Lesson 4

Date	November 6, 2025
Topics	Conditions, loops, functions
Assignments	Delivery Assignment 01 + Start Assignment 02
Coding lab (16:00-1...	☐
Luca?	Written test + Talk
Talks	✓
Teacher	ML
Written test (13:30-...	✓

▾ Operators

Mathematics and strings

```
A + B // numerical sum or string concatenation A - B // numerical
subtraction A * B // multiplication A / B // division A ** B //
exponentiation (equivalent to Math.pow()) A % B // returns the
remainder left over
```

Assignment

```
A = B // assignment; A gets the value or property of B A++ // increment
by 1 A-- // decrement by 1 A += B // addition assignment; A increments
by B A -= B // subtraction assignment; A decrements by B A *= B //
multiplication assignment: A is multiplied by B A /= B // division
assignment; A is divided by B // example: A += B is quivalent to A = A
+ B
```

Comparison

```
A > B // grater then; returns true if A is bigger than B A >= B //
greater than or equal; returns true if A is bigger or equal to B A < B
// smaller then; returns true if A is smaller than B A <= B // smaller
than or equal; returns true if A is smaller or equal to B A == B //
equality; returns true if A is equal to B A != B // inequality; returns
true if A is different than B A === B // strict equality: returns true
when both the value and type of A are equal to B A !== B // strict
inequality: returns true when both the value and type of A are
different to B
```

Logical operands

```
conditionA && conditionB // logical AND: returns a Boolean value if
both A and B are true conditionA || conditionB // logical OR: returns a
Boolean value if either A or B is true !A // logical NOT (negation):
inverts the true to false and vice-versa
```

Strings

Strings are sequences of characters, written in **single** `'...'`, **double** `"..."`, or **backticks** `` `...` ``

```
let single = 'Leopard'; let double = "Leopard"; let template =
`Leopard`;
```

Concatenation

It is possible to connect strings by using the +

Combine (+)

```
let cat = 'Leopard'; let fur = "spotted"; let habitat = `forest`;
let about = 'The ' + cat + ' has a ' + fur + ' fur, and it lives in
the ' + habitat; console.log(about); // The Leopard has a spotted
fur, and it lives in the forset
```

Append (+=)

Change the current string appending to the end of it

```
let cat = 'Leopard'; cat += 's have a spotted fur.';
console.log(cat); // Leopards have a spotted fur.
```

▾ Template Literals

Use **backticks** ` to include variables and expressions inside `${}`.

```
let cat = 'Leopard'; let fur = "spotted"; let legs = 4; let habitat
= `forest`; let about = `The ${cat} has a ${fur} fur and ${legs}
legs, and it lives in the ${habitat}`; console.log(about); // The
Leopard has a spotted fur and 4 legs, and it lives in the forset
console.log(`2 ${cat.toLowerCase()}s have a combined number of
${legs * 2} legs.`); // 2 leopards have a combined number of 8
legs.
```

It is very useful in the DOM editing!

```
const list = document.getElementById('todo-list'); const
newItemContent = "My new task"; const itemNumber = 0; const
newElement = document.createElement('li'); newElement.innerHTML = `
// multiline HTML using template literals <p>${newItemContent}</p>
<button id="delete-btn-${itemNumber}">Remove</button> `;
list.appendChild(newElement);
```

▾ Common String Methods

length

Returns the number of characters

```
let cat = 'Leopard'; console.log(cat.length); // 7
```

toUpperCase()

```
'Leopard'.toUpperCase(); // 'LEOPARD' let cat = 'Leopard';
console.log(cat.toUpperCase()); // 'LEOPARD'
```

toLowerCase()

```
'Leopard'.toLowerCase(); // 'leopard' let cat = 'Leopard';
console.log(cat.toLowerCase()); // 'leopard'
```

charAt(index)

Returns the character at a specific **index** (starts form 0, zero-indexing)

```
'Leopard'.charAt(0); // 'L' 'Leopard'.charAt(1); // 'e'
'Leopard'.charAt(2); // 'o' let cat = 'Leopard';
console.log(cat.charAt(0)); // 'L'
```

includes(substr)

Returns **true** or **false** (boolean) if the **substring** is included in the string

```
'Leopard'.includes('opa'); // true 'Leopard'.includes('k'); //
false let cat = 'Leopard'; console.log(cat.includes('opa')); //
true console.log(cat.includes('k')); // false
```

indexOf(substr)

It searches for the **substring** and returns the index of the firs occurrence found in the string. If not found, it returns **-1**

```
'Leopard'.indexOf('L'); // 0 'Leopard'.indexOf('opa'); // 2
'Leopard'.indexOf('k'); // -1 let cat = 'Leopard';
console.log(cat.indexOf('L')); // 0
console.log(cat.indexOf('opa')); // 2
console.log(cat.indexOf('k')); // -1
```

slice(start, end) or substring(start, end)

Extract from **index start** (included), to **index end** (non included)

```
'Leopard'.slice(2, 5); // opa let cat = 'Leopard';
console.log(cat.slice(2, 5)); // opa
```

replace(search, replaceWith) and replaceAll(search, replaceWith)

