

LINEE GUIDA DA RISPETTARE ED OSSERVAZIONI UTILI

Mettere tutti i nomi di sintassi in CORSIVO e non tra "virgolette"

LAB 4-10-24

- [HTML] Chiamiamo il file della home "**index.html**". In generale il file principale della pagina che viene mostrata inizialmente all'utente si chiama "index" perché il web server di default il primo file che cerca si chiama index (.html, .php, etc.).
- [HTML] I due tag che dobbiamo inserire sempre all'interno della nostra pagina HTML (dunque all'interno del tag principale che contiene tutta la nostra pagina "html") sono "**head**" (dove vanno specificati i metadati, l'unico obbligatorio è il tag "title") e "**body**".
- [HTML] Il contenuto principale della pagina va inserito all'interno del tag "**main**". Un contenuto che è presente all'interno della pagina ma non è centrale dovrebbe essere "**aside**".
- [HTML] Il **testo** può essere inserito o all'interno di un titolo, quindi nei tag "**h1**"-"**h6**", o di un **paragrafo**.
- [HTML] Il tag "**a**" per i link è un elemento di linea e va dunque inserito all'interno di un **testo** o in generale di un **contenitore**, per evitare di inserirlo direttamente in un elemento macro di blocco. La strategia più comune consiste nell'**organizzare i link come lista non ordinata**.
- [HTML] Un cancelletto (#) all'interno di un link fa self-reference, dunque quando l'utente lo clicca rimane sulla stessa pagina. È utile da utilizzare come placeholder, quando ancora non si sa il link di destinazione.
- [HTML] La differenza tra "**main**" ed "**article**" è solo semantica: entrambi sono elementi di blocco, cambia solo l'obiettivo del contenuto (così come per "header", "footer", "nav", "aside", etc.).
- [HTML] L'immagine la inseriamo dentro ad un "**div**" un blocco generico, senza particolare significato semantico, che usiamo per avere delle immagini di dimensioni proporzionate nella parte di stile.
- [HTML] Quelli come "**img**" sono detti empty tag, poiché non hanno tag di apertura e chiusura con del contenuto all'interno ma un unico tag.

- [HTML] Il codice compliant e well-formed prevede che gli empty tag abbiano lo SLASH FINALE per chiudere il tag.
- [HTML] Specificare i percorsi relativi, cosicché nel caso variasse l'URL del sito o lo spostassimo in un una sottocartella non dovremmo cambiare il codice.
- [HTML] Definire "h1" solo titoli di ugual importanza rispetto a quello della pagina (raro).
- [HTML] Quelli che sono bottoni lato CSS, sono link in HTML.
- [HTML] Non utilizzare mai la parte di HTML per gestire aspetti di presentazione (salvo eccezioni, come avvolgere le immagini da un div). Quando scriviamo codice HTML dobbiamo trattare esclusivamente il contenuto della pagina, tutta la parte di presentazione viene gestita con CSS.
- [HTML] Lo screen reader si muove all'interno della pagina su uno stesso tipo di tag, e.g. potrebbe scorrere tutti gli "h2". Questo rappresenta un elemento per stabilire il livello e dunque l'importanza da dare ad un titolo: voglio che dopo i titoli degli articoli vengano menzionati anche "Post popolari" e "Categorie"? La pagina HTML noi possiamo vederla come un albero di tag in cui ci sono delle relazioni, quindi abbiamo <html> che è la radice, i due figli <head> e <body>, dove quest'ultimo contiene a sua volta 5 figli: <header>, <nav>, <main>, <aside> e <footer>. Quindi possiamo vedere la pagina come una gerarchia di tag. Un tag è figlio di un altro quando è inserito direttamente all'interno del primo tag. Poi c'è la relazione di discendenza, più generale, per la quale tutti i tag all'interno di un altro sono considerati suoi discendenti (<a> è figlio diretto di ma anche discendente di e <nav> etc.). Infine abbiamo la relazione di fratellanza, tra elementi che hanno lo stesso padre (i sono fratelli), ed hanno generalmente lo stesso livello di intestazione.
- [HTML] Il testo nel footer non è un titolo ma contenuto, per cui andrà inserito in <p>.
- [HTML] Il titolo della pagina è riportato nel tab.
- [HTML] Per rendere le tabelle accessibili, ad ogni cella di intestazione assoceremo un id, poi nelle celle di dati incroceremo gli id delle celle di intestazione di riga e di colonna (e.g. [1^origa] <th id="autore">Autore</th>, ... , [2^origa] <th id="ginopino">Gino Pino</th>, <td

headers="ginopino email">ginopino@logtw.com</td>).

- [HTML] Il collegamento tra una cella e la sua intestazione di colonna è immediato, motivo per cui nelle intestazioni di riga non specifichiamo la colonna corrispondente, ma il discorso sarebbe stato lo stesso se invece che <th> sarebbero stati <td> (in questo caso non sarebbe servito nemmeno specificare gli headers nelle altre <td> poiché banali). Il problema sorge quando ho la doppia cella di intestazione.
- [HTML] Quando noi vogliamo dare la possibilità all'utente di compilare dei campi per inviare delle informazioni ad un server dobbiamo utilizzare i form. L'attributo "method" indica il metodo HTTP che viene usato per inviare le informazioni: l'invio del form da vita ad una richiesta HTTP, che può essere di tipo GET (dove i parametri che scriviamo viaggiano nell'URL, quindi sono visibili nella history del browser) o POST (dove i dati sono nel corpo della richiesta; il fatto che non siano visibili all'utente non implica che siano crittografati, sono comunque in chiaro quindi se siamo in ascolto sulla rete ed intercettiamo il pacchetto HTTP i dati li vediamo comunque). La "action" indica la pagina a cui noi vogliamo mandare i dati (possiamo ancora una volta usare il cancelletto come placeholder).
- [HTML] All'interno di un form usiamo le <label> e non i <p> per motivi di accessibilità, potendo collegarle all'<input> (in questo input devo inserire questa informazione). Organizziamo ogni coppia di label & input all'interno di un di una . Collegiamo i due elementi della coppia o inserendo il campo input direttamente all'interno del tag <label> (sconsigliato) o tramite l'attributo "for" del tag <label>, dove specifichiamo l'id dell'<input> a cui si riferisce. Se ho collegato la <label> all'<input> correttamente, cliccando sulla label verrà fatto focus sull'input corrispondente.
- [HTML] Che differenza c'è tra gli attributi "name" e "id" di <input>? "name" è la chiave che viene usata quando i dati vengono inviati (e con cui vengono poi recuperati), "id" invece serve per collegare la label.
- [HTML] Specificare come "type" il valore "password" all'interno di <input>, che oscura i caratteri durante l'inserimento, mi garantisce sicurezza quando i dati vengono inviati? No, vengono sempre inviati in chiaro o nell'URL o nel corpo della richiesta se uso POST. Se vogliamo sicurezza dobbiamo usare HTTPS come canale sicuro all'interno del quale inviare le informazioni.
- [HTML] I bottoni sono elementi di input con type "submit" per i quali non è richiesta una label.

LAB 18-10-24

All CSS Simple Selectors

Selector	Example	Example description
<code>#id</code>	<code>#firstname</code>	Selects the element with id="firstname"
<code>.class</code>	<code>.intro</code>	Selects all elements with class="intro"
<code>*</code>	<code>*</code>	Selects all elements
<code>element</code>	<code>p</code>	Selects all <p> elements
<code>element,element,...</code>	<code>div, p</code>	Selects all <div> elements and all <p> elements

What is Greater-than Sign (>) Selector in CSS?

Last Updated : 20 Sep, 2024



In CSS, the greater than sign (>) is known as the child combinator selector. It is used to style elements that have a specific parent. Unlike other selectors, it only targets direct children of an element, meaning it only looks one level down in the HTML structure.

How the Child Combinator Selector Works

The child combinator is written as:

```
parent > child
```

This means it selects the child element that is directly inside the parent element. Any elements that are not direct children, even if they are further down the structure, won't be selected.

Syntax

```
element > element {  
    // CSS Property  
}
```

In this code, the child-element will only be styled if it is a direct child of the parent-element.

Attribute selectors



Baseline Widely available



The CSS **attribute selector** matches elements based on the element having a given attribute explicitly set, with options for defining an attribute value or substring value match.

Syntax

[attr]

Represents elements with an attribute name of *attr*.

[attr=value]

Represents elements with an attribute name of *attr* whose value is exactly *value*.

[attr~=value]

Represents elements with an attribute name of *attr* whose value is a whitespace-separated list of words, one of which is exactly *value*.

[attr|=value]

Represents elements with an attribute name of *attr* whose value can be exactly *value* or can begin with *value* immediately followed by a hyphen, `-` (U+002D). It is often used for language subcode matches.

[attr^=value]

Represents elements with an attribute name of *attr* whose value is prefixed (preceded) by *value*.

[attr\$=value]

Represents elements with an attribute name of *attr* whose value is suffixed (followed) by *value*.

[attr*=value]

Represents elements with an attribute name of *attr* whose value contains at least one occurrence of *value* within the string.

[attr operator value i]

Adding an `i` (or `I`) before the closing bracket causes the value to be compared case-insensitively (for characters within the [ASCII](#) range).

[attr operator value s]

Adding an `s` (or `S`) before the closing bracket causes the value to be compared case-sensitively (for characters within the [ASCII](#) range).

There are four different combinator in CSS:

- Descendant Selector (space)
- Child Selector (>)
- Adjacent Sibling Selector (+)
- General Sibling Selector (~)

- [HTML] Non specificare il css inline ma collegare un file esterno per separare contenuto da presentazione.
- [HTML] Ricordarsi lo slash di chiusura prima di ">".
- [HTML] Indicare sempre il tipo del file menzionato in "link" tramite l'attributo "type".
- [CSS] Resetare lo stile di default del browser (almeno margine e padding).
- [CSS] Non usare gli id e le classi per cambiare lo stile di un elemento specifico ma sfruttare le relazioni per identificarlo.
- [CSS] Specificare il font direttamente nell'elemento che si intende cambiare (quindi non in "header se vogliamo modificare h1, sebbene questo erediti il font del padre, cosa che non accade per proprietà come larghezza, margin e padding).
- [CSS] Selezionare i tag HTML sfruttando il percorso di parentela (se scrivessi solo h1 cambierei anche un possibile h1 scritto in seguito).
- [CSS] Inserire margin e padding del body & header nella regola di reset, anche se ho già altre regole aperte per i due.
- [CSS] Scegliere tra padding del contenitore e margin dell'elemento contenuto e rimanere coerenti con la scelta.
- [CSS] Fornire i file del font nel caso sia molto particolare e potrebbe non essere installato (utilizzare Google Font).

- [CSS] Layout fluidi che si adattano alla larghezza della pagina: specificare il margin orizzontale in percentuale, e quello verticale in pixel. Questo perchè l'utente è abituato a fare scroll verticale e non orizzontale, quindi lateralmente vorremo adattarci al massimo.
- [CSS] Fare attenzione ad eventuali padding o margin messi automaticamente in certi elementi (e.g. padding a sinistra nell'ul): inserire per sicurezza tali elementi nella lista di quelli a cui è destinata la regola di reset.
- [CSS] Per far apparire in linea elementi di blocco come le ul utilizzare la proprietà "display", consapevoli del fatto che poi verranno combinate le caratteristiche delle due tipologie, e.g. sarà possibile specificare una larghezza (funzionalità non supportate negli elementi inline, che occupano la larghezza del contenuto) ma gli elementi non andranno a capo.
- [CSS] Se indico il color su un listItem non viene ereditato dall'elemento "a", perchè i link hanno un loro stile predefinito, che prevale su quello del listItem poiché il selettore è più specifico.
- [CSS] Vogliamo che il testo "a" all'interno di un bottone si illumini anche se sono sopra il bottone ma non sopra il testo con il cursore (e dunque che l'utente venga re-indirizzato quando questa parte viene cliccata), per questo motivo devo allargare l'area di "a" a tutto il "listItem" (e non gestire gli spazi con il padding internamente al "listItem", altrimenti l'area di a rimarrebbe la stessa). Tuttavia per far ciò dovremmo cambiare la natura in-line di "a" attraverso "display: inline block" poiché altrimenti occuperebbe solamente la larghezza del contenuto, ed impostare "height" e "width" a 100%. Anche se ci sarà del padding per centrare il testo il link si attiverà anche ai bordi.
- [CSS/HTML] Fare i conti bene e NON decrementare a tentativi la percentuale del margin per far sì che le listItems si dispongano nella maniera desiderata. Sebbene potrebbe apparentemente funzionare nelle particolari circostanze in cui ci si trova, quando l'1% è inferiore ai 2 pixel, andrà a capo comunque. Un fattore che potrebbe compromettere i conti è l'andata a capo nell'HTML, in effetti azzerando padding e margin, i container risulteranno comunque spaziati. Questo perchè, quando noi scriviamo il testo e mettiamo uno spazio tra due caratteri, questo viene visualizzato. Se noi trasformiamo gli elementi di blocco in elementi di linea inizieranno a contare anche gli spazi, con la regola che tutti gli spazi/andate a capo consecutivi (andata a capo della i-esima riga e gli spazi/tab di indentazione della

$i+1$ esima) verranno collassati in un unico spazio a livello di renderizzazione. Si tratta di uno dei pochi casi in cui è consentito violare la separazione tra contenuto e presentazione (cambiamo l'HTML per gestire un aspetto di presentazione). In HTML, solo gli spazi tra "" e "" mi danno problemi, all'interno non li ho.

- [CSS] Piuttosto che specificare singolarmente il margin per ogni direzione, così come indicare separatamente il colore, la larghezza e lo stile del bordo, utilizzare rispettivamente le proprietà aggregate "margin" e "border".
- [CSS/HTML] Per motivi di stile, nel codice HTML racchiudiamo le immagini all'interno di un "div" poiché ci consente di specificare un'altezza massima (e.g. 200px) e poi dire all'immagine di prendere larghezza ed altezza massima (entrambe 100%). Se l'immagine è verticale, quindi altezza > larghezza, occuperà 200px di altezza e la larghezza verrà calcolata in accordo. Se succede il contrario, occuperà tutta la larghezza a disposizione e ci sarà più spazio tra l'immagine ed il resto.
- [CSS] Il valore attuale di un margin ha sempre precedenza rispetto al massimo selettore (nuovo).

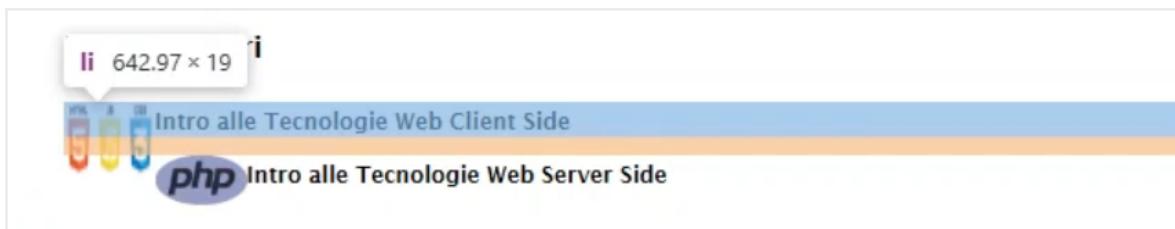
LAB 25-10-24

- [CSS] Quando si parla di margine verticale, non si fa la somma tra il "margin-bottom" di un elemento ed il "margin-top" di quello sottostante ma prevale quello più grande.
- [CSS] A parità di specificità (valore basato sull'inclusione del foglio di stile in-line o meno, numero di id, numero di classi, numero di tag, pseudo-classi etc.) vale la regola più recente (e.g. "margin" di un elemento modificato due volte).
- [CSS] Se il cliente volesse un colore diverso per l'hover, dovremmo cambiarlo tante volte, in ogni punto in cui avevamo inserito quello precedente. Questo perché CSS non nasce come linguaggio di programmazione, quindi senza variabili (nonostante le ultime specifiche lo stiano introducendo, o superset del linguaggio come Sass con funzioni, cicli etc.). La soluzione consiste nel raggruppare tutti i selettori di elementi di egual colore separandoli con virgole.
- [CSS] La proprietà "float" rimuove un elemento dal flusso normale del documento e lo "butta" da un lato.

PRIMA: il listItem è costituito da due elementi, uno più alto dell'altro, per cui, per contenere tutto, l'altezza corrisponde a quella dell'oggetto più alto.



DOPO: Rendendo floating le immagini, queste verranno rimosse dal contenuto della pagina e non saranno più considerate figlie di qualcuno, e verranno dunque trascurate nel calcolare l'altezza di ciascun listItem, non facendone più parte (in generale non saranno più un elemento che modifica le altezze dei contenitori e le varie occupazioni). L'unica cosa per cui continuano ad essere all'interno della pagina è il fatto che, in qualità di elementi, occupano dello spazio, per cui i link iniziano immediatamente dopo le immagini. Il secondo listItem non inizia al di sotto del bordo inferiore dell'immagine soprastante, ma guarda solo il margin-bottom del primo listItem. Mettendo delle immagini float una affianco all'altra queste, se possibile, verranno disposte nella stessa linea, altrimenti potrebbe formarsi una sorta di diagonale, poiché ciascuna cercherà di partire dal primo (verticalmente) spazio a sinistra a disposizione. In conclusione gli elementi float hanno influenza sui testi laterali e si influenzano a vicenda, ma non influenzano i "padri": i contenitori listItem sono infatti posizionati a prescindere dai due elementi float.



Post Popolari



- [CSS] Una differenza tra "float" e "display". Il problema nell'usare "display" per l'allineamento è che dipende dal codice HTML, cosa che ci porta a violare la separazione tra contenuto e presentazione. Usando float questo vincolo verrebbe rispettato (o comunque infranto leggermente poiché si richiede una certa organizzazione degli elementi, ma non di certo la rimozione degli spazi).

