UD02 - Conceptes bàsics

Índex

CSS - Conceptes Bàsics

CSS - Cascade Style Sheets

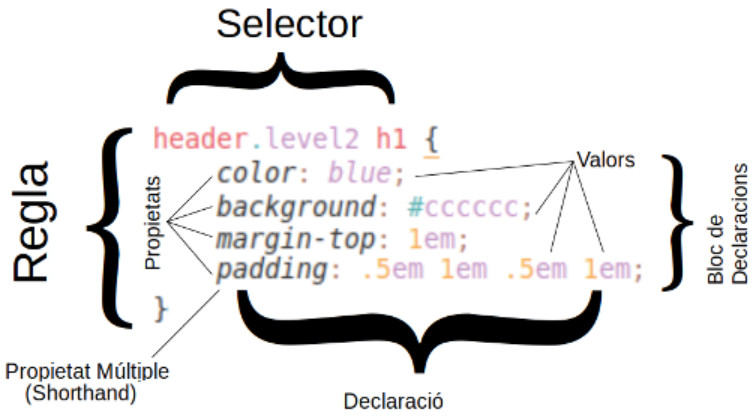
Nivells d'estandardització:

1. Editor's Draft (ED)
2. Working Draft (WD)
3. Transition – Last Call Working Draft (LCWD)
4. Candidate Recommendation (CR)
5. Transition – Proposed Recommendations (PR)
6. Recommendation (REC)

Font: https://css-tricks.com/css-standards-process/

Nomenclatura

- **Propietat:** Designa un paràmetre o característica que desitjam ajustar a l'element o elements seleccionats.
- **Valor:** El valor que li volem assignar a una *propietat*.
- **Declaració:** És el conjunt d'una *propietat* i el seu *valor*, separades per dos punts (":").
- **Bloc de Declaracions:** És una llista d'una o més *declaracions*, separades per punt i coma (";"), i envoltes en claus ("{ ... }").



- **Selector:** És una expressió que ens permet seleccionar un o varis elements del DOM als que volem aplicar un *bloc de declaracions*.
- **Regla:** Es el conjunt format per un *selector* i un *bloc de declaracions*.
  - També pot tenir varis selectors separats per comes (",").
- **"Multipropietats" (shorthand property):**
  - Ex: `font` = `font-style` + `font-weight` + `font-size` + `line-height` + `font-family`.
  - Ex: `margin` = `margin-top` + `margin-right` + `margin-bottom` + `margin-left`.
  - Acostumen guardar coherència:
  - Ex: Top - Right - Bottom - Left ("TRouBLe" / Des d'amunt i en sentit de les agulles del rellotge).
  - Valors per defecte "simètrics"
  - En cas de dubte, consultar documentació.
  - **Efecte col·lateral:** SEMPRE estableixen totes les propietats (encara que s'ometin).

Selectors

Selectors Bàsics

- `tagname`
- `.class`
- `#id`
- `\*`

Combinadors (Combinators)

Nom	Sintaxi		Exemple
Compound (composició)	(res)	`(a)(b)`	`h1.page-header`
Descendant (descendent)	(espai)	`(a) (b)`	`header h2`
Child (descendent directe)	(>)	`(a)>(b)`	`li>ul`
Adjacent Sibling **(posterior)**	(+)	`(a)+(b)`	`th+tr`
Following Sibling (el següent)	(~)	`(a)~(b)`	`tr.today~tr`

Selectors de Pseudo-Classe (Pseudo-class Selectors)

Els selectors de pseudo-classe ens permeten adreçar elements per alguna característica extrínseca a ells com ara la ubicació en relació a altres elements o determinats estats transitoris com ara si ténen el focus del teclat o si el cursor del ratolí es troba a sobre d'ells.

La llista completa de selectors de pseudo-classe es pot trobar al MDN sota:

- https://developer.mozilla.org/en-US/docs/Web/CSS/Pseudo-classes.