Replace the **first match** with, and replace **all matches** with

```
'Hippopotamus'.replace('p', 'K'); // HiKpopotamus
'Hippopotamus'.replaceAll('p', 'K'); // HiKKoKotamus let animal =
'Hippopotamus'; console.log(animal .replace('p', 'K')); //
HiKpopotamus console.log(animal .replaceAll('p', 'K')); //
HiKKoKotamus
```

trim()

Remove whitespaces from both ends

```
' Leopard. '.trim(); // 'Leopard.'
```

repeat(n)

Repeat the string n times

```
"Hippopotamus".repeat(3); // HippopotamusHippopotamusHippopotamus
```

split(separator)

Convert to array, splitting the string at the separator character/s

```
"Leopard,Jaguar,Tiger,Lion".split(','); // ["Leopard", "Jaguar",
"Tiger", "Lion"] "Leopard and Jaguar and Tiger and Lion".split('
and '); // ["Leopard", "Jaguar", "Tiger", "Lion"] let cats =
'Leopard,Jaguar,Tiger,Lion'; console.log(cats.split(',')); //
["Leopard", "Jaguar", "Tiger", "Lion"]
```

▸ Functions

▸ Callback Function

Methods and . (dot) operator

A **method** is simply a **function that belongs to an object.** It does some kind of things with the object it belongs to.

```
let cat = "Leopard"; let catUppercase = cat.toUpperCase();
console.log(catUppercase); // LEOPARD
```

▼ Dot (.) member access operator

The `.` is called **member access operator.** It’s used to **access a property or a method of an object.** Methods are called with the **( )**. Only few default one don’t use the round parenthesis (string.length for example).

Syntax

```
object.method(arguments)
```

Example: DOM interaction

HTML

```
<button id="the-button">I'm the button!</button>
```

JS

```
const theButton = document.getElementById('the-button');
console.log(theButton); // <button id="the-button">I'm the button!
</button>
```

Example: String modification

```
let animal = 'Hippopotamus'; animal.replaceAll('p', 'K'); //
HiKKoKotamus
```

Default methods of the datatypes

String

Most of the methods returns a new string

```
string.length; string.toUpperCase(); string.toLowerCase();
string.trim(); string.includes(substring); string.replace(search,
replaceWith); string.repeat(n);
```

Number, integer (123) and float (3.14)

Numbers are primitive, but JS temporarily wraps them in a **Number object** to allow methods.

```
number.toFixed(n); // format to n decimals (3.14159).toFixed(2); //
3.14 toString(); // convert number to string (123).toString(); //
"123" toExponential(n); // scientific notation
(1239).toExponential(0); // 1e+3 (1239).toExponential(1); // 1.2e+3
(1239).toExponential(2); // 1.24e+3 (1239).toExponential(3); //
1.239e+3 (24056).toExponential(0); // 2e+4
(24056).toExponential(1); // 2.4e+4 (24056).toExponential(2); //
2.41e+4 (24056).toExponential(3); // 2.406e+4
(24056).toExponential(4); // 2.4056e+4
```

Boolean

```
(true).toString(); // "true" const result = 10 > 100; // false
result.toString(); // "false"
```

Array ([1, 2, 3])

Arrays are **objects**; many methods **modify the array**, some **return new arrays**

```
const array = [1, 2, 3]; array.length; // 3 array.pop(); // [1, 2]
array.push(value); array.map(function)
```

Object ({ key: value })

Objects don’t have too many built-in methods, but JS provides **global object helpers**.

```
const furrs = { // key: value leopard: "spotted", tiger: "striped",
lion: "plain" }; Object.keys(obj); // return an array of keys
Object.keys(furrs); // ["leopard","tiger", "lion"]
Object.values(obj); // return an array of values
Object.values(furrs); // ["spotted","striped", "plain"]
Object.entries(obj); // retrun an array of [key, value] pairs
Object.entries(furrs); // [["leopard", "spotted"], ["tiger",
"striped"], ["lion", "plain"]] obj.hasOwnProperty(key); // check if
key exists, retrun true or false furrs.hasOwnProperty("tiger"); //
true furrs.hasOwnProperty("hippopotamus"); // false
```

Object with method

Aside from the default JavaScript methods seen before, it is possible to build any custom object with any custom methods

```
const leopard = { // key: value name: "Leopard", fur: "spotted",
habitat: "forest", legs: 4, about: function() { // key (name of the
method): value (method itself) return `The ${this.name} has
${this.fur} fur and ${this.legs} legs, and lives in the
${this.habitat}.`; }, legsCount: function(leopardCount) { return `A
group of ${leopardCount} ${this.name.toLowerCase()}s has a combined
number of ${this.legs * leopardCount} legs`; } };
console.log(leopard.about()); // The Leopard has spotted fur and 4
legs, and lives in the forest. console.log(leopard.legsCount(3));
// A group of 3 leopards has a combined number of 12 legs
```

Arrays

In JavaScript, an array is defined as a special **variable** that **can hold multiple values at once**. The **values can be of any data type**—strings, numbers, objects, or even other arrays.
In JavaScript, **arrays are zero-indexed**, meaning the first element is at index `0`, the second element at index `1`, and so on.

Array declaration

Items in an array are numbered, starting from zero. This number is called **index.**

```
const cats = ["Leopard", "Jaguar", "Tiger", "Lion"];
console.log(cats[0]); // Leopard console.log(cats[1]); // Jaguar
console.log(cats[2]); // Tiger console.log(cats[3]); // Lion
```