ΕΠΛ425

Τεχνολογίες Διαδικτύου

(Internet Technologies)

Introduction to Frond-End Development:

HyperText Markup Language (HTML)

Cascading Style Sheets (CSS)

JavaScript (JS)

Διδάσκων

Δρ. Χριστόφορος Χριστοφόρου

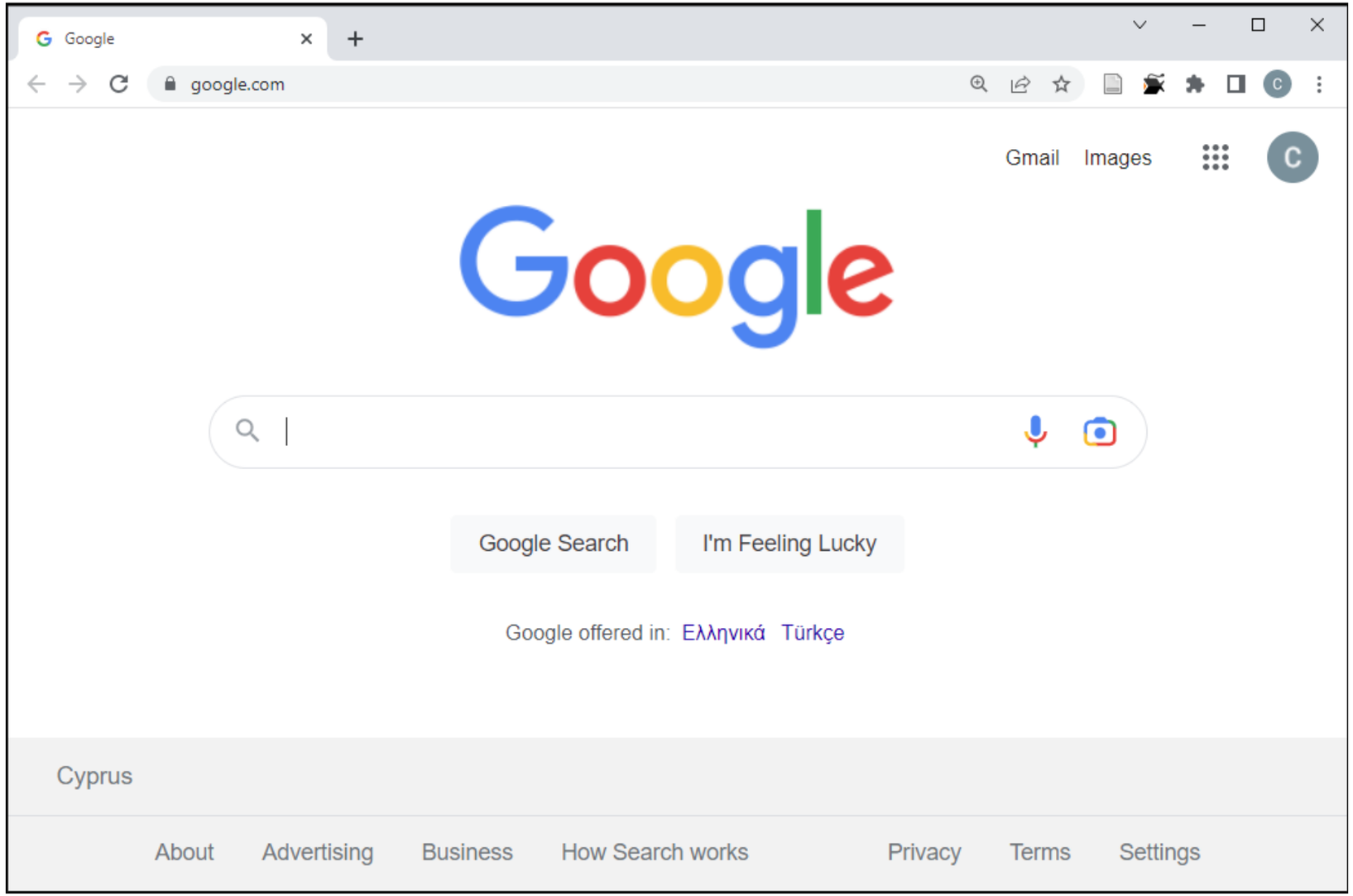
christophoros@cs.ucy.ac.cy

Goals

Introduction to Front-End Development:

- **HTML** to create the document structure and content
- **CSS** to control its visual/stylist aspect
- **Javascript** for interactivity