Selectors de Pseudo-classe Genèrics

`:first-child`	`:first-of-type`
`:last-child`	`:last-of-type`
`:only-child`	`:only-of-type`
`:nth-child(an+b)`	`:nth-of-type(an+b)`
`:nth-last-child(an+b)`	`:nth-last-of-type(an+b)`
`:not(<selector>)`	`:root`
`:focus`	`:hover`
`:empty`	`":blank"` (proposta)

Notes:

- `:root` sel·ecciona el tag `html` en documents HTML.
- `:empty` no pot contenir ni un espaï en blanc o salt de línia.
- `":blank"` de moment només és una proposta: no suportat per cap navegador a dia d'avui.

Selectors de Pseudo-classe Específics per a Formularis

Alguns selectors de pseudo-classe es refereixen específicament a característiques de camps de formularis (tags `<input>`, `<textarea>`, `<button>`, `<select>` i/o `<option>`, segons el cas):

`:enabled`	`:disabled`
`:valid`	`:invalid`
`:required`	`:optional`
`:checked`	

Avís: Alguns d'ells han estat introduïts o redefinits recentment pel que no funcionaran com s'espera en alguns navegadors.

Haurem de consultar cadascun a http://caniuse.com per a saber exactament com estan de suportats als distints navegadors.

Selectors de Pseudo-element

Els pseudo-elements, a diferència de les pseudo-classes, no adrecen elements (tags) realment existents al document, sinó que *virtualment* en creen de nous en un lloc determinat o sobre una part determinada de l'element seleccionat.

Per exemple, `p::first-letter` ens permetria aplicar un estil distint **únicament** a la primera lletra de cada paràgraf com si aquesta estigués continguda en el seu propi *subtag*.

Es distingeixen de les pseudo-classes per anar prefixats per dos caràcters de "dos punts" (`::`) en comptes d'un sol com els pseudo-elements.

Avís: Ens podem trobar però casos en que s'hagin especificat amb només ':' ja que així fou com s'especificaren originalment i per ara tots els navegadors els reconeixen també. Nosaltres **sempre** els escriurem com a `::after` i no `:after`, per exemple. I si detectam codi antic o algun cas en que, per error, hem fet servir la versió incorrecta, la rectificarem.

::before i ::after

Creen un pseudo-element que passa a ser el primer o darrer fill, respectivament, de l'element seleccionat.

- Es poden utilitzar per pre/su-fixar texts o imatges.
- **És obligatori** definir la propietat `content`, encara que sigui amb un espaï en blanc, ja que **del contrari no apareixeran**.

::first-letter i ::first-line

Permeten adreçar, respectivament, la primera lletra o la primera línia de text, d'un element.

::selection

Permet adreçar text (i sub-elements) que estiguin sel·leccionats per l'usuari.

Només permet unes poques propietats. Incloses `color`, `background-color`, `cursor`, and `text-decoration`.

Selectors d'Atribut

Els selectors d'atribut permeten seleccionar elements en funció dels seus atributs HTML i ténen la mateixa especificitat que els selectors de classe (0,1,0).

- `[attr]`: Existència de l'atribut (l'elements tenen l'atribut especificat, independentment del seu valor).
  - Exemple: `input[disabled]`
- `[attr="value"]`: Concordancia de valor.
  - Exemple: `input[type="button"]`
- `[attr^="value"]`: Concordancia inicial (que el valor "comença per...")
  - Exemple: `a[href^="https"]`
- `[attr$="value"]`: Concordancia final (que el valor "acaba en...")
  - Exemple: `a[href$=".pdf"]`
- `[attr*="value"]`: Concordancia parcial (l'atribut conté el text especificat).
  - Exemple: `*[class*="sprite-"]`.
- `[attr~="value"]`: Concordancia d'element en llista (d'elements separats per espaï)
  - Exemple: `div[class="menu"]` (en aquest cas equivaldria a `div.menu`, però no esta limitat a l'atribut "class").
- `[attr|="value"]` Igualtat o concordància de prefix (prefix = fins al primer "-").
  - Exemple: `div[lang|="es"]` (concordaria amb `lang="es"`, `lang="es-ca"`, etc..)

