FREE TO CODE

# Improving Digital and Coding Skills for Inmates

## *Training Content (Intellectual Output 2)*

2018-1-RO01-KA204-049298

Co-funded by the Erasmus+ Programme of the European Union

FREE TO CODE

Co-funded by the Erasmus+ Programme of the European Union

Project Number 2018-1-RO01-KA204-049298

**Programme**
**FREE TO CODE** - *Improving Digital and Coding Skills for Inmates*

**Partners**
European Strategies Consulting (Romania)
APROXIMAR – Cooperativa de Solidariedade Social, Crl (Portugal)
Asturia vzw (Belgium)
Avaca Technologies Consulting, Informatics AE (Greece)
BeCode (Belgium)
Dlearn – European Digital Learning Network (Italy)
Penitenciarul Bucaresti-Jilava (Romania)

**Design**
**Aproximar, Cooperativa de Solidariedade Social**

**Publication date**
February, 2020

**www.free2code-initiative.eu**

# /// Index

# a } General Introduction

## General Intro to the Course


### *Hello World!*

Since the computers and the Internet appeared, they have changed deeply how we play, how we communicate, how we work, how we live. Because of these deep changes in our Society, there are many jobs available for people who have learned to code and solve problems by coding websites and web applications.

Perhaps you think you need a University degree in Mathematics and nuclear engineering to be able to code? Not at all! Coding is easy once you "get it" : even kids can learn programming... Because it actually is a lot of fun! If you ask programmers what they do every day, they will say things like: "it's like playing with Lego Bricks" or "It's like cooking, writing recipes...".

### *What you can expect*

The objective of this training is to allow you to have a try at programming and logical thinking, to see if you too find it fun and something you would like to pursue once you get out of jail.

This course will have you learn the basics of website making and programming using the 3 main languages of the web: HTML, CSS, and JavaScript. By the end of the training, you will be more than ready to join a coding bootcamp that will help you reach a level in which you can get a real job as a programmer!

### *Learn by doing*

The best way to learn coding is just like cooking... By doing it! This is why the course is organized as a series of exercises that you need to succeed in order to move on to the next step. Don't worry about being fast, it's not like school! The system remembers your progress so you don't have to start over from one session to the next.



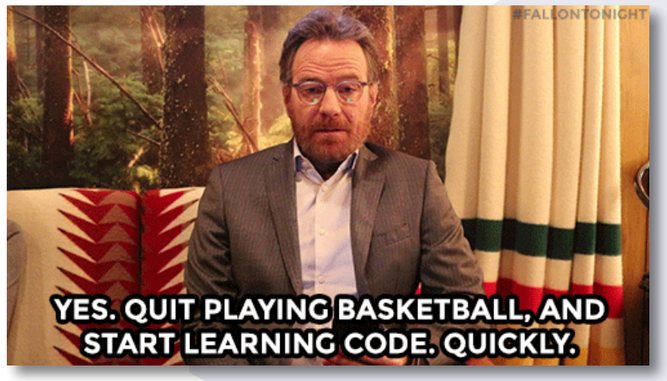


### *... It doesn't work! I suck at this!!*

No, you don't. Coding is about trying until it works. So we all start by failing! Get used to it, failure is part of the learning process. Just keep trying, you will make it!

### *A final word from the team*


*Ready ? Let's go!*



# b ( Module 0

# Internet and Web Info

## Background section

### *@ The internet*

### *_Birth of the internet*

**Tim Berners-Lee**, a British scientist, invented the World Wide Web (WWW) in 1989, while working at CERN. The web was originally conceived and developed to meet the demand for automated information-sharing between scientists in universities and institutes around the world.

On 30 April 1993, CERN put the World Wide Web software in the public domain. Later, CERN made a release available with an open licence, a more reliable way to maximise its dissemination. These decisions allowed the web to flourish.

### *_Interface of the Internet*

With the creation of the web, there was also the need for a way to consult the files and resources made accessible. Sir Tim Berners-Lee therefore created a specific software, called the WorldWideWeb browser.

WorldWideWeb

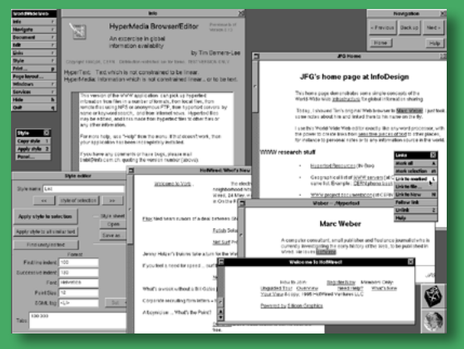



Figure 1 - Screen capture of the first browser developed by Sir Tim Berners-Lee

The concept of **hypertext** preceded the World Wide Web by decades. But nearly all hypertext systems worked on local files. Tim Berners-Lee wanted to create a system that would work across networks so that people could link from a file on one machine to another file on another machine.

WorldWideWeb wasn't just a programme for browsing files. It was a browser and editor. The introductory text reads:

*“HyperMedia Browser/Editor, An exercise in global information availability by Tim Berners-Lee”*





At its heart, WorldWideWeb is a word processor... but with hyperlinks. And just as you can use a word processor purely for reading documents, the real fun comes when you write your own. Especially when you throw hyperlinks into the mix!

There was one major downside to the WorldWideWeb browser: it could only be used on a NeXT computer... Almost nobody had a NeXT machine.

### *_Line Mode browser*

To make the Web more widely accessible, a second browser project was developed at CERN: the Line Mode browser. The Line Mode browser was first released in 1991 and was compatible with most unix / linux systems. Thereby instantly bring the Web to commonly used, much lower powered devices, such as the "line mode" terminals that were used to access minicomputers, still common at the time.


Figure 2 - Screen capture of the Line Mode browser

## _Language of the Internet

HyperText Markup Language

HTML (Hypertext Markup Language) is the code that is used to structure a web page and its content. For example, content could be structured within a set of paragraphs, a list of bulleted points, or using images and data tables.

HTML consists of a series of elements, which you use to enclose, or wrap, different parts of the content to make it appear a certain way, or act a certain way. The enclosing tags can make a word or image hyperlink to somewhere else, can italicize words, can make the font bigger or smaller, and so on.

```
<p>This is a paragraph, because it is enclosed in a "p" html tag.</p>

<p>This is another paragraph, also enclosed in a "p" html tag.</p>
```

# Internet

## @ How does the internet work

### _In short

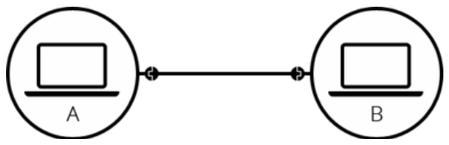
The Internet is the backbone of the Web, the technical infrastructure that makes the Web possible. At its most basic, the Internet is a large network of computers which communicate all together.

The history of the Internet is somewhat obscure (you can read more about it here). It began in the 1960s as a US-army-funded research project, then evolved into a public infrastructure in the 1980s with the support of many public universities and private companies. The various technologies that support the Internet have evolved over time, but the way it works hasn't changed that much: Internet is a way to connect computers all together and ensure that, whatever happens, they find a way to stay connected.

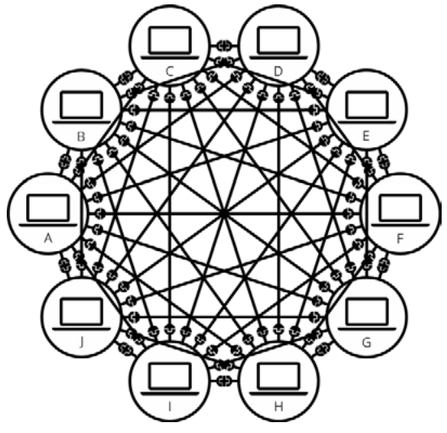
### _Deeper Dive

**A simple network**

When two computers need to communicate, you have to link them, either physically (usually with an Ethernet cable) or wirelessly (for example with WiFi or Bluetooth systems). All modern computers can sustain any of those connections.

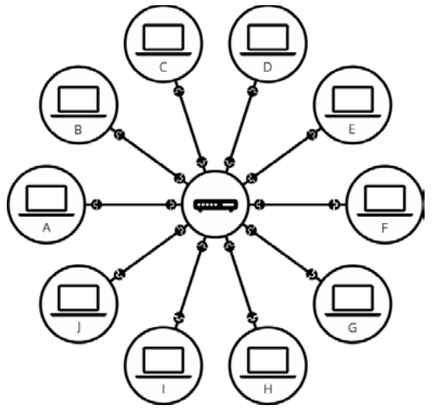



Such a network is not limited to two computers. You can connect as many computers as you wish. But it gets complicated quickly. If you're trying to connect, say, ten computers, you need 45 cables, with nine plugs per computer!



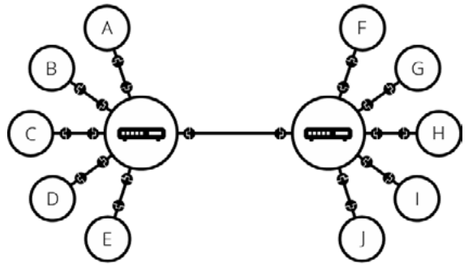

To solve this problem, each computer on a network is connected to a special tiny computer called a router. This router has only one job: like a signaler at a railway station, it makes sure that a message sent from a given computer arrives at the right destination computer. To send a message to computer B, computer A must send the message to the router, which in turn forwards the message to computer B and makes sure the message is not delivered to computer C.

Once we add a router to the system, our network of 10 computers only requires 10 cables: a single plug for each computer and a router with 10 plugs.

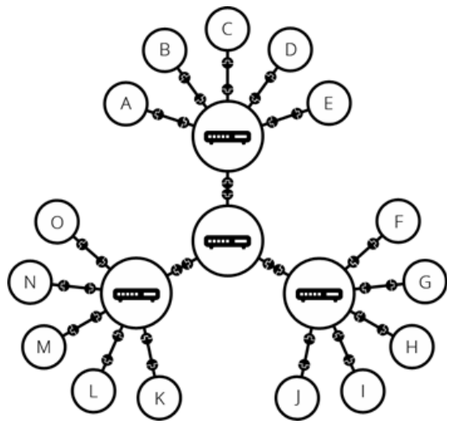



**A network of networks**

So far so good. But what about connecting hundreds, thousands, billions of computers? Of course a single router can't scale that far, but, if you read carefully, we said that a router is a computer like any other, so what keeps us from connecting two routers together? Nothing, so let's do that.

