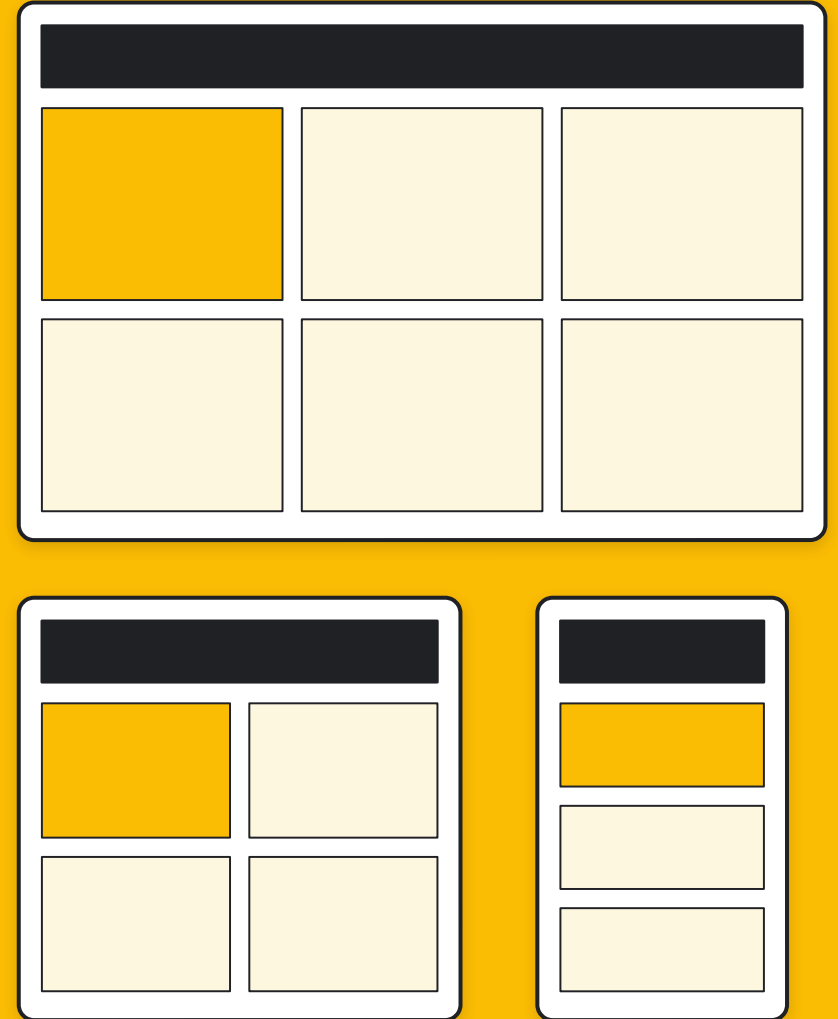


CSS3 Moderno, Layout (Flexbox/Grid), Container Queries y Responsive

RA2 · Crea interfaces web homogéneas definiendo y aplicando estilos

- Entender quién gana cuando dos reglas chocan (cascada y especificidad)
- Maquetar con Flexbox (una dimensión) y Grid (dos dimensiones)
- Adaptar la página y sus componentes a cualquier pantalla





DÓNDE ESTAMOS

AURIA ya tiene esqueleto: hoy le ponemos la piel



La pregunta de la unidad: ¿quién decide cómo se ve y dónde va cada cosa, en cualquier pantalla?

Partimos del HTML semántico de la UD3 y de la carpeta scss/ que dejamos preparada. El HTML casi no se toca.



¿Por qué esto importa en el mundo real?

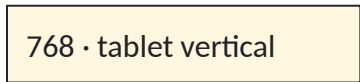
Anchos de pantalla reales (CSS px)



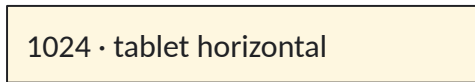
360 · móvil pequeño



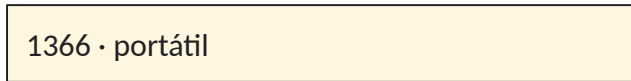
390 · móvil



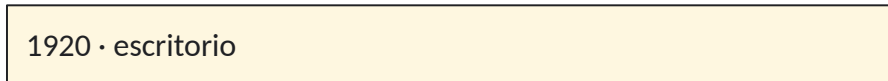
768 · tablet vertical



1024 · tablet horizontal



1366 · portátil



1920 · escritorio

...y todos los anchos intermedios.