Selectors d'Atribut *Case-insensitive*

Els selectors d'atribut *tradicionalment* distingeixen entre majúscules i minúscules, pel que, per exemple, `a[href^="mailto://"]` **no seleccionaria, per exemple `<A HREF="MAILTO://someone@example.com>Email me</A>`**.

Recentment s'ha especificat un modificador que ens permet convertir els selectors anteriors en *Case Insensitive* només afegint una "i" al final.

Per exemple: `a[href^="mailto://" i]` sí seleccionaria l'enllaç anterior.

AVÍS: Encara hi ha molts de navegadors que no uporten el modificador *i*. Per aquest motiu és millor procurar no dependre'n mentres ens sigui possible (en el cas de l'exemple anterior, procurant normalitzar tots els enllaços a minúscules (o majúscules).

Autoprefixer

Com ja hem comentat, no tots els selectors i propietats descrits anteriorment estaran suportats als navegadors antics.

Molts d'ells però, ja els suportaven (de vegades no al 100% conformes amb l'especificació definitiva) amb el que s'anomenen *vendor prefixes*.

Els *vendor prefixes* son prefixs tipus "-webkit-" (Chrome, Safari...), "-moz-" (Firefox), etc... que posats al davant d'una propietat (o també alguns valors), componen un nom específic de la propietat que aquell navegador reconeixia, a títol experimental, abans de que dita característica fos definitivament estandarditzada.

Per tant, repetint les declaracions afectades amb els *vendor prefixes* de cada navegador, podem ampliar el nombre de navegadors en els que la nostra pàgina funcionarà bé.

El problema és que fer això a mà, a més d'una *feina de xinos*, empitjoraria greument la legibilitat del nostre CSS.

L'*autoprefixer* (podeu trobar l'enllaç a les referències) és un preprocessador que agafa el nostre full d'estil original i hi afegeix automàticament totes aquestes declaracions addicionals per nosaltres.

...aconseguint així que el nostre CSS funcioni bé en més navegadors.

Exercici 2

Donat el següent markup (Pug):

```
div.menu
  label Opcions
  ul
    li
      a(href="#") Opció 1
    li
      a(href="#") Opció 2
    li
      a(href="#") Opció 3
    li
      a(href="#") Opció 4
```

Implementar el CSS necessari per convertir-lo en un "popup". Això és:

- La llista (ul) ha d'estar oculta inicialment.
- En passar el ratolí per sobre de l'etiqueta, ha d'aparèixer la llista.
- Si el ratolí surt de l'àrea del menú aquest es torna ocultar.
- Quan el ratolí passa per sobre d'una de les opcions, aquesta es ressalta amb un color de fons distint.

Provau a afegir contingut després del menú (per exemple un paràgraf de text). Què passa en desplegar el menú?

Cascada

Com hem vist fins ara, els *selectors* ens permeten seleccionar elements segons diverses característiques: el *tipus* (tag), el seu *Id* (si el ténen), les *Classes* que puguin tenir atribuïdes, etc...

No és difícil però que dues o més *Regles CSS* puguin resultar d'aplicació a un mateix element tot i tenir selectors distints.

Exemple:

```
<style>
  h1 {...}
  .title {...}
<style>
<h1 class="title">Títol</title>
/* Les dues regles anteriors son d'aplicació a n'aquest tag
```

Això d'entrada no és problema a no ser que les dues regles continguin declaracions conflictives...

Exemple:

```
<style>
  h1 {color: blue;}
  .title {color: green;}
<style>
<h1 class="title">Títol</title>
```

Les normes que, en cas de conflicte entre dues *declaracions*, determinen quina és la que s'aplica son el que anomenem *Cascada*. D'aquí el terme *"**Cascade** Style Sheet"*.

1. Origen del Full d'Estil
2. Especifitat del Selector
3. Ordre al Codi Font

Origen

L'Origen fa referència a la procedència de les regles d'estil.

