# DEV211.1x Introduzione a HTML, CSS e JavaScript per il Web Development

```
browser-sync start --server --directory --files "*"
```

## INSTALLARE VISUAL STUDIO CODE

Per questo corso useremo Visual Studio Code, una IDE leggera disponibile su più OS.

Per installare Visual Studio Code, vai al sito: https://code.visualstudio.com/

e clicca il bottone **Download** nella home page.

Documentazione https://code.visualstudio.com/docs?start=true

Per aprire una tua pagina html e vederla a browser premi **F1** `F1` or `Ctrl+Shift+P` e sua l'estensione "**View in Browser**" o anche detta "**View in Default Application**" **Ctrl-K e Ctrl-b**

Per zoomare e dezoomare **ctrl+** e **ctrl-**

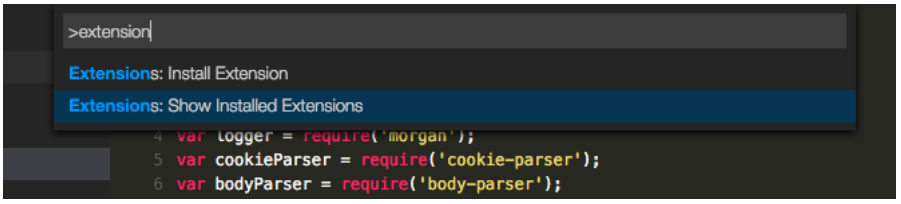
Puoi usare il controllo di versione con il terzo bottone **Git**

## INSTALLARE L'ESTENSIONE "VIEW IN DEFAULT APPLICATION"

Puoi guardare il video o seguire i passi sotto per installare l'estensione.

## INSTALLAZIONE

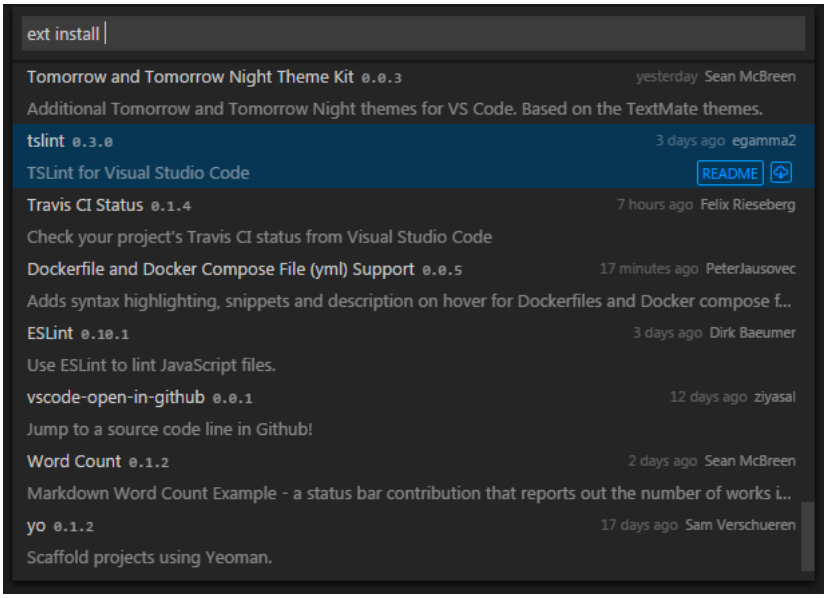
Puoi navigare e installare l'estensione direttamente dal VS Code. Premi `F1` o `Ctrl+Shift+P` e riduci l'elenco dei comandi scrivendo `extension`:



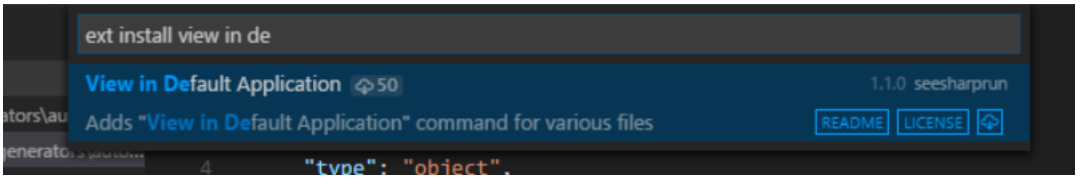

Seleziona il comando `Extensions: Install Extension` .

> **Tip:** Come alternativa, premi `Ctrl+P` ed inserisci `ext install con` unacon uno spazio finale.

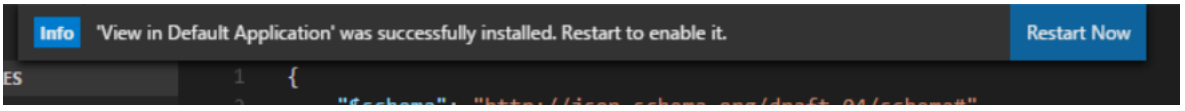
Puoi vedere un elenco delle estensioni disponibili:




Cerca l'estensione scrivendo `view nellain default application`:




Una volta installato, ti verrà chiesto di fare il restart della tua istanza di Visual Studio Code per usare l'estensione:




## USO DELL'ESTENSIONE

Puoi usare l'estensione sia invocando il comando `View in Default Application` oppure usando la shortcut `F1`:


Oppure usando le seguenti combinazioni:

| OS | Keybinding |
|---|---|
| Windows/Linux | `ctrl+k ctrl+b` |
| OSX | `cmd+k cmd+b` |

## MARKUP SEMANTICO

Fin dagli albori dell'umanità, l'umanità ha cercato di esprimere i propri pensieri e registrare i propri ricordi. Prima delle lingue, l'iconografia ed i numeri furono usati per mantenere i ricordi. Non molto dopo la nascita del linguaggio scritto, abbiamo cominciato a volere memorizzare i nostri pensieri, trionfi e le nostre scoperte. Poco dopo aver iniziato a scrivere la nostra storia, siamo andati alla ricerca di modi + efficienti per scrivere. Nel sistema numerico, abbiamo usato la notazione posizionale di modo da poter scrivere grossi numeri solo con l'uso di pochi simboli (o cifre). Nella scrittura invece gli autori cercavano di catturare sempre + storie e pensieri da una popolazione che stava crescendo.

Oggi stiamo ancora cercando modi efficienti per registrare le nostre scoperte. Molti accademici e autori scrivono libri che contengono centinaia di pagine. Non hai scritto un documento in Microsoft Word? Quante pagine ha il documento + lungo in Word che hai? Riesci ad immaginare cosa voglia dire scrivere un documento con un word processor di centinaia o persino migliaia di pagine?

## Cos'è un Markup Semantico?

Semantic markup sono dati in formato testo che vengono annotati usando un linguaggio speciale che offra al testo maggior siginificato.

Ad esempio, consideriamo un elenco di parole nell'alfabeto fonetico:

Delta

Echo

Foxtrot

Nel contesto del testo, possiamo semplicemente elencare le parole in questo modo:

```
Delta Echo Foxtrot
```

Ma come può un'altra persona decifrare il significato di questo blocco di testo? E' per caso una citazione? Oppure fa parte di una frase incompleta?

Ci serve dunque **annotare** questo elenco di modo da poter capire il **significato** di questo testo. Vediamo un esempio:

```
[list]
  [item]Delta[\item]
  [item]Echo[\item]
 [item]Foxtrot[\item][\list]
```

Con questa annotazione, facciamo capire agli altri che il **significato** di questo blocco di testo è una lista. Il significato in questo contesto viene tradotto in **semantica** del linguaggio. E l'annotazione in questo caso viene tradotta in **markup** del linguaggio.

Possiamo ora inserire questo blocco di testo in grossi documenti e le persone potranno capire ora che si tratta di un elenco. Guarda l'esempio:

```
Below is a subset of the words in the phonetic alphabet.[list]
  [item]Delta[\item]
  [item]Echo[\item]
 [item]Foxtrot[\item][\list]There are other words that are not listed here.
```

Una persona che legge questo documento può chiaramente capire che c'è del testo, seguito da una lista e concluso con un altro blocco di testo. Il significato del documento ora è molto chiaro grazie all'uso del **semantic markup**.

## Uso del Semantic Markup

Semantic Markup viene usato da una larga varietà di aziende al giorno d'oggi e l'innovazione continua aiuta anche il suo incedere.

#### Uso scolastico

A scuola, gli studenti lavorano alle loro tesi o disserzioni usando delle tecniche che gli permettano di focalizzarsi sul significato (semantica) del loro contenuto (markup) e di non preoccuparsi dell'aspetto grafico. Quando usi un word processor o qualsiasi altro strumento per il design dei documenti, spendi la maggior parte del tempo nel configurare il layout del documento. I ricercatori delle università trovano che è molto + semplice scrivere solo il contenuto e poi gestire in seguito il layout con un tool per il "design". Ancor + importante, separando il contenuto dal layout nei documenti fan si che il processo sia ancora + efficiente. Possono in questo modo cambiare il layout per molte e molte volte dopo che il contenuto è stato riempito. Possono anche aggiornare il contenuto del documento senza preoccuprarsi che il documento appaia in modo corretto.

Negli ultimi anni sono stati inventati degli innovativi linguaggi di Markup come **Markdown** e **LaTeX** che permettono ai ricercatori di scrivere il loro contenuto in un modo molto conciso. Strumenti come **Pandoc** vengono usati per convertire il markup in formati che vengono distribuiti come documenti **PDF** o **Word**. Puoi persino specificare la formattazione dei documenti generati. Vediamo un esempio:

#### Markdown markup

In questo esempio, abbiamo un documento Markdown di markup che rappresenta una tabella di studenti in uan scuolathat represents a table of students at a school.

```
age | name | grade
--- | --- | ---
8 | Peter | 3
7 | Susan | 2
8 | Jessica | 3
```

Se runniamo attraverso un parser questo markup, ci verrà renderizzata una tabella. Ecco una tabella renderizzata che usa lo stile di GitHub:

| age | name | grade |
|---|---|---|
| 8 | Peter | 3 |
| 7 | Susan | 2 |
| 8 | Jessica | 3 |

Esempio di Markdown

Siamo stati in grado di rappresentare il significato (semantica) dei nostri dati definendo una tabella. In base allo stile usato, un altro tool può renderizzare la tabella in ogni aspetto che è necessario.

Questo va benissimo per grossi documenti in cui l'autore deve foclaizzarsi nello scrivere contenuti e che usi un editor o un altro gruppo si occupi della formattazione vera e propria che verrà usata per pubblicarlo nel web o stamparlo.

**Campo Scientifico**

L'innovazione di separare il contenuto dal design divenne molto importante pure nel campo scientifico. Gli Scienziati avevano bisogno di condividere informazioni agli albori della comunicazione via rete e gli serviva che questi trasferimenti fossero molto veloci. Inviando del semplice testo, potevano focalizzarsi in ciò che per loro era realmente importante (le loro ricerche) invece di provare a verificare che il loro contenuto "apparisse" in modo corretto. Quando erano pronti a pubblicare le loro ricerche, loro (o un'altra persona) possono semplicemente applicare una formattazione al testo.

**Internet**

Semantic Markup viene usato da una larga varietà di forum e siti web. Se crei un progetto in GitHub, GitHub usa un linguaggio di markup chiamato "Markdown". Molti forum o blog hanno il loro specifico linguaggio di markup. Tutti questi comunque ti permettono di scrivere il tuo contenuto velocemente e lasciare all'applicazione web di gestire la visualizzazione ed il design del tuo contenuto.

L'implementazione + comune dei linguaggi di markup semantici è HTML. HTML ci permette di annotare la nostra semantica usando un linguaggio di markup e poi di renderizzarlo lato client mediante il browser. Se volessimo quindi cambiare l'aspetto della nostra pagina web, lo potremmo fare semplicemente cambiando il CSS. Vedremo HTML + avanti. Ecco un esempio:

**HTML Markup**

In questo esempio, abbiamo un markup HTML che rappresenta un elenco di elementi dell'alfabeto fonetico.

```
<ul>
    <li>Alpha</li>
    <li>Bravo</li>
  <li>Charlie</li></ul>
```

se mostriamo a browser questo HTML, il browser lo renderizzerà come una lista che poi apparirà in modo diverso in base ai diversi browser. Ecco un esempio usando Google Chrome:

- Alpha
- Bravo
- Charlie

HTML Esempio

Siamo stati in grado di rappresentare il significato (semantica) dei nostri dati definendo una lista. Possiamo modificare l'aspetto di qiesta lista modificando il CSS se vogliamo.

NASCIATA DEL WEB

Nel 1989, Tim Berners-Lee lavorava in un particolare laboratorio fisico chiamato CERN. Mentre stava lì capì che c'era un grande bisogno di condividere una grossa quantità di dati e documenti. A causa del testo che vi era all'epoca, i ricercatori già utilizzavano un linguaggio di markup di modo che potessero scrivere documenti usando un semplice formato SOLO testo che potesse essere condiviso facilmente e velocemente con altri laboratori.

Questo formato che molti ricercatori usavano era chiamato Standard Generalized Markup Language (SGML). SGML includeva annotazioni che permettevano ai ricercatori di applicare una formattazione ai loro documenti in modo molto veloce tipo:

- **<P>:** paragrafi
- **<H1>-<H6>:** header
- **<TITLE>:** titoli
- **<OL>:** ordered lists
- **<UL>:** unordered lists

La specifica di SGML era molto aperta e questo permetteva ai diversi laboratori di essere creatvi nell'usare il linguaggio. Questo portò, a volte, a varie incompatibilità tra i diversi laboratori in quanto avevano diverse interpretazioni dello stesso markup.

SGML inoltre mancava una funzionalità che divenne chiave in HTML. I Documenti creati usando SGML non avevano un modo standardizzato di referenziarsi ad altri documenti. Quando Tim Berners-Lee creò il linguaggio HTML, estese SGML aggiungendovi molte + annotazioni e l'abilità, dei doumenti, di referenziarsi l'un l'altro.

## Linking (Collegamento)

Una delle cose su cui si focalizzò Tim Berners-Lee nei suoi primi propositi in HTML era l'abilità di referenziare altri documenti o testi dentro ad un documento. Usando questi riferimenti, i lettori potevano velocemente avere accesso ai documenti e continuare a leggere. Questo fu molto utile nell'ambito accademico dove i ricercatori potevano andare "+ in profondità" in ricerche collegate e scoprire informazioni che potevano aver difficoltà di reperire nei sistemi precedenti. Cominciarono a riferirsi ai testi e al design come **hypertext** ed **hypermedia**.

## HTML Markup

HTML è una vera e propria estensione di SGML. HTML fa alcune cose in modo diverso:

- HTML restringe il numero di tag SGML permessi nelle pagine web
- HTML creò un tag dedicato (<a>) chiamato anchor tag per riferirsi ad altri documenti

L'abilità di HTML di creare collegamenti fra i vari documenti lo aiutò a guadagnare una popolarità in poco tempo fra i ricercatori e gli scienziati. I documenti di ricerca potevano facilmente collegarsi l'un l'altro o avere riferimenti ad altri documenti per creare il web di informazioni. Nel 1990, Tim Berners-Lee pubblicò la prima proposta formale intitolata **WorldWideWeb** che descriveva un sistema di ipertesti visualizzati attraverso un software chiamato **browser** in un'architettura client-server.

## World Wide Web Consorzio

Non molto dopo le prime due versioni di HTML (HTML 1.0 ed HTML 2.0), Tim Berners-Lee fondò e condusse il World Wide Web Consortium (W3C). Il gruppo formato dai membri dell'organizzazione e dallo staff lavorò insieme per sviluppare nuovi standard per le basi del World Wide Web. Il W3C creò le specifiche per **HTML**, **XML**, **CSS** ed anche **XHTML**. Inoltre svilupparono altre specifiche che includevano:

- SOAP
- SPARQL
- MathML

XML

Extensible Markup Language (XML) era un altro linguaggio XML nato dal World Wide Web Consortium (W3C). XML era progettato per rappresentare dati in formati che fossero sia **leggibili dagli umani che dalle macchine**. XML era molto + legato a SGML che HTML ma XML era + vincolante di SGML dato che richiedeva che tutti i tag fossero chiusi e che gli attributi fossero espliciti.

La chiave di XML è che usava un linguaggio di markup molto rigoroso che rappresentava dati in formato gerarchico. Vediamo un esempio:

```
<?xml version="1.0" encoding="UTF-8"?><list>
  <item price="0.75">Apple</item>
  <item price="0.25">Banana</item>
  <item price="1.00">
    Orange
    <plucodes>
      <plucode description="Navel Orange">4012</plucode>
      <plucode description="Blood Orange">4381</plucode>
      <plucode description="unknown" />
    </plucodes>
 </item></list>
```

Questo esempio di dati è una list di frutti in un supermercato. Per ogni frutto/item **elemento**, abbiamo il prezzo settato come **attributo** dell'**elemento**. Per l'elemento **oranges**, abbiamo anche un **elemento** figlio che ha tre **elementi** figli. Ognuno di questi **elementi** plucode specifa la sua descrizione come un **attributo** ed il codice come **contenuto** dell'**elemento**.

## Esempi XML

XML ha delle parole chiave per descrivere i costrutti del linguaggio. Molta di questa terminologia viene usata anche in HTML.

### Tag

A tag è un costrutto del ling. di markup XML che comincia col carattere < e termina col carattere >.

Abbiamo tre tipi di tag visti in XML:

**Opening (or Start) Tag:** `<item>` Rappresenta l'inizio di un componente logico di XML.

**Closing (or End) Tag:** `</item>` Rappresenta la fine di un componente logico di XML.

**Empty Element Tag:** `<item />` Questi tag rappresentano entrambe le componenti logiche sopra che non contengono figli.