Más de la mitad del tráfico es móvil

Una web que solo funciona en escritorio pierde a la mayoría



Google indexa la versión móvil

Mobile-first indexing: lo que no está en móvil, no cuenta



Flexbox y Grid son el estándar

Ya no se maqueta con float ni con tablas



Puesto de trabajo real

Todo proyecto trae requisitos responsive desde el día 1



Planificar sale barato

Parchear el responsive al final cuesta mucho más



DATO CLAVE — No se diseña «para el iPhone»: se diseña para un rango continuo de anchos.



Tres preguntas, en este orden

1 • ¿QUIÉN GANA?



Cascada y
especificidad



Modelo de caja



2 • ¿DÓNDE VA CADA COSA?



Flexbox



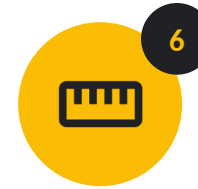
Grid



3 • ¿Y EN OTRA PANTALLA?



Media
queries



Unidades
fluidas



Container
queries



Verificar

DevTools: Styles, Computed, overlays y
modo dispositivo



Laboratorio

La página de AURIA, de fea a responsive



Extra

Modo oscuro y hoja de impresión



CLAVE — Sin entender la 1 no se entiende la 2, y sin la 2 no hay 3. No nos saltamos pasos.



BLOQUE 1

Cómo llega el CSS y quién gana

Formas de aplicar estilos, cascada, especificidad, herencia y modelo de caja



Tres formas de aplicar CSS (RA2.b, RA2.c)

En línea

```
<p style="color:red">
```

Máxima prioridad, imposible de reutilizar

Casi nunca

Interno

```
<style> ... </style>
```

Solo sirve para esa página

Prototipos rápidos

Externo

```
<link rel="stylesheet"
href="css/main.css">
```

Se cachea y se reutiliza en todas las páginas

Siempre en proyectos reales

En AURIA: escribimos en scss/ → compilamos → enlazamos css/main.css con <link>. Un solo fichero externo.



NOTA — Si ves style="..." en un HTML profesional, casi siempre lo ha puesto JavaScript, no una persona.



La cascada: tres filtros para decidir quién gana

1

Origen e importancia

navegador < autor < !important



2

Especificidad

¿qué selector es más concreto?



3

Orden de aparición

si empatan, gana la última



color: blue; ← la que se aplica

main.css

```
p { color: gray; }  
  
.intro { color: blue; }  
  
p { color: red; }
```

`<p class="intro">` → ¿de qué color?

- Filtro 1: las tres son del autor → empate
- Filtro 2: .intro (clase) gana a p (elemento)
- El orden ni se mira: azul



CLAVE — El orden solo decide cuando hay empate de especificidad. Es el error de concepto más frecuente.



La especificidad es un marcador de tres cifras

ID	CLASE • ATRIBUTO • PSEUDOCASE	ELEMENTO • PSEUDOELEMENTO		
#cabecera	.btn [type] :hover	p div ::before	Selector	Marcador
0	0	1	div	(0, 0, 1)
0	1	0	.btn	(0, 1, 0)
0	1	2	nav a:hover	(0, 1, 2)
1	1	0	#nav .btn	(1, 1, 0)
1	2	1	div#nav .btn:hover	(1, 2, 1)



REGLA — Se compara cifra a cifra, de izquierda a derecha, como en un marcador: (1,0,0) gana a (0,12,0). style="" gana a todos; !important, incluso a style.



Ejercicio: ¿de qué color es el botón de AURIA?

auria.html

```
<button id="comprar" class="btn btn--primary">Añadir al carrito</button>
```

```
button { background: gray; }
```



(0, 0, 1)

```
.btn.btn--primary { background: #1E88E5; }
```



(0, 2, 0)

```
#comprar { background: green; }
```



(1, 0, 0)

GANA

```
.btn { background: orange; }
```



(0, 1, 0)



PIENSA — ¿Y si quito el id del HTML? Gana (0,2,0): el azul. ¿Y si .btn estuviera después? Da igual: (0,1,0) pierde.