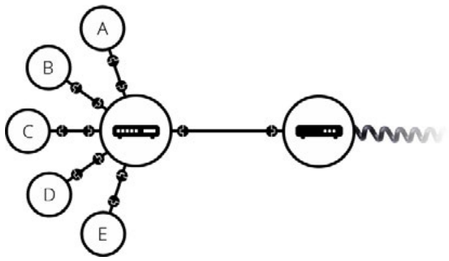



By connecting computers to routers, then routers to routers, we are able to scale infinitely.

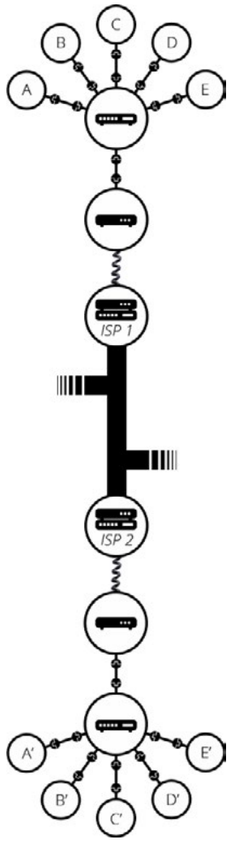



Such a network comes very close to what we call the Internet, but we're missing something. We built that network for our own purposes. There are other networks out there: your friends, your neighbors, anyone can have their own network of computers. But it's not really possible to set

cables up between your house and the rest of the world, so how can you handle this? Well, there are already cables linked to your house, for example, electric power and telephone. The telephone infrastructure already connects your house with anyone in the world so it is the perfect wire we need. To connect our network to the telephone infrastructure, we need a special piece of equipment called a modem. This modem turns the information from our network into information manageable by the telephone infrastructure and vice versa.




So we are connected to the telephone infrastructure. The next step is to send the messages from our network to the network we want to reach. To do that, we will connect our network to an Internet Service Provider (ISP). An ISP is a company that manages some special routers that are all linked together and can also access other ISPs' routers. So the message from our network is carried through the network of ISP networks to the destination network. The Internet consists of this whole infrastructure of networks.




**Finding computers**

If you want to send a message to a computer, you have to specify which one. Thus any computer linked to a network has a unique address that identifies it, called an "IP address" (where IP stands for Internet Protocol). It's an address made of a series of four numbers separated by dots, for example: 192.168.2.10.

That's perfectly fine for computers, but we human beings have a hard time remembering that sort of address. To make things easier, we can alias an IP address with a human readable name called a domain name. For example (at the time of writing; IP addresses can change) google.com is the domain name used on top of the IP address 173.194.121.32. So using the domain name is the easiest way for us to reach a computer over the Internet.

**Internet and the web**

As you might notice, when we browse the Web with a Web browser, we usually use the domain name to reach a website. Does that mean the Internet and the Web are the same thing? It's not that simple. As we saw, the Internet is a technical infrastructure which allows billions of computers to be connected all together. Among those computers, some computers (called Web servers) can send messages intelligible to web browsers. The Internet is an infrastructure, whereas the Web is a service built on top of the infrastructure. It is worth noting there are several other services built on top of the Internet, such as email and IRC.

# Web

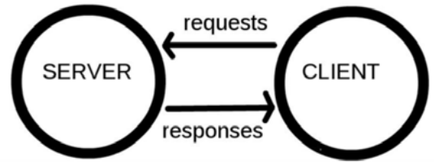
## @ How the Web works

How the web works provides a simplified view of what happens when you view a webpage in a web browser on your computer or phone.

This theory is not essential to writing web code in the short term, but before long you'll really start to benefit from understanding what's happening in the background.

### _Clients and servers

Computers connected to the web are called clients and servers. A simplified diagram of how they interact might look like this:




Clients are the typical web user's internet-connected devices (for example, your computer connected to your Wi-Fi, or your phone connected to your mobile network) and web-accessing software available on those devices (usually a web browser like Firefox or Chrome).

Servers are computers that store webpages, sites, or apps. When a client device wants to access a webpage, a copy of the webpage is downloaded from the server onto the client machine to be displayed in the user's web browser.

### _Other parts of the toolbox

The client and server we've described above don't tell the whole story. There are many other parts involved, and we'll describe them below.

For now, let's imagine that the web is a road. On one end of the road is the client, which is like your house. On the other end of the road is the server, which is a shop you want to buy something from.

In addition to the client and the server, we also need to say hello to:

- **Your internet connection:** Allows you to send and receive data on the web. It's basically like the street between your house and the shop.
- **TCP/IP:** Transmission Control Protocol and Internet Protocol are communication protocols that define how data should travel across the web. This is like the transport mechanisms that let you place an order, go to the shop, and buy your goods. In our example, this is like a car or a bike (or however else you might get around).
- **DNS:** Domain Name Servers are like an address book for websites. When you type a web address in your browser, the browser looks at the DNS to find the website's real address before it can retrieve the website. The browser needs to find out which server the website lives on, so it can send HTTP messages to the right place (see below). This is like looking up the address of the shop so you can access it.
- **HTTP:** Hypertext Transfer Protocol is an application protocol that defines a language for clients and servers to speak to each other. This is like the language you use to order your goods.
- **Component files:** A website is made up of many different files, which are like the different parts of the goods you buy from the shop. These files come in two main types:
- **Code files:** Websites are built primarily from HTML, CSS, and JavaScript, though you'll meet other technologies a bit later.
- **Assets:** This is a collective name for all the other stuff that makes up a website, such as images, music, video, Word documents, and PDFs.

## _So what happens, exactly?

When you type a web address into your browser (for our analogy that's like walking to the shop):

1. The browser goes to the DNS server, and finds the real address of the server that the website lives on (you find the address of the shop).
2. The browser sends an HTTP request message to the server, asking it to send a copy of the website to the client (you go to the shop and order your goods). This message, and all other data sent between the client and the server, is sent across your internet connection using TCP/IP.
3. If the server approves the client's request, the server sends the client a "200 OK" message, which means "Of course you can look at that website! Here it is", and then starts sending the website's files to the browser as a series of small chunks called data packets (the shop gives you your goods, and you bring them back to your house).
4. The browser assembles the small chunks into a complete website and displays it to you (the goods arrive at your door — new shiny stuff, awesome!).

## _DNS explained

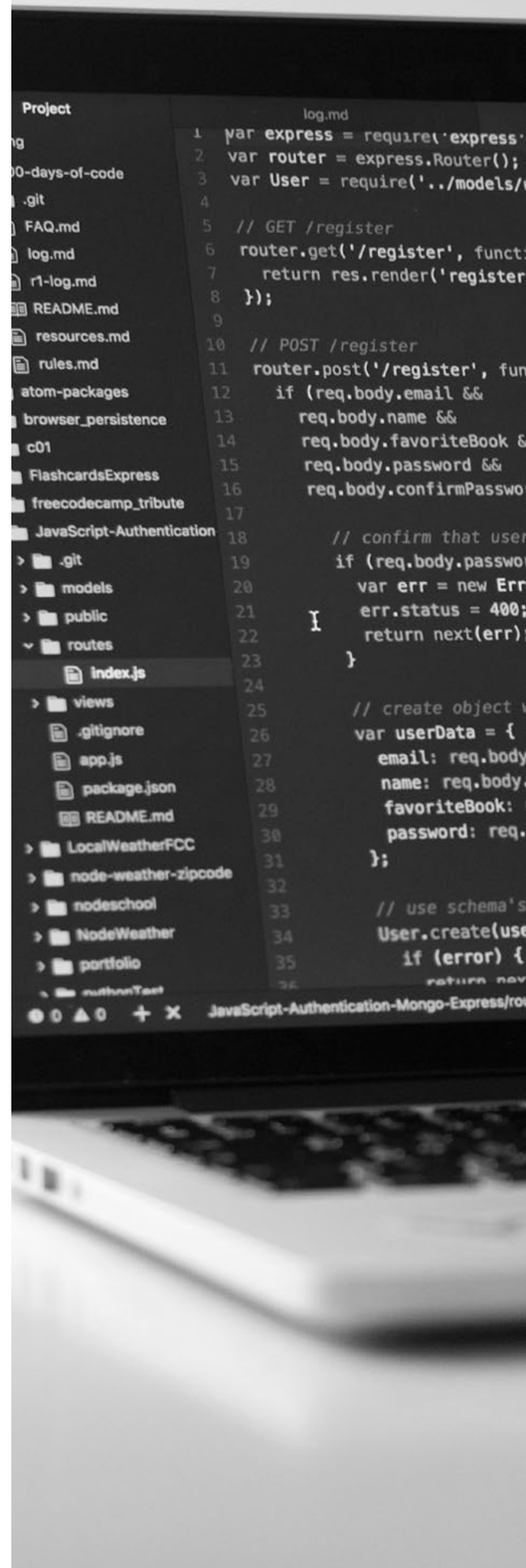
Real web addresses aren't the nice, memorable strings you type into your address bar to find your favorite websites. They are special numbers that look like this: 63.245.215.20.

This is called an IP address, and it represents a unique location on the web. However, it's not very easy to remember, is it? That's why Domain Name Servers were invented. These are special servers that match up a web address you type into your browser (like "mozilla.org") to the website's real (IP) address.

Websites can be reached directly via their IP addresses. You can find the IP address of a website by typing its domain into a tool like IP Checker.

## _Packets explained

Earlier we used the term "packets" to describe the format in which the data is sent from server to client. What do we mean here? Basically, when data is sent across the web, it is sent as thousands of small chunks, so that many different web users can download the same website at the same time. If websites were sent as single big chunks, only one user could download one at a time, which obviously would make the web very inefficient and not much fun to use.



# c > Module 1

# HTML

## Introduction to HTML

### @ Why use HTML

Computers and by extension web browsers are inherently dumb. We can't expect a computer to read a text document an make a distinction between titles, paragraphs, links, lists, etc

Let's put ourselves in the shoes of a computer and take a look at a document you most likely won't understand:


Just like we can't give **meaning** to the text in the above example, computers can't give meaning to the data they get from the web. Therefore we use a **markup language** to add additional meaning to the text. On the web, the markup language we use is HTML or **hypertext markup language**.


By adding some <tags> to the document that describe the content, we have a better understanding of the meaning and structure of the text. The area of linguistics that is concerned with the meaning of text is called semantics. When writing HTML it is important





to use the correct HTML tags for your content. This is called writing semantic HTML, and it's importance can't be understated.