Principalment tenim dos origens:

1. Els *Author Styles* que son els que definim nosaltres als fulls d'estil.
2. Els *User Agent Styles* que son els estils que el navegador du predefinits de sèrie i gràcies als quals, fins i tot sense cap full d'estil (d'autor), un document HTML es renderitza amb uns estils mínims:
   - Marges
   - Lletres més grosses als títols (h1, h2, h3...)
   - El color blau i el subratllat dels enllaços (a).
   - Etcètera...

Addicionalment, alguns navegadors permeten configurar un full d'estil personalitzat per a l'usuari. En tal cas tindriem un tercer origen (els *User Styles* o Estils d'Usuari) que **es situarien al mig dels altres dos**.

 Les declaracions especificades *inline* al propi *tag* HTML (mitjançant l'atribut *class*) s'apliquen directament a l'element i ténen preferència sobre qualsevol altra independentment del seu origen.

Exemple: `<h1 style="color:red;">`

Els estils del *User Agent* son els que ténen menys preferència de tots. Això ens permet canviar-los a gust.

...però també ens permet **no fer-ho** si els estils que proveeix el navegador per defecte son suficients per a nosaltres.

 Però els *User Agent Styles* poden variar d'un navegador a l'altre. El que **en alguns casos pot arribar a ser un problema:**

Una diferència mínima en el tamany (o tipus) de lletra, per exemple, pot fer que un text passi d'ocupar-nos una línia a dues.

Per evitar aquest problema, es sol incorporar un fill de ja predefinit amb els estils que volem **per a tots** els navegadors.

Si no volem fer-ho manualment, en podem fer servir un de ja fet, com normalyze.css.

!important

El modificador `!important` al final d'una *declaració* CSS ens permet botar-nos la preferència d'origen fent que la regla s'apliqui fins i tot si hi ha una altra regla amb més preferència que la contradiu.

El problema però és que si dues declaracions en conflicte ténen la clausula `!important` aleshores prevaldrà de nou la preferència d'origen (i la resta que veurem després) que puguin aplicar.

Una forma senzilla d'entendre-ho és pensar que en realitat tenim 6 possibles origens en comptes de 3:

1. **Author !important**
2. **(*User Styles*) !important**
3. **User Agent !important**
4. Author
5. (*User Styles*)
6. User Agent

Especificitat

Donades dues declaracions conflictives i del mateix origen s'aplica sempre la més específica.

EXEMPLE: Imaginem un extens manual de circulació on apareixen, no necessàriament en aquest ordre i pot ser múltiples vegades en distints capítols, les següents regles:

- Les bicicletes circularan per l'esquerra.
- Tots els vehicles circularan per la dreta.

...Si una bicicleta és un vehicle, per ón ha de circular una bicicleta?

Com veieu intuitivament el concepte és molt senzill: Els *selectors* de l'exemple serien "Les bicicletes" i "Tots els vehicles". I tots hem entès què ha de fer cadascun encara que les regles siguin contradictòries.

Quan es tracta de *Selectors CSS* però, la cosa es complica. Considerem per exemple el següent fragment:

```
<style>
    h1 {
      font-size: 16px;
    }
    .big {
      font-size: 18px
    }
</style>
<h1 class="big">Títol</h1>
```

...quin tamany de lletra s'aplicarà al text "Títol"?

Per poder decidir-ho necessitam unes regles precises que es puguin aplicar a qualsevol selector (pensau que un selector pot ser la composició de molts altres).

I per poder definir aquestes regles classificarem els selectors segons tres criteris:

- Nombre d'Identificadors.
- Nombre de Classes.
- Nombre de Tags.

Per representar aquesta classificació es sol fer servir la notació `(nI,nC,nT)` ón *nI* seria el nombre d'identificadors, *nC* el nombre de classes i *nT* el nombre de tags.

Les *Pseudo-Classes* i els *Pseudo-Elements* **ténen la mateixa especificitat que les *Classes*.** Pel que els comptarem també com a tals.