- [CSS] Attenzione alla generalità dei selettori: se indicassimo solamente "footer" come selettore potremmo non renderci conto dello star applicando la regola anche ai "footer" degli, ad esempio, "article".
- [CSS] Per passare dal layout mobile a quello desktop impostiamo un "breakpoint" in pixel (e.g. 768px, che equivarrebbe approssimativamente alle dimensioni di un tablet) attraverso la media-query "@media screen and (min-width:768px){}" dove "screen" identifica il supporto. Se entrambe le condizioni sono verificate (abbiamo uno schermo & la sua larghezza è almeno quella specificata) allora si applicano le regole contenute nelle grappe.
- [CSS] Come facciamo a stabilire, guardando al foglio di stile di un sito, se questo è stato progettato mobile o desktop-first? Lo capiamo dalla seconda condizione della media-query: se viene utilizzato "min-width" significa che è stato architettato secondo il primo approccio, viceversa nel caso di "max-width".
- [CSS] Come faccio a calcolare, nel caso del menu di navigazione con 4 voci disposte orizzontalmente, la larghezza dei listItem e dei margini ad essi interposti in modo che rientrino nel 100% di larghezza della pagina? Sottraggo a 100 la somma della larghezza dei 4 listItem e devo ottenere un numero divisibile per 3, dovendo anche i 6 margini aver uguali dimensioni. Sappiamo inoltre che i listItems dovranno essere meno larghi di 25% (caso senza margin). È possibile che siano larghi il 20? No, perché i 3 (o 6) margini dovrebbero poi spartirsi il 20% di spazio. Con 19%, ci rimarrebbero 24px, equamente distribuibili ai margini. Attenzione a non assegnare un margin-right maggiore di zero all'ultimo listItem.
- [CSS] Nei selettori, non inserire spazi bianchi tra i due punti che seguono il nome di un elemento ed una pseudoclasse, pena funzionamento imprevedibile.
- [CSS] Venendo gli elementi di blocco posizionati uno sotto all'altro, per affiancare (orizzontalmente) "main" ed "aside" dovrò cambiarne la natura, con "display: inline-block". Ci accorgeremo tuttavia che l'aside si troverà accanto alla parte finale del main (il secondo "article", e non al primo), poiché di base gli elementi inline sono elementi di testo, ed il testo quando ha elementi che sono uno più alto dell'altro, vengono allineati in basso, non in alto: le lettere quando hanno altezze diverse, sono tutte allineate in basso. I due blocchi (main ed aside) vengono dunque trattati come testo che, come abbiamo detto, viene allineato in basso. Per ovviare a ciò faccio uso della proprietà "vertical-align: top".

- [CSS] Se guarda l'immagine, di default il testo viene sempre allineato alla base. Se nota, le lettere più "alte" (l, b, i, d, ...) sono allineate alle altre alla base. L'aside, una volta cambiato il valore della proprietà display in inline-block, segue questo comportamento, di default è allineato alla base. Con vertical-align, viene allineato top invece.
- [CSS] Attenzione ai bordi durante il calcolo delle larghezze nel passare dal layout mobile a quello desktop. Specificare la proprietà "box-sizing: border-box" per l'elemento con bordo, anche al di fuori della media-query, non avendo ripercussioni a schermo piccolo. Il valore "border-box" funziona anche sul padding e non solo sul bordo. Quando lo specifichiamo su, e.g. "aside", una volta definita la sua larghezza, questa sarà comprensiva di padding e bordi, i quali verranno sottratti ad essa e non sommati.
- [CSS] ATTENZIONE: La dimensione del padding va aggiunta alla dimensione che sia larghezza o altezza. Quindi se noi impostassimo una larghezza del 100% ed inserissimo un padding a dx e sx del 10% avremmo un elemento che in totale occupa il 120% spazio. Quindi il padding, così come il bordo, si va a sommare alla larghezza per definire lo spazio occupato dall'elemento.
- [CSS] Orizzontalmente non vi è il fenomeno di margin collapsing dei margini osservato sull'asse verticale.
- [CSS] Usando body > aside, la specificità del selettore è 0002. Essendo che nella media query il selettore è aside (specificità 0001), la regola nella media query NON sovrascrive la precedente. Se invece imposta anche il selettore della media query con body > aside, non ci dovrebbero essere differenze.
- [HTML] Simulando per mezzo della Toggle Device toolbar la visualizzazione del sito nel display di un iPhone, ci accorgeremo che verrà mostrata la versione desktop del sito, nonostante questa dovrebbe entrare in azione solamente per schermi di larghezza maggiore di 768px. Questo perché per ragioni storiche (per garantire compatibilità con il 98% dei siti che non erano ottimizzati per dispositivi mobili) le pagine vengono caricate su delle finestre virtuali (viewport) di grandi dimensioni, per cui il sito viene adattato al viewport per poi essere scalato alle dimensioni dello schermo dello smartphone. Per evitare ciò bisogna inserire nel codice HTML "<meta name='viewport' content='width-device-width, initial-scale-1.0'>".
- [CSS] Se dessimo il bordo all'elemento "table" visualizzeremmo

solamente quello esterno. Dovremmo darlo anche agli elementi "td" e "th", rimuovere gli spazi tra i bordi di celle contigue e tra i bordi di celle e quello della tabella lavorando a livello di tabella con la proprietà "border-spacing: 0", per poi collassare i bordi "raddoppiati" (dati dall'accostamento di due bordi precedentemente separati da spazio) con "border-collapse: collapse". Infine si potrebbe anche eliminare il bordo della tabella.

- [CSS] Possiamo omettere il padding a destra ed a sinistra nelle celle di una tabella poiché, allineando il testo al centro, se abbiamo una cella che ha più spazio a disposizione abbiamo già della spaziatura tra il testo ed il bordo del contenitore.
- [CSS] "La tabella deve occupare il 100% dello spazio a disposizione (80% per schermi grandi)" si intende relativamente al suo contenitore. Se dovessi poi centrarla non potrei usare text-align non essendo un elemento di testo, ma dovrei fare i calcoli con i margin: occupando l'80% spazierò sia a destra che a sinistra del 10%.
- [CSS] TIPICAMENTE IL FOGLIO DI STILE È SEMPRE UNO. La struttura è difficile che cambi molto da una pagina all'altra dello stesso sito. Solitamente si fa un unico foglio di stile anche per performance. È comodo avere un unico foglio di stile e non fare un foglio di stile per ogni pagina per un discorso di cache. È vero che è inutile scaricare tutto lo stile, anche di pagine che non sto visualizzando, ma il codice in termini di byte occupati è talmente leggero che sarebbe molto più costoso fare una richiesta aggiuntiva al server per un ulteriore file. Tutti i browser moderni mettono in cache tutti i file statici (static files are typically files such as scripts, CSS files, images, etc... that don't change when your application is running and aren't server-generated, but must be sent to the browser when requested).
- [CSS] Di default l'input ha un padding ed un bordo, quindi devo applicare la proprietà "box-sizing: border-box" per far tornare i conti sulla larghezza.
- [CSS] Per distinguere gli elementi di input come le caselle di testo dai bottoni nei selettori posso far riferimento al tipo (che trovo nel codice HTML), e.g., "form ul li input[type="submit"]{}", dove in generale type rappresenta un attributo ed il testo compreso tra i doppi apici il valore. Se invece volessi applicare un regola a tutti gli elementi di input eccetto quelli di tipo "submit" posso sfruttare la classe "not": "form ul li input:not(type="submit")){}".
- [CSS] Per rimpiazzare il cursore con la manina quando mi trovo sopra

ad un bottone devo, nella classe hover del bottone, usare "cursor:pointer".

CORREZIONE ESERCIZI IN PREPARAZIONE ALL'ESAME 19-11-2024

- [HTML] Il tag radice <html> ha un attributo obbligatorio, ossia "lang", attraverso cui viene indicata al browser la lingua utilizzata. Non vi sarà alcun controllo sulla coerenza del valore assegnato con il contenuto del documento tuttavia, nel caso non vi fosse, insorgerebbero problemi di accessibilità: lo screenreader leggerebbe, ad esempio, un documento in inglese con la pronuncia italiana. Inoltre favorisce i motori di ricerca ed i sistemi che fanno parsing non solo del codice ma anche del contenuto.
- [HTML] <meta charset="utf-8" /> è un metadato che andiamo ad inserire all'interno di <head> che ci permette di definire un set di caratteri, in questo caso l'utf-8, grazie al quale non dobbiamo gestire differentemente (utilizzando delle entità) le lettere accentate tipiche della lingua italiana.
- [HTML] Se, nel testo dell'esercizio, invece che *intestazione di 1°livello*, ci fosse scritto *intestazione* sarebbe corretto mettere <h2> o <h3>? No, perché c'è anche scritto che è necessario rispettare la semantica dei tag: <h1> è il titolo di 1°livello e non essendoci delle intestazioni di maggiore importanza prima, dovrò necessariamente iniziare da <h1>.
- [HTML] Per ciascuna parte in cui è diviso un <form> dovrò usare un tag <fieldset>, al cui interno specifico il titolo del modulo in <legend>. Andiamo poi ad inserire tante coppie label-input quanti sono i dati richiesti nella porzione di form. Nel tag <label> devo specificare nell'attributo "for" lo stesso valore che andrò ad assegnare all'attributo "id" del corrispondente tag <input>, per l'associazione esplicita. Utilizziamo "for" poiché sarebbe contraddittorio cercare di associare i due tag assegnandoli il medesimo "id", dovendo questo essere univoco all'interno della pagina (non saremmo in grado di identificarli in JS). Nel tag <input> inseriamo inoltre gli attributi "type", "placeholder" ed eventualmente "value" (per inserire già un valore valido per il submit, potrebbe tornare utile per l'auto-fill dei campi richiesti in fase di acquisto, nel caso l'utente abbia effettuato il log-in).
- [HTML] Come potrei visualizzare le coppie label-input in righe diverse

e non una affianco all'altra?

- (1) Forzando l'andata a capo con dei tag
 (line break)
- (2) Modificando il foglio di stile
- (3) Usando delle liste puntate, dove i punti vengono però nascosti nel CSS

- [HTML] Il validator ci segnala come "info" lo slash per gli elementi puntuali
 poiché se noi avessimo degli script JS legati ad un attributo di gestione di un evento e non avessimo usato correttamente le virgolette per contenere i valori degli attributi allora potrebbero esserci dei conflitti nell'utilizzo del simbolo "/".
- [HTML] Nel tag di <input> relativo al codice fiscale impostiamo l'attributo "maxlength" a 16, in quello associato alla data di nascita imposteremo invece l'attributo "type" a "date". Per controlli più accurati (e.g. la lunghezza minima, il fatto che la stringa debba essere composta da 6 lettere seguite da 6 cifre) dovremmo usare JS per analizzare la stringa inserita.
- [HTML] Nel caso fosse necessario specificare l'indirizzo il comportamento più corretto consisterebbe nell'usare <fieldset>, <legend> (con valore "indirizzo"), e poi i campi <input> con il nome della via, il civico ed il cap (number), città e provincia (limitando la stringa a 2 caratteri oppure preparando una <dataList> con tutte le provincie italiane).
- [HTML] Se renderizzando la pagina mi dovesse accorgere che il placeholder viene troncato per via delle dimensioni ridotte del campo di <input>, potrei specificare nel relativo tag l'attributo "size" ed esprimere una lunghezza in termini di numero di caratteri (influisce solo sulla visualizzazione, non va a limitare la stringa da inserire).
- [HTML] In "*Assicurazione, a scelta (radio) tra Primo Livello, Secondo Livello e Terzo Livello*" è semanticamente richiesto annidare un <fieldset> (oppure un paragrafo) con <legend> "Assicurazione" all'interno di quello più esterno con <legend> "Preferenze" (nonostante visivamente ridondante) poiché avremo tre coppie label-input, con <input> di tipo radio ed una <label> specifica per ogni tipologia di assicurazione. Associando esplicitamente "id" e "for" di ciascuna coppia, cliccando sulla <label> verrà selezionato il radio-button di <input> corrispondente.
- [HTML] Per visualizzare i tasti di submit e reset alla fine del form, scrivo <input type="submit"/> ed <input type="reset"/>. In base alla lingua in cui è impostato il mio browser verranno mostrati messaggi

differenti sui bottoni (e.g. in italiano vedrò "Invia" e "Reimposta"). Se volessi personalizzare l'etichetta mi basterebbe aggiungere in ciascun tag l'attributo "value" ed assegnarli la stringa che preferisco. I bottoni non necessitano di una <label>, dunque ignorare AChecker nel caso dovesse sostenere il contrario.

- [HTML] Se in una <select> volessi inserire un'opzione di default vuota allora dovrei aggiungere un tag <option> senza testo, ma specificando l'attributo "label="vuoto"". Questo perché in generale, quando faccio la submission di una form, vengono create delle coppie nome-valore tra il name dell'elemento di input ed il valore che viene effettivamente inserito testualmente (o selezionato tramite checkbox o radio-button). Se avessi un valore vuoto in uno degli <option>, la coppia nome-valore non si formerebbe correttamente, quindi si utilizza l'attributo "label" (non il tag!) cosicché al file di action lato server venga inviata la coppia {*id_select*, *label*}.
- [CSS] Per visualizzare un gradiente sullo sfondo che varia in diagonale dall'angolo in alto a sinistra a quello in basso a destra dai colori rosa, giallo, rosa e viola, "background-image: linear-gradient(to bottom right, pink, yellow, pink, purple);"
- [CSS] Se dovessi avere delle specifiche riguardo ad esempio l'allineamento ed il colore del testo nell'header, dovrò specificarle all'interno di "header{}" nel file CSS. Il testo all'interno del suddetto contenitore sarà invece specificato nel file HTML.
- [CSS] Per far sì che gli elementi <nav>, <aside> e <section> siano affiancati uno accanto all'altro (considerato che nell'HTML sono nell'ordine <section>, <nav> e <aside>) dovrò specificare all'interno dei tre blocchi di codice corrispondenti rispettivamente "float: left;", "float: left;" e "float: right". Prima di impostare i margini per gli elementi di blocco da affiancare potrebbe verificarsi il fatto che il <footer> venga visualizzato tra <aside> e <section>, poiché le percentuali di width non sommano al 100% della finestra del browser. Per ovviare a ciò dobbiamo specificare nel <footer> la proprietà "clear: both;" per "pulire i floating che potrebbero fluttuare" dicendo - guarda che questo elemento di blocco me lo togli da quello che è il flusso dei float che potrebbe andare a condizionarlo.
- [CSS] Se mi venisse chiesto di modificare l'interlinea di un testo e di portarlo al doppio del valore di default dovrò specificare la proprietà "line-height: 2em;", essendo "1em" l'altezza di default, oppure "line-height: 2.0;" o ancora "line-height: 200%;".

- [CSS] Per visualizzare un quadratino come simbolo per una scrivo "list-style-type: square;".
- [CSS] In un ottica di utilizzo di dimensioni relative e non assolute la versione più corretta è "font-weight:bolder" poiché significa "con più peso rispetto a quello di default", mentre "font-weight:bold" indica "in grassetto" in generale.
- [CSS] First we need to understand the difference between specified weight, computed weight and rendered weight.
- For **bold**, the specified weight is "bold", the computed weight is "700" and the rendered weight depends on the font, and the only guarantee is that it won't be lighter than elements with lower computed weights. (i.e. Since "normal" equates to "400", "bold" will never be rendered lighter than "normal" is (though it could be rendered identically.)
- For **bolder**, the specified weight is "bolder" and the computed weight is "400" if the container element has a computed weight of less than or equal to 300, otherwise "700" if the container element has a computed weight of less than or equal to 500, otherwise "900". The rendered weight again depends on the font with the same guarantee. Since typefaces (design of letters, numbers and other symbols) typically only natively support normal and bold weights, this often means that "font-weight:bold" and "font-weight:bolder" get rendered identically even if they have different computed weights. But it doesn't *have* to be the case, even if the font only supports those two weights. In particular, if the container element has a computed weight less than "400", then "bolder" will compute to a weight of "400" and render the same as an element with a specified weight of "normal".
- [CSS] Values of 'bolder' and 'lighter' indicate values relative to the weight of the parent element. Based on the inherited weight value, the weight used is calculated using the chart below. Child elements inherit the calculated weight, not a value of 'bolder' or 'lighter'.

