HTML5

Inizialmente le pagine Web erano semplici ipertesti; l’evoluzione degli utenti, della velocità di connessione hanno portato nuovi bisogni, inizialmente aggirati con altre tecnologie (es. Flash).

* Oggi HTML5 e CSS3 forniscono un supporto per sviluppare siti e applicazioni Web
* A luglio 2014 il WWW ha rilasciato il primo standard HTML5, il primo con approccio modulare

*La diatriba tra Flash e HTML è stata aperta da Steve Jobs nel 2010*

* *Per versioni mobile Flash 11 sarà l’ultima versione*
* *Google, Amazon, Apple, hanno abbandonato Flash per HTML5*
* *Attualmente non si può ancora fare tutto in HTML5 ma è troppo più leggero ed efficiente per considerare ancora Flash (vi è Javascript in soccorso di HTML5)*

**Vantaggi HTML5**

1. *Interattività*: senza l’ausilio di plugin esterni, valida per le animazioni grafiche e video streaming/music player
2. *Pulizia del codice*: per una facilità di lettura da parte di motori di ricerca e leggerezza delle pagine
3. *Semantica intuitiva* grazie ai nuovi TAG di formattazione
4. *Form efficaci* per la comunicazione con l’utente
5. *Compatibilità* con tutti i browser: le vecchie versioni di HTML non erano supportate in maniera uniforme da tutti i browser
6. *Cache non in linea* (forzata e controllata autonomamente, diversa da quella dei browser) per poter usufruire del sito web anche in modalità offline
7. *Abbandono dei cookie*, sostituiti dal SessionStorage e LocalStorage per una velocità di memorizzazione e apertura sito ancora più performante
8. *La geolocalizzazione* (ora parte dello standard) applicabile in ogni parte del sito , dalla compilazione automatica per la provenienza del visitatore nei form al tracciamento per i tablet
9. *SEO (Search Engine Optimization) con i Microdata*, per inserire nei motori di ricerca molte più informazioni oltre che al classico title e description, che portano grande visibilità in **serp** (search engine result page). Il SEO riguarda il contenuto visibile così come quello non visibile della pagina, i link entranti, …
10. *Fogli di stile moderni* che, grazie al CSS3, non avranno bisogno di un supporto esterno per arrotondare gli angoli dei pulsanti, creare ombreggiature, ecc.

## Pagina HTML

La pagina è contenuta tutta all’interno del tag <html> (non ha un ruolo, è la radice della pagina) e contiene una serie di tag che la dividono in aree logiche

* *Agli inizi:* la divisione delle pagine avveniva grazie a tabelle (più lento, strutture più elaborate sono complesse da gestire)
* *Successivamente*: tabelle rimpiazzate dai <div>, contenitori generici
* ***HTML5****:* si esprime semanticamente il ruolo di un tag, per la leggibilità e per aiutare a esprimere quali elementi sono più o meno importanti per i motori di ricerca (in fase di ritrovamento ciò che sta in article viene boostato)

<!DOCTYPE html> → inserire descrizione

La pagina HTML è composta da due sezioni:

* <head>: contiene i tag META, informazioni sulla pagina visibili ai motori di ricerca e browser ma non all’utente
  + Può contenere il titolo della pagina che comparirà sul tab del browser
  + Tag keyword e description per SEO
  + viene inserito codice javascript per la dinamicità della pagina
  + regole css per la formattazione
* <body>: definisce il corpo del documento e contiene tutti gli elementi html come testo, hyperlink, immagini, tabelle, ecc.

**<article>**: contenuto indipendente e self-contained, ha senso quando può essere distribuito indipendentemente dal resto del sito

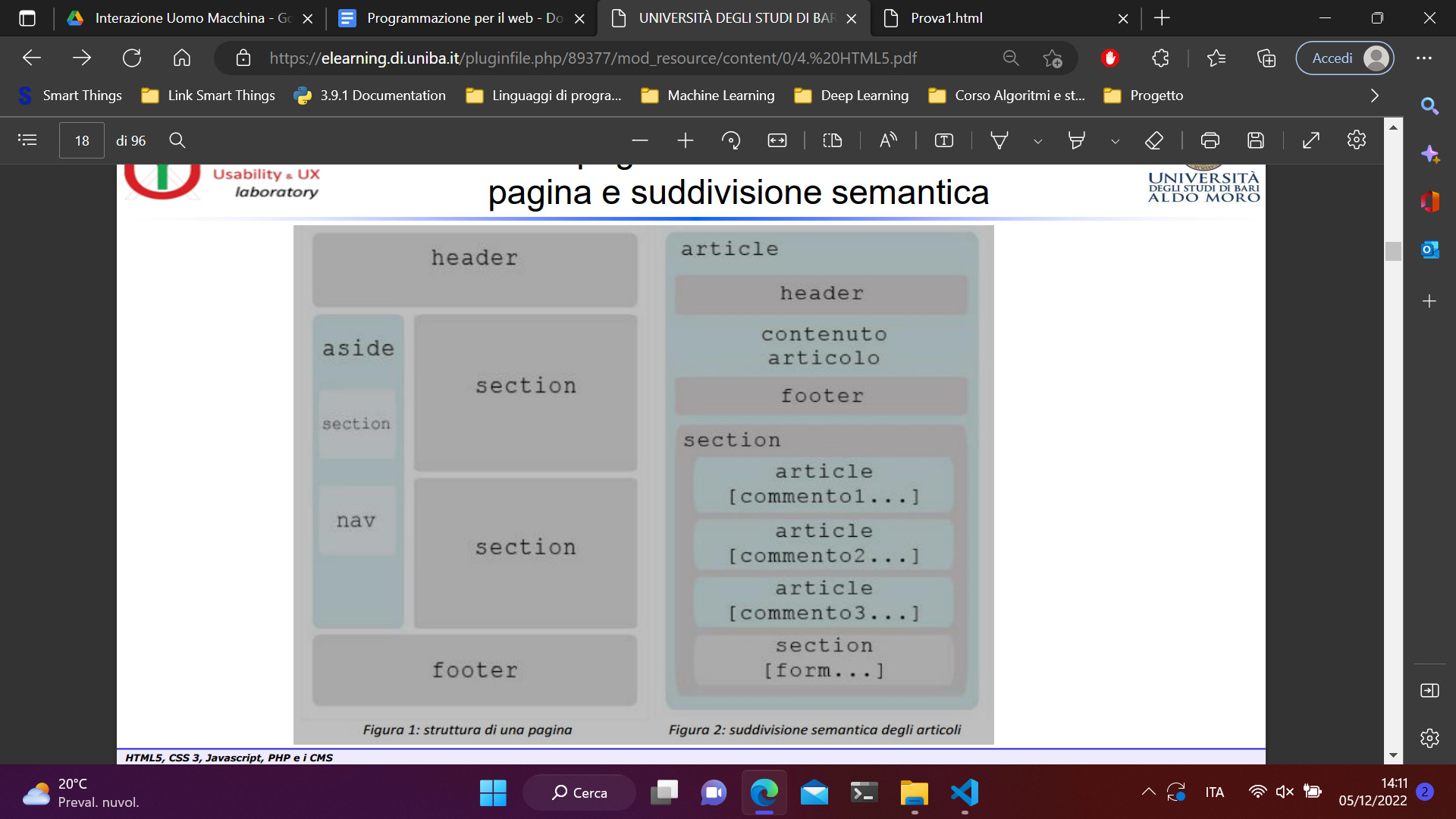
* esempi: forum post, news, comment

**<header>**: intestazione di un documento o di una sezione

**<section>**: sezione generica, raggruppamento tematico di contenuti

**<aside>**: usato per aggiungere informazioni aggiuntive a un elemento

**<footer>**: piè di pagina



**Inserire immagini e caption nella pagina**

* **<figure>** → un contenitore per elenchi di codice, immagini, diagrammi
* **<img src=“...” alt=“...”>** serve a inserire immagini, indicando *source\_path* e *alternative text*, nel caso in cui non si riesca a caricare l’immagine
  + attributi: title (titolo in modalità mouse over), width e height
* **<figcaption>** per la didascalia dell’immagine

