UD03 - Posicionament i Ancoratge

Índex

Posicionament i ancoratge

Tipus de posicionament

La propietat `position` estableix la forma en que es posicionen els elements dins el document i pot agafar els següents valors:

static

- És el valor per defecte.
- L'element es posiciona d'acord al flux normal del document.
- Les propietats *top*, *right*, *bottom*, *left* i *z-index* no ténen efecte amb aquest tipus de posicionament.
- Es diu que els elements amb *posítion* = *static* **no estan posicionats**.

relative

- Com amb *static*, l'element es posiciona d'acord al flux normal del document.
- Les propietats *top*, *right*, *bottom* i *left* el desplacen de la seva posició natural **però les posicions dels altres elements no es veuen afectats**.
- Els valors de *z-index* distints de "auto" crearàn un nou "context d'amuntegament" o *stacking context*.

absolute

- L'element s'elimina del flux normal del document (ja no es crea un espaï per a ell).
- És posicionat, mitjançant les propietats *top*, *right*, *bottom* i *left*, en relació al seu ancestre posicionat més proper o, si no n'hi ha cap, relatiu al contenidor inicial.
- Els valors de *z-index* distints de "auto" crearàn un nou "context d'amuntegament" o *stacking context*.

fixed

- L'element s'elimina del flux normal del document (ja no es crea un espaï per a ell).
- És posicionat, mitjançant les propietats *top*, *right*, *bottom* i *left*, en relació al contenidor inicial.
- Sempre crea un nou *stacking context*.
- En documents impresos (@media print) apareix a totes les pàgines i sempre a la mateixa posició.

sticky

- De recent introducció (pot no funcionar en alguns navegadors).

- L'element es posiciona d'acord al flux normal del document. Però si el document (o un subcontenidor) es desplaça de manera que l'element anés a quedar ocult, es "desenganxa" quedant-se fitxat a un *offset* determinat.
- No funciona en elements en que la propietat *overflow* valgui *hidden* o *auto*.

Ancoratge dels elements

La propietat "display" especifica el comportament de l'element en relació als elements que l'envolten i als que conté.

El seu valor per defecte depèn del tipus d'element: Per exemple, per a un `div` és "block" mentre que per un `span` és "inline".

De fet aquests dos son els seus possibles valors més elementals:

- `inline`: L'element s'integra "en línia" amb els altres elements. Les seves dimensions s'adapten al contingut (els atributs *width* i *height* no ténen cap efecte).
- `block`: Mostra l'element com un bloc que comença en una nova línia i agafa tota l'amplada disponible i l'alçada necessària per allotjar el contingut a no ser que les fixem amb les propietats *width* i *height*, respectivament..

Addicionalment tenim un tercer valor que és una combinació dels dos anteriors:

- `inline-block`: Que flueix igual que els elements *inline* però permet fixar les seves dimensions (comportant-se, de portes endins, com un element de bloc).

N'hi ha bastantes més, com *flex* i *grid* de les que parlarem més endavant. Però aquestes son les més essencials i, juntament amb *flex* i *grid*, les úniques que manejarem en la majoria dels casos.

Model de Capes

Quan activam el posicionament (*position* distint de *static*), els elements es poden moure (*relative*) del lloc que ténen assignat o fins i tot deixar d'ocupar cap espaï pel que fa al flux normal del contingut (*absolute*, *fixed*...).

Això fa que puguin sol·lapar-se amb altres.

Per decidir quin element es dibuixa a sobre de quin altre tenim una regla molt senzilla: **El que apareix més avall al DOM es pinta davant**. És a dir: van "trepitjant-se" a mida que es renderitzen.

La propietat *z-index*

De vegades això no és suficient i necessitam poder definir quins elements cobriran quins...

Per això tenim la propietat *z-index*:

- El valor per defecte de la propietat *z-index* és "auto", que significa que els elements seguiran la regla anterior.
- Però *z-index* també pot agafar un valor numèric enter.
- Els elements amb z-index positíu o zero es situaran per sobre dels que ténen *z-index* "auto" i més al front com més elevat sigui el seu valor.
- Els elements amb z-index negatiu es situaran per davall fins i tot dels elements no posicionats i més avall com menor sigui el seu valor.