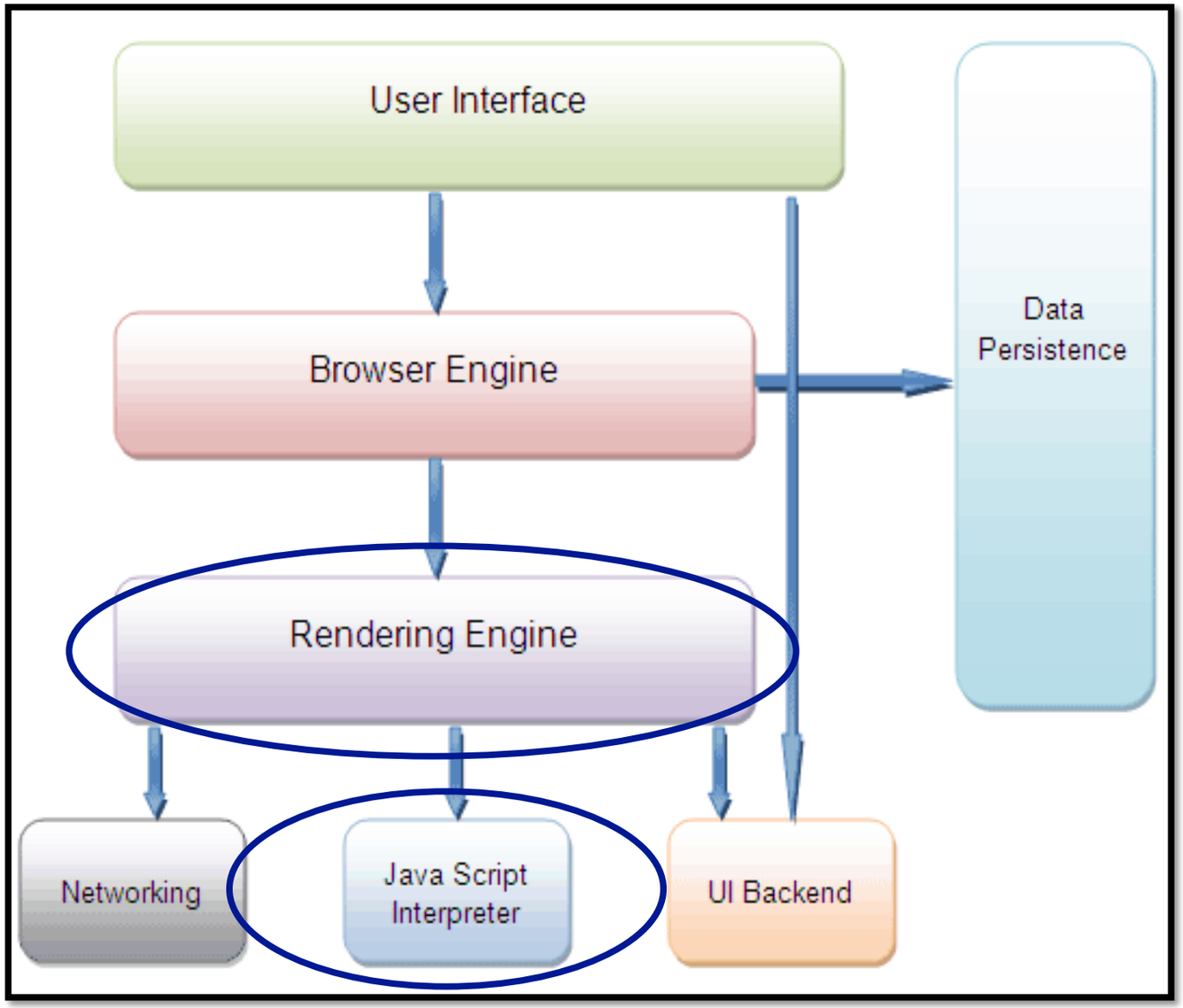
Anatomy of a Browser



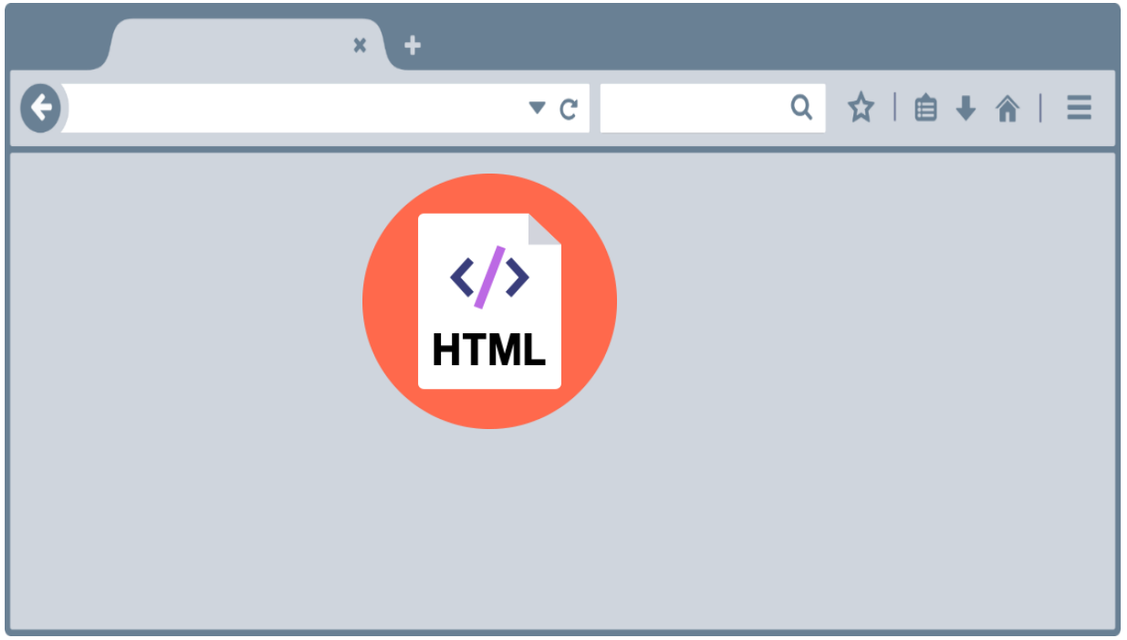
Anatomy of a Browser

Browsers have very **differentiate parts**. We are **interested** in **two of them**:

- the **Rendering Engine,** in charge of **transforming** our **HTML + CSS** in a visual image.
- The **Javascript Interpreter** (also known as VM), in charge of **executing** the **Javascript** code.



How do web pages work?




```html
<!DOCTYPE HTML>
<html>

<head>
    <title>An Example using HTML CSS and JavaScript</title>
    <link rel="stylesheet" href="CSS/style.css" />
    <script src="JS/code.js" defer></script>
</head>

<body>
    <h1>An Example including HTML CSS and JavaScript</h1>
    <button id="btn1">Click Me!</button>
    <p id="greeting"></p>
</body>

</html>
```

Web pages are written in a **markup language** called **HTML.**

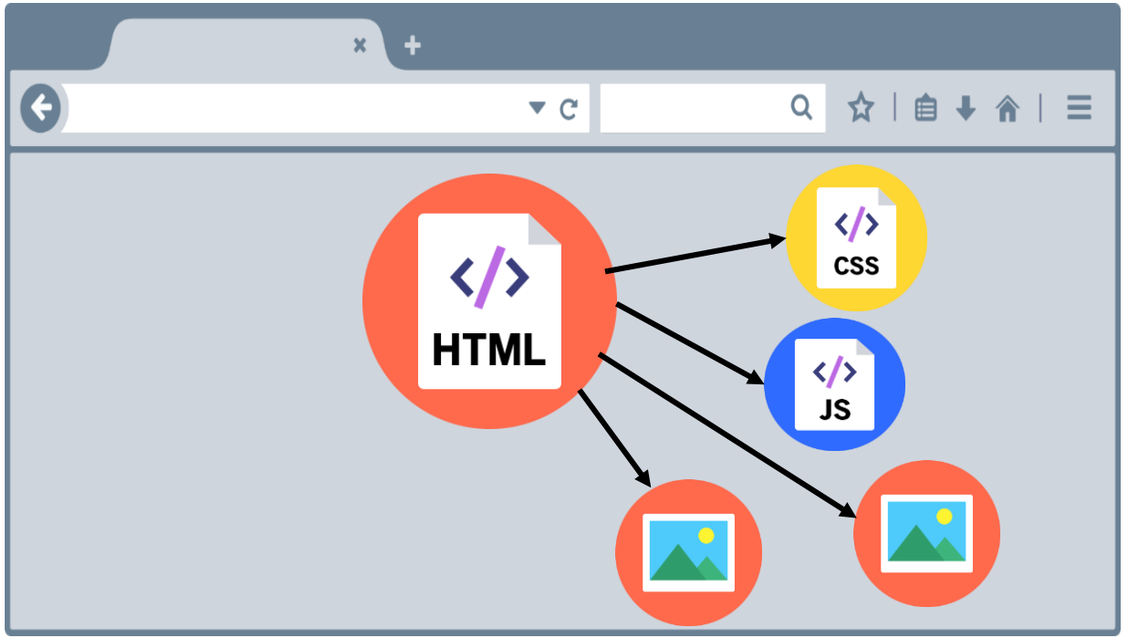
Browsers **display** a web page by **reading** and **interpreting** its **HTML code**.

How do web pages work?


```html
<!DOCTYPE HTML>
<html>

<head>
    <title>An Example using HTML CSS and JavaScript</title>
</head>

<body>
    <h1>An Example including HTML CSS and JavaScript</h1>
    <button id="btn1">Click Me!</button>
    <p id="greeting"></p>
</body>

</html>
```

This is what would be displayed in the browser without CSS and JavaScript

How do web pages work?

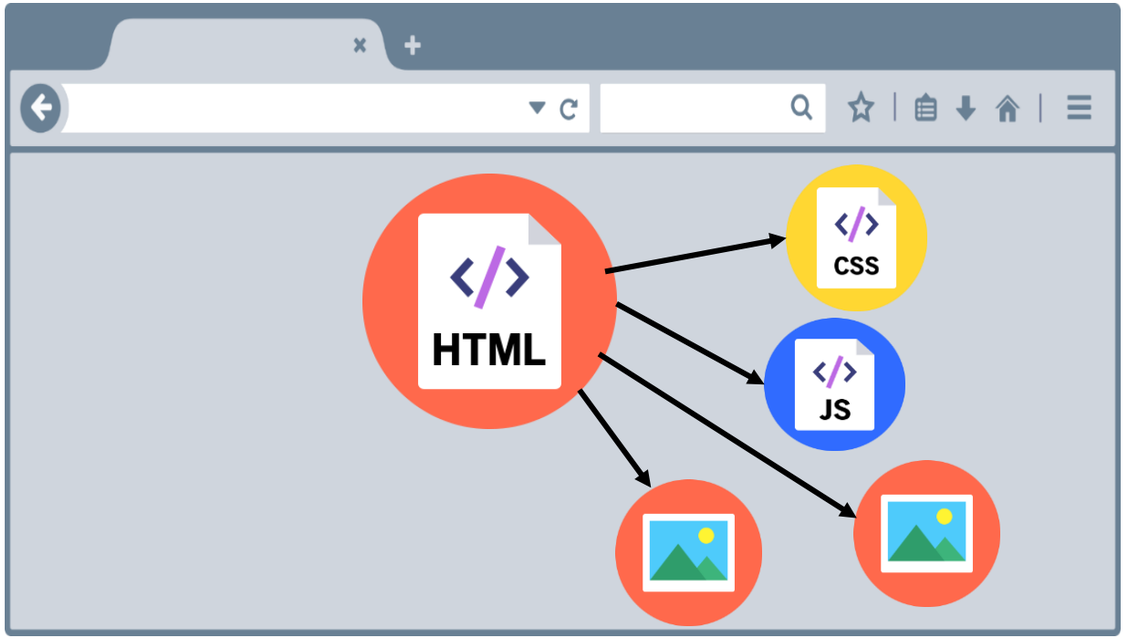


index.html

```html
<!DOCTYPE HTML>
<html>

<head>
    <title>An Example using HTML CSS and JavaScript</title>
    <link rel="stylesheet" href="CSS/style.css" />
    <script src="JS/code.js" defer></script>
</head>

<body>
    <h1>An Example including HTML CSS and JavaScript</h1>
    <button id="btn1">Click Me!</button>
    <p id="greeting"></p>
</body>

</html>
```

The **HTML file** might **link** to **other resources**, like images, videos, as well as **JavaScript** and **CSS** (stylesheet) **files**, which **the browser then also loads**.

