# **Javascript Basics**

## **Variables and Operators**

### **Intro:**

Javascript: makes the webpage *interactive*

console.log(): print something to the developer console in your browser.

Include JS in a webpage by:

1. <script> tags at the bottom of the webpage
2. As an external script: <script src="javascript.js"></script>

JavaScript files have the extension .js similar to .css for stylesheets. External JavaScript files are used for more complex scripts.

### **Variables:**

The building blocks of any program. Think of Variables as “storage” for data in your code.

You can declare variables using the let keyword.

    <script>

        let name = "Tayyab";

        let surname = "Hussein";

        console.log(name);

        console.log(surname);

    </script>

console.log() outputs the values of your code. It is good for checking your code is correct and debugging errors.

You can also re-assign variables:

    <script>

        let age = 27;

        console.log(age); //outputs 27 to console

        age = 57;

        console.log(age); //outputs 57 to console

    </script>

Notice the lack of let when called again? - we don’t need it since the variable has already been *declared* earlier and we are just re-assigning it here!

Re-assigning is cool and all, but what if we *don’t* want it to happen? For example we might have a *constant* value which will never change. We can accomplish this using the const keyword.

    <script>

        const height = 6;

        height = 7;

        console.log(height); // error in console

    </script>

Attemping to re-assign const throws an error in the console

Summary, there are two ways to declare a variable:

* let, which we can re-assign.
* const which we **can’t** re-assign and will throw an error if we try.

There is also a third way, var, which was the original way variables were declared in JavaScript. But is now largely obsolete but still good to know as you may come across it.

## [**Numbers**](https://www.theodinproject.com/lessons/foundations-variables-and-operators#numbers)

Numbers inputs also work in Javascript. In fact, they are the building blocks of any programming language, so it helps to know basic maths.

Play with it in console:

    <script>

        console.log(4 + 2); // 6

        console.log(4 + 2 + 1 + 3 + 5 + 7); // 22

        console.log((4 + 6 + 9) / 77); // 0.24675324675324675

    </script>

Now let’s declare a variable too:

    <script>

       let a = 10;

       console.log(a); // 10

       console.log(9 \* a); // 90

       let b = 7 \* a;

       console.log(b); // 70

    </script>

    <script>

        const max = 57;

        const actual = max - 13;

        const percentage = actual / max;

        console.log(percentage); // 0.7719298245614035

    </script>

Note: numbers can be strings too. E.g. let a = “9”;

### **Variable Naming:**

1. The name must contain only letters, digits, or the symbols $ and \_.
2. The first character must be a letter and not a digit.
3. When the name contains multiple words, [camelCase](https://en.wikipedia.org/wiki/CamelCase) is commonly used.
4. Case matters: apple and APPLE are two different variables.
5. Use caps and underscores for difficult to remember const values like color codes and also for values that are hard coded and don’t change during code execution, like someone’s birthday. E.g. const COLOR\_RED = "#F00"; const BIRTHDAY = '18.04.1982'; for more, see end example.
6. There is a list of reserved words, which cannot be used as variable names because they are used by the language itself. For example: let, class, return, and function are reserved. E.g. let let = 5; // can't name a variable "let", error!
7. It is possible to use any language, including Cyrillic letters, Chinese logograms and so on, but not recommended. E.g. let имя = '...';
8. A variable name should have a clean, obvious meaning, describing the data that it stores. E.g userName or shoppingCart
9. Declaring more variables will not negatively impact performance of your app, as Modern JavaScript minifiers and browsers optimize code well enough, so it won’t create performance issues.
10. Normally, we need to define a variable before using it. But in the old times, it was technically possible to create a variable by a mere assignment of the value without using let. This still works now if we don’t put use strict in our scripts to maintain compatibility with old scripts (and throws an error if you do put strict, strict is generally what we want to use). But nowadays is considered bad practice. E.g. num = 5;

Examples of valid variable names:

let userName;

let test123;

let $ = 1; // declared a variable with the name "$"

let \_ = 2; // and now with the name "\_"

alert($ + \_); // 3

Examples of incorrect variable names:

let 1a; // cannot start with a digit

let my-name; // hyphens '-' aren't allowed in the name

Quiz:

Would it be right to use upper case for const birthday? For age? Or even for both?

const BIRTHDAY = '18.04.1982'; // make birthday uppercase?

const AGE = someCode(BIRTHDAY); // make age uppercase?

We generally use upper case for constants that are “hard-coded”. Or, in other words, when the value is known prior to execution (and not during code execution) and directly written into the code.

In this code, birthday is exactly like that. So we could use the upper case for it.

