Web Application Development
(LTAT.05.004)

JAVASCRIPT - I

MOHAMAD GHARIB
UNIVERSITY OF TARTU
How CSS works? - Recap

(2) It parses the html file
(3) It creates the DOM
(4) It fetches linked resources (CSS)
(5) It parses the fetched CSS, and (Document Object Model) creates the CSS Object Model (CSSOM)
(6) Render Tree is generated that contain only visible nodes.
(7) It calculates the positions and dimensions of each node on screen.
(8) It converts each node in the render tree to actual pixels on screen.
DOM - Recap

```html
<!DOCTYPE html>
<html>
  <head>
    <meta ... />
    <title>...</title>
  </head>
  <body>
    <h1>...</h1>
    <p>...<span>...</span>...</p>
    <ul>
      <li>...</li>
      <li>...</li>
      <li>...<span id: x>...</span>...</li>
    </ul>
    <p>...</p>
  </body>
</html>
```
JavaScript
Before we start ...

What do you need to know before starting?
• HTML, CSS and Document Object Model.

What we will cover:
• What JavaScript is,
• JavaScript data types, variables, loops, etc.
• How we can use JavaScript to dynamically create, update our page content, etc.

How this lecture contributes to the course

**HTML**
The structure of the webpage

**CSS**
The styling of the webpage

**JavaScript**
Makes a webpage interactive
Before we start ...

What do you need to know before starting?
- HTML, CSS and Document Object Model.

What we will cover:
- What JavaScript is,
- JavaScript data types, variables, loops, etc.
- How we can use JavaScript to dynamically create, update our page content, etc.

What can JavaScript do?
JavaScript can dynamically modify/manipulate an HTML page, validate user input (Form validation), react to events and user input (special effects), basic math calculations, etc.
JavaScript history

1993
Mosaic

1994
Netscape

1995
Mocha → liveScript → JS
Brendan Eich at Netscape

1995
IE

1997
ECMA-1 (European Computer Manufacturers Association) - ES1
JavaScript history

- **1993** Mosaic
- **1994** Netscape
- **1995** IE
- **1995** Mocha → liveScript → JS
  - Brendan Eich at Netscape
- **1997**
  - ECMA-1 (European Computer Manufacturers Association) - ES1
- **1998**
  - ECMAScript 2 (ES2)
- **2000-2008**
  - ES4 - Never released
- **2000-2005**
  - AJAX
- **2001** JSON
  - JSON support
- **2009**
  - ES5

Regular expressions
- try/catch
- switch do-while
JavaScript history

1993 Mosaic
1994 Netscape
1995 IE
1995 Mocha → liveScript → JS
1998 ECMAScript 2 (ES2)
1999 ES3
1999 ECMA1 (European Computer Manufacturers Association) - ES1
1999 Brendan Eich at Netscape
2000-2008
2000-2005
AJAX
2001
JSON
2009
ES5
2015
ES6 / ECMAScript 2015
let and const
2016
ECMAScript 2016
2020
ECMAScript 2020
What is JavaScript?

JavaScript is a lightweight client-side programming language ("scripting language") that allows you to implement complex things on web pages.

JavaScript can be described as multi-paradigm, non-blocking and asynchronous language.

Different paradigms in JavaScript:

Object-Oriented Programming that has four main principles: abstraction, inheritance, encapsulation and polymorphism.

Functional programming promotes the concept of modularity, generating small blocks of pure functions to comprise the more prominent application.

Event-driven programming, where the flow of the program is determined by events such as user actions (mouse clicks, key presses).
JavaScript vs Java

JavaScript is not Java, it is related to Java by name and some similarities.

JavaScript has some features that resemble features in Java:
JavaScript has Objects and primitive data types.
JavaScript has Events and event handlers.
JavaScript - Statements resemble statements in Java.
JavaScript - Exception handling is almost the same as in Java.

Why?
JavaScript vs Java

JavaScript is not Java, it is related to Java by name and some similarities.

JavaScript has some features that resemble features in Java:
- JavaScript has Objects and primitive data types.
- JavaScript has Events and event handlers.
- JavaScript - Statements resemble statements in Java.
- JavaScript - Exception handling is almost the same as in Java.

Why? because both languages borrowed heavily from the C language.
JavaScript vs Java

JavaScript is not Java, it is related to Java by name and some similarities.