How do web pages work?



style.css file in CSS folder

```
html,
body {
    height: 100%;
    width: 100%;
}

h1 {
    width: 50%;
    text-align: center;
    margin-top: 0;
    font-size: xx-large;
    line-height: 1.2;
    color: white;
    background-color: black;
}

p {
    font-size: large;
    line-height: 1.2;
    color: blue;
}

button {
    border: 2px solid grey;
    font-size: 20px;
    background-color: darkgray;
    size: 20px;
    height: 50px;
    width: 100px;
}
```

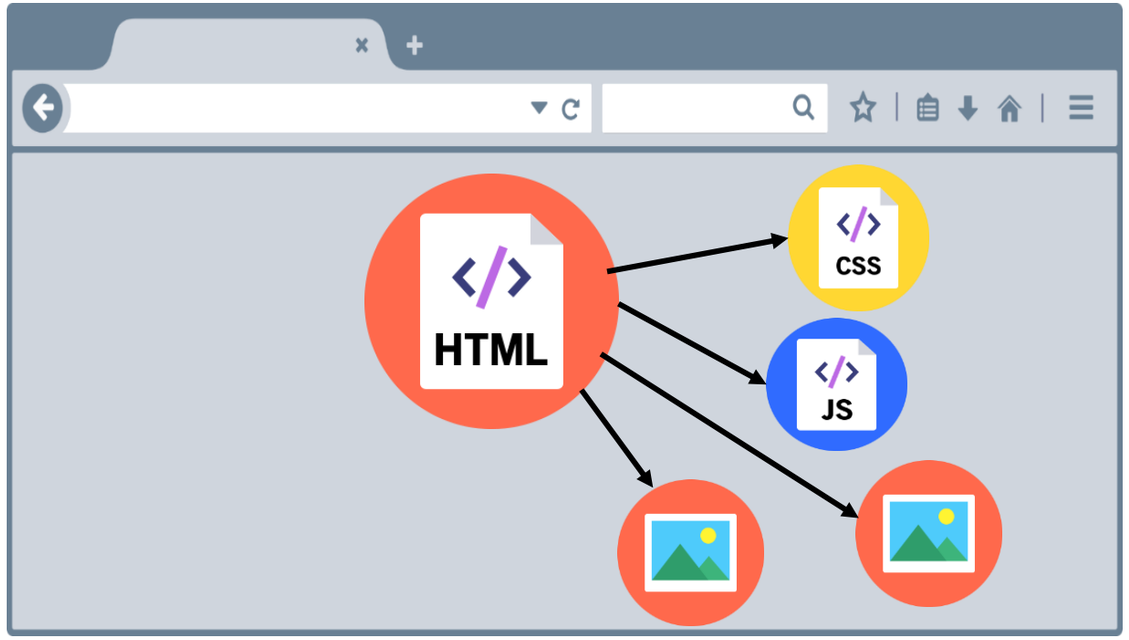
CSS allows us to **specify how to present (render)** the **document** info stored in the HTML.

How do web pages work?

This is what will be displayed in the browser with CSS!!!

However, we are missing now is Interactivity.
If you press the button, nothing will happen!
We need JavaScript for that!!!

How do web pages work?



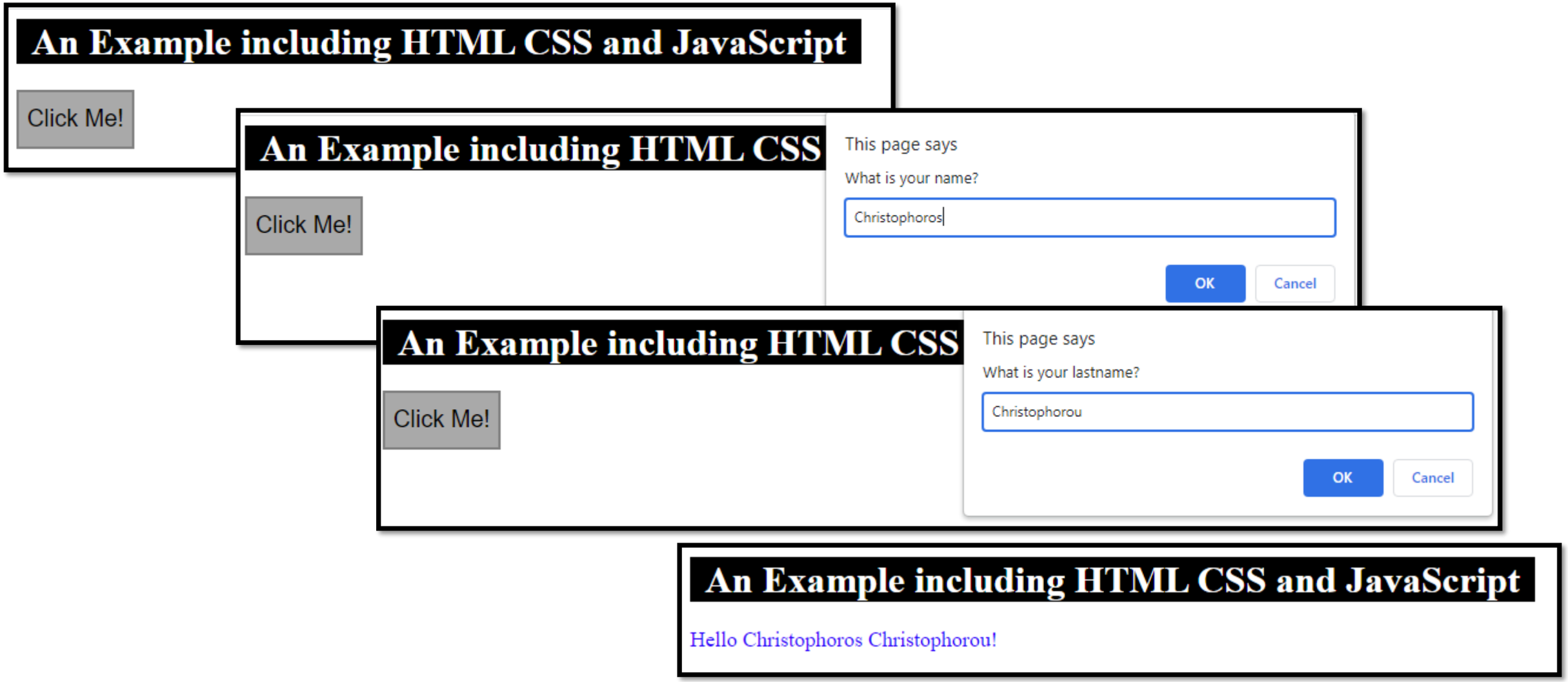
JavaScript allows **interaction with the user**. With JavaScript you can **change the content of the HTML** or the **CSS style** applied to an element **without reloading the page**.

code.js file in JS folder

```
// First Fetch the button from the DOM
let button = document.getElementById("btn1");

// Attach an event when the user clicks it.
// When clicked myFunction will be invoked
button.addEventListener("click", myFunction);

// Create the function that will be called when the
// button is clicked.
function myFunction() {
    // Gets user input
    let onoma = prompt("What is your name?");
    let epitheto = prompt("What is your lastname?");

    // Hide the button from being redered
    button.style.display = "none";

    // Fetch the p node from the DOM that the greeting
    // message will be displayed and assiged to it the
    // greeting string to be dispayed in the browser.
    let par = document.getElementById("greeting");
    par.innerHTML = "Hello " + onoma + " " + epitheto + "!";
}
```

How do web pages work?

This is what will be displayed in the browser when you click the button!!!



Front-End Technologies

- HTML
- CSS
- Javascript

HTML

```
<!DOCTYPE HTML>
<html>

<head>
    <title> </title>
</head>

<body>
    ……
</body>

</html>
```

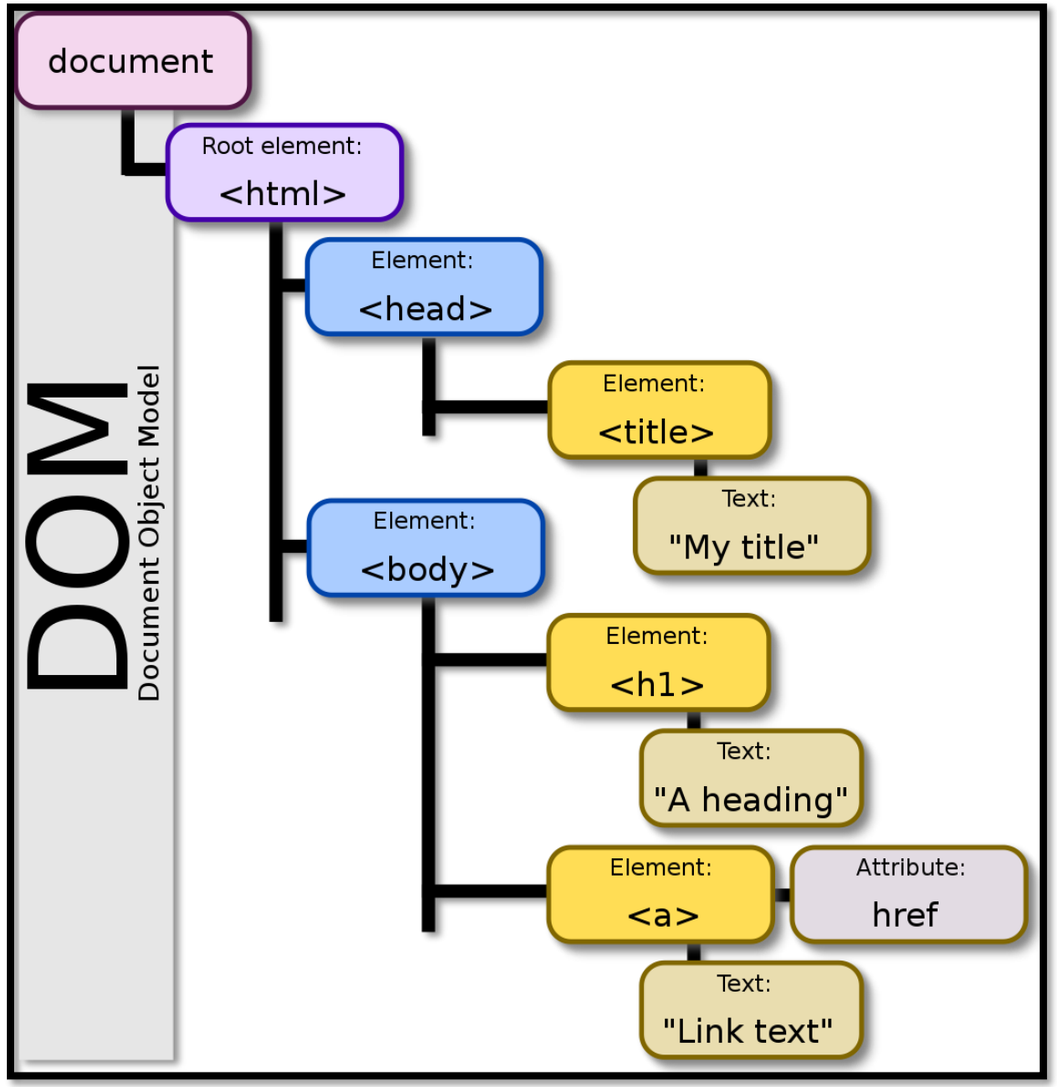
- **Web pages** are text files containing HTML. HTML means **Hyper Text Markup Language** and allow us to **define** the **structure** and **the content** of a **document** or a website.
- HTML is **NOT a programming language**, **it's a markup language**, which means its **main purpose** is to **give structure to the content** of the website.
- It is a series of **nested tags** (it is a subset of XML) that **contain all the web page information** (like texts, images and videos). Here is an example of a tag:

