**Data Types and Variables**

**Intro to Data Types**

Everything can be represented with data. Data helps us to understand the world, recognise trends, make educated guesses, and inform our future decisions.

In programming, data and data types are the building blocks.

It’s important to be aware of what types of data we’re using and when it’s appropriate to use each type.

**Numbers**

The number data type includes any positive or negative integer, as well as decimal.

With number, we can perform calculations:

3 + 2.1  
***Returns:*** *5.1*

Just like in mathematics, we can compare two numbers to see if one is greater than, less than, or equal than the other:

5 > 10  
***Returns:*** *false*

5 < 10  
***Returns:*** *true*

5 == 10  
***Returns:*** *false*

|  |  |
| --- | --- |
| **Operator** | **Meaning** |
| < | Less than |
| > | Greater than |
| <= | Less than or Equal to |
| >= | Greater than or Equal to |
| == | Equal to |
| != | Not Equal to |

Comparisons between numbers will either evaluate to true or false. These values are called Booleans and are other data types in JavaScript.

**Comments**

We use comments to help explain our code and things easier. In JavaScript comments are marked with a double forward-slash // or /\* … \*/. It’s a good practice to include code comments to improve code readability.

*// this is a single-line comment*

*/\**

*this is*

*a multi-line*

*comment*

*\*/*

**Strings**

Strings can be single letters, like the character ‘h’, or even contain numbers, like ‘123’.

The important thing is that we need to use quotes to signify a string. It can be single or double quotes.

Strings are a collection of characters enclosed inside double or single quotes. We can use strings to represent data like sentences, names, addresses, and more.

**String Concatenation**

Strings can be added together - concatenation. Concatenating two strings together is actually pretty simple:

"Hello," + " New York City"  
***Returns:*** *"Hello, New York City"*

In JavaScript there is a peculiar behaviour called **implicit type coercion**, which can change a data type when concatenating:

"Hello" + 5\*10

***Returns:*** *"Hello50"*

**Variables**

With variables, we no need to work with one-time-use data.

Variables store data and can be reused at any time. We can store any data into variables.

To create a variable in JavaScript, we use the **var** keyword followed by the variable name, after the assignment operator we give the value.

**var** greeting = "Hello";

We can use the greeting variable at any time in our code. We don’t need to duplicate the “Hello” string:

greeting + " World!";  
***Returns:*** *Hello World!*

greeting + " Mike!";  
***Returns:*** *Hello Mike!*

We can also reassign a new value for the same variable, replacing the old one.

greeting = "Hola";

greeting + " World!";  
***Returns:*** *Hola World!*

greeting + " Mike!";  
***Returns:*** *Hola Mike!*

**Variables Naming Convention**

When we create a new variable, we give the name by using camelCase (the first word is lowercase, and all following starts with uppercase). Also, we try to use variable names that succinctly describes what the data is about:

**var** totalAfterTax = 53.03; *// uses camelCase if the variable name is multiple words*

**var** tip = 8; *// uses lowercase if the variable name is one word*

To know more about the convention access the [Google JavaScript Style Guide](https://google.github.io/styleguide/jsguide.html)

**Converting Temperature**

To convert Celsius to Fahrenheit we can use the formula:

F = C \* 1.8 + 32

var cel = 12;

var far = (cel \* 1.8) + 32;

console.log(far);

**String Index**

We can access individual characters in a string by using the character’s location - Index.

The index starts with 0 and can be used inside square brackets immediately after the string:

**var** name = "James";  
name[0];  
  
Returns “J”;

var phrase = "I love learning at Udacity!";

console.log(phrase[phrase.length - 1]);