!important: la escalada que nadie gana

```
style="color: pink !important"
```

Viernes: nadie sabe por qué es rosa

```
#x .btn { color: green !important; }
```

Miércoles: otro compañero

```
.btn { color: red !important; }
```

Martes: «no me hace caso»

```
.btn { color: blue; }
```

Lunes

Cuándo Sí es legítimo

- Clases de utilidad que deben ganar siempre (.oculto, .sr-only)
- Accesibilidad: prefers-reduced-motion
- Sobrescribir CSS de terceros que no controlas

La alternativa

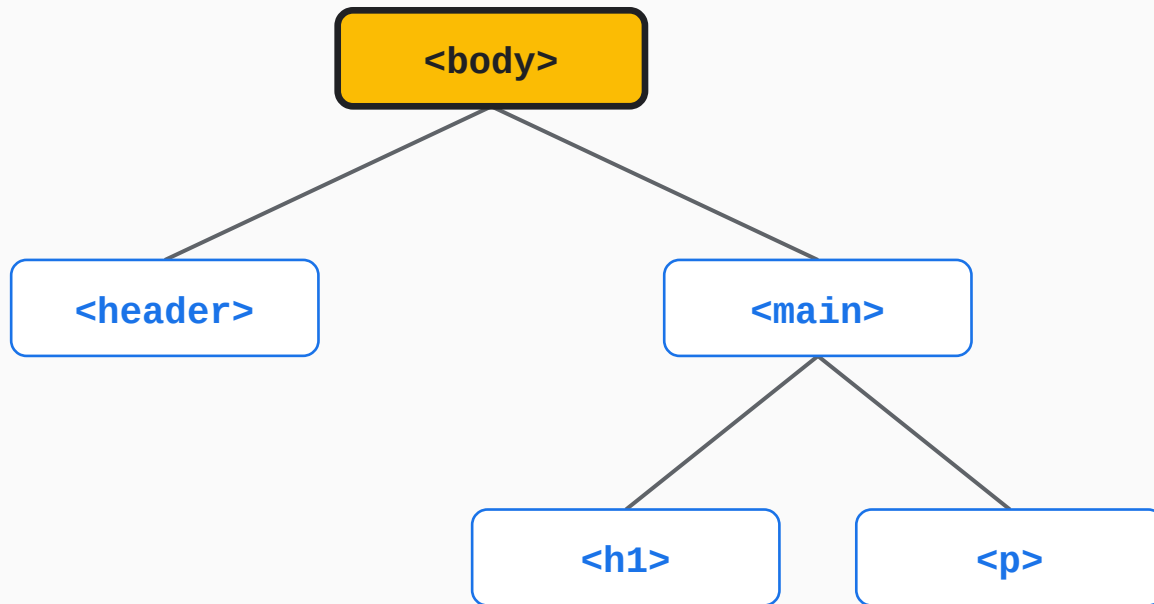
Selectores de una sola clase, todos con (0,1,0). Es la idea de BEM, que veremos en la UD5.



CLAVE — Cada !important obliga al siguiente a poner otro más fuerte. Es deuda técnica.



Herencia: lo que los hijos reciben sin pedirlo



_base.scss

```
body { color: #1a73e8; font-family: sans-serif;
border: 3px solid black; }
```

SÍ se heredan (texto)	NO se heredan (caja)
color	margin · padding
font-family · font-size	border
line-height · text-align	width · height
visibility · cursor	background · display · position