  <title>This is a title</title>

- The web page information is **stored** in a **tree-like structure** (**elements that contain other elements**) called **Document Object Model (DOM).**

HTML

Every element can only have one parent, and every element can have several children, so the structure looks like a tree.



HTML Tags

- **HTML Tags** **defines** **HTML Elements**. Tags can also have **attributes**.

Syntax:	*<tag_name **attribute**="value"> content </tag_name>*
Examples:	<h1 **id**="heading1">My First Heading</h1>
	<p **id**="parag1">My first paragraph.</p>

- **Tags** gives the document some **semantic structure** (e.g., this is a title, this is a heading, this is a paragraph, this is a form, etc.) which is **helpful for computers** to understand websites content.

- It **should NOT contain** **information** related to **how it should be displayed** (that information belongs to the **CSS**), so no color information, font size, position, etc.

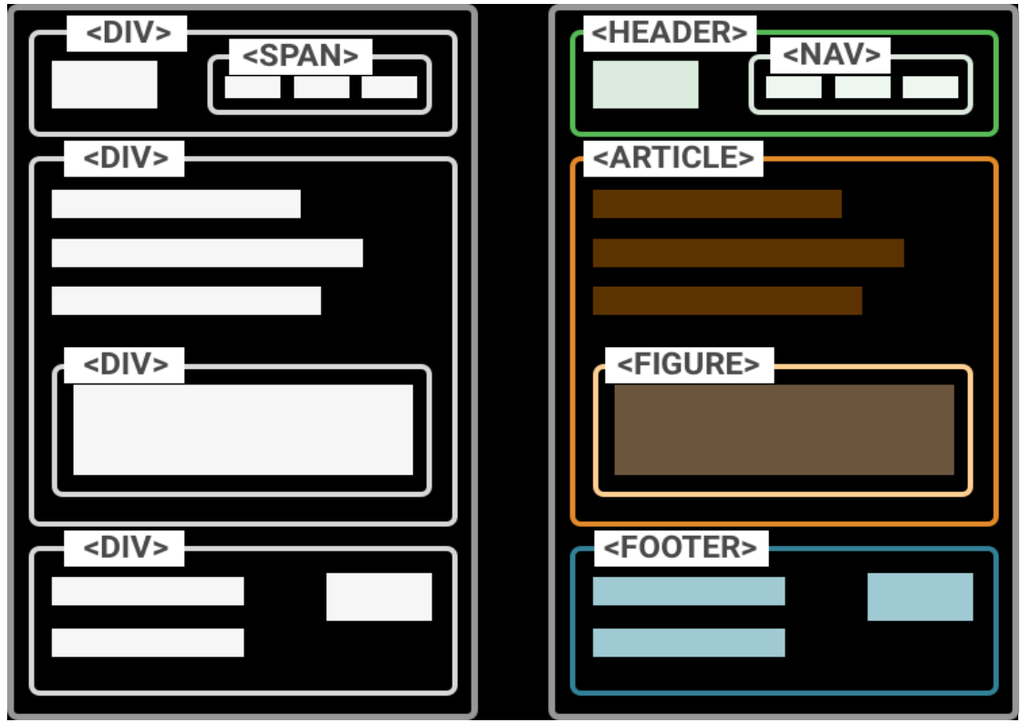
HTML: Main Tags

Although there are lots of tags in the HTML specification, 99% of the webs use a subset of **HTML tags** with less that 10 tags. The most important are:

- ** :** a container, usually represents a rectangular area with information inside.
- **<img>:** an image
- **<a>:** a clickable link to go to another URL
- **<p>:** a text paragraph
- **<h1>:** a title (h2,h3,h4 are titles of less importance)
- **<input>:** a widget to let the user introduce information
- **<style>:** to insert CSS rules
- **<link>:** to define a relationship between the document and an external resource (usually a CSS file)
- **<script>:** to embed in the document a client-side script (JavaScript)
- **:** a null tag (doesn't do anything)

HTML: Wrapping the info

- We use **HTML tags** to **wrap different information** on our site.
- The **better the structure** of the HTML file, **THE EASIER** will be **to access** the **different elements contained**, **through JavaScript and CSS**.



DO NOT DO THIS

```
<div>
Heading Title

Here is some content.
Here is more content.
</div>
```

DO THIS

```
<div>
    <h1>Heading Title</h1>
    <p>Here is some content.</p>
    <p>Here is more content.</p>
</div>
```

HTML: Other interesting Tags

There are some tags that could be useful sometimes:

- **<button>:** to create a button
- **<audio>:** for playing audio
- **<video>:** to play video
- **<canvas>:** to draw graphics from javascript
- **<iframe>:** to put another website inside ours

HTML Good Use

- It is good to **have all the information properly wrapped in tags** that **give it some semantics**.
- We also can **extend the HTML code semantics** by adding **extra attributes** to the tags:

  id: tells a **unique identifier** for this tag

  class: tells a **generic identifier** for this tag

```
<button class="btns" id="btn1">Press me</button>
```

The **class** and **id** attributes helps us to **access and manipulate** the **elements contained**, **through JavaScript and CSS**.

HTML: Syntax example

Tag name

Attribute

Comment using <!-- … --> Tag

```
<div id="main">
    <!-- this is a comment -->
     This is text without a tag. <!- don't do that -->
    <button class="btns" id="btn1">Press me</button>
    <img src="me.png" />
</div>
```

Self-closing tag

We will see more about HTML5 Syntax in a future Lecture!!!

HTML References To Read

- A description of **all HTML tags** is provided here.
- A list of **all HTML attributes** and by **what HTML elements can be used** within is provided here.
- A **list of properties** and **methods** that **can be used by all HTML elements** is provided here.
- Some **guidelines** and **tips** for **creating good HTML code** is provided here.