The meaning of 'bolder' and 'lighter'

Inherited value bolder lighter

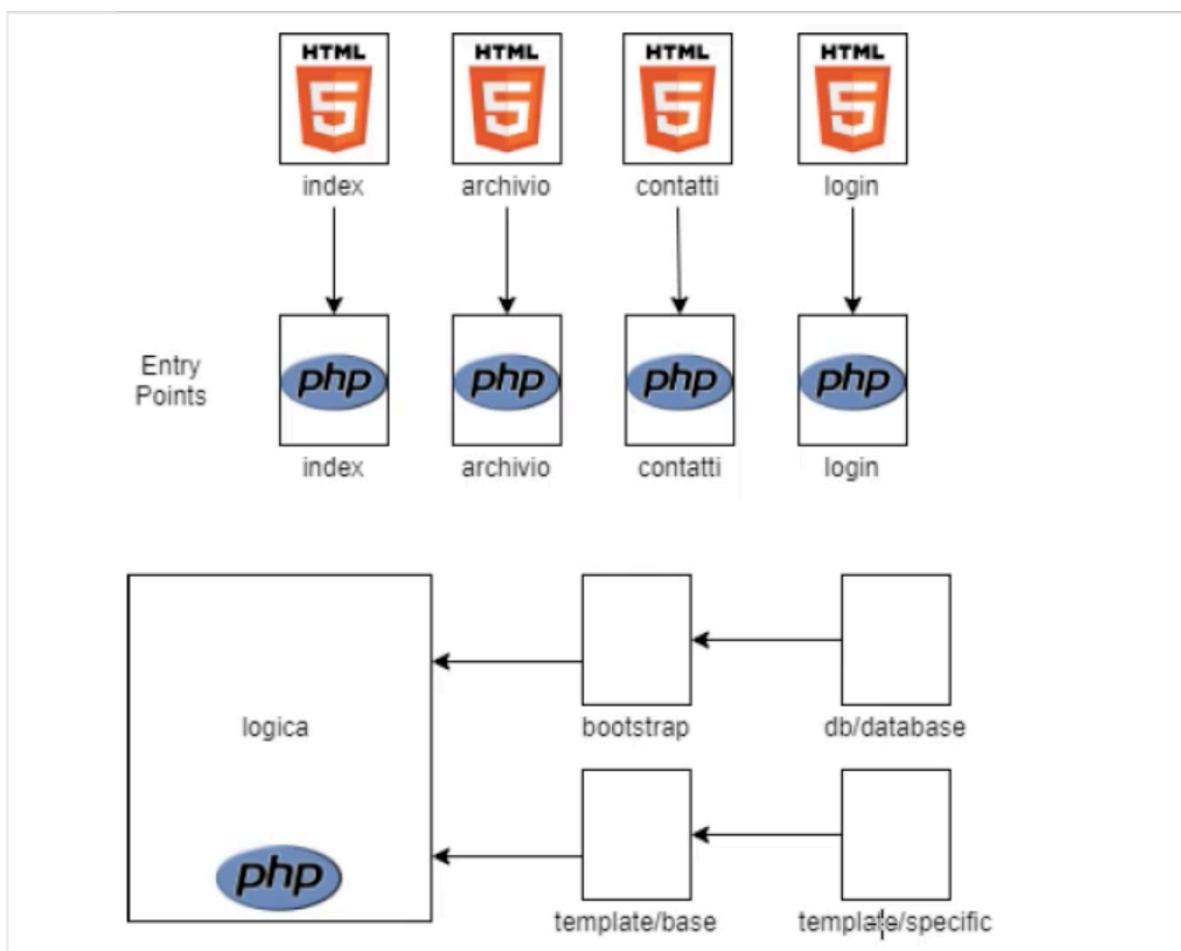
100	400	100
200	400	100
300	400	100
400	700	100
500	700	100
600	900	400
700	900	400
800	900	700
900	900	700

- [CSS] Margin e padding, soprattutto quando sono piccoli, si specificano in pixel, che garantiscono una maggiore precisione, mentre per le width e le dimensioni dei font è fortemente consigliato avere misure relative, dunque risultano più consone le percentuali.

LAB 14-11-24

- [PHP] Linguaggio di programmazione (di scripting) vero e proprio, server-side, utilizzato per realizzare la parte backend di un sito. Siccome è un linguaggio lato server avremo bisogno di un server, che otteniamo con **XAMPP**. Iniziamo a trasformare il nostro sito da semplici pagine HTML statiche a pagine dinamiche in cui recuperiamo i dati da un database e poi l'output delle nostra pagina PHP sarà la pagina HTML. Evita la duplicazione del codice e da la possibilità ad un utente di modificare in autonomia i dati. Vogliamo mappare ogni pagina HTML che abbiamo fatto in una corrispondente pagina PHP (index.html -> index.php, archivio.html -> archivio.php, etc.), cosa cambia? Mentre prima avevamo tutto il codice in tutte le pagine, vogliamo organizzarlo diversamente: ogni pagina conterrà un file che chiameremo "**bootstrap**", che racchiude le operazioni di inizializzazione comuni a tutte le pagine (quando dobbiamo fare qualcosa comune a tutte le pagine lo mettiamo qui), e che a sua volta includerà il file "**db/database**", il quale ospiterà tutte le operazioni per interagire con il db.

- [PHP] All'interno di ogni pagina PHP quindi includeremo "bootstrap" (bootstrapping in operating systems refers to the process of loading and initializing the operating system on a computer) come prima cosa, avremo poi una logica nella pagina per, ad esempio, recuperare i dati (in base alla pagina che stiamo visitando, recupereremo dati diversi) e prepareremo questi dati alla visualizzazione in due template: template/base e template/specific che conterranno il codice HTML. In particolare, il primo conterrà le parti di HTML comuni a tutte le pagine (e.g. <header>, <footer>, etc.), il secondo ci serve, in questo caso, per riempire il main, il cui contenuto sarà parametrizzato ed in base alla pagina in cui ci troviamo includeremo nel main una parte diversa da visualizzare.



Il quadrato grande indica **ogni** pagina php!

- [PHP] Apache gira sulla porta 80, HTTPS in teoria sulla 443 ma se c'è già un servizio sul pc in esecuzione su quella porta viene cambiata (ad esempio in 444). MySQL invece gira sulla porta 3306 che è quella di default. Come facciamo ad interagire con il DB? Utilizzando phpMyAdmin, quindi se clicchiamo sul bottone "Go to application" e poi "phpMyAdmin" ci dovrebbe aprire il percorso "**localhost/phpmyadmin/**". Se abbiamo cambiato la porta di default di Apache e non abbiamo più la 80 ma ad esempio la 8080 dobbiamo cambiare il

percorso in "localhost:8080/phpmyadmin/".

The screenshot shows the phpMyAdmin configuration page. On the left is a sidebar with 'Recent' and 'Favourites' sections, and a 'Console' button at the bottom. The main area has four tabs: 'General settings', 'Appearance settings', 'Database server', and 'phpMyAdmin'. The 'General settings' tab shows 'Server connection collation: utf8mb4_unicode_ci'. The 'Appearance settings' tab shows 'Language: English (United Kingdom)' and 'Theme: pmahomme'. The 'Database server' tab lists MySQL server details: Server: Localhost via UNIX socket, Server type: MariaDB, Server connection: SSL is not being used, Server version: 10.4.28-MariaDB - Source distribution, Protocol version: 10, User: root@localhost, and Server charset: UTF-8 Unicode (utf8mb4). The 'Web server' tab lists Apache details: Apache/2.4.56 (Unix) OpenSSL/1.1.1t PHP/8.2.4 mod_perl/2.0.12 Perl/v5.34.1, Database client version: libmysql - mysqld 8.2.4, PHP extension: mysqli curl mbstring, and PHP version: 8.2.4. The 'phpMyAdmin' tab lists version information: Version information: 5.2.1 (up to date), Documentation, Official Homepage, Contribute, Get support, List of changes, and Licence.

Sulla sinistra abbiamo una lista dei DB presenti di default del sistema. Abbiamo la possibilità di creare un DB, di eseguire una query, di fare export/import etc. MOMENTO CRITICO che porta solitamente ad errori => Siccome dobbiamo lavorare all'interno di un web server, Apache funziona nel seguente modo: mappa una sua sottocartella (htdocs) come root di "localhost", tagliamo quindi la cartella "03-lab-php", e la portiamo all'interno di "**xampp/htdocs**". Ora, se volessimo vedere il contenuto della cartella "03-lab-php" non dovremmo digitare nel nostro URL "localhost/htdocs/sito/index.html": "htdocs" non serve poiché il suo contenuto è la root del sito, quello che ci servirà invece per visualizzare "03-lab-php" sarà "localhost/03-lab-php/". Questo è un comportamento che abbiamo solo perché stiamo lavorando in locale: di default se in un web-server scrivessimo il contenuto di una cartella che non abbia un file index non ci verrebbe mostrato niente (per ragioni di sicurezza non viene esibito il contenuto di una cartella di un web-server remoto).

The screenshot shows a web browser window. The address bar says "localhost/03-lab-php/". The main content area displays a file listing for the directory "/03-lab-php". The table has columns: Name, Last modified, Size, and Description. The rows are: Parent Directory (with a back arrow icon), db/ (modified 2024-11-13 17:15, size -), and sito/ (modified 2024-10-03 09:40, size -). To the right of the browser is a sidebar titled "Favourites" containing icons for GitHub, Corso, Reverso, and Translate.

Name	Last modified	Size	Description
Parent Directory		-	
db/	2024-11-13 17:15	-	
sito/	2024-10-03 09:40	-	

In HTML recuperavamo i file con "**file:///**", e questo era il nostro modo di accedere tramite il file-system alle pagine web. Adesso invece non accediamo

più con il file-system ma tramite **HTTP**, cosa che ci tornerà utile poiché lavoriamo con un interprete PHP ed anche per la possibilità di effettuare chiamate asincrone al server. Copiando ed incollando il percorso precedentemente scritto ci accorgiamo di come sia stata aggiunta la parte iniziale relativa ad HTTP <http://localhost/03-lab-php/>, ossia il protocollo di comunicazione con cui vengono ottenuti i file.

- [PHP] Con il termine **Apache** si fa riferimento ad Apache HTTP Server, chiamato anche httpd. Lanciato nel 1995, Apache HTTP Server è un web server open-source che consente agli utenti di ospitare e mantenere i propri siti su internet. Fa parte degli oltre 350 progetti gestiti dalla Apache Software Foundation. Apache viene ampiamente utilizzato perché offre alte prestazioni, stabilità e sicurezza. Supporta una vasta gamma di protocolli come HTTP, HTTPS e FTP, ed è in grado di gestire più richieste simultanee, rendendola una scelta popolare per siti web grandi e complessi. Supporta tutti i principali sistemi operativi, tra cui Windows, Linux e MacOS, oltre a varie varie tecnologie web, come PHP, Perl e Python. Inoltre, Apache è altamente flessibile e personalizzabile, con una vasta gamma di moduli disponibili che possono essere aggiunti per fornire funzionalità supplementari. Apache è ampiamente utilizzato nell'hosting e nel servizio di applicazioni basate sul web ed è considerato uno dei software più popolari per i siti web di hosting.
- [PHP] Ora su VSCode apriamo "xampp/**htdocs**/03-lab-php" ed ogni modifica che apporteremo al codice sarà già sul web-server. Nella cartella "db" abbiamo due file SQL, il primo per creare lo schema mentre il secondo per inserire dei dati. Incolliamo il contenuto di "creazione_db.sql" sul tab "SQL" della finestra *phpmyadmin* e clicchiamo su "Go". Aggiornando poi la lista dei DB sulla sinistra dovremmo vedere apparire la nostra base di dati, ci clicchiamo sopra e poi nuovamente sul tab "SQL" (questa volta relativo a "blogtw") inseriamo il codice di "inserisci_dati.sql". Siamo ora pronti per interagire con il DB e recuperare dati tramite interrogazioni.
- [PHP] Se volessi visualizzare l'output "Ciao!" generato dall'istruzione "echo "Ciao!"" scritta nel file index.php, che percorso dovrei visitare? "localhost/03-lab-php/index.php". Omettendo "**index.php**" otterrei lo stesso risultato poiché il file index è quello che viene visualizzato (quindi richiesto al server) di default se noi domandiamo la root di una cartella, ossia "localhost/03-lab-php/", ed è il motivo per cui solitamente tutti i file di homepage vengono chiamati index.*
- [PHP] Le istruzioni di PHP per includere file esterni sono "require" ed "include" che differiscono per la gestione dell'errore: se il file richiesto

attraverso "require" non venisse trovato verrebbe lanciato un fatal-error e l'esecuzione verrebbe interrotta, mentre "include" ci darebbe soltanto un warning. Includiamo il file "**bootstrap.php**" (che non ha nulla a che vedere con l'omonimo framework CSS) in "index.php" con l'istruzione "require_once("bootstrap.php");". Tutte le variabili, funzioni, classi etc. dichiarate in bootstrap.php saranno visibili nello scope di index.php.

- Creiamo il file "**database.php**" all'interno della cartella "db", che fungerà da helper per interagire con il database. In particolare creiamo una classe un cui metodo verrà chiamato ogni qualvolta risulti necessario interagire con il db. Il database, sempre sul server locale, è un servizio in ascolto sulla porta 3306, quindi sarà imprescindibile aprire una connessione su di essa quando dobbiamo interrogarlo, cosa che faremo nel costruttore, richiedendo i parametri "\$servername", "\$username", "\$password", "\$dbname", "\$port" e creando un'istanza tramite l'API di MySQLi, assegnata poi all'unica proprietà "\$db". Per controllare che la connessione sia andata a buon fine verifichiamo il campo-flag connect_error di "\$db", interrompendo se necessario lo script con "die("messaggio errore")".
- [PHP] Dove istanziamo l'oggetto della classe "**DatabaseHelper**"? In "bootstrap.php" (dove abbiamo detto di inserire tutte le inizializzazioni comuni a tutte le pagine), così da farlo (previa l'inclusione di "db/database.php") una volta sola: se lo avessimo dovuto fare in ogni pagina l'avremmo fatto n volte, mentre scrivere la classe ed istanziarne un oggetto nello stesso punto (in "database.php") è fortemente sconsigliato.
- [PHP] Creiamo ora la cartella "**template**" al cui interno inseriamo il file "base.php", dove incolliamo il contenuto di "index.html". In quest'ultimo file vado poi a specificare "require("template/base.php");", ed apprendo sul browser "<http://localhost/03-lab-php/>" vedrò la pagina HTML non formattata poiché abbiamo rotto i link: il file "base.php" si aspetta il foglio di stile nella cartella "css" ("href=".css/style.css") ma noi lo abbiamo attualmente dentro a "sito", quindi spostiamo l'intera cartella "css" come figlia di "<http://localhost/03-lab-php/>".
- [PHP] Ora vogliamo cambiare tutto il codice in "base.php" che varia da una pagina all'altra. Il primo valore che dovrà cambiare da pagina a pagina è <title>: "base.php" verrà incluso da tutte le pagine, come facciamo a fare in modo che vari da una pagina all'altra? Con una variabile che definiamo in "logica", caratteristica della singola pagina, e non in "template/specific" che è incluso da "template/base", a sua volta incluso in ciascuna pagina. Nella logica di ogni pagina quindi noi

creiamo (ci sarà una variabile array diversa per pagina) un **array associativo** (da dichiarare prima dell'inclusione require("template/base.php")) che chiamiamo "\$templateParams" dove mettiamo tutti i valori che poi andranno visualizzati all'interno del template. Tornando al titolo, in "base.php" inseriremo "<title><?php echo \$templateParams ["titolo"];></title>" ed, ad esempio, in "index.php" avremo "\$templateParams ["titolo"] = "Blog TW - Home";".

- [PHP] Continuando a scorrere il "template/base", tra le altre cose da cambiare troveremo le estensioni dei file, che da ".html" diventeranno ".php".
- [PHP] Vogliamo ora gestire i post casuali della prima <section> nell'<aside>, che dovranno essere letti dal database. La prima macro-operazione consisterà nello scrivere le **interrogazioni**, dunque aggiungere un metodo al nostro punto di contatto con il database, ossia la classe "databaseHelper" all'interno di "database.php" per recuperare i dati (post in questo caso). Specifichiamo come parametro della funzione il numero di post da visualizzare, impostando un valore di default nel caso non venisse specificato dall'utente "\$n=2". Usiamo i prepared statements per interagire con il db onde evitare problematiche quali SQL injections etc, quindi prima prepareremo statement scrivendo la query rimpiazzando i parametri con il placeholder "?", poi faremo il binding dei parametri ed infine eseguiremo l'interrogazione. Per riuscire a stampare gli articoli in Post Casuali ci serviranno titolo, id (poiché è un link/collegamento) e l'immagine di ciascuno. Il 1° parametro di "bind_param()" è una stringa indicante il tipo delle variabili da associare. La penultima istruzione indica il fetch dei risultati, restituiti sotto forma di puntatore alle righe estratte dal db. Infine ritorniamo i dati come array associativo.

```
public function getRandomPosts($n=2){
    $stmt = $this->db->prepare("SELECT idarticolo, titoloarticolo, imgarticolo FROM articolo ORDER BY RAND() LIMIT ?");
    $stmt->bind_param("i", $n);
    $stmt->execute();
    $result = $stmt->get_result();

    return $result->fetch_all(MYSQLI_ASSOC);
}
```

- [PHP] **Prepared statements** are a feature of the programming language used to communicate with the database. For example, C#, Java, and PHP provide abstractions for sending statements to a database. These abstractions can either be literal queries created via **string concatenation** of variables (bad!) or prepared statements. This should also highlight the point that database insecurity is not an

artifact of the database or the programming language but how the code is written. Prepared statements create a template for a query that establishes an immutable grammar. We'll ignore for a moment the implementation details of different languages and focus on how the concept of prepared statements protects the application from [SQL](#) injection. For example, the following psuedo-code sets up a prepared statement for a simple SELECT that matches a name to an e-mail address.

```
statement = db.prepare("SELECT name FROM users WHERE email = ?")
statement.bind(1, "mutant@mars.planet")
```

In the previous example, the question mark was used as a placeholder for the dynamic portion of the query. The code establishes a statement to extract the value of the name column from the users' table based on a single restriction in the WHERE clause. The bind command applies the user-supplied data to the value used in the expression within the WHERE clause. Regardless of the content of the data, the expression will always be email=something. This holds true even when the data contain SQL commands such as the following examples. In every case, the query's grammar is unchanged by the input, and the SELECT statement will return records only where the e-mail column exactly matches the value of the bound parameter.

```
statement = db.prepare("SELECT name FROM users WHERE email = ?")
statement.bind(1, "*")
statement = db.prepare("SELECT name FROM users WHERE email = ?")
statement.bind(1, "1 OR TRUE UNION SELECT name,password FROM users")
statement = db.prepare("SELECT name FROM users WHERE email = ?")
statement.bind(1, "FALSE; DROP TABLE users")
```

Many languages provide type-specific binding (legante, bind -> lega) functions for data such as strings or integers. These functions help sanity check the data received from the user.

