fractional Number

10.56

//Scientific Notation

3.14e5

// 3.14 \* 10^5 = 314000

The primary usage of numbers is to do arithmetic operations.

3 + 5 \* 2

// -> 13

Just like you studied in elementary school, multiplications and divisions happen first. However, you can change this by using parentheses.

(3 + 5) \* 2

// -> 16

There is one operator that you might not recognize, it is the modulo (%) operation. X % Y calculates the remainder of dividing X by Y. For example:

25 % 5

// -> 0

25 % 10

// -> 5

25 % 15

// -> 10

**Strings**

Strings are used to represent texts, and they are all enclosed in quotes like this:

"This is a string."

'This is a string.'

Both single and double quotes work exactly the same, as long as the quotes at the start and the end of the string match.

Whenever a backslash (\) is found inside a string, it means the character after it has a special meaning. For example, when the character n follows the backlash (\n), this will be interpreted by your computer as a new line:

"This is the first line\nThis is the second line"

The actual text would look like this:

This is the first line

This is the second line

The + operation can also be used on strings. But obviously, strings can not be used in arithmetic operations, the plus sign here means concatenate (connecting two strings together).

"con" + "cat" + "e" + "nate"

// -> "concatenate"

Finally, there is a special kind of string in JS(JavaScript), the backtick-quoted strings, usually called template literals. It allows us to embed other values inside the string:

`half of 100 is ${100 / 2}`

In this example, the division inside ${} will be calculated, the result will be converted into a string and printed in that position. So this example will give us:

half of 100 is 50

**Boolean Values**

The boolean type only includes two values, true and false. Comparison is the most common way to produce boolean values.

console.log(1 == 1)

// -> true

console.log(1 > 2)

// -> false

console.log(1 < 0)

// -> false

console.log(1 != 2)

// -> true

In this example, == means equal, and != means not equal. Other similar operators include >= (greater than or equal to) and <= (less than or equal to).

There are three logical operators that we can apply to boolean values in JS, && (and), || (or), and ! (not).

The && operator denotes logical and, it produces true only if both values given to it are true.

console.log(true && false)

// -> false

console.log(false && true)

// -> false

console.log(false && false)

// -> false

console.log(true && true)

// -> true

The || operator denotes logical or, it produces true if either of the values given to it is true.

console.log(true || false)

// -> true

console.log(false || true)

// -> true

console.log(false || false)

// -> false

console.log(true || true)

// -> true

The ! operator denotes logical not. It flips the value given to it.

console.log(!true)

// -> false

console.log(!false)

// -> true

We can also mix arithmetic operations with comparisons and logical operations.

1 + 1 == 2 && 1 + 1 < 0

In this example, 1 + 1 == 2 gives us true, and 1 + 1 < 0 gives us false, so we have

true && false

// -> false

**Empty Values**

There are two special values in JS, null and undefined. They indicate the absence of a meaningful value. In computer programs, there are a lot of operations that do not produce meaningful results (which we will see later in this course), and these results will be denoted by null or undefined.

These two values have virtually no difference, in fact, in most cases, you can treat them as interchangeable. The fact that there are two different values indicating the same thing is just an accident of JS’s design.

**Data Type Conversion**

JS is a very intelligent programming language, it will always try to execute the program you give it, even though the program does not make sense. For example:

console.log(8 \* null)

// -> 0

console.log("5" - 1)

// -> 4

console.log("5" + 1)

// -> "51"

In the first example, the null gets converted into number 0, and in the second example, the string "5" becomes a number 5. However, in the third example, the number 1 gets converted into a string "1", and the plus sign here means concatenate, so the result becomes "51".

This is why you should never try to do this when you are coding, even though it “works”, it still might lead to unexpected results.

**PROGRAM STRUCTURE**

**Statements and Bindings**

In computer programming, you can think of a “program” as an instruction manual to solve a complex problem. Each instruction/sentence in that manual is called a statement. In JavaScript, a statement should always end with a semicolon(;).

let num = 10;

This example is called a binding, or variable. It binds the value 10 to the name num using the = operator, which allows us to do something like this:

let num = 10;

console.log(num \* num);

// -> 100

The keyword let indicates that this statement is going to define a binding.