**Elemento**

Un elemento è un costrutto del ling di markup XML che può includere o un tag vuoto o un insieme di tag di apertura e chiusura.

```
<plucode>4012</plucode>
```

```
<plucode />
```

Se un elemento ha sia il tag di apertura che di chiusura, il testo in essi contenuto è chiamato il **contenuto** dell'elemento.

```
<plucode>4381</plucode>
```

Un elemento può anche contenere altri elementi tra i suoi tag di apertura e chiusura. Questi elementi vengono chiamati **child elements**.

```
<plucodes>
  <plucode>4012</plucode>
  <plucode>4381</plucode></plucodes>
```

**Attributo**

Un elemento XML può opzionalmente avere dei costrutti key-value che descrivono il tag. Questi costrutti sono chiamati attributi e forniscono metadata aggiuntivi riguardo uno specifico elemento XML.

```
<plucode description="Blood Orange">4381</plucode>
```

**Dichiarazione**

Ogni doc XML deve cominciare con una dichiarazione. La dichiarazione specifica info importanti riguardo il doc XML ib**version** and **encoding**:

```
<?xml version="1.0" encoding="UTF-8"?>
```

## HTML & XML

Both XML and HTML began as markup languages that built on SGML by either restricing language features or adding new features.

It was not until years later that a new HTML standard **(XHTML 1.0)**was created to apply the strict rules of XML to HTML. While this standard lasted for years, it was eventually replaced by the more modern and widely accepted **HTML5** standard.

**WEB BROWSER**

Porte di accesso:

- recuperano il contenuto dandogli un URL (Universal Resource Locator)
- mostrano il contenuto

se scrvi una app html accessbile devi assicurarti che funzioni su almeno il 95%

## HTML VERSIONS

The World Wide Web Consortium (http://www.w3.org) develops the various standards and protocols used throughout the web. One of their key jobs is to develop new and updated standards for the HTML markup language.

There have been many different version of HTML over the years. The table below highlights some of the versions of HTML and brief descriptions of each version.

| | |
|---|---|
| **SGML** | Standard Generalized Markup Language (SGML) |

| | |
|---|---|
| | paved the way for HTML. It was a simple way to structure text so that you can focus on content and semantics rather than style choices. SGML contained **paired tags** such as titles (<TITLE>), paragraphs (<P>), ordered/unordered lists (<OL> & <UL>), and headings (<H1>-<H6>). |
| **HTML 1.0** | Using some of the paired tags from SGML, the concept of **hypertext** links were added so that one document can reference another. Each anchor (<A>) element contained a **HREF** attribute that specified the location of another document. To address remote machines, a new specification was invented using the format "www.nameofotherplace.name". Not long after, early browsers such as Mosaic were released to interact with the early world-wide web. |
| **HTML 2.0** | After many individuals had an opportunity to interact with HTML, there were many ideas and revisions to the standard. The Internet Engineering Task Force created a working group to work on a new HTML spec that became HTML 2.0. It was updated with small modifications later adding features such as tables, image maps, internationalization and file uploads. Immediately prior to the release of HTML 2.0, the Netscape and Internet Explorer browsers were released. |
| **HTMl 3.0-3.2** | This version of HTML was released by the newly formed W3 Consortium and included input from many different browser vendors. This version included features such as frames, applets, wrapping text around images (flow), and additional text elements. Around this time other specifications such as scripting (JavaScript) and styles (CSS) were beginning to take shape. |
| **HTML 4.0-4.01** | This version of HTML removed a lot of browser-specific formatting markup in favor of using CSS to have a consistent look for web pages across various browsers. This version also introduced variations that allowed you to control how many deprecated elements (such as marquees and frames) were allowed on your web page. |
| **XHTML 1.0** | XHTML was a version that focused heavily on making sure that all HTML markup was compliant XML. Various XML features were rigorously enforced in order for your web page to render in a browser. While there are many XHTML web pages still on the internet, this standard is no longer being maintained. |
| **HTML5** | This is the latest version of HTML and the one you will use throughout this course. HTML5 included many more new semantic elements and attributes than previous versions. This version also took a lot longer than others to create as it dropped a lot of the SGML features and focused on creating a clear, forward-looking specification that could last for many more years. |


### HTML ELEMENT

The html element basically tells your computer that this is an HTML document.

To learn more about the HTML element, hover over the parts of the HTML markup (`underlined`) below.

```
<!DOCTYPE HTML>
<html xmlns="http://www.w3.org/1999/xhtml" n
US">
</html>
```

### BODY AND HEAD ELEMENTS

To learn more about the BODY and HEAD elements, hover over the parts of the HTML markup (`underlined`) below. You will learn more about JavaScript in the later modules

```
<head>
<title>Sample Page</title>
</head>
<body onload="handleLoad" ononline="handleNe
</body>
```

onload = attributo che indica una funzione js chiamata in caricamento

ononline = attributo che indica una funzione js chiamata quando una conenssione è disponibile

onresize = attributo che indica una funzione js chiamata qando si dimensiona pagina

TEXT

p => ogni paragrafo è distanziato da una riga vuota

ins => insertion alla fine è il sottolineato

del => deleted testo cancellato, è barratto

mark => per EVIDENZIARE del testo

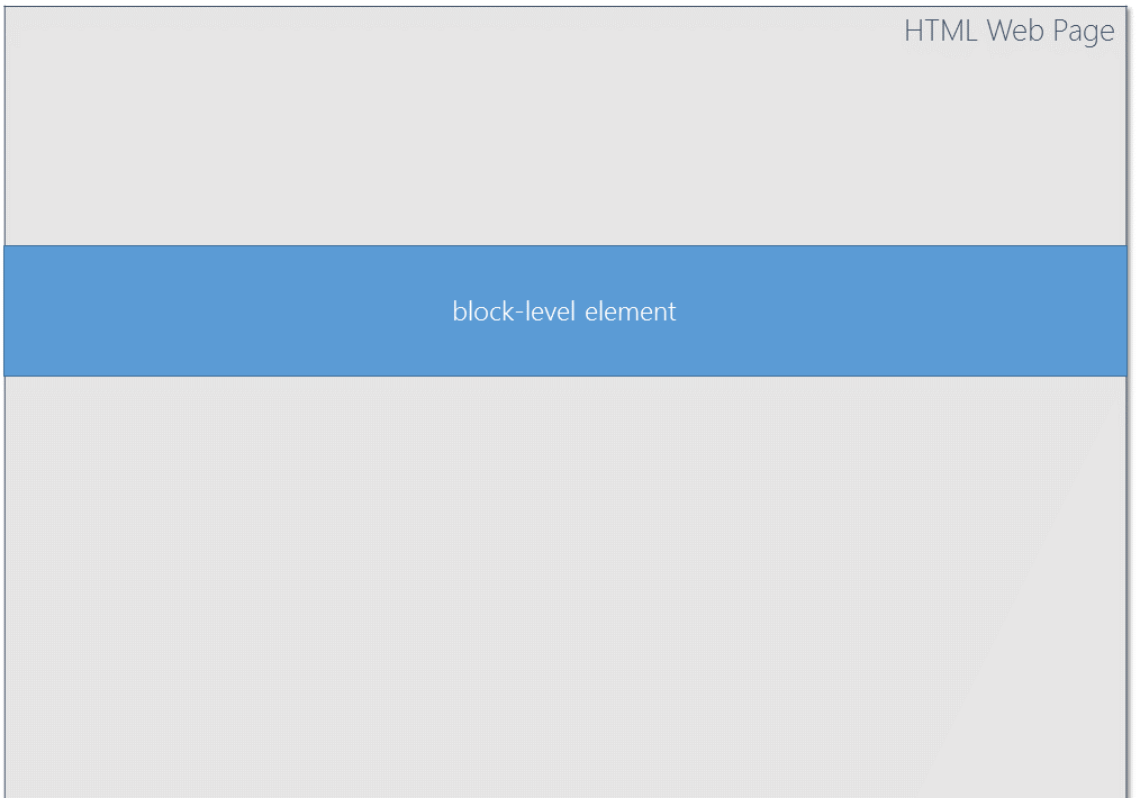
### BLOCK AND INLINE ELEMENTS

In HTML, there are two primary types of container elements: - **Block Elements** - **Inline Elements**

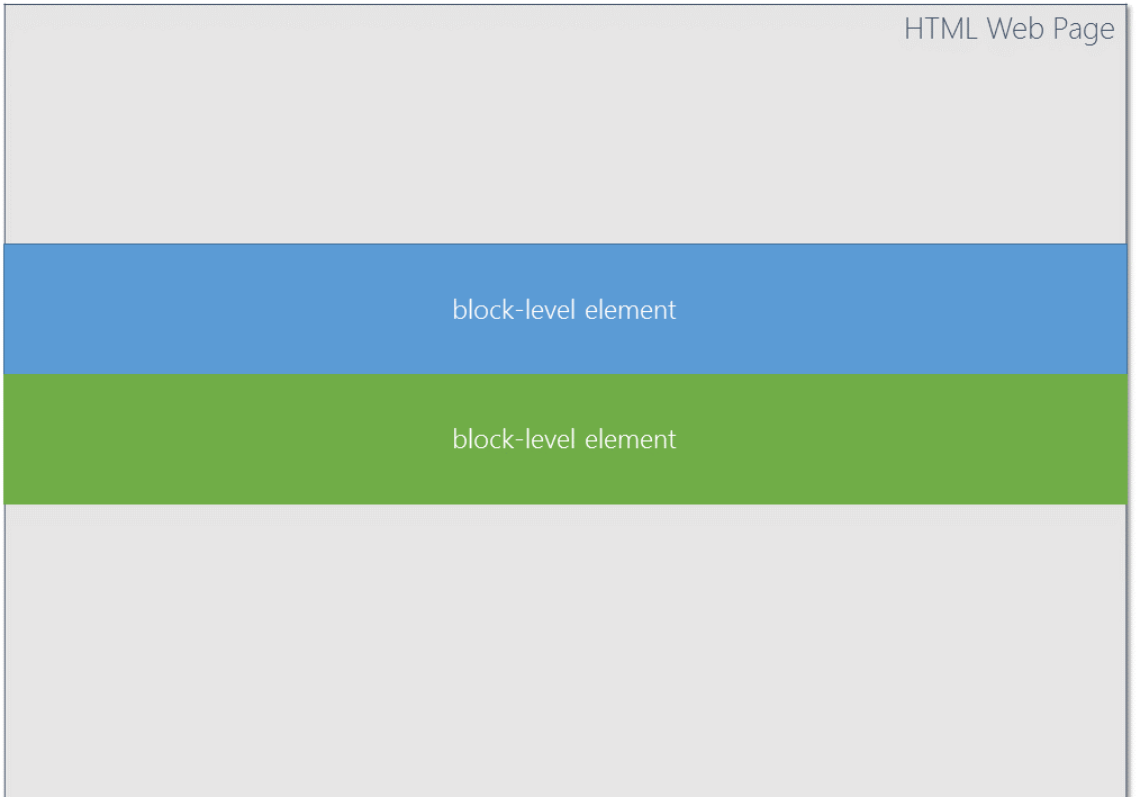
The type of element dictates what the default rendering logic will be for the content within that element or the element itself.

### BLOCK ELEMENTS

Block-level elements always take up the full-width that is available on-screen. By rendering using the full-width of the screen, the content of this element is rendered on a new line.




If you add another block-level element, the content of that element is also rendered on a new line.




There are three primary examples of block level elements that you may have used already in your HTML pages:

- <br> element
- <body> element
- <p> elements

## <br> Element

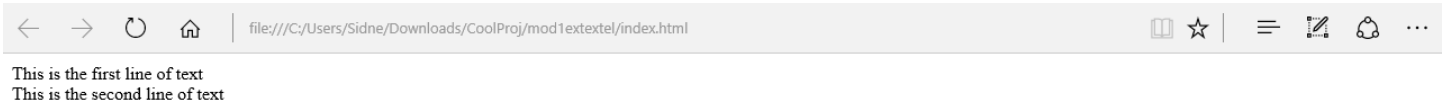
This element creates a single line break between text content. Since the line break takes up the entire width of the HTML page, the br element is considered a block element.

**Example**

**Code**

```
This is the first line of text
<br />
This is the second line of text
```

**Result**




This is the first line of text
This is the second line of text

##  and Other Block Elements

The  element is a multi-purpose element that simply gives you a container for other content. You can nest many different div elements within each other, you can create sibling elements with content that should be rendered on different lines or you can create complex nested block structures.

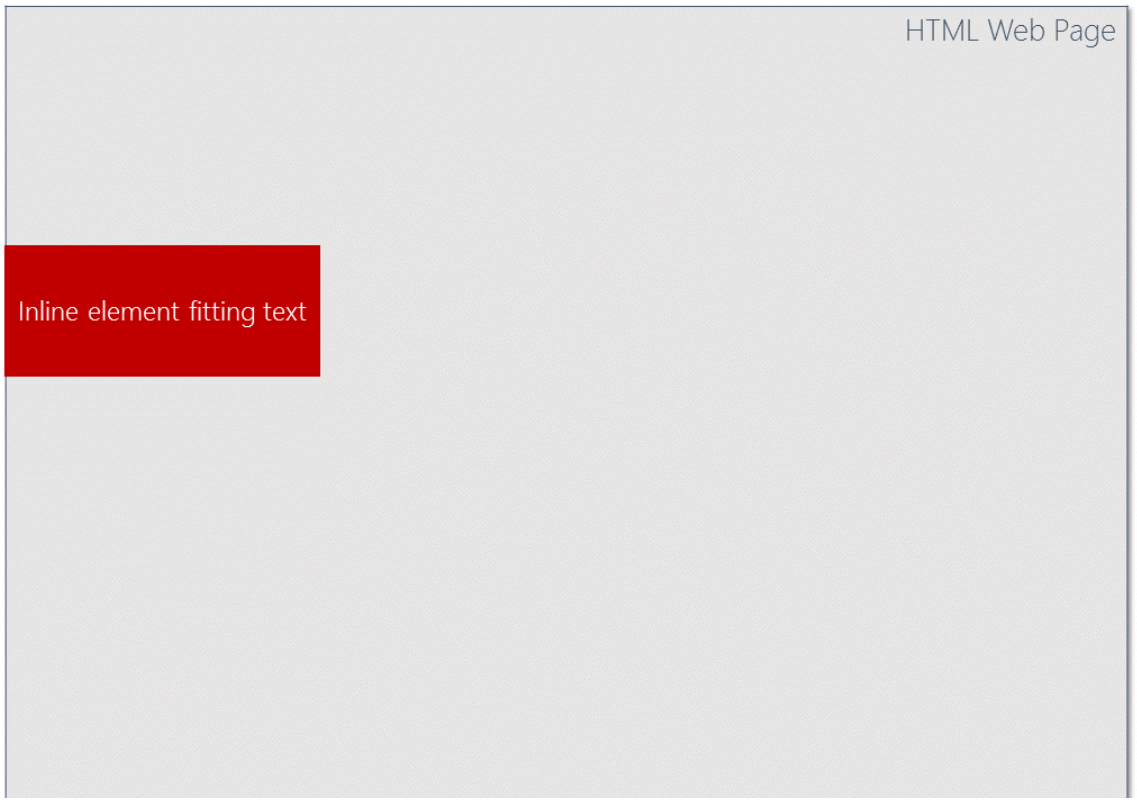
There are many other block elements including:

- article
- aside
- blockquote
- br
- button
- canvas
- caption
- dd, dl dt
- h1, h2, h3, h4, h5, h6
- footer, header
- fieldset
- form

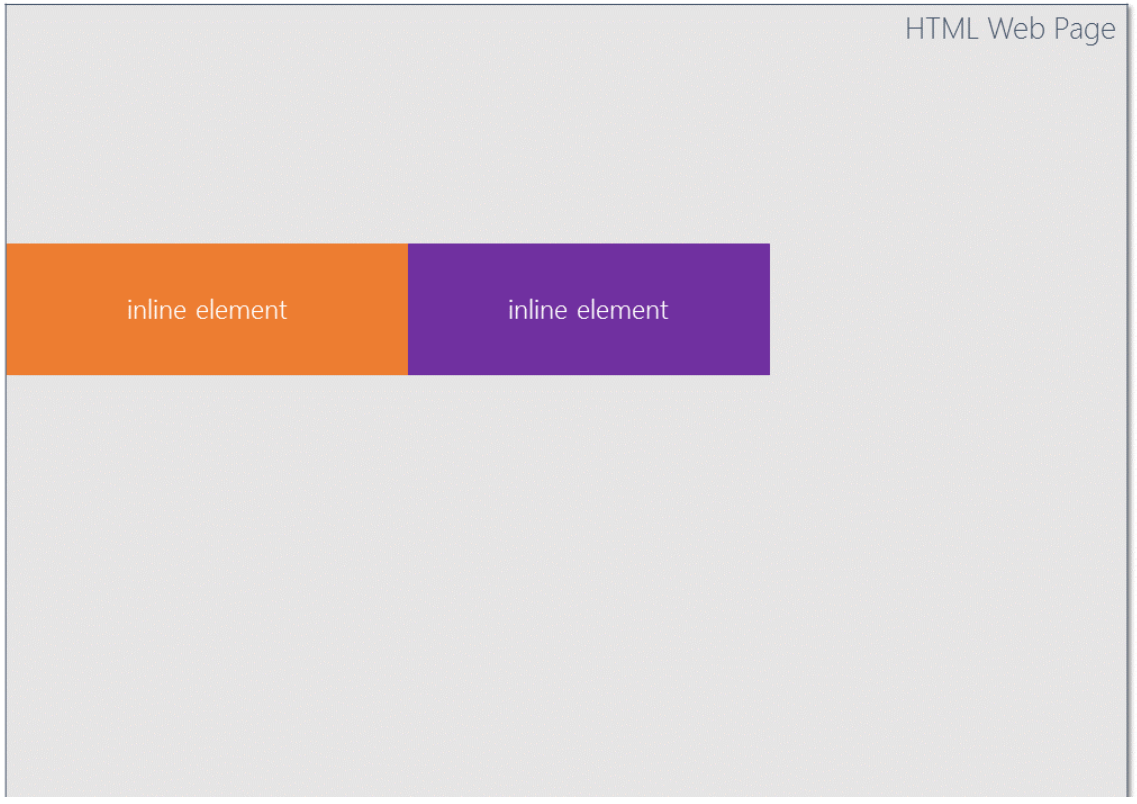
> **Note:** Elements such as **article**, **header**, and **footer** provide more semantic detail than a typical **div** element. When possible use these elements so that other developers can infer meaning from your HTML structure instead of guessing what you were trying to accomplish.