A la taula següent podem veure uns quants exemples de selectors i la classificació que els correspondria.

Selector	Nº d'IDs	Nº de Classes	Nº de Tags	Notació
html body header h1	0	0	4	0,0,4
section header.section-header h1	0	1	3	0,1,3
.page-header .title	0	2	0	0,2,0

#sidebar	1	0	0	1,0,0

Ordre

Finalment, quan l'Origen i l'Especificitat coincideixen, la declaració que apareixi **després** guanya.

Així:

- Si el nostre document té varis fulls d'estil (ja siguin externs o inline), les regles que apareixin als fulls d'estil inclosos més avall en el document, tendran preferència sobre les que apareixin als anteriors.
- Si dues regles en conflicte apareixen al mateix fitxer (o dins el mateix tag `<style>...</style>`), la que apareixi més al final tindrà preferència.

No és extrany tampoc especificar la mateixa propietat, fins i tot dins la mateixa regla. **Exemple:**

```
.menu { display: block; display: flex; }
```

...en aquest cas guanya **sempre** la segona declaració. Però en un navegador que no suporti *Flexbox*, aquesta seria ignorada, pel que la primera ens serveix per definir un valor per defecte quan el navegador no suporta quelcom que volem fer servir (*fallback*).

Herència

La *Cascada* ens permet definir les característiques d'elements específics mitjançant les *Regles CSS*.

En contraposició, l'*Herència* ens permet no haver-ho de fer per tots i cadascun dels elements d'un document.

Així, si definim la propietat *color* (color del text) per a un element, aquesta s'aplicarà a n'aquest element i, recursivament, a tots els sub-elements que aquest contingui fins que topem amb algún en que, una altra regla CSS ens ho canvii.

No totes les propietats s'hereten. Per exemple, si les propietats *width* i *height* s'heretessin, tots els elements tindrien per defecte les dimensions del séu contenidor (el que resultaria poc pràctic...)

 En general, només s'hereten aquelles propietats que *en bona lògica* normalment voldriem que s'heretessin:

- Les relacionades amb el text (*color*, *font*, *text-align*, ...).
- Algunes relacionades amb les llistes (*list-style..*, *..-type*, *..-position*, ...).
- Etcètera...

L'herència ens permet, per exemple, definir un estil de text (color, tipus de lletra, tamany, etc...) al `<body>` i que aquests s'apliquin consistentment a la resta del document excepte alla on nosaltres explícitament especifiquem quelcom diferent.

Variables

Una variable CSS es defineix mitjançant una *declaració* a la qual, el nom de la propietat comença per dos guions ("--").

Exemple:

```
--home_background_color: cyan;
```

Funcionen com una declaració normal i son heretades pels elements descendents amb la única diferència que no tenen cap efecte per si mateixes.

Per llegir-les fem servir la funció `var()`.

Exemple:

```
background-color: var(--home_background_color);
```


En realitat amb la funció `var()` podem llegir el valor de qualsevol propietat CSS.

Alguns *frameworks CSS*, com el SASS, també proveeixen la funcionalitat de definir variables que després tradueïxen als seus respectius valors a l'hora de generar el CSS.

L'avantatge de les variables CSS natives és que funcionen com a propietats CSS:

- Es defineixen com a *declaracions* dins una regla CSS i s'apliquen només als objectes que compleixin les condicions del *selector*.
- S'hereten als elements continguts.


Per exemple, si volem aplicar un valor globalment, podem fer servir el selector `:root`.

L'inconvenient de les variables CSS és que, en navegadors antics que no les suportin, faran fallar totes les declaracions que les facin servir. **Fins i tot declaracions en les que la propietat que es declara sí està suportada**.