In contrast, age is evaluated in run-time. Today we have one age, a year after we’ll have another one. It is constant in a sense that it does not change through the code execution. But it is a bit “less of a constant” than birthday: age is calculated, so we should keep the lower case for it.

Age depends on code calculation so it is lowercase.

A screenshot of a computer

Description automatically generated

Terminology:

|  |  |  |
| --- | --- | --- |
| **Operand** | **Operator** | **Operand** |
| 100 | + | 50 |

The **modulus** operator (%) returns the division **remainder**.

    <script>

        let a = 5;

        let b = 4;

        console.log(a % b); // 1

    </script>

Quotient: Result of normal division. E.g. 5 / 2 = 2.5 is the quotient

Remainder: is the remainder of a division

You can express very large/small numbers with **“scientific exponent”** different from exponent:

**Exponent**: Indicates repeated multiplication. E.g. n^

**Scientific Exponent**: written in javascript like, 123e5 which means m × 10^ and is for work with very large or small numbers. E.g. 123e5 123 x 10^5 = 12,300,000

JavaScript Numbers are Always 64-bit Floating Point. All you need to know for now is that it’s a way that computers store numbers.

In JavaScript, integers (whole numbers) are only accurate up to 15 digits. When we say integers are accurate up to **15 digits**, it means that if you use integers that have more than 15 digits, you risk losing precision. This is because after the 15th digit, the representation might not be precise due to rounding or loss of significant digits.

The maximum number of floating numbers (decimals) is **17**.

Floating point arithmetic is not always 100% accurate:

let a = 0.2 + 0.1;

console.log(a); // 0.30000000000000004

To resolve this, multiply and divide to it’s whole number equivalent:

let a = (0.2 \* 10 + 0.1 \* 10) / 10;

console.log(a); // 0.3

You can concatentate strings with numbers. This will convert both of them to strings

        let a = '10'

        let b = 20;

        console.log(a + b); // 1020

When two or more strings are numbers and arithmetic is used, Javascript will convert those strings to numbers.

        let a = "5";

        let b = "2";

        console.log(a - b); // 3

let a = "5";

let b = "2";

console.log(a + b); // 52

Here is more complex example:

alert(2 + 2 + '1' ); // "41" and not "221"

Here, operators work one after another. The first + sums two numbers, so it returns 4, then the next + adds the string 1 to it, so it’s like 4 + '1' = '41'.

+ is the only operator that concat strings. Other arithmetic operators work only with numbers and always convert their operands to numbers.

To convert a string number back to a number, you can use the Number() constructor.

let a = "7";

console.log(a = Number(a) + 3); // 10

NaN - Not a Number. This is an error (reserved word) javascript will throw up if you try to do maths with a non-number, like a word or symbol. Use the global JavaScript function **isNaN()** to find out if a value is a not a number:

let a = 5 / "Apple";

console.log(isNaN(a)); // true i.e. it's not a number

But Nan itself, as a keyword, is a number according to Javascript. You can verify this by using the typeof operator.

        let a = NaN;

        console.log(typeof a); // number

Infinity (or -Infinity) is the value JavaScript will return if you calculate a number that is greater than any finite number.

        let a = 2;

        while (a != Infinity) {

            a = a \* a;

        }

        console.log(a); // infinity. While is a loop, != operator is used to compare two values are not equal to each other. If not equal, it will return true. So you are looping a = 2 \* 2 until infinity here, so JS returns infinity

Dividing by 0 also returns infinity.

Hexadecimals are preceded by 0x in JS

let x = 0xFF;

Never write a number with a leading zero (like 07). Some JavaScript versions interpret numbers as octal if they are written with a leading zero. So if you are creating a list with the seventh item simply write it as 7 and not 07.

Different types of number systems in maths:

* **Decimal** — base 10 (meaning it uses 0–9 in each digit), most common.
* **Binary** — The lowest level language of computers; 0s and 1s.
* **Octal** — Base 8, uses 0–7 in each digit.
* **Hexadecimal** — Base 16, uses 0–9 and then a–f in each digit. You may have encountered these numbers before when setting [colors in CSS](file:///C:\Users\moham\OneDrive\Desktop\Study\Self%20Study\TOP\Javascript.docx#hexadecimal_rgb_values).

Decimal numbers are the most common; you'll rarely come across a need to start thinking about other types, if ever.

**Note:** Actually, JavaScript has a second number type, [BigInt](https://developer.mozilla.org/en-US/docs/Glossary/BigInt), used for very, very large integers. But for the purposes of this course, we'll just worry about Number values.

