JavaScript 1 - Module 1

Getting Started

Variables

Making Decisions

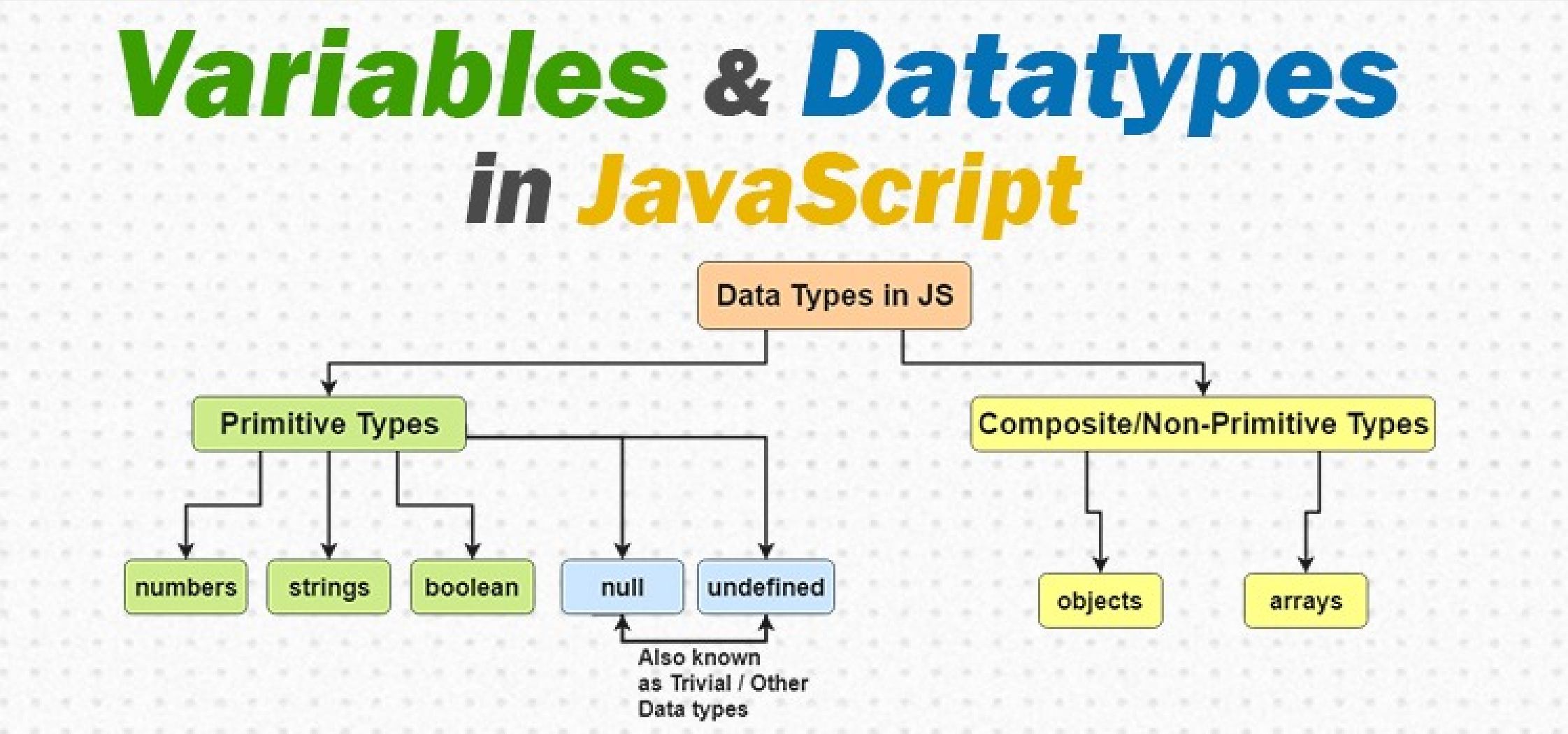
Loops

Solutions for JS1 Lesson 1.1 Getting Started

```
// -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
// Exercise 2
const p = document.getElementById("blankParagraph");
p.textContent = "Hello from xiaolasse";

// -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
// Exercise 3
const ul = document.getElementById("emptyList");
ul.innerHTML = "<li>HTML</li>";
ul.innerHTML += "<li>CSS</li>";
ul.innerHTML += "<li>JavaScript</li>"; // b

// -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
// Exercise 4

// 4a
// Setting the style of the p#blankParagraph red and bold
p.style.color = "red";
p.style.fontWeight = "bold";

// 4b
ul.style.color = "blue";
ul.style.fontStyle = "italic";

// 4c
document.body.style.backgroundColor = "beige";
```

```
// -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
// Exercise 5

// Writes the result of 7 x 6 to the page
document.write(7 * 6);

// 5b
// When the button is pressed, the whole page is deleted, then 42 is added

// -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
// Exercise 6

// Information window pops up with different labels:
// window.alert(23 + 19); // a: "42"
// window.alert("The answer is: " + 23 + 19); // b: "The answer is: 2319"
// window.alert("The answer is: " + (23 + 19)); // c: "The answer is: 42"