### INLINE ELEMENTS

Inline elements differ from block elements as they only take up the width that they need to render their content.




Multiple inline elements may render on the same line if they are not too large to be displayed on the same line.




If they are too large to fit on the same line, they can alternatively wrap their content to another line.


The most common inline element is the  element. There are many others including:

- strong
- ins
- del
- sup
- sub
- i
- em

## Mixing Inline Elements and Block Elements

Inline Elements and Block Elements can be used together as sibling elements within HTML content. Typically you can only place inline elements within a block element since an inline element will be able to fit within the entire width of the HTML page but a block element may not fit within the potential width of an inline element.

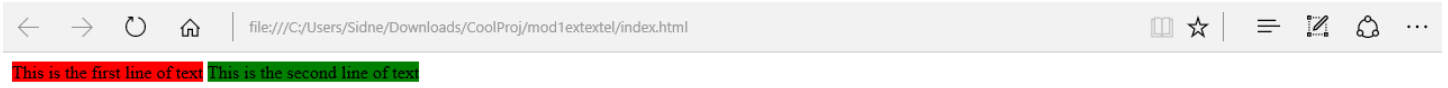
Using the **display** CSS attribute, you can change the rendering or any element between **block** of **inline** but it is not a good idea to change the way HTML elements are expected to behave on a page.

## Examples

**Code**

```
<span style="background-color: red;">This is the first line of text</span><span style="background-color: green;">This is the second line of text</span>
```

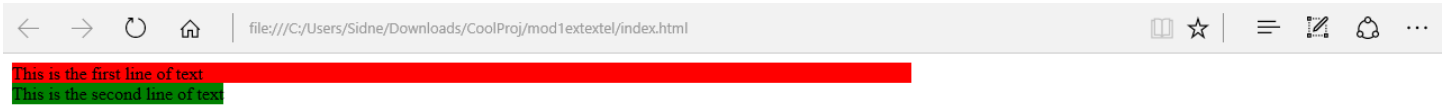
**Result**




**Code**

```
<div style="background-color: red; width: 800px;">This is the first line of text</div><span style="background-color:
green;">This is the second line of text</span>
```

**Result**



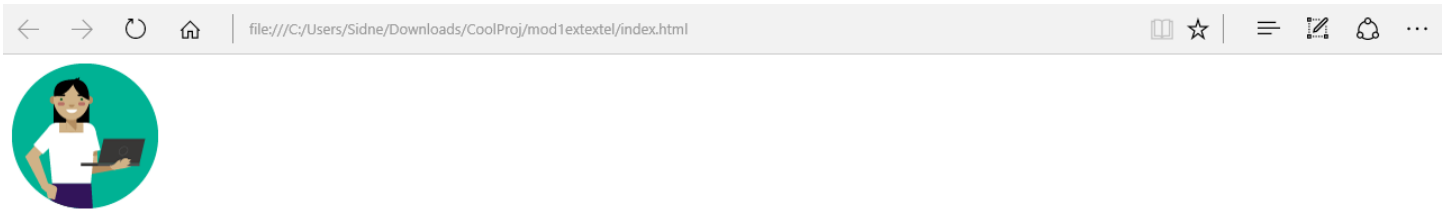

## IMAGES

In HTML images are represented using the <img> tag. This tag provides metadata about the image file and preferred sizing to help the browser decide how to render the image.

Let's look at a simple example that shows an image located in the relative **images** folder name **person.png**.

```
<img src="images/person.png" />
```

The browser will interpret this and render an image like such:

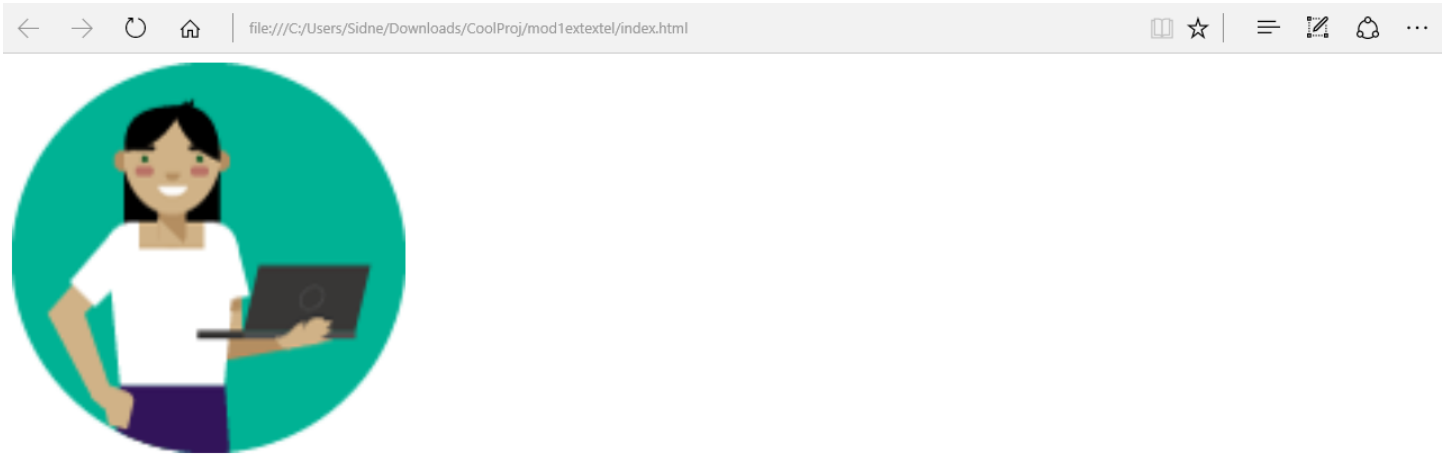



You can adjust the size of the image using the **height** and **width** attributes.

If you adjust one of the attributes, the image will change size while maintaining its aspect ratio. In this example, the image is 130x129 to start. If I set the **width** to 350 using the following HTML:

```
<img src="images/person.png" width="350" />
```

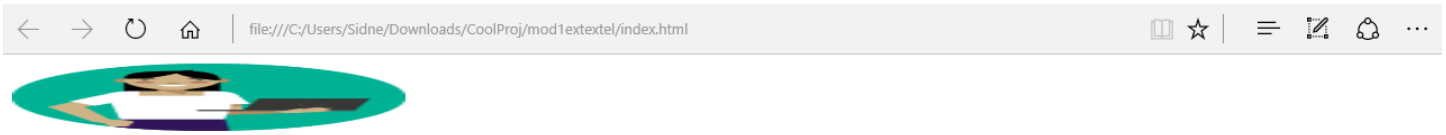
The image will stretch to a **height** of 347.30:




If you set both the height and width, the image will change to the exact size ignoring aspect ratio.

```
<img src="images/person.png" width="350" height="60" />
```

This will result in an image that looks "warped" (deformata) like in this example:






## Alternate text

You can set the alternate text for an image using the **alt** attribute:

```
<img src="images/person.png" alt="Person holding computer" />
```

This text is rendered in scenarios where the image cannot be displayed.

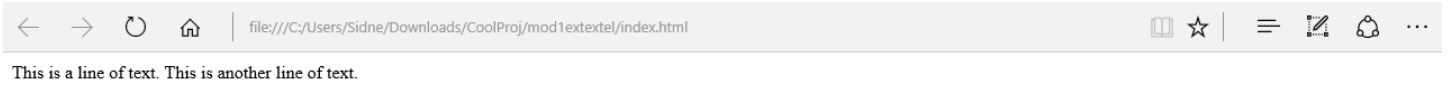
> **Accessibility**: Some modern browsers, such as those for the visually impaired, do not render images. It is important to specify alt text for these browsers as it will allow these visitors to understand what you intended to show in your image.

### FORMATTING ELEMENTS

In HTML, we can create text by simply writing the text to our web page:

```
This is a line of text. This is another line of text.
```

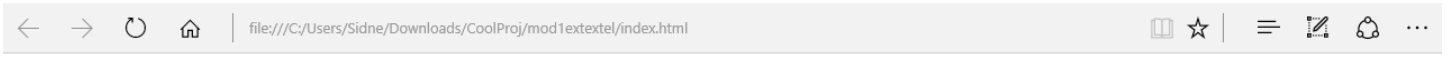
However, when we render this text, it will show as standard text in the default font for the browser:




This is a line of text. This is another line of text.

If we wish to change the appearance of the text, we can use various formatting elements. For example, the **<b>** element allows us to create bold text:

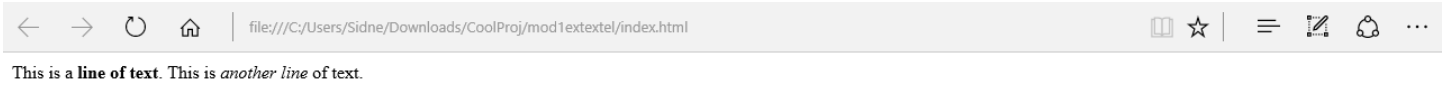
```
This is a <b>line of text</b>. This is another line of text.
```




This is a **line of text**. This is another line of text.

The **<i>** element creates italic text:

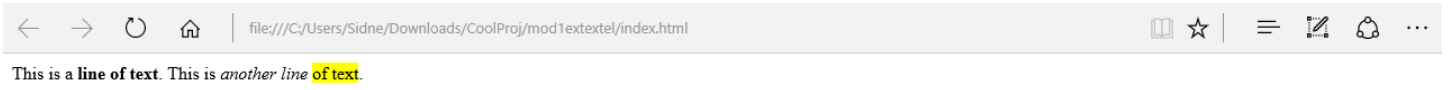
```
This is a <b>line of text</b>. This is <i>another line</i> of text.
```




This is a **line of text**. This is *another line* of text.

The **<mark>** element highlights a block of text:

```
This is a <b>line of text</b>. This is <i>another line</i> <mark>of text</mark>.
```




This is a **line of text**. This is *another line* of text.

You can use a variety of formatting elements together to change the way your HTML web pages appear in the browser.

## Semantic Formatting Elements

There are additional formatting elements that at first glance may appear to offer the same functionality.

For example, the `<b>` and `<strong>` tags both make a block of text appear bold.

These two blocks of HTML would be rendered identically in the browser:

```
This is a <b>line of text</b>.
```

```
This is a <strong>line of text</strong>.
```

The difference between these two elements lies in the **semantic meaning** of each element. The `<b>` tag indicates that a block of text should be rendered as bold text without any additional meaning. The`<strong>` tag indicates that the block of text semantically is "strong". The browsers render strong text as typical bold text but you could always change the way that "strong" text is rendered.

The same relationship is true for the `<i>` tag which defines italic text and the `<em>` tag which semantically represents "emphasized" text.

### FORMATTING ELEMENTS

There are a wide variety of elements that you can use in HTML to format your text blocks. A sample of these elements is listed below.

| Format | Markup | Result |
|---|---|---|
| Bold | `<b>` | **Bold** Text |
| Strong | `<strong>` | **Strong** Text |
| Italic | `<i>` | Italic Text |
| Emphasized | `<em>` | Emphasized Text |
| Small | `<small>` | Small Text |
| Marked | `<mark>` | Marked Text |
| Deleted | `<del>` | ~~Deleted~~ Text |
| Inserted | `<ins>` | Inserted Text |
| Subscript | `<sub>` | Subscript Text |
| Superscript | `<sup>` | Superscript Text |

#### HYPERLINKS

A webpage can contain various links that take you directly to other pages and even specific parts of a given page. These links are known as hyperlinks.

Hyperlinks allow visitors to navigate between Web sites by clicking on words, phrases, and images. Therefore, you can create hyperlinks using text or images available on a webpage.

In this section, you will learn how to create external and internal hyperlinks.

IPERTESTO è il concetto fondamentale con cui è nato HTML condividere documenti da diverse università o centri di ricerca mediante link

I vari documenti e risorse condivisi in tal modo oggi **formano il World Wide Web**

**target="_blank"** attributo dell'elemento a che dice come deve comportarsi il browser nell'apertura del link, di default il valore è **_self**

**rel = "author"** specifica a cosa CORRISPONDE IL LINK, che relazione c'è

**download="download"** attributo che serve per specificare se bisogna SCARICARE il conenuto del link invece di vederlo direttamente nel browser, il value è il nome del file scaricato

### INTERNAL HYPERLINKS

Not only can HTML reference external or remote documents (web pages), HTML can also reference elements within the same page.

Let's look at this example page:

```
<html>
  <head>
    <title>Sample Page</title>
  </head>
  <body>
```

```
    <p>Some content......</p>
    <p>More content......</p>
    <img src="images/interestingImage.jpg" alt="Interesting Image" />
  </body></html>
```

Your text content may be very long and require the user to scroll through the page. You may even wish for the user to be able to navigate directly to your image.

To create a hyperlink to a location within the same HTML page you must do two things:

You must create an id attribute for the target HTML element. The id attribute is universal and can be placed on all HTML elements:

```
<img id="coolImage" src="images/interestingImage.jpg" alt="Interesting Image" />
```

You can create an anchor tag (<a>) and specify the id as the target of this anchor tag. You must use a **number sign** immediately in front of the element id:

```
<a href="#coolImage">Navigate to Cool Image</a>
```

Now the hyperlink will jump directly to the image when someone clicks it.

## External Links to Internal elements

You can also use the same internal ids# when someone navigates to your web page in their browser.

For example, if your image has an id of **coolImage** you can give the user a URL such as http://www.mywebsites.com/index.html" and they will be navigated to your web page at the top of the home page (default behavior).

If you use this url: http://www.mywebsites.com/index.html#coolImage", they will be navigated to your web page but they will start at the location of your image instead of the top of the home page.

This technique is very useful for web pages with large quanitites of text where you want a user to "jump" to a specific section.

### INPUT CONTROLS

To build a form in HTML you can use a variety of input elements. Each input element offers options that allow you to control features such as validation or watermarks.

## <textarea> element

The **textarea** element renders a multi-line text input control. This control allows the user to enter large quantities of text that may wrap and span (avvolto e esteso su + righe) across multiple lines.

**Example**

In this example, we pre-populated the **textarea** element with text from the intro to Microsoft courses on edX.

```
<textarea rows="3" cols="30">
We live in a mobile-first and cloud-first world. Computing is ubiquitous, and experiences span devices and exhibit ambient intelligence.
</textarea>
```

## <select> element

THe **select** element renders a drop-down list with multiple options. A user can select one of the options. This element contains child **option** elements that allow you to designate (for each option) a value to be shown to the user and a value to be saved when the form is saved.

**Example**

In this example, we have listed some of the courses available on edX from Microsoft. The **value** attribute for each **option** element annotates the data that will be saved to the server when this form is saved. The content of the **option** element is displayed to the user.

```
<select>
    <option value="DAT202x">Processing Big Data with Azure HDInsight</option>
    <option value="DAT204x">Introduction to R Programming</option>
    <option value="CLD203x">Office 365: Managing Identities and Services</option>
    <option value="DEV208x">Introduction to jQuery</option></select>
```

Alternatively you can use the **optgroup** element to group related options:

```
<select>
    <optgroup label="Data Platform">
        <option value="DAT202x">Processing Big Data with Azure HDInsight</option>
        <option value="DAT204x">Introduction to R Programming</option>
    </optgroup>
    <optgroup label="Cloud">
        <option value="CLD203x">Office 365: Managing Identities and Services</option>
```

```
    </optgroup>
    <optgroup label="Development">
        <option value="DEV208x">Introduction to jQuery</option>
    </optgroup></select>
```

## <input> element

The **input** element is the most basic form element in HTML. It can be used to capture a wide variety of input including text and numeric values. This element has an **type** attribute that allows you to provide additional metadata about a specific input field. For example, you can designate a field to capture e-mails or URLs. The behavior of the web page may change depending on the browser or client device. For example, a mobile device may display a specific keyboard for URL input.

**Example**

Here is a basic form that captures a name and e-mail address:

```
<form ...>
  Username: <input type="text" name="username" />
  E-mail Address: <input type="email" name="emailaddress" /></form>
```

**Input Types**

There are a wide variety of types that can be used with the **input** element. This table shows the types and their respective functionality:

| Type | Functionality |
|---|---|
| button | This renders a button that is typically used with JavaScript frameworks (such as Angular or Knockout) |
| checkbox | This renders a checkbox that can be independently clicked. The checkbox has a value of on or off |
| file | This renders a field for file selection and a Browse button to select the file on the client device |
| hidden | This does not render anything but it will store a value. This can be used to store values that need to be sent to the server when the form is saved |
| image | This renders an image that functions as a submit button (click on image to save form) |
| password | This renders a text input where the characters are masked for privacy. |
| radio | This renders a radio button that can be selected as part of a group of radio buttons. |
| text | This renders a basic text input field. |
| submit | This renders a button that submits the HTML form. |

## <button> element

This special **button** element functions similarly to an **input** element of type button but it also allows you to render custom content within the button.