JavaScript has very different features than Java:

JavaScript is interpreted in the browser while Java is compiled\(^1\) and can run as stand alone applications.

JavaScript has more relaxed syntax and rules than Java: fewer and "looser" data types; variables do not need to be declared; errors often silent, etc.

JavaScript can access browser objects and functionality, while Java cannot.

JavaScript is seldom used to write complete “programs”.

JavaScript is reasonably platform-independent.

\(^1\)server-side JavaScript is compiled
Where can you put JavaScript code?
Where do you put JS code?

In HTML, JavaScript code is inserted between the `<script>` and `</script>` tags.

```html
<script>
    document.querySelector("h1").style.fontSize = '90px';
</script>
```

JS code can be placed:
- Externally to the page(s), in a separate .js file.
- Within the `<head>` of an HTML document.
- Within the `<body>` of an HTML document.
External JS code

```html
<html>
<head>
<script type="text/javascript" src="src/js/script.js"></script>
</head>
<body>
<h1> Page heading</h1>
</body>
</html>
```

**script.js**

```javascript
document.querySelector("h1").style.fontSize = '90px';
```

**JS code can be placed:**
- **Externally to the page(s), in a separate .js file.**
- **Within the <head> of an HTML document.**
- **Within the <body> of an HTML document.**
JS code can be placed:

- Externally to the page(s), in a separate .js file.
- **Within the <head> of an HTML document.**
- Within the <body> of an HTML document.
JS code within the `<body>`

```html
<html>
<head> .. </head>
<body>
<script>
    document.querySelector("h1").style.fontSize = '90px';
</script>
<h1> Page heading</h1>
</body>
</html>
```

JS code can be placed:

- Externally to the page(s), in a separate .js file.
- Within the `<head>` of an HTML document.
- Within the `<body>` of an HTML document.
Where do you put JS code?

If you tried any of the previous solutions, you’ll get an error.

Why?
Where do you put JS code?

If you tried any of the previous solutions, you’ll get an error.

Why?
Where do you put JS code?

If you tried any of the previous solutions, you’ll get an error.

Why?

How it can be solved?
Where do you put JS code?

If you tried any of the previous solutions, you’ll get an error.

**Why?**

**How it can be solved?**

The code **should** be executed after the HTML and CSS have been assembled and **put together into a web page.**

```javascript
window.onload = function() {
  document.querySelector("h1").style.fontSize = '90px';
};
```
JavaScript comments can be used to explain JavaScript code, and to make it more readable.

JavaScript comments can also be used to prevent execution, when testing alternative code.

```javascript
// single line comment
/*
Multi-line comment
Multi-line comment
*/

//shortcut for commenting/
uncommenting in VSCode Ctr +/
```
Output to screen

The **log()** method writes (logs) a message to the console.

The **log()** method is useful for testing purposes.

The **error()** method writes an error message to the console.

The **error()** method can be very useful as it report the error line number.

The **table()** writes a table to the console.

```javascript
> console.log("message");
// The message to write to the console is required

> console.error("error message");
// The error message to write to the console is required

> console.table(["error", "message"]) // array

> console.table({firstMessage:"error", secondMessage:"message"}); // objects
JS variables and constants
A **variable** is a **container** for a **value**, like a number we might use in a sum, or a string that we might use as part of a sentence. The main feature of a **variable** is that its **values** can **change**.
Declaring and initializing variables

// Declaring and initializing a variable syntax
// [Variable keyword] | variable name = variable value

> var name1 // declaring – you can use let instead of var

> name1 = "John" // initializing

> var name1 = "John" // OK – declaring and initializing

> name1 = "Paolo" // Updating the value of a variable

> name2 = "Sarah" // OK, but not a good idea

> name3 // Uncaught ReferenceError: name3 is not defined

Variable Naming: you can use any style to name your variables, but you should use Latin characters (0-9, a-z, A-Z) and the underscore character (_). Common styles: Pascal case (FirstName), camel case (firstName), snake case (first_name), kebab case (first-name).
var vs let

```javascript
var foo = true;
if (foo) {
    var bar = foo * 2;
    console.log(bar);
}
for (var i = 0; i < 10; i++) {
    console.log(i);
}
console.log(bar); // 2
console.log(i); // 10
```