### _What is HTML?

HTML is not a programming language; it is a markup language that defines the structure and meaning of your content. HTML consists of a series of elements, which you use to enclose, or wrap, different parts of the content to make it appear a certain way, or act a certain way. The enclosing tags can make a word or image hyperlink to somewhere else, can italicize words, can make the font bigger or smaller, and so on. For example, take the following line of content:

```
My cat is very grumpy.
```

If we wanted the line to stand by itself, we could specify that it is a paragraph by enclosing it in paragraph tags:

```
<p>My cat is very grumpy</p>
```

### _Anatomy of an HTML element

Let's explore this paragraph element a bit further.


The main parts of our element are as follows:

1. **The opening tag:** This consists of the name of the element (in this case, p), wrapped in opening and closing angle brackets. This states where the element begins or starts to take effect — in this case where the paragraph begins.

2. **The closing tag:** This is the same as the opening tag, except that it includes a forward slash before the element name. This states where the element ends — in this case where the paragraph ends. Failing to add a closing tag is one of the standard beginner errors and can lead to strange results.

3. **The content:** This is the content of the element, which in this case, is just text.

4. **The element:** The opening tag, the closing tag and the content together comprise the element.

### _Nesting elements

HTML elements can be nested in other HTML elements. Let's continue with the HTML element from the previous example.

I would like to put emphasis on the word grumpy. To do this I will use the  element and wrap it around the word I want to emphasize.

```
<p>My cat is very <em>grumpy</em></p>
```

In this example the  element is **nested** in the <p> element.

***Important:** When nesting html elements, make sure to close the nested element before closing the parent element*

# HTML 5

## @ First HTML document

### _File Extension

Your computer knows what application to use to open a certain file by checking the extension of the file. Some examples are:

- Documents ending in **.docx** will be opened by **Microsoft Word**
- Documents ending in **.pdf** will be opened with a **pdf reader**
- A file ending in **.mp3** will be opened with a **music player**

When you write HTML code, you want your code to be read by the **browser**. Your computer will automatically open all files ending in **.html** in the browser. The browser will then convert the HTML code inside the document to a human-readable webpage.

If you want to view the pure HTML code, you will have to use a different type of application to open these documents: a **code editor**.

### _Code editors

You could use the default text editor that is installed on your computer to write your HTML files. On Windows for example there is Notepad. That would work perfectly fine as long as you don't write any errors in your HTML code and save it as a .html file.

There are also applications created specifically to write code. These applications have built-in functionalities such as code highlighting, indentation of your code, autocomplete, etc. which help a lot with writing code efficiently.

## @ File Structure

```
<!DOCTYPE html>
<html>
  <head>
    <title>The Most Simple HTML page in the world</title>
  </head>
  <body>
    <h1>The Most Simple HTML page in the world</h1>
    <p>It might just be this page.</p>
  </body>
</html>
```

Figure 3 - Example of a simple html document





Let's take a look at the above example of an HTML document.

- `<!DOCTYPE html>` The first line of a html document starts with the doctype declaration. This line tells the browser what version of html you are using.
- `<html>...</html>` All your html code will be placed inside the html element. The html element consists of an opening tag: <html>, and a closing tag: </html>.
- `<head>...</head><body>...</body>` Inside the html element we have a head and a body element. The head element contains information about the document. It will not be visible to the human consulting your web page. The body element contains the actual content that will be visible on the website.
- `<title>...</title>` The title element is part of the head. As the name suggests, it contains the title for the html page.
- `<h1>...</h1>` The h1 element is placed in the body of the document. It is one of the 6 section headers in html. (h1-h6). "h" stands for "headline".
- `<p>...</p>` The p element or paragraph element is used to display text.

# Semantics

## @ Why use semantics

Semantics are relied on everywhere around us — we rely on previous experience to tell us what the function of an everyday object is; when we see something, we know what its function will be. So, for example, we expect a red traffic light to mean "stop", and a green traffic light to mean "go". Things can get tricky very quickly if the wrong semantics are applied (Do any countries use red to mean "go"? I hope not.)

## @ Semantic HTML

The HTML5 specification includes several semantic elements to help organize documents. Semantic elements are specifically designed to communicate meaning to the browser, developer, reader, and any other technologies interpreting the document (e.g. voice assistants, screen readers, search engines...).

### _Section elements in HTML5

- **HTML Navigational Element** (`<nav>`) defines a section that contains navigation links that appear often on a site. You can have primary and secondary menus, but you never nest, or put a `<nav>` element inside a `<nav>` element.

- **HTML Article Element** (`<article>`) defines a piece of self-contained content. It does not refer to the main content alone and can be used for comments and widgets.
- **HTML Section Element** (`<section>`) defines a section of a document to indicate a related grouping of semantic meaning. Utilize the section element providing extra context from the parent element makes sense.
- **HTML Aside Element** (`<aside>`) defines a section that, though related to the main element, doesn't belong to the main flow, like an explanation box or an advertisement. It has its own outline, but doesn't belong to the main one.

## _Other Semantic HTML elements used in Sectioning

- **HTML Body Element** (`<body>`) defines all the content of a document. It contains all the content and HTML tags.
- **HTML Header Element** (`<header>`) defines a page which typically contains the logo, title, and navigation. The header can also be used in other semantic elements such as `<article>` or `<section>` — or section header, containing perhaps the section's heading, author name, etc. `<article>`, `<section>`, `<aside>`, and `<nav>` can have their own <header>. Despite its name, it is not necessarily positioned at the beginning of the page or section.
- **HTML Footer Element** (`<footer>`) defines a page footer which typically contains the copyright, legal notices and sometimes some links — or section footer, containing perhaps the section's publication date, license information, etc. `<article>`, `<section>`, `<aside>`, and `<nav>` can have their own `<footer>`. Despite its name, it is not necessarily positioned at the end of the page or section.

## _Example layout

The example below is a layout for the body of blog page. There is a <header> with an <h1> element and a <nav> element which contains the navigation links. On the same level as the <header> we have a <section> and <footer> element. The main <section> element has a <h2> title, 2 <articles> and an <aside>.

```
<body>
    <header>
      <nav>
        <ul>
          <li><a href="#">link</a></li>
          <li><a href="#">link</a></li>
          <li><a href="#">link</a></li>
        </ul>
      </nav>
      <h1>
        Page Title
      </h1>
    </header>
    <section>
      <h2>
        My Blog Posts
      </h2>
      <article>
        <header>
          <p>
            Article Title
          </p>
        </header>
        <p>
          content
        </p>
      </article>
      <article>
        <header>
          <p>
            Article Title
          </p>
        </header>
        <p>
          content
        </p>
      </article>
      <aside>
        <p>
          Author info
        </p>
      </aside>
    </section>

    <footer>
      Copyright Info
    </footer>
</body>
```

# d / Module 2

CSS

## Introduction to CSS

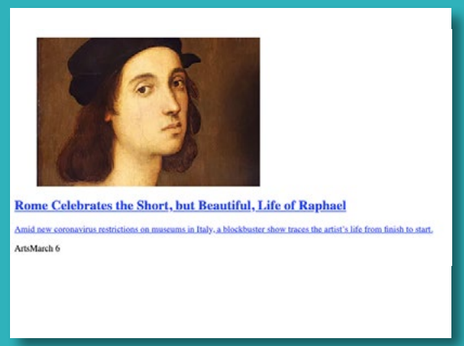
In our previous journey, we have discovered the power of HTML. As you've seen, HTML was created to describe the content of a web page, like:

```
<h1>This is a heading</h1>

<p>This is a paragraph.</p>
```

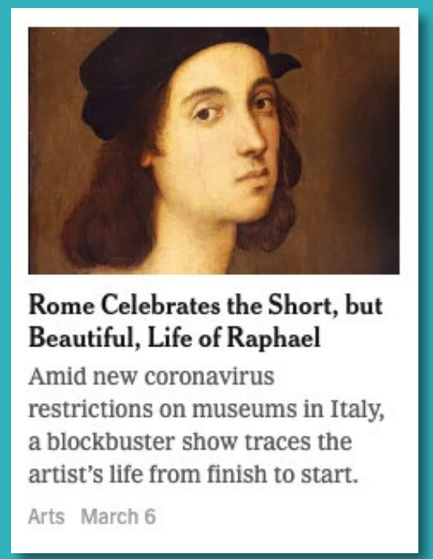
But you might have noticed that there is no color at all, it just looks like a plain word document. Which is boring to look at.

That's because on top of their HTML, these pages are using another language, called CSS, which job is to make the html look visually better.

In other words, CSS allows us to turn this:




Into that:




### @ What is CSS?

- CSS Stands for Cascading Style Sheets
- CSS describes how HTML Elements are to be displayed on screen.
- It can control the layout of multiple pages at once.

In short, CSS is what makes our web pages look good and presentable. It's a must-have skill for any web developer out there.

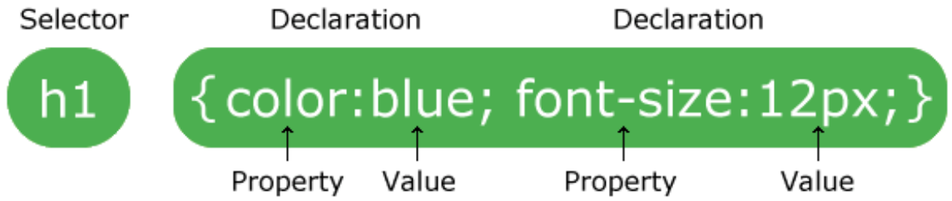




to use the correct HTML tags for your content. This is called writing semantic HTML, and it's importance can't be understated.

### @ The CSS Syntax

Look at this schematic, which sums all how to write CSS in a way that the browser understands it.




As you can see, there are fancy words here. Don't panick, there won't be much more :-)

**"selector"** indicates which element(s) of your HTML file should receive the instructions. It points to the HTML element you want to style. In this example, all `<h1>` tags on the html page will receive the CSS properties assigned to the `h1`css selector.

The declaration block contains one or more declarations separated by semicolons.

Each declaration includes a CSS **property name and a value**, separated by a colon (this character `:`).

A CSS declaration always ends with a semicolon ( `;`), and declaration blocks are surrounded by curly braces. Example:

```
h1 { color: blue; font-size:12px; }
```

### @ Getting started

There are three ways of implementing CSS in our web-pages.

