Intro to Javascript

What is JavaScript?

[JavaScript](https://developer.mozilla.org/en-US/docs/Glossary/JavaScript) ("JS" for short) is a full-fledged [dynamic programming language](https://developer.mozilla.org/en-US/docs/Glossary/Dynamic_programming_language) that, when applied to an [HTML](https://developer.mozilla.org/en-US/docs/Glossary/HTML) document, can provide dynamic interactivity on websites. It was invented by Brendan Eich, co-founder of the Mozilla Foundation.

JavaScript is incredibly versatile. You can start small, with carousels, image galleries, fluctuating layouts, and responses to button clicks. With more experience, you'll be able to create games, animated 2D and 3D graphics, comprehensive database-driven apps, and much more!

JavaScript contains a **standard library of objects**, such as **Array**, **Date**, and **Math**, and a core set of language elements such as operators, control structures, and statements. Core JavaScript can be extended for a variety of purposes by supplementing it with additional objects; for example:

* ***Client-side (in the browser) JavaScript*** extends the core language by supplying objects to control a browser and its Document Object Model (DOM). For example, client-side extensions allow an application to place elements on an HTML form and respond to user events such as mouse clicks, form input, and page navigation.
* ***Server-side JavaScript*** extends the core language by supplying objects relevant to running JavaScript on a server. For example, server-side extensions allow an application to communicate with a database, provide continuity of information from one invocation to another of the application, or perform file manipulations on a server.

What can in-browser JavaScript do?

The modern JavaScript is a “safe” programming language. It does not provide low-level access to memory or CPU, because it was initially created for browsers which do not require it.

In-browser JavaScript can do everything related to webpage manipulation, interaction with the user and the webserver.

For instance, in-browser JavaScript is able to:

* Add new HTML to the page, change the existing content, modify styles.
* React to user actions, run on mouse clicks, pointer movements, key presses.
* Send requests over the network to remote servers, download and upload files (so-called [AJAX](https://en.wikipedia.org/wiki/Ajax_(programming)) and [COMET](https://en.wikipedia.org/wiki/Comet_(programming)) technologies).
* Get and set cookies, ask questions to the visitor, show messages.
* Remember the data on the client-side (“local storage”).

What CAN’T in-browser JavaScript do?

JavaScript’s abilities in the browser are limited for the sake of the user’s safety. The aim is to prevent an evil webpage from accessing private information or harming the user’s data.

The examples of such restrictions are:

* JavaScript on a webpage may not read/write arbitrary files on the hard disk, copy them or execute programs. It has no direct access to OS system functions.

Modern browsers allow it to work with files, but the access is limited and only provided if the user does certain actions, like “dropping” a file into a browser window or selecting it via an <input> tag.

There are ways to interact with camera/microphone and other devices, but they require a user’s explicit permission. So a JavaScript-enabled page may not sneakily enable a web-camera, observe the surroundings and send the information to the [NSA](https://en.wikipedia.org/wiki/National_Security_Agency).

* Different tabs/windows generally do not know about each other. Sometimes they do, for example when one window uses JavaScript to open the other one. But even in this case, JavaScript from one page may not access the other if they come from different sites (from a different domain, protocol or port).

This is called the “Same Origin Policy”. To work around that, *both pages* must contain a special JavaScript code that handles data exchange.

The limitation is again for user’s safety. A page from http://anysite.com which a user has opened must not be able to access another browser tab with the URL http://gmail.com and steal information from there.

* JavaScript can easily communicate over the net to the server where the current page came from. But its ability to receive data from other sites/domains is crippled. Though possible, it requires explicit agreement (expressed in HTTP headers) from the remote side. Once again, that’s safety limitations.

## Our First JS Commands and Program

alert("hello world");  
document.write("hello world");  
console.log("hello world");

# JavaScript Syntax

A few examples of syntax:

// Two slashes start single-line comments   
let x; // declaring a variable   
x = 3 + y; // assigning a value to the variable `x`   
  
foo(x, y); // calling function `foo` with parameters `x` and `y`   
  
// A conditional statement   
if (x == 0) { // Is `x` equal to zero?   
 x = 123;   
}   
  
// Defining function `baz` with parameters `a` and `b`   
function baz(a, b) {   
 return a + b;   
}  
  
// Example of loop  
for (var i=0; i < arr.length; i++) {  
     console.log(arr[i]);  
 }

## Statements

Statements are syntax constructs and **commands that perform actions**.