```javascript
let foo = true;
if (foo) {
    let bar = foo * 2;
    console.log(bar);
}
for (let i = 0; i < 10; i++) {
    console.log(i);
}
console.log(bar); // error
console.log(i); // error
```

let was introduced in ES6 (2015)
**var vs let**

<table>
<thead>
<tr>
<th>var</th>
<th>let</th>
</tr>
</thead>
<tbody>
<tr>
<td>var foo = true;</td>
<td>let foo = true;</td>
</tr>
<tr>
<td>if (foo) {</td>
<td>if (foo) {</td>
</tr>
<tr>
<td>var bar = foo * 2;</td>
<td>let bar = foo * 2;</td>
</tr>
<tr>
<td>console.log(bar);</td>
<td>console.log(bar);</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
</tr>
<tr>
<td>for (var i = 0; i &lt; 10; i++) {</td>
<td>for (let i = 0; i &lt; 10; i++) {</td>
</tr>
<tr>
<td>console.log(i);</td>
<td>console.log(i);</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
</tr>
<tr>
<td>console.log(bar); // 2</td>
<td>console.log(bar); // error</td>
</tr>
<tr>
<td>console.log(i); // 10</td>
<td>console.log(i); // error</td>
</tr>
</tbody>
</table>

**var**
- var is function-scoped.
- var allows to redeclare variables.
- Hoisting occurs in var.

**let**
- let is block-scoped.
- let does not allow to redeclare variables.
- Hoisting does not occur in let.

let was introduced in ES6 (2015)
Dynamic typing

JavaScript is a dynamically-typed (loosely-typed) language, which means that, unlike strongly typed languages, you do not need to explicitly specify types of variables (e.g., numbers, strings, arrays).

```javascript
//String
let myVar = "String"

//Number
myVar = 20

// Array/Object
myVar = [10, 15, 17];

//Boolean
myVar = true;
```
**Constants**

`const` is a value that once **declared and initialized**, and its **value** cannot be changed after that.

`const` was introduced in **ES6 (2015)**.  
`const` cannot be **Redeclared**.  
`const` cannot be **Reassigned**.  
`const` cannot be used before **declaring** it.  
`const` is **Block-scoped**.

```javascript
//const declaration and initialization
const daysInWeek = 7;

//declaration but no initialization
const daysInWeek; // Error

//redeclare
const daysInWeek // Error

//reassign/change value
daysInWeek = 8 // Error

//Access before declare
console.log(daysInWeek);
const daysInWeek = 7 // Error
```
JS data types
There are **five different data types** that can contain values:

- Number
- String
- Object
- Boolean
- Function

There are **two data types** that cannot contain values:

- Null → Object
- Undefined
There are **five different data types** that can contain values:

- **Number**
- **String**
- **Object**
- **Boolean**
- **Function**

There are **two data types** that cannot contain values:

- **Null → Object**
- **Undefined**

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

```
//Number
closeup.log(typeof 1000);

//String
closeup.log(typeof "WAD");

// Object
console.log(typeof {name: "Sarah", age: 17});

//Array -> Object. It is considered as object
console.log(typeof [10, 15, 17]);

//Boolean
console.log(typeof true);

//Function
console.log(typeof function(){


//Null -> Object. It is considered as object
console.log(typeof null);

//Undefined
console.log(typeof Undefined);
```
JavaScript has only one type of number. Numbers can be written with or without decimals. Unlike many other programming languages, JavaScript does not define different types of numbers, like integers, short, long, floating-point etc. Extra large or extra small numbers can be written with scientific (exponent) notation.

```javascript
let x = 3.14; // A number with decimals
let y = 3; // A number without decimals

// Extra large or extra small numbers
let x = 123e5; // 12300000
let y = 123e-5; // 0.00123
```
## Arithmetic Operators

<table>
<thead>
<tr>
<th>Oper.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
</tr>
<tr>
<td>%</td>
<td>Remainder</td>
</tr>
<tr>
<td>**</td>
<td>Exponent</td>
</tr>
<tr>
<td>++</td>
<td>Increment</td>
</tr>
<tr>
<td>--</td>
<td>Decrement</td>
</tr>
</tbody>
</table>

```javascript
let x = 5;
let y = 6;
console.log(x + y);  // 11
console.log(x - y);  // -1
console.log(x * y);  // 30
console.log(x / y);  // 0.8333333333
console.log(x % y);  // 5
console.log(x ** y); // 15625

console.log(x ++);  // 6
console.log(y --);  // 5
```
What will be the value of variable `num3`?

- 12
- 5
- 2.4