## _1. Inline CSS

Firstly we can include CSS directly in our HTML elements. To accomplish this, we make use of the style attribute, and write CSS code inside its value (the part in between the quotes). Example:

```
<h1 style="color: blue"> Hello world! </h1>
```

As you see, the css instructs the browser to use the color blue to render the h1tag. Easy, isn't it ? And you can combine several properties. For example, let's make it blue and bold:

```
<h1 style="color:blue;font-weight:bold">Hello world!


</h1>
```


Easy again! As you can see, you can add up properties using a semicolon (the character ;) to separate them so the browser does not get confused.

We could add many more properties inside of this method but it can get really messy in our HTML file so it's not really recommended. The next two methods are much cleaner...

## _2. The <style> block

Another way to include CSS is by using the <style> tag inside of our head section of our HTML page.

```
<head>

  <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-
scale=1.0">

 <meta http-equiv="X-UA-Compatible" content="ie=edge">

  <title>CSS is awesome! - BeCode</title>

  <style> h1{  color:blue; }

  </style>

  </head>
```

We just found a solution to not mix our HTML with our CSS, but our styling is still inside of our HTML file.

Maybe, there is a better way to include our CSS..? Well, yes there is!

## _3. External CSS

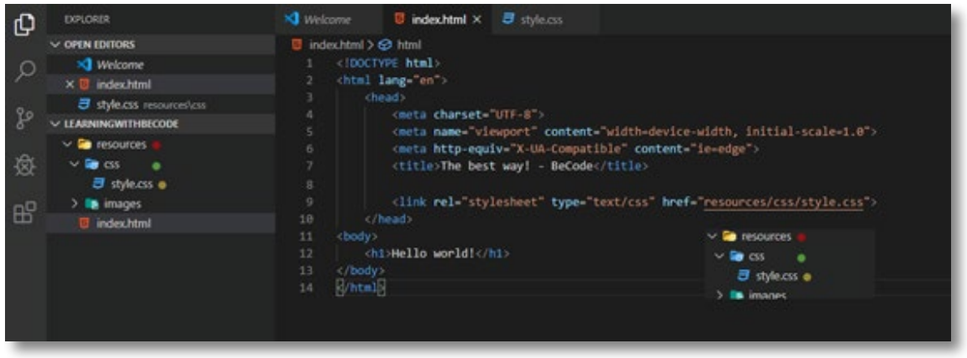
Like the name gives it away, we will have some **external** CSS file(s), which we will import inside the <head> of our HTML page.

```
<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-
scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>The best way! - BeCode</title>

<link rel="stylesheet" type="text/css" href="resources/css/style.
css">

</head>
```

As you can see, we use a <link> tag this time to make a connection with our CSS file. This link tag will need a few attributes to work, the rel=" stylesheet" specifies the relationship between the HTML and CSS file, the browser knows now that we are trying to link a Stylesheet (CSS file). The type=" text/CSS" will tell the browser what kind of resource we are linking. It's not an obligation to use this, but we recommend using it to avoid any problems in the future. Last but not least, the href="resources/css/style.css" is our path that the link will use to find the document.

*Having an external CSS file is the most recommended way to do, because it "separates concerns": the HTML file is for content, the CSS file is for decoration!*

***Here is an example of our folder structure:***




Inside of our CSS file we have written the following:

```
h1{color:blue;}
```

This will give the same output as our result in example 1, the benefit of this is that our CSS is separated from our HTML and we can import this CSS file in multiple pages at once!

## @ Working with colours

Colours are a big part of how things look. And we, humans, love colour !

There are many millions of colours available in Nature.... Which was quite a challenge to transfer in the digital world of computers. First, there were only a few colours available, using predefined colour names (like "red", "blue", "beige", "chocolate")...

### _Named colours

A set of standard color names have been defined, letting you use these keywords instead of numeric representations of colors if you choose to do so and there's a keyword representing the exact color you want to use. Color keywords include the standard primary and secondary colors (such as red, blue, or orange), shades of gray (from `black` to `white`, including colors like `darkgray` and `lightgrey`), and a variety of other blended colors including `lightseagreen`, `cornflowerblue`, and `rebeccapurple`.

It's nice to know they exist, but you will feel fastly limited by having only 140 colours.. So let's rather move on to the next way to express colour values...



### _The RGB system

Quickly computers became more powerful and able to manipulate millions of different colors, using a mix of the fundamental 3 colours of the screen: Red, Green and Blue, which is known as the "RGB system".

```
p{color: rgb(255, 0, 0); }
```

This says "I want the maximum of Red (maximum is 255), no green (0), and no blue (0)... Leading to a full bright red color.

This is exactly the same as

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

So if, for example, you really want that specific shade of blue that describes the Scottish sky in Spring, then you need to find its correct translation in the RGB system.

```
p{color: rgb(0, 182, 255);}
```

### _Transparency!

You can also use a fourth value, to set the "alpha", which means "transparency" (or "opacity" if you prefer). Its value goes from 0 (totally transparent, the tag would be invisible) to 1 (fully opaque). Instead of RGB we use RGBA to add the transparent layer to our colour.

So let's say you want an orange square with 60% transparency, you would do this:

```
div { width: 100px; height:
100px; background-color:

rgba(255, 221, 0,0.6); }
```

### _The Hexadecimal system

For your information, there is yet another way to express the colour values, using the Hexadecimal system. In that system, red for example is expressed as `#FF0000`, `blackis #000000 and white: #FFFFFF`.

Hexadecimal system functions from values going from 0 to 9 and continues to A up to F for a total of 16 values. By using 6 Hexadecimal values, you are able to express 256 millions of colours.

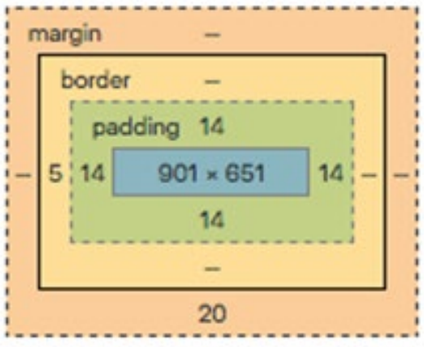
The first 2 digits describe the value of Red, the next 2 the values of Green, and the last two the values of Blue.

Just know that it exists and that you will be able to use it if you want. We will not really dig into that in this training.

## @ Playing with borders

By default, without styling, each tag is rendered as a rectangle which background and borders are **transparent**. It does not have to stay that way!

Here is a visual representation of that rectangle, called the "box model".



This image represents how you can play with border, margin, padding to style any HTML tag!

Okay to explain this a little deeper, let's get our hands dirty!

***Create an HTML file and copy these lines in our body:***

```
<div class="box1">  <div class="box2">  </div> </div> <div
class="box3"> </div>
```

***Next create an CSS file and copy the following lines inside of this:***

```
.box1{ width:200px;
    height:200px;
    border-top:1px solid red;
    border-right: 1px solid black;
    border-bottom: 2px dotted green;
    border-left: 2px dashed green;
    padding:100px;
    padding-right:50px;
    background-color: yellow;

    /*--We will cover this later--*/
    display:inline-block;
    /*----------------------------*/}

.box2{ width:200px;
    height:200px;
    background-color:red;

    /*--We will cover this later--*/
    display:inline-block; /*----------------------------*/}

  .box3{
width:100px;
height:100px;
background-color:green;
margin-left:200px;
/*--We will cover this later--*/
display:inline-block;
/*----------------------------*/}
```

That sure won't look exactly pretty, but that's not the concern yet.

As you can see, you can specify each border of the rectangle using 3 parameters: the **thickness** of the line (here, in pixels), the line **type** ( solid, dashed, dotted), and its colour.

Now while you are at it, try to figure out the difference between padding and margin. Play with it's values, we will discuss this in group later on.

***Borders can be used to turn rectangles into a square!***

As you learn CSS, you will see that CSS is full of hacks and tricks. One really useful one is that you can turn an image like this:

*into that:*

Here is the one property that makes it possible:

```
border-radius:50%;
```

You'll get a chance to experiment with it in the exercises....

to use the correct HTML tags for your content. This is called writing semantic HTML, and it's importance can't be understated.

### @ Comments in CSS

Comments are used to explain the code, and may help when you edit the source code at a later date. Comments are **ignored by browsers**.

A CSS comment starts with /* and ends with */:

```
/* This is a single-line comment */

  p { color: red; }
```

You can add comments wherever you want in the code:

```
p { color: red;

  /* Set text color to red */}
```

Comments can also span multiple lines:

```
/* This isa multi-linecomment */

  p { color: red; }
```

### @ Congratulations!

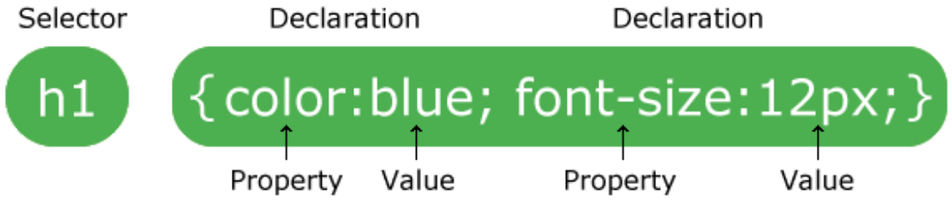
***You just discovered the magic of CSS. Now let's put our knowledge to the test by doing a few exercises!***

## Selectors

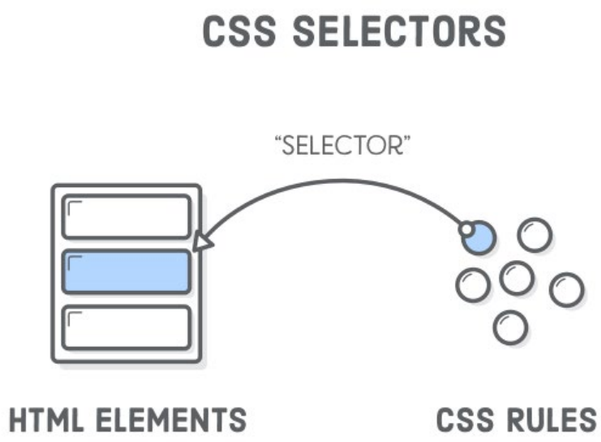
### @ CSS Selectors

Welcome to a very important part of css, the selectors.

Do you remember this visual from the introduction?




A CSS selector is the part of a CSS rule set that actually selects the content you want to style. Selectors are what allows the browser to know which html element(s) should be painted with which CSS properties.




For example I got this piece of HTML:

```
<div class="becode">

<p>I am on my way to become a web-developer!</p>

</div> <div id="white-background"> </div>
```

If I want to have a blue background on the first div, I could do this in CSS:

```
div{height:100px;

      width:100px;

      background-color:blue;}
```

However, if you do this, you will notice that both div elements will have a blue background.