// -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
// Exercise 7

// Deletes all content on the page (though not the style)
// document.body.innerHTML = "";

// -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
```

Variables - strings, numbers and booleans



Graphic Source

JavaScript Programs

A computer program is a list of "instructions" to be "executed" by a computer.

In a programming language, these programming instructions are called `statements`.

A JavaScript program is a list of programming `statements`.

In HTML, JavaScript programs are executed by the web browser's JavaScript engine.

JavaScript Statements

JavaScript statements are composed of:

`Values`, `Operators`, `Expressions`, `Keywords`, and `Comments`.

This statement tells the browser to write "Hello World." inside an HTML element with the attribute id="demo":

```
document.getElementById("demo").innerHTML = "Hello World.";
```

Semicolons `;` separate JavaScript statements.

> Ending statements with semicolon is technically not required, but **highly** recommended.

JavaScript Variables

JavaScript variables are **containers** for storing data values.

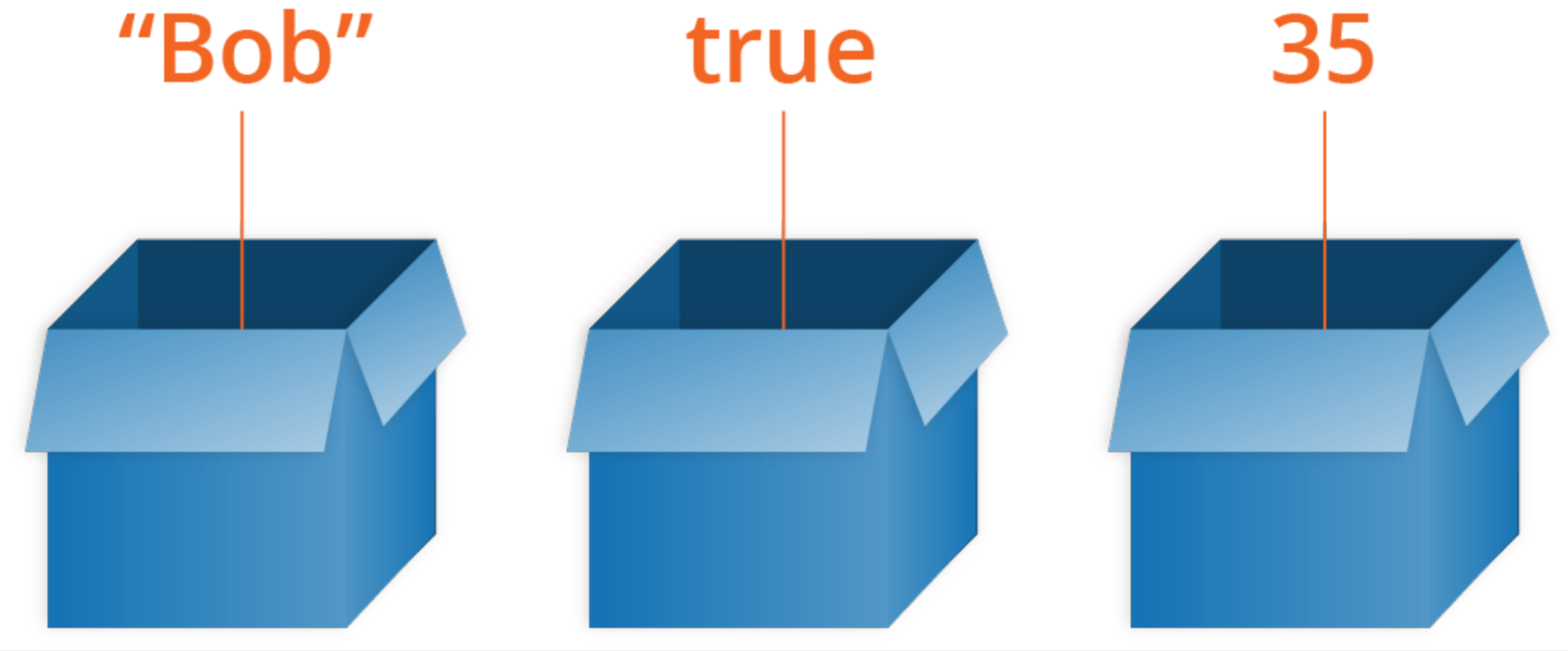
In the following example, `a`, `b`, and `sum`, are variables, declared using the `var` keyword:

```
var a = 13;
var b = 29;
var sum = a + b;
```