Use prepared statements for any query that includes tainted (contaminato) data. Data should always be considered tainted when collected from the Web browser whether explicitly (such as asking for an e-mail address or credit-card number) or implicitly (such as reading values from [hidden form fields](#) or browser headers). In terms of modifying the sense of an SQL query, prepared statements will not be affected by alternate character sets or encoding techniques found in attacks such as XSS. This doesn't mean that the result set of a query can't be affected. Wildcards, in particular, can still affect the amount of results from a query even if the sense of the query can't be changed. Special characters like the asterisk (*), percent symbol (%), underscore (_), and question mark (?) can be inserted into a bound parameter with [undesirable effect](#). Consider the following code that changes the e-mail comparison from an equality test (=) as in the previous examples to a LIKE statement that would support wildcard matches. As you can see from the bound parameter, this query would return every name in the users' table whose e-mail address

contains the at symbol, (@).

```
statement = db.prepare("SELECT name FROM users WHERE email LIKE ?")  
statement.bind(1, "%@%")
```

- [PHP] La seconda macro-operazione consiste nel chiamare il metodo sopra-definito ed **assegnarne il valore** di ritorno a qualche campo di \$templateParams. Questo avverrà all'interno di "index.php", poiché i Post Casuali non sono altro che un parametro del template:
"templateParams["articolicasuali"] = \$dbh->getRandomPosts(2);"
- [PHP] Come terza ed ultima macro-operazione dovremo gestire la logica di **visualizzazione** nel **template**, in questo caso rimpiazzando in "template/base.php" il contenuto dei <listItem> con gli articoli casuali, generandoli inserendo i dati che abbiamo letto dal db. Come facciamo a visualizzarne n, come i posti casuali che abbiamo letto, piuttosto che limitarci a 2? Facendo un ciclo (sfruttando la sintassi ":"- "end" invece delle graffe, confusionarie all'intero di codice HTML):
- [PHP] Ripetiamo ora i 3 step per visualizzare le categorie della seconda <section> nell'<aside>. Nella fase di interrogazione utilizziamo sempre un prepared statement pur non avendo parametri, nell'eventualità in cui dovessimo aggiungerli.
- [PHP] Ripetiamo nuovamente i 3 step questa volta per visualizzare i post più recenti. Come parametro della funzione "getPosts(\$n=-1){}" in "database.php" utilizziamo un valore negativo cosicché nel caso non venisse specificato diversamente li leggeremo tutti. Salviamo inizialmente la query in una variabile d'appoggio poiché dovremo fare un controllo sul valore del parametro \$n: se dovesse essere > 0, andremo a concatenare all'interrogazione " LIMIT ?" (ricordandoci lo spazio iniziale, altrimenti la stringa risulterebbe attaccata alla prima parte di interrogazione) per poi fare il binding della variabile (sempre qualora \$n fosse > 0).
- [PHP] Se adesso noi costruendo la pagina "Contatti" includessimo "template/base.php" il <main> da cosa sarebbe popolato? Dagli articoli recenti che abbiamo inserito durante la realizzazione della pagina iniziale "Home", il che è scorretto e dovremmo pertanto trovare il modo di parametrizzare il contenuto di <main>, come facciamo? Il **template specifico** non è altro che un template adatto alla visualizzazione di una singola pagina: creiamo il file "template/lista-articoli.php" dove spostiamo il contenuto del <main> di "template/base.php" e per dire ad "index.php" che in <main> deve visualizzare

"template/lista-articoli" aggiungiamo una nuova chiave a \$templateParams, ossia "nome", che mappi il nome del template specifico che dobbiamo visualizzare. Infine in "template/base.php" all'interno di <main> includiamo il file "\$templateParams["nome"]" il cui nome, quindi contenuto, varierà da una pagina all'altra.

```
index.php
1  <?php
2  require_once("bootstrap.php");
3
4  $templateParams ["titolo"] = "Blog TW - Home";
5  $templateParams ["nome"] = "lista-articoli.php";
6  $templateParams ["articolicasuali"] = $dbh->getRandomPosts(2);
7  $templateParams ["categorie"] = $dbh->getCategories();
8  $templateParams ["articoli"] = $dbh->getPosts(2);
9
10 require("template/base.php");
11
12 ?>
```

LAB 15-11-24

- [PHP] Quali sono i passi da fare per creare una nuova pagina, dato il codice HTML? Vogliamo mappare ogni file ".html" con un ".php" quindi per prima cosa andrà creato il corrispettivo ".php" della pagina che si desidera implementare. Dovremo in primis includere il file "bootstrap.php" dove abbiamo istanziato un oggetto della classe DatabaseHelper, previa l'implementazione di metodi per recuperare i dati dal db (**1°step**), riempire opportunamente i campi dell'array associativo "\$templateParams[]" (tra cui il riferimento al template specifico della pagina che verrà espanso nel <main> di "template/base.php"), sfruttando i valori di ritorno dei metodi appena definiti (**2°step**), ed infine importare il file "template/base.php" contenente la struttura comune ad ogni pagina con le parte variabili parametrizzate., dopo aver gestito la visualizzazione dei dati nel template specifico (**3°step**).
- [PHP] Abbiamo bisogno di dati per visualizzare il form di login? No, quindi salteremo i primi due step e dovremo concentrarci unicamente sul template specifico. Copiamo ed incolliamo il codice di "login.html" contenuto all'interno del tag <main> in "template/login-form.php". Attenzione: questa volta non abbiamo bisogno di aprire-chiudere PHP

poiché non dobbiamo eseguire istruzioni particolari. In "template/base.php" dobbiamo cambiare l'estensione del link alla pagina Login all'interno del <nav> da ".html" a ".php".

- [PHP] <>Fare in modo che venga aggiunta la classe active al link della pagina attualmente visitata >>. Nel file "style.css" troviamo la classe "active" associata all'elemento "a" che semplicemente colora il testo del link. L'idea è che se io stessi visitando la pagina di Login vorrei che la scritta "Login" all'interno del bottone sia rossa, per veicolare all'utente l'informazione su quale pagina stia visitando in quel momento. Per ogni link all'interno del <nav> controlliamo se corrisponde o meno alla pagina attuale. Aggiungiamo un nuovo file "utils/functions.php" che conterrà tutte le funzioni che definiremo in modo che siano accessibili a tutto il resto del codice

```
php > utils > 🐱 functions.php
1  <?php
2  function isActive($pageName) {
3      if(basename($_SERVER["PHP_SELF"]) == $pageName) {
4          echo " class = 'active' ";
5      }
6  }
7  ?>
```

Il parametro rappresenta la voce che vogliamo comparare alla pagina attuale, esplicitata nell'URL, che ricaviamo da una delle variabili super-globali di PHP: "\$_SERVER" che possiede una chiave "PHP_SELF" indicante la pagina attuale. "\$_SERVER["PHP_SELF"]" contiene tuttavia l'intero URL mentre noi vogliamo semplicemente il nome finale dello script che stiamo visitando, come facciamo ad ottenerlo? Tramite una funzione pre-definita di PHP che estrae il singolo nome della pagina: "basename()". Come utilizziamo questa funzione in "template/base.php"?

```
<nav>
  <ul>
    <li><a <?php isActive("index.php") ?> href="index.php">Home</a></li><li><a <?php isActive("archivio.html") ?> href="archivio.html">Archivio</a></li><li><a <?php isActive("contatti.html") ?> href="contatti.html">Contatti</a></li><li><a <?php isActive("login.php") ?> href="login.php">Login</a></li>
  </ul>
</nav>
```

Dobbiamo ora rendere visibile la funzione "isActive()", ma come? Sarebbe scorretto includere il file "utils/functions.php" in "template/base.php" perché magari io definisco delle funzioni che vorrò poi utilizzare anche in altri parti del codice: il punto dove invece l'import verrebbe condiviso da tutti sarebbe in "bootstrap.php".

- [PHP]



Esercizio 02

- Dati i file nella cartella php, aggiungere la pagina contatti.php (vecchia contatti.html) nella quale:
 - Verranno visualizzati tutti gli autori del blog.
- Linee guida:
 - Chiamare il template contatti
 - Chiamare il metodo per recuperare i dati degli autori getAuthors



Esercizio 02 - Hint

- Query SQL per recuperare autori:
 - SELECT username, nome, GROUP_CONCAT(DISTINCT nomecategoria) as argomenti FROM categoria, articolo, autore, articolo_ha_categoria WHERE idarticolo=articolo AND categoria=idcategoria AND autore=idautore AND attivo=1 GROUP BY username, nome
- Funzione per creare gli ID:
 - ```
function getIdFromName($name){
 return preg_replace("/[^a-z]/", "", strtolower($name));
}
```

Recuperiamo gli autori tramite "getAuthors()", contenente la query messa a disposizione nella consegna.

```

contatti.php
1 <?php
2 require_once 'bootstrap.php';
3
4 //Base Template
5 $templateParams["titolo"] = "Blog TW - Contatti";
6 $templateParams["nome"] = "contatti.php";
7 $templateParams["categorie"] = $dbh->getCategories();
8 $templateParams["articolicasuali"] = $dbh->getRandomPosts(2);
9 //Home Template
10 $templateParams["autori"] = $dbh->getAuthors();
11
12 require 'template/base.php';
13 ?>

```

```

db > database.php
73 public function getAuthors(){
74 $query = "SELECT username, nome, GROUP_CONCAT(DISTINCT nomecategoria) as
75 argomenti FROM categoria, articolo, autore, articolo_ha_categoria WHERE
76 idarticolo=articolo AND categoria=idcategoria AND autore=idautore AND attivo=1
77 GROUP BY username, nome";
78 $stmt = $this->db->prepare($query);
79 $stmt->execute();
80 $result = $stmt->get_result();
81
82 return $result->fetch_all(MYSQLI_ASSOC);
83 }

```

Nella tabella l'unica parte complicata era generare gli id (da attribuire in primis ai table headers `<th></th>` relativi a ciascun autore, poi a ciascuna cella `<td></td>`) a partire dal nome, utilizzando la funzione fornita.

```

contatti.php
1 <section>
2 <h2>Autori del Blog</h2>
3 <table>
4 <tr>
5 <th id="autore">Autore</th><th id="email">Email</th><th
6 id="argomenti">Argomenti</th>
7 </tr>
8 <?php foreach($templateParams["autori"] as $autore): ?>
9 <tr>
10 <th id="<?php echo getIdFromName($autore["nome"]); ?>"><?php echo
11 $autore["nome"]; ?></th><td headers="email <?php echo getIdFromName
12 ($autore["nome"]); ?>"><?php echo $autore["username"]; ?></td><td
13 headers="argomenti <?php echo getIdFromName($autore["nome"]); ?
14 >"><?php echo $autore["argomenti"]; ?></td>
15 </tr>
16 <?php endforeach; ?>
17 </table>
18 </section>

```

- [PHP]



## Esercizio 03

- Dati i file nella cartella php, aggiungere la pagina archivio.php (vecchia archivio.html) nella quale:
  - Verranno visualizzati tutti gli articoli ordinati per data in maniera decrescente.

Nella pagina di Archivio potevamo sfruttare il fatto di disporre già un template specifico per una lista di articoli, ossia quello di Home, che poteva quindi rimanere il medesimo, ricordandoci però di cambiare la stringa relativa al titolo "\$templateParams["titolo"]" e di omettere l'argomento "2" nella chiamata alla funzione "\$dbh->getPosts(2)" poiché siamo interessati a recuperare tutti gli articoli piuttosto che i due più recenti.

```
🐘 archivio.php
1 <?php
2 require_once 'bootstrap.php';
3
4 //Base Template
5 $templateParams["titolo"] = "Blog TW - Archivio";
6 $templateParams["nome"] = "lista-articoli.php";
7 $templateParams["categorie"] = $dbh->getCategories();
8 $templateParams["articolicasuali"] = $dbh->getRandomPosts(2);
9 //Home Template
10 $templateParams["articoli"] = $dbh->getPosts();
11
12 require 'template/base.php';
13 ?>
```

- [PHP]



## Esercizio 04

- Dati i file nella cartella php, aggiungere la pagina articolo.php nella quale:
  - Verrà visualizzato l'articolo il cui id è stato passato in Get.
- Tutti i link relativi ad un singolo articolo dovranno puntare a questa pagina.
- Linee guida:
  - Chiamare il metodo per recuperare i dati dell'articolo `getPostById`

```
❶ articolo.php
1 <?php
2 require_once 'bootstrap.php';
3
4 //Base Template
5 $templateParams["titolo"] = "Blog TW - Articolo";
6 $templateParams["nome"] = "singolo-articolo.php";
7 $templateParams["categorie"] = $dbh->getCategories();
8 $templateParams["articolicasuali"] = $dbh->getRandomPosts(2);
9 //Home Template
10 $idarticolo = -1;
11 if(isset($_GET["id"])){
12 $idarticolo = $_GET["id"];
13 }
14 $templateParams["articolo"] = $dbh->getPostById($idarticolo);
15
16 require 'template/base.php';
17 ?>
```

In ciascun bottone “Leggi Tutto” all’interno del file “lista-articoli.php”, era necessario specificare l’id dell’articolo nella query-string dell’URL di destinazione, ottenuto accedendo all’array associativo, nonché variabile di ciclo, “\$articolo[“idarticolo”]”. Una volta reindirizzati sulla pagina “articolo.php” come recuperiamo l’id del post da mostrare? Essendo che di default tutte le richieste fatte partire dal browser cliccando sui link sono di tipo GET, il parametro lo si reperisce dall’array associativo di variabili “\$\_GET”. In particolare, indicizzando il vettore “\$\_GET[“id”]” recuperiamo dal database la struttura dati relativa all’articolo in questione e la assegniamo ad “\$templateParams[“articolo”]”. Dobbiamo però ricordarci di effettuare il controllo sul fatto che l’id sia o meno settato, potendo il secondo scenario verificarsi qualora l’utente richiedesse manualmente (piuttosto che tramite click su <a>), digitando nella barra di ricerca del browser, la risorsa omettendo nell’indirizzo la query-string.

```

template > 🐛 lista-articoli.php
1 <?php if(isset($templateParams["titolo_pagina"])): ?>
2 <h2><?php echo $templateParams["titolo_pagina"]; ?></h2>
3 <?php endif;?>
4 <?php foreach($templateParams["articoli"] as $articolo): ?>
5 <article>
6 <header>
7 <div>
8 " alt="" />
9 </div>
10 <h2><?php echo $articolo["titoloarticolo"]; ?></h2>
11 <p><?php echo $articolo["dataarticolo"]; ?> - <?php echo $articolo["nome"]; ?></p>
12 </header>
13 <section>
14 <p><?php echo $articolo["anteprimaarticolo"]; ?></p>
15 </section>
16 <footer>
17 <a href="articolo.php?id=<?php echo $articolo["idarticolo"]; ?>">Leggi tutto
18 </footer>
19 </article>
20 <?php endforeach; ?>

```

Il template del singolo articolo "singolo-articolo.php" era analogo a quello della lista degli articoli "lista-articoli.php" eccetto per la mancanza del for-each nel primo. Era necessario aggiungere un controllo sul fatto di aver effettivamente memorizzato un articolo in "\$templateParams["articolo"]", poiché nel caso in cui l'utente avesse inserito a mano un id non esistente, o non l'avesse inserito proprio (facendo quindi eseguire "\$dbh->getPostById(-1)") l'array avrebbe 0 elementi. Se "\$templateParams["articolo"]" non fosse vuoto allora avrebbe esattamente un elemento, ed essendo "\$templateParams" un array di array associativi dovremmo accedere all'elemento in posizione 0 del vettore "articolo", e non limitarci alla chiave poiché rappresenterebbe un errore. Infine la stampa era pressoché identica eccetto per il fatto che mancasse il bottone "Leggi Tutto" e l'anteprima dell'articolo fosse sostituita dal testo intero dell'articolo.

```

template > 🐛 singolo-articolo.php
1 <?php if(count($templateParams["articolo"])==0): ?>
2 <article>
3 <p>Articolo non presente</p>
4 </article>
5 <?php
6 else:
7 $articolo = $templateParams["articolo"][0];
8 ?>
9 <article>
10 <header>
11 <div>
12 " alt="" />
13 </div>
14 <h2><?php echo $articolo["titoloarticolo"]; ?></h2>
15 <p><?php echo $articolo["dataarticolo"]; ?> - <?php echo $articolo["nome"]; ?></p>
16 </header>
17 <section>
18 <p><?php echo $articolo["testoarticolo"]; ?></p>
19 </section>
20 </article>
21 <?php endif; ?>

```



## Esercizio 05

- Dati i file nella cartella php, aggiungere la pagina articoli-categoria.php nella quale:
  - Verranno visualizzati gli articoli appartenenti ad una categoria passata in Get.
- Tutti i link relativi agli articoli di una singola categoria dovranno puntare a questa pagina (sezione Categoria nell'aside).
- Linee guida:
  - Chiamare il metodo per recuperare gli articoli di una categoria getCategoryById e il metodo per recuperare il nome di una categoria getCategoryByld