Per exemple:

```
:root {
  --fons_1: #cccccc;
}


.someClass {
  background: var(--fons_1);
}
```

Davant aquest problema tenim tres opcions:

- La més òbvia és ignorar-ho si no tenim interés en suportar navegadors antics (tret d'Internet Explorer i uns poc més molt minoritaris, la majoria les suporta bé des de fa temps).
- La segona és no fer-les servir i, en tot cas, fer servir algún framework com SASS que proveeixi la funcionalitat de variables, tot i no tenir la mateixa funcionalitat.
- La tercera (i probablement la millor) és simplement tenir cura de proveïr *fallbacks* en els casos més importants:

Per exemple, en el fragment anterior, hauriem pogut fer:

```
.someClass {
  background: white;
  background: var(--fons_1);
}
```

D'aquesta manera podem continuar fent servir les variables per ajustar certs valors del full d'estil (o fins i tot proveïr a l'usuari la possibilitat de, per exemple, escollir entre distints perfils de color) però, a l'hora, garantint que els usuaris de navegadors més antics, com al menys podran visualitzar la pàgina en unes condicións mínimes d'usabilitat.

Valors CSS

Com hem vist abans, en CSS, una *declaració* es composa d'una *propietat* i un (o més) valors que assignam a ella.

Els valors que accepta cada propietat depenen específicament de quina sigui aquesta.

Si be hi ha tres valors especials que podem fer servir a qualsevol propietat. Aquests son:

- **initial:** A l'especificació de tota propietat CSS es defineix un valor "inicial" pel cas en que aquesta no sigui explícitament definida. Aquest valor de vegades fins i tot pot variar en funció del *Tag*. La paraula clau *initial* ens serveix per assignar explícitament aquest valor a una propietat (anul·lant així l'efecte d'altres valors obtinguts a través de la cascada o l'herència.
- **auto:** Significa que el valor real s'ajustarà de forma automàtica segons uns criteris preestablerts.
- **inherit:** Força que el valor sigui heretat del pare.

Unitats

En alguns casos, els valors poden tenir unitats associades.

Pel que fa a les propietats que expressen mesures (*height*, *width*, *padding*, *border-width*, *margin*, etcètera...) aquestes unitats poden ser *absolutes* o *relatives*.

Longituds absolutes

Unitat	Descripció
cm	Centímetres
mm	Milímetres
in	Polzades (1in = 96px = 2.54cm)
px *	"Pixels" (1px = 1/96th of 1in)
pt	Punts (1pt = 1/72 of 1in)
pc	Piques (1pc = 12 pt)

Els 'px' son en realitat una unitat aproximada i no es corresponen amb els pixels físics del dispositiu sino que van en funció de la definició d'aquest amb la finalitat que les mesures en 'px' siguin el més semblants possibles en el món real independentment del dispositiu de visualització.

Longituds relatives

Unitat	Descripció
em	Relatiu al tamany de la font de l'element (2em significa 2 vegades el tamany de font actual)
ex	Relatiu a l'altura del caràcter "x" (rarament usat)
ch	Relatiu a l'amplada del caràcter "0" (rarament usat)

rem	Com "em" però relatiu a la font de l'element arrel (:root)
vw	Relatiu a l'1% de l'amplada del *viewport*
vh	Relatiu a l'1% de l'altura del *viewport*
vmin	Relatiu a l'1% de la dimensió menor del *viewport*
vmax	Relatiu a l'1% de la dimensió major del *viewport*
%	Relatiu a l'element pare

Càlculs

La funció *calc()* ens permet realitzar operacions de càlcul senzilles:

- Suma (+)
- Resta (-)
- Multiplicació (*)
- Divisió (/)

Accepta tants paràmetres com vulguem i podem fer servir parèntesis ((i)) si ho necessitam.

 *calc()* és una mica sensible a l'espaïat. És recomanable sempre deixar un espaï en blanc a banda i banda dels operadors.

Combinada amb la funció *var()* per llegir tant variables com altres propietats ens permet especificar propietats en forma de càlculs sobre altres paràmetres (per exemple marges, paddings, etc...) en comptes d'haver de posar el resultat literal.