El *stacking context*

Sempre que assignam un valor distint de "auto" a la propietat "z-index" (o el tipus de posicionament és un dels que ja hem indicat que el creen sempre), es crearà un nou *Stacking Context* en relació a l'element.

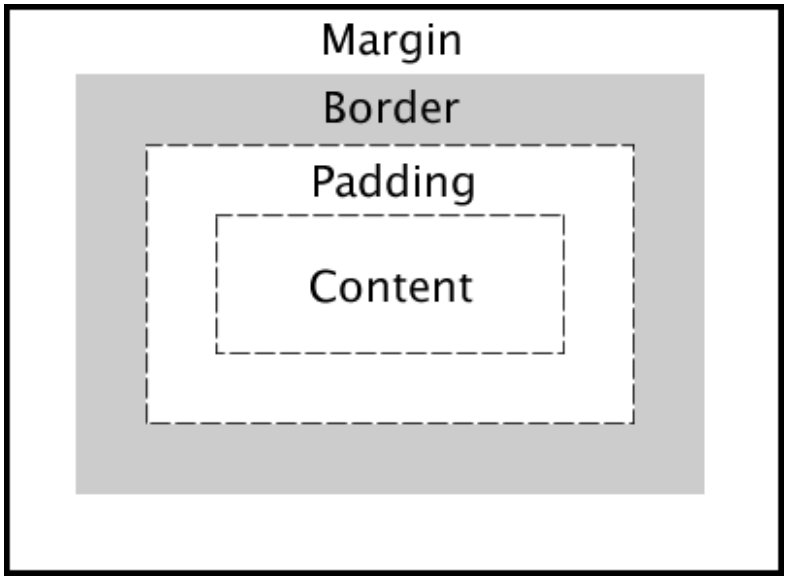
Això vol dir que tots els elements que contengui podràn intercalar-se entre ells segons les regles anteriors, però **cap element exterior podrà intercalar-se entre dos d'ells**.

Per aquest motiu no és recomanable abusar de la propietat *z-index*.

Model de Caixa (Box Model)

Tots els elements HTML es poden considerar "caixes".

Cadascuna d'aquestes caixes consta (de dins cap a fora) de 4 "sub-caixes": el propi contingut, el "padding" (farcit), el "border" (vorera) i el "margin" (marge).



Les propietats *padding*, *border* i *margin* ens permeten controlar les dimensions d'aquests.

Encara que el nom de les propietats sigui en singular, consideram que tenim 4 *paddings*, 4 *borders* i 4 *margins*. De fet, aquestes propietats anteriors son *shorthands* de *padding-top*, *padding-right*, *padding-bottom* i *padding-left*, i el mateix amb *margin-*, respectivament.

Voreres

La propietat *border*, en canvi, és el shorthand de:

- *border-width*
- *border-style* (requerit, per defecte "none")
- *border-color*

I només permet especificar tots a l'hora (si volem valors distints segons el costat haurem de fer servir *border-top*, *border-right*, etcètera... o dirèctament *border-top-width*, *border-top-style*, *border-top-color*...

border-style pot agafar entre d'altres els següents valors:

- **"none" (per defecte):** No hi ha voreres.
- **"solid":** Estableix un "border" sòlid (el més comú).
- **"dotted":** Puntejat.
- **"dashed:** Discontinu.
- ...
- **hidden:** Invisible (però ocupa l'espaï que li correspòn). Útil per a transicions (per exemple *:hover*) sense que les coses s'ens moguin de lloc.

Propietat `border-radius`

La propietat *border-radius* ens permet arrodonir els cantons de la caixa.

Com que "border-radius-top" seria ambigu, en aquest cas, **només tenim la propietat "border-radius" exclusivament**.

Així *border-radius* ens permet especificar tots els radis a l'hora, dos a dos, etc... Per saber en quin ordre els hem d'introduïr podem continuar fent servir la regla de seguir el sentit de les agulles del

rellotge amb només tombar el nostre "rellotge" 45º cap a l'esquerra (o "partir de les 10 i mitja en comptes de les 12).

Propietat `box-sizing`

La propietat *box-sizing* estableix què mesuren les propietats `height`, `width`, respectivament com també les seves derivades (`min-height`, `max-height`, ...).