*Nota: non è obbligatorio inglobare con il tag figure, è meglio*

**<hgroup> →** raggruppa gli elementi h1, …, h6

* Maschera l’outline dell’elemento padre che lo contiene; infatti, l’algoritmo dell’outliner riconosce come un titolo solamente l’heading con il valore più alto e considera tutti gli altri elementi sottotitoli

**<time>** → inserire date in formati comprensibili agli utenti e alle macchine (contenuto non strutturato)

* l’attributo datetime indica data e ora nel formato ISO
* l’attributo pubdate è di tipo booleano e serve a indicare se la data riportata fa riferimento alla pubblicazione dell’articolo in cui è contenuto il tag <time>

**<progress> →** inserisce una barra di completamento

* parametri max e value (valore corrente)

**<meter>** → rappresenta una grandezza scalare all’interno di un intervallo noto (progress bar statica)

* attributi value, min, max
* il testo all’interno del tag viene visualizzato in caso di mancata visualizzazione della barra

**<mark>** → evidenzia una porzione di testo all’interno di un documento

**<ul>** (unordered list)

**<ol>** (ordered list) → attributo start

**<li>** → list item

**<button>** → crea un pulsante cliccabile, differentemente dagli input si possono definire all’interno un testo o un’immagine

* attributi type=”button”, onclick= “*alert(‘Hello World’)”*

*Nota: l’attributo id esiste per tutti i tag (identifica in maniera univoca un tag all’interno della pagina, in ogni caso l’univocità è a carico del programmatore)*

### Web Form

**<form> →** per un form, attributi name, method=”post” action=”/search”

Il valore di action è la pagina php che sta sul server, nel form dobbiamo dire qual è la pagina php a cui passare dati (nell’esempio “/search”).

Nell’HTML5 l’attributo type dell’elemento <input> supporta 23 valori

* Input type *search* → diversa visualizzazione in base a browser e SO
  + autocomplete=”on” placeholder=”...” name=”...” required maxlength=”50”
* input type *submit* → bottone con attributo value= testo\_da\_stampare\_sul\_bottone

input type *reset* → bottone per effettuare il reset degli input nei campi del form

* input type *email* → controllo prima dell’invio della correttezza dell’email (no js), se l’email non è valida viene visualizzato un errore
  + attributo booleano *multiple* per inserire più indirizzi nella stessa casella
* input type *url*, anche in questo caso controllo automatico della correttezza prima dell’invio + attributo multiple
* input type *tel* → no controlli a runtime, comportamento responsive su iPhone (apre tastiera dedicata)
* input type *number* → frecce per incrementare/decrementare valore, attributi (min, max, step)
* per le date: *datetime* (con fuso orario), datetime-local (senza), date, month, week, time → attributi specifici per ciascuno, es min max step
* input type *color* → apre color chooser
* input type *datalist* → prima di HTML5 per permettere la scelta di un’opzione da un elenco preimpostato veniva utilizzata una combinazione dei tag <select> e <option>
  + si usa input type text, con list=”id\_lista” e un tag <datalist id=”id\_lista”>, contenente multipli tag <option value=”...”>

*Nota: <input> non viene chiuso e contiene solo attributi*

Altri attributi per <input>

* *autocomplete*, sfruttato dal browser per capire se autocompilare un campo in base alle precedenti compilazioni
* *autofocus*, serve a impostare il focus su un elemento di un form
* *pattern*, per inserire espressioni regolari
* *placeholder*
* *required* rende obbligatoria la compilazione del campo
* *novalidate*: salta le validazioni previste dal tag.

### Video e audio

HTML5 prevede la possibilità di riprodurre video e audio senza software di terze parti (es. Flash).

**<video>** → supporta MP4, WebM, Ogg

Attributi

* *controls* → booleano, attiva la toolbar per la riproduzione video e altre opzioni
* *poster* → immagine copertina video nel player
* *preload* →
  + “none” user-agent non effettua caricamento preventivo del video, buffer inizia a essere riempito alla prima pressione del tasto play
  + “metadata” prima dell’esecuzione vengono recuperate solo informazioni essenziali come durata, dimensione, …
  + “auto”: si presuppone che l’utente voglia prendere visione del video, quindi il browser lo caricherà preventivamente
* autoplay, loop, muted, width, height

**<source> →** per visualizzare video su tutti i principali browser servirà associare all’elemento più formati dello stesso video. Si può fare grazie all’elemento source ripetuto più volte nei tag video

* type per comportamenti del browser (il browser memorizza le ricerche e le ripropone)

**<audio> →** formato simile a video, formati supportati sono MP3, Wav, Ogg. Anche in questo caso più <source> per riprodurre su più browser.

### Canvas

*Creare ambienti interattivi 2D e 3D con html5*

Tecnologia canvas è una “tela” sulla quale si può disegnare con il “pennello” Canvas 2D API, ovvero funzioni Javascript che permettono un rendering dinamico di immagini bitmap.

Esiste anche una versione sperimentale di Canvas 3D.