Exemple:

```
:root { --sidebar_width: 200px; }
.sidebar {
  width: var(--sidebar_width);
  ...
}
.contents {
  width: calc(100vw - var(--sidebar_width));
  ...
}
```

D'aquesta manera, si en el futur volem alterar qualsevol d'aquests paràmetres, basta modificar-lo a un lloc. Sense necessitat de recalcular tots els paràmetres que en depenen.

 A més, podem mesclar distintes unitats. Fins i tot *relatives* (com ara *vw*) amb *absolutes*. Cosa que no podem fer manualment perquè, per exemple, no podem conèixer a priori les dimensions del *viewport*).

Regles "AT" (@)

Les "Regles AT" son comandes especials, que es distingeixen per començar amb el caràcter '@' que serveixen per donar certes instruccions sobre el comportament del CSS.

Exemples:

- `@import` ens permet insertar un altre full css.
- `@media <condicions> { (css) }` ens permet definir un bloc de CSS que s'aplicarà només quan es compleixin determinades condicions.
  - És el que es coneix com a *media query*.

Sobre els *media querys* hi tornarem més endavant en parlar de *disseny responsiu*.

Però hi ha un *media query*, habitualment oblidat, del que sí en farem menció especial:

```
@media print {
  ...
}
```

`@media print` ens permet controlar què (i com) veurem quan imprimim la nostra pàgina.

Moltes vegades imprimim una pàgina web i ens surt infinitat d'informació absolutament inútil (menús de navegació, etc...) i el contingut real apareix de qualsevol manera relegat a un racó.

Amb `@media print` podem ocultar tots aquells elements que no desitjam que apareixin a la impressió així com reposicionar la resta o, fins i tot mostrar-ne d'altres que no volem que surtin en pantalla.

 Per no haver de rectificar tot el CSS, també podem fer servir `@media screen` sobre els blocs que dirèctament sapiguem que no ens interessen per res a la impressió.

També, la regla `@page` ens permet establir el tamany del paper i els marges. **Com a curiositat,** a les referències us deixo un enllaç on s'expliquen moltes més característiques d'`@page`. Però desgraciadament, llevat de les que acabo de mencionar, la resta només estan suportades en eines específiques per a la generació de llibres electrònics.

Referències

- Documentació HTML i CSS:
  - W3C: https://www.w3schools.com
  - Mozilla Documentation Project: https://developer.mozilla.org
- Autoprefixer: https://github.com/postcss/autoprefixer
  - (En realitat és un mòdul del preprocessador modular PostCSS: https://postcss.org/)
- Especificitat: https://developer.mozilla.org/en-US/docs/Web/CSS/Specificity
- Frameworks CSS:
  - SASS: https://sass-lang.com/
- Unitats CSS: https://www.w3schools.com/cssref/css_units.asp
- CanIUse: https://caniuse.com.
  - Ens permet saber com de suportada està una funcionalitat als múltiples navegadors que hi ha al mercat.
  - A més, de vegades també ens suggereix *fallbacks* per casos específics.
- Regles "AT": https://developer.mozilla.org/en-US/docs/Web/CSS/At-rule

Altres (Curiositats)

- Media print:
  - Designing For Print With CSS: https://www.smashingmagazine.com/2015/01/designing-for-print-with-css/
  - Maquetació avançada d'impressió
  - (Desgraciadament cap navegador suporta massa més que establir els marges).
  - Però si algun dia volem escriure un llibre o generar documents impresos des d'una aplicació amb tecnologia HTML5, ens podria ser de molta utilitat.

Javascript - jQuery

jQuery és una llibreria/framework molt utilitzada per manipular el DOM.

No s'ha de confondre amb *jQuery-UI* o *jQuery-Mobile* que son compilacions de widgets implementats en forma de plugins de *jQuery*.

jQuery ens permet manipular el DOM de forma més senzilla a la que ho fariem dirèctament amb les funcions nadives d'aquest (*document.getElementById()*, etc...)