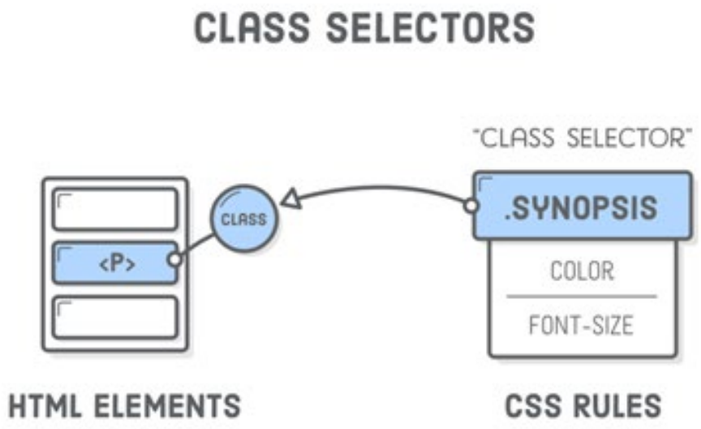
That's why there are more selectors than just the div selector! For example, we have written class 'becode', why not use it in our css file?

We can select class elements like this:

```
.becode{height:100px;
        width:100px;
        background-color:blue;}
```

Now only the background of our first div will be changed, just like we wanted!

What you did here, my friend, was using a class selector.




The Class Selector is indicated by a leading period (the .character) immediately follow by the class name you want to use. It can be anything, as long as it is the same as in the HTML file.

So, if in the html, the class name is "burger", the CSS class selector will be .... .burger. If the html class name is "ferrari", the CSS class selector becomes.... .ferrari. Got it ?

There are many ways to select html elements using the CSS selectors. Too many for this course. Know that the class selectors are the most used and will allow you to select everything you need, especially if you also know how to use...



## _Pseudo-class selectors

Most elements like H1, P ... are not meant to be interactive. But some are, like links which in HTML are called Anchors and are coded using the Atag.

Examine this CSS, styling all anchors on your page so that they look like the IKEA logo.

a.ikea {text-decoration:none; /* remove underline */

```
    background-color: blue; /* paint the background blue */
    color:yellow; /* paint the text yellow */
    font-weight:bold; /* make the characters bold */
    text-transform: uppercase; /* turn characters in UPPERCASE */
    padding:5px 10px; /* add some padding */
    font-size:23px; /* make the text bigger */}
```

So now, in your html, you can add the class "ikea" to all links that you want to look like they were bought at IKEA :-)

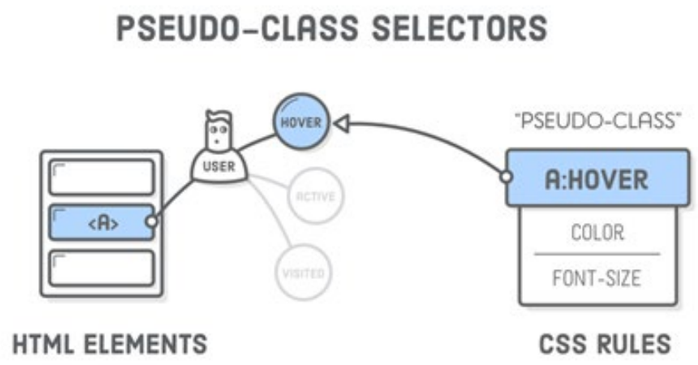
```
<p>I looooooove design, especially furniture from  <a href="http://
aeki.com" class="ikea">Aeki</a>. It's cheap and  stylish... Okay, if
you actually manage to build them!!</p>
```

Result:

I looooooove design, especially furniture from AEKI. It's cheap and stylish... Okay, if you actually manage to build them!!

Cool, but if you go with your mouse on top of it, nothing happens. Sad, for an interactive element!

That's where Pseudo-class enters the game...




CSS “pseudo-classes” provide a mechanism for hooking into this kind of temporary user information. At any given time, an <a href> element can be in a number of different states, and you can use pseudo-classes to style each one of them individually. Think of them as class selectors that you don’t have to write on your own because they’re built into the browser.

### *_Basic link styles*

Pseudo-classes begin with a colon followed by the name of the desired class. The most common link pseudo-classes are as follows:

- `link` – A link the user has never visited.
- `visited` – A link the user has visited before.
- `hover` – A link with the user’s mouse over it.
- `active` – A link that’s being pressed down by a mouse (or finger).

That's about it for the CSS Selectors. You have enough information to do the exercises! Don't hesitate to go back here if you cannot manage the exercises - you cannot remember all this. Practice will help you remember ;-)

## Basic properties

We know how to call our elements in css, we have used some basic properties, but there are way many more properties available to you, Picasso! In this section you will learn to use a few more interesting properties through short exercises.



## Positioning

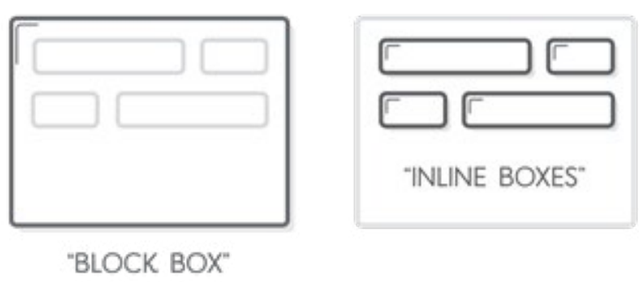
The big part of your job is to place elements on the screen, and control how they should adapt to the adjacent content as well as to the screen size (mobile phone or Desktop computer). In a word: **positioning**

CSS offers several ways to control the positioning of the elements on the screen. Let's start by giving you an understanding of how the element itself is laid out.

### *@ The Box Model*

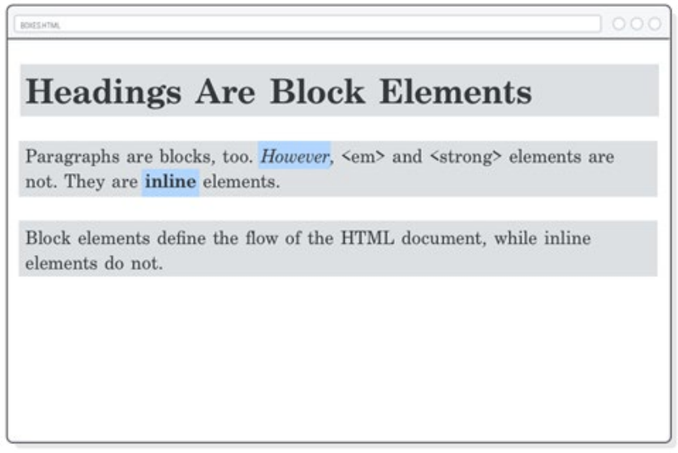
Each HTML element rendered on the screen is a box, and they come in two flavors: “block” boxes and “inline“ boxes.

Take a look at this visual representation of the difference between a block and an inline element.




All the HTML elements that we’ve been working with have a default type of box. For instance, <h1> and <p> are **block-level elements**, while  and  are **inline elements**.

Here is an example of the difference in behaviour between the two. (Blue = inline element, Grey = block element)




This shows us a couple of very important behaviors associated with block and inline boxes:

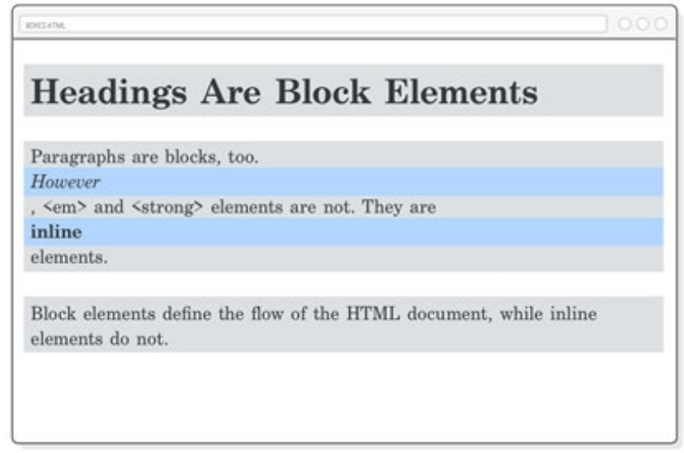
- **Block boxes** always appear below the previous block element. This is the "natural" or "static" flow of an HTML document when it gets rendered by a web browser.
- The **width of block boxes** is set automatically based on the width of its parent container. In this case, our blocks are always the width of the browser window.
- The default **height of block boxes** is based on the content it contains. When you narrow the browser window, the <h1> gets split over two lines, and its height adjusts accordingly.
- **Inline boxes** don't affect **vertical spacing**. They're not for determining layout—they're for styling stuff *inside* of a block.
- The **width of inline boxes** is based on the content it contains, not the width of the parent element.

## _Changing box behaviour

We can override the default box type of HTML elements with the CSS display property. For example, if we wanted to make our  and  elements blocks instead of inline elements, we could update our rule in `box-styles.css` like so:

```
em, strong { background-color: #B2D6FF; display: block;}
```





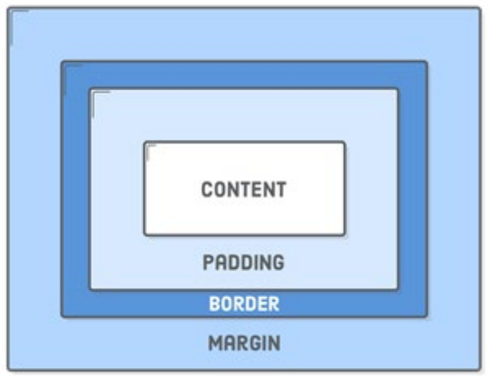

Now, the blue inline elements act like our headings and paragraphs: they start on their own line, and they fill the entire width of the browser. This comes in handy when you're trying to turn <a> elements into buttons or format <img/> elements (both of these are inline boxes by default).

## @ Content, padding, border, and margin

The "CSS box model" is a set of rules that determine the dimensions of every element in a web page. It gives each box (both inline and block) four properties:

- **Content** – The text, image, or other media content in the element.
- **Padding** – The space between the box's content and its border.
- **Border** – The line between the box's padding and margin.
- **Margin** – The space between the box and surrounding boxes.

Together, this is everything a browser needs to render an element's box. The content is what you author in an HTML document, and it's the only one that has any semantic value. The rest of them are purely presentational, so they're defined by CSS rules.