<canvas> → attributi width e height

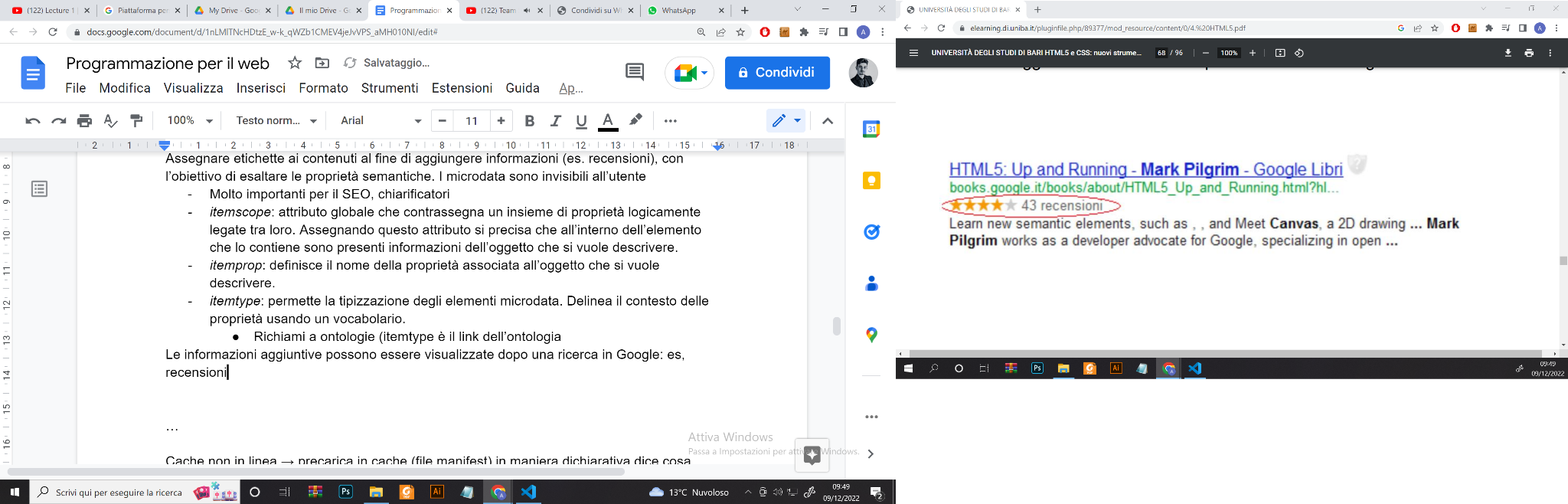
* Prima di iniziare a disegnare è necessario ottenere un riferimento all’oggetto che rappresenta l’elemento con il metodo JavaScript “getElementById()” e verificare che il browser lo supporti. Con l’utilizzo della libreria Modernizr si può testare il supporto
  + var canvas = document.getElementById(“tela”)
  + if(Modernizr.canvas) {...}
* si prende il contesto (es “2d”) con il metodo getContext, si impostano colori e offset delle shadow
* metodi fillStyle, rect (crea rettangolo), stroke (riempie i bordi), fill (riempie l’area interna)

### Microdata

Assegnare etichette ai contenuti al fine di aggiungere informazioni (es. recensioni), con l’obiettivo di esaltare le proprietà semantiche. I microdata sono invisibili all’utente

* Molto importanti per il SEO, chiarificatori
* *itemscope*: attributo globale che contrassegna un insieme di proprietà logicamente legate tra loro. Assegnando questo attributo si precisa che all’interno dell’elemento che lo contiene sono presenti informazioni dell’oggetto che si vuole descrivere.
* *itemprop*: definisce il nome della proprietà associata all’oggetto che si vuole descrivere.
* *itemtype*: permette la tipizzazione degli elementi microdata. Delinea il contesto delle proprietà usando un vocabolario.
* Richiami a ontologie (itemtype è il link dell’ontologia)

Le informazioni aggiuntive possono essere visualizzate dopo una ricerca in Google: es, recensioni → si parla di **rich snippets**



I vocabolari più popolari, supportati da Google, sono:

* *Breadcrumbs*: serve a rappresentare un insieme di collegamenti che possano aiutare un utente a comprendere e navigare all’interno del sito
* *Business and organizations*: definisce una società, un negozio e più in generale un luogo
* *People*: definisce una persona
* *Address*: definisce un indirizzo
* *Events*: definisce un evento (info titolo, data, luogo)
* *Product*: definisce un prodotto
* *Offer*: definisce un’offerta
* *Offer-aggregate*: aggregato di offerte
* *Recipes*: ricetta
* *Review/Review aggregate*: recensione/recensione aggregata
* *Rating*: valutazione

### Application Cache

Cache non in linea → API offline introdotte per la memorizzazione in locale di documenti per la visualizzazione di pagine web anche in assenza di connessione internet.

* Le informazioni riguardanti i file necessari alla navigazione offline sono definite all’interno di un file dedicato con estensione **manifest**. Tale file specifica in maniera dichiarativa cosa mettere in cache (riteniamo utile far salvare per quando non c’è connessione).
* Funzione utile per navigare e usare dispositivi mobili in aree con scarsa copertura di rete.

1. Si aggiunge l’attributo *manifest* all’elemento html, il cui valore deve corrispondere al percorso del file contenente l’elenco dei documenti per i quali si richiede la memorizzazione. Il file in questione dovrà essere *MIME type “text/cache-manifest”*

* *Nota: MIME type significa (Multipurpose Internet Mail Extensions) e un MIME type è la stringa che specifica il formato di un file. E’ composta da due parti: un tipo, che identifica la categoria generale del file, e un sottotipo, che specifica il formato del file sulla base della categoria.*

1. Lo user agent (il browser) viene a conoscenza dell’esistenza del file con le specifiche per una consultazione offline

In fallback vado a dire qual è la risorsa che richiede la connessione internet e qual è la sua versione statica.

Esempio:

CACHE MANIFEST

CACHE: (*descrive le risorse che si devono rendere disponibili in assenza di connessione con il server)*

/pagina.html

/css/cssPagina.css

NETWORK: (*contiene un’unica risorsa per riga. Gli indirizzi indicati si riferiscono a risorse per le quali il reperimento richiede una connessione internet attiva)*

/inviaDocumento.aspx

FALLBACK: (*per ogni riga sono indicate due risorse distinte. La prima indica quella disponibile nel caso di possibile accesso alla rete. La seconda rappresenta invece l’alternativa di cui disporre in caso di offline)*

news/\* /offline.html

/contatti.aspx /contatti.html

Possibili situazioni:

* *Prima visita*: quando il browser visita una pagina web nella quale sarà rilevato un riferimento a un file manifest per la prima volta, significa che l’utente non conserva una copia in locale di tale file.
  + Il browser scarica il file, lo interpreta e richiede al server tutte le risorse indicate al suo interno che saranno necessarie in caso di mancato accesso alla rete.
  + Fasi (correlate a eventi corrispondenti): checking (analisi del file manifest) → downloading (scaricamento di tutti i file) → progress (per ogni risorsa da scaricare viene scatenato tale evento) → cached (operazioni concluse)
* *Visita successiva senza modifiche del manifest*: il browser rileva che è già stato scaricato un file manifest associato a questo sito
  + Controllo tra copia locale del manifest e quello del file server per rilevare modifiche → nessuna modifica rilevata
  + Eventi: checking → noupdate (evento che comunica l’assenza di differenze tra i due file)
* *Visita successiva con modifiche del manifest*: sarà rilevata la presenza del file manifest e durante l’analisi saranno rilevate modifiche, quindi il file sarà sostituito con quello presente sul server. In seguito saranno scaricate in locale tutte le risorse elencate.
  + Eventi: checking → downloading (file di differenza tra i manifest) → progress (evento scatenato per ogni risorsa da scaricare in locale) → updateready (segnala che tutte le risorse sono state nuovamente scaricate in locale)
* *Mancanza di risorse definite nel manifest:* durante lo scaricamento dei file necessari potrebbe accadere che una o più risorse non siano raggiungibili, allora verrà scatenato un evento con il fine di comunicare un errore durante l’esecuzione.
  + Eventi: checking → downloading → progress → error (evento scatenato a causa dell’impossibilità di scaricare una determinata risorsa locale).

### Local storage

Le *WebStorage* API permettono il salvataggio locale d’informazioni concernenti la navigazione web.