Per incloure jQuery al nostre projecte tenim dues opcions:

1. Descarregar-lo i incloure'l com a script. Exemple:

```
<script src="jquery-3.3.1.min.js"></script>
```

2. Fer servir un *CDN* (Content Delivery Network) com per exemple el de Google:

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3
```

Introducció

Un cop carregat, podem accedir a *jQuery* a través de dues variables indistintament:

- *jQuery:* Més formal però rarament usat.
- *$:* El símbol del *dollar*, que és un dels pocs caràcters no alfanumèrics que és vàlid al nom d'una variable en Javascript.

Nosaltres utilitzarem sempre "$".

"$" és una funció Javascript i, en conseqüència, també un objecte amb múltiples mètodes que podem fer servir.

La gran potència de jQuery radica en que ens permet capturar elements del DOM **fent servir selectors CSS**.

Per exemple, amb:

```
let seccions = $("div.section");
```

...obtindriem tots els elements de tipus "div" que tenguin assignada la classe "section".

 La funció "$" també accepta un segón paràmetre que pot ser un element DOM o una instància jQuery sobre ell (el resultat d'una "captura" anterior).

En aquest cas, només es capturen els elements que estiguin continguts dins ell.

El que ens retorna no és pròpiament un Array com podriem pensar, sino un objecte *semblant* però que a l'hora és una intància del propi jQuery.

Així:

- `sections.length()` ens donaria el nombre total d'elements seleccionats.
- `sections[0]` ens retornaria el primer d'ells.

Però també:

- `sections.css({color: "red"})` ens permetria alterar qualsevol propietat css a tots ells.
- `sections.on("click", function(ev) {...})` assignariem una funció a l'event de "click" que s'executaria cada cop que aquest event ocorri en cadascun dels elements.

A les referències trobareu els enllaços a la documentació compltea de jQuery així com un senzill tutorial del portal *w3schools*.

Un exemple d'ús senzill podria ser el següent:

```
$(".menu").on("click", function(ev){
    // console.log(ev); // (Descomentar per inspeccionar l
    $(ev.target)
        .closest(".menu")
        .toggleClass("open")
    ;
});
```

...on fem el següent:

- Capturam tots els elements amb la classe "menu".
- Els assignam un manejador (callback) per a l'event de click.
- Quan l'event ocorre, obtenim l'element sobre el que ha passat (ev.target)
- ...l'embolicam amb `$(...)` per convertir-lo en un objecte jQuery.
- Després, amb el mètode `closest()` obtenim l'ancestre més pròxim amb la classe "menu" (ja que el click l'haurem fet sobre l'etiqueta o, si estava obert, sobre alguna de les opcions).
- Utilitzam el mètode `toggleClass()` per afegir o treure (alternativament) la classe "open" al contenidor del menú.

Notes:

- Capturant l'event sobre el menú sencer en comtes de només sobre l'etiqueta aconseguim que, en fer click sobre qualsevol de les opcions, el menú es tanqui (que és el comportament que normalment s'espera en un menú "pop-up").
- Alternativament podriem haver desat la captura del menú en una variable (`var $menu = $(".menu")`) i posteriorment applicar el mètode `toggleClass()` sobre aquesta variable.
  - Però d'aquesta manera només podriem tenir un únic menú ja que, en cas de tenir-ne més, en clicar sobre qualsevol d'ells s'activarien tots.

Exercici 3

Implementau un menú "popup" com el de l'exercici anterior. Però que en aquest cas calgui fer click sobre l'etiqueta per obrir-lo o tancar-lo.

Per aconseguir-ho podeu fer servir el fragment en jQuery de l'exemple anterior que commuta la classe "open".

Opcionalment:

Ajustau els colors de la pàgina i el menú fent servir variables css.

- Ajustau aquestes variables al selector `body`.
- Modificau el menú de manera que les opcions estableixin un valor a la propietat "data-theme" de l'element "body" (feu servir jQuery).
- Feu servir aquesta propietat al CSS per definir distints colors segons el tema seleccionat.

Referències

- jQuery: https://jquery.com/
  - Documentació https://api.jquery.com
  - Tutorial: https://www.w3schools.com/jquery/
- You might not need jQuery: http://youmightnotneedjquery.com/