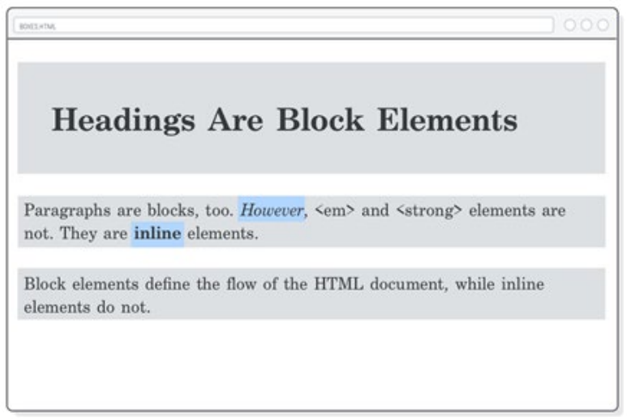



## _Padding

Let's start from the inside out. We've already been working with content, so on to padding. The `padding` property…you guessed it…defines the padding for the selected element:

```
h1 {  padding: 50px;}
```

This adds 50 pixels to *each side* of the `<h1>` heading. Notice how the background color expands to fill this space. That's always the case for padding because it's inside the border, and everything inside the border gets a background.




Sometimes you'll only want to style one side of an element. For that, CSS provides the following properties:

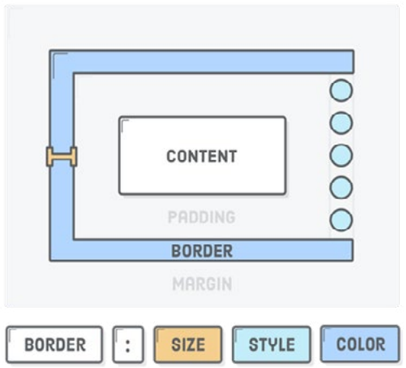
```
h1 {padding-top: 20px;
    padding-bottom: 20px;
    padding-left: 10px;
    padding-right: 10px;}
```

When you only mention `padding`: 50px; it's a **shortcut notation**: you are actually saying:

```
h1 {padding-top: 50px;
    padding-bottom: 50px;
    padding-left: 50px;
    padding-right: 50px; }
```

## _Borders

Continuing our journey outward from the center of the CSS box model, we have the border: a linedrawn around the content and padding of an element. The `border` property requires a new syntax that we've never seen before. First, we define the stroke width of the border, then its style, followed by its color.




Example:

```
h1 { border: 1px solid #5D6063;}
```

This would draw a rectangle around the H1 element.

Like padding, there are -top, -bottom, -left, and -right variants for the border property:

```
h1{ border-bottom: 1px solid #5D6063;}
```

This would simply draw a line underneath the H1 element.

## _Margins

Margins define the space outside of an element's border. Or, rather, the space between a box and its surrounding boxes. Let's add some space to the bottom of each <p> element:

```
p { padding: 20px 0 20px 10px; margin-bottom: 50px; }
```

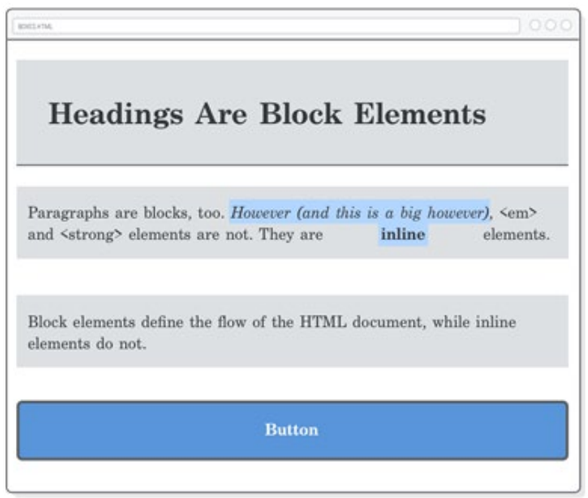
Margins and padding can accomplish the same thing in a lot of situations, making it difficult to determine which one is the "right" choice. The most common reasons why you would pick one over the other are:

- The padding of a box has a background, while margins are always transparent.
- Padding is included in the click area of an element, while margins aren't.

If none of these help you decide whether to use `padding` over margin, then don't fret about it—just pick one. In CSS, there's often more than one way to solve your problem.

## _Explicit dimensions

So far, we've let our HTML elements define their dimensions automatically. The paddings, borders, and margins we've been playing with all wrap around whatever content happens to be inside the element's box. If you add more text to our `<em>` element, everything will expand to accommodate it:




## _Centering with auto-margins

When you set the left and right margin of a block-level element to auto, it will center the block in its parent element.

For example, we can center our button with the following:

```
div{color:#FFF;background-color:#4A90E2;font-weight:bold;padding:
20px;text-align: center;width: 200px;box-sizing: border-box;
margin: 20px auto; }
```

Note that this only works on blocks that have an explicit width defined on them. Remove that width: 200px line, and our button will be the full width of the browser, making "center alignment" meaningless.

## @ Summary

We'll learn more about the practical uses of the CSS box model as we get deeper into constructing complex web pages. For now, think of it as a new tool in your CSS toolbox. With a few key concepts from this chapter, you should feel much more equipped to convert a design mockup into a real-life web page:

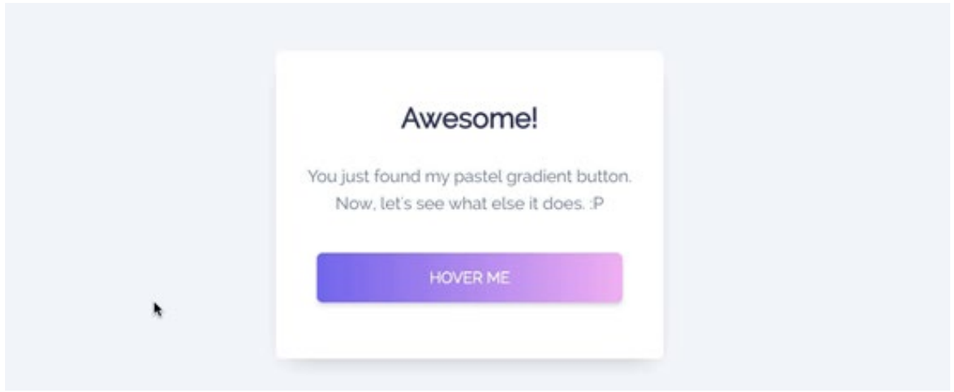
- Everything is a **box**.
- Boxes can be **inline** or **block-level**.
- Boxes have content, padding, **borders**, and **margins**.
- They also have seemingly arbitrary rules about how they interact.
- Mastering the CSS box model means you can lay out most web pages.

Like the last chapter, the CSS properties we just covered might seem simple—and they sort of are. But, start looking at the websites you visit through the lens of the CSS box model, and you'll see this stuff literally everywhere.

All right, let's go ahead and continue learning by doing a few positioning challenges using 3 techniques: Display, Position, and Flexbox!

# CSS Animations

## @ Transitions




CSS transitions allows you to **change CSS property values smoothly**, over a given **duration**. For example if you hover over a button, the color changes instantly without adding a transition property. With the transition property and the correct values, we can get smooth transitions between widths, heights, colors, ...

### _Syntax

To create a transition effect, you must specify two things:

the CSS property you want to add an effect to (ex: `background-color`))

- the duration of the effect (ex: 3s= 3 seconds)

Note: If the duration part is not specified, the transition will have no effect, because the default value is 0.

The following example shows a 100px * 100px red `<div>` element. The `<div>` element also has a transition effect for the width property, with a duration of 2 seconds:

```
div {width: 100px;
      height: 100px;
      background: red;
      transition: width 2s;}
```

The transition effect will start when the specified CSS property (width) changes value.

Now, let us specify a new value for the width property when a user mouses over the `<div>` element:

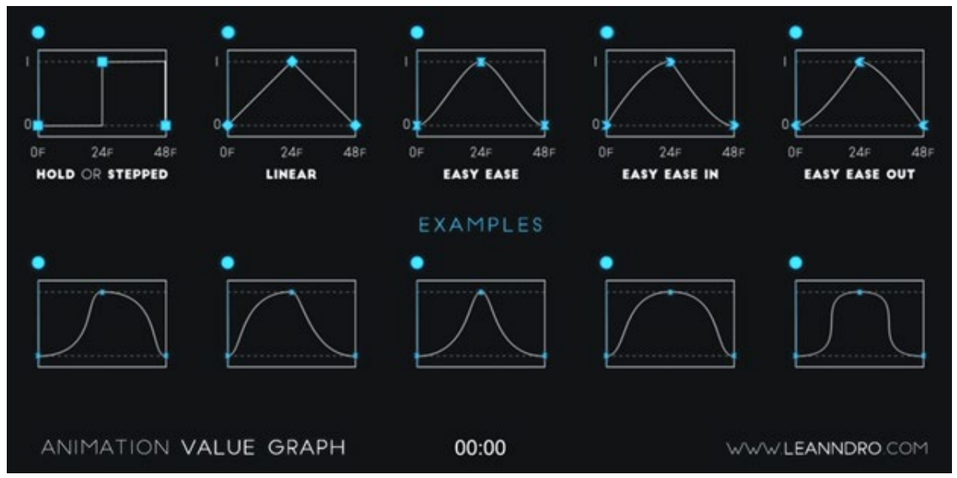
```
div:hover {width: 300px;}
```

When we hover over the div, it's width increases by 200px over a duration of 2 seconds.

### _Transition timing

But the animation looks a bit flat, right? That's because in animation terms this animation is "linear", like a flat line. If we are honest with ourselves, we rather look at rounded lines/shapes rather than a boring flat line.

It also makes your animations more natural..




Here is an example of how we can add a transition timing to our transition:

```
div{width: 100px;
      height: 100px;
            background: red; transition: width 2s;
            transition-timing-function: ease-in-out;
            /*You can also just write transition: width 2s ease-in-out-*/}
```

Feel free to experiment with other easing functions (search for them in the Documentation - remember searching is part of the job of a professional developer! ;-) )

## @ Animations

An animation lets an element gradually change from one style to another. You can change as many CSS properties you want, as many times you want.

To use CSS animations you must first specify some *keyframes* for the animation. A keyframe holds what CSS properties the element will have at a given time.

### _The @keyframes Rule

When you specify CSS styles inside the `@keyframes` rule, the animation will gradually change from the current style to the new style at certain times.

To get an animation to work, you must bind the animation to an element.