We’ve already seen a statement alert('Hello, world!'), which shows the message.

We can have as many statements in the code as we want. Another statement can be separated with a semicolon.

For example, here we split the message into two:

alert('Hello'); alert('World');

Usually each statement is written on a separate line – thus the code becomes more readable:

alert('Hello');

alert('World');

## Blocks

A**block** is used to group zero or more statements. The block is delimited by a pair of curly brackets. We generally use blocks with if, switch, and for, and function

// A conditional statement   
if (x == 0) { // Block  
 x = 123; // A statement  
}   
  
// Defining function `baz` with parameters `a` and `b`   
function baz(a, b) { // Block  
 return a + b; // A statement  
}  
  
// Example of loop  
for (var i=0; i < arr.length; i++) { // Block  
     console.log(arr[i]); // A statement  
 }

## Functionality and Readability

Functionality and readability are two important reasons to focus on syntax as you begin to work with JavaScript.

There are some syntax rules that are mandatory for JavaScript functionality. If they are not followed, the console will throw an error and the script will cease execution.

Consider a syntax error in the "Hello, World!" program:

// Example of a broken JavaScript program

console.log("Hello, World!"

This code sample is missing the closing parenthesis and semicolon, and instead of printing the expected "Hello, World!" to the console, the following error will appear:

Output

Uncaught SyntaxError: missing ) after argument list

The missing ); must be added before the script will continue to run. This is an example of how a mistake in JavaScript syntax can break the script, as correct syntax must be followed in order for code to run.

## Whitespace

Whitespace in JavaScript consists of spaces, tabs, and newlines (pressing ENTER on the keyboard). Excessive whitespace outside of a string and the spaces between operators and other symbols are ignored by JavaScript. This means the following three examples of variable assignment will have the exact same computed output:

let userLocation = "New York City, " + "NY";

let userLocation="New York City, "+"NY";

let userLocation = "New York City, " + "NY";

userLocation will represent "New York City, NY" no matter which of these styles are written in the script, nor will it make a difference to JavaScript whether the whitespace is written with tabs or spaces.

A good rule of thumb to be able to follow the most common whitespace conventions is to follow the same rules as you are used to in math and language grammar.

For example, let x = 5 \* y is more readable than let x=5\*y.

One notable exception to this style you may see is during assignment of multiple variables. Note the position of = in the following example:

let companyName = "Boston College";

let companyHeadquarters = "Chestnut Hill";

let companyHandle = "@bc";

All the assignment operators (=) are lined up, with the whitespace after the variable. This type of organization structure is not used by every codebase, but can be used to improve readability.

## Indentation

A complete JavaScript program can technically be written on a single line. However, this would quickly become very difficult to read and maintain. Instead, we use newlines and indentation.

Here's an example of a conditional if/else statement, written on either one line or with newlines and indentation.

// Conditional statement written on one line

if (x === 1) { /\* execute code if true \*/ } else { /\* execute code if false \*/ }

// Conditional statement with indentation

if (x === 1) {

// execute code if true

} else {

// execute code if false

}

Notice that any code included within a block is indented. The indentation can be done with two spaces, four spaces, or by pressing the tab character.

Any nested block statement will be indented further.

// Initialize a function

function isEqualToOne(x) {

// Check if x is equal to one

if (x == 1) {

// on success, return true

return true;

} else {

return false;

}

}

# Variables

Most of the time, a JavaScript application needs to work with information. Information such as age, color, totals, description, credit card number, address, nationality, and many more. Variables are used to store this information.

[Variables](https://developer.mozilla.org/en-US/docs/Glossary/Variable) are containers that you can store values in. You start by declaring a variable with either the var  or let keyword, followed by any name you want to call it. We will be using the let keyword to declare our variables:

//declaring a variable with the let keyword  
let age;

After declaring a variable, you can give it a value:

//we can put some data into it by using the assignment operator "="  
age = 72;

Here is another example:

//declaring a variable  
let message;  
//assigning data to a variable  
message = "Your around the World Vacation purchase has been processed successfully. Bon Voyage!";

We can also create/declare the variable and assign it at the same time:

let message = 'Hello World!'; // declare the variable and assign the value

Now that we have declared and assigned a value to our variable we can use one of the commands we learned in the previous lesson:

alert(message);  
console.log(message);  
document.write(message);

