JAVA SCRIPT

**Data Types**

* A value in JavaScript is always of a certain type. e.g, string, number.
* There are eight (8) basic data types in JavaScript, these are:
  + number
  + bigInt
  + string
  + boolean
  + null
  + undefined
  + object
  + symbol
* We can put any type in a variable. For example, a variable can at one moment be a string and then store a number: **let message = “hello”; message = 123456;**
* Programming languages that allow such things, such as JavaScript, are called **“dynamically typed”**, meaning that there exist data types, but variables are not bound to any of them.

**Data types (number)**

* The number type represents both integer and floating point numbers.
* The following operations work on numbers:  **multiplication \*, division /, addition +, subtraction -.**
* There are **“special numeric values”** which also belong to this data type: **Infinity**, **-Infinity** and **NaN**.
* Infinity represents the mathematical Infinity **∞**. e.g divide by 0
* NaN represents a computational error. It is a result of an incorrect or an undefined mathematical operation, for instance: console.log(“name” / 2)
* NaN is sticky. Any further mathematical operation on NaN returns NaN
* So, if there’s a NaN somewhere in a mathematical expression, it propagates to the whole result (there’s only one exception to that: NaN \*\* 0 is 1).
* Doing maths is “safe” in JavaScript. We can do anything: divide by zero, treat non-numeric strings as numbers, etc. The script will never stop with a fatal error (“die”). At worst, we’ll get NaN as the result.

**Data types (BigInt)**

* The **bigInt** type represents extremely large or small integers which cannot be represented by the **number** type.
* The maximum and minimum size for a **number type is (253-1) and -(253-1)**
* This number type is enough for many purpose, so the bigInt type is rarely used.
* A BigInt is created by adding n to the end of a number or using the BigInt constructor

**Data types (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”.
* Boolean values are mostly gotten from comparisons:

**Data types (null)**

* The value null represents the intentional absence of a value
* The null type is a special value which represents “nothing”, “empty” or “value unknown”.
* Note that type of null is object for legacy reasons
* An unassigned variable is automatically set to type null
* Use null to assign an “empty” or “unknown” value to a variable

**Data types (undefined)**

* The value undefined means a value is not assigned
* A declared variable without assigned values becomes undefined by default
* **Show example of type of undefined**

**Data types (object)**

* Objects are one of the most important data type in JavaScript
* Objects are used to store keyed collections of data and more complex entities.
* All other types are called “**primitive**” because their values can contain only a single value
* Objects are associative arrays with several special features. They store properties (key-value pairs), where:
* You can retrieve values using dot notation or with square brackets
* Objects are store any valid JavaScript type including objects themselves
* delete a property: delete obj.prop.

**Data types (String)**

* **A String type is used to represent and manipulate a sequence of characters.**
* A string type must be surrounded by quotes or backticks.
* Double and single quotes are “simple” quotes. There’s practically 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 ${…}
* Other quotes don’t have this embedding functionality.
* Strings are special types, and therefore have extra functionalities built on it.
* You can call methods on strings to do special things like the examples below
* You can use special characters in strings, like the new line character (**\n**)
* Other special characters are \’, \”, \\, \t, \u{1F60D}.  Show some examples in console
* Access single characters in strings using subseting a[0]
* Strings are immutable, you can’t change a character
* Get substrings with slice
* Check the mozilla docs to see more helpful methods on strings : https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/String

**Terminologies**

* An operator is a reserved syntax consisting of punctuation or alphanumeric characters that carries out built-in functionality. e.g addition operator (“**+**“), subtraction operator (“**–**“).
* An operator can be **unary** or **binary**.
* An operand – is what operators are applied to.

**Math operators**

* The following math operations are supported in JavaScript:
  + Addition +,
  + Subtraction -,
  + Multiplication \*,
  + Division /,
  + Remainder %,
  + Exponentiation \*\*
* Remember If an expression has more than one operator, the execution order is defined by their [precedence](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Operator_Precedence), or, in other words, the default priority order of operators.
* **binary + is applied to strings, it merges (concatenates) them:**
* We know many operators from school. They are things like addition +, multiplication \*, subtraction -, and so on. **Show examples**
* Remember If an expression has more than one operator, the execution order is defined by their [precedence](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Operator_Precedence), or, in other words, the default priority order of operators.
* Parentheses override any precedence, so if we’re not satisfied with the default order, we can use them to change it. For example, write (1 + 2) \* 2.
* Let’s meet features of JavaScript operators that are beyond school arithmetics. So operators like **addition** works for strings. **Show example**
  + **if the binary + is applied to strings, it merges (concatenates) them:**

Next, let’s talk about JavaScript-specific operators, not covered by school arithmetic.