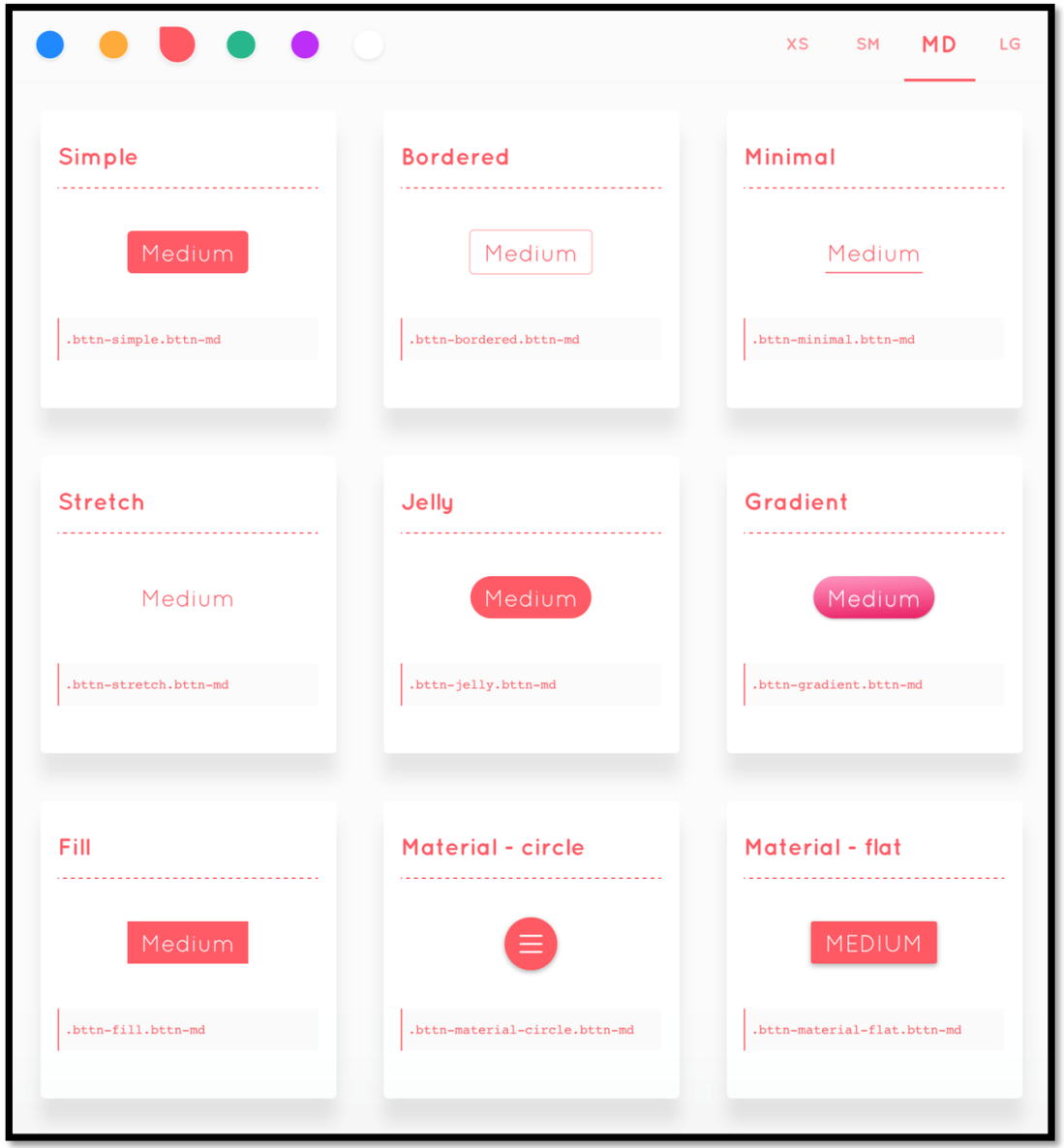
Front-End Technologies

- HTML
- **CSS**
- Javascript

Cascading Style Sheets (CSS)

CSS allows us to specify **how to present (render)** the document info stored in the HTML. Thanks to CSS we can **control all the aspects of the visualization**:

- **Colors:** content, background, borders
- **Margins:** interior margin, exterior margin
- **Position:** where to put it
- **Sizes:** width, height
- **Behaviour:** changes on mouse over



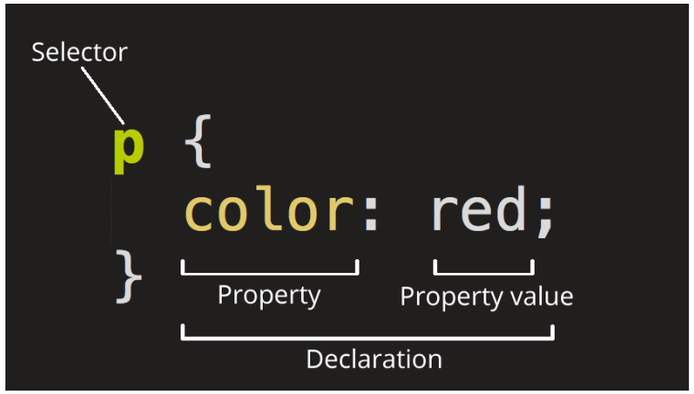
CSS Rulesets

- **A CSS Ruleset** determine the **style/design** and **behavior** of **HTML element(s)**. Examples include: **color**, **border**, **margin**, **font.**
- A **CSS declaration** in a **CSS ruleset** appears as a **property: value** pair, for example: `color: red;`
- **Each CSS Ruleset** is **assigned** to a **selector** with **one** or **more CSS Declarations**.

Here is the CSS syntax.
A selector with a property declaration.

```
selector {
    property1: value1;
    property2: value2;
}
```

Anatomy of a CSS ruleset



CSS examples

```
* {
    color: blue; /* a comment */
    margin: 10px;
    font: 14px Tahoma;
}
```

This will change all the tags in the web page (* means all) to look blue with font Tahoma and size 14px, and leaving a margin of 10px around.

Comments in CSS are included using /* … */

CSS examples

```
p {
    color: blue; /* a comment */
    margin: 10px;
    font: 14px Tahoma;
}
```

This will change all the <p> tags in my web site to look blue with font Tahoma and size 14px, and leaving a margin of 10px around.

CSS examples

```
p h1 {
    color: blue; /* a comment */
    margin: 10px;
    font: 14px Tahoma;
}
```

This will change all the <p> and <h1> tags in my web site to look blue with font Tahoma and size 14px, and leaving a margin of 10px around.

CSS Properties

Here is a list of the most common CSS properties with an example:

```
color: #FF0000;   red;   rgba(255,00,100,1.0); /* different ways to specify colors */
background-color: red;
background-image: url('file.png');
font: 18px 'Tahoma';
border: 2px solid black;
border-top: 2px solid red;
border-radius: 2px;                /* to remove corners and make them more round */
margin: 10px;                      /* distance from the border to the outer elements */
padding: 2px;                      /* distance from the border to the inner elements */
width: 100%;  300px;  1.3em;       /* many different ways to specify distances */
height: 200px;
text-align: center;
box-shadow: 3px 3px 5px black;
cursor: pointer;
display: inline-block;
overflow: scroll;         /* handles content that exceeds the size of the container */
```

CSS Properties

- CSS supports more than 200 CSS properties.
- A complete list of CSS properties and the details of each is provided here.

CSS: How to add it in the HTML file

There are **many ways** to **add CSS rules** to your **website**:

- Inserting the code inside a style tag (in the Head part). **Do not do that!!**

```
<style>
   p { color: blue; }
</style>
```

- **Referencing** an external CSS file, i.e., a file named "style.css" (in the Head part). **It is better to use this approach!!!**

```
<link rel="stylesheet" href="CSS/style.css"/>
```

- Using the **attribute style** on a **tag! Can do but NOT recommended!!**

```
<p style="color: blue; margin: 10px">A Paragraph.</p>
```

CSS Selectors

- **CSS selectors** is the **first part** of a **CSS Ruleset** that **select the elements** that need to be styled.
- Selectors **find elements based** on their **id**, **class**, **type**, **attribute**, and more.

- There are 6 CSS **selector** types:
  - CSS **class** selectors
  - CSS **id** selectors
  - CSS **element** selectors
  - CSS **attribute** selectors
  - CSS **pseudo-class** selectors
  - CSS **global** selectors

A complete list of CSS type selectors and the details of how each can be used is provided here.

CSS Selectors

- Let's start by changing the background color of **div** type tags of our website:

```
div {
    background-color: red;
}
```

- This CSS rule means that every tag **div** element found in our website should have a red background color. Note that **divs** are used mostly to **represent areas** of **our website**.

- We could also **change** the **whole website background** by affecting the tag **body**:

```
body {
    background-color: red;
}
```

CSS Selectors

- But what if we want to **change one specific tag or some tags** of the **same type**.

- We can specify **more precise selectors besides the name of the tag**. For instance, by **class** or **id**. To specify a tag with a given class name, we use the **dot**:

```
p.intro {
    color: red;
}
```

- This will affect **only the** p **tags** with class name "intro" in the **HTML file**:

```
<p class="intro">This is the intro.</p>
```

CSS Selectors

There are **several selectors** we can use to **narrow our rules** to very specific tags of our website. The main selectors are:

- **tag name**: just the name of the tag

  p { ... } /* affects to all <p> tags */
- **dot (.):** followed by the name of the **class**

  p.highlight { ... } /* affects all <p> tags with **class**="highlight" */
- **hash character (#):** followed by an **id**

  p#intro { ... } /* affects the <p> tag with the **id**="intro" */
- **two dots (:):** followed by **behaviour states** (mouse on top)

  p:hover { ... } /* affects <p> tags when the **mouse over** */
- **brackets ([attr='value']):** tags with the **attribute attr** with the **value 'value'**

  input[type="text"] { ... } /* affects all **input** tags of **type text** */

CSS Selectors

You can also **specify tags** by its **context**, for example: **tags** that are **inside of tags matching a selector**. Just **separate** the selectors by a **space**:

```
div#main   p.intro { ... }
```

This will affect the p tags of class intro that are inside the tag div of id main

```
<div id="main">
   <p class="intro">....</p>  ← Affects this one
</div>

<p class="intro">....</p>     ← But NOT this one
```

CSS Selectors

And you can **combine selectors** to narrow it down more.

```
div#main.intro:hover { ... }
```

This will apply the CSS to any tag div with id main and class intro when the mouse is over it.

Also, you **do not need to specify a tag**, you can use the **class** or **id** selectors **without tag.**

```
#main { ... }  // This means it will affect any node of id main.

.btns { ... }  // This means it will affect any node of class btns.
```

CSS Selectors

Finally, if you want to use the same CSS **property declarations** to several selectors, you can use the comma , character:

```
div, p { ... }  /* This means it will be applied to all divs and p tags */
```

We will see more about CSS in a future Lecture!!!