**Example**

In this example, we render the Microsoft logo within a button.

```
<button type="button">
    <img src="images/microsoft.png" alt="Microsoft" />
    <br />
    Apply to Microsoft!
</button>
```

<form>

</form>

serve per creare la form

dentro ad esso devi specificare una AZIONE a cui inviare form e il metodo di invio "Esempio POST o GET"

<form **action="" method=""**>

**INPUT ELEMENTS**

**ATTRIBUTI**

**name** => quando inviii i dati al server il nome verrà usato come key -value con il valore inserito nell'input della form

**value => il valore di default**

**placeholder**

organizzare GLI INPUT DI UNA FORM IN + SEZIONI CON IL

<fieldset></fieldset>

al cui interno come primo ele posso mettere una legenda

<legend>Valore legenda</legend>

FIELDSET USATI ANCHE DAGLI SCREENREADERS per ipovedenti

SUBMIT BUTTON

To learn more about HTML forms and submit buttons, hover over the parts of the HTML markup (`underlined`) below.

action => URL cui viene inviata form, un server locale o remoto che processa i dati

method => indica il metodo HTTP (GET o POST) usato per inviare dati

```
<form action="http://www.example.com/formsub
First Name: <input type="text" name="fname"
/> Last Name: <input type="text"
name="lname" /> <input type="submit"
value="Save Information" /> </form>
```

INPUT VALIDATION

HTML5 introduced field validation that applied to both existing HTML input controls and a new set of input controls.

## Input Attributes

Multiple attributes were introduced that allowed for the most common and simple validation requirements. If all validation requirements weren't met, the browser would simply not save your HTML form. There were also new attributes introduced that enhanced the functionality of forms in HTML5.

**Required**

The **required** attribute specifies that a text input must have valid data before the HTML form is saved:

```
<input name="email_address" type="email" required="required" />
```

**Pattern**

The **pattern** attribute specifies a regular expression that the value of the input field must match before the HTML form is saved:

```
<input name="zip_code" type="text" pattern="\d{5}(-\d{4})?" required="required" />
```

**Readonly**

The **readonly** attribute specifies that a particular field is readonly and cannot be modified by the user:

```
<input name="profile_url" type="url" readonly="readonly" />
```

**Disabled**

The **disabled** attribute specifies that a particular field is disabled and cannot be modified by the user:

```
<input type="submit" disabled="disabled" />
```

**Min/Max**

The **min** and **max** attributes are used with the **range** input type to specify boundaries for the selected numeric value:

```
<input name="issue_quantity" type="range" min="1" max="15" />
```

They can also be used with **date** inputs:

```
<input name="service_date" type="date" min="2000-01-01" max="2999-12-31">
```

**Autocomplete**

The **autocomplete** attribute is used with various text inputs to toggle (per attivare) the auto-complete feature found in most modern browsers.

```
<input name="api_key" type="text" autocomplete="off" />
```

**Placeholder**

The **placeholder** attribute renders a "watermark" that is shown when the input is empty. This hint text can be used to give the user further instructions for the specific field.

```
<input name="account_name" type="text" placeholder="Account name must contain at least 4 characters." />
```

## New Input Types in HTML5

HTML5 introduced a new set of input **types** that can be used in your HTML form for data capture validation without the need for additional code or JavaScript:

| Type | Functionality |
|---|---|
| color | This renders a color picker. |
| date | This renders a date control that allows you to select year, month and day without time. |
| datetime-local | This renders a date control that allows you to select year, month, day and time without time-zone information stored. |
| email | This renders a text input field and validates that the e-mail address is valid. |
| month | This renders a date control that allows you to select year and month only. |
| number | This renders a text input field that only allow numeric input. |
| range | This renders a control (typically a slider in most browsers) that allows a user to select an imprecise number. |
| search | This renders a text input field used for search. |
| tel | This renders a text input field used for telephone numbers. |
| time | This renders a date control that allows you to enter time without time-zone information stored. |
| url | This renders a text input field and validates that the url is valid. |
| week | This renders a date control that allows you to select year and week only. |

## Skipping Validation

If you would rather (invece) handle validation manually or using your own code, you can always skip validation using the **novalidate** attribute on a HTML form:

```
<form novalidate="novalidate"></form>
```

HTML FORMS

Modern HTML does more than just display content and link documents. Today, the web allows you to perform transactions, sign-up for services and interact with other people. The root of much of this functionality is centered in the Input functionality for HTML. In HTML input is handled using three main concepts.

In HTML, the <form> element is used to create an HTML form where you can capture user input. Within an HTML form, you can use a variety of input elements to construct your form. Once you are done, you can simply add a special input element called a **submit** input. This special input element renders a button that will save the contents of the form to a web server or web service.

To create a HTML form, you start by creating a **form** element:

```
<form action="http://www.example.com/formsubmission" method="POST"></form>
```

The form element has two primary attributes: - **action:** This is the URL of the web server or web service where you want you form data saved. - **method:** This is the HTTP method you are using to save your form data. Only GET and POST are valid in this context.

Inside of the form element, you can add as many input controls as necessary. Once you are done, you create a submit input to complete the form. A basic form may look like this:

```
<form action="http://www.example.com/formsubmission" method="POST">
    First Name: <input type="text" name="fname" />
    Last Name: <input type="text" name="lname" />
    <input type="submit" value="Save Information" /></form>
```

You can also optionally add **fieldset** and **legend** elements. These help you group your form into categories to make it easier for others to complete the form. Using **fieldsets** you can create a form with distinct sections. Each **fieldset** can optionally have a **legend** that displays as a label for the section. An example of multiple **fieldsets** and **legends** would look like this:

```
<form action="http://www.example.com/formsubmission" method="GET">
    <fieldset>
        <legend>User Information</legend>
        First Name: <input type="text" name="fname" />
        Last Name: <input type="text" name="lname" />
    </fieldset>
    <fieldset>
        <legend>Location</legend>
        Hometown: <input type="text" name="home" />
    </fieldset>
    <input type="submit" value="Submit Information" />
</form>
```


## SEMANTIC ELEMENTS

In HTML you can place all of your content into **p** or **div** elements. While this is convenient, you will quickly find that these elements do not have any obvious meaning to other developers or designers. These elements are known as **non-semantic elements**.

A traditional fix for **non-semantic elements** was to provide a large amount of comments and very descriptive CSS class names. This presents two common problems:

> CSS class names were never designed to describe the content of a HTML element. They are too short to have a real impact.
>
> HTML comments can greatly increase the size of your web page. Making your website more efficient is critical in today's modern world of mobile browsers and thin clients.

### HTML5 Semantic elements

HTML5 introduced a new series of **semantic elements** that function identically to other block-level containers such as the **div** element. These elements have meaning that others can infer when managing and modifying your web application.

| Semantic Element |
| --- |

| |
|---|
| Header |
| Footer |
| Nav |
| Section |
| Article |
| Aside |
| Details |
| Figure |
| Figcaption |
| Main |
| Mark |
| Summary |
| Time |

These elements can be used anywhere where you would normally use a block-level element without requiring any additional configuration or customization.

**Machine Inference**

The semantic elements are unique because they have meaning that machines can infer from your web application. For example, the **article** element signals to a crawler that this section contains text information that is worth scanning. A search index crawler may not care what content is contained in the **header** or **nav** elements. This not only makes search engines more efficient, this can open up scenarios where you can parse a HTML web page for information. For example, you may write a JavaScript extension that reads a web page, finds any **time** semantic elements and display them using a special panel.

**Design**

Another advantage of semantic elements is the ability to use CSS to design the element directly without requiring additional CSS classes. You can use features such as flexbox in CSS to put an **aside** element on the right-hand side of your webpage. You can also use features such as grid in CSS to place the **nav** element on a row by itself. More importantly, you can use features such as CSS Media Queries to re-arrange your semantic elements for a mobile browser.

**Note**: In the next module we will dive deeper into various **CSS** features.

**MICROFORMATS**

Elemento TIME

permette di specificare delle date, per spiegare al browser che stiamo specificando una data

<time datetime="aaaa-dd-mmThh:mm:ss:ms">Stringa data

</time>

il time da un VALORE SEMTANTICO al contenuto, viene visto come data e usato per i search engine

un vantaggio dei microformats e che puoi aggiungere **JS**

**FIGURE ELEMENT**

Costrutto che ti serve per wrappare una immagine e darle una Caption che la descriva

```
<figure>
            <img src="img/figure_elements_journey.png" alt="Journey" />
            <figcaption>
                This is how we get there.
            </figcaption>
        </figure>
```

SVG

Scalable Vector Graphics (SVG) is an image format that can draw 2D graphics directly in the browser. SVG is based on XML and supports features such as interactivity, transitions and animations. SVG was one of the many standards designed by the World Wide Web Consortium (W3C). Typically SVG graphics are stored in XML files and edited using vector-based image manipulation programs. May browsers can display SVG graphics in a manner similar to how they display other image formats.

HTML5 introduced the ability to embed SVG graphics directly in web pages. Now you can create SVG shapes in your HTML web page and manipulate them directly with CSS or JavaScript.

## SVG Shapes

HTML5 introduced some basic SVG shapes that can be used in a wide variety of scenarios. These shapes offer simple attributes that allow you to create SVG graphics quickly without the need to learn the syntax for drawing paths. You can combine various SVG shapes to create complex objects such as logos. See the example below:

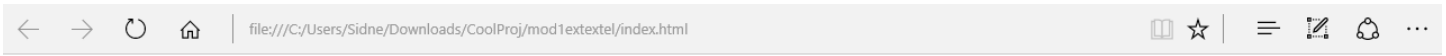
```
<svg height="200" width="400">
    <rect x="100" y="50" rx="20" ry="20" width="250" height="100" fill="#1B043C" />
    <rect x="100" y="50" width="200" height="100" fill="#1B043C" />
    <circle cx="100" cy="100" r="50" fill="#472772" />
    <text fill="#D7BFF3" font-size="28" font-family="Segoe UI Light" x="160" y="108">CompanyName</text></svg>
```


**Circle**

An SVG circle element creates a simple circle with a defined center point and radius:

| Attribute | Description |
|---|---|
| cx & cy | These two attributes together define the coordinates for the center of the circle. By default, the center of the circle is (0,0) |
| r | This attribute specifies the radius of the circle |
| fill | This attribute defines the color used for the interior of the circle |
| stroke | This attribute defines the color used for the border of the circle |
| stroke-width | This attribute defines the width of the border of the circle |

```
<svg height="200" width="200">
    <circle cx="100" cy="100" r="50" fill="#6CBEE2" /></svg>
```

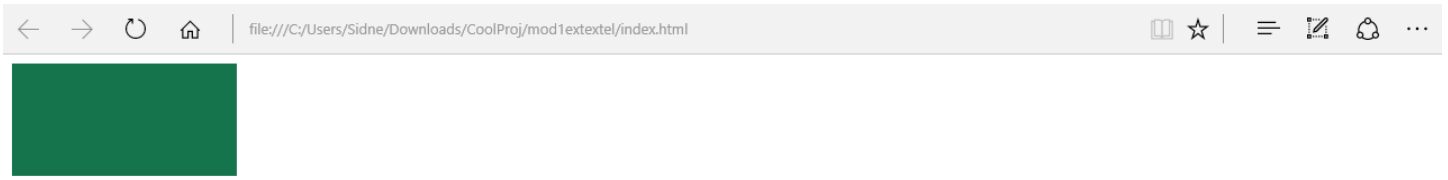



**Rectangle**

A SVG rectangle element creates a single 4-sided rectangular polygon that is drawn from a defined top-left point, width and height:

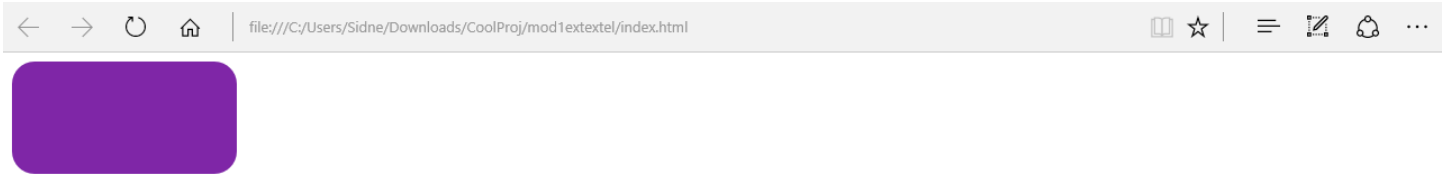
| Attribute | Description |
|---|---|
| x & y | These two attributes together define the coordinates for the top-left of the rectangle. By default, the top-left of the rectangle is (0,0) |
| rx & ry | These two attributes specify the radius for rounded corners on the x or y axis |
| width | This attribute defines the width of the rectangle |
| height | This attribute defines the height of the rectangle |
| fill | This attribute defines the color used for the interior of the rectangle |
| stroke | This attribute defines the color used for the border of the rectangle |
| stroke-width | This attribute defines the width of the border of the rectangle |

```
<svg height="200" width="200">
    <rect x="0" y="0" width="200" height="100" fill="#15744C" /></svg>
```




You can also optionally specify rounded corners for the rectangle:

```
<svg height="200" width="200">
    <rect x="0" y="0" rx="20" ry="20" width="200" height="100" fill="#7F26A7" /></svg>
```



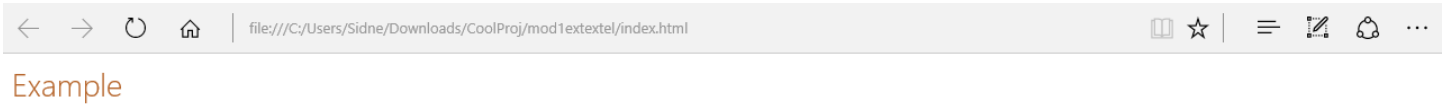

**Text**

A SVG text element renders text to the screen from a defined bottom-left point.

| Attribute | Description |
|---|---|
| x & y | These two attributes together define the coordinates for the bottom-left of the text. |
| fill | This attribute defines the color used for the text |
| font-size | This attribute defines the size of the text font |

| | |
|---|---|
| font-family | This attribute defines the text font |

```
<svg height="200" width="200">
    <text font-size="28" font-family="Segoe UI Light" x="0" y="30" fill="#B6652A">Example</text></svg>
```




SVG ELEMENT

To learn more about the SVG element and some of its shapes, hover over the parts of the HTML markup (`underlined`) below.

svg => elemento root usato per creare le forme svg, sarebbe la tavola da lavoro e deve avere dimensioni che contengano le altre forme

```
<svg height="200" width="400">
<rect x="100" y="50" rx="20" ry="20"
width="250" height="100" fill="#1B043C" />
<rect x="100" y="50" width="200"
height="100" fill="#1B043C" />
<circle cx="100" cy="100" r="50"
fill="#472772" /> <text fill="#D7BFF3"
font-size="28" font-family="Segoe UI Light"
x="160" y="108">CompanyName</text> </svg>
```

HTML MEDIA PLAYBACK

Out of the box, HTML5 supports basic multimedia playback with a new set of media elements that are supported cross-browser. These media elements allow you to specify video/audio source[s], along with some parameters to play video or audio directly on your HTML page.

You can specify multiple sources for your multimedia so that each browser can play the files that it natively support. You can also style the media player using CSS to fit your design needs. You can even control which aspects of the media player are available using attributes such as `controls`.

## Video Element

The video element in HTML is used to play video files:

```
<video height="500" controls="controls" poster="screenshot.png">
    <source src="advertisement.webm" type="audio/webm" />
    <source src="advertisement.ogg" type="audio/ogg" />
    <source src="advertisement.mp4" type="audio/mp4" /></video>
```

## Audio Element

The audio element in HTML is used to play audio files:

```
<audio autoplay="autoplay" controls="controls" loop="loop">
    <source src="popopen.wav" type="audio/wav" />
    <source src="popopen.ogg" type="audio/ogg" />
    <source src="popopen.mp3" type="audio/mpeg" /></audio>
```

HTML VIDEO/AUDIO FORMATS

For HTML Video, there are five primary video/audio formats that are used throughout major industry browsers:

## HTML Video Formats

| Format | Media Type | Edge | Internet Explorer | Chrome | Firefox | Safari | Opera |
|---|---|---|---|---|---|---|---|
| MP4 | video/mp4 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| WebM | video/webm | | | ✓ | ✓ | | ✓ |
| Ogg | video/ogg | | | ✓ | ✓ | | ✓ |

## HTML Audio Formats

| Format | Media Type | Edge | Internet Explorer | Chrome | Firefox | Safari | Opera |
|---|---|---|---|---|---|---|---|
| MP3 | audio/mpeg | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Ogg | audio/ogg | | | ✓ | ✓ | | ✓ |
| Wav | audio/wav | | | ✓ | ✓ | ✓ | ✓ |

## Supporting Multiple browsers

You can always define multiple sources for the video or audio control to handle the most browsers possible: The sources will be attempted in order from the top to the bottom:

```
<audio controls="controls">
    <source src="audio/music.ogg" type="audio/ogg" />
    <source src="audio/music.wav" type="audio/wav" />
    <source src="audio/music.mp3" type="audio/mpeg" /></audio>
```

```
<video controls="controls">
    <source src="video/movie.webm" type="video/webm" />
    <source src="video/movie.ogg" type="video/ogg" />
    <source src="video/movie.mp4" type="video/mp4" /></video>
```

## Handling Unsupported browsers

The **audio** and **video** elements in HTML5 allow you to handle Unsupported browsers by displaying HTML content if the video/audio player is not available:

```
<audio controls="controls">
    <source src="audio/music.ogg" type="audio/ogg" />
    Sorry, your browser does not support the OGG format.
</audio>
```

```
<video controls="controls">
    <source src="video/movie.webm" type="video/webm" />
    Sorry, your browser does not support the WebM format.
</video>
```

### WHY CSS?

Cascading Style Sheets (CSS) is a language that controls and allows one to define the look of an HTML document. It permits the separation between the content of the HTML document from the style of the HTML file. CSS enables one to specify things such as the font you want on your page, the size of your text, the columns of the page, whether the text on a page is to be in bold or italics, background, link colors, margins, and placement of objects on a page and so on. by way of explanation, it is the part that controls the looks of a web page. With CSS, it is much easier to manage the appearance of multiple web pages since it separates the HTML element from display information. CSS also enables faster downloading of web pages, which is works best with older computers and modems. It provides a method for retaining a common style.