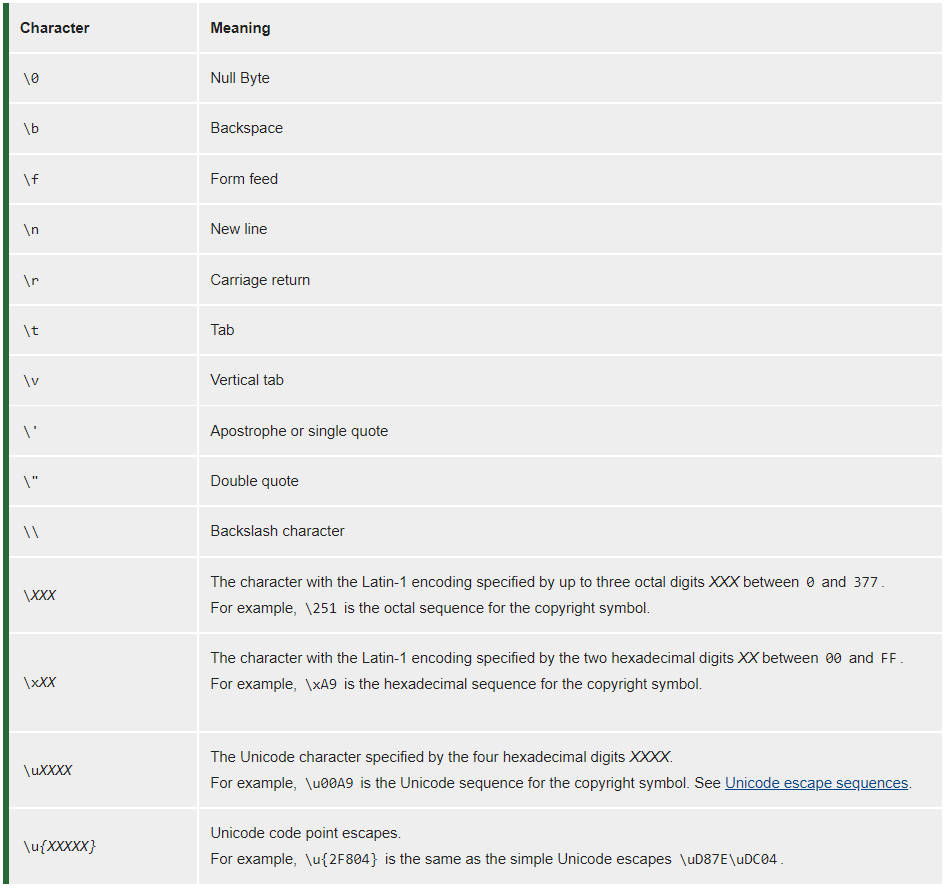
**Escaping Strings**

There are some cases where we might want to add quotes in our string. However, if we try to use quotes within a string, we will receive a SyntaxError . To avoid this error, we can use the backslash to escape other characters. Escaping character tells JavaScript to ignore the character’s special meaning and just use the literal value of the character.

console.log("The man whispered, \"please speak to me.\"");

console.log("The file located at \"C:\\\Desktop\\My Documents\\Roster\\names.txt\" contains the names on the roster.");

Quotes are not the only special characters that need to be escaped, there are [quite a few](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Grammar_and_types#using_special_characters_in_strings):



**Comparing Strings**

When comparing strings, we use the operator == and !=.

**Case Sensitive**

"Yes" == "yes"

***Returns*** = false

When comparing strings, case matters.

**Internal Working**

In JavaScript, strings are compared character-by-character in alphabetical order. Each character has a **numeric** value, coming from the [ASCII value of Printable characters](https://en.wikipedia.org/wiki/ASCII#Printable_characters).

For example, the character ‘A’ has a value of 65, and ‘a’ has a value of 97. To identify the ASCII value of a particular character, you can try running the code:

// Pick a string. Your string can have any number of characters.

var my\_string = "a";

// Calculate the ASCII value of the first character, i.e. the character at the position 0.

var ASCII\_value = my\_string.charCodeAt(0);

// Let us print

console.log(ASCII\_value);

To identify the ASCII value for multiple characters, we need loop the code:

var my\_string = "Udacity";

// Iterate using a Loop

for (var i = 0; i < my\_string.length; i++) {

console.log(my\_string.charCodeAt(i));

}

Because JavaScript uses ASCII values to compare strings, we can use the operators > and < as well.

**Booleans**

A boolean variable can take either of two values: true or false.

**var** studentName = "John";

**var** haveEnrolledInCourse = true;  
**var** haveCompletedTheCourse = false;

**if** (haveEnrolledInCourse){

console.log("Welcome "+studentName+" to Udacity!"); *// Will run only if haveEnrolledInCourse is true*  
}

A boolean variable is mainly essential in evaluating the outcome of conditionals (comparisons). **The result of a comparison is always a boolean variable**.

In general cases (regular equality check), a true corresponds to number 1, whereas false represents a number 0:

**if**(1){

console.log("This statement will always execute because conditional is set to 1 i.e., true");

}

**if**(0){

console.log("This statement will NEVER execute because conditional is set to 0 i.e., false");  
}

**Null, Undefined, and Na N**

There are couple more data types that we need to look at:

* Null: data type ‘value of nothing’. We can explicitly set a variable to a value of null.
* Undefined: data type ‘absence of value’. It’s not the same as null. Undefined it is when a variable exists but has no value assigned to it.



NaN stands for ‘Not a Number’ and it’s often returned indicating an error with number operations. For instance, if we wrote a code with some math calculation, and the calculation failed to produce a valid number, NaN might be returned:

*// calculating the square root of a negative number will return NaN*

Math.sqrt(-10)

*// trying to divide a string by 5 will return NaN*  
"hello"/5

**Equality**

There are situations that if using == and != to compare values with different data types, it can lead to some interesting results. For example:

"1" == 1

***Returns***: True

0 == false  
***Returns***: true. The == operator is unable to differentiate 0 from false.

' ' == false  
***Returns***: true. Both the operands on either side of the == operator are first converted to zero, before comparison.

The reason for such interesting outcomes is **Type Conversion**. In the case of regular comparison, the operands on either side of the == operator are first converted to numbers, before comparison. Therefore, a ' ', false, and 0 are all considered equal. Similarly, a '1' and 1 are also considered equal. If we don't want to convert the operands, before comparison, we have to use a strict comparison ===.

JavaScript first converts each operand to the same data type.

Because of the type coercion, it’s considered a bad practice to use == and != operators when comparing values for equality.

**Implicit Type Coercion**

JavaScript is a loosely typed language, which means that when we’re writing code, we don’t need to specify data types. JavaScript engine automatically converts into the appropriate data type - implicit type coercion:

"julia" + 1  
***Returns:*** *"julia1"*

**Example of strongly typed programming language code  
int** count = 1;  
string name = "Julia";  
**double** num = 1.2932;  
**float** price = 2.99;

**Equivalent code in JavaScript***// equivalent code in JavaScript*  
**var** count = 1;  
**var** name = "Julia";  
**var** num = 1.2932;  
**var** price = 2.99;

**Strict Equality**

To avoid errors related to the implicit type coercion, we use strict equality, by adding an additional equal sign to the end of == and != operators:

"1" === 1  
***Returns:*** *false*

**Semicolons**

We use semicolons at the end of each line to make it clear where one statement ends and another begins.

Not adding semicolons to the end of each line can cause bugs and errors in our programs.