```
let num1 = 16;
let num2 = 8;
let num3 = num2 + num1 / 8 + 2;
```
What will be the value of variable `num3`?

- 12
- 5
- 2.4

```
let num1 = 16;
let num2 = 8;
let num3 = num2 + num1 / 8 + 2;
```
Assignment operators

<table>
<thead>
<tr>
<th>Oper.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+=</td>
<td>( x = x + y )</td>
</tr>
<tr>
<td>-=</td>
<td>( x = x - y )</td>
</tr>
<tr>
<td>*=</td>
<td>( x = x * y )</td>
</tr>
<tr>
<td>/=</td>
<td>( x = x / y )</td>
</tr>
<tr>
<td>%=</td>
<td>( x = x % y )</td>
</tr>
<tr>
<td>**=</td>
<td>( x = x ^ y )</td>
</tr>
</tbody>
</table>

```javascript
let x = 5;
let y = 6;
console.log(x += y); // 11
console.log(x -= y); // -1
console.log(x *= y);  // 30
console.log(x /= y);  // 0.8333333333
console.log(x %= y);  // 5
console.log(x **= y); // 15625
```
## Comparison operators

<table>
<thead>
<tr>
<th>Oper.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>==</code></td>
<td>equal to</td>
</tr>
<tr>
<td><code>===</code></td>
<td>equal value and equal type</td>
</tr>
<tr>
<td><code>!=</code></td>
<td>logical not</td>
</tr>
<tr>
<td><code>!==</code></td>
<td>not equal value or not equal type</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>greater than</td>
</tr>
<tr>
<td><code>&lt;</code></td>
<td>less than</td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td>greater than or equal to</td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td>less than or equal to</td>
</tr>
</tbody>
</table>

```javascript
let x = 5;
console.log(x == 5);  // true
console.log(x == '5'); // true
console.log(x ===5);  // true
console.log(x ==='5'); // false
console.log(x !=8);   // true
console.log(x !=='5'); // true
console.log(x < 5);   // false
console.log(x < 5);   // false
console.log(x >= 5);  // true
console.log(x >=5);   // true
```
Logical operators

<table>
<thead>
<tr>
<th>Oper.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;&amp;</td>
<td>logical and</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>!</td>
<td>logical not</td>
</tr>
</tbody>
</table>

```javascript
console.log(true && true); // true
console.log(false && true); // false
console.log(true && false); // false
console.log(false && false); // false

console.log(true || true); // true
console.log(false || true); // true
console.log(true || false); // true
console.log(false || false); // false

console.log(!true); // false
```
String

A JavaScript string is zero or more characters written inside double or single quotes.

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

```javascript
//const declaration and initialization
console.log("Web App. Development");
console.log('Web App. Development');
console.log('Web App. "Development"');

//Escape character
console.log('"WAD"'); //“WAD”

// \n newline
console.log("Web\n App.\n Development");
/*
 Web
 App.
 Development
 */
```
String methods and properties

The `length` property returns the length of a string.

The `slice(start, end)` method extracts a part of a string and returns the extracted part in a new string.

The `replace()` method replaces a specified value with another value in a string.

The `toLowerCase()` method converts a string to lower case.

The `toUpperCase()` method converts a string to upper case.

```javascript
let a = "Web Application";
console.log(a.length); // 15
console.log(a.slice(0,3)); // Web
//substring(start, end), substr(start, length)

let c = a.replace("Web Application", "WAD")
console.log(c); // WAD

console.log(c.toLowerCase()); // wad
console.log(c.toUpperCase()); // WAD
```
String methods and properties

The `concat()` method joins two or more strings.

The `trim()` method removes whitespace from both sides of a string.

The `trimStart()`/`trimEnd()` method works like `trim()`, but removes whitespace only from the start/end of a string.

```javascript
let b = "Development";
c = a.concat(" ", b);
console.log(c); // Web Appl. Development
console.log("WAD" +""+ "wad"); // WAD wad

let d = "   trim  ";
console.log(d); // trim
console.log(d.trim()); // trim

console.log(d.trimStart()); // trim
```
Arrays can be described as "list-like objects"; they are basically single objects that contain multiple values stored in a list.

Accessing and modifying array items: you can access/modify individual items in the array using bracket notation + index.

The length property provides an easy way to know the number of items in an array.