* Nascono per risolvere due principali problematiche relative ai cookie: lo spazio per la memorizzazione (da 4KB a 5MB) e l’impossibilità di avere due sessioni distinte in un browser relative allo stesso dominio.
* Prima del Web Storage di HTML5 ci furono altri tentativi per la risoluzione di queste due problematiche come userData di Microsoft (tecnologia proprietaria) o flash script (richiedeva installazioni aggiuntive)
* Oggi tutti i browser supportano queste API a vantaggio di una rapida ascesa Web Storage sui cookie

**sessionStorage** → consente di avere un meccanismo di persistenza dei dati distinto per ogni sessione di navigazione in ogni finestra, o scheda, del browser eliminando i dati alla chiusura.

* è possibile coordinare l’apertura contemporanea di due distinti account Gmail sullo stesso browser
* questo oggetto è da utilizzare nel caso in cui le informazioni da memorizzare sono di carattere temporaneo

**localStorage** → mantiene il comportamento dei cookie essendo comune a tutte le finestra del browser che condividono lo stesso dominio

* i dati persistono anche dopo la chiusura del browser
* permette di aggiungere articoli nel carrello e poter completare l’acquisto nei giorni successivi
* l’utilizzo ideale riguarda applicazioni in cui un processo può essere interrotto per poi essere ripreso anche a distanza di giorni

*sessionStorage* e *localStorage* implementano l’interfaccia *Storage*, che raggruppa un insieme di metodi comuni ai due oggetti.

La memorizzazione è consentita solo per valori testuali e avviene tramite una coppia chiave-valore: metodi per accedere sono *setItem()* e *getItem()*; l’ausilio dei metodi parseInt() o parseFloat() permette di memorizzare e riottenere dati in formato numerico.

* Se si richiede la memorizzazione di un oggetto, si può utilizzare JSON.stringify() (per ottenere la rappresentazione stringa di un oggetto JSON); funzione inversa è JSON.parse()
* nota: forme equivalenti per setItem(“chiave”, “valore”) e getItem(“chiave”) sono rispettivamente sessionStorage.chiave = “valore” e temp=sessionStorage.chiave

### Geolocalizzazione

Geolocation API non è parte della specifica HTML5 (invece oggetto di standardizzazione da parte del Geolocation Working Group).

* **navigator:** dà l’accesso al dispositivo hardware per la geolocalizzazione
  + .geolocation → per capire se esiste sul device la geolocalizzazione

Metodi (di navigator.geolocation)

* getCurrentPosition → rileva la posizione un’unica volta
* watchPosition → dopo la prima rilevazione, nel caso in cui la posizione abbia subito una variazione significativa, trasmette una notifica contenente le nuove informazioni. Restituisce anche un identificativo numerico univoco che servirà a localizzare i successivi spostamenti del dispositivo.
* clearWatch → per terminare la lettura ripetuta delle coordinate, il metodo accetta come argomento l’identificativo visto.

Le funzioni accettano tre argomenti: *successCallback*, *errorCallback* e *options*.

* successCallback → parametro facoltativo, rappresenta la funzione da eseguire se il rilevamento della posizione va a buon fine.
  + Alla funzione viene passato come argomento un riferimento all’oggetto Position, che ha come proprietà **timestamp** (data e ora del rilevamento della posizione) e l’oggetto **Coords** per accedere a
* longitudine, latitudine, altitudine
* accuracy (valore in metri di approssimazione), altitudeAccuracy
* heading (direzione espressa in gradi, rotazione oraria rispetto al nord) oppure NaN
* speed (velocità m/s) oppure NaN
* errorCallback → parametro facoltativo, rappresenta la funzione da eseguire se il rilevamento della posizione sia fallito.
  + Alla funzione è passato come argomento un riferimento all’oggetto *PositionError*, contenente le proprietà *message* e *code* (per comprendere la natura dell’errore). La proprietà code può assumere i seguenti valori (interi)

1. PERMISSION\_DENIED, l’utente non ha dato il consenso relativo alla rilevazione della posizione
2. POSITION\_UNAVAILABLE, il tracciamento è fallito a causa di un errore relativo al fornitore dei dati
3. TIMEOUT, il tempo prestabilito per il tracciamento è stato superato

* options → oggetto di tipo PositionOptions le cui tre proprietà *enableHighAccuracy*, *timeout* e *maximumAge* permettono di intervenire nel modo in cui avviene il tracciamento
  + enableHighAccuracy: booleana, se true richiede di determinare la posizione nel modo più accurato possibile (+ tempo di elaborazione, + consumo energetico)
  + timeout: soglia di tempo prefissata entro la quale si vuole la risposta. Valore espresso in millisecondi, se non è impostata allora vale 0
  + maximumAge: intervallo di tempo (ms) che, nel caso di acquisizioni ripetute, stabilisce il tempo che deve trascorrere per l’aggiornamento dei valori.

*Nota: il browser si fa carico di domandare preventivamente all’utente se rendere visibili al sito richiedente le informazioni relative alla posizione.*

CSS3

Il **CSS** (**Cascading Style Sheets**, *fogli di stile*) è un linguaggio usato per definire la formattazione di documenti HTML, XHTML e XML, le cui linee guida sono descritte nelle Recommendations emanate a partire dal 1996 dal W3C.

L’introduzione del CSS ha consentito la separazione dei contenuti dalla formattazione, permettendo una programmazione più chiara e facile da utilizzare, sia per gli sviluppatori che per gli utenti, garantendo anche il riuso del codice e una più facile manutenibilità.

* supporto al linguaggio HTML per la visualizzazione delle pagine.
* La specifica 3 di CSS adotta un approccio modulare, ciascun modulo copre una determinata area e il ciclo di vita e l’avanzamento è diverso per ognuno di essi
* Esistono strategie di supporto alternativo per trascurare l’incompatibilità con vecchie versioni dei browser