Anche qui ciascuna pagina contenente articoli relativi solamente ad una determinata categoria andava linkata in "base.php", dove ogni <a> contenuta nella seconda <section> di <aside> deve puntare alla pagina "articoli-categoria.php" avente come id quello della categoria. A questo punto il file "articoli-categoria.php" che cosa fa? Rimane comunque una lista di articoli, per cui si può riutilizzare il template specifico "lista-articoli.php", pensando eventualmente di aggiungere un titolo "\$templateParams["titolo\_pagina"]" (diverso da "\$templateParams["titolo"]", usato invece come <title>, tag obbligatorio figlio di <head> e visualizzato sulla tab della pagina dal browser) del tipo "Articoli della categoria x". In "articoli-categoria.php" c'è poi un doppio controllo. Il primo viene condotto sull'esistenza o meno della categoria: si recupera l'id della categoria richiesta, si controlla sul DB che la questa esista, ed in caso affermativo si compone il titolo "\$templateParams["titolo\_pagina"]" e si recuperano tutti gli articoli ad essa associati, altrimenti se non dovesse esistere si imposterebbe come "\$templateParams["titolo\_pagina"]" un messaggio di errore e si assegnerebbe un'array vuoto a "\$templateParams["articoli"]" onde evitare che il ciclo di "lista-articoli.php" iteri su elementi sbagliati (così facendo non ciclerà mai). Infine la lista di articoli essenzialmente non cambia poiché ciascun articolo avrà sempre l'anteprima ed il bottone leggi tutto.

```

template > # base.php
1 <!DOCTYPE html>
2 <html lang="it">
3 <head>
4 <meta name="viewport" content="width=device-width, initial-scale=1.0" />
5 <title><?php echo $templateParams["titolo"];></title>
6 <link rel="stylesheet" type="text/css" href="/css/style.css" />
7 </head>
8 <body>
9 <header>
10 <h1>Blog di Tecnologie Web</h1>
11 </header>
12 <nav>
13
14 <?php echo $templateParams["titolo"];><?php echo $templateParams["archivio"];><?php echo $templateParams["login"];><?php echo $templateParams["contatti"];>
15
16 </nav>
17 <main>
18 <?php
19 if(isset($templateParams["nome"])){
20 require($templateParams["nome"]);
21 }
22 ?>
23 </main>
24 <main>
25 <h2>Post Casuali</h2>
26
27 <?php foreach($templateParams["articolicasuali"] as $articolicasuale): ?>
28
29 <img src=<?php echo UPLOAD_DIR.$articolicasuale["imgarticolo"];> alt="" />
30 $articolicasuale["idarticolo"]<?php echo $articolicasuale["titoloarticolo"];>
31
32 <?php endforeach; ?>
33
34 </section>
35 <section>
36 <h2>Categorie</h2>
37
38 <?php foreach($templateParams["categorie"] as $categoria): ?>
39 $categoria["idcategoria"]<?php echo $categoria["nometategoria"];>
40 <?php endforeach; ?>
41
42 </section>
43 </aside>
44 <footer>
45 <p>Tecnologie Web – A.A. 2022/2023</p>
46 </footer>
47 </body>
48 </html>

```

```

🐘 articoli-categoria.php
1 <?php
2 require_once 'bootstrap.php';
3
4 //Base Template
5 $templateParams["titolo"] = "Blog TW – Articoli Categoria";
6 $templateParams["nome"] = "lista-articoli.php";
7 $templateParams["categorie"] = $dbh->getCategories();
8 $templateParams["articolicasuali"] = $dbh->getRandomPosts(2);
9 //Articoli Categoria Template
10 $idcategoria = -1;
11 if(isset($_GET["id"])){
12 $idcategoria = $_GET["id"];
13 }
14 $nometategoria = $dbh->getCategoryById($idcategoria);
15 if(count($nometategoria)>0){
16 $templateParams["titolo_pagina"] = "Articoli della categoria ".$nometategoria[0]
17 ["nometategoria"];
18 $templateParams["articoli"] = $dbh->getPostByCategory($idcategoria);
19 }
20 else{
21 $templateParams["titolo_pagina"] = "Categoria non trovata";
22 $templateParams["articoli"] = array();
23 }
24 require 'template/base.php';
25 ?>

```

```

template > 🐾 lista-articoli.php
1 ... <?php if(isset($templateParams["titolo_pagina"])): ?>
2 ... <h2><?php echo $templateParams["titolo_pagina"]; ?></h2>
3 ... <?php endif;?>
4 <?php foreach($templateParams["articoli"] as $articolo): ?>
5 <article>
6 <header>
7 <div>
8 |
18 </footer>
19 </article>
20 <?php endforeach; ?>

```

- [PHP] Nel file "lista-articoli.php" all'interno del src dell'immagine abbiamo anteposto la costante UPLOAD\_DIR al nome dei .png/.jpg memorizzati in "\$articolo["imgarticolo"]". In PHP le costanti vengono scritte in maiuscolo e si definiscono in "bootstrap.php" tramite la funzione "define()" che accetta due parametri: il primo è il nome che si vuole dare alla costante, mentre il secondo è il valore ad essa associato. Verrà poi fatta una semplice sostituzione: a livello di pre-processing tutte le volte che viene incontrata "UPLOAD\_DIR", questa sarà sostituita con "./upload/". L'idea è quella di avere un unico punto nel codice in cui scriviamo il percorso della cartella contenente le immagini, cosicché qualora questo dovesse variare avremmo un'unica riga da modificare.

```

🐘 bootstrap.php
1 <?php
2 session_start();
3 define("UPLOAD_DIR", "./upload/");
4 require_once("utils/functions.php");
5 require_once("db/database.php");
6 $dbh = new DatabaseHelper("localhost", "root", "", "blogtw");
7 ?>

```

- [PHP] Considerando che tutte le pagine hanno categorie "\$templateParams["categorie"] = \$dbh->getCategories();" ed articoli casuali "\$templateParams["articolicasuali"] = \$dbh->getRandomPosts(2);" non avremmo potuto metterli direttamente in "bootstrap.php" cosicché venissero condivisi a tutte le pagine? Se a livello di sito si dovesse decidere che tutte le pagine debbano avere questi due campi, allora si potrebbe fare e rappresenterebbe

sicuramente un'ottimizzazione.

- [PHP] In informatica una **query-string** o stringa di ricerca è la parte di un URL che contiene dei dati da passare in input ad un programma. L'URL conterrà l'indirizzo di un server, e il percorso nel suo file system per arrivare alla directory nella quale è presente l'eseguibile del programma. Al termine di tale indirizzo, il segno di "?" apre la query-string. Propriamente, la query-string non comprende il segno di "?", ma è tutto ciò che lo segue. La sintassi della querystring non è formalmente definita, si può tuttavia definire standard (perché implementata in tutti i browser e nei linguaggi di scripting) il seguente schema:  
"parametro1=valore1&parametro2=valore2&parametro3=valore3". A ciascun parametro (che può avere nome arbitrario) viene assegnato un valore utilizzando il separatore "=" . I vari parametri (limitati nel numero solo dalla lunghezza della querystring) sono intervallati dal simbolo "&" . La query-string è tipicamente usata per passare al server i dati che l'utente inserisce nei vari spazi bianchi di una web form. Se "campo\_1", "campo\_2", "campo\_3" sono gli spazi della web form da colmare, l'URL generata salvando la form sarà del tipo: "[http://server/percorso/programma?campo\\_1=valore\\_1&campo\\_2=valore\\_2&campo\\_3=valore\\_3](http://server/percorso/programma?campo_1=valore_1&campo_2=valore_2&campo_3=valore_3)". Esempio: "[http://it.wikipedia.org/application/new\\_user/registration\\_form?nome=Mario&cognome=Rossi&ID\\_utente=M\\_Rossi](http://it.wikipedia.org/application/new_user/registration_form?nome=Mario&cognome=Rossi&ID_utente=M_Rossi)". La query string viene generata dal browser ed inviata ad un programma (di solito scritto in JavaServer Pages, PHP, Asp o Perl) che abita sul server.
- [PHP] "**\$\_GET**" contains an associative array of variables received via the HTTP GET method (note that the array is not only populated for GET requests, but rather for all requests with a query string). There are two main ways to send variables via the HTTP GET method: (1) Query strings in the URL (2) HTML Forms. In the PHP file we can use the "**\$\_GET**" variable to collect the value of the query string. A HTML form submits information via the HTTP GET method if the form's method attribute is set to "GET". When a user clicks the submit button, the form data is sent to a PHP file specified in the "action" attribute of the <form> tag. The form fields are sent to the PHP file, with your input, as query strings. In the "action" file we can use the "**\$\_GET**" variable to collect the value of the input fields. "**\$\_GET**" is not stored on the server, but is passed with each HTTP request, and exists only within the latter.

[The first part of the URL is the **schema**, which indicates the protocol that the browser must use to request the resource (a protocol is a set method for exchanging or transferring data around a computer network). Usually for

websites the protocol is HTTPS or HTTP (its unsecured version). Addressing web pages requires one of these two, but browsers also know how to handle other schemes such as mailto: (to open a mail client), so don't be surprised if you see other protocols. The **domain** indicates which Web server is being requested. Usually this is a domain name, but an IP address may also be used (but this is rare as it is much less convenient). The **port** indicates the technical "gate" used to access the resources on the web server. It is usually omitted if the web server uses the standard ports of the HTTP protocol (80 for HTTP and 443 for HTTPS) to grant access to its resources. Otherwise it is mandatory. "/path/to/myfile.html" is the path to the resource on the Web server. In the early days of the Web, a path like this represented a physical file location on the Web server. Nowadays, it is mostly an abstraction handled by Web servers without any physical reality. "?key1=value1&key2=value2" are extra parameters provided to the Web server. Those parameters are a list of key/value pairs separated with the & symbol. The Web server can use those parameters to do extra stuff before returning the resource. Each Web server has its own rules regarding parameters, and the only reliable way to know if a specific Web server is handling parameters is by asking the Web server owner.

"**#SomewhereInTheDocument**" is an anchor to another part of the resource itself. An anchor represents a sort of "bookmark" inside the resource, giving the browser the directions to show the content located at that "bookmarked" spot. On an HTML document, for example, the browser will scroll to the point where the anchor is defined; on a video or audio document, the browser will try to go to the time the anchor represents. It is worth noting that the part after the #, also known as the fragment identifier, is never sent to the server with the request.



A **domain name** (often simply called a domain) is an easy-to-remember name that's associated with a physical IP address on the Internet. It's the unique name that appears after the @ sign in email addresses, and after www. in web addresses. For instance, the domain name example.com might translate to the physical address 198.102.434.8. Other examples of domain names are google.com and wikipedia.org. Using a domain name to identify a location on the Internet rather than the numeric IP address makes it much easier to remember and type web addresses. An **Internet Protocol (IP) address** is a series of numbers that identifies the physical location of a particular device on the Internet network. An IP address looks something like this: 74.125.19.147]

[L'HTTP è un protocollo "stateless" (senza memoria) che permette sia la ricerca che il recupero dell'informazione in maniera veloce, e permette quindi di seguire i rimandi ipertestuali. La scelta di un protocollo "stateless", cioè di un protocollo che non "conserva memoria" della connessione fatta, è stata necessaria affinché fosse possibile saltare velocemente da un server ad un altro attraverso i "links" ipertestuali. HTTP ad ogni richiesta effettua una nuova connessione al server che viene chiusa al termine del trasferimento

dell'oggetto richiesto (pagina HTML, immagine, ecc.). È gestito da un software (server HTTP) residente sugli host che intendono essere fornitori di informazioni. Chi vuole accedere alle informazioni fornite dal server HTTP deve utilizzare un software client (browser) in grado di interpretare le informazioni inviate dal server. Il server, informalmente, è un programma che "gira" in attesa di una richiesta di connessione sul suo socket (la porta assegnatagli, tipicamente la 80). Il protocollo viene utilizzato da un processo daemon (cioè sempre in esecuzione)

Questo protocollo è invocato da TCP/IP ogni qualvolta l'URL (che è una stringa che specifica la risorsa a cui riferirci) istanziata contiene nel primo campo la parola http. I comandi utilizzati per comunicare con esso sono detti metodi. Un server WWW ha il compito (potenzialmente computazionalmente dispendioso) di rispondere a tutte le richieste che giungono dalla rete. Basti pensare che server WWW di siti professionali raggiungono facilmente le 300.000 richieste al giorno]

[The **GET** HTTP method requests a representation of the specified resource. Requests using GET should only be used to request data and shouldn't contain a body. "GET <request-target>["?"<query>] **HTTP/1.1** <request-target> Identifies the target resource of the request when combined with the information provided in the Host header (specifies the host and port number of the server to which the request is being sent). This is an absolute path, e.g., "/path/to/file.html", in requests to an origin server (a specialized type of web server where the original version of the web resources reside), and an absolute URL in requests to proxies (a proxy server is a server application that acts as an intermediary between a client requesting a resource and the server providing that resource; it improves privacy, security, and possibly performance in the process) e.g., "<http://www.example.com/path/to/file.html>". <query> Is an optional query component preceded by a question-mark "?". Often used to carry identifying information in the form of "key=value" pairs]

## LAB 29-11-24

- [JS] Tutti gli esercizi hanno il file HTML e quello JS associato.



# Esercizio 1

- Dati i file nella cartella 01-hello-world, scrivere il codice Javascript in modo che:
  - Venga stampato in console la stringa «Hello World»
  - Venga inserita la stringa «Hello World» all'interno dello span con id ciao.
  - Venga inserita la stringa «2021» all'interno del paragrafo con classe anno.

[1] Noi abbiamo la nostra pagina HTML, in fondo alla quale viene incluso lo script JS. L'output in console si ottiene per mezzo di "**console.log()**" dove "console" rappresenta un oggetto e "log()" un suo metodo, che richiede come argomento la stringa da visualizzare.

Aprendo ora la pagina HTML visualizziamo i placeholder ma "Hello World!" non compare, cosa abbiamo sbagliato? La pagina non è la console: noi inviamo l'output in console, pertanto il risultato non compare all'interno della pagina.

Per aprire la console dovremmo tenere premuto il tasto "control", cliccare nella pagina web, selezionare "Inspect Element" ed entrare nella sezione "Console", dove noteremo esserci il messaggio, accompagnato da un riferimento al file ed alla riga dove si trova l'istruzione che ha generato l'output. L'utilizzo della console va limitato a quello di strumento verso cui indirizzare l'output in fase di debugging, non dobbiamo fornire informazioni all'utente in console.

[2] Come facciamo a modificare il DOM inserendo un nuovo testo? Occorre farlo in due passaggi: in primis dovremo recuperare dal DOM un riferimento all'elemento (tag) all'interno di cui si ha intenzione di inserire il testo poi, una volta ottenuto, dovremo manipolarne il contenuto. Il riferimento lo salviamo in una variabile, che dichiareremo con la keyword "const".

- [JS] Una variabile dichiarata con "**const**" non può essere riassegnata, ossia non potremo fare una seconda operazione di assegnamento ma, nonostante ciò, per alcuni tipi di dato rimarrà possibile modificarne il valore, ad esempio gli array: in seguito all'inizializzazione saremo comunque in grado di aggiungere elementi alla struttura dati, poiché il

valore associato alla variabile, nonché l'indirizzo della prima cella di memoria del vettore, rimane invariata. Perché adottiamo questa filosofia? Dichiariamo tutte le variabili che sappiamo non dovranno subire modifiche con "const" in modo da prevenire errori. Le variabili che invece dovranno inevitabilmente cambiare valore, ad esempio l'indice di un ciclo, verranno dichiarate con la keyword "**let**".

- [JS] Quali differenze ci sono tra "let" e "var"? "var" è il modo antiquato di dichiarare le variabili in JS ed utilizzarlo porta penalità. "let" introduce lo scope di blocco, per cui se noi dichiarassimo una variabile con "let" all'interno di un ciclo, fuori da questo la variabile non esisterebbe. "var", al contrario, genera una variabile globale e vi è il cosiddetto "hoisting" ("to lift something heavy", the process whereby the interpreter appears to move the declaration of functions, variables, classes, or imports to the top of their scope, prior to execution of the code) della variabile.
- [JS] "**getElementById()**" è un metodo di "document", ossia l'oggetto che rappresenta il documento della pagina, e mette a disposizione diversi metodi per recuperare un'elemento (tag) in base alle sue caratteristiche. Ci sono alternative a "getElementById()"? Posso fare la stessa cosa con un metodo diverso? Sì, attraverso "**querySelector()**" che richiede in input un selettore con la stessa sintassi utilizzata nel foglio di stile. Vediamo esserci anche un metodo "**querySelectorAll()**": qual è la differenza con quello visto in precedenza? "querySelector()" applica il selettore, recupera tutti gli elementi che fanno match con esso e ritorna la prima corrispondenza trovata. "querySelectorAll()" invece recupera tutti gli elementi e li restituisce tutti.

Come posso scrivere il selettore per recuperare lo <span>? Tramite l'id con "#ciao" o, con maggiore specificità "span#ciao", oppure essere più specifico ancora e scrivere "body > span#ciao" o, analogamente, "body span#ciao". Per verifica, se digitassimo "console.log(tagHello)" dovrebbe stampare in console il codice HTML relativo al tag referenziato dalla variabile. Ora, come faccio a cambiare il contenuto di un tag? Ci sono due metodi: "innerText" e "innerHTML".

- [JS] "**innerHTML**" è una vera e propria proprietà di "Element". Se noi ne facessimo il log, in console verrebbe stampato il testo contenuto tra il tag di apertura e quello di chiusura, in questo caso "Placeholder". Possiamo anche settarla ed assegnarle un valore, ad esempio "Hello World!". Se adesso ricaricassimo la pagina, in teoria, quello che dovremmo vedere sarebbe la stringa "Placeholder" che dopo qualche istante viene rimpiazzata da "Hello World!".

[3] Il primo passo sarà recuperare il tag dal DOM attraverso il metodo "**getElementsByClassName()**" dell'oggetto "document", salvandolo sempre in una variabile. Che differenze ci sono tra il metodo "getElementsByClassName()" e "getElementById()"? Il primo è al plurale poiché nel DOM, mentre dovremo essere sicuri, a meno di errori, che vi sia un solo elemento con un determinato id, è invece consentita la presenza di più elementi aventi la stessa classe. Quindi il nome del metodo stesso ci suggerisce l'output, che non sarà più un singolo elemento ma un'**array** di elementi, per cui se volessimo ottenere una sola corrispondenza dovremmo accedere in posizione 0 (perché sappiamo essere l'unico elemento all'interno del DOM con quella classe), stessa cosa nel caso di "querySelectorAll()".