When a binding is formed, it does not mean that the name is tied to the value forever, we can still use the = operator on existing bindings.

let num = 10;

console.log(num);

// -> 10

num = 20;

console.log(num);

// -> 20

Notice that we only used the keyword let in line 1. That is because let is used to define/declare a binding, and in line 5, we are merely updating the value that is tied to the name num.

let num1 = 10;

let num2 = 20;

console.log(num1); // -> 10

console.log(num2); // -> 20

num2 = num1;

console.log(num1); // -> 10

console.log(num2); // -> 10

let num = 10;

num = num - 5;

console.log(num); // -> 5

The keywords const and var can also be used to create bindings just like let, however, they are different in terms of scopes, which we will talk about in the next article.

**Functions**

A function is a piece of program that returns a value or has some side effects, or both. For example, the console.log() function we have seen a few times is used to output values in the terminal.

Graphical user interface, text, application, email

Description automatically generated

Or, in this example, the prompt() function will show you a dialog that asks for user input, and that input will be bound with the name num.

let num = prompt("Enter A Number");

console.log(num);

Graphical user interface, application

Description automatically generated

Both showing a dialog and writing text to screen are side effects. A function can also be useful without the side effect. For example:

console.log(Math.max(2,4,6,8));

The Math.max() function does not have any side effects, it simply takes a set of numbers and returns the greatest.

All of these functions are built into our browser. We can, however, create our own functions using JavaScript. We will discuss this topic in the next article.

**if Statements**

The if statements offers us a way to excute different pieces of code under different conditions. For example:

let num = prompt("Enter A Number");

if (num < 10) {

console.log("Small");

} else {

console.log("Large");

}

This program asks you input a number, if the number is less than 10, console.log("Small"); will be excuted, the program will output "Small". If the number is larger than 10, the program will output "Large".

We can also chain multiple if/else pairs if there are multiple conditions we need to include:

if (num < 10) {

console.log("Small");

} else if (num < 100) {

console.log("Medium");

} else {

console.log("Large");

}

This program will first check if the number is less than 10, if it is, it will output "Small".

If the number is greater than 10, the program will then check if it is less than 100. If it is, the program will output "Medium".

Finally, if the number is greater than 100, the program will show "Large".

**for Loops**

The for loops offer us a way to execute the same code over and over again, as long as some conditions are satisfied.

for (let num = 0; num <= 12; num = num + 2){

console.log(num);

}

// ->

A for loop takes three expressions, separated by two semicolons. In this example, the first expression let num = 0 declares a new variable num, whose initial value is 0. The second expression means the loop will iterate until the condition num <= 12 is violated (num is larger than 12). The last expression means for each iteration, num will add itself by 2.

Graphical user interface, text, application, email

Description automatically generated

**while Loops**

while loops work in a similar way, except it only takes one expression. In fact, we can easily change our previous for loop example into a while loop.

let num = 0;

while (num <= 12) {

console.log(num);

num = num + 2;

}

In this example, we initiated the num variable first, outside of the while loop. Inside the parentheses after the keyword while is the expression that checks whether the loop should continue. Finally, we update the value of num at the end of the while loop.

**do while Loop**

A do-while loop differs from a while loop only on one point, it guarantees that the body of the loop executes at least once.

let num = 10;

do {

num = num + 1;

console.log(num);

} while (num <= 1);

This time the initial value of num is 10, which violates the condition for the loop to continue. But because this is a do-while loop, the body is still executed once. If this was a while loop, it would not execute at all.

Graphical user interface, text, application

Description automatically generated

**Breaking Out of a Loop**

Violating the condition for the loop to continue is not the only way we can stop a loop. For instance, you are asked to find a number that is greater than 100, and divisible by 9 (Recall that % operator is used to calculate reminder, so if the remainder of x/9 equals 0, that means x is divisible by 9.). We can use a for loop to solve this problem:

for (let num = 100; ; num = num + 1) {

if (num % 9 == 0) {

console.log(num);

break;

}

}

Notice that we do not have an expression that decides whether the loop should continue. Instead, we have an if statement with a break keyword inside, which will break out of the loop if it is excuted.

If you remove the break keyword, this for loop becomes an infinite loop and will run forever, which is something you should always avoid.

**Share this:**