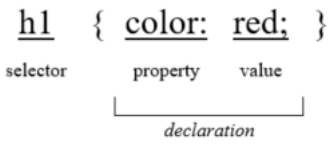
The coding of CSS style rules can be done in three places, namely:

- **Inline** - done in the HTML tag.
- **Internal Style Sheet** - coded at the beginning of a HTML document i.e. inside the <head></head> tags, and closed by the <style type="text/css"> </style> tags.
- **External Style Sheet** - this is a separate file with a .css extension which serves as a reference for multiple HTML pages with a path/link in the HTML pages pointing to browsers where to look for the styles.

### CSS SYNTAX

CSS has two parts to a style rule.

- **CSS selectors** - this is the core foundation of CSS since it defines the HTML element being manipulated with the CSS code.
- **The declaration**- consists of one or more **property** (is the CSS element being manipulated) **value** (represents the value of the specified property) pairs, usually ends in a semi-colon and enclosed in curly brackets. For example,




::before

indica delle rule applicabili prima del selettore

::after

indica delle rule applicabili dopo il selettore

### HTML ELEMENT STYLES

To learn more about styles that target HTML elements, hover over the parts of the CSS markup (`underlined`) below.

```
body { font-weight: bold; } header, footer { background-color: black; color: white; } section > nav { background-color: darkgray; } section article { background-color: lightgray; }
```

### CSS IDS

A CSS ID is a unique identifier for an element which can only be declared once within the same HTML file and is normally very specific, unlike classes which can be declared multiple times in the same HTML file. CSS Styles IDs is used to style the layout elements of an HTML page and is usually preceded by a hash sign (“#”). CSS can be instructed to grab onto the ID attribute, so as to style an element with particular ID in a certain way.

The CSS style IDs are customized styles that are created to work with div tags. All content inside that div tag follows the style rules set by the customized style. This is done by creating declaration set in the CSS.

IDS are usually powerful tool in CSS since an IDs element can be the target of a piece of JavaScript that manipulates the element or its contents in some way. The ID attribute can be used as the target of an internal link, replacing anchor tags with name attributes. Clear and logical IDs can also serve as a sort of “self-documentation” within the document. For example, one does not necessarily need to add a comment before a block stating that a block of code will contain the main content if the opening tag of the block has an ID of "main" or "header".

IDs are used as fragment identifiers (an href that ends in #anchor directs one to id="anchor"), and for JavaScript’s `getElementById`.

### SELECTING CHILDREN ELEMENTS IN STYLES

In CSS, we can select elements that are within other elements (children) using two special selectors.

For the following examples, we are going to use the following HTML content:

```
<body>
    <section>
        <article>
            <p>First Paragraph</p>
            <aside>
                <p>Second Paragraph</p>
            </aside>
        </article>
        <p>Third Paragraph</p>
    </section></body>
```

## Indirect Descendant Selector

This selector selects any element that is a descendant of the first element specified.

```
section p {
  font-family: monospace;}
```

In this example, any **P** element within the section element's tags will have this style applied. In our sample HTML, all three paragraphs will have this style applied.

## Direct Descendant Selector

This selector selects only elements that are a direct descendant of the first element specified. This selector is useful if you want to narrow the scope of your style only to a specific descendant.

```
section > p {
  font-weight: bold;}
```

In this example, only the "Third Paragraph" **P** element will have this style applied.

You can nest multiple direct selectors if you want to "drill" deeper and select another **P** element.

```
section > article > aside > p {
  font-weight: bold;}
```

In this example, only the *"Second Paragraph"" **P** element will have this style applied.

You can also combine the selectors.

```
section > article p {
  font-style: italic;}
```

In this example, both the "First Paragraph" and "Second Paragraph" **P**elements will have this style applied.

### INHERITING CSS STYLES

Inheriting CSS Styles is the process by which styling rules of a parent element are applied (inherited) by multiple elements thus (in modo che) making the children of the parent element inherit the characteristics of its parent as far as these properties are concerned. However, some CSS properties are automatically inherited, some like margins and borders cannot be inherited since they may conflict with the elements that are already set to the child elements.

With inheritance, it is easy to specify e.g. font properties for html elements, and they will automatically be inherited by all other elements.

The inheritance mechanism is separate from the cascade in that inheritance only applies to the Document Object Model (DOM) tree, while the cascade is concerned with the style sheet rules. However, CSS properties attached to an element via cascade can be inherited by the child elements of the parent.

All elements in an HTML document inherits every inheritable property from the parent except an HTML element which does not have a parent. As much as most characteristics can be inherited automatically in CSS, there are situations in which one may want to increase the weight of the property inherited. Inherit property value makes this possible. It enables one to increase the weight of the property inherited by adding it to the author style sheet.

Inheritance, therefore, makes it possible to set characteristics to multiple elements without having to specify element property for every single element type thus making it possible to apply styles to the tag that was not directly applied to the particular HTML tag.

### CSS STYLE INHERITANCE

CSS styles can come from many different places:

- **Browser (User Agent)**: Browser's have default style sheets that determine how various pages and elements are rendered for the user.

  **User Options**: User's may apply specific options to their browser such as specific font sizes for readability.

  **Author CSS**: The CSS that you create for a HTML page.

When you apply CSS styles to a HTML page, these styles join an ascending order of inheritance It is important to understand where your content exists in this order of inheritance. Items at the bottom of this table are considered "more important" and will override items that proceed them:

| Source | Description |
|---|---|
| Browser stylesheet | All browsers have a default stylesheet and this style is used when no other CSS styles are specified |
| User settings | When a user configures specific settings in a browser, the visual settings are usually preserved as a CSS stylesheet or set of CSS declarations. These styles override the ones set by default in the browser. |
| Author CSS | The declarations created in stylesheets (inline or external) by the author of the HTML page override both the user settings/declarations and browser settings. |

## !important keyword

In CSS you can use the keyword **!important** to override existing CSS declarations (styles) and skip to the front of the order of inheritance.

In this example, the CSS author would like to ensure that font color is red and that no other author CSS files change that:

```
body {
    color: red !important;}
```

Users can also optionally specify CSS declarations (styles) using the !important tag if they want to override the author of the HTML page. This is especially critical for scenarios where users may have issues font size (visual impairment) font colors (colorblindness) or otherwise may be unable to read a page.

Considering the !important tag, this adds two more sources to our table that override the previous three sources:

| Source | Description |
|---|---|
| Author**!important**declarations | These declarations override any existing author CSS styles on the HTML page. |
| User**!important**declarations | These declarations override all other sources of CSS styling and are considered the most "important" declarations. |

### MARGIN AND PADDING

## Margin

The margin property in CSS defines the space between an HTML element and the surrounding element. It defines a shorthand property for setting the margin properties in one declaration.

The margin elements possess the four main properties that can be set on the top, left, right and bottom of an element. Margin attributes can be defined using actual values, and can be defined with the use of percentages as well. The four main properties are listed below.

• The **margin-bottom** defines the bottom margin of an element.

• The **margin-top** defines the top margin of an element.

• The **margin-left** defines the left margin of an element.

• The **margin-right** defines the right margin of an element.

When defining the attribute of the CSS margins, there are several methods to go about it.

**CSS margin:** 4 values If all the four values are declared i.e. margin: 10px 10px 10px 10px;, corresponding order will be as follows; top, right, bottom, left.

Four margin values can be defined at once by specifying both two and four values. When only employing two values, the first will define the margin on the top and bottom whereas the second value will define the margin on the left and right.

**CSS margin: value** - Margin can be consistent outside an element. If only one value is declared i.e. margin: 10px;, it sets the margin on all sides: top, right, bottom, and left. Hence creating a uniform margin on all four sides.

**CSS margin: margin- (direction):** usually HTML element contains four different margins namely top, right, bottom, and left. The individual margins can be defined by adding a direction suffix to the margin attribute. This means that when one direction is defined other three margins are left intact.

Margin property can also be set to negative values. The default value of a margin if not declared is zero. However, there are elements such as paragraphs which have default margins in some browsers; this can be overcome by setting the margin to 0 (zero).

## Padding

The Padding property in CSS defines the amount of space that should appear between the content of an element and the element border. With CSS Padding, it is possible to change the default padding that is within various HTML elements such as paragraphs, tables, etc. CSS padding rules are more or less the same as margin rules, the only difference being, negative and auto values cannot be declared for padding.

The padding elements possess the four main properties that can be set on the top, left, right and bottom of an element. Padding attributes can be defined using length, a percentage, or the word inherit. The four main properties are listed below.

• The **padding-bottom** defines the bottom padding of an element.

• The **padding-top defines** the top padding of an element.

• The **padding-left defines** the left padding of an element.

• The **padding-right defines** the right padding of an element.

When defining the attribute of the CSS padding, there are several methods to go about it.

**CSS padding: 4 values** If all the four values are declared i.e. padding: 10px 10px 10px 10px;, corresponding order will be as follows; top, right, bottom, left.

Four padding values can be defined at once by specifying both two and four values. When only employing two values, the first will define the margin on the top and bottom while the second value will specify the margin on the left and right.

**CSS padding: value** - Padding can be consistent outside an element. If only one value is declared i.e. padding: 10px;, it sets the padding on all sides: top, right, bottom, left. Hence creating a uniform padding on all four sides.

**CSS padding: padding- (direction)**-Usually HTML element contains four different paddings namely top, right, bottom, and left. The individual paddings can be defined by adding a direction suffix to the padding attribute. This means that when one direction is defined other three paddings are left intact.

i border-styles (solid, insect, ridge ...) possono esser combinati cioè usare fino a 4 value dentro una sola pty

### MEDIA QUERIES

CSS Media Queries allow us to conditionally apply CSS in specific scenarios based on properties of our current browser. Using these Media Queries, we can provide a custom experience that is responsive and that reacts to changes in our user's browser environment such as resizing, print preview or switching to a screen reader.

## Examples

> It is far easier to use examples to describe Media Queries so we will explore some examples below:

## Media Types

There are various types of media including monitors (display screens), screen readers and even printers. Media queries can discern between these media types and apply appropriate CSS styles for each type.

- This media query is applied only if the user is looking at the browser through a screen (desktop, tablet, phone, laptop, etc.). This is the most common scenario. In this example, we have a lively background color of red for our web application.

```
@media screen {
    body {
        background-color: red;
    }}
```

- This media query is applied only if the user is attempting to print the page. In this example, we would like to change the background color of the web page to white to give a "more traditional" print appearance.

```
@media print {
    body {
        background-color: white;
    }}
```

- This media query is applied only if the user is using a screen reader. In this example, we may be displaying visual ads in our **aside** element. We can use this rule to not render the ad images or videos for screen readers.

```
@media speech {
    aside {
        display: none;
    }}
```

## Media Widths

Media Queries can be used in scenarios where your screen may be of a specific size. Using a width or height query, we don't have to design for specific devices. Instead, we can make sore our design looks appropriate at specific screen widths and allow the media query to select the correct design based on a screen width.

- In this example, we look at a media query that is applied ONLY if the browser is at least 500 pixels wide. In this example, we set the width of our nav element to 100 pixels if the screen is at least 500 pixels wide.

```
@media (min-width: 500px) {
    nav {
        width: 100px;
    }}
```

- You can use the same type of expression for browser height.

```
@media (min-height: 500px) {
    nav {
        height: 300px;
    }}
```

- In this example, we can make the nav element smaller (50 pixels) on browsers where the width is less than or equal to 500 pixels.

```
@media (max-width: 500px) {
    nav {
        width: 50px;
    }}
```

## Simple Media Query Logic

Media queries can contain simple logic using **AND** and **OR** expressions.

- In this sample, we can apply CSS styling if the width of the browser is between 500 and 1000 pixels. The **and** keyword represents the **AND** logical operator. We make the font size relatively larger in this scenario.

```
@media (min-width: 500px) and (max-width: 1000px) {
    header {
        font-size: larger;
    }}
```

- We can also using a comma to represent the **OR** logical operator. In this example, we make the footer bold if the browser is less than 500 pixels wide or more than 1000 pixels wide.

```
@media (min-width: 1000px), (max-width: 500px) {
    footer {
        font-weight: bold;
    }}
```

Finally, we can combine multiple **AND** and **OR** expressions to create complex rules.

```
@media screen and (min-width: 500px) and (max-width: 500px) {
    article {
        font-style: italic;
    }}
```

display: flex mostra degli aside invece che come blocchi come inline

Add the following CSS rule for the **nav > aside** selector to give each aside element equal space in the flexbox layout:

```
nav > aside {
flex: 1;}
```

Add a new media query that is active when the web page is being printed:

```
@media print {}
```

ADDITIONAL MEDIA QUERY FEATURES

## Link Element

You can use a media query with **LINK** elements. This allows you to separate your stylesheets for each platform and modularize your CSS. Additionally, the browser will only retrieve the relevant CSS files based on media query.

For example, you can use a media query with three stylesheets:

```
<head>
  <link rel="stylesheet" href="base.css" />
  <link rel="stylesheet" href="mobile.css" media="screen and (max-width: 1000px)" />
  <link rel="stylesheet" href="print.css" media="print" /></head>
```

In this example, the mobile stylesheet is only retrieved if we are using a device screen and the browser's width is less than or equal to 1000 pixels. The print stylesheet is only retrieved if we are attempting to print the web page.

## Additional Keywords

Using various keywords, you can implement additional logic to have media queries that cover a wide range of scenarios.

**All Keyword**

The all keyword specifies that a media query should be used with all device types (screen, speech, print):

```
@media all {
    body > article {
        font-family: serif;
    }}
```

**Not Keyword**

The **not** keyword inverses the logic in a media query.

```
@media speech {
    body > section.screenReaderOverview {
        display: block;
    }}@media not speech {
    body > section.screenReaderOverview {
        display: none;
    }}
```

In this example, the not keyword is used to apply a CSS style if the browser is NOT currently a screen reader.

**Only Keyword**

The only keyword is a special case keyword used for legacy browser support. Older browsers expect a comma-delimited list of media types when using the media attribute of the link element. Using this keyword forces the older browsers to ignore the rest of the media query.

```
<head>
  <link rel="stylesheet" href="base.css" />
  <link rel="stylesheet" href="print.css" media="only print" /></head>
```

In this example, an older browser will return the media type "only" and ignore the rest of the media query.

## Learn More

This is only a subset of the features available in media queries, many more queries are listed here:

(http://www.w3schools.com/cssref/css3_pr_mediaquery.asp)

## CSS FONTS

We can change the font of content on our HTML pages using the CSS styles. The syntax is flexible enough to support listing multiple fonts and including external font files.

We can specify different fonts to be used for our HTML page using the "font-family" attribute:

```
article {
    font-family: Segoe;}
header {
    font-family: Times;}
```

If we are unsure which fonts are installed on the user browser, we can specify multiple fonts. The browser will attempt to load the fonts from left to right until it finds a font that exists:

```
article {
    font-family: "Palatino Linotype", "Book Antiqua", Palatino ;}
header {
    font-family: "Lucida Sans Unicode", "Lucida Grande", Arial ;}
```

Each browser also has a default font for specific font types including:

- sans-serif
- serif
- monospace

You can use the keyword for these basic fonts as a catch-all fallback in case you can't find any of the fonts you prefer:

```
article {
    font-family: "Segoe UI", Segoe, sans-serif;}
header {
    font-family: "Times New Roman", Times, serif;}
```

## LINKING TO EXTERNAL FONT FILES IN CSS

In CSS, you can define your own font families. These font families can import an external font file. These definitions use the **@font-face** keyword:

```
@font-face {
    font-family: "Demo Font";
    src: url(/demofont.otf);}
```

You can use a wide variety of file formats to create fonts including:

| Font Type | Browser Support |
|---|---|
| OpenType Font (OTF) | Most browsers |
| TrueType Font (TTF) | Most browsers |
| Web Open Font Format (WOFF) | Most browsers |
| Web Open Font Format 2.0 (WOFF 2.0) | Not supported in IE |
| Embedded OpenType Fonts (EOT) | Supported only in IE |
| SVG Fonts (SVG) | Supported only in Chrome and Opera |

Once defined, you can then use the new font family later in your other CSS styles:

```
footer {
  font-family: "Demo Font", Arial, sans-serif;}
```

The **@font-face** style declaration can also include other properties if you are using a font that supports different features. Many fonts are shipped with different features (bold, semi-bold, italic) in separate font files:

```
@font-face {
    font-family: "Simple Font";
    src: url(simplefont.ttf);}@font-face {
    font-family: "Simple Font";
    font-weight: bold;
    src: url(simplefont_bold.ttf);}@font-face {
```

```
    font-family: "Simple Font";
    font-style: italic;
    src: url(simplefont_italic.ttf);}
```

You can then use the font like normal throughout your CSS file[s]:

```
body {
    font-family: "Simple Font";
}
footer {
    font-style: italic;
}
header {
    font-weight: bold;
}
```

---

## INTRODUCING JAVASCRIPT

JavaScript is a programming language or an object-based programming written into an HTML page to make it more interactive.

JavaScript can be used to:

- Add multimedia elements such as showing, hiding, changing, creating image role overs, scrolling text across the status bar, etc.

- JavaScript makes it possible to create tailored dynamic page contents (contenuti su misura per pagine dinamiche), date and time, or other external data.

- Form processing such as user input validation during the form submission, modify the contents of the form.