Palabras clave para forzarlo

inherit

copia el valor del padre

initial

vuelve al valor por defecto de CSS

unset

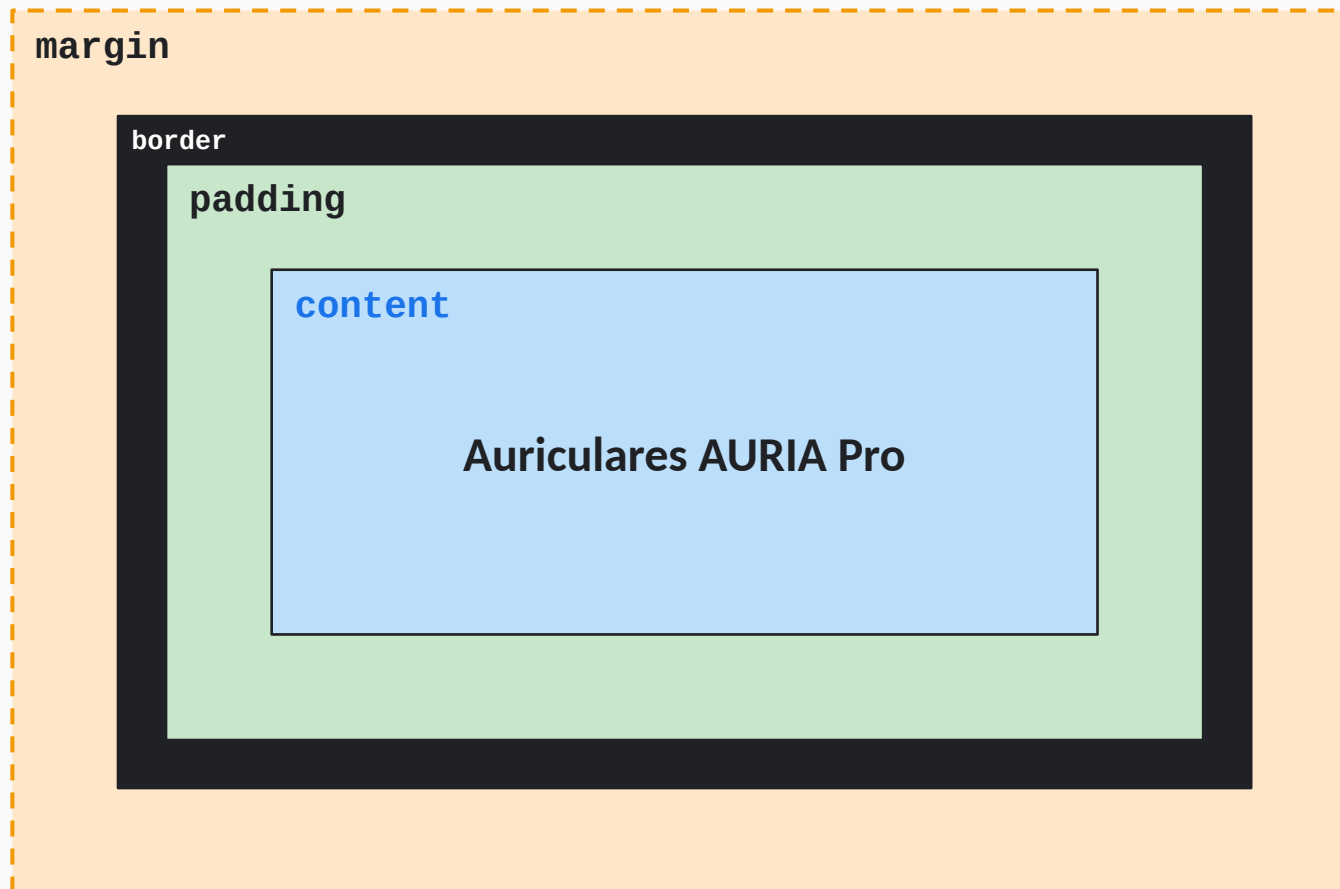
hereda si es heredable; si no, initial



TRUCO — Regla de memoria: lo que es de TEXTO se hereda; lo que es de la CAJA, no.



Todo en CSS es una caja con cuatro capas



margin

Separa la caja de las demás. No tiene color. Dos márgenes verticales seguidos se «funden» (colapso de márgenes).

border

Línea alrededor del padding. Tiene grosor, estilo y color.

padding

Separa el contenido del borde. Se pinta con el background.

content

Lo que mide width × height (por defecto).



DEMO — F12 → Elements → pestaña Computed: el dibujo de la caja con sus números aparece ahí, para cualquier elemento.

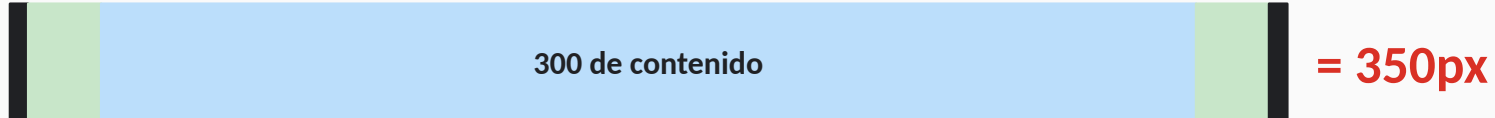


box-sizing: ¿cuánto mide de verdad una caja?