Ma visto che "tagHello" lo utilizziamo solamente un'altra volta, non potremmo concatenare la modifica della proprietà al metodo "document.querySelector()", ottenendo "document.querySelector("#ciao").innerHTML = "Hello World!";"? Si, è consentito essendo JS oltre ad un linguaggio imperativo anche un linguaggio di **paradigma funzionale**, il quale implica che non esistano le funzioni chiamate "procedure", ossia senza output: tutte le funzioni hanno un output che è l'oggetto che le ha chiamate. Quindi se un metodo dovesse restituire, ad esempio, una stringa, noi potremmo fare chaining dei vari metodi associati alle stringhe. Perché è importante invece, nel caso in cui andassimo a riutilizzare più volte un elemento all'interno di una pagina, salvarne il riferimento in una variabile? Quando selezioniamo un elemento dal DOM stiamo sprecando **potenza computazionale** poiché il DOM, che nel nostro esempio è molto semplice, potrebbe essere arbitrariamente complesso ed fare una ricerca ed una selezione anziché N su di esso, consentirebbe un risparmio di tempo & risorse. Solitamente viene data poca attenzione lato client essendo il tempo di computazione a carico di chi vi visita, tuttavia è buona norma ottimizzare il codice lato client.

```
01-hello-world > <> hello.html > ...
1 <!doctype html>
2 <html lang="it">
3 <head>
4 <meta charset=utf-8 />
5 <title>Esercizio Hello World</title>
6 </head>
7 <body>
8 Placeholder
9 <p class="anno">
10 | Placeholder
11 </p>
12 <script src="hello.js"></script>
13 </body>
14 </html>
```

```
01-hello-world > js hello.js > ...
1 console.log("Hello World");
2
3 //const tagHello = document.getElementById("ciao");
4 const tagHello = document.querySelector("#ciao");
5 tagHello.innerHTML = "Hello World";
6
7 //const tagYear = document.getElementsByClassName("anno")[0];
8 //const tagYear = document.querySelector(".anno");
9 const tagYear = document.querySelectorAll(".anno")[0];
10 tagYear.innerText = "2021";
```

- [JS]



## Esercizio 2

- Dati i file nella cartella 02-stringhe, scrivere il codice Javascript in modo che:
  - Al click sul bottone «Testo uppercase» il testo contenuto all'interno del div con id risultato diventi tutto maiuscolo.
  - Al click sul bottone «Testo lowercase» il testo contenuto all'interno del div con id risultato diventi tutto minuscolo.
  - Al click sul bottone «Testo substring» i primi 5 caratteri dovranno essere spostati in fondo al testo.

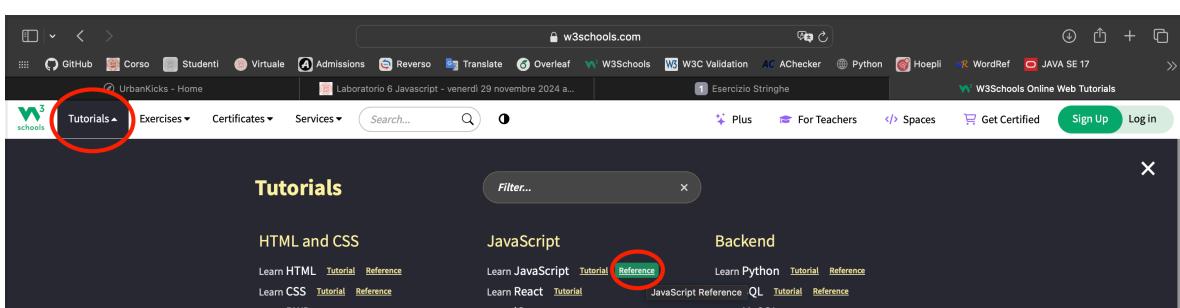
Analogamente all'esercizio precedente, l'import del file javascript all'interno del file HTML viene fatto appena prima della chiusura del tag <body>, tramite "<script src='stringhe.js' type='text/javascript'></script>".

[1] Per risolvere il primo punto cos'è che dobbiamo gestire? Gli eventi associati al bottone, quindi noi dobbiamo fare in modo che quando l'utente clicca sul primo bottone succeda qualcosa, per cui bisognerà innanzitutto recuperare il bottone. Per far ciò utilizziamo "querySelectorAll()", in particolare preleviamo il valore in prima posizione all'interno del vettore restituito con "document.querySelectorAll(input)[0];" che saremo sicuri essere quello voluto poiché l'**ordine** è lo stesso con cui gli elementi compaiono all'interno del codice **HTML**. In alternativa sarebbe bastato "document.querySelector(input);" poiché nonostante il metodo faccia match con tutte le occorrenze del tag <input> solo la prima verrà restituita. Utilizzando le pseudo-classi avremmo potuto scrivere "document.querySelector("input:first-child");" oppure "document.querySelector("input:nth-child(1)");" o ancora "document.querySelector("input:**first-of-type**");" che non indica necessariamente il primo figlio ma rappresenta il primo elemento del suo tipo (tag name) tra un gruppo di elementi gemelli. Volendo invece fare un selettore che tenga conto del valore di uno dei due attributi potremmo sfruttare la notazione "element[attribute='value']". Attenzione in questo caso a non innestare doppi apici.

```
document.querySelector("input[value='Testo uppercase']");
document.querySelector("input[value=Testo uppercase]");
```

- [JS] Come facciamo ad aggiungere un listener al nostro bottone? Attraverso il metodo "**addEventListener()**" che accetta due parametri: il primo è una stringa che indica l'evento il cui svolgersi vogliamo triggeri un'azione , in questo caso "click", il secondo è appunto la funzione che viene invocata al compiersi dell'evento. Qui potremmo mettere il nome di una funzione e poi definirla esternamente o definire una funzione chiamata "anonima", ossia senza nome, direttamente sul posto.

Corpo della funzione: cosa dobbiamo fare? Prendere il testo, modificarlo, e riassegnarlo al DOM. Per prelevare il testo risaliamo al tag <div> attraverso una "querySelector()" e ne selezioniamo il contenuto con "innerHTML", salvandolo in una variabile. Per trasformare la stringa in uppercase utilizziamo il metodo "toUpperCase()" ricordandoci di riassegnare il valore di ritorno poiché essendo le **stringhe in JS immutabili**, i metodi non modificano le stringhe su cui vengono applicate.



Sarebbe opportuno navigare il DOM una volta soltanto, assegnando ad una variabile dichiarata con "const" la selezione del <div>.

[2] Analogamente al primo, cambiando il selettore del bottone e la funzione di manipolazione della stringa.

- [JS] "document.querySelector("input:last-child")" ritorna "null" perché l'ultimo figlio di <body> non è un tag <input>!
- [JS] La concatenazione di stringhe in JS si ottiene con il simbolo "+".

[3] Perché al primo click sul bottone "Testo substring" non accade nulla mentre dal secondo in poi svolge la funzione prevista? Poiché ci sono degli spazi nel file HTML. Qui veniamo alla differenza tra "**innerText**" ed "**innerHTML**": quest'ultimo prende anche i tag HTML contenuti nell'elemento, mentre il primo prende solo il testo, ma cambia anche il testo che testo prendono: "innerHTML" prende tutto il contenuto dalla chiusura del tag di apertura ">", all'apertura del tag di chiusura "<" considerando quindi anche gli spazi di intestazione.

"innerText" invece seleziona dal primo carattere che non sia né un'andata a capo né uno spazio, fino all'ultimo carattere che non sia né un'andata a capo né uno spazio, quindi dopo qualche di click inizierà a mangiarsi gli spazi del Lorem Ipsum (Lorem sono 5 caratteri e dopo essere stati messi in fondo, alla seconda lettura il testo viene ripreso direttamente dalla "i" di ipsum senza spazio poiché questo finisce all'inizio e non viene considerato).

```
02-stringhe > <> esercizio_stringhe.html > ...
1 <!DOCTYPE html>
2 <html lang="it">
3 <head>
4 <meta charset="UTF-8">
5 <title>Esercizio Stringhe</title>
6 </head>
7 <body>
8 <input type="button" value="Testo uppercase" />
9 <input type="button" value="Testo lowercase" />
10 <input type="button" value="Testo substring" />
11 <div>
12 | Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do
13 | </div>
14 <script src="stringhe.js" type="text/javascript"></script>
15 </body>
16 </html>
```

```
02-stringhe > JS stringhe.js > ...
1 const risultato = document.querySelector("div");
2
3 document
4 //querySelector("input")
5 //querySelectorAll("input")[0]
6 //querySelector("input:first-child")
7 //querySelector("input:first-of-type")
8 //querySelector("input:nth-child(1)")
9 .querySelector("input[value='Testo uppercase']")
10 .addEventListener("click", function(){
11 let testo = risultato.innerHTML;
12 testo = testo.toUpperCase();
13 risultato.innerHTML = testo;
14 });
15
16 document
17 //querySelectorAll("input")[1]
18 //querySelector("input:nth-child(2)")
19 .querySelector("input[value='Testo lowercase']")
20 .addEventListener("click", function(){
21 let testo = risultato.innerHTML;
22 testo = testo.toLowerCase();
23 risultato.innerHTML = testo;
24 });
25
26 document
27 //querySelectorAll("input")[2]
28 //querySelector("input:nth-child(3)")
29 .querySelector("input:last-of-type")
30 .addEventListener("click", function(){
31 let testo = risultato.innerHTML;
32 //let testo_spostato = testo.substring(5, testo.length) + testo.substring(0, 5);
33 let testo_spostato = testo.slice(5, testo.length) + testo.slice(0, 5);
34 risultato.innerHTML = testo_spostato;
35 });
36
```

- [JS]



## Esercizio 3

- Dati i file nella cartella 03-oggetti, scrivere il codice Javascript in modo che:
  - Sia possibile definire una classe Computer con 3 proprietà (processore, disco e ram) e due metodi (infoComputerConsole e infoComputerDOM).
  - infoComputerConsole si occupa di stampare in console le 3 proprietà.
  - infoComputerDOM si occupa di visualizzare le 3 proprietà nel DOM all'interno di paragrafi distinti. Il metodo accetta in input l'id del tag html all'interno del quale andranno inseriti i paragrafi.

- [JS] Come si definisce una **classe** in JS? Ci sono due alternative: una utilizzando il metodo originario e l'altra mediante quello che è uno zucchero sintattico introdotto con ECMAScript 2015. Il primo segue la sintassi "function nomeDellaClasse(par1, par2){}" dove tra parentesi tonde mettiamo i parametri del costruttore della classe, ed all'interno delle graffe definiamo le proprietà con la keyword "this". In ECMAScript 2015 è stato messo a disposizione uno zucchero sintattico il cui output è esattamente equivalente al tipo di sintassi utilizzata in questo esercizio, ma rende JS più familiare per chi viene dal paradigma OO. VSCode ci suggerisce di convertire il costruttore "function" ad una dichiarazione di classe ES2015 (con la keyword "class", il costruttore, i metodi), ma sotto ci sarà sempre il concetto di prototipo e la classe di fatto non esisterà.
- [JS] A questo punto come definisco il metodo? Potremmo avere la tentazione di scrivere "this.infoComputerConsole = function(){})", e sarebbe funzionante, poiché nei linguaggi di paradigma funzionale le funzioni sono elementi di prima classe e possono pertanto essere passate come parametri ed anche associate come valori ad una variabile ed ad una proprietà. Non è tuttavia il modo corretto di farlo. Essendo JS object-based ma non class-based, sfrutta gli oggetti utilizzando il **prototipo**: noi sul nome della classe abbiamo una proprietà che è "prototype" che condivide il prototipo di tutti gli

oggetti di quella classe (che poi in realtà non è una classe) e a questo punto noi possiamo andare a modificare il prototipo aggiungendo quelli che sono i metodi. Definiamo quindi ciascun metodo sul prototipo in modo che siano poi fruibili da tutti gli oggetti ed assegniamo loro una funzione.

- [JS] Come definisco un **oggetto**? Lo salviamo in una variabile quindi usiamo "const", keyword "new" per istanziare l'oggetto, nome della classe seguito da parentesi tonde contenenti i valori da passare al costruttore. Posso poi invocare i metodi tramite l'oggetto.

Nel secondo metodo usiamo i backtick che ci consentono di definire stringhe multi-linea ed organizzare bene il codice HTML in modo che venga strutturato in maniera corretta. Come faccio ora a visualizzare i valori delle variabili all'interno della stringa? Noi sapevamo che nelle stringhe delimitate dai doppi apici potevamo mettere il nome della variabile direttamente all'interno, e che con i singoli apici questa interpolazione non viene fatta. Con i backtick dobbiamo utilizzare la notazione "**`\${nomeVariabile}`**".

```
03-oggetti > <> esercizio_oggetti.html > ...
1 <!DOCTYPE html>
2 <html lang="it">
3 | <head>
4 | | <meta charset=utf-8 />
5 | | <meta name="viewport" content="width=device-width, initial-scale=1.0">
6 | | <title>Esercizio Oggetti</title>
7 | </head>
8 | <body>
9 | | <main>
10 | | | <h1>Caratteristiche PC</h1>
11 | | | <div id="miopc">
12 |
13 | | | </div>
14 | | | <div id="miopc2">
15 |
16 | | | </div>
17 | | </main>
18 | | <script src="oggetti.js" type="text/javascript"></script>
19 | </body>
20 </html>
```

```

03-oggetti > js oggetti.js > Computer > constructor
 1 function Computer(processore, disco, ram) {
 2 this.processore = processore;
 3 this.disco = disco;
 4 this.ram = ram;
 5 }
 6 Computer.prototype.infoComputerConsole = function() {
 7 console.log("Processore: " + this.processore + "\nDisco: " + this.disco + "\nRam: " + this.ram)
 8 };
 9
10 Computer.prototype.infoComputerDOM = function(id){
11 document.getElementById(id).innerHTML =
12 <p>Processore: ${this.processore}</p>
13 <p>Disco: ${this.disco}</p>
14 <p>RAM: ${this.ram}</p>
15 `;
16 }
17
18 class Computer2 {
19 constructor(processore, disco, ram) {
20 this.processore = processore;
21 this.disco = disco;
22 this.ram = ram;
23 }
24 infoComputerConsole = function () {
25 console.log("Processore: " + this.processore + "\nDisco: " + this.disco + "\nRam: " + this.ram);
26 }
27 infoComputerDOM (id) {
28 document.getElementById(id).innerHTML =
29 <p>Processore: ${this.processore}</p>
30 <p>Disco: ${this.disco}</p>
31 <p>RAM: ${this.ram}</p>
32 `;
33 }
34 }
35 }
36
37
38 const mioPc = new Computer("i7", "500GB", "16GB");
39 mioPc.infoComputerConsole();
40 mioPc.infoComputerDOM("miopc");
41
42 const mioPc2 = new Computer("i5", "250GB", "16GB");
43 mioPc2.infoComputerConsole();
44 mioPc2.infoComputerDOM("miopc2");

```

- [JS]



## Esercizio 4

- Dati i file nella cartella 04-articoli, scrivere il codice Javascript in modo che:
  - Vengano correttamente inseriti gli articoli all'interno della pagina, secondo la struttura generata nelle lezioni precedenti.

I dati degli articoli sono in un array di oggetti nel file "articoli.js". Il tag `<main>` del file HTML sarà vuoto ed io voglio **generare** il contenuto relativo agli articoli attraverso JS, come fare? In primis dovremo ricavare un riferimento a `<main>`, poi dovremo iterare sul nostro array generando la struttura ed appendendola all'interno del tag `<main>`. Attenzione a delimitare con doppi apici la variabile che verrà espansa nel percorso dell'immagine, poiché necessari a definire l'inizio e la fine del valore assegnato all'attributo "src". Infatti nonostante la variabile sia una stringa, quando viene stampata non avrà i doppi apici.

Perché dovrei preferire fare una roba di questo tipo, e tutti i siti moderni lo fanno, rispetto a generare la stessa cosa con PHP come abbiamo visto nel laboratorio scorso? L'idea di base è quella di spostare la **computazione** lato **client**, quindi la generazione della pagina la faccio lato client e non più lato server. Inoltre facendo ciò ottimizzo anche la banda poiché anziché mandare tutti i dati degli articoli all'interno già dei tag nella pagina, io mando la pagina vuota, lo scheletro dell'`<article>` ce l'ho una sola volta ed i dati li mando poi tutti insieme sotto forma di JSON, quindi consumo meno **banda**. E tutte le volte che andrò a fare una richiesta non richiederò più la pagina intera ma solamente il contenuto che mi serve. Quindi l'`<header>`, il `<nav>`, l'`<aside>` ed il `<footer>` li mando una volta sola mentre con PHP li rimandavamo ad ogni richiesta.

```

04-articoli > index.html > html > body > aside > section > ul > li > a
1 <!DOCTYPE html>
2 <html lang="it">
3 <head>
4 <meta name="viewport" content="width=device-width, initial-scale=1.0" />
5 <title>Blog TW - Home</title>
6 <link rel="stylesheet" type="text/css" href=".//css/style.css" />
7 </head>
8 <body>
9 <header>
10 <h1>Blog di Tecnologie Web</h1>
11 </header>
12 <nav>
13
14 HomeArchivioContattiLogin
15
16 </nav>
17 <main>
18 </main><aside>
19 <section>
20 <h2>Post Popolari</h2>
21
22
23
24 Intro alle Tecnologie Web Client Side
25
26
27
28 Intro alle Tecnologie Web Server Side
29
30
31 </section>
32 <section>
33 <h2>Categorie</h2>
34
35 HTML
36 CSS
37 PHP
38 Javascript
39 jQuery
40 Apache
41
42 </section>
43 </aside>
44 <footer>
45 <p>Tecnologie Web - A.A. 2019/2020</p>
46 </footer>
47 <script src="articoli.js" type="text/javascript"></script>
48 </body>
49 </html>

```

```

04-articoli > JS articoli.js > ...
1 const datiArticoli = [
2 "Autore": "Gino Pino",
3 "Data": "2 Ottobre 2019",
4 "Titolo": "Intro alle Tecnologie Web Client Side",
5 "Immagine": "./img/html5-js-css3.png",
6 "Testo": "'Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.'"
7 },
8 {
9 "Autore": "Cippa Lippa",
10 "Data": "2 Ottobre 2019",
11 "Titolo": "Intro alle Tecnologie Web Server Side",
12 "Immagine": "./img/php.png",
13 "Testo": "'Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.'"
14 }
15
16 const main = document.querySelector("main");
17
18 for(let i=0; i < datiArticoli.length; i++){
19 let articolo =
20 <article>
21 <header>
22 <div>
23
24 </div>
25 <h2>${datiArticoli[i]["Titolo"]}</h2>
26 <p>${datiArticoli[i]["Autore"]} - ${datiArticoli[i]["Data"]}</p>
27 </header>
28 <section>
29 <p>${datiArticoli[i]["Testo"]}</p>
30 </section>
31 <footer>
32 Leggi tutto
33 </footer>
34 </article>
35 ;
36 main.innerHTML += articolo;
37 }