```javascript
let shopping = ['bread', 'milk', 'eggs'];
let sequence = [1, 1, 2, 3, 5, 8, 13];
let random = ['tree', 795, [0, 1, 2]];

shopping[0]; // returns "bread"
shopping[0] = 'noodles';
// ["noodles", "milk", "eggs"]

shopping.length; // returns 3
```
Arrays methods

**push()** adds a new element to the end of an array.

**pop()** removes an element from the end of an array.

```javascript
let shopping=['bread','milk','cheese'];
shopping.push('eggs');
// shopping list will now contain
['bread', 'milk', 'cheese', 'eggs']
shopping.pop();
//remove "eggs" from the shopping array
```

Javascript array methods: [https://www.w3schools.com/js/js_array_methods.asp](https://www.w3schools.com/js/js_array_methods.asp)
unshift() adds a new element to the begging of an array.

splice() removes element(s) and add others from an array.

array.splice(index, number of element to be removed, item1, item2, item3, to be added).

```javascript
let shopping=['bread','milk','cheese'];
shopping.unshift('eggs');
// ['eggs', 'bread', 'milk', 'cheese']

let shopping=['bread','cheese','eggs'];
shopping.splice(1, 2, 'apples');
// ['bread', 'apples']
```

Javascript array methods: https://www.w3schools.com/js/js_array_methods.asp
Arrays methods

`sort()` and `reverse()` can be used to order the element of an array.

```javascript
let shopping=['eggs','bread','cheese'];
shopping.sort();
// ['bread', 'cheese', 'eggs']
shopping.reverse();
// ['eggs', 'cheese', 'bread']
```

`indexOf()` can be used to find the order of an element in an array.

```javascript
let shopping=['eggs','bread','cheese'];
shopping.indexOf('eggs');
// 0
```

Javascript array methods: [https://www.w3schools.com/js/js_array_methods.asp](https://www.w3schools.com/js/js_array_methods.asp)
Objects

Objects are one of the JavaScript data types, and can be used to store various **keyed collections** and more complex entities.

An **Object** is an entity that have a state and behaviour (properties and method).

```javascript
const person = {
    name: ['Bob', 'Smith'],
    age: 32,
    gender: 'male',
    interests: ['music', 'skiing'],
    bio: function() {
        alert(this.name[0] + ' ' + this.name[1]);
    }
};
```
Conditional Statements
if is used to specify a block of code to be executed, if a specified condition is true.

else is used to specify a block of code to be executed, if the same condition is false.

else if is used to specify a new condition to test, if the first condition is false.

```java
if (condition1) {
    //executed if condition1 is true
} else if (condition2) {
    //executed if the condition1 is false and condition2 is true
} else {
    //executed if the conditions are false
}
```
if is used to specify a block of code to be executed, if a specified condition is true.

else is used to specify a block of code to be executed, if the same condition is false.

else if is used to specify a new condition to test, if the first condition is false.

```java
if (time < 10) {
    greeting = "Good morning";
} else if (time < 20) {
    greeting = "Good day";
} else {
    greeting = "Good evening";
}
```
Switch statement

**switch statement** is used to select **one** of many code blocks to be **executed**.

The switch expression is **evaluated once**.
The **value** of the expression is compared with the **values** of each case.
If **there is a match**, the associated block of code is executed.
If **there is no match**, the **default** code block is executed.

```python
switch(expression) {
    case x:
        // code block
        break;
    case y:
        // code block
        break;
    default:
        // code block
}
```
Switch statement

**switch statement** is used to select one of many code blocks to be **executed**.

The switch expression is **evaluated once**. The **value** of the expression is compared with the **values** of each case. If **there is a match**, the associated block of code is executed. If **there is no match**, the **default** code block is executed.

```java
switch (choice) {
    case 'sunny':
        wearTshirt();
        break;
    case 'rainy':
        wearJacket();
        break;
    default:
        stayHome();
}
```
The **ternary operator** is a simplified conditional operator like if / else.