These are the values **assigned** to each variable:

`a` stores the value `13`
`b` stores the value `29`
`sum` stores the value `42`

Note: We say variables contain values. This is an important distinction to make. Variables aren't the values themselves; they are containers for values. You can think of them being like little cardboard boxes that you can store things in.



Using `let` and `const`

Before 2015, using the `var` keyword was the only way to declare a JavaScript variable.

The 2015 version of JavaScript (ES6 - ECMAScript 2015) allows the use of the `const` keyword to define a variable that cannot be reassigned, and the `let` keyword to define a variable with restricted *scope*.

Later in the course we will start using `let` and `const`, but for now let's stick to `var`.

JavaScript Identifiers

All JavaScript **variables** must be **identified** with **unique names**. These unique names are called **identifiers**.

Identifiers can be short names (like `x` and `y`) or more **descriptive names** (`age`, `sum`, `totalVolume`). In most cases it's adviceable to use descriptive names.

JavaScript identifiers are **case-sensitive**. This means `Apple` is different from `apple`.

Use camelCase for variable names with more than one word.

JavaScript Identifiers, cont.

The general rules for constructing **names for variables** (unique identifiers) are:

- Names can contain letters, digits, underscores, and dollar signs. Names cannot contain spaces.
- Names must begin with a letter (or with `$` or `_`).
- Names are case sensitive (`y` and `Y` are different variables).
- Reserved words (like JavaScript keywords) cannot be used as names.

Declaring JavaScript Variables

Creating a variable in JavaScript is called "**declaring**" a variable. You declare a JavaScript variable with the `var` keyword:

```
var programmingLanguage;
```

After the declaration, the variable has no value (technically it has the value of `undefined`).

To **assign** a value to the variable, use the equal sign:

```
programmingLanguage = "JavaScript";
```

You can also assign a value to the variable when you declare it:

```
var programmingLanguage = "JavaScript";
```

Declaring JavaScript Variables, cont.

You can also declare many variables in one statement. Start the statement with var and separate the variables by comma:

```
var x = "Dill", y = "Whatever", z = 42;
```

> Note: `var` is just used once, at the start of the statement but applies to all the variables!

It's a good programming practice to declare all variables you need at the beginning of a script.

JavaScript Data Types

JavaScript variables can hold many data types: `numbers`, `strings`, `objects` and more:

```
var length = 1609;                              // Number
var lastName = "Hægland";                       // String
var nn = {firstName:"John", lastName:"Doe"}; // Object
```

A data type is an attribute of data which tells the compiler or interpreter how the programmer intends to use the data.

This data type defines the operations that can be done on the data, the meaning of the data, and the way values of that type can be stored.

JavaScript Types are Dynamic

JavaScript has **dynamic types**. This means that the same variable can be used to hold different data types:

```
var x;          // Now x is undefined
x = 42;          // Now x is a Number
x = "Blåbær";    // Now x is a String
x = false;        // Now x is a Boolean
x = { a:1, b:2 }; // Now x is an Object
```

JavaScript Strings

A (text) **string** is a series of characters like **"Alphabet Soup"**.