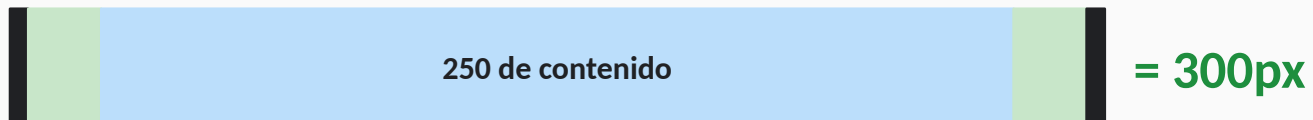
_card.scss

```
.card {  
  width: 300px; padding: 20px;  
  border: 5px solid; }
```

content-box (por defecto)



border-box (lo que queremos)



El reset de todo proyecto

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

Va al principio de _base.scss. Con él, width es el ancho real de la caja: sumar padding o borde ya no la hace crecer.

La cuenta de la izquierda

content-box: $300 + 2 \times 20 + 2 \times 5 = 350$
border-box: $250 + 2 \times 20 + 2 \times 5 = 300$



CLAVE — Con content-box, width solo mide el contenido: padding y borde se suman por fuera.



BLOQUE 2

Layout: Flexbox y Grid

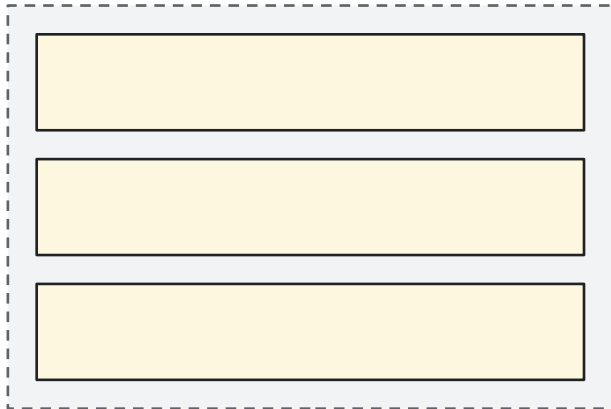
Colocar las cajas en una dimensión (Flexbox) o en dos (Grid)



Tres formas de colocar cajas

Flujo normal

`display: block`



Una debajo de otra. Es lo que tiene AURIA ahora.

Flexbox

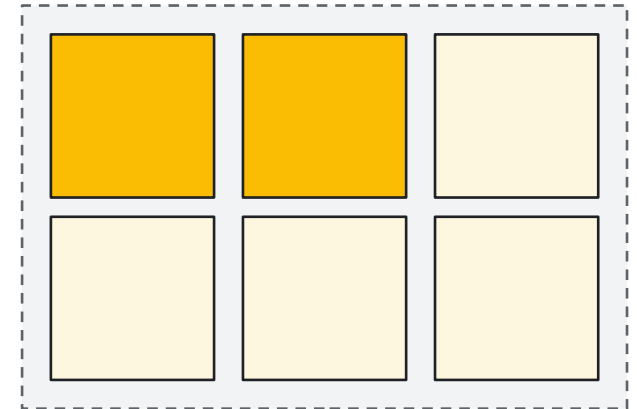
`display: flex`



En una fila O en una columna: 1 dimensión.

Grid

`display: grid`



En filas Y columnas a la vez: 2 dimensiones.