Sense comptar `initial` (que en aquest cas equivaldria a `content-box` i `inherit`, la propietat `box-sizing` pot agafar dos valors:

- content-box.
- border-box.

Així, quan està establerta a `content-box` (per defecte), `height`, `width`, etc... defineixen el tamany del contingut. **Per això, si afegim voreres, marges o, fins i tot *padding*, L'ELEMENT ENS OCUPARÀ MÉS ESPAÏ del que hem especificat**.

En canvi, si l'establim a `border-box` aquesta mesura inclourà també el *padding* i les vores (*border*), fent molt més senzilla la tasca de sumar els espaïs que ens ocupa cada element a l'hora de maquetar.

 **Alerta:** Així i tot, encara haurem de tenir en compte els marges, si en tenim.

Establir el box-sizing de forma global (WIP)

Si ho preferim, podem establir el `box-sizing` a `border-box` de forma global amb una senzilla regla:

```
*,
::before,
::after {
  box-sizing: border-box;
}
```

 Però si estam treballant en un projecte preexistent, podria passar que tinguéssim codi css anterior que, al estar calculat segons el comportament per defecte, deixas de funcionar corrèctament.

El mateix ens pot passar si fem servir frameworks css externs que hagin estat dissenyats en base a `border-sizing: content-box`. Si be molts, com *Bootstrap* ja fan servir `border-box` i/o fixen explícitament aquesta propietat per al seu css.

En qualsevol cas, el següent exemple ens mostra una forma més refinada de fer-ho:

```
:root {
  box-sizing: border-box;
}


*,
::before,
::after {
  box-sizing: inherit;
}
```

Establint `box-sizing` a `inherit` obrim la possibilitat de canviar aquesta propietat recursivament de forma selectiva a les parts que ens interessi del DOM (per exemple `.oldCss {box-sizing: content-box;}`) mentre que la regla que hem definit sobre el selector `:root` ens garanteix que la resta heredarà el valor `border-box`.

Marges

Marges negatius

Una cosa que no tot-hom sap és que establir marges negatius és vàlid a CSS.

Si establim un valor negatiu als marges passarà el que tots intuïtivament pensam: el contingut **(més el padding)** sobresurtirà del contenidor. De manera que si el padding és menor que el valor absolut del marge negatiu, el contingut ens pot sortir fora.

El que passi aleshores depèn de la propietat *overflow*: si és "display" és sol·laparà amb els elements de fora del contenidor i si és "hidden", "scroll" o "auto" simplement quedarà ocult.

Marges colapsats

Els marges, com els *paddings* serveixen per deixar espaï en blanc, evitant així que el nostre disseny quedi sobrecarregat o fins i tot costi de llegir.

Però els marges concretament, serveixen per establir un mínim espaï de separació **entre dos elements**.

En canvi, a tots els elements els podem establir un marge. Així que entre dos elements donats **tenim dos valors distints a considerar**.

Com hem dit, els marges estableixen una separació **mínima** entre dos elements (no entre un element i la part exterior del marge de l'altre).

Per això, **els marges (normalment) no es sumen**, sino que s'agafa sempre el valor més gran dels elements a separar.

Quan passa això diem que **els marges han col·lapsat**.

Però no en tots els casos els marges col·lapsen...

Casos en que els marges no colapsen:

- Establint la propietat `overflow` a qualsevol valor distint de `visible` a un contenidor s'evita que els marges interiors colapsin amb els exteriors.
- Si hi ha una vorera (border) entre dos marges (per exemple transparent, si la volem fer servir com a truc per evitar el colapsament).
- Cap a fora d'un contenidor dels següents contenidors:
  - Flotant ("floated")
  - Amb la propietat `display` a `inline-block`.
  - Amb posicionament absolut (`position: absolute`) o fix (`position: fixed`)
- A dins d'un contenidor flexbox (`display: flex`) o grid (`display: grid`).
- Els elements amb `display: table-cell`, `table-row` i la majoria de la resta de tipus de display de taules a excepció de `table`, `table-inline` i `table-caption` no tenen marges, així que no poden colapsar.

Recursos

- Model de Caixa: https://www.w3schools.com/css/css_boxmodel.asp

Javascript - Components CSS

CSS Modular

Quan començam a escriure CSS és molt comú que anem escrivint les regles a mesura que les necessitam per a donar estil a un element concret.