JavaScript can also be used with CSS to make Dynamic HyperText Markup Language (DHTML) which enables the appearance and disappearance of webpages. JavaScript executes only on the web page that is in the window of the browser at any set time. This means that the scripts running on a page are immediately stopped the moment the user stops viewing the page. The only exemption to this is various client-side storage APIs and cookies which can be utilized by multiple pages to accumulate and pass information between them. This can happen even after the closure of the pages.

JavaScript statements are written with a script tag <SCRIPT> which informs the browser that the JavaScript code follows. The script is embedded in HTML to be interpreted and executed by the client's browser.

JavaScript relies on the browser to run, although it can also be utilized in other contexts other than a Web browser. JavaScript is a client side, interpreted, object-oriented, high-level scripting language.

creare un alert in JS

```
window.alert('Hello');
```

scrivere dentro al documento html

```
document.write('<h1>Header</h1><p>Test Text to the body</p><footer><small>Footer</small></footer>');
```

## OPERATORS

### Assignment

In JavaScript, you can assign values to a variable and even perform arithmetic on those values before assigning them to the variable.

```
var x = 8;          // assign the value 8 to xvar y = 2;          // assign the value 2 to yvar z = x + y;      // assign the value 10 to z (x + y)
```

There are other assignment operators including:

| Operator | Usage | Equivalent To |
|---|---|---|
| = | x = y | x = y |
| += | x += y | x = x + y |

| | | |
|---|---|---|
| -= | x -= y | x = x - y |
| *= | x *= y | x = x * y |
| /= | x /= y | x = x / y |
| %= | x %= y | x = x % y |

## Equality

> Comparison operators are used in logical statements to determine equality or difference between variables or values.

**Equal**

The == operator determines if two variables or values are equivalent in value.

```
7 == 9
b == 67 == d
```

**Not Equal**

The != operator determines if two variables or values are not equivalent in value.

```
m != 39 != t
```

**Equal Value and Equal Type**

The === operator determines if two variables or values are equivalent in both value and type.

```
l === 8"7" === x
```

**Equal vs. Equal Value and Equal Type:**

The equality operator == may perform type conversion to determine if the the values on the left and right are equivalent. The alternative operator === ensures that both values have equivalent values and types and does not perform intrinsic type conversion.

```
var fourNumber = 4;var fourString = "4";
fourNumber == fourString; // is true because of type converstion
fourNumber === fourString; // is false because of different types
```

## Arithmetic

You can use these arithmetic operators to perform the most common arithmetic (mathematics) operations on variables and literals.

| Operator | Action |
|---|---|
| + | Addition |
| - | Subtraction |
| * | Multiplication |
| / | Division |
| % | Modulus |
| ++ | Increment |
| -- | Decrement |

```
var z = 5;var v = 2;var i = z + v; // 7var o = z - v; // 3var p = x * y; // 10
```

## String

The plus sign (+) operators can be used to concatenate (combine) multiple strings together.

```
var first = "Example";var last = "Person";var name = first + " " + last; // name is now "Example Person"
```

The increment (+=) operator can be used to affix a new string to the end of the current string.

```
var greeting = "Happy ";
greeting += "Holidays"; // greeting is now "Happy Holidays"
```

## Logical

There are many other operators available for common logical comparisons.

| Operator | Comparison |
|---|---|

| | |
|---|---|
| == | equal to |
| === | equal value and equal type |
| != | not equal |
| !== | not equal value or not equal type |
| > | greater than |
| < | less than |
| >= | greater than or equal to |
| <= | less than or equal to |
| ? | ternary operator |

```
var z = 12;var y = 23;var g = false;var t = 15;var f = 12;var ZGreaterThanY = z > y; // is falsevar ZLessThanOrEqualToY =
z <= y; // is truevar IfGIsTrueReturnBOtherwiseReturnA = g ? t : f; // is 12
```

## Type

JavaScript also includes operators that allows you to infer information based on the type of a variable or value.

| Operator | Behavior |
|---|---|
| typeof | Returns the type of a variable |
| instanceof | Returns true if an object is an instance of an object type |

CONTROL STATEMENTS

Most programming code performs various actions depending upon the result of decisions along the way. The statements that pick a path of execution based on a decision are called control statements. The statements conditionally run code depending on the result of some evaluation.

## If/Else

The `if` statement runs the code within it's block only if the statement within the parentheses evaluates to true.

In this example, if the value of the **day** variable is greater than the literal **5**, than the code within the statement will execute. The greeting will then be "Have a good weekend".

```
if (day > 5) {
    greeting = "Have a good weekend!";}
```

You can use an else statement to execute if the evaluated expression is false.

In this example, if the value of the **day** variable is less than or equal to **5**, than the greeting will be "Welcome to work".

```
if (day > 5) {
    greeting = "Have a good weekend!";} else {
    greeting = "Welcome to work";}
```

The `else if` statement allows you to chain multiple if and else statements together so that you can evaluate multiple conditions. The else statement at the end will ony excute if ever if or else if statement is false.

```
if (day > 5) {
    greeting = "Have a good weekend!";} else if (day < 2) {
    greeting = "Are you having a case of the Mondays!";} else {
    greeting = "Welcome to work";}
```

## For

> Loops can execute a block of code a number of times. Loops are handy, if you want to run the same code over and over again, each time with a different value.

The `for` loop allows you to execute a block of code a set number of times. It is composed of three parts: - The creation of the indexer variable. In this example, we create a variable named **i** and set it's initial value to 0. - The expression to evaluate before each loop. In this example we want to make sure that the value of **i** is less than the length of the countries array (3). - A statement to execute at the end of each loop, after the code within the block has been executed. In this example, we increment the **i** variable by 1. Since array indexers are zero-based, this loop will evaluate true for 0, 1, 2 and 3 but then fail when i is 4. This will allow us to run our code block for every item in the array.

```
var countries = ["USA", "JPN", "RUS", "ENG"];for (var i = 0; i < countries.length; i++) {
    text += countries + "<br />";}
```

### Foreach

The `forEach` function conveniently invokes a function (named or anonymous) for each item in the array. This can be used to easily iterate over the items in an array.

> **Warning:** If you have a multi-dimensional array in JavaScript, this statement will execute for every single element in the array and can have unintended consequences.

```
var sum = 0;var numbers = [4, 9, 16, 25];function increment(item, index) {
    sum += item;}
numbers.forEach(increment);
alert(sum);
```

### Switch

The switch statement gets the value within the parentheses (the expression to evaluate) and then compares it to various case values. It goes in order from top to bottom and will execute the code in a case only if the value of the case matches the value from the switch. If the code block for the case includes a break statement, then execution ends. Otherwise the switch will continue on with other cases.

```
var day;switch (new Date().getDay()) {
    case 0:
        day = "Sunday";
        break;
    case 1:
        day = "Monday";
        break;
    case 2:
        day = "Tuesday";
        break;
    case 3:
        day = "Wednesday";
        break;
    case 4:
        day = "Thursday";
        break;
    case 5:
        day = "Friday";
        break;
    case 6:
        day = "Saturday";
        break;}
alert(day);
```

## JAVASCRIPT DATA TYPES (PRIMITIVE)

Each variable in JavaScript contains a data type which dictates the values that can be stored in it. Data of a primitive data type is not an object and lacks methods. Primitives represent the most fundamental types of data supported by a language. Primitive data types include Number, Undefined, Boolean, Null, and String. The section describes the details of primitive data types.

### Boolean Data Type

The Boolean data type is made up of two literals, true and false; that is parallel to the truth values of Boolean logic. The Boolean variables are usually used to represent the outcomes of a comparison i.e. less than and greater than. The variables are also useful for representing the presence, or absence thereof, of value.

### Number Data Type

This data type represents all numeric data. Number data type comprises of the positive and negative. Negative integers have a (-) minus sign before them while positive are preceded by a (+) sign (but not necessary). There is various methods that numbers can be formatted. They include integers, real numbers, scientific notation, hexadecimal, and octals.

### String Data Type

This a methodical sequence of zero or more characters. String data type represents textual data. String literals are created by closing a character sequence in double or single quotes. The only restriction is that the opening and closing quote must be of the same type.

## JAVASCRIPT OBJECTS

An object is a list of unorganized primitive data types (and at times reference data types) stored as a sequence of name-value pairs. All the items in present in the list is called a properties of the object. When a JavaScript Object property stores a function, the property becomes known as a method.

The name of the object followed by a period followed by the property name is used when referring to a property of an object. Objects can also serve as associative arrays. Associative arrays, unlike regular arrays are indexed via strings contrary to numbers.

A programming language qualifies to be called object-oriented if it has the following four basic capabilities:

• Encapsulation − store related information, data or methods, together in an object.

• Aggregation − store one object in another object.

• Inheritance − the ability of a class to depend upon another class (or number of classes) for some of its properties and methods.

• Polymorphism − to write one function or ways that serves in a variety of different ways.

## Understanding JavaScript Objects

There are a few key points to keep in mind when working with JavaScript objects:

**Object Properties**- Object properties are any of the three primitive, or any of the abstract data types, like another object. They are normal variables used in the internal object's methods, and may also be globally visible variables used in the page.

**Object Methods**- These are the functions that allow the object to do an action or allow something be done to it. Responsible for content display, executing mathematical operations.

**User-Defined Objects** - All user-defined objects and built-in objects are descendants of an object called Object.

**The 'with' Keyword**- This keyword is used to reference shorthand for an object's properties or methods.

**JavaScript Native Objects**- JavaScript has several built-in or native objects. These objects are accessible anywhere in your program and will work the same way in any browser running in any operating system.

## JavaScript Built-In Objects

Here are the important JavaScript Built-In (Native) Objects −

| **Built-In Objects** |
|---|
| JavaScript Number Object |
| JavaScript Boolean Object |
| JavaScript String Object |
| JavaScript Array Object |
| JavaScript Date Object |
| JavaScript Math Object |
| JavaScript RegExp Object |

COLLECTIONS

JavaScript arrays can be used to collect multiple values and store them within a single variable. Arrays are created using the "square bracket" character.

```
var cars = ["Saab", "Volvo", "BMW"];
```

The code above creates an array named cars.

If you would like to reference an item in an array, you can reference it using a number referring to it's position in the array called it's **index**. Array indexes are zero-based meaning that the first item in the array has an index of 0, the second item has an index of 1 and so on...

```
cars[0] // Saab
cars[1] // Volvo
cars[2] // BMW
```

The code above selects the various element in the array.

You can also create an array using the `Array` keyword in JavaScript:

```
var cars = new Array("Saab", "Volvo", "BMW");
```

JavaScript Arrays are special objects that can contain values of various types. the values within an array do not need to be the same type and it's perfectly valid to place values of different types in the same array. You can even have arrays within an array:

```
function myFunc() {};
myArray[0] = Date.now;
myArray[1] = myFunc;
myArray[2] = new Array("Demo", "Test");
```

#### UNDEFINED AND NULL

JavaScript includes two possible states for a variable if it does not have a value assigned: **undefined** and **null**.

### Undefined Data Type

This data type represents a value that does not exist or is not known. The undefined date type consists of a single value, undefined. Variables that haven't been assigned a value are defaulted to the value: "undefined". You can also intentionally assign the value "undefined" to a variable.

### Null Data Type

This is almost similar to undefined data type, in the sense that it also represents the inexistent value. What differentiates it from Undefined, is that it represents an intentional absence of value. The Null data type just as the name suggests is represented by a single literal, null. The null value is usually used to initialize or clear object variables. The following example shows how the null value is assigned to a variable.

### Creare Callbacks

In JS puoi aggiungere degli **event handlers** per gestire gli eventi di click su diversi elementi HTML. Una delle cose interessanti che possiamo fare con JS è che **possiamo passare come parametri delle funzioni** questo pattern è chiamato **callback functions, puoi inserire firma oppure definirla direttamente dentro il parametro**

Nell'esercizio definisci la firma delle funzioni JS e da console inserisci dei breakpoint direttamente alla graffa di chiusura dei metodi (vai in Sources e clicchi sulla riga dove vuoi mettere i breakpoint) cliccando sui bottoni ad esecuzione della funzione esecuzione va in pausa

document.writeln('Test Message');

riempie la viewport con "Test Message"

#### EVENT HANDLING

When something happens to an HTML element, you can write JavaScript logic to be executed at that point in time. Your JavaScript functions can essentially "react" to something that happens to a HTML element. These are referred to as **events**.

Here's some common HTML events: - Web page has finished loading - Someone put content in a HTML input form - A button was clicked on the web page

JavaScript allows you to assign a function to be executed when the event is detected. This can be done in JavaScript or through the use of HTML element attributes.

In this example, we have a button and using an attribute, we run JavaScript logic immediately when the button is clicked.

```
<p id="dateTarget"></p><button onclick='getElementById("dateTarget").innerHTML = Date()'>Get Current Time</button>
```

In this example, we do the same thing, but we instead reference a function in JavaScript:

```
function handleClick() {
    document.getElementById("dateTarget").innerHTML = Date();}
```

```
<p id="dateTarget"></p><button onclick='handleClick()'>Get Current Time</button>
```

In this example, we also move the event assignment to JavaScript:

```
document.getElementById('dateButton').onclick = handleClick;function handleClick() {
    document.getElementById('dateTarget').innerHTML = Date();}
```

```
<p id="dateTarget"></p><button id="dateButton">Get Current Time</button>
```

Alternatively, you can use the `addEventListener` function for event assignment by either referencing a function using it's named identifier or supplying an anonymous function:

**Named Identifier:**

```
document.getElementById('dateButton').addEventListener('click', handleClick);function handleClick() {
    document.getElementById('dateTarget').innerHTML = Date();}
```

**Anonymous Function:**

```
document.getElementById('dateButton').addEventListener('click', function handleClick() {
    document.getElementById('dateTarget').innerHTML = Date();});
```

## Common HTML Element Events

Here's the most common HTML element events that you may use in your web pages.

| Event | Details |
|---|---|
| onload | The browser has finished loading the page |
| onmouseover | The user moves the mouse over an HTML element |
| onmouseout | The user moves the mouse away from an HTML element |
| onclick | The user clicks an HTML element |
| onkeydown | The user pushes a keyboard key |

EVENTI BUBBLE

Gli eventi in Js si propagano su + elementi HTML e possono continuare oppure stopparsi lungo il cammino.

Un evento lo possiamo vedere su un singolo elemento oppure come si propaga su + elementi

Vediamo come un evento si propaga dall'elemento + interno al + esterno

creiamo 3 livelli di section ed all'intenro della + interna inserisco del testo

```
<section class="outermost">
            <section class="regular">
                <section class="innermost">
                    Click Me!
                </section>
            </section>
    </section>
```

poi definiamo un css ove do diverse sfumature di verde come sfondo alle tre sezioni

```
section.outermost {
    background-color: #0E4241;
}

section.regular {
    background-color: #256363;
}

section.innermost {
    background-color: #217773;
}
```

ora creo un pezzo di codice JS ove assegno all'evento onclick dell sezione + interna un **HANDLER**

e sposto il link dello script appena prima del body

```
var innerMostElement = document.getElementsByClassName('innermost')[0];


innerMostElement.onclick = handleClick;


function handleClick (event) {
   console.log(event.target.className);
}
```

ricorda sempre di definire il link allo script in fondo al body di modo che tutti gli elementi del DOM della pagina siano correttamente caricati.

applico la funzione a tutti gli elementi section

```
var sections = document.getElementsByTagName('section');

for (var index = 0; index < sections.length; index++) {
    sections[index].onclick = handleClick;
}

function handleClick (event) {
  console.log(event.target.className);
}
```

vediamo che cliccando sulle sezioni + interne il console log VIENE PROPAGATO SUGLI ELEMENTI ESTERNI

dunque vi sono 2 o 3 console log

ma SE VOLESSIMO STOPPARE LA PROPAGAZIONE DELL'EVENTO SU UN PARTICOLARE ELEMENTO????

```
function handleClick (event) {
    console.log('current element: ' + this.className + ' | target element: ' + event.target.className);

    if (this.className == 'regular') {
        event.stopPropagation();
        console.log('event propogation stopped');
    }
}
```

in questo modo quando la propagazione giunge all'elemento regular si stoppa

quindi possiamo gestire e stoppare quando vogliamo la propagazione di un evento

FUNZIONI ANONIME

Di solito, le funzioni hanno un nome che le identifica univocamente. Questo ci permette di usarle nelle nostre applicazioni JavaScript. Una funzione senza nome viene detta una funzione anonima. Dato che non ha nome, una funzione anonima di solito è di difficile accesso dopo che è stata creata inizialmente.

## Normal function

```
function greet() {
  alert('Hello Everyone');
}
greet();
```

## Funzione Anonmima

```
var anonymousFunctionInAVariable = function() {
  alert('I am anonymous');
};
anonymousFunctionInAVariable();
```

L'uso comune delle funzioni anonime è come argomenti per altre funzioni.

```
function normalFunction(input, callbackFunction) {
    var newValue = input + 1;
  callback(newValue);
};
normalFunction(1, function(inputViaCallback) {
  alert(inputViaCallback);
});
```

Nell'esempio sopra, la funzione anonima viene chiamata alla fine della funzione Regolare. Mostra un alert con valore "2".