Strings are written with quotes. You can use single or double quotes:

```
var carName1 = "Citroen C-Zero"; // Using double quotes
var carName2 = 'Citroen C-Zero'; // Using single quotes
```

You can use quotes inside a string, as long as they don't match the quotes surrounding the string:

```
var s1 = "It's alright";           // Single quote inside double quotes
var s2 = "He is called 'Johnny'"; // Single quotes inside double quotes
var s3 = 'He is called "Johnny"'; // Double quotes inside single quotes
```

> Tip: Use double quotes `"`, and **escape** any special chars: `var s4 = "It\'s alright"`

Escape Character

Because strings must be written within quotes, **JavaScript will misunderstand this string**:

```
var x = "We are the so-called "Vikings" from the north.";
```

The string will be chopped to "We are the so-called " (and you will get a "SyntaxError").

The solution to avoid this problem, is to use the **backslash escape character**.

The backslash (`\`) escape character turns special characters into string characters:

```
var x = "We are the so-called \"Vikings\" from the north.";
```

String Length

To find the length of a string, use the built-in `length` property:

```
var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZÆØÅ";
var len = txt.length;
console.log(len); // 29
```

JavaScript String Operators

The `+` operator can be used to add (*concatenate*) strings together:

```
var firstName = "Lasse";
var lastName = "Hægland";
var fullName = firstName + " " + lastName;
console.log (fullName); // Lasse Hægland
```

> Note that we add a space by putting `" "` between the two variables.

JavaScript String Operators, cont.

The `+=` assignment operator can also be used to add (concatenate) strings:

```
var txt1 = "What a very ";
txt1 += "nice day";
console.log (txt1); // What a very nice day
```

When used on strings, the `+` operator is called the **concatenation operator**.

> More on strings in lesson 2.1

JavaScript Numbers

JavaScript has only one type of numbers.

Numbers can be written with, or without decimals:

```
var x = 42.00;  // Written with decimals
var y = 42;     // Written without decimals
```

Arithmetic operators perform arithmetic on numbers:

JavaScript Arithmetic Operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation (ES2016)
/	Division
%	Modulus (Remainder)
++	Increment
--	Decrement

Arithmetic Operations

A typical arithmetic operation operates on two numbers.

The two numbers can be literals:

```
var x = 100 + 50;
console.log (x); // 150
```

or variables:

```
var a = 100, b = 50;
var x = a + b;
console.log (x); // 150
```

Arithmetic Operations, cont.

We can also use expressions with both literals and variables:

```
var a = 10;

var x = (100 + 50) * a;
console.log (x); // 1500

var y = 100 + 50 * a;
console.log (y); // 600
```

We'll look at *Operator Precedence* in a little bit...

Operators and Operands

The numbers (in an arithmetic operation) are called **operands**. The operation (to be performed between the two operands) is defined by an **operator**.

Operand	Operator	Operand
100	+	50

Adding

The addition operator (+) adds numbers:

```
var x = 5;
var y = 2;
var z = x + y; // result is 7
```

Subtracting

The subtraction operator (-) subtracts numbers.

```
var x = 5;
var y = 2;
var z = x - y; // result is 3
```

Multiplying

The multiplication operator (*) multiplies numbers.

```
var x = 5;
var y = 2;
var z = x * y; // result is 10
```

Dividing

The division operator (/) divides numbers.

```
var x = 5;
var y = 2;
var z = x / y; // result is 2.5
```

Exponentiation

The exponentiation operator (**) raises the first operand to the power of the second operand.

```
var x = 5;
var z = x ** 2; // result is 25
```

Remainder

The remainder operator (%) returns the remainder left over when one operand is divided by a second operand. It always takes the sign of the dividend.

```
var x = 5;
var y = 2;
var z = x % y; // Result is 1
```

A few examples about the Remainder:

```
 5 : 2 = 2          24 : 7           138 : 60 = 2
 4                  21               120
--                 ---              ----
 1 = remainder      3 = remainder     18 = remainder
```

```
139 : 5 = 27
10
--
 39
 35
---
  4 = remainder
```

> Try typing `5 % 2`, `24%7`, `138%60` and `139%5` directly into the Console

Related article: The Modulus Operator in JavaScript

Operator Precedence

Operator precedence describes the order in which operations are performed in an arithmetic expression.

```
var x = 100 + 50 * 3;
```

Is the result of example above the same as 150 * 3, or is it the same as 100 + 150? Is the addition or the multiplication done first?

As in traditional school mathematics, the multiplication is done first.

Multiplication (`*`) and division (`/`) have higher precedence than addition (`+`) and subtraction (`-`).

Operator Precedence, cont.

The precedence can be changed by using parentheses:

```
var x = (100 + 50) * 3;
```

When using parentheses, the operations inside the parentheses are computed first.