Front-End Technologies

- HTML
- CSS
- **Javascript**

Javascript (JS)

- A light-weight **interpreted** programming language with **object-oriented capabilities**, **that adds interactivity to your website** → Syntax is similar to C or Java but with **no types**.
- JavaScript is the only programming language **native to the web browser (**i.e., you **do not need to install** any specific software to run JavaScript**)**!!!
- **Client-Side JavaScript allows interaction with the user**, **control the browser** and **dynamically "create" HTML content**.
- You can **change the content of the HTML** or the **CSS** applied to an element **without reloading the page**.

Javascript: How to add it in the HTML file

There are **three ways** to **execute javascript** code in a website:

- **Embed** the code in the HTML document using the <script> tag.

```
<script> /* some code */ </script>
```

- **Import** a Javascript file using the <script> tag in the <head> part:

```
<script src="JS/code.js" defer></script>
```

- **Inject** the code on **an event inside a tag (e.g., a button)**:

```
<button id="btn1" onclick="myFunction()">Click Me!</button>
```

Javascript: Syntax

Very similar to C++ or Java but **much simpler (no type declaration needed)**.

```
var my_number = 10;          // this is a comment
let my_string = "hello";     /* this is also a comment */
const my_array = [10,20,"name",true];
const my_object = { name: "javi", city: "Barcelona" };

function myFunction( str ){
    for(var i = 0; i < 10; ++i)
        console.log(" String: " + str );
}
```

Example 1: Embed Javascript in HTML code using the <script> tag

```
<!DOCTYPE HTML>
<html>

<head>
    <title>Page Title</title>
</head>

<body>
    <h1 id="title">This is a title</h1>
    <button id="btn1">Click Me!</button>

    <script>
        var element = document.getElementById("btn1");
        element.addEventListener("click", function () {changeTitle ("This is a new Title!")})

        function changeTitle(newTitle){
            document.getElementById("title").innerHTML = newTitle;
        }
    </script>

</body>

</html>
```

Example 2: Import a Javascript file using the <script> tag in the <head> part

```html
<!DOCTYPE HTML>
<html>

<head>
    <title>Page Title</title>
    <script src="JS/jsCode.js" defer></script>
</head>

<body>
    <h1 id="title">This is a title</h1>
    <button id="btn1">Click Me!</button>
</body>

</html>
```

Here is very important to include keyword defer. With this keyword the jsCode.js file is loaded after all the elements of the web page are loaded first.

```javascript
var element = document.getElementById("btn1");
element.addEventListener("click", function () { changeTitle("This is a new Title!") });

function changeTitle(newTitle) {
    document.getElementById("title").innerHTML = newTitle;
}
```

jsCode.js in JS folder

Example 3: Inject the code on an event inside a tag

Always use defer!!!

```html
<!DOCTYPE HTML>
<html>

<head>
    <title>Page Title</title>
    <script src="JS/jsCode.js" defer></script>
</head>

<body>
    <h1 id="title">This is a title</h1>
    <button id="btn1" onclick="changeTitle('This is the new Title')">Click Me!</button>
</body>

</html>
```

```js
function changeTitle(newTitle) {
    document.getElementById("title").innerHTML = newTitle;
}
```

jsCode.js in JS folder

The result will be the same with all approaches!!!

Before the button is Clicked

After the button is Clicked

Javascript API

Javascript comes with a rich **API** to do **many things** like:

- **Access** the **DOM** (HTML nodes)
- **Do HTTP Requests (GET, POST, etc.)**
- **Play videos** and **sounds**
- **Detect user actions** (mouse move, mouse over, key pressed)
- **And many more….**

And the **API keeps growing** with every new update of the standard.

Check the WEB API reference to learn more!

Javascript: Find an element (node)

You can **find elements** from the DOM (HTML tree) using **different approaches, like:**

- **Crawling the HTML tree:** Starting from the body, and traversing its children.
- **Using a selector method:** Like **getElementById**(…), **querySelectorAll**(…), etc.

Javascript: Crawling the DOM

From javascript you have different ***objects*** that you can access to get information about the website:

- **document**: the DOM information (HTML node tree)
- **window**: the **browser** window

The **document object** allows to **crawl the tree**:

Note1: The **document** object **represents your web page**. If you want to **access any element** in an **HTML page**, you **always start with accessing** the **document object**.

```
// returns the first node inside body tag
document.body.children[0]
```

Javascript: Using Selector methods

You can **retrieve** an **element by its id**:

```
var mynode = document.getElementById("button1");
```

This will return a node with `id="button1"`.

You can retrieve **a number of elements using selectors**:

```
var nodes = document.querySelectorAll("p.intro");
```

This will return an **array** like collection with all `<p class="intro">` nodes in the web.

Or if we already **got a node** (i.e., mynode) and we want to **search inside**:

```
var node = mynode.querySelectorAll("p.intro")
```

Javascript: Modify nodes

Using JavaScript you can **change the attributes of an HTML Element**

```
var mynode = document.getElementById("par1");
mynode.id = "intro";  //sets a different id
mynode.className = "important"; //set class
```

Change the content

```
mynode.innerHTML = "Paragraph Text to show"; //change content
```

Modify the style (CSS properties)

```
mynode.style.color = "red"; //change color to red
```

Add behaviour to a node

```
mynode.addEventListener("mouseover", function() {
     //JavaScript Code to be executed when mouseover
} );
```

Javascript: Create or Delete Elements

Using JavaScript you can **Create** an element**:**

```
var element = document.createElement("div");
```

Attach an element anywhere **to the DOM**:

```
document.querySelector("#main").appendChild( element );
```

Remove an element **from its parent**:

```
var element = document.querySelector("foo");
element.parentNode.removeChild( element );
```

Clone an element:

```
var cloned = element.cloneNode(true);
```

Javascript: Hide and Show Elements

- Sometimes it may be useful to **hide** one **element** or **show** another.
- To **avoid being displayed** on the web, change **display style** property to **"none"**.

```
//hides elements from being rendered
element.style.display = "none";

//displays it again
element.style.display = "";
```

Using Inputs from User

To ask the user for input we use the **prompt()** function. We store user input in a **variable** so that we can use the information in our program.

```
// Gets user input
var onoma = prompt("What is your name?");
var num = prompt("What is your favorite number? ");
```

Installing Basic Software for Web Development

- To do simple web development we will need:
  - **A text Editor:** We will use Visual Studio Code, which is a free editor, that offers live previews and code hints.
  - **A web browser:** I recommend installing both Firefox and Chrome and have them ready for testing.
  - **A local Web Server:** Some examples will need to be run by a **web server** to work successfully. One of the easiest ways to do this for our purposes is to use **Python's http.server** module.

Installing Basic Software for Web Development
Running a simple local web server

Step 1: Install Python. You can get an installer from the **Python homepage** and follow the instructions to install it:

- Go to **https://www.python.org/downloads/** and download and install the latest version for Python "3.xxx".
- On the first installer page, **make sure** you check the "Add Python 3.xxx to PATH" checkbox.

Installing Basic Software for Web Development
Installing a local web server

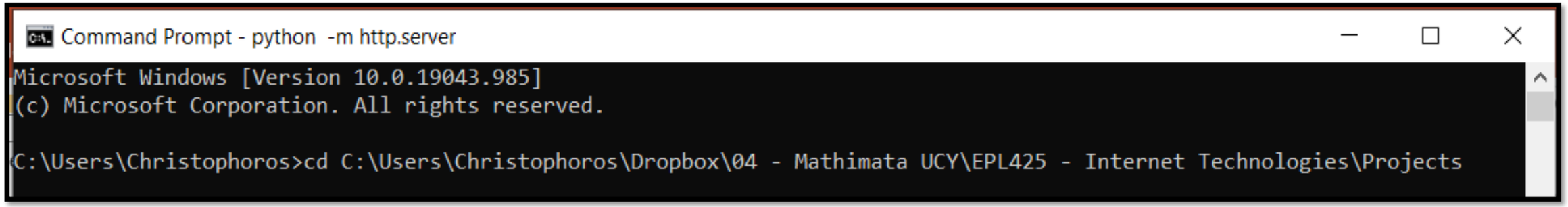
Step 2: Open your command prompt (Windows) / Terminal (macOS/ Linux). To check if Python is installed, enter the following command (This will display on the screen a version number):

```
python -V
# If the above fails, try:
python3 -V
# Or, if the "py" command is available, try:
py -V
```

Installing Basic Software for Web Development
Installing a local web server

Step 3: If everything is ok, navigate to the directory (enter the path) **that your example is inside**, using the cd command.