De manara que acabam tenint un CSS infinit plè de regles super-específiques que, a sobre, acaben aplicant-se accidentalment a llocs pels que no havien estat pensades.

A més, la nostra pàgina acaba per no tenir consistència visual ja que els elements d'una secció i d'una altra han estat dissenyats per separat sense un criteri uniforme.

Pensar le nostre CSS de forma *modular* ens permet evitar aquest problema.

Per exemple, si necessitam posar un botó, podem assignar-li la classe "button" i estilitzar-lo de la següent manera:

```
.button {
  padding: 0.5em 0.8em;
  border: 1px solid #265559;
  border-radius: 0.2em;
  background-color: transparent;
  font-size: 1rem;
}
```

Fins aquí res de nou.

Però que passa quan necessitam un botó lleugerament distint? Per exemple un botó que denoti que quelcom ha anat bé.

Instintivament podriem crear-ne un de nou o copiar l'anterior, per exemple com a ".successButton" i fer les modificacions necessàries.

Això funcionaria i tindria certa coherència: Fins al dia que decidim canviar el tamany de ".button" que, si no ens recordam de fer-ho també a tots els seus "clons", acabarem tenint múltiples tipus de botons que no ténen res a veure els uns amb els altres.

Així que, en comptes d'això, definirem la següent regla:

```
.button--success {
  border-color: #cfe8c9;
  color: #fff;
  background-color: #2f5926;
}
```

Aquí definim **una variant** del botó anterior. Per això abans dels dos guions (--) hem fet servir el mateix nom de la classe original. I després indicam la variació que aquesta regla implementa.

Així, per crear un "Success Button" fariem el següent:

```
<span class="button button--success">Ok</span>
```

Fent servir la mateixa tècnica podem definir:

Botons de perill:

```
.button--danger {
  border-color: #e8c9c9;
  color: #fff;
  background-color: #a92323;
}
```

Botons més petits:

```
.button--small {
  font-size: 0.8rem;
}
```

Botons més grans:

```
.button--large {
  font-size: 1.2rem;
}
```

...i combinar-los a gust:

```
<span class="button button--danger button--large">D'acord!<
```

Components Javascript/jQuery + CSS

Ara imaginem un component una mica més complex.

Recordem l'Exercici 2 en que vam implementar un menú de tipus "popup".

Algú ha intentat fer-lo servir des del telèfon mòbil?

Que passa quan passau el rat... Oh!! Espereu!!!! ...no tenim ratolí!!! Com obrim el menú???

NO PODEM!!

I tampoc no cal anar tan enfora. Fins i tot en un PC, no sempre que el ratolí passi per sobre del títol del menú serà perquè voliem obrir aquest: A lo millor només hi hem passat perquè estava a mig camí d'un altre element que, ara que s'ens ha obert el menú a lo millor fins i tot ens el tapa!!.

El selector `:hover` va molt bé per crear *tooltips* més elaborats que una simple propietat "title". Pero per a obrir un menú rarament serà una bona idea des del punt de vista d'usabilitat.

Normalment per obrir o tancar un menú, es fa servir l'event de "click". Però per això necessitarem javascript.

Exercici 4

Recuperau el menú de l'exercici anterior al que ja resoliem aquest problema canviant la Pseudo-Classe *:hover* per una Classe normal i afegiem i treiem aquesta cada cop que l'usuari clicava sobre el títol del menú.

- Acabau-lo si no el teniu acabat.
- Què passava al contingut de davall del menú quan obriem aquest?
  - Ara, amb el que heu après avui sobre posicionament CSS teniu les eines per a corregir-ho.
- Revisau el codi. Podriem considerar-lo un *component CSS*?
- Si la classe principal és prou específica, la part de *Javascript/jQuery* també la podrem considerar part del component.
- **Opcional:** Modificau el menú afegint un atribut "action" que especifiqui una acció. Seguidament modificau el controlador javascript per tal que quan seleccionem una opció, ens mostri un "alert()" amb el seu contingut i tanqui el menú.
  - Addicionalment podeu canviar aquest "alert()" per disparar un event sobre el document i implementar la seva captura.