We can also change the value of variables. This is their main purpose in our programs, to hold different values:

let main\_city;

main\_city = 'Boston';

main\_city = 'Paris'; // value changed

alert(main\_city);

And we can copy the value of one variable into another:

let main\_city = "Boston";  
let secondary\_city = "Paris";  
alert(main\_city);

main\_city = secondary\_city; // value changed

alert(main\_city);

## Variable naming

There are two limitations for a variable name in JavaScript:

1. The name must contain only letters, digits, symbols $ and \_.
2. The first character must not be a digit.

Valid names, for instance:

let userName;

let color1;  
let \_ssn;  
let $name;

Examples of incorrect variable names:

let 1a; // cannot start with a digit

let my-name; // a hyphen '-' is not allowed in the name

### **Reserved names**

There is a list of reserved words, which cannot be used as variable names, because they are used by the language itself.

For example, words let, class, return, function are reserved.

The code below gives a syntax error:

let let = 5; // can't name a variable "let", error!

let return = 5; // also can't name it "return", error!

## Constants

We use constants for everything that does not change, and we use the const keyword to declare them:

const PI = 3.141592653589793238;  
const BLUE = '#00F';

## Comments

You can put comments into JavaScript code, just as you can in CSS:

/\*

Everything in between is a comment.

\*/

If your comment contains no line breaks, it's often easier to put it behind two slashes like this:

// This is a comment

# Strings

A string in JavaScript must be quoted. In JavaScript, there are 3 types of quotes.

1. Double quotes: "Hello".
2. Single quotes: 'Hello'.
3. Backticks: `Hello`.

let phrase1 = "Double quotes are commonly used";

let phrase2 = 'But single quotes are used too';

let phrase3 = `Backticks can embed ${str}`;

Double and single quotes are “simple” quotes. There’s no difference between them in JavaScript.

Backticks are “extended functionality” quotes. They allow us to embed variables and expressions into a string by wrapping them in ${…}, for example:

let name = "John";

// embed a variable

alert( `Hello, ${name}!` ); // Hello, John!

or:

// embed an expression

alert( `the result is ${1 + 2}` ); // the result is 3

### **Escaping Characters**

There are certain characters that we will have to 'escape' when we write strings. For example the phrase:

'Peter's Corner'  
This phrase will give us an error because Javascript does not know exactly where the string really ends.

//This does not work. Notice the single quotes  
alert('Peter's Corner');  
// To fix it we have to escape the single quotes using the "\" like this:  
alert('Peter\'s Corner');  
//We can also use double quotes when we know that our text might have single quotes inside.  
alert("Peter's Corner");

If we want to add a new line to our text then we will use the "new-line character" \n

alert( "Happy\nBirthday" );

## String length

We can find out the length of a string with the length property:

let message = "Welcome to Miami, Bienvenido a Miami";  
alert( message.length ); // 36  
alert("Tuckerman’s Ravine".length); // 18

# Numbers

 A variable in JavaScript can contain any data. A variable can at one moment be a string and later receive a numeric value:

let n = 123;

let bitcoinPrice = 1287.345;  
let age = 45;  
let temp = 21.7;  
let total = 1000000000;  
let total = 1e9;

The number type serves both for integer and floating point numbers.  
There are many operations for numbers, e.g. multiplication \*, division /, addition +, subtraction - and so on. Javascript uses PEMDAS as the order of operation.

alert( 4 +5 ); // 9  
alert( 4 - 5); // -1  
alert( 4 \* 5); // 20  
alert( 4 / 5); // 0.8

Infinity represents the mathematical [Infinity](https://en.wikipedia.org/wiki/Infinity) ∞. It is a special value that’s greater than any number.