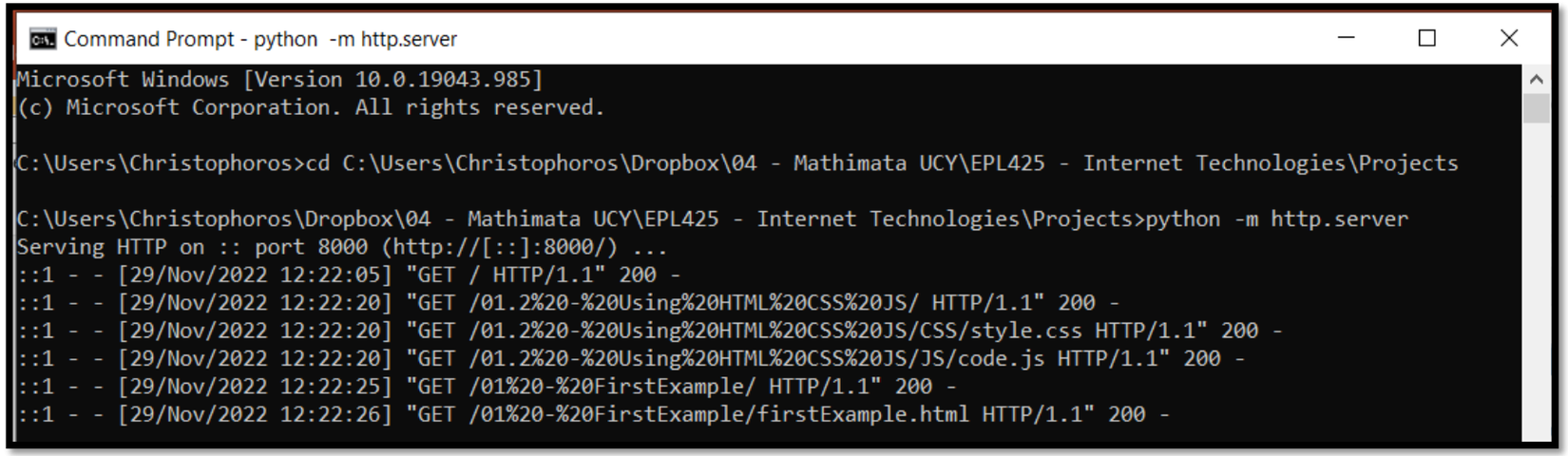
```
# include the path to the directory you want access, for example
cd C:\Users\Christophoros\Dropbox\04 – Mathimata UCY\ ……
```



Installing Basic Software for Web Development
Installing a local web server

Step 4: Enter the command to **start up the server** in that directory:

```
# If Python version returned above is 3.X
# On Windows, try
python -m http.server
# if the above fails try
py -3 -m http.server
# or
python3 -m http.server
```



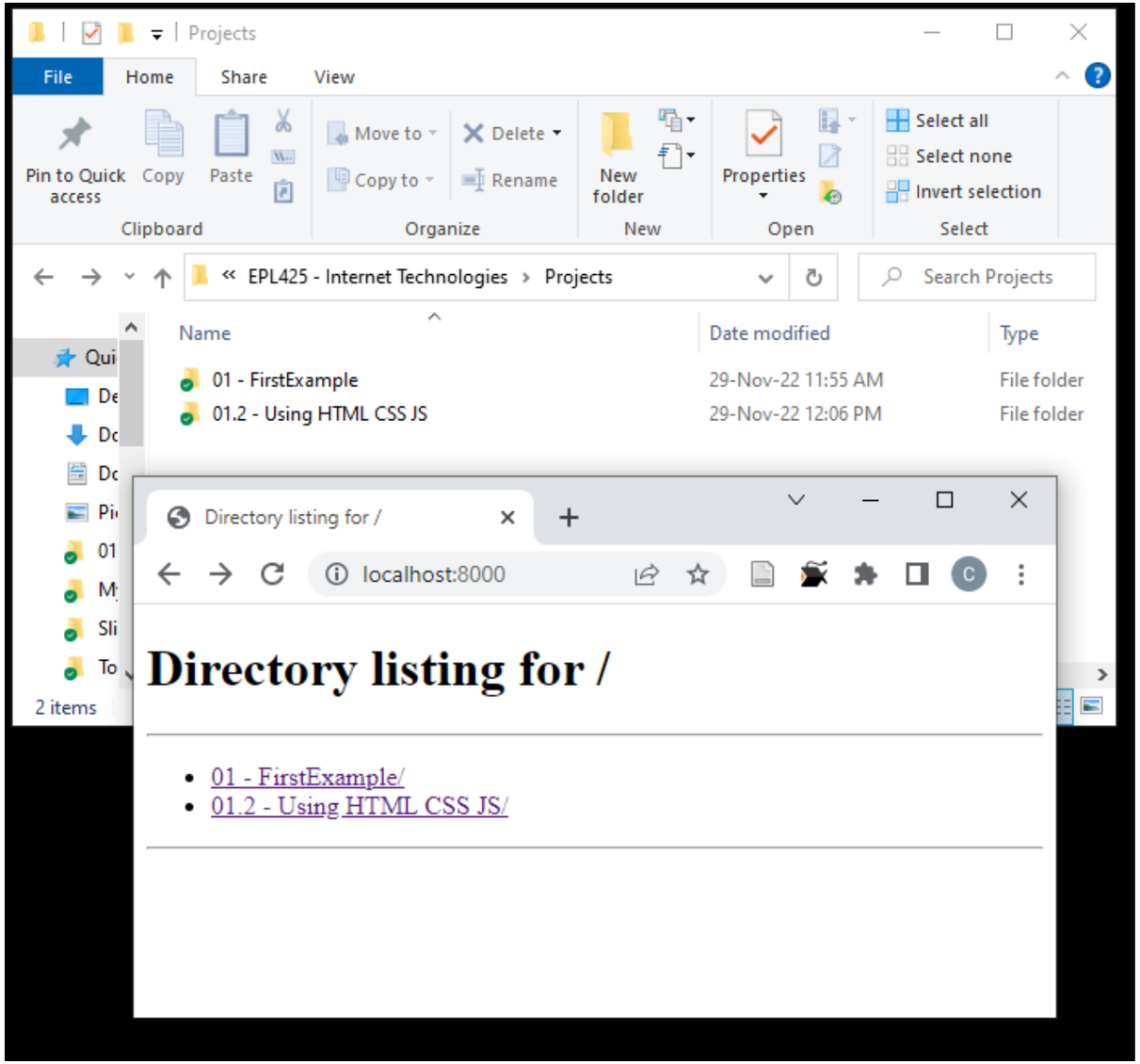
***Note**: If you already have something running on **port 8000**, you can choose another port by running the server command followed by an alternative port number, e.g., **python3 -m http.server 7800**. You can then access your content at **localhost:7800**.*

Installing Basic Software for Web Development
Installing a local web server

Step 5: By default, this will run the contents of the directory on a local web server, on **port 8000**.

You can "go" to this server by navigating to the URL: **localhost:8000** in your web browser.

Here you'll see the contents of the **directory listed** — click the HTML file you want to run.



Installing Basic Software for Web Development
Running server-side languages locally

- Python's **http.server** module is useful, but it is merely a **static file server**; it **doesn't know** how to run code written in **server-side** languages like **Python, PHP or Node.js**.

- To handle them, you'll need something more which **depends** on the **server-side language** you are **trying to run**. Here are a **few examples**:
  - To run **PHP** server-side code, launch PHP's built-in development server

```
# Path to your php code
cd C:\……
php –S localhost:8000
```

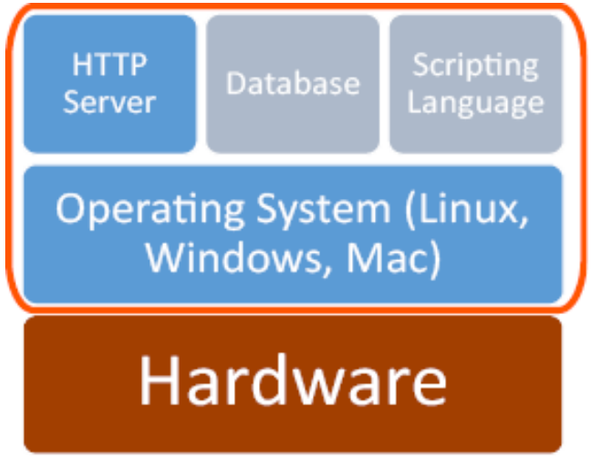
Installing Basic Software for Web Development
Running server-side languages locally

- To run **Python server-side code**, you'll need to use a **Python web framework**. There are many popular Python web frameworks, such as **Django** (a guide is available), Flask, and Pyramid.

- To run **Node.js** (JavaScript) **server-side code**, you'll need to use raw node or a framework built on top of it. **Express** is a good choice — see Express Web Framework (Node.js/JavaScript).

Setting up a real Web Server

- A **typical web server** today contains **four elements**. These are the:
  - **Operating system**,
  - **Web server**,
  - **Database**
  - **Scripting language**.

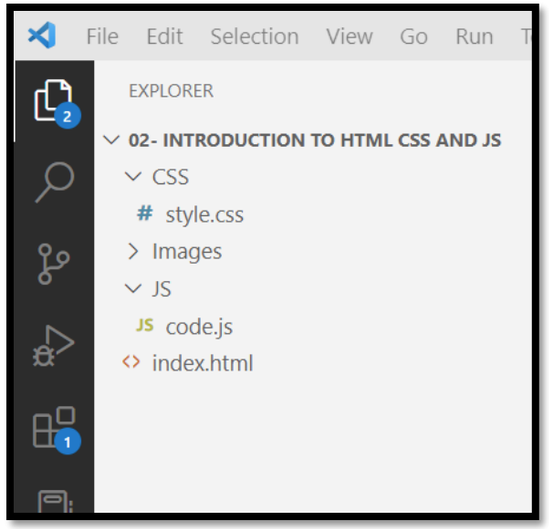


- One of the **most popular combinations** of these systems has been abbreviated to **LAMP for Linux**, **Apache**, **MySQL**, and **PHP**
- Other combinations of solutions are:
  - **WAMP** for **W**indows, **A**pache, **M**ySQL, **P**HP
  - **MAMP** for **M**ac, **A**pache, **M**ySQL, **P**HP

You will see how to set up a real web server during the Labs!