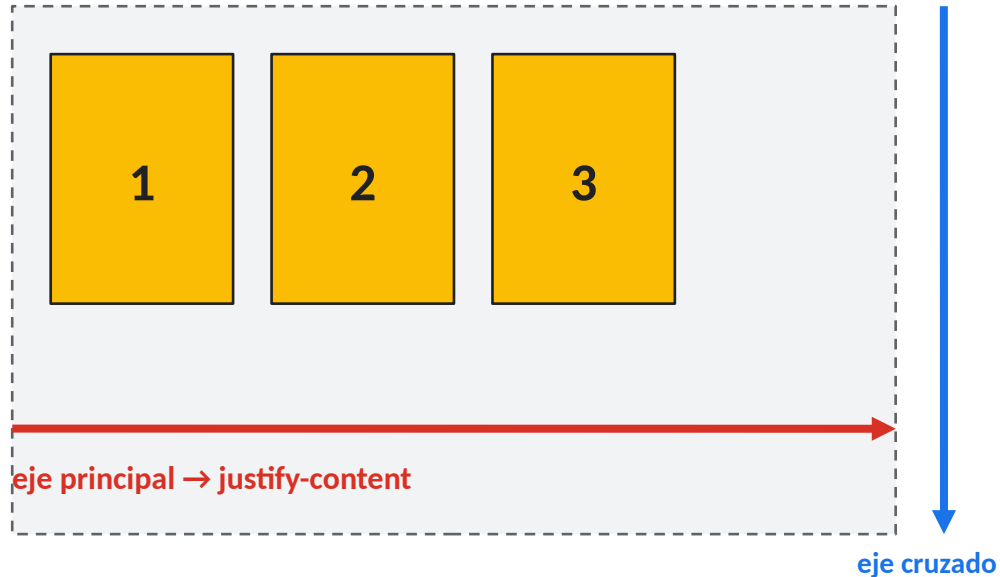


CLAVE — No compiten: Grid para el esqueleto de la página, Flexbox para colocar lo de dentro de cada pieza.

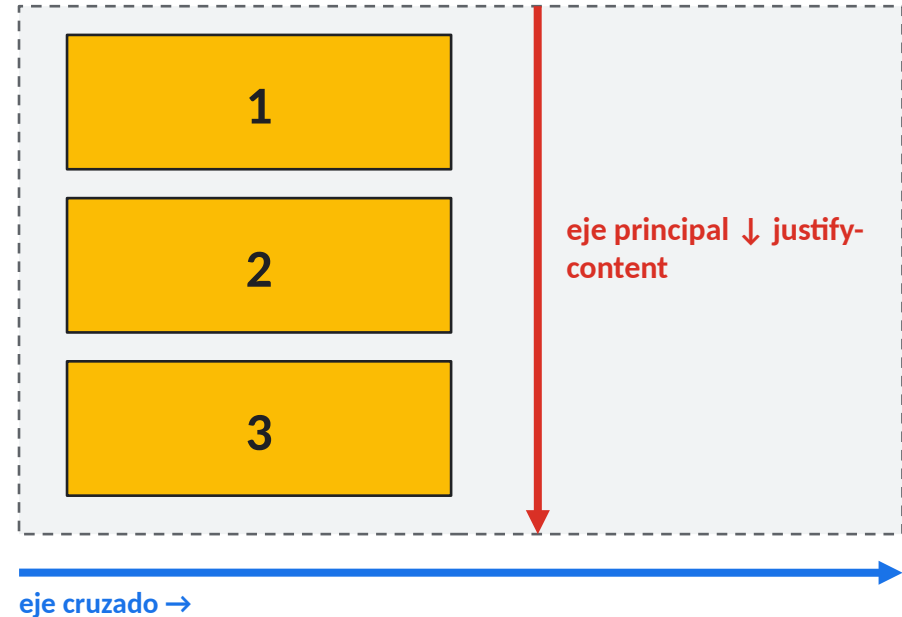


Flexbox: un eje principal y un eje cruzado

`flex-direction: row`



`flex-direction: column`



justify-content alinea en el eje principal. **align-items** alinea en el cruzado.



OJO — Si cambias flex-direction, los ejes se intercambian: justify-content pasa a ser vertical.



Las propiedades del contenedor, de un vistazo

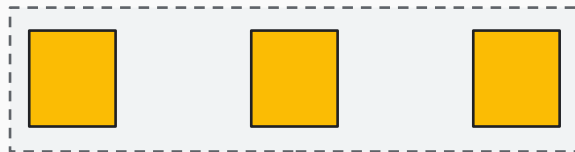
justify-content (eje principal)