We can get it as a result of division by zero:

alert( 1 / 0 ); // Infinity

NaN represents a computational error. It is a result of an incorrect or an undefined mathematical operation, for instance:

alert( "a string divided by " / 2 ); // NaN, such division returns an error

## Rounding

One of the most used operations when working with numbers is rounding.  
There are several built-in functions for rounding:

**Math.floor**

Rounds down: 3.1 becomes 3, and -1.1 becomes -2.

alert( Math.floor(3.1) ); // 3

**Math.ceil**

Rounds up: 3.1 becomes 4, and -1.1 becomes -1.

alert( Math.ceil(3.1) ); // 4

**Math.round**

Rounds to the nearest integer: 3.1 becomes 3, 3.6 becomes 4 and -1.1 becomes -1.

alert( Math.round(3.1) ); // 3   
alert( Math.round(3.6) ); // 4

## Getting a Random Number

Math.random()

Returns a random number from 0 to 1 (not including 1)

alert( Math.random() );

### **Getting a random number between two values**

This example returns a random number between the specified values. The returned value is no lower than (and may possibly equal) min, and is less than (and not equal) max.

function getRandomArbitrary(min, max) {  
  return Math.random() \* (max - min) + min;  
}

### **Getting a random integer between two values**

This example returns a random integer between the specified values. The value is no lower than min (or the next integer greater than min if min isn't an integer), and is less than (but not equal to) max.

function getRandomInt(min, max) {  
  min = Math.ceil(min);  
  max = Math.floor(max);  
  return Math.floor(Math.random() \* (max - min)) + min; //The maximum is exclusive and the minimum is inclusive  
}

### **Getting a random integer between two values, inclusive**

While the getRandomInt() function above is inclusive at the minimum, it's exclusive at the maximum. What if you need the results to be inclusive at both the minimum and the maximum? The getRandomIntInclusive() function below accomplishes that.

function getRandomIntInclusive(min, max) {  
  min = Math.ceil(min);  
  max = Math.floor(max);  
  return Math.floor(Math.random() \* (max - min + 1)) + min; //The maximum is inclusive and the minimum is inclusive   
}

## Tests: isNaN

Sometimes we need to test whether the value of a variable is really a number. We can accomplish thi with the  isNaN(value) function.

isNaN(value) converts its argument to a number and then tests it for being NaN:

alert( isNaN(NaN) ); // true

alert( isNaN("str") ); // true  
alert( isNaN(5)); // false

# Booleans

## A boolean

The boolean type has only two values: true and false.

This type is commonly used to store yes/no values: true means “yes, correct”, and false means “no, incorrect”.

For instance:

let patriotsWin = true; // true value means patriots won

let otherWin = false; // false value means the other team lost

Boolean values also come as a result of comparisons:

let isGreater = 4 > 1;

alert( isGreater ); // true (the comparison result is "yes")

## The “null” value

The special null value does not belong to any type of those described in this module.

It forms a separate type of its own, which contains only the null value:

let age = null;

In JavaScript null is not a “reference to a non-existing object” or a “null pointer” like in some other languages.

It’s just a special value which has the sense of “nothing”, “empty” or “value unknown”.

The code above states that the age is unknown or empty for some reason.

## The “undefined” value

The special value undefined stands apart. It makes a type of its own, just like null.

The meaning of undefined is “value is not assigned”.

If a variable is declared, but not assigned, then its value is exactly undefined:

let x;

alert(x); // shows "undefined"

Technically, it is possible to assign undefined to any variable:

let x = 123;

x = undefined;

alert(x); // "undefined"

…But it’s not recommended to do that. Normally, we use null to write an “empty” or an “unknown” value into the variable, and undefined is only used for checks, to see if the variable is assigned or similar.

# 6. Variables

In this assignment, you will be creating variables.  
Create a new folder inside your local repository called '**week4**' (please use make sure to use this name).  
Download this file and save it to your 'week4' folder: [inclassvar.htmlPreview the document](https://bostoncollege.instructure.com/courses/1601933/files/66226379/download?wrap=1)  
Create a new file called var.js and save it inside your week4 folder.

All your work will happen inside your var.js file.

### **Part 1**

* Declare 3 string variables: first name, last name, and phone number
* Assign your own name and phone number to these variables
* Use the console.log command to display each one of these variable values

### **Part 2**

* Declare 3 number variables: rent, phone bill, internet
* Assign a number value to each one of these variables
* Use the alert() command to display each one of these variables

### **Part 3**

* Declare two logical variables, you choose the names (should be something that has a true/false or yes/no answer)
* Assign the "true" or "false" values
* Declare a variable and assign the null value to it
* Console log each one of these three variables

### **Part 4**

* Finally, open inclassvar.html file, open the developer tools and reload your file.

Make sure you submit both your inclassvar.html and your var.js files. And don't forget to push your files to Gitlab.