What structure should your website have

- The **most common things** we'll have on any website project we create are an **index.html** file and **folders** to contain **images**, **style files**, and **script files**. Let's create these now:
  - **index.html**: This file will generally contain your homepage content, that is, the text and images that people see when they first go to your site. Using your text editor, create a new file called **index.html** and save it just inside your test-site folder.
  - **images folder**: This folder will contain all the images that you use on your site. Create a folder called images, inside your test-site folder.
  - **styles folder**: This folder will contain the CSS code used to style your content (for example, setting text and background colors). Create a folder called styles, inside your test-site folder.
  - **scripts folder**: This folder will contain all the JavaScript code used to add interactive functionality to your site (e.g., buttons that load data when clicked). Create a folder called scripts, inside your test-site folder.




```html
<!DOCTYPE HTML>
<html>
<head>
  <title>An Example using HTML CSS and JavaScript</title>
  <link rel="stylesheet" href="CSS/style.css" />
  <script src="JS/code.js" defer> </script>
</head>
<body>
  <h1 id="h1">An Example including HTML CSS and JavaScript</h1>
  <button id="btn1">Click Me!</button>
  <p id="greeting"></p>
</body>
</html>
```

```javascript
// First Fetch the button from the DOM
var button = document.getElementById("btn1");

// Attach an event when the user clicks it.
// When clicked myFunction will be invoked
button.addEventListener("click", myFunction);

// Create the function that will be called when the
// button is clicked.
function myFunction() {
    // Gets user input
    let onoma = prompt("What is your name?");
    let epitheto = prompt("What is your lastname?");

    // Hide the button from being redered
    button.style.display = "none";

    // Fetch the p node from the DOM that the greeting
    // message will be displayed and assiged to it the
    // greeting string to be dispayed in the browser.
    let par = document.getElementById("greeting");
    par.innerHTML = `Hello ${onoma} ${epitheto}!`;
}
```

```css
html{
    background-color: red;
}

body {
    background-color: orange;
    height: 100%;
    width: 50%;
}

h1 {
    width: 100%;
    text-align: center;
    font-size: 50px;
    color: white;
    background-color: black;
}

p {
    font-size: 40px;
    text-align: center;
    color: blue;
}

button {
    border: 2px solid grey;
    font-size: 20px;
    color: white;
    background-color: black;
    size: 20px;
    height: 50px;
    width: 100px;
}
```

Our first example using HTML, CSS and JavaScript!

Tip 1: The **code formatting** is available through the following shortcuts or key combinations:

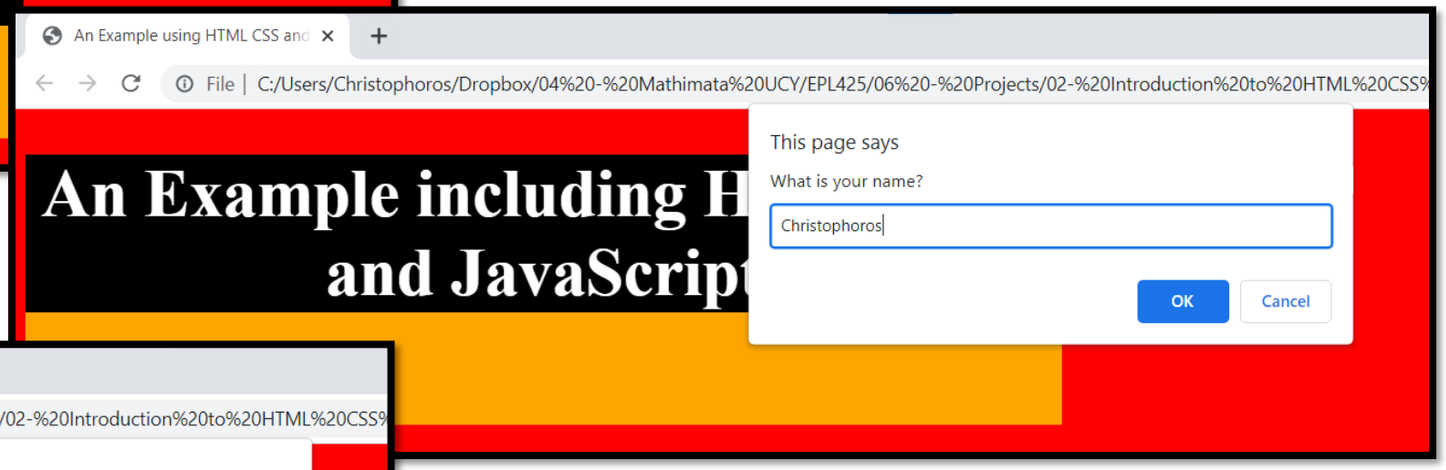
- On **Windows:** Shift + Alt + F
- On **macOS:** Shift + Option + F
- On **Linux:** Ctrl + Shift + I

Tip 2: You can select and **put a part** of your code **in comments** using: Ctrl + /

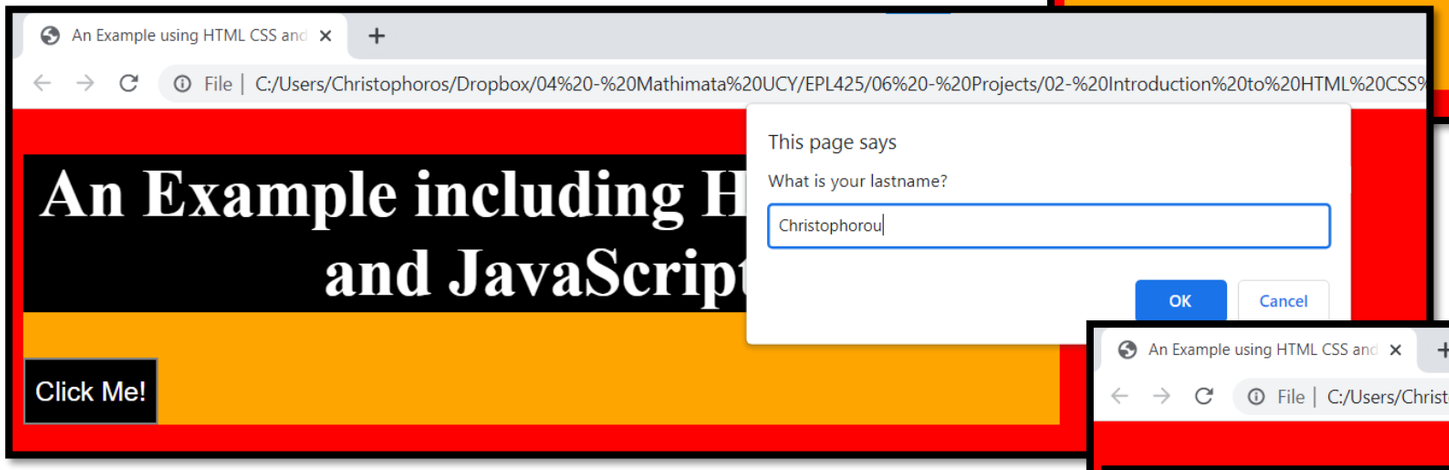
Running the Example

1

2



3



4

Publishing your Website

Once you finish with your website, you need to put it online so people can find it. Next are the steps that you have to follow!

Publishing your Website: Get a Domain Name

- Your **domain name** should be engaging and tell visitors **what your site is all about**.
- For example, if you're an established business, you'll want to locate your site at yourbrandname.com or a similar domain.
- For a **domain name search** major domain registrars like **GoDaddy**, **Namecheap**, **Bluehost**, and **HostGator**.

Publishing your Website: Get Web Hosting

- The **files** and **folders** that **make up your web site** take up space and need a place to live. Without an **online home**, your files would just sit on your computer and no one would ever see them.
- When you pay for **hosting services**, you are simply **renting storage space on the internet**!
- A **hosting provider** will provide a **place** on a **web server** to store all of your files and are **responsible** for **delivering the files** of your website as soon as **a browser makes a request** by typing in your domain name.

HOSTINGER HostGator DreamHost SiteGround

GreenGeeks WEB HOSTING bluehost GoDaddy A2 HOSTING DreamHost

Publishing your Website: Transfer files to the Web Server

- For this you will need a **File Transfer Protocol (FTP)** (e.g., FileZilla) program.
- FTP programs vary widely, but generally, you have to connect to your web server using details provided by your hosting company (typically **username**, **password**, **hostname**).
- Then the program shows you your **local files** and the **web server's files** in two windows, and provides a way for you to **transfer files back and forth**.

Ερωτήσεις?