**DOM**

avendo un doc HTML nella console puoi vederne il DOM

e da console puoi invocare una serie di istruzioni su di esso

document.head

ti restituisce l'head dell'elemento

```
document.head
<title>HTML Page</title>
```

document.title

ti restituisce il titolo dell'elemento

```
document.title
> "HTML Page"
```

nella console puoi fare anche ASSEGNAZIONI di modo da cambiare il contenuto della pagina

ad esempio questo cambia titolo della pagina

```
document.title = "Demo Page"
> "Demo Page"
```

puoi stampare l'url della tua pagina

document.URL

```
document.url
>file:///C://Users/cg08586/Documents/MEGAsync/Intro%20to%20HTML%20and%20CSS/Corsi/DEV211.1x%20Introduction%20to%20HTML,%20CSS%20and%20JavaScri
web-development/modulo04/nav-dom/index.html
```

per vedere SE CI SONO SCRIPT basta digitare

document.scripts

```
document.scripts
> []
```

per vedere SE CI SONO LINK ESTERNI basta digitare

document.links

```
document.links
> [<a href="http://www.microsoft.com">Microsoft</a>, <a href="http://www.edx.org">edX</a>]
```

posso stampare solo un link

document.links[0]

```
document.links[0]
> <a href="http://www.microsoft.com">Microsoft</a>
```

posso stampare L'INDIRIZZO di un link, un suo attributo

document.links[0]

```
document.links[0].href
> "http://www.microsoft.com"
```

oppure assegnargli un valore

```
document.links[0].href = "http://www.microsoft.com/learning"
> "http://www.microsoft.com/learning"
```

o creare un nuovo attributo

```
document.links[0].target = "_blank"
> "blank"
```

Recuperare un document via TAG

```
document.getElementByTagName("footer")
> [
<footer>
        Copyright information
</footer>
]
```

possiamo cambiarne l'HTML interno

```
document.getElementByTagName("footer")[0].innerHTML = "Demo Footer"
> "Demo Footer"
```

possiamo cambiarne l'HTML interno

c'è anche

getElementsByClassName

getElementById

## GESTIONE ECCEZIONI

Diversi errori possono accadere quando esegui codice scritto in JavaScript. Gli Errori possono arrivare da diversi posti: - Codice scritto dal tuo development team - Valori in input non corretti - Condizioni Network (Temporanee o Permanenti) -Problemi imprevisti

JavaScript ti permette di specificare blocchi di gestione degli errori che esegui il codice e lo salvaguarda in diversi scenari. L'istruzione `try` ti permette di eseguire il codice dentro una "sandbox" che teta il codice per controllare se vi sono errori. Se succede un errore, il codice dentro l'istuzione `catch` viene eseguito e ti concede l'opportunità di gestire tale errore in sicurezza.

```
try {
  callExternalFunction();
}
catch(error) {
    console.log(error.message);
}
```

L'istruzione `finally` è molto utile se vuoi assicurarti che un blocco di codice venga comunque eseguito, se possibile, dopo che la tua eccezione è stata gestita in sicurezza. Se hai aperto connessioni a servizi esterni, questo ad esempio è il posto esatto per assicurarti che la connessione sia chiusa. L'istruzione finally viene eseguito indipendentemente dal risultato.

```
var connection = getExternalConnection();
try {
  connection.open();
 connection.callExternalFunction();
}catch(error) {
  console.log(error.message);
}finally {
  connection.close();
}
```

Puoi persino creare i tuoi errori personalizzati usando l'istruzione `throw`.

```
try {
  callExternalFunction();
  if (result.isNotValid) {
    throw "issue with external service";
 }
}catch(error) {
   console.log(error.message);
}
```

### XMLHTTPREQUEST

I browser moderni hanno un oggetto pre-costruito in JavaScript per gestire l'atto di richiedere DATI da un Server. A questo oggetto ci si riferisce con l'identificativo **XMLHttpRequest**. Molti browser hanno in se delle facility per gestire l'accesso, il parsing e la manipolazione dell'XML.

```
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (xhttp.readyState == 4 && xhttp.status == 200) {
       // Action to be performed when the document is read;
    }};
xhttp.open("GET", "filename", true);
xhttp.send();
```

L'esempio sopra dimostra la creazione e l'uso dell'oggetto XMLHttpRequest. Anzitutto, l'oggetto viene creato e poi un EVENT HANDLER (che è una funzione anonima) vein assegnato all'evento **onreadystatechange**. Ogni volta che la pty **readyState** cambia, allora la funzione viene eseguita. Una volta fatta l'assegnazione, semplicemente invochiamo la funzione **open** e poi la funzione **send**.

Valori di Status per `readyState`

0. richiesta non inizializzata

1. stabilita connessione al server

2. richiesta ricevuta

3. processata richiesta

`4. richiesta finita e response pronta`

Quando **readyState** cambia e diventa `4` lo stato della response è `200`, in tal caso sai di avere ricevuto con successo la response dalla request in formato file XML.

Accesso Fra i Diversi Domini

Per proteggere gli utenti, i browser moderni sono sicuri e non permettono request da un dominio all'altro (across domains o cross-domain). Il file XML che provi a caricare deve quindi essere locato nello stesso server della tua pagina web.

## API HTML5

HTML5 cambia considerevolmente lo scenario del web, poiché la sua specifica delega supporto acne alle API JavaScript che possono essere usate per costruire pagine web che possano comportarsi come web applications.

### STORAGE (MEMORIZZAZIONE)

Prima di HTML5, era difficile memorizzare dati localmente dentro al tuo browser. Ti veniva richiesto di usare i cookies e questi dovevano necessariamente essere inclusi in ogni request al server causando potenzialmente un overhead nelle tue richieste.

HTML5 ha introdotto il **local storage**. Il Local storage permette alle web app di memorizzare dati direttamente nel browser utente. Dato che i dati non vengono mandati "in giro per la rete", è molto + sicuro ed eventualmente si possono memorizzare grossi quantità di dati senza doversi preoccupare di inviarli in ogni richiesta HTTP. Il locale storage inoltre può immagazzinare molti + dati (fino a 5MB) dato che non è necessario trasferirli.

Il Local storage è catalogato (differenziato) per dominio del sito web (www.test.com, www.example.com) e protocollo (http, ftp, etc.). Questo significa che ogni pagina web nel tuo sito che usa lo stesso protocollo e lo stesso dominio **può accedere allo stesso local storage di dati**.

### Local Storage in JavaScript

Ci sono due oggetti JavaScript usati per gestire il salvataggio dati localmente nel client browser:

### localStorage Object

L'oggetto localStorage memorizza dati che non scadono. Ovvero questi dati sono disponibili persino dopo che l'utente ha chiuso il browser o spento il computer, lo trovi anche nello inspect.

```
localStorage.setItem("lastname", "Smith");
localStorage.getItem("lastname");
```

## sessionStorage Object

L'oggetto sessionStorage memorizza i dati di una singola sessione. Se chiudi la finestra o il tab del browser i dati vengono persi.

```
sessionStorage.clickcount = Number(sessionStorage.clickcount) + 1;
var getClickCount = 'Click Count: ' + sessionStorage.clickcount;
```

GEOLOCALIZZAZIONE

HTML5 include un API di **Geolocalizzazione** che ti permette di Chiedere la posizione di un utente sia ad un dato istante che lungo un periodo di tempo (tracking). La localizzazione di un utente è considerata un'informazione privata quindi all'utente verrà aperto un prompt per esplicitare l'approvazione della tua richiesta al primo uso dell'API.

Ottenere la Posizione Corrente

Anzitutto, usa la funzione `getCurrentPosition()` per ottenere la posizione corrente dell'utente. I dati che ti verrano restituiti saranno la latitudine e la longitudine dell'utente. La funzione prende un parametro di callback (un'altra funzione) che invocherà una volta determinata la tua localizzazione:

```
function getPositionResult(position) {
    console.log(position.coords.latitude + ' ' position.coords.longitude);}
navigator.geolocation.getCurrentPosition(getPositionResult);
```

## Watch Position

La funzione `watchPosition()` invoca la funzione di callback mentre l'utente si sposta di modo che puoi tracciarne la posizione durante il tempo. Per smettere di guardare la posizione dell'utente invoca la funzione `clearWatch()`.

```
function refreshPosition(position) {
    var list = document.getElementById('positionList');
    list.innerHtml += ('<li>' + position.coords.latitude + ' ' position.coords.longitude + '</li>');
}
navigator.geolocation.watchPosition(refreshPosition);
```

**CANVAS**

Preparo il piano da lavoro

## index.html

```
<!DOCTYPE html><html lang="en-US">
    <head>
        <title>HTML Page</title>
        <link rel="stylesheet" href="css/style.css" />
    </head>
    <body>
        <canvas id="demoCanvas" width="800" height="400">
            Test Text
        </canvas>
        <script src="js/script.js"></script>
    </body></html>
```

## style.css

```
canvas {
    border-style: solid;
    border-width: thick;
    border-color: #FCA90F;
    background-color: #C1D6CD;
}
```

poi creo il js per muovere le figure

### script.js

```
var canvas = document.getElementById("demoCanvas");
var context = canvas.getContext("2d");

// creo colore, ampiezza riga
context.fillStyle = "#A9B0B3";
context.strokeStyle = "#746C73";
context.lineWidth = 5;

// creo la riga
context.moveTo(0, 0);
context.lineTo(700, 350);
context.stroke();

// creo un cerchio
context.beginPath();
context.arc(160, 80, 70, 0, 2 * Math.PI);
context.fill();
context.stroke();

// crea freccia
context.beginPath();
context.moveTo(650, 345);
context.lineTo(700, 350);
context.lineTo(665, 315);
context.stroke();

// cambio colore
context.fillStyle = "#20293F";
context.strokeStyle = "#20293F";

// scrivo
context.font = "78px Segoe UI";
context.fillText("Hello", 190, 230);
context.strokeText("World", 190, 310);
```

## WEB WORKERS

Normalmente, il codice JavaScript eseguito su una pagina HTML la rende immediatamente non responsive finché lo script non finisce la sua esecuzione. Questo significa che le funzioni JS + lunghe e con + logica rendono la tua applicazione + lenta e di difficile uso per l'utente finale. Per risolvere questa cosa, possiamo rendere fuori dal caricamento le attività a lunga esecuzione e lente mediante i **Web Workers**.

Un web worker è un file JavaScript con logica che viene eseguita in background, senza influenzare le performance della pagina o renderla non responsive.

Creare un Web Worker

Un web worker viene scritto in un file JavaScript separato che viene referenziato dalla tua web application. Il file appare così:

```
var array = [ 7, 8, 9, 0, 1, 2, 3 ];function processArray() {
    for (var i = 0; i < array.length; i++) {
        var result = handleData(arrayOfData[i]);
        postMessage(result);
    }
}function handleData(data) {
    var response = sendToRemoveServer(data);
    return response;}
```

Per creare una nuova istanza di un web worker, devi solo usare il codice seguente in un file JavaScript referenziato dalla tua web application:

```
var worker = new Worker("WorkerFile.js");
```

Controllare un Web Worker

Puoi continuare a usare la variabile che referenzia il tuo web worker sia per stopparlo oppure per ascoltare i messaggi provenienti dal web worker.

**Stoppare il Worker**

Puoi usare la variabile per stoppare il web worker:

```
worker.terminate();
```

**Ascoltare messaggi**

Il web worker può inviare messaggi vero la pagina web che ha invocato lo stesso worker. La tua pagina può ascoltare e leggere questi messaggi gestendo l'evento **onmessage**:

```
worker.onmessage = function (event) {
    console.log(event.data);
}
```

Web Sockets

I WebSockets sono una tecnologia avanzata che rende possibile aprire una sessione di comunicazione interattiva tra il browser dell'utente e un server. Con questa API si possono mandare messaggi al server e ricevere risposte event-driven senza doverle richiedere al server.

Creiamo un web socket, facendo riferimento al codice online del web socket

```
var socket = new WebSocket('ws://echo.websocket.org');
```

Poi dobbiamo gestire un serie di funzioni ed eventi che vediamo

Esempio evento del socket che si apre, assegniamo una funzione anonima (callback) a questa var

```
socket.onopen = function () {
    console.log('Our socket has been opened!');
}
```

Creiamo un altro event handler che riguarda la ricezione di messaggi

```
socket.onmessage = function (event) {
    window.alert(event.data);
}
```

Una volta fatto questo e controllato su web browser

passiamo al prossimo step ovvero mandare un messaggio e ritornarmelo indietro, con un bottone che scatena l'evento e la funzione testMessage

al cui interno viene definita l'istruzione

socket.send("Hello World")

```
function testMessage () {
    socket.send("Hello World!");
}
```

Per chiudere il socket basta inviare socket.close() che chiude la connessione

AGGIUNGERE LIBRERIE ESTERNE PER MIGLIORARE LE APP HTML

OVERVIEW

Molte delle + popolari pagine web sono sviluppate usando dei framework. Questi sono usati sia per rendere le pagine web consistenti sia per facilitare il lavoro agli sviluppatori web. Molti di questi framework contengono funzionalità che altrimenti dovrebbero esser sviluppate ex-novo per ogni singola

pagina web.

Anche se questo corso tratta le fondamenta di HTML, JavaScript e CSS; andremo brevemente ad esplorare alcuni dei + popolari framework in maniera veloce. Questa è solo can breve introduzione quindi ogni sezione sarà un visione generale + che un'immersione. Ognuno di questi framework ha un corso su edX che va nella profondità e dettagli del framework.

> Per ogni framework, vedremo la descrizione resa disponibile dagli stessi autori del framework e poi vedremo demo e esercizi iniziali per provare il framework in scenari del mondo reale.

Alla fine inseriremo i link in **Next-Steps** ai corsi edX che affrontano in profondità i framework.

## JQUERY

> Dagli autori di jQuery e il contributo di Wikipedia

jQuery è una veloce, piccola e ricca di funzionalità libreria JavaScript. Rende cose come l'attraverso del DOM HTML e sua manipolazione, la gestione di eventi, le animazioni e le chiamate Ajax molto + semplici con una API di semplice uso che funziona su una moltitudine di browser. Con una combinazione di versatilità e estendibilità, jQuery ha cambiato il modo in cui milioni di persone scrivono JavaScript[1].

La sintassi di jQuery è stata progettata per rendere molto semplice la navigazione di un documento, selezionare elementi del DOM, creare animazioni, gestire eventi, e sviluppare applicazioni Ajax. jQuery inoltre fornisce funzionalità per creare plug-in sulla libreria JavaScript. Questo permette agli sviluppatori di creare astrazioni di interazioni di basso livello ed animazioni, effetti avanzati e di alto livello, abilitare temi sui widget. L'approccio modulare della libreria jQuery permette la creazione di potenti e dinamiche pagine web e web applications[2].

DOM Querying e manipoliazione

jQuery è focalizzato su metodi (funzioni) che ti aiutano a manipolare il Document Object Model (DOM). Tu sei in grado di cambiare e manipolare il DOM con il normal e lineare codice JavaScript, ma jQuery rende ciò molto + semplice dandoti dei metodi  che ti permettono di trovare, selezionare e manipolare element usando una sintassi che è incredibilmente simile alla sintassi CSS.

Per esempio, puoi usare jQuery per trovare un elemento hyperlink nella tua pagina mediante il suo **id** attribute con valore *example* e settore il suo contenuto interno HTML col testo **Click Me**:

```
$('a#example').html('Click Me');
```

Puoi anche trovare tutti gli ele button della tua pagina che hanno la **classe** *blueButton* e impostare il loro **background-color** a *blue* usando il CSS:

```
$('button.blueButton').css('background-color', 'blue');
```

## EVENT HANDLING

jQuery può esser usato per rispondere ad eventi in JavaScript.

```
$('button.register').click(function () {
    alert('You are registered!');});
```

## AJAX

jQuery può usare l'oggetto $.ajax e i suoi metodi per fare queries a risorse esterne (web services, files, APIs, etc.) in un modo indipendente dal browser.

```
$.ajax({
  url: "/api/historical/stocks/averagePrice",
  data: {
    tradingAs: 'VNET',
    year: 2014
  },
  success: function( data ) {
    $('p#weather').html('$' + data + ' USD');
  }});
```

1. jQuery, Write Less Do More: https://jquery.com/↩
2. jQuery: https://en.wikipedia.org/wiki/JQuery↩

Esempi di uso JQuery

```
$('*').length; // il numero degli elementi presenti nel document
```

```
$('*:empty').length; // il numero di elementi vuoti nel document
$('p').length; // il numero di p
$('.blueBg').length;
$('.redBg').length;
$('article.redBg').length;
$('#test1').length;

$('article[name=topFirst]').length // cerco ele con attributo name == a

$('article[name!=top]').length

$('article[name^=top]').length // con attributo name che comincia con top

$('article[name^=bottom]').length
$('article[name$=Second]').length // con attributo name che termina con Second
$('footer > p').length // direct child

$('article[class]').length // ogni article che ha class come attributo

$('article[id]').length
```

## ANGULARJS

Dagli autori di AngularJS

HTML va benissimo per dichiarare documenti statici, ma è molto debole quando proviamo a dichiarare viste dinamiche nelle nostre web-app. AngularJS ti permette di estendere il vocabolario HTML per la tua applicazione. L'ambiente risultante è straordinariamente espressivo, leggibile e di rapido sviluppo.

AngularJS è un insieme di strumenti per costruire il framework più adatto allo sviluppo della tua applicazione. E' totalmente estendibile e funziona bene con altre librerie. Ogni funzione può essere modificata o sostituita per adattarsi al tuo workflow di sviluppo e ai tuoi bisogni di funzionalità1.

## TEMPLATES(MODELLI)

Angular ha un linguaggio di tempting nativo che ci permette di associare diversi controlli in input ed attributi di elementi html, loro contenuto e valori a pty del model.

```
<body ng-app>
    <div>
        <label>Name:</label>
        <input type="text" ng-model="name" placeholder="Enter a name here">
        <hr>
        <h1>Hello {{name}}!</h1>
    </div></body>
```

### DUE VERSI DI DATA BINDING

In Angular, un controller può esser usato per implementare logica nel model per la view. Questo data binding in due direzioni ti permette di creare viste molto dinamiche per la tua web application.