Another way to work with numbers in JS is using the number object (not to be confused with number constructor mentioned earlier), although this is generally not recommended as it adds more overhead. E.g. instead of:

let a = 123;

You do:

let a = new Number (123);

This method is uncommon though and generally not recommended as it wraps the numbers (123) in an ‘object’. Resulting in more overhead than simply storing it as a number value. You will **rarely** ever use this.

Round your number to decimal places, with the [toFixed()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number/toFixed) method.

let a = 9.012345;

console.log(a.toFixed(2)); // 9.01

When you use toFixed() on a numeric value, it becomes a string. Check with typeof:

        let a = 9.012345;

        let rounded = a.toFixed(2);

        console.log(typeof rounded); // string

Let’s look at the increment/decrement operator:

let a = 4;

console.log(a++); // 4

The code you provided will increment a from 4 to 5, but it won’t immediately log 5 to the console. Here’s why:

In JavaScript, a++ is the post-increment operator. This means it returns the current value of a (which is 4) before incrementing it. So, when you run console.log(a++);, it will print 4, but then a will increment to 5 afterward.

If you want 5 to be logged immediately, you can use the pre-increment operator ++a instead, which increments a first and then returns the new value.

let a = 4;

console.log(++a); // Logs 5

Alternatively, Log a After Incrementing:

let a = 4;

a++;

console.log(a); // Logs 5

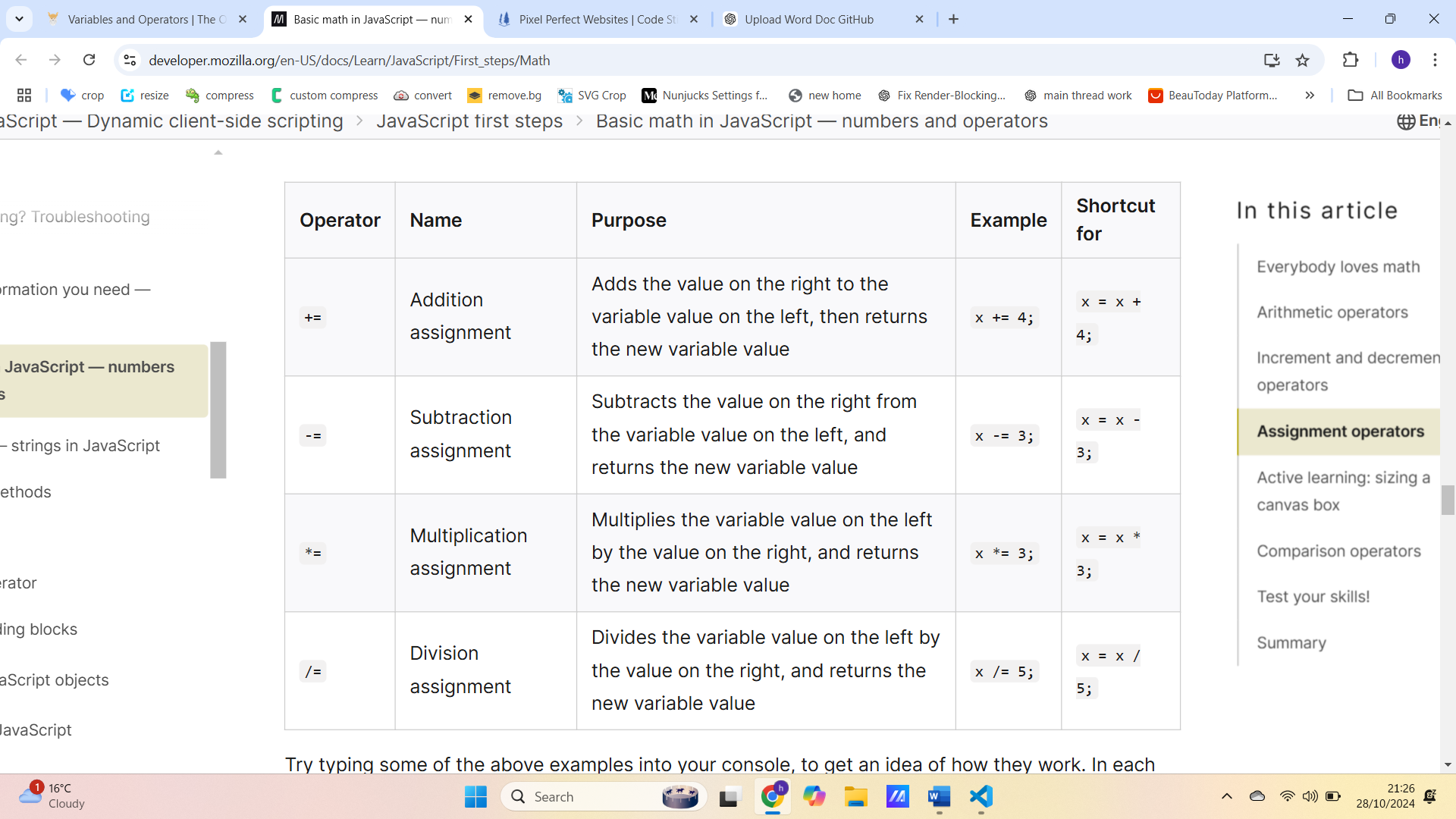
- a++ logs the value before incrementing (prints 4, then increments to 5).