The following example binds the "example" animation to the `<div>` element. The animation will last for 4 seconds, and it will gradually change the background-color of the `<div>` element from "red" to "yellow":

```
/* The animation code */
@keyframes example {from {background-color: red;}to {background-color: yellow;}}
/* The element to apply the animation to */
     div {width: 100px;
          height: 100px;
          background-color: red;
          animation-name: example;
          animation-duration: 4s; }
```

**This would result something like this:**

**Note:** The `animation-duration` property defines how long time an animation should take to complete. If the `animation-duration` property is not specified, no animation will occur, because the default value is 0s (0 seconds).

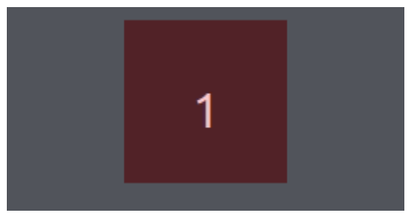
In the example above we have specified when the style will change by using the keywords `"from"` and `"to"` (which represents 0% (start) and 100% (complete)).

It is also possible to use percent. By using percent, you can add as many style changes as you like.

The following example will change the background-color of the `<div>` element when the animation is 25% complete, 50% complete, and again when the animation is 100% complete:

```
/* The animation code */
@keyframes example {0% {background-color: red;}
     25% {background-color: yellow;}
     50% {background-color: blue;}
     100% {background-color: green;}}
/* The element to apply the animation to */
     div {width: 100px;
          height: 100px;
          background-color: red;
          animation-name: example;
          animation-duration: 4s; }
```

**Result:**




## Final Challenge

This is your final assessment. If you can solve it, you will receive a certificate that testify on your ability to start a full training on Frontend Web Development and become a professional web developer (frontend).

# Annexes

## Session Plan – Module 0

| ***Training: Free to Code** – Improving Digital and Coding Skills for Inmates* | | |
|---|---|---|
| **Module: 0 – General introduction** | | *Trainer:* |
| **Session nr.: 0** *(half of a session)* | **Duration: 1h** | **Date:** |
| **General objectives:** | The learners (prisoners from 18 to 99 y.o.) will be able to modify an existing webpage and create basic web pages and using HTML and CSS with a code editor.<br>They will also acquire basic on how to describe a solution to simple web development problems in logical steps and understand what each frontend technology is best used for (html, css and Javascript). | |
| **Learning outcomes:** | • Frontend coding fundamentals: Fix webpage and create a basic webpage using HTML and CSS<br>• Describe a solution to simple problems in logical steps<br>• Basic terminology of frontend web development technologies<br>• Be able to use the Coding interface provided<br>• Algorithmic and problem solving | |
| **Methods:** | Expository, interrogative, and active.<br>Computer-based self-learning based on active learning techniques, such as project-based learning, quizzes and learning by doing, under the guidance of a technical monitor – blended learning. | |
| **Content:** | • Background<br>• Internet<br>• Web | |



| Time | Activities: | Resources and materials: |
|---|---|---|
| 10m | **Icebreaking activity**<br>**Wool ball activity** – put the trainees and trainers in circle, and ask them to introduce themselves (name, age, where they are from) and then pass the ball to whoever they wish<br>The purpose of this activity is in the end to say that they have started a network of contacts at this time and that they must keep with each other in order to help each other now and in the future - the training aims to be self-learning and the trainees must help each other to succeed. This activity also serves as the starting point of the next **brainstorming** – analogy between the network that was created with the wool ball and technology to begin the brainstorming questions (see below) | • Ball of wool and space in the room to create a circle with all participants |
| 10m | **Brainstorming** regarding Technology to open the discussion and to introduce the topic<br>Questions suggestions:<br>• *"What do you see when thinking in Technology?"*<br>• *"Makes our lives easier?"*<br>• *"What is an website? And an app?"*<br>• *"Where does coding stand in the technology world?"*<br>• *Other questions you might find relevant* | • Flipchart and pens or<br>• Word document<br>• Etc. |
| 40m | Presentation of the 3 main topics covered in the learning part of the module 0:<br>1. Background<br>a. The Internet – birth, interface and language<br>2. The Internet<br>a. How does it work, network of networks and the web<br>3. Web<br>a. How does it work – clients and serves, DNS and packets<br>Conclusion, discussion, and questions<br>*(to be continued this 1st session in the next table)* | • Learning platform |
| **Assessment:** | Formal assessments on the learning outcome (quizzes, Platform exercises)<br>Trainers must gather participants' feedback as well as take their own notes regarding the session. | |
| **Activity (Time):** | Lessons 1-3 (1h) | |

# Session Plan – Module 1

| ***Training: Free to Code** – Improving Digital and Coding Skills for Inmates* | | |
|---|---|---|
| **Module: 1 – HTML** | | *Trainer:* |
| **Session nr.: 0 to 2** *(2 and half sessions)* | **Duration: 8h** | **Date:** |
| **General objectives:** | The learners (prisoners from 18 to 99 y.o.) will be able to modify an existing webpage and create basic web pages and using HTML and CSS with a code editor.<br>They will also acquire basic on how to describe a solution to simple web development problems in logical steps and understand what each frontend technology is best used for (html, css and Javascript). | |
| **Learning outcomes:** | • Frontend coding fundamentals: Fix webpage and create a basic webpage using HTML and CSS<br>• Describe a solution to simple problems in logical steps<br>• Basic terminology of frontend web development technologies<br>• Be able to use the Coding interface provided<br>• Algorithmic and problem solving | |
| **Methods:** | Expository, interrogative, and active.<br>Computer-based self-learning based on active learning techniques, such as project-based learning, quizzes and learning by doing, under the guidance of a technical monitor – blended learning. | |
| **Content:** | • Intro<br>• HTML5<br>• Semantics | |



| Time | Activities: | Resources and materials: |
|---|---|---|
| 1h | **Session 0 (continuation)**<br>The training of the 1st module will be divided in two parts - a presentation part, and exercises-part.<br>Presentation of the 3 main topics covered in the learning part of module 1:<br>1. Intro<br>a. The Internet – birth, interface and language<br>2. HTML5<br>a. First HTML document<br>3. Semantics<br>a. Why use semantics, semantic HTML | • Platform – learning part |
| 1h | After the presentation trainees must do the exercises designated for this 1st training module.<br>The exercises should be carried out by the trainees, but always supported by the trainers, and with continuous access to the learning part of the platform so that they can review content that helps them perform the exercises.<br>***Exercises:***<br>• Exercise 1 – hello world (15 minutes)<br>• Exercise 2 – Titles (15 minutes)<br>• Exercise 3 – Paragraphs (15 minutes)<br>• Exercise 4 – Bold Cursive (15 minutes)<br>*After 3 hours close the first session with conclusion, discussion, and questions* | • Platform – learning part |
| **Assessment:** | Formal assessments on the learning outcome (quizzes, Platform exercises and Final Challenge) – if there is any drop out, they must fill the "Satisfaction form" of "How to setup your pilot" support document.<br>Trainers must gather participants' feedback as well as take their own notes regarding the session. | |

| Time | Activities: | Resources and materials: |
|---|---|---|
| 3h | **Session 1**<br>Review of the learning contents discussed before and keep working in the exercises.<br>The exercises should be carried out by the trainees, but always supported by the trainers, and with continuous access to the learning part of the platform so that they can review content that helps them perform the exercises.<br>Act accordingly for the following sessions of all the modules.<br>*Exercises:*<br>• Exercise 5 – Lists (30 minutes)<br>• Exercise 6 – Relative Links (30 minutes)<br>• Exercise 7 – Images (30 minutes)<br>• Exercise 8 – The letter (1h30)<br>*After 3 hours close the session with conclusion, discussion, and questions* | • Platform – learning and training parts |
| **Assessment:** | Formal assessments on the learning outcome (quizzes, Platform exercises and Final Challenge) – if there is any drop out, they must fill the "Satisfaction form" of "How to setup your pilot" support document.<br>Trainers must gather participants' feedback as well as take their own notes regarding the session. | |
| 3h | **Session 2**<br>*Exercises:*<br>• Exercise 9 – Recipe formatting (1h)<br>• Exercise 10 – Chinese farmer (2h)<br>*After 3 hours close the session with conclusion, discussion, and questions* | • Platform – learning and training parts |
| **Assessment:** | Formal assessments on the learning outcome (quizzes, Platform exercises and Final Challenge) – if there is any drop out, they must fill the "Satisfaction form" of "How to setup your pilot" support document.<br>Trainers must gather participants' feedback as well as take their own notes regarding the session. | |
| **Activity (Time):**<br>*Lessons 1-3 (1h)* | Exercise 1 – hello world (15 minutes)<br>Exercise 2 – Titles (15 minutes)<br>Exercise 3 – Paragraphs (15 minutes)<br>Exercise 4 – Bold Cursive (15 minutes)<br>Exercise 5 – Lists (30 minutes)<br>Exercise 6 – Relative Links (30 minutes)<br>Exercise 7 – Images (30 minutes)<br>Exercise 8 – The letter (1h30)<br>Exercise 9 – Recipe formatting (1h)<br>Exercise 10 - Chinese farmer (2) | |