Quello che rende AngularJS unico è il suo data-binding bidirezionale. Tale funzione è gestita in JavaScript il che toglie al server la responsabilità di bestie molti template o l'aggiornamento dei dati. Come autore, scrivi semplici template in semplice HTML usando uno specifico linguaggio di template e alcuni attributi speciali di angular da inserire nell'HTML che hanno un prefisso `ng-` . Angular ha un servizio chiamato **$scope** che rileva i cambiamenti nel modello dei dati e aggiorna l'HTML dinamicamente e automaticamente per riflettere il modello aggiornato.

In JavaScript, puoi fare cambiamenti al model usando il servizio **$scope**:

```
$scope.title = "Hello World";
```

In HTML, puoi manipolare lo scope usando le form e gli element standard di HTML:

```
<input type="text" ng-model="title" />
```

Ecco come appare un esempio di app AngularJS:

```
<body ng-app="sampleApp">
    <div ng-controller="sampleController">
        <label>Increment:</label>
        <button ng-click="increment()">Add 1</button>
        <hr />
        <h1>Value: {{value}}</h1>
    </div></body>
```

```
angular.module('sampleApp', [])
    .controller('sampleController', function($scope) {
        $scope.value = 1;
        $scope.increment = function () {
          $scope.value++;
        }
    });
```

Con AngularJS, **non ti serve monitorare il DOM o i cambiamenti ai tuoi elementi** poiché sono gestiti automaticamente da Angular per te. AngularJS inoltre porta con sé altri services che ti permettono di fare richieste ad API esterne o gestire il routing in una Single-Page Application (SPA).

1. AngularJS, HTML enhanced for web apps!: https://angularjs.org/↩

EXAMPLE

## index.html

```
<!DOCTYPE html><html lang="en-US">
    <head>
        <title>HTML Page</title>
        <script src="https://code.jquery.com/jquery-2.2.0.min.js"></script>
        <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.0/angular.min.js"></script>
        <script src="js/app.js"></script>
    </head>
    <body>
        <div ng-controller="homeController">
          <label>Username:</label>
          <input type="text" ng-model="uname" placeholder="Enter username" />
          <hr />
          <h2>Hello {{uname}}</h2>
          <button type="button" ng-click="testMe()">Test Me</button>
        </div>
    </body></html>
```

## app.js

```
var app = angular.module('basicApp', []);
app.controller('homeController', function ($scope) {
   $scope.uname = "demouser";
    $scope.testMe = function () {
        $scope.uname += "123";
    };
});
```

**Esempio di invocazione di Web Services con AngularJS**

index.html

```
<!DOCTYPE html>
<!-- setto root della App -->
<html lang="en-US" ng-app="demoApp">
   <head>
      <title>HTML Page</title>
      <!-- link CDN Google a Angularjs -->
      <script type="text/javascript" src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.2/angular.min.js"></script>
      <script type="text/javascript" src="js/script.js"></script>
   </head>
   <!-- definisco inizio controller -->
   <body ng-controller="demoController">
      <h1>Result</h1>
      <hr>
      <p>
         <!-- aggiorno contenuto p -->
```

```
        {{resultJSON}}
        <br>
        <mark>{{resultProperty}}</mark>
    </p>
  </body>
</html>
```

script.js

```
var module = angular.module('demoApp', []);

// creo il controller
var controller = module.controller('demoController', initController);

// aggiungo anche il parametro $http
function initController($scope, $http) {
    /*
    uso il metodo get di angular
    per fare la chiamata http al webservice
    */
    $http.get('http://httpbin.org/ip')
        .then(function (response) {
            // assegno a resultJSON la response
            // Parserizzata con JSONSTringify
            $scope.resultJSON = JSON.stringify(response);

            // assegno a var resultPty il value di origin
            $scope.resultProperty = response.data.origin;
        });
}
```

**Esempio di gestione del routing in una single-page-app**

EXAMPLE

**index.html**

```
<!DOCTYPE html><html lang="en-US">
    <head>
        <title>HTML Page</title>
        <script src="https://code.jquery.com/jquery-2.2.0.min.js"></script>
        <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.0/angular.min.js"></script>
        <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.0/angular-route.min.js"></script>
        <script src="js/app.js"></script>
    </head>
    <body ng-app="navigationApp">
        <div ng-view></div>
        <script type="text/ng-template" id="list.html">
            <a href="#">Back Home</a>
            <h2>The Journey</h2>
            <p>Click on a destination to view more details:</p>
            <table>
                <thead>
                    <th>ID</th>
                    <th>ICON</th>
                    <th>LINK</th>
                </thead>
                <tbody>
                    <tr ng-repeat="item in items">
                        <th>{{item.id}}</th>
                        <td>
                            <img src="{{item.icon}}" alt="{{item.title}}" height="20" />
                        </td>
                        <td>
                            <a href="#detail/{{item.id}}">{{item.title}}</a>
```

```
                        </td>
                    </tr>
                </tbody>
            </table>
        </script>
        <script type="text/ng-template" id="detail.html">
            <a href="#list">Back to List</a>
            <h2>{{item.title}}</h2>
            <img src="{{item.icon}}" alt="{{item.title}}" />
            <p>{{item.description}}</p>
        </script>
        <script type="text/ng-template" id="home.html">
            <h2>Learn About Your New Career!</h2>
            <p>Click Below to get started</p>
            <a href="#list">List of Career Steps</a>
        </script>
    </body></html>
```

**app.js**

```
var app = angular.module('navigationApp', ['ngRoute']);
app.controller('listController', function ($scope, model) {
    $scope.items = model.getAll();});
app.controller('detailController', function ($scope, $routeParams, model) {
    $scope.itemId = $routeParams.itemId
    $scope.item = model.get($scope.itemId);});
app.config(function($routeProvider) {
    $routeProvider
        .when('/', {
            templateUrl : 'home.html'
        })
        .when('/list', {
            templateUrl : 'list.html',
            controller  : 'listController'
        })
        .when('/detail/:itemId', {
            templateUrl : 'detail.html',
            controller  : 'detailController'
        })
        .otherwise({redirectTo : "/"});});
app.factory ('model', function () {
    var data = [
        { id: 0, title: 'Degree', icon: 'img/degree.png', description: 'First, you might graduate from college.' },
        { id: 1, title: 'Networking', icon: 'img/networking.png', description: 'Then you could network with colleagues.'
},
        { id: 2, title: 'Certification', icon: 'img/certification.png', description: 'Earn a certification to stay ahead
of the pack' },
        { id: 3, title: 'Job Hunt', icon: 'img/jobhunt.png', description: 'Take your network and certifications on the
hunt for the coolest jobs.' }
    ];
    return {
        getAll:function () {
            return data;
        },
        get:function(id){
            return data[id];
        }
    };});
```

**Images**

- certification.png
- networking.png
- degree.png
- jobhunt.png

BOOTSTRAP

> From Wikipedia Contributors

Bootstrap is a free and open-source collection of tools for creating websites and web applications. It contains HTML- and CSS-based design templates for typography, forms, buttons, navigation and other interface components, as well as optional JavaScript extensions. It aims to ease the development of dynamic websites and web applications. Bootstrap is a front end web framework, that is, an interface for the user, unlike the server-side code which resides on the "back end" or server[1].

**Responsive Design**

Bootstrap was designed for a responsive web. A responsive web page has a layout that adjusts to changes in screen size and resolution by dynamically re-arranging elements to fit. A responsive web page can be used on a desktop machine, laptop computer, a tablet device or a mobile phone without the need to change the HTML or CSS for each individual screen.

Bootstrap accomplishes it's responsive design by implementing many complex CSS media queries.

**HTML components**

Bootstrap ships with a CSS stylesheet that themes the majority of HTML elements that you use. Bootstrap also ships with a set of CSS classes that can be applied to normal HTML elements to create advanced elements such as panels, pagination, breadcrumbs, navigation menus, progress bars and even image thumbnails. Aside from those elements, Bootstrap ships with even more CSS classes that can be applied to various HTML elements to format them, change their position, change their font size and do many more things:

```
<a href="#" class="btn btn-link">Link</a><div class="alert alert-dismissible alert-warning">
    <button type="button" class="close" data-dismiss="alert">&close;</button>
    <h4>Error!</h4></div><div class="progress progress-striped active">
    <div class="progress-bar" style="width: 45%"></div></div>
```

**JavaScript Features**

Bootstrap also comes with a JavaScript file that utilizes jQuery for advanced UI elements such as image carousels, tooltips and dialog boxes. You can use these elements throughout your page by invoking special JavaScript functions or using special Bootstrap HTML element attributes:

```
<button type="button" class="btn btn-default" data-toggle="tooltip" data-placement="left" title="Tooltip on left">Tooltip
on left</button><button type="button" class="btn btn-primary" data-toggle="button">
  Single toggle
</button>
```

**Customizable**

Bootstrap's CSS stylesheet can be customized by using the Saas sheets provided with Bootstrap's source code. You can quickly compile your own CSS with your values set for different properties such as background-color and margin/paddings.

---

Bootstrap (front-end framework):https://en.wikipedia.org/wiki/Bootstrap_(front-end_framework)↩

USING BOOTSTRAP CLASSES IN HTML

```
<!DOCTYPE HTML>
<HTML LANG="EN-US">
 <HEAD>
  <TITLE>HTML PAGE</TITLE>
  <!-- JQUERY -->
  <SCRIPT TYPE="TEXT/JAVASCRIPT" SRC="HTTPS://CODE.JQUERY.COM/JQUERY-1.11.3.MIN.JS"></SCRIPT>
  <!-- BOOTSTRAP JS -->
  <SCRIPT TYPE="TEXT/JAVASCRIPT" SRC="HTTPS://MAXCDN.BOOTSTRAPCDN.COM/BOOTSTRAP/3.3.6/JS/BOOTSTRAP.MIN.JS"></SCRIPT>
  <!-- BOOTSTRAP CSS -->
  <!-- TEMA BOOTSTRAP -->
  <LINK REL="STYLESHEET" HREF="HTTPS://MAXCDN.BOOTSTRAPCDN.COM/BOOTSWATCH/3.3.6/PAPER/BOOTSTRAP.MIN.CSS" />
  <META CHARSET="UTF-8">
  <!-- AGGIUNGO COMPATIBILITA CON EDGE -->
  <META HTTP-EQUIV="X-UA-COMPATIBLE" CONTENT="IE=EDGE">
  <META NAME="VIEWPORT" CONTENT="WIDTH=DEVICE-WIDTH, INITIAL-SCALE=1">
 </HEAD>
 <BODY>
    <H2>H2 HEADER</H2>
    <DIV CLASS="JUMBOTRON">
     <H1>HEADER</H1>
     <P>A PARAGRAPH CONTAINING A SENTENCE OF TEXT.</P>
     </DIV>
```

```
    <DIV CLASS="PANEL PANEL-PRIMARY">
    <DIV CLASS="PANEL-HEADING">
      <H3 CLASS="PANEL-TITLE">PANEL USING PRIMARY THEME</H3>
    </DIV>
    <DIV CLASS="PANEL-BODY">
      CONTENT OF THE PANEL
    </DIV>
    </DIV>

    <DIV CLASS="ALERT ALERT-DISMISSIBLE ALERT-SUCCESS">
    <BUTTON TYPE="BUTTON" CLASS="CLOSE" DATA-DISMISS="ALERT">&TIMES;</BUTTON>
    <H4>ALERT SHOWING SUCCESS</H4>
    <P>
      A PARAGRAPH CONTAINING A SENTENCE OF TEXT. YOU CAN ALSO LINK TO SITES SUCH AS <A HREF="HTTP://WWW.EDX.ORG"
TARGET="_BLANK" CLASS="ALERT-LINK">EDX.ORG</A>
    </P>
    </DIV>
 </BODY>
</HTML>
```

## Bootstrap JavaScript

**index.html**

```
<!DOCTYPE html><html lang="en-US"><head>
        <title>HTML Page</title>
        <link href="css/style.css" rel="stylesheet">
        <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" rel="stylesheet">
        <link href="https://maxcdn.bootstrapcdn.com/bootswatch/3.3.6/flatly/bootstrap.min.css" rel="stylesheet">
        <script src="https://code.jquery.com/jquery-2.2.0.min.js">
        </script>
        <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/js/bootstrap.min.js">
        </script>
        <script src="js/script.js">
        </script></head><body>
        <nav class="navbar navbar-default navbar-fixed-top">
                <div class="container">
                        <div class="navbar-header">
                                <a class="navbar-brand" href="#">Cool Web App</a>
                        </div>
                </div>
        </nav>
        <div class="container">
                <div class="jumbotron">
                        <h1>Learning about HTML?</h1>
                        <p class="lead">This course will help you get ready to build HTML web applications.</p>
                </div>
                <ul class="list-group">
                        <li class="list-group-item"><span class="badge">14</span> Cras justo odio</li>
                        <li class="list-group-item"><span class="badge">2</span> Dapibus ac facilisis in</li>
                        <li class="list-group-item"><span class="badge">1</span> Morbi leo risus</li>
                </ul>
                <div class="alert alert-warning">
                        <strong>Heads up!</strong> This <a class="alert-link" href="#">alert needs your attention</a>, but it's not super
important.
                </div>
                <div class="alert alert-dismissable alert-danger alert-info">
                        <button class="close" data-dismiss="alert" type="button">X</button>
            <strong>Heads up!</strong> This <a class="alert-link" href="#">alert needs your attention</a>, but it's not super important.
                </div>
        <button class="btn btn-default" data-placement="left" data-toggle="tooltip" title="Tooltip on left" type="button">Tooltip on
left</button> <button class="btn btn-lg btn-danger" data-content="And here's some amazing content. It's very engaging. Right?" data-
toggle="popover" title="Popover title" type="button">
            Click to toggle popover
        </button>
        <button aria-controls="collapseExample" aria-expanded="false" class="btn btn-primary" data-target="#collapseExample" data-
toggle="collapse" type="button">
            Button with data-target
        </button>
                <div class="collapse" id="collapseExample">
```

```
                        <div class="well">
                                Anim pariatur cliche reprehenderit, enim eiusmod high life accusamus terry richardson ad squid. 3 wolf moon
officia aute, non cupidatat skateboard dolor brunch. Food truck quinoa nesciunt laborum eiusmod. Brunch 3 wolf moon tempor, sunt aliqua put
a bird on it squid single-origin coffee nulla assumenda shoreditch et. Nihil anim keffiyeh helvetica, craft beer labore wes anderson cred
nesciunt sapiente ea proident. Ad vegan excepteur butcher vice lomo. Leggings occaecat craft beer farm-to-table, raw denim aesthetic synth
nesciunt you probably haven't heard of them accusamus labore sustainable VHS.
                        </div>
                </div>
        </div></body></html>
```

**script.js**

```
$(function () {
    $('[data-toggle="tooltip"]').tooltip();
    $('[data-toggle="popover"]').popover();});
```

**style.css**

```
body {
    padding-top: 80px;}
```

## TYPESCRIPT

> From TypeScript's Authors and Wikipedia Contributors

TypeScript is a free and open source programming language developed and maintained by Microsoft. It is a strict superset of JavaScript, and adds optional static typing and class-based object-oriented programming to the language. TypeScript may be used to develop JavaScript applications for client-side or server-side (Node.js) execution[1].

TypeScript offers classes, modules, and interfaces to help you build robust components. These features are available at development time for high-confidence application development, but are compiled into simple JavaScript. TypeScript types let you define interfaces between software components and to gain insight into the behavior of existing JavaScript libraries[2].

### Translates to JavaScript

TypeScript is based on the syntax of JavaScript. Any existing JavaScript code (including external libraries such as jQuery and Angular) is considered valid TypeScript. TypeScript as a language extends JavaScript by adding additional functionality. TypeScript compiles to regular JavaScript that can run in a browser or in an environment such as Node.js.

### Strong typing

TypeScript brings many of the high-producitivity features of object-oriented programming to JavaScript. Classes, inheritance, accessors and generics are just some of the features that are now available for JavaScript developers using TypeScript.

Because TypeScript uses OOP principles, development tools can be updated to add features that are normally difficult or impossible to have in regular JavaScript due to it's "loose" typing system including: - static type checking - code refactoring - statement completion - symbol-based navigation

In the example below, we created a simple class with a public method that can be invoked:

```
interface IGreeter {
    greet(): void}class Greeter implements IGreeter {
    protected _greeting: string;
    constructor(message: string) {
        this._greeting = message;
    }
    public greet() {
        let greetingString: string = this.getGreeting();
        alert(greetingString);
    }
    private getGreeting(): string {
        return "Hello, " + this._greeting;
    }}
let greeter: IGreeter = new Greeter('World');
greeter.greet();
```

Here is the compiled JavaScript result:

```
var Greeter = (function () {
    function Greeter(message) {
        this._greeting = message;
    }
    Greeter.prototype.greet = function () {
        var greetingString = this.getGreeting();
        alert(greetingString);
    };
    Greeter.prototype.getGreeting = function () {
        return "Hello, " + this._greeting;
    };
    return Greeter;}());var greeter = new Greeter('World');
greeter.greet();
```

---

TypeScript: https://en.wikipedia.org/wiki/TypeScript↩

TypeScript, JavaScript that scales: http://www.typescriptlang.org/↩

## ADDING LOGIC TO A HTML PAGE USING TYPESCRIPT

> You must have the Node Package Manager installed to complete this lab. If you don't have NPM installed, you can install it from http://nodejs.org. Version 4.4.1 LTS or above is sufficient for this exercise

**Objective**: In this guided exercise, you will use TypeScript to write logic for your web page.

**Pre-requisite**: You need to have Visual Studio Code installed on your local machine (Windows, OSX or Linux) in order to complete this exercise.

In the open command prompt, execute the following

```
npm install -g typescript
```

**index.html** file

```
<!DOCTYPE html>
<html lang="en-US">
    <head>
        <title>HTML Page</title>
        <script type="text/javascript" src="ts/script.js"></script>
    </head>
  <body>
        <button onclick="showAlert()">Show Alert</button>
    </body>
</html>
```