HTML → struttura

CSS → grafica

php → contenuti

javascript → la parte interattiva

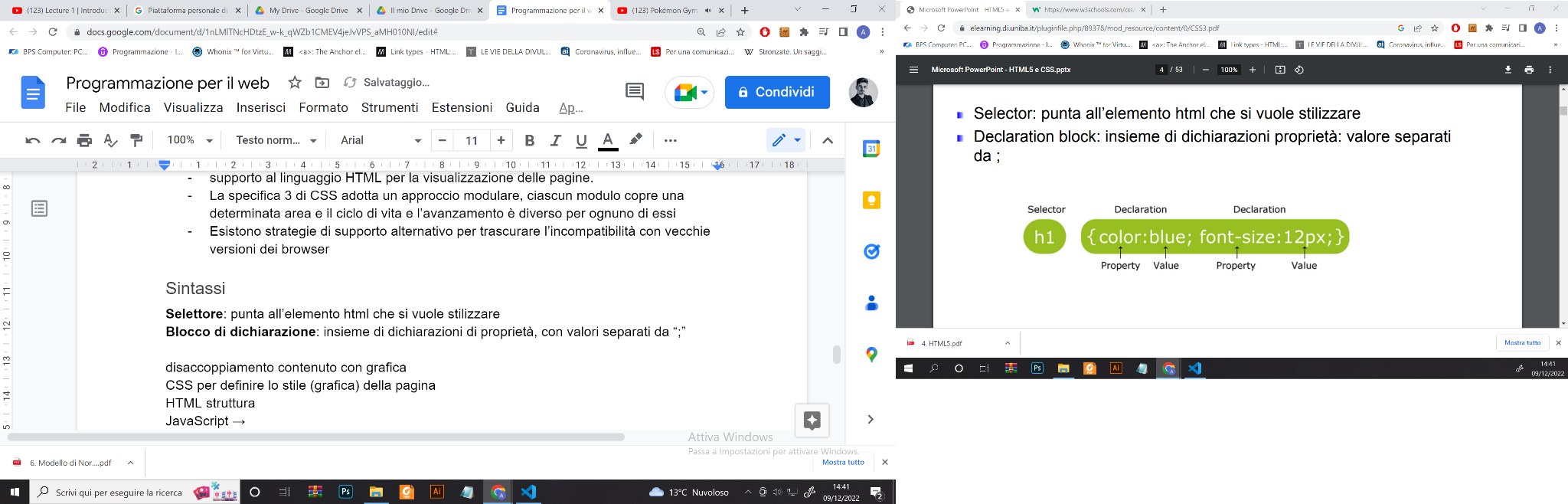
### Sintassi

domanda d’esame

*come funziona il css*

**Selettore**: punta all’elemento html che si vuole stilizzare

**Blocco di dichiarazione**: insieme di dichiarazioni di proprietà, con valori separati da “;”



Un foglio di stile CSS è una sequenza di istruzioni composte da selettore e blocco di dichiarazione: le istruzioni vengono interpretate in sequenza, per cui un’istruzione può sovrascrivere l’effetto delle precedenti. *La sequenzialità potrebbe portare problemi quando si importano css esterni (negli import di diversi file css, che entrano in conflitto, importanza ordine degli import)*

*Un selettore indica gli elementi della pagina sui quali si devono applicare le proprietà nel blocco di dichiarazione. Selettori di diversi tipi:*

* *Di elemento (selezione tutti i tag corrispondenti) → es.* ***p*** *{...}*
* *Di id (si riferisce al tag con tale valore all’attributo id) → #****para1*** *{...}*
* *Di classe (si riferisce a tutti i tag di tale classe, che sarà composta da elementi di nomi diversi da raggruppare in qualche per applicare lo stesso stile (attributo class=“nome\_classe”)) → .****center*** *{...}*

*E’ possibile fare uso combinato di diversi tipi di selettori (es. p.center {...}) oppure raggruppare stili (h1,h2,p {...})*.

**Inserimento**: è possibile utilizzare External style sheet, internal style sheet o inline style

* External → riferimento a file .css con definizione degli stili

<head>

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

</head>

* Internal: definiti con tag <style> all’interno del tag <head>
* Inline: stili definiti all’interno dei tag del contenuto html → modo peggiore per definire degli stili, si perde il disaccoppiamento tra struttura dei dati e loro visualizzazione, uno degli scopi fondamentali del css.
  + Es: <p style=“color:sienna; margin-left: 20px;”>This is a paragraph.</p>

*Nota: se lo stesso stile viene definito in diversi punti, vale il più generale*

*Nota: l’inline può essere utile, in quanto dinamicamente a run-time la pagina potrebbe essere manipolata e cambia aspetto, effetti grafici tramite JavaScript che interviene sul DOM (seleziona l’elemento e ne modifica lo style). Serve per creare dinamicità dal punto di vista grafico.*

#### Selettori di attributo

Permettono di identificare un attributo in funzione di una parte o dell’intero valore assegnato, associando un determinato stile all’elemento che lo contiene.

Il problema riguardo al loro utilizzo riguarda le prestazioni: calcolare a quanti e quali elementi applicare la regola di stile rallenta il caricamento della pagina.

Tipologie:

* p[id] → seleziona tutti i paragrafi che hanno l’attributo id
* a[href=“#”] → seleziona tutti i tag il cui valore per l’attributo “href” è “#”
* a[title~=“applicato”] → seleziona tutti i tag con attributo title contenente diverse parole, separate da spazi, tra le quali almeno una di esse è “applicato”
* a[title|=“applicato”] → seleziona tutti i tag con attributo title contenente diverse parole, separate da un trattino, tra le quali la prima è uguale ad “applicato”
* a[title^=“applicato”] → seleziona tutti i tag con attributo title il cui valore comincia per “applicato”
* a[title$=“applicato”] → seleziona tutti i tag con attributo title il cui valore termina per “applicato”
* a[title\*=“applicato”] → seleziona tutti i tag con attributo title il cui valore contiene “applicato”

**Pseudo-classi strutturali**: permettono di applicare uno stile a un elemento in funzione della posizione occupata nella struttura della pagina web. Una pseudo-classe si definisce assegnando il simbolo “:”, seguito dal nome della pseudo-classe.

Nel seguito, esempi che si applicano ad elementi di tipo E, per i quali valgono ulteriori condizioni

* E:first-child → il primo figlio di un nodo
* E:last-child → l’ultimo figlio di un nodo
* E:empty → non ha figli (non è applicato a un paragrafo contenente testo)
* E:root → indica la radice della pagina (= <html>)
* E:only-child → unico figlio dell’elemento padre
* E:only-of-type → unico figlio con il tipo specificato (E) odi elemento
* E: first-of-type → primo elemento di tipo E figlio del proprio padre
* E:last-of-type → ultimo elemento di tipo E figlio del proprio padre
* E:nth-child(n) → la regola si applica all’n-esimo figlio del padre, a condizione che sia un elemento di tipo E.
  + In alternativa, si può applicare agli elementi di tipo E, figli di un elemento padre, con indice *an + b*, con *a* e *b* numeri interi
  + altra sintassi per pari e dispari → tr:nth-child(even/odd)

**Web font**: Impostando la proprietà font-family, il browser cercherà di utilizzare il primo font. Nel caso in cui non sia disponibile, proverà con il secondo.

es. *p:{font-family: Arial, Verdana;}*

La regola @font-face permette di assegnare e rendere interpretabile qualsiasi tipo di font indipendentemente dalla macchina che visiterà il sito web. Questo è possibile perché sarà presente un collegamento ad un file contenente tutte le direttive necessarie all’interpretazione del font.

@font-face{

font-family: ‘Mio Font’; // nome arbitrario da assegnare al font, sarà riutilizzato

src: url(‘MioFont.eot’); // formato IE < 9

src:

local(‘🙂’),

url(‘MioFont.woff’) format (‘woff’) //Safari,Chrome,IE>= 9,Firefox>=3.6

url(‘MioFont.ttf’) format (‘truetype’) // Vecchi browser, iPhone, iPad

**Effetti tipografici** (nota: gli offset/scostamenti si calcolano rispetto al bordo in alto a sinistra dello schermo

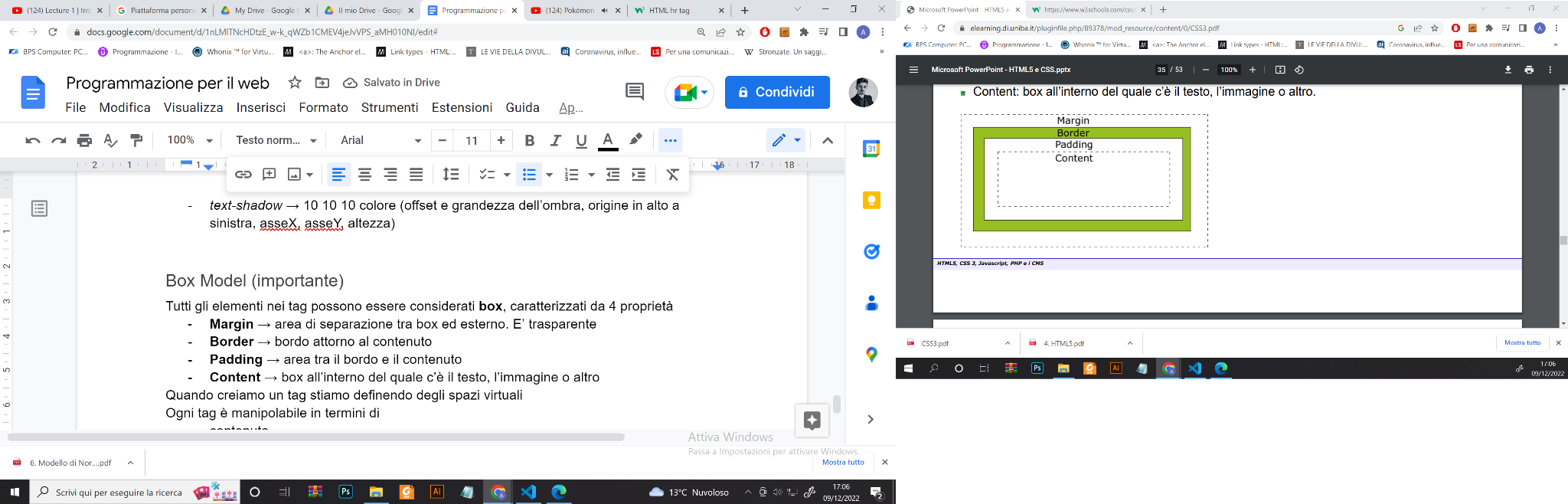
* Ombreggiato: *text-shadow* 
  + offsetX, offsetY,
  + raggio di sfumatura (pixel, 0=senza sfocatura)
  + colore
* Testo abbreviato: *text-overflow* per testo posto in contenitore non sufficientemente grande, valore “ellipsis” in modo da troncare il testo aggiungendo i puntini di sospensione
* Disposizione su più colonne
  + column-width → larghezza minima
  + column-count
  + column-gap → distanza tra colonne
  + column-rule → 3 valori: larghezza riga, stile (analogo a stili di border-style), colore
* Altri effetti: color, direction (text direction), letter-spacing, line-height, text-align, text-decoration, text-indent, text-transform (controls capitalization), vertical-align, white-space, word-spacing

Effetti grafici

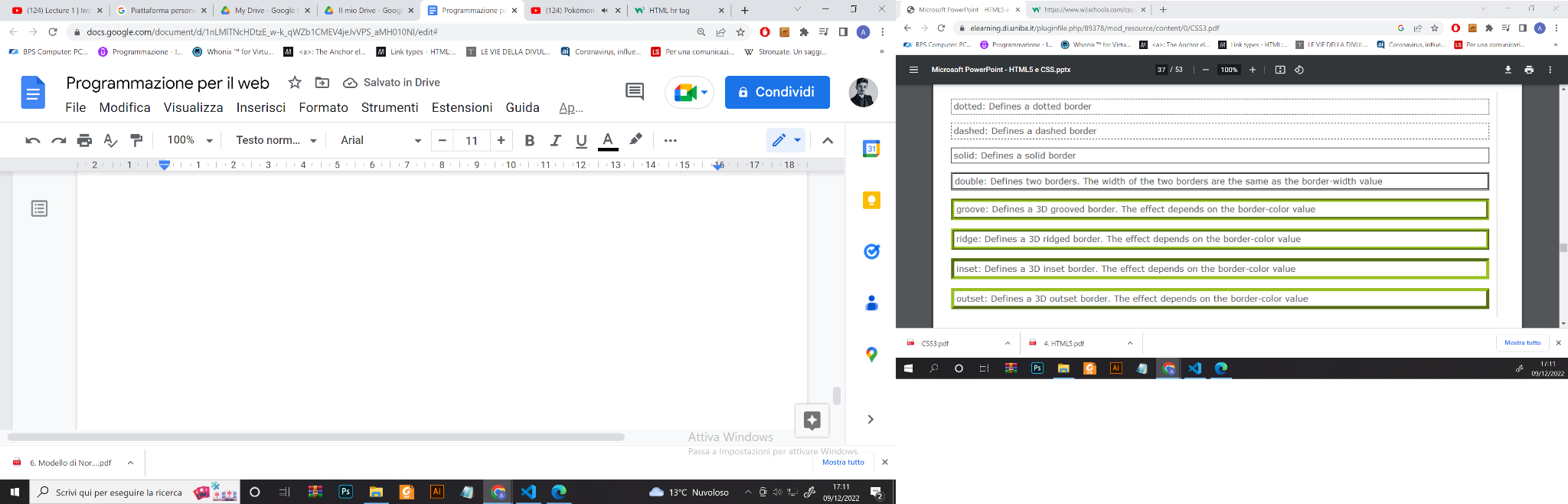
* *text-shadow* → 10 10 10 colore (offset e grandezza dell’ombra, origine in alto a sinistra, asseX, asseY, altezza)

### Box Model (importante)

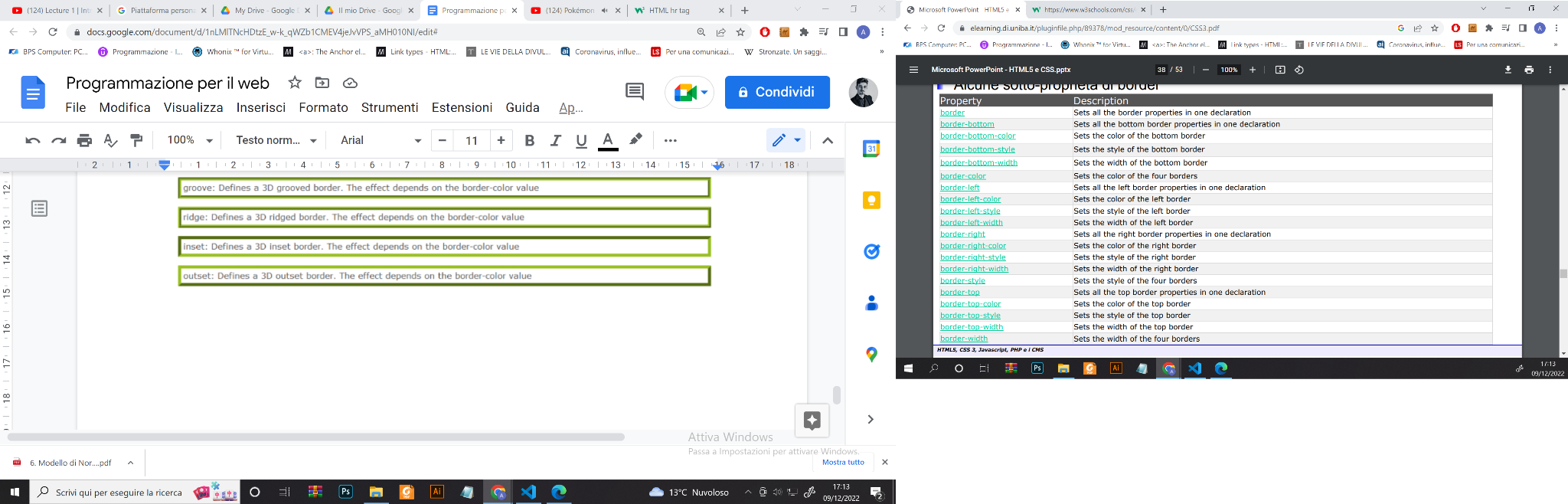
Tutti gli elementi nei tag possono essere considerati **box**, caratterizzati da 4 proprietà

* **Margin** → area di separazione tra box ed esterno. E’ trasparente
* **Border** → bordo attorno al contenuto
* **Padding** → area tra il bordo e il contenuto
* **Content** → box all’interno del quale c’è il testo, l’immagine o altro

*Nota:* se diamo un solo valore vale dappertutto (nord, sud, ovest, est), se ne diamo 2 (nord & sud, ovest & est) oppure ne diamo 4.