# Session Plan – Module 2

| ***Training: Free to Code** – Improving Digital and Coding Skills for Inmates* | | |
|---|---|---|
| **Module: 2 – CSS** | | *Trainer:* |
| **Session nr.: 3 to 18** | **Duration: 43h** | **Date:** |
| **General objectives:** | The learners (prisoners from 18 to 99 y.o.) will be able to modify an existing webpage and create basic web pages and using HTML and CSS with a code editor.<br>They will also acquire basic on how to describe a solution to simple web development problems in logical steps and understand what each frontend technology is best used for (html, css and Javascript). | |
| **Learning outcomes:** | • Frontend coding fundamentals: Fix webpage and create a basic webpage using HTML and CSS<br>• Describe a solution to simple problems in logical steps<br>• Basic terminology of frontend web development technologies<br>• Be able to use the Coding interface provided<br>• Algorithmic and problem solving | |
| **Methods:** | Expository, interrogative, and active.<br>Computer-based self-learning based on active learning techniques, such as project-based learning, quizzes and learning by doing, under the guidance of a technical monitor – blended learning. | |
| **Content:** | • Introduction to CSS<br>• Selectors<br>• Basic Properties<br>• Positioning<br>• CSS Animations<br>• Final Challenge | |

| Time | Activities: | Resources and materials: |
|---|---|---|
| 2h | **Session 3**<br>The training will be divided in two parts - a presentation part, and exercises-part.<br>Presentation of the topics covered in the learning part of the Platform:<br>1. Introduction to CSS<br>a. What is, syntax, getting started, working with colours, borders, comments<br>2. HTML5<br>a. CSS selectors, class selectors, pseudo-class selectors<br>3. Basic properties<br>4. Positioning<br>a. The box model, padding, borders, margins, dimensions<br>5. CSS animations<br>a. Transitions, animations | • Platform – learning part |
| 1h | **Session 4**<br>*Exercises:*<br>• Exercise 1 – Add some inline CSS (30 minutes)<br>• Exercise 2 – Plan your vacations with style (30 minutes)<br>*After 3 hours close the first session with conclusion, discussion, and questions* | • Platform – Training Part |
| **Assessment:** | Formal assessments on the learning outcome (quizzes, Platform exercises) – if there is any drop out, they must fill the "Satisfaction form" of "How to setup your pilot" support document.<br>Trainers must gather participants' feedback as well as take their own notes regarding the session. | |
| 3h | **Session 5**<br>*Exercises:*<br>• Exercise 3 – Canary Islands (30 minutes)<br>• Exercise 4 – Classes to differentiate elements and style them differently (30 minutes)<br>• Exercise 5 – Borders (30 minutes)<br>• Exercise 6 – Class Selector (1h)<br>• Exercise 7 – Pseudo-class selector (30 minutes)<br>*After 3 hours close the session with conclusion, discussion, and questions* | • Platform – learning and training parts |
| **Assessment:** | Formal assessments on the learning outcome (quizzes, Platform exercises) – if there is any drop out, they must fill the "Satisfaction form" of "How to setup your pilot" support document.<br>Trainers must gather participants' feedback as well as take their own notes regarding the session. | |



| Time | Activities: | Resources and materials: |
|---|---|---|
| 3h | **Session 6**<br>*Exercises:*<br>• Exercise 8 – Style this page (1h)<br>• Exercise 9 – Basic properties (1h)<br>• Exercise 10 – Display properties exercise 1 (1h)<br>*After 3 hours close the session with conclusion, discussion, and questions* | • Platform – learning and training parts |
| **Assessment:** | Formal assessments on the learning outcome (quizzes, Platform exercises) – if there is any drop out, they must fill the "Satisfaction form" of "How to setup your pilot" support document.<br>Trainers must gather participants' feedback as well as take their own notes regarding the session. | |
| 3h | **Session 7**<br>*Exercises:*<br>• Exercise 11 – Display properties exercise 2 (1h)<br>• Exercise 12 – Position properties exercise 1 (2h)<br>*After 3 hours close the session with conclusion, discussion, and questions* | • Platform – learning and training parts |
| **Assessment:** | Formal assessments on the learning outcome (quizzes, Platform exercises) – if there is any drop out, they must fill the "Satisfaction form" of "How to setup your pilot" support document.<br>Trainers must gather participants' feedback as well as take their own notes regarding the session. | |
| 3h | **Session 8**<br>*Exercises:*<br>• Exercise 13 – Position properties exercise 2 (1h)<br>• Exercise 14 – Position properties exercise 3 (1h)<br>• Exercise 15 – Flex-box 1 (1h)<br>*After 3 hours close the session with conclusion, discussion, and questions* | • Platform – learning and training parts |
| **Assessment:** | Formal assessments on the learning outcome (quizzes, Platform exercises) – if there is any drop out, they must fill the "Satisfaction form" of "How to setup your pilot" support document.<br>Trainers must gather participants' feedback as well as take their own notes regarding the session. | |

| Time | Activities: | Resources and materials: |
|---|---|---|
| 3h | **Session 9**<br>*Exercises:*<br>• Exercise 16 – Flexing card (2h)<br>• Exercise 17 – Flexing page (1 of 3h)<br>*After 3 hours close the session with conclusion, discussion, and questions* | • Platform – learning and training parts |
| **Assessment:** | Formal assessments on the learning outcome (quizzes, Platform exercises) – if there is any drop out, they must fill the "Satisfaction form" of "How to setup your pilot" support document.<br>Trainers must gather participants' feedback as well as take their own notes regarding the session. | |
| 3h | **Session 10**<br>*Exercises:*<br>• Exercise 17 – Flexing page (2 of 3h)<br>• Exercise 18 – CSS integration series (1 of 4h)<br>*After 3 hours close the session with conclusion, discussion, and questions* | • Platform – learning and training parts |
| **Assessment:** | Formal assessments on the learning outcome (quizzes, Platform exercises) – if there is any drop out, they must fill the "Satisfaction form" of "How to setup your pilot" support document.<br>Trainers must gather participants' feedback as well as take their own notes regarding the session. | |
| 3h | **Session 11**<br>*Exercises:*<br>• Exercise 18 – CSS integration series (3 of 4h)<br>*After 3 hours close the session with conclusion, discussion, and questions* | • Platform – learning and training parts |
| **Assessment:** | Formal assessments on the learning outcome (quizzes, Platform exercises) – if there is any drop out, they must fill the "Satisfaction form" of "How to setup your pilot" support document.<br>Trainers must gather participants' feedback as well as take their own notes regarding the session. | |



| Time | Activities: | Resources and materials: |
|---|---|---|
| 3h | **Session 12**<br>*Exercises:*<br>• Exercise 19 – CSS Animations (1h)<br>• Exercise 20 – Small CSS animations series (2h)<br>*After 3 hours close the session with conclusion, discussion, and questions* | • Platform – learning and training parts |
| **Assessment:** | Formal assessments on the learning outcome (quizzes, Platform exercises) – if there is any drop out, they must fill the "Satisfaction form" of "How to setup your pilot" support document.<br>Trainers must gather participants' feedback as well as take their own notes regarding the session. | |
| 3h | **Session 13**<br>*Exercises:*<br>• Exercise 21- Small CSS animations series 1 (2h)<br>• Exercise 22 – Small CSS animations series 2 (1 of 3h)<br>*After 3 hours close the session with conclusion, discussion, and questions* | • Platform – learning and training parts |
| **Assessment:** | Formal assessments on the learning outcome (quizzes, Platform exercises) – if there is any drop out, they must fill the "Satisfaction form" of "How to setup your pilot" support document.<br>Trainers must gather participants' feedback as well as take their own notes regarding the session. | |
| 3h | **Session 14**<br>*Exercises:*<br>• Exercise 22 – Small CSS animations series 2 (2 of 3h)<br>• Exercise 23 – Small CSS animations series Bonus (1 of 4h)<br>*After 3 hours close the session with conclusion, discussion, and questions* | • Platform – learning and training parts |
| **Assessment:** | Formal assessments on the learning outcome (quizzes, Platform exercises) – if there is any drop out, they must fill the "Satisfaction form" of "How to setup your pilot" support document.<br>Trainers must gather participants' feedback as well as take their own notes regarding the session. | |

| Time | Activities: | Resources and materials: |
|---|---|---|
| 3h | **Session 15**<br>*Exercises:*<br>• Exercise 23 – Small CSS animations series Bonus (3 of 4h)<br>*After 3 hours close the session with conclusion, discussion, and questions* | • Platform – learning and training parts |
| **Assessment:** | Formal assessments on the learning outcome (quizzes, Platform exercises) – if there is any drop out, they must fill the "Satisfaction form" of "How to setup your pilot" support document.<br>Trainers must gather participants' feedback as well as take their own notes regarding the session. | |
| 3h | **Session 16**<br>*Exercises:*<br>• Exercise 24 – Final Challenge (3 of 9h)<br>*After 3 hours close the session with conclusion, discussion, and questions* | • Platform – learning and training parts |
| **Assessment:** | Formal assessments on the learning outcome (quizzes, Platform exercises) – if there is any drop out, they must fill the "Satisfaction form" of "How to setup your pilot" support document.<br>Trainers must gather participants' feedback as well as take their own notes regarding the session. | |
| 3h | **Session 17**<br>*Exercises:*<br>• Exercise 24 – Final Challenge (3 of 9h)<br>*After 3 hours close the session with conclusion, discussion, and questions* | • Platform – learning and training parts |
| **Assessment:** | Formal assessments on the learning outcome (quizzes, Platform exercises) – if there is any drop out, they must fill the "Satisfaction form" of "How to setup your pilot" support document.<br>Trainers must gather participants' feedback as well as take their own notes regarding the session. | |



| Time | Activities: | Resources and materials: |
|---|---|---|
| 3h | **Session 18**<br>*Exercises:*<br>• Exercise 24 – Final Challenge (3 of 9h)<br>*After 3 hours close the session with conclusion, discussion, and questions* | • Platform – learning and training parts |
| **Assessment:** | Formal assessments on the learning outcome (quizzes, Platform exercises and Final Challenge) – if there is any drop out, they must fill the "Satisfaction form" of "How to setup your pilot" support document.<br>Trainers must gather participants' feedback as well as take their own notes regarding the session.<br>To close the training, trainers must fill "Observation & feedback template" and trainees "Satisfaction form" available in the "How to setup up your pilot" support document | |
| **Activity (Time):**<br>*Lessons 1-5 (1h)* | Exercise 1 – Add some inline CSS (30 minutes)<br>Exercise 2 – Plan your vacations with style (30 minutes)<br>Exercise 3 – Canary Islands (30 minutes)<br>Exercise 4 – Classes to differentiate elements and style them differently (30 minutes)<br>Exercise 5 – Borders (30 minutes)<br>Exercise 6 – Class Selector (1h)<br>Exercise 7 – Pseudo-class selector (30 minutes)<br>Exercise 8 – Style this page (1h)<br>Exercise 9 – Basic properties (1h)<br>Exercise 10 – Display properties exercise 1 (1h)<br>Exercise 11 – Display properties exercise 2 (1h)<br>Exercise 12 – Position properties exercise 1 (2h)<br>Exercise 13 – Position properties exercise 2 (1h)<br>Exercise 14 – Position properties exercise 3 (1h)<br>Exercise 15 – Flex-box 1 (1h)<br>Exercise 16 – Flexing card (2h)<br>Exercise 17 – Flexing page (3h)<br>Exercise 18 – CSS integration series (4h)<br>Exercise 19 – CSS Animations (1h)<br>Exercise 20 – Small CSS animations series (2h)<br>Exercise 21- Small CSS animations series 1 (2h)<br>Exercise 22 – Small CSS animations series 2 (3h)<br>Exercise 23 – Small CSS animations series Bonus (4h)<br>Exercise 24 – Final Challenge (9h) | |

# FREE TO CODE

Co-funded by the
Erasmus+ Programme
of the European Union

Project Number 2018-1-RO01-KA204-049298