- ++a increments first, then logs the result (immediately logs 5).

### Increment/decrement can only be applied to variables. Trying to use it on a value like 5++ will give an error.

### [**Assignment operators**](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Math#assignment_operators)

Assignment operators are operators that assign a value to a variable. We have already used the most basic one, =, loads of times — it assigns the variable on the left the value stated on the right:



      let name = "Tayyab";

      name += " Hussein";

      console.log(name); // Tayyab Hussein

      let value = 6;

      value += 4;

      console.log(value); // 10

-= , /= , and \*= do not work with strings and will result in NaN. Only += works in concating strings

      let car = "Honda";

      car -= "Toyota";

      console.log(car); //Nan

There are more assignment operators but these are the ones we need to know for now.

These assignment operators only work with variables and not values:

Correct:

let a = 43;

a += 7; // This works, updating a to 50

console.log(a); // Output: 50

Incorrect:

let a = 43 += 7; // Result: Error as 43 and 7 are values. Should be 43 + 7 instead.

console.log(a);

### **Comparison operators**

Sometimes we will want to run true/false tests

A screenshot of a computer

Description automatically generated

let a = 5 === 2 + 2;

console.log(a); // false

let a = 5 === 2 + 3;

console.log(a); // true

**Note:** You may see some people using == and != in their tests for equality and non-equality. These are valid operators in JavaScript, but they differ from ===/!==. The former versions test whether the values are the same but not whether the values' datatypes are the same. The latter, strict versions test the equality of both the values and their datatypes. The strict versions tend to result in fewer errors, so we recommend you use them.

True and false is called ‘Boolean Values’

### **Some Terms:**

**Unary and binary operators do not include the assignment operator**. Just arithmetic operators like + - \* / % \*\* and comparison and logical operators which you haven’t covered yet.

Unary operator: one with a single operand. E.g. -a below, below

Binary operator: with two operands. E.g. let binary = x – y;

        let a = 5;

        a = -a; // unary operator

        console.log(a); // -5

When + is applied to a single operand (unary), it has no affect on numbers. But when you add it to non-numbers (e.g. strings), it converts it to numbers. So it actually does the same thing as Number() constructor, but is shorter.

        let apples = "2";

        let oranges = "3";

        console.log(+apples + +oranges); // 5

console.log(Number(apples) + Number(oranges)); // 5, longer version

Here unary pluses are applied first, they convert strings to numbers, and then the binary plus sums them up. unary pluses applied to values before the binary ones because of their higher precedence

Here’s [precedence table](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Operator_Precedence) for JS with larger numbers indicating higher precedence (you don’t need to remember this, just know that unary operators are higher than binary ones):

A screenshot of a computer

Description automatically generated

Because the assignment operator has the lowest priority watch this:

        let a = 3;

        let b = 5;

        b \*= a + 8;

        console.log(b); // 55

55 is the value of b because you will do the operators on the right before the assignment operator as you do it last as it is lowest priority

To write clean code, we advise a style of “one line – one action”:

let counter = 1;

alert( 2 \* counter );

counter++;

The comma operator , is one of the rarest and most unusual operators. Sometimes, it’s used to write shorter code, so we need to know it in order to understand what’s going on.

The comma operator allows us to evaluate several expressions, dividing them with a comma ,. Each of them is evaluated but only the result of the last one is returned:

let a = (1 + 2, 3 + 4);

alert( a ); // 7

Here, the first expression 1 + 2 is evaluated and its result is thrown away. Then, 3 + 4 is evaluated and returned as the result.

Please note that the comma operator has very low precedence, lower than =, so parentheses are important in the example above.

Without them: a = 1 + 2, 3 + 4 evaluates + first, summing the numbers into a = 3, 7, then the assignment operator = assigns a = 3, and the rest is ignored. It’s like (a = 1 + 2), 3 + 4.