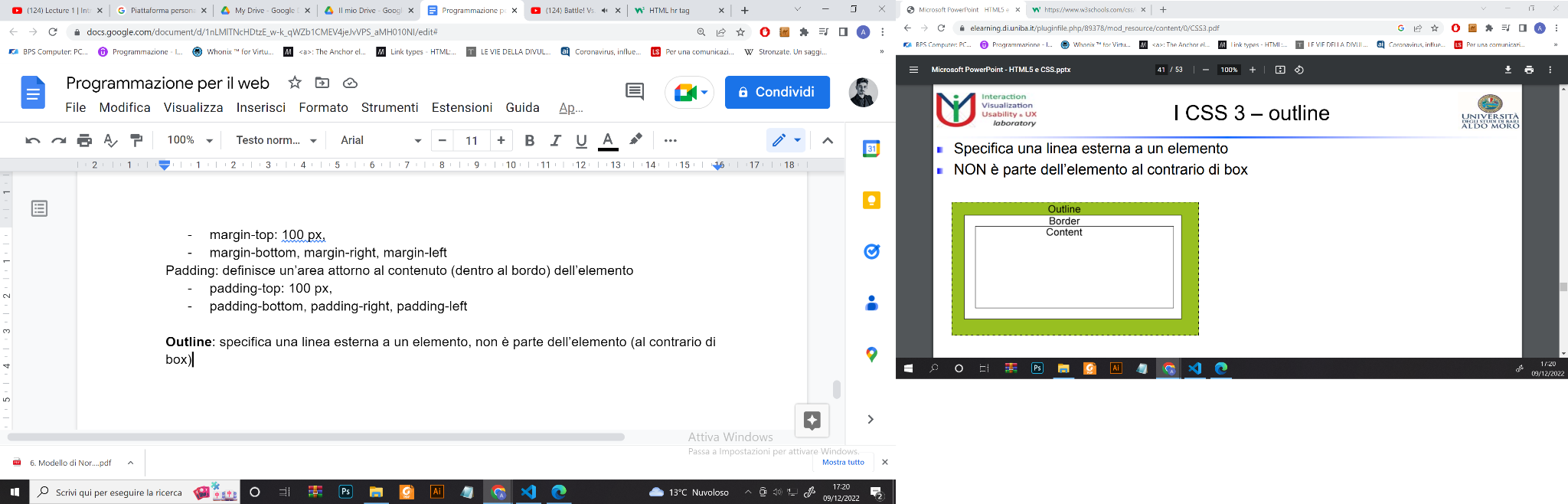


Sotto-proprietà di border



I margini definiscono un’area vuota attorno a un elemento (fuori dal bordo), trasparenti.

* margin-top: 100 px,
* margin-bottom, margin-right, margin-left

Padding: definisce un’area attorno al contenuto (dentro al bordo) dell’elemento

* padding-top: 100 px,
* padding-bottom, padding-right, padding-left

**Outline**: specifica una linea esterna a un elemento, non è parte dell’elemento (al contrario di box)

* Dimension: proprietà width e height (si potrebbe mettere “auto”)
* Gli elementi per default sono statici, seguono il normale flusso della pagina. Si può però impostare in maniera fissa una posizione rispetto al browser, alterando il flusso naturale usando l’attributo *position:fixed*
* Si può spostare un elemento relativamente alla sua normale posizione → *position:relative*; *left:20px;*
* Si può spostare l’elemento relativamente al suo primo genitore che abbia una posizione diversa da static (se non ha genitori, il genitore diventa il blocco html) → *position:absolute*

**Trasparenza**: definire trasparenza su elementi

* opacity:0.4;
* filter:alpha(opacity=40); // per IE <= 8

es.

a:hover { opacity:0.6 }

**Trasformazioni 2D**: metodi

* translate(50px, 100px)
* rotate(30deg), oppure rotateX(120deg), rotateY(120deg)
* scale(2,5)
* skew(30deg, 20deg),
* matrix(0.886, 0.5, -0.5, 0.866, 0, 0)

**Transition**: effetti che cambiano gradualmente un elemento da uno stile a un altro

* transition:width 2s

**Animazioni**: sostituiscono quelle tradizionali di Flash, immagini animate e JavaScript

@keyframes rule: indica verso dove deve evolvere l’animazione di un elemento; specificando un @keyframe all’interno del css l’animazione cambierà gradualmente dallo stile attuale a quello nuovo definito in @keyframe

* Le animazioni possono essere anche molto complesse e passare attraverso diversi momenti (con @keyframe si indica solo il momento finale ma non quelli intermedi)
* Si usano le % per indicare i vari momenti

JavaScript

**HTML Scripts**

Porzioni di JavaScript incluse nella pagina HTML, eseguite dal Browser, permettono di creare pagine web dinamiche, permettendo effetti grafici o inserendo logica di controllo (es. per form HTML.

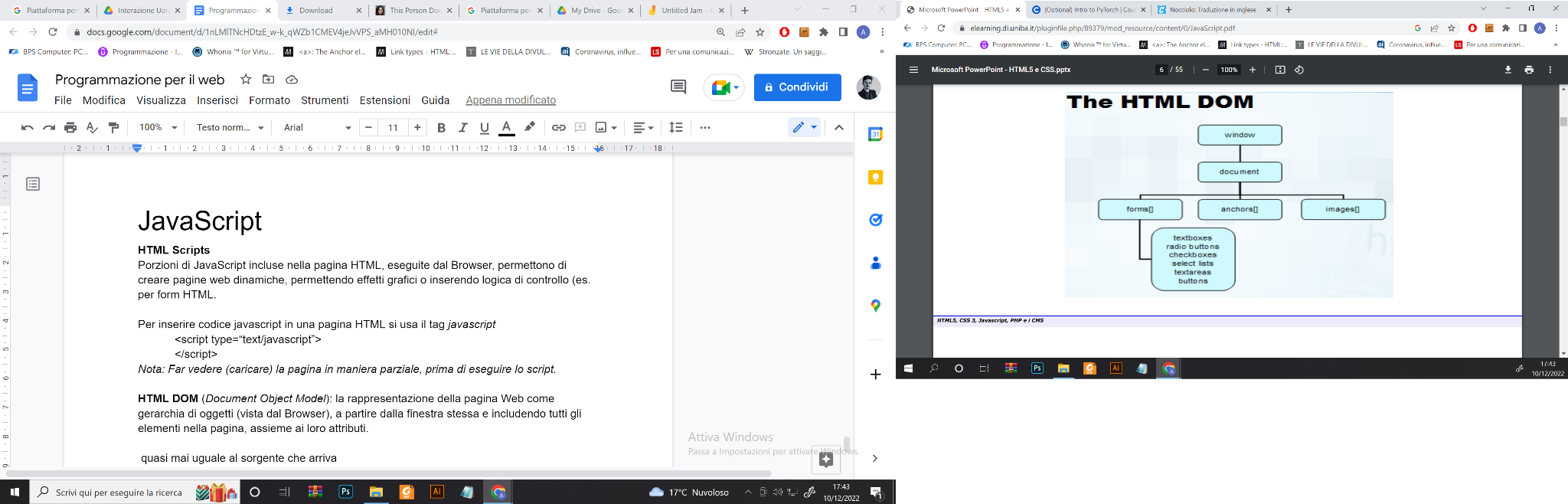
Per inserire codice javascript in una pagina HTML si usa il tag *javascript*