```

- [JS]



## Esercizio 5

- Dati i file nella cartella 05-tabella-dinamica, scrivere il codice Javascript in modo che vengano correttamente inserite le informazioni relative agli autori del blog nella tabella. 3 step incrementali:
  - Aggiungere correttamente le informazioni (l'intestazione della tabella può essere hard-coded).
  - Rendere la tabella accessibile.
  - Generalizzare la soluzione in modo che se vengono aggiunti (o rimossi) dei campi relativi agli autori, il codice non necessiti di modifiche.

```

05-tabella-dinamica > ◊ contatti.html > ◊ html > ◊ body > ◊ aside > ◊ section > ◊ ul > ◊ li > ◊ img
1 <!DOCTYPE html>
2 <html lang="it">
3 <head>
4 <meta name="viewport" content="width=device-width, initial-scale=1.0" />
5 <title>Blog TW - Home</title>
6 <link rel="stylesheet" type="text/css" href=".css/style.css" />
7 </head>
8 <body>
9 <header>
10 | <h1>Blog di Tecnologie Web</h1>
11 | </header>
12 <nav>
13 |
14 | HomeArchivioContattiLogin
15 |
16 </nav>
17 <main>
18 <section>
19 | <h2>Autori del Blog</h2>
20 | <table>
21 | <tr>
22 | <td></td>
23 | <td>Autore 1</td>
24 | <td>Autore 1</td>
25 </section>
26 <aside>
27 <section>
28 | <h2>Post Popolari</h2>
29 |
30 |
31 |
32 | Intro alle Tecnologie Web Client Side
33 |
34 |
35 |
36 | Intro alle Tecnologie Web Server Side
37 </section>
38 <section>
39 | <h2>Categorie</h2>
40 |
41 | HTML
42 | CSS
43 | PHP
44 | Javascript
45 | jQuery
46 | Apache
47
48 </section>
49 </aside>
50 <footer>
51 | <p>Tecnologie Web - A.A. 2019/2020</p>
52 </footer>
53 <script src="contatti.js" type="text/javascript"></script>
54 </body>

```

```

05-tabella-dinamica > js contatti.js > ...
1 const datiTabella = [
2 "Autore": "Gino Pino",
3 "Email": "ginopino@blogtw.com",
4 "Argomenti": "HTML, CSS, JS"
5 },
6 {
7 "Autore": "Cippa Lippa",
8 "Email": "cippalippa@blogtw.com",
9 "Argomenti": "PHP"
10]
11
12 function stringaToID(stringa){
13 return stringa.toLowerCase().replace(/[^a-zA-Z]/g, "");
14 }
15
16 const ths = Object.keys(datiTabella[0]);
17 const tabella = document.getElementById("tabella");
18
19 let th_row = "<tr>";
20 for(let i = 0; i < ths.length; i++){
21 th_row += `<th id="${stringaToID(ths[i])}">${ths[i]}</th>`;
22 }
23 th_row += "</tr>";
24
25 tabella.innerHTML += th_row;
26
27 for(let row = 0; row < datiTabella.length; row++){
28 let td_row = "<tr>";
29 let row_id = stringaToID(datiTabella[row][ths[0]]);
30 td_row += `<th id="${row_id}">${datiTabella[row][ths[0]]}</th>`;
31 for(let i = 1; i < ths.length; i++){
32 td_row += `<td headers="${row_id} ${stringaToID(ths[i])}">${datiTabella[row][ths[i]]}</td>`;
33 }
34 td_row += "</tr>";
35 tabella.innerHTML += td_row;
36 console.log(td_row)
37 }

```

## LAB 06-12-2024

- [PHP]



# Esercizio 1

- Dati i file nella cartella php, modificare la pagina login.php in modo che:
  - Di default dovrà essere visualizzato il form.
  - Se passati i parametri in Post, si dovrà controllare se esiste nel db un autore con quegli username e password e nel caso definire delle variabili di sessione.
  - Se l'utente è loggato, mostrare il template login-home, che visualizza gli articoli di un utente (richiede che siano passati tramite il campo articoli di templateParams).

Se l'utente tentasse di fare login con credenziali errate noi salveremmo un messaggio di errore all'interno di "\$templateParams["errorelogin"]", campo che verrà visualizzato prima degli <input> relativi ad username e password. "login-home.php" sarà invece il template da mostrare all'utente nella pagina "login.php" dopo aver effettuato l'accesso, e si compone di una tabella contenente la lista di articoli pubblicati dall'autore collegato. Per ogni articolo viene mostrato il titolo, l'immagine corrispondente e, nella colonna "Azione", due link alla pagina di gestione degli articoli, dove nell'url vengono specificati l'intervento che si intende fare sull'articolo (modifica o rimozione) e l'id di quest'ultimo. In riga 6 notiamo un terzo riferimento a "gestisci-articoli.php" con il parametro "action" posto ad 1, che coincide con l'azione di inserimento.

```

template > 🐄 login-form.php
1 <form action="#" method="POST">
2 <h2>Login</h2>
3 <?php if(isset($templateParams["errorelogin"])): ?>
4 <p><?php echo $templateParams["errorelogin"]; ?></p>
5 <?php endif; ?>
6
7
8 <label for="username">Username:</label><input
9 type="text" id="username" name="username" />
10
11
12 <label for="password">Password:</label><input
13 type="password" id="password" name="password" />
14
15
16 <input type="submit" name="submit" value="Invia" />
17
18
19 </form>

```

```

template > 🐄 login-home.php
1 <section>
2 <h2>Articoli</h2>
3 <?php if(isset($templateParams["formmsg"])): ?>
4 <p><?php echo $templateParams["formmsg"]; ?></p>
5 <?php endif; ?>
6 Inserisci Articolo
7 <table>
8 <tr>
9 <th>Titolo</th><th>Immagine</th><th>Azione</th>
10 </tr>
11 <?php foreach($templateParams["articoli"] as $articolo): ?>
12 <tr>
13 <td><?php echo $articolo["titoloarticolo"]; ?></td>
14 <td>" alt="" /></td>
16 <td>
17 <a href="gestisci-articoli.php?action=2&id=<?php
18 echo $articolo["idarticolo"]; ?>">Modifica
19 <a href="gestisci-articoli.php?action=3&id=<?php
20 echo $articolo["idarticolo"]; ?>">Cancella
21 </td>
22 </tr>
23 <?php endforeach; ?>
24 </table>
25 </section>

```

Nella query del metodo "checkLogin()" controlliamo che l'utente sia attivo, ipotizzando l'esistenza di un amministratore in grado di disabilitare gli utenti, negando loro la possibilità di accedere. Ma perché tenere ugualmente in memoria il record di un utente inattivo? Per rispettare i vincoli di integrità referenziale, considerando che ogni articolo mostra informazione sull'autore da cui è stato scritto.

```

db > database.php
148
149 public function checkLogin($username, $password){
150 $query = "SELECT idautore, username, nome FROM autore WHERE attivo=1 AND username = ? AND password
151 = ?";
152 $stmt = $this->db->prepare($query);
153 $stmt->bind_param("ss", $username, $password);
154 $stmt->execute();
155 $result = $stmt->get_result();
156
157 return $result->fetch_all(MYSQLI_ASSOC);

```

Avendo a disposizione due template specifici per la pagina "login.php" dovrò condurre dei controlli all'inizio di questa per decretare quale dei due mostrare. In primis dovrò stabilire se l'utente stia o meno cercando di fare login ed in secondo luogo se abbia effettuato l'accesso. Subentreremo nel primo scenario alla pressione del tasto "Invia" all'interno del form, ma come ce ne accorgeremo? Appurando di aver ricevuto in "\$\_POST" i parametri username e password. Cosa ci restituisce il metodo "checkLogin()" di "database.php"? "Idautore", "username" e "nome". Devo accertarmi che non mi sia stato restituito più di un autore? Assumiamo che il risultato sia al massimo di un record (alternativamente avremmo dovuto verificare o che "username" se non fosse stato ID sarebbe comunque stato UNIQUE, oppure avremmo potuto controllarlo a mano nel codice, sebbene questi vincoli siano solitamente implementati a livello di database, tipicamente in fase di registrazione, quindi possiamo essere sicuri).

Tutte le volte che leggiamo un input da "\$\_POST" o "\$\_GET" dovremmo fare un controllo sul tipo di dato che riceviamo. Utilizzando i *prepared statements* di MySQLi abbiamo la garanzia che l'utente non provi a fare injection, e che il tipo dei parametri sia coerente con quello che ci si aspetta. Regole più restrittive potrebbero essere applicate qualora imponessimo ad esempio un pattern per la password, e.g. almeno 1 numero ed 1 simbolo, in tal caso prima di inviare i dati sul database dovremmo validare la stringa lato codice. Un'altra accortezza consiste nel predisporre un utente ad eseguire solo query di selezione e non di inserimento/modifica/cancellazione per cercare di minimizzare il rischio che il database venga droppato nel caso ci bucassero il sistema. "\$login\_result" è un array di array associativi. Il trade-off nel mostrare un messaggio di errore generico nell'eventualità di credenziali sbagliate piuttosto che esplicitare il campo scorretto sta nel fatto che essendo il sistema protetto da username e password, se noi dicessemmo quale delle due è inesatta ci giocheremmo metà della nostra sicurezza, ma l'utente ne gioverebbe in termini di user-experience facilitandoli la correzione dell'errore.

Quale strumento di PHP mi consente di mantenere memoria del fatto che l'utente abbia eseguito il login, sapendo che HTTP è un protocollo state-less? Abbiamo visto "\$\_COOKIE" e "\$\_SESSION": i primi vengono salvati lato client, adatti quindi ad informazioni non sensibili, ma essendo il fatto che l'utente abbia eseguito o meno l'accesso un'informazione sensibile, ci serviremo della sessione. Dato un utente che ha inserito credenziali corrette, imposto tre

variabili di sessione per conservare memoria del fatto che sia entrato. Quindi per controllare se un utente sia o meno loggato mi basterà verificare l'esistenza di uno dei tre campi. Non c'è la funzione per effettuare il log-out ma consisterebbe nel cancellare il contenuto delle tre variabili tramite "unset()".

```
utils > 🐘 functions.php

12 function isUserLoggedIn(){
13 return !empty($_SESSION['idautore']);
14 }
15
16 function registerLoggedUser($user){
17 $_SESSION["idautore"] = $user["idautore"];
18 $_SESSION["username"] = $user["username"];
19 $_SESSION["nome"] = $user["nome"];
20 }
```

```
	utils > 🐘 login.php

4 // Controllo se l'utente sta facendo login.
5 if (isset($_POST["username"]) && isset($_POST["password"])) {
6 $login_result = $dbh->checkLogin($_POST["username"], $_POST
7 ["password"]);
8 if (count($login_result) == 0) {
9 // Login fallito
10 $templateParams["errorelogin"] = "Errore! Controllare username o
11 password";
12 } else {
13 registerLoggedUser($login_result[0]);
14 }
15 }
```

Terminata l'istruzione condizionale "if(){}" soprastante avremmo due possibilità:  
o l'autenticazione è andata a buon fine oppure il riconoscimento è fallito e  
dovrò continuare a mostrare all'utente il form. [Attenzione: le sessioni devono  
essere avviate per essere poi in grado creare/modificare/cancellare elementi,  
come lo facciamo? All'interno di "bootstrap.php" inseriamo la chiamata a  
funzione "session\_start();"]

## login.php

```
15 // Controllo se l'utente è loggato.
16 if (isUserLoggedIn()) {
17 // Utente loggato
18 $templateParams["titolo"] = "Blog TW - Admin";
19 $templateParams["nome"] = "login-home.php";
20 } else {
21 // Utente non loggato
22 $templateParams["titolo"] = "Blog TW - Login";
23 $templateParams["nome"] = "login-form.php";
24 }
25 $templateParams["categorie"] = $dbh->getCategories();
26 $templateParams["articolicasuali"] = $dbh->getRandomPosts(2);
27
28 require 'template/base.php';
29 ?>
```

In "login-home.php", "\$templateParams['formmsg']" rappresenta il feedback sulla riuscita o fallimento di un'azione (inserimento, modifica o cancellazione), da mostrare la prima volta che ritorno in "login.php" dopo aver richiesto l'operazione e cambiato pagina. Come lo imposto? Viene passato tramite "\$\_GET" per cui controlliamo che esista al suo interno un parametro con nome "formmsg" e nel caso lo assegnamo a "\$templateParams". In che senso "cambiato pagina"? Per via dell'ordine in cui compaiono le istruzioni: essendo che per compiere, ad esempio, un inserimento, dovrò trovarmi necessariamente in "login.php" (con template specifico "login-home.php"), e sebbene la stringa venga memorizzata in "\$\_GET", il codice dell'assegnamento prima "\$templateParams['formmsg'] = \$\_GET['formmsg'];" e della stampa poi "echo \$templateParams['formmsg'];?>" sarà già stato eseguito in precedenza, per cui per far sì che venga ri-eseguito dovrò, appunto, cambiare pagina e ritornare. Inoltre se l'assegnamento si trovasse dopo la stampa, vedrei il messaggio la \*seconda\* volta che ritorno sulla pagina in seguito all'operazione, ma questo non è il caso poiché il template specifico viene importato in "base.php", a sua volta accodato in fondo a "login.php".

## login.php

```
15 // Controllo se l'utente è loggato.
16 if (isUserLoggedIn()) {
17 // Utente loggato
18 $templateParams["titolo"] = "Blog TW - Admin";
19 $templateParams["nome"] = "login-home.php";
20
21 if(isset($_GET["formmsg"])) {
22 $templateParams["formmsg"] = $_GET["formmsg"];
23 }
24 } else {
25 // Utente non loggato
26 $templateParams["titolo"] = "Blog TW - Login";
27 $templateParams["nome"] = "login-form.php";
28 }
29 $templateParams["categorie"] = $dbh->getCategories();
30 $templateParams["articolicasuali"] = $dbh->getRandomPosts(2);
31
32 require 'template/base.php';
33 ?>
```

Ci serve ora la collezione di articoli dell'autore autenticato, su cui cicliamo in "login-home.php", che recuperiamo sfruttando il metodo "getPostByAuthorId()" già implementato in "database.php" ed utilizzando la variabile "\$\_SESSION" per recuperare l'ID dell'autore.

```
login.php

15 // Controllo se l'utente è loggato.
16 if (isUserLoggedIn()) {
17 // Utente loggato
18 $templateParams["titolo"] = "Blog TW - Admin";
19 $templateParams["nome"] = "login-home.php";
20 $templateParams["articoli"] = $dbh->getPostByAuthorId($_SESSION
21 ["idautore"]);
22 if(isset($_GET["formmsg"])) [
23 $templateParams["formmsg"] = $_GET["formmsg"];
24 }
25 } else {
26 // Utente non loggato
27 $templateParams["titolo"] = "Blog TW - Login";
28 $templateParams["nome"] = "login-form.php";
29 $templateParams["categorie"] = $dbh->getCategories();
30 $templateParams["articolicasuali"] = $dbh->getRandomPosts(2);
31
32 require 'template/base.php';
33 ?>
```

- [PHP]



## Esercizio 02

- Per l'esercizio 2, dovremo creare una pagina `gestisci-articoli.php`, che dovrà mostrare il template `admin-form.php` personalizzando il form in base all'azione scelta dall'utente (inserimento, modifica e cancellazione).
- Al submit del form, i dati dovranno essere inviati ad una pagina `processa-articolo.php`, che si occuperà di eseguire le azioni richieste e reindirizzerà l'utente alla pagina `login.php`.