When many operations have the same precedence (like addition and subtraction), they are computed from left to right:

```
var x = 100 + 50 - 3;
```

MDN: Operator precedence

Adding Strings and Numbers

Adding two numbers, will return the sum, but adding a number and a string will return a string:

```
var x = 5 + 5;
var y = "5" + 5;
var z = "Hello" + 5;

console.log(x); // 10
console.log(y); // 55
console.log(z); // Hello5
```

> If you add a number and a string, the result will be a string!

Numeric Strings

JavaScript strings can have numeric content:

```
var x = 100;     // x is a number
var y = "10";    // y is a string
```

JavaScript will try to convert strings to numbers in all numeric operations (except `+`):

```
var x = "100";
var y = "10";
var z1 = x / y;  // z1 will be 10
var z2 = x * y;  // z2 will be 1000
var z3 = x - y;  // z3 will be 90
var z3 = x + y;  // z4 will NOT be 110 (It will be 10010)
```

> Avoid doing arithmetic directly on numeric stings; use the Number(), parseFloat() or parseInt() methods, first.

Converting Variables to Numbers

There are 3 JavaScript methods that can be used to convert variables to numbers:

- The `Number()` method
- The `parseInt()` method
- The `parseFloat()` method

These methods are not number methods, but global JavaScript methods.

> If the number cannot be converted, `NaN` (Not a Number) is returned.

The `Number()` Method

Returns a number, converted from its argument:

```
Number(true);           // returns 1
Number(false);          // returns 0
Number("10");           // returns 10
Number("  10");         // returns 10
Number("10  ");         // returns 10
Number(" 10  ");        // returns 10
Number("10.33");        // returns 10.33
Number("10,33");        // returns NaN
Number("10 33");        // returns NaN
Number("John");         // returns NaN

Number(new Date("2017-09-30"));
// returns 1506729600000
// the number of milliseconds since 1.1.1970
```

The `parseInt()` Method

Parses its argument and returns an integer:

```
parseInt("10");          // returns 10
parseInt("10.33");       // returns 10
parseInt("10 20 30");    // returns 10
parseInt("10 years");    // returns 10
parseInt("years 10");    // returns NaN
```

The `parseFloat()` Method

Parses its argument and returns a floating point number:

```
parseFloat("10");          // returns 10
parseFloat("10.33");       // returns 10.33
parseFloat("10 20 30");    // returns 10
parseFloat("10 years");    // returns 10
parseFloat("years 10");    // returns NaN
```

JavaScript Booleans

Booleans can only have two values: `true` or `false`.

```
var a = true;
var b = false;
```

> Note that `true` and `false` are written in small caps without any use of quotes.

Booleans are often used in conditional testing:

```
var x = 5, y = 5, z = 6;
(x == y)       // Returns true
(x == z)       // Returns false
```

> More on conditional testing in Lesson 1.3.

JavaScript Arrays

JavaScript arrays are written with square brackets `[]`.

Array items are separated by commas.

The following code declares (creates) an array called cars, containing three items (car names):

```
var cars = ["Saab", "Volvo", "BMW"];
```

Array indexes are zero-based, which means the first item is [0], second is [1], and so on.

> More about arrays in lesson 2.2

JavaScript Objects

JavaScript objects are written with curly braces `{}`.

Object properties are written as **name:value pairs**, separated by commas.

```
var person = {
    firstName:"Lasse",
    lastName:"Hægland",
    age:50,
    eyeColor:"blue"
};
```

The object (person) in the example above has 4 properties: `firstName`, `lastName`, `age`, and `eyeColor`.

> More about objects in lesson 2.3

The `typeof` Operator

You can use the JavaScript typeof operator to find the type of a JavaScript variable.

The typeof operator returns the type of a variable or an expression:

```
var a = 42, b = "42", c = "dill", d, e = true, f = {a: 1, b: 2}, g = [1, 2];

typeof a;  // Returns "number"
typeof b;  // Returns "string" !!!
typeof c;  // Returns "string"
typeof d;  // Returns "undefined"
typeof e;  // Returns "boolean"
typeof f;  // Returns "object"
typeof g;  // Returns "object" !!!
```

Todos

Github Classroom

JS1 Lesson 1.2 Variables

Mollify

Read Variables and do the Lesson Task

Extra

Read MDN: Storing the information you need — Variables