<script type=“text/javascript”>

</script>

*Nota: Far vedere (caricare) la pagina in maniera parziale, prima di eseguire lo script.*

**HTML DOM** (*Document Object Model*): la rappresentazione della pagina Web come gerarchia di oggetti (vista dal Browser), a partire dalla finestra stessa e includendo tutti gli elementi nella pagina, assieme ai loro attributi.



Nota: quasi mai uguale al sorgente che arriva, nel sorgente da qualche parte ci sarà dello script.

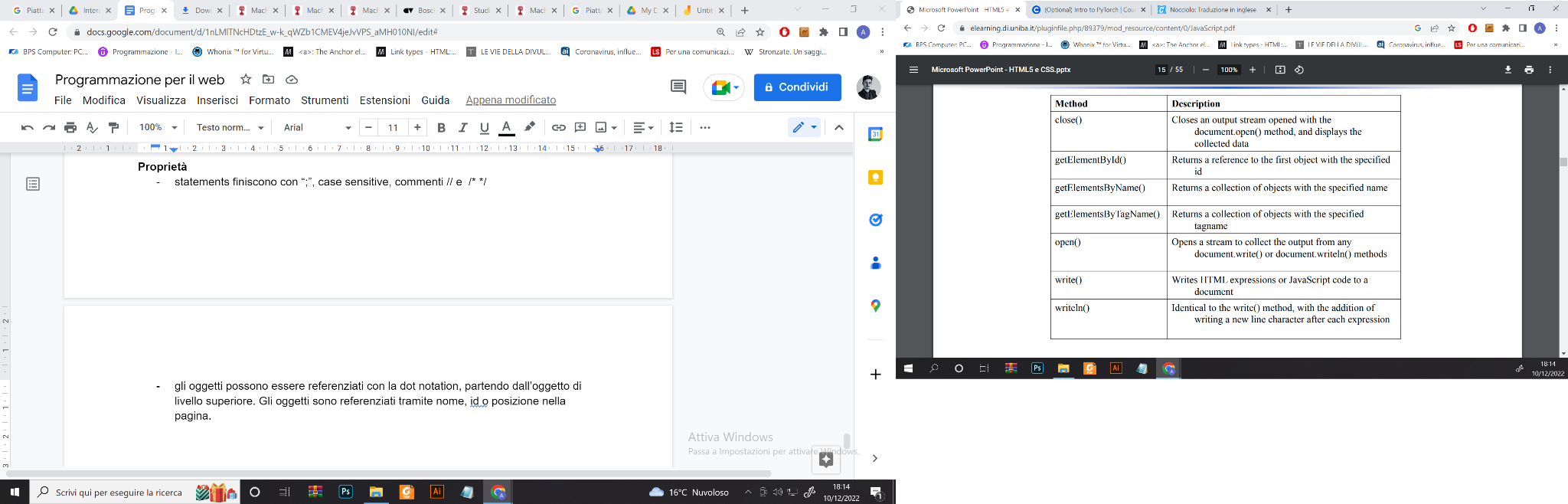
JavaScript è un linguaggio interpretato (lightweight, progettato per aggiungere interattività nelle pagine HTML), può essere embedded in pagine HTML.

* JavaScript è un tool di programmazione per designers HTML
* JavaScript può inserire testo (sorgente) dinamicamente in una pagina HTML
* JavaScript può reagire a determinati eventi (es. codice da eseguire alla fine del caricamento di una pagina o al click di un utente su un elemento html)
* JavaScript può leggere e cambiare il contenuto di elementi HTML

Inoltre può essere usato per validare dati prima di inviarli al server, può individuare il browser dell’utente e può essere usato per creare cookies.

**Proprietà**

* statements finiscono con “;”, case sensitive, commenti // e /\* \*/
* gli oggetti possono essere referenziati con la dot notation, partendo dall’oggetto di livello superiore. Gli oggetti sono referenziati tramite nome, id o posizione nella pagina.



**Variabili**: devono iniziare con una lettera o con \_

* assegnando un valore a una variabile non ancora dichiarata, essa sarà implicitamente dichiarata
* se si ridichiara una variabile, essa non perderà il suo valore originale
  + Es: var x=5; var x;

Operatori:

* + usato anche per la concatenazione di stringhe → se gli operatori sono un numero e una stringa, il numero viene convertito nella sua rappresentazione sotto forma di stringa e viene eseguita la concatenazione.
* == è “uguale a”, === è “esattamente uguale a”, includendo valore e tipo
* variablename=(condition)?value1:value2

**Funzioni**: ponendo script in funzioni il browser non eseguirà lo script al caricamento della pagina → una funzione è richiamata a seguito di un evento, la si può richiamare da una qualsiasi posizione nella pagina, o anche da altre pagine se la funzione è contenuta in un file esterno .js

* Una funzione può essere definita sia nell’<head> che nel <body> di un documento
* Per assicurarsi che la funzione venga letta/caricata dal browser prima di essere chiamata, è opportuno inserirla nell’ <head>.

Sintassi per definire una funzione

function *functionname(var1, var2, …, varX) { … }*

Se si dichiara una variabile fuori da una funzione, tale variabile appartiene allo scope globale della pagina (tempo di vita inizia alla dichiarazione e termina alla chiusura).

In JavaScript una funzione è un **oggetto di prima classe**, ovvero un’entità che può essere dinamicamente creata, distrutta, passata ad una funzione, assegnata ad una variabile e restituita come valore.

* *Immediately-invoked function expression* (**IIFE**): l’intera espressione viene racchiusa tra parentesi tonde, altrimenti il parser JavaScript, trovando la parola chiave function, tenterebbe di interpretare l’espressione come una definizione di funzione nel contesto globale
* Una funzione prevede proprietà e metodi (essendo un oggetto)
  + name, length (numero argomenti previsti)
* La funzione passata come parametro è detta generalmente *funzione di callback*
  + es: function mostraValore(valore, indice) {...}
    - var numeri = [11, 3, 24];
    - numeri.forEach(mostraValore);

**Async**

completare