[1] Alla pagina "gestisci-articoli.php" deve poterci accedere chiunque? No, solamente gli utenti che hanno effettuato l'accesso, per cui dovremo condurre un controllo sfruttando le funzioni presenti in "functions.php", e se ottenessimo che le variabili di "`$_SESSION`" non sono impostate allora andremo a re-indirizzare l'utente (infatti nonostante a "gestisci-articoli.php" ci si arrivi solamente dalla versione di "login.php" con template specifico "login-home.php", riservato ad utenti autenticati, io potrei comunque digitare il nome della pagina nell'URL). Come si fa il **redirect**? Cambiando l'intestazione con la funzione "header("location: login.php")".

```

🐘 gestisci-articoli.php
1 <?php
2 require_once "bootstrap.php";
3
4 if (!isUserLoggedIn()) {
5 header("location: login.php");
6 }
7
8 $templateParams["titolo"] = "BlogTW – Gestisci Articoli";
9 $templateParams["nome"] = "admin-form.php";
10 $templateParams["categorie"] = $dbh->getCategories();
11 $templateParams["postcasuali"] = $dbh->getRandomPosts(2);
12
13 require "template/base.php";
14 ?>

```

```

template > 🐘 admin-form.php
1 <form action="processa-articolo.php" method="POST" enctype="multipart/form-data">
2 <h2>Gestisci Articolo</h2>
3
4
5 <label for="titoloarticolo">Titolo:</label><input type="text" id="titoloarticolo"
6 name="titoloarticolo" value="" />
7
8
9 <label for="testoarticolo">Testo Articolo:</label><textarea id="testoarticolo"
10 name="testoarticolo"></textarea>
11
12
13 <label for="anteprimaarticolo">Anteprima Articolo:</label><textarea id="anteprimaarticolo"
14 name="anteprimaarticolo"></textarea>
15
16
17 <label for="imgarticolo">Immagine Articolo:</label><input type="file" name="imgarticolo"
18 id="imgarticolo" />
19
20
21 <input type="checkbox" id="" name="categoria_" /><label for="">Categoria</label>
22
23
24 <input type="submit" name="submit" value="Azione Articolo" />
 Annulla

 </form>

```

Gestiamo la logica dell'inserimento di un nuovo articolo nella pagina assegnata all'attributo "action" del `<form>` presente in "admin-form.php", dove ci sposteremo in seguito al submit. Internamente al form dovremo visualizzare una `<input type="checkbox">` per ogni categoria tramite un ciclo, non sarà necessario effettuare una lettura dal database essendo queste già memorizzate in `$templateParams["categorie"]`. Modifichiamo poi il testo presente sul tasto di invio del `<form>`, ma dove troviamo l'informazione sul tipo di azione che si sta cercando di compiere (inserimento/modifica/cancellazione)? In "login-home.php" i link relativi alle azioni assegnano nella query string al parametro `action` un valore tra {1, 2, 3} per cui in "gestisci-articoli.php" ci basterà esaminare il valore di `$_GET["action"]`. Sfruttiamo poi la funzione `getAction($action)` in "functions.php" che dato un valore numerico restituisce

la stringa dell'azione corrispondente. Infine reindirizziamo l'utente non solo se non è autenticato ma anche se ad action non è associato alcun valore poiché a quel punto non sapremmo cosa andare a far fare all'utente.

```
template > gestisci-articoli.php
1 <?php
2 require_once "bootstrap.php";
3
4 if (!isUserLoggedIn()) { // Controllo su action se non è settato o se non vale da 1 a 3
5 header("location: login.php");
6 }
7
8 $templateParams["titolo"] = "BlogTW - Gestisci Articoli";
9 $templateParams["nome"] = "admin-form.php";
10 $templateParams["categorie"] = $dbh->getCategories();
11 $templateParams["articolicasuali"] = $dbh->getRandomPosts(2);
12 $templateParams["azione"] = $_GET["action"];
13
14 require "template/base.php";
15 ?>
```

```
template > admin-form.php
1 <?php
2 $azione = getAction($templateParams["azione"]);
3 ?>
4 <form action="processa-articolo.php" method="POST" enctype="multipart/form-data">
5 <h2>Gestisci Articolo</h2>
6
7
8 <label for="titoloarticolo">Titolo:</label><input type="text" id="titoloarticolo" name="titoloarticolo" value="" />
9
10
11 <label for="testoarticolo">Testo Articolo:</label><textarea id="testoarticolo" name="testoarticolo"></textarea>
12
13
14 <label for="anteprimaarticolo">Anteprima Articolo:</label><textarea id="anteprimaarticolo" name="anteprimaarticolo"></
15 textarea>
16
17
18 <label for="imgarticolo">Immagine Articolo:</label><input type="file" name="imgarticolo" id="imgarticolo" />
19
20
21 <?php foreach ($templateParams["categorie"] as $categoria): ?>
22 <input type="checkbox" id=<?php echo $categoria["idcategoria"] ?>" name="categoria_<?php echo $categoria
23 ["idcategoria"] ?>" /><label for=<?php echo $categoria["idcategoria"] ?>"><?php echo $categoria["nomecategoria"] ?></
24 label>
25 <?php endforeach; ?>
26
27
28 <input type="submit" name="submit" value=<?php echo $azione ?>" />
29 Annulla
30
31
32 </form>
```

```
template > login-home.php
1 <section>
2 <h2>Articoli</h2>
3 <?php if(isset($templateParams["formmsg"])): ?>
4 <p><?php echo $templateParams["formmsg"]; ?></p>
5 <?php endif; ?>
6 Inserisci Articolo
7 <table>
8 <tr>
9 <th>Titolo</th><th>Immagine</th><th>Azione</th>
10 </tr>
11 <?php foreach($templateParams["articoli"] as $articolo): ?>
12 <tr>
13 <td><?php echo $articolo["titoloarticolo"] ; ?></td>
14 <td><img src=<?php echo UPLOAD_DIR.$articolo["imgarticolo"] ; ?>" alt="" /></td>
15 <td>
16 <a href="gestisci-articoli.php?action=2&id=<?php echo $articolo["idarticolo"] ; ?>">Modifica
17 <a href="gestisci-articoli.php?action=3&id=<?php echo $articolo["idarticolo"] ; ?>">Cancella
18 </td>
19 </tr>
20 <?php endforeach; ?>
21 </table>
22 </section>
```

[2] Andiamo ora ad implementare la pagina dove verremo reindirizzati all'atto di submit del form contenuto nel template specifico "admin-form.php" di

"login.php" [The action attribute specifies where to send the form-data when a form is submitted <form action="*URL*"> - W3schools]. Una volta terminato l'inserimento dei dati ed inviato il modulo avrebbe senso tornare alla tabella di "login.php", per cui nel file "processa-articolo.php" non dovremo né importare "base.php" né in generale preoccuparci dei template, ma al massimo dovremo impostare un messaggio. Se io non controllassi che l'utente si sia autenticato e verificassi solamente la presenza di dati passati in "\$\_POST" per l'inserimento, un utente non loggato potrebbe aggiungere un articolo se indovinasse i campi? Sì, quindi dovremo implementare nuovamente un reindirizzamento nell'eventualità in cui i campi opportuni di "\$\_SESSION" non siano impostati. Come facciamo a capire se i dati ricevuti si riferiscono ad una richiesta di aggiunta, modifica o cancellazione di un articolo? Potrei accorgermene dal valore dell'attributo "value" del bottone di submit, venendo spedito. Potrebbe però essere comodo avere "\$azione", per cui lo inoltro attraverso il <form> tramite un <input> con attributo "type=hidden" [A control that is not displayed but whose value is submitted to the server - MDN Web Docs] considerando che non avrebbe senso mostrare all'utente un campo <input> con già un valore in {1, 2, 3} non modificabile.

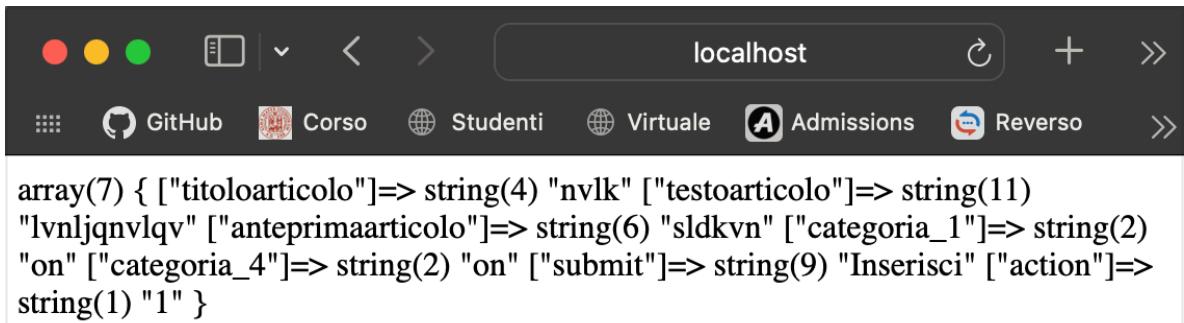
- [PHP] The "**var\_dump()**" function dumps (svuota, scarica) information about one or more variables. The information holds type and value of the variable(s).

Come faccio a controllare quali categorie, quindi quali <input type="checkbox"/> siano stati selezionati, avendo appurato dalla stampa "var\_dump(\$\_POST)" come vengano inviati solamente i valori dell'attributo "name" corrispondenti alle caselle spuntate? Le dobbiamo leggere tutte dal database e tramite un ciclo cerchiamo quelle per cui la variabile "\$\_POST['categoria\_X']" risulta impostata.

```

processa-articolo.php
1 <?php
2
3 require_once "bootstrap.php";
4
5 if (!isUserLoggedIn()) { // Controllo su action se esiste.
6 header("location: login.php");
7 }
8
9
10 if ($_POST["action"] == 1) {
11 // Inserimento.
12 $titoloarticolo = $_POST["titoloarticolo"];
13 $testoarticolo = $_POST["testoarticolo"];
14 $anteprimaarticolo = $_POST["anteprimaarticolo"];
15 $dataarticolo = $_POST["dataarticolo"];
16 $autore = $_SESSION["idautore"];
17
18 $categorie = $dbh->getCategories();
19 $categorie_inserite = array();
20 foreach($categorie as $categoria) {
21 if (isset($_POST["categoria_". $categoria["idcategoria"]])) {
22 array_push($categorie_inserite, $categoria["idcategoria"]);
23 }
24 }
25
26 var_dump($_POST);
27
28
29
30 $msg = "Qualcosa";
31 // Tornare su login.php
32 ?>

```



Come viene gestita l'immagine? Nel file "functions.php" è presente la funzione "uploadImage()" che richiede come parametri l'immagine letta dal form ed il percorso-destinazione della cartella dove inserirla. Estraggo il nome dell'immagine tagliandone il percorso, venendo la figura caricata sul server in una cartella temporanea, e compongo il path completo concatenando il percorso-destinazione con il nome dell'immagine. Definisco dei vincoli sulla dimensione dell'immagine e sull'estensione, oltre alla variabile \$result che varrà 1 se l'operazione è andata a buon fine, 0 altrimenti. Infatti in seguito vengono condotti svariati controlli (e.g. sul fatto che l'immagine sia realmente tale, sul peso del file, se esiste un'immagine con lo stesso nome nel percorso specificato ed in tal caso gestisco questo conflitto appendendo un

incrementale in fondo al nome) dove, in caso di errore assegno un messaggio che lo descriva a \$msg. Al termine dei controlli, se non ho registrato errori provo a spostare l'immagine dal percorso provvisorio del server al percorso corretto

Come si usa questa funzione lato server? L'immagine è presente nell'array \$\_POST? NO, perché finisce in \$\_FILES. Quando la chiamiamo passiamo come percorso dove caricare il documento la costante che punta alla nostra cartella "upload", mentre l'immagine la recuperiamo da \$\_FILES utilizzando come chiave il nome dell'`<input>` del form, ossia "imgarticolo". La funzione restituisce un array contenente due variabili, che scomponiamo per mezzo della funzione "list()".

Se l'operazione è terminata con esito positivo in \$img troverò il nome dell'immagine. Infine inserisco l'articolo nel database mediante un metodo di "database.php": "insertArticle()", che ritorna l'id incrementale fornito dal DBMS, che lo assegna autonomamente al record appena introdotto nella base di dati.

Manca ancora qualcosa? Sì, le categorie dell'articolo. Perché non le abbiamo trattate fino ad ora? Poiché per come è stato progettato il database queste non sono campi dell'entità "articolo" ma risultano collegate ad esso attraverso l'associazione reificata "articolo\_ha\_categoria", per cui avevamo bisogno dell'ID dell'articolo per inserirle.

```
utils > 🌐 functions.php
44 function uploadImage($path, $image){
45 $imageName = basename($image["name"]);
46 $fullPath = $path.$imageName;
47
48 $maxKB = 500;
49 $acceptedExtensions = array("jpg", "jpeg", "png", "gif");
50 $result = 0;
51 $msg = "";
52 //Controllo se immagine è veramente un'immagine
53 $imageSize = getimagesize($image["tmp_name"]);
54 if($imageSize === false) {
55 $msg .= "File caricato non è un'immagine! ";
56 }
57 //Controllo dimensione dell'immagine < 500KB
58 if ($image["size"] > $maxKB * 1024) {
59 $msg .= "File caricato pesa troppo! Dimensione massima è $maxKB KB. ";
60 }
61
62 //Controllo estensione del file
63 $imageFileType = strtolower(pathinfo($fullPath,PATHINFO_EXTENSION));
64 if(!in_array($imageFileType, $acceptedExtensions)){
65 $msg .= "Accettate solo le seguenti estensioni: ".implode(", ", $acceptedExtensions);
66 }
67
68 //Controllo se esiste file con stesso nome ed eventualmente lo rinomino
69 if (file_exists($fullPath)) {
70 $i = 1;
71 do{
72 $i++;
73 $imageName = pathinfo(basename($image["name"]), PATHINFO_FILENAME)."_" . $i . ".$imageFileType";
74 } while(file_exists($path.$imageName));
75 $fullPath = $path.$imageName;
76 }
77
78 //Se non ci sono errori, sposto il file dalla posizione temporanea alla cartella di destinazione
79 if(strlen($msg)==0){
80 if(!move_uploaded_file($image["tmp_name"], $fullPath)){
81 $msg.= "Errore nel caricamento dell'immagine.";
82 }
83 else{
84 $result = 1;
85 $msg = $imageName;
86 }
87 }
88 }
89
90 return array($result, $msg);
91 }
```

```
processa-articolo.php
1 <?php
2 require_once 'bootstrap.php';
3
4 if(!isUserLoggedIn() || !isset($_POST["action"])){
5 header("location: login.php");
6 }
7 if($_POST["action"]==1){
8 //Inserisco
9 $titoloarticolo = htmlspecialchars($_POST["titoloarticolo"]);
10 $testoarticolo = htmlspecialchars($_POST["testoarticolo"]);
11 $anteprimaarticolo = htmlspecialchars($_POST["anteprimaarticolo"]);
12 $dataarticolo = date("Y-m-d");
13 $autore = $_SESSION["idautore"];
14
15 $categorie = $dbh->getCategories();
16 $categorie_inserite = array();
17 foreach($categorie as $categoria){
18 if(isset($_POST["categoria_".$categoria["idcategoria"]])){
19 array_push($categorie_inserite, $categoria["idcategoria"]);
20 }
21 }
22
23 list($result, $msg) = uploadImage(UPLOAD_DIR, $_FILES["imgarticolo"]);
24 if($result != 0){
25 $imgarticolo = $msg;
26 $id = $dbh->insertArticle($titoloarticolo, $testoarticolo, $anteprimaarticolo, $dataarticolo, $imgarticolo, $autore);
27 if($id!=false){
28 foreach($categorie_inserite as $categoria){
29 $ris = $dbh->insertCategoryOfArticle($id, $categoria);
30 }
31 $msg = "Inserimento completato correttamente!";
32 }
33 } else{
34 $msg = "Errore in inserimento!";
35 }
36 }
37 }
38 header("location: login.php?formmsg=".$msg);
39 }
```

La sintassi è molto semplice. La funzione ha un solo parametro in input posto tra parentesi tonde. In genere è una variabile stringa.

```
htmlspecialchars($stringa);
```

- [A cosa serve?](#)
- [Un esempio pratico](#)
- [La funzione htmlentities\(\)](#)

### A cosa serve?

Questa funzione converte i caratteri speciali di una stringa trasformandoli nei corrispondenti codici HTML.

### Cosa sono i caratteri speciali?

Ad esempio i caratteri speciali <, >, ", & potrebbero causare problemi durante l'esecuzione dello  script, in particolar modo se la stringa viene salvata su un database  SQL.

La funzione htmlspecialchars li converte nei codici &lt;, &gt;, &quot;, &amp;. In quest'ultima forma i caratteri non causano nessun problema.

|   |        |
|---|--------|
| < | &lt;   |
| > | &gt;   |
| " | &quot; |
| & | &amp;  |

Inoltre, quando sono visualizzati sul browser dell'utente il risultato finale è sempre lo stesso.

### Un esempio pratico

In questo script assegno a una variabile una stringa contenente dei caratteri speciali.

```
<?
$stringa = "<testo di prova>";
$stringa = htmlspecialchars($stringa);
echo $stringa;
?>
```

Nella seconda riga sostituisco i caratteri speciali della stringa ( < e > ) con i codici Html corrispondenti (&lt; al posto di < e &gt; al posto di >).

```
<texto di prova>
```

Quando lo script visualizza a video la stringa, il risultato finale è sempre lo stesso.

```
<texto di prova>
```

Così facendo la stringa è interpretata da tutti i browser, indipendentemente dalla lingua e dal font di caratteri utilizzati dal computer client.

### **La funzione htmlspecialchars()**

Per omogeneizzare le stringhe è utile anche la funzione htmlspecialchars().

Quest'ultima è molto più potente rispetto alla Htmlspecialchars, perché converte tutti i caratteri che hanno un equivalente Html.

PREPARAZIONE COMPITO E CONCLUSIONI 10-12-2024

## LAB 13-12-2024 - COME FAR COMUNICARE JS & PHP ATTRAVERSO DEI FILE JSON

L'idea è di rifare alcune pagine: Home, Login e Contatti, andando a recuperare i dati (tramite delle chiamate asincrone HTTP) dal server in formato JSON e poi questi dati li andiamo a visualizzare con JS. Come facciamo a fare queste richieste asincrone? Utilizzando la funzione "fetch()", ossia una API nativa di JS per far richieste HTTP e storicamente sin dalle prime versioni del linguaggio è stato possibile fare queste richieste, inizialmente si usava l'oggetto XMLHttpRequest che basava lo scambio con dati XML. Anche in AJAX, la X sta per XML ma poi JSON è diventato lo standard de facto e lo scambio adesso è tutto basato su JSON.

Come funziona fetch()? È basato sulle "promise". (Programmazione asincrona). Quando facevamo la gestione degli eventi sulla pagina associevamo ad un elemento un listener in ascolto su di un evento, al cui verificarsi veniva chiamata una funzione di call-back (pattern observer). L'idea qui è simile: io faccio una richiesta al server, siccome la richiesta è asincrona, cioè io mando la richiesta ma non so quando ho la risposta io predispongo una funzione di call-back che viene chiamata quando viene restituito il risultato, un pò come facevamo con il click. Le promise sono un meccanismo per gestire queste chiamate asincrone e si basa su due funzioni principali che sono "then()" e "catch()". Qual è l'idea? Che io faccio una richiesta HTTP alla risorsa con l'istruzione fetch() ed un URL come parametro, l'istruzione che è in then() non viene eseguita subito ma quando il risultato viene restituito quindi questo porta a tutta una serie di side-effects, ovvero:

FINO AL MINUTO 16 TEORIA.