```java
condition ? <expression if true> : <expression if false>

authenticated ? renderApp() : renderLogin();

// comparing it with the if statement
if (authenticated) { renderApp(); } else { renderLogin(); }
```
Loops
Loops

A loop requires:

A **counter**, which is initialized with a certain value — this is the starting point of the loop.

An **exit condition**, which is the criteria under which the loop stops — usually the counter reaching a certain value.

An **iterator**, which generally increments the counter on each successive loop, until it reaches the exit condition.
Loops

JavaScript supports different kinds of loops:

**for - loops** through a block of code a number of times.

**for/in - loops** through the properties of an object.

**for/of - loops** through the values of an iterable object.

**while - loops** through a block of code while a specified condition is true.

**do/while - loops** through a block of code while a specified condition is true.
Loops

JavaScript supports different kinds of loops:

for - loops through a block of code a number of times.

for/in - loops through the properties of an object.

for/of - loops through the values of an iterable object.

while - loops through a block of code while a specified condition is true.

do/while - loops through a block of code while a specified condition is true.

```javascript
for(i=0; i<10; i++) {
  //executed while i <10
  if (i == 5) {
    break;
  }
  // you can use the break statement to exit a loop
  if (i == 7) {
    continue;
  }
  // The continue statement skips to the next iteration of the loop.
}
```
Loops

JavaScript supports different kinds of loops:

for - loops through a block of code a number of times.

for/in - loops through the properties of an object.

for/of - loops through the values of an iterable object.

while - loops through a block of code while a specified condition is true.

do/while - loops through a block of code while a specified condition is true.

```javascript
for (key in Object) {
  .. code
}
// The JavaScript for in statement loops through the properties of an Object.
```
Loops

JavaScript supports different kinds of loops:

for - loops through a block of code a number of times.

for/in - loops through the properties of an object.

for/of - loops through the values of an iterable object.

while - loops through a block of code while a specified condition is true.

do/while - loops through a block of code while a specified condition is true.

```javascript
for(variable of iterable)
{
  .. code
}

// The JavaScript for of statement loops through the values of an iterable object.
```
Loops

JavaScript supports different kinds of loops:
for - loops through a block of code a number of times.
for/in - loops through the properties of an object.
for/of - loops through the values of an iterable object.
while - loops through a block of code while a specified condition is true.
do/while - loops through a block of code while a specified condition is true.

While(i<10) {
    //execute while i <10
    i++
}
do {
    //execute
    i++
}While(i<10)
//continue executing if i <10
Pros and Cons of JavaScript

Pros:
• JavaScript is the first web scripting language.
• JavaScript is easier to learn than most programming languages.
• JavaScript allows you to make interactive Web pages … It can be fun.
• The advantages of JavaScript are too many to list here ……

Cons:
• Requires a JavaScript-enabled browser (not really a problem now).
• JavaScript has some security mechanisms but can still cause security problems for clients.
Keep in mind

JavaScript is not a very simple language
  We will cover the essentials (scratch the surface).
  If you want to take full advantage of JS, you need to invest time in learning it professionally.

JavaScript is not totally platform independent
  Expect different browsers to behave differently.

Browsers are not designed to report errors
  Do not expect to get helpful error messages (errors often silent).
JavaScript

Try it yourself …
Open developer tool: right clicking, inspect, then, select the console tab.

Copy document.write("<h1> This is an h1 heading </h1>") into the console and press enter. What do you see? Check the Elements tab, is there an h1 tag?

Copy document.querySelector("h1").style.color = 'blue'; into the console and press enter. Check the styles of the HTML page.
Copy `document.querySelector("h1").style.fontSize = '90px';` into the console and press enter. Check the styles of the HTML page again.

Write `window` in the console and press enter. Navigate through the window’s methods, try to use some of these methods.

Use the console to declare and initialize variables, do basic operations and math …
Setting up your environment - Extensions

**Live server**: lunches a development local server with live reload feature for static and dynamic pages.

**Indent-rainbow**: a simple extension to make indentation more readable by colorizing the indentation in front of your text alternating four different colours on each step.

**Prettier - Code formatter**: enforces a consistent style by parsing your code and re-printing it with its own rules.

**Material icon theme**: offers vast amount of icon theme that surely will help you locate your files based on their types.
Extra reading/exercises

• **W3Schools** is a free educational website for learning to code online. With their "Try it Yourself" editor, you can edit the HTML code and view the result immediately: https://www.w3schools.com/js/

• **Edabit** (https://edabit.com/challenges) is a coding platform with over 10,000 interactive coding challenges ranging from Very Easy to Expert.

• **LeetCode** (https://edabit.com/challenges) is also a coding platform with interactive coding challenges, but its challenges are more complex than Edabit.
Thank You for your attention

Mohamad Gharib
mohamad.gharib@ut.ee

unitartu
tartuuniversity