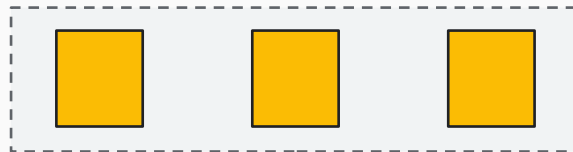
flex-start



center

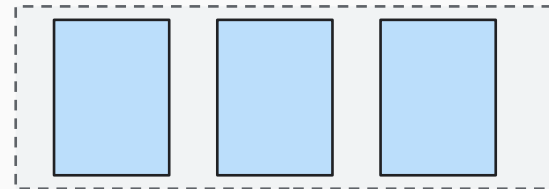


space-between

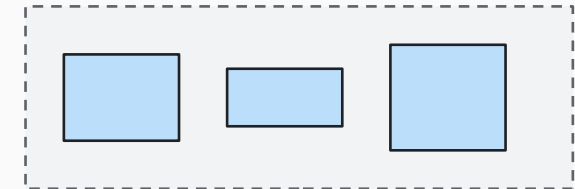


space-around

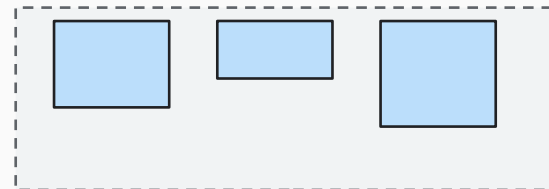
align-items (eje cruzado)



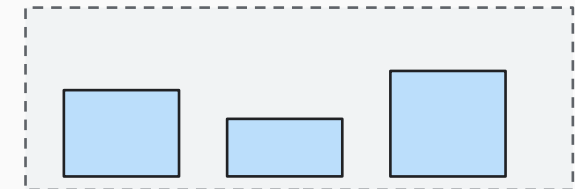
stretch



center



flex-start



flex-end

gap: 16px

Espacio entre hijos, sin márgenes

flex-wrap: wrap

Si no caben, bajan a otra línea

display: flex

Solo afecta a los hijos DIRECTOS

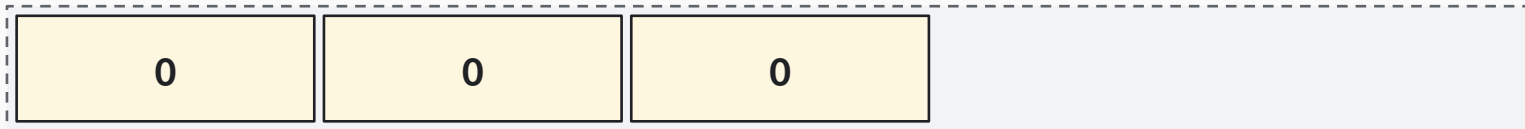


DEMO — flexboxfroggy.com: 24 niveles para practicar justify-content y align-items jugando.



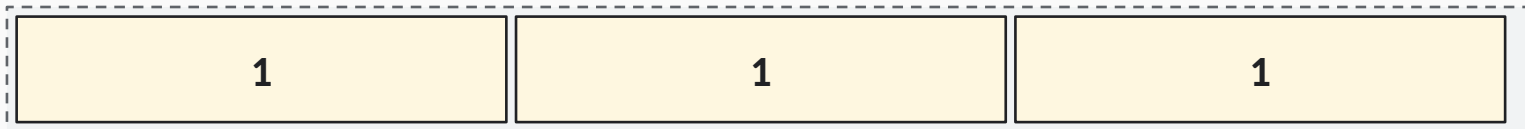
Los hijos: cómo se reparten el espacio que sobra

flex-grow: 0 (por defecto)



No crecen: sobra espacio a la derecha

flex-grow: 1 en todos



Se reparten el sobrante a partes iguales

flex-grow: 2 en el del medio



El del medio se lleva el doble del sobrante

Propiedad	Qué decide	Por defecto
flex-grow	Cuánto crece si SOBRA espacio	0
flex-shrink	Cuánto encoge si FALTA espacio	1
flex-basis	Tamaño de partida antes de repartir	auto

flex: 1 = flex: 1 1 0%
El atajo que más usarás: «ocupa lo que sobre».



Ya lo habéis usado: el Flex Layout de Penpot es CSS

Penpot • Flex Layout

Dirección

→ fila

Alinear

centro

Justificar

espacio entre

Gap

16

Padding

24

Inspect → Code



_card.scss

```
.card__fila {  
  display: flex;  
  flex-direction: row;  
  align-items: center;  
  justify-content: space-between;  
  gap: $space-md; // 16px · token UD2  
  padding: $space-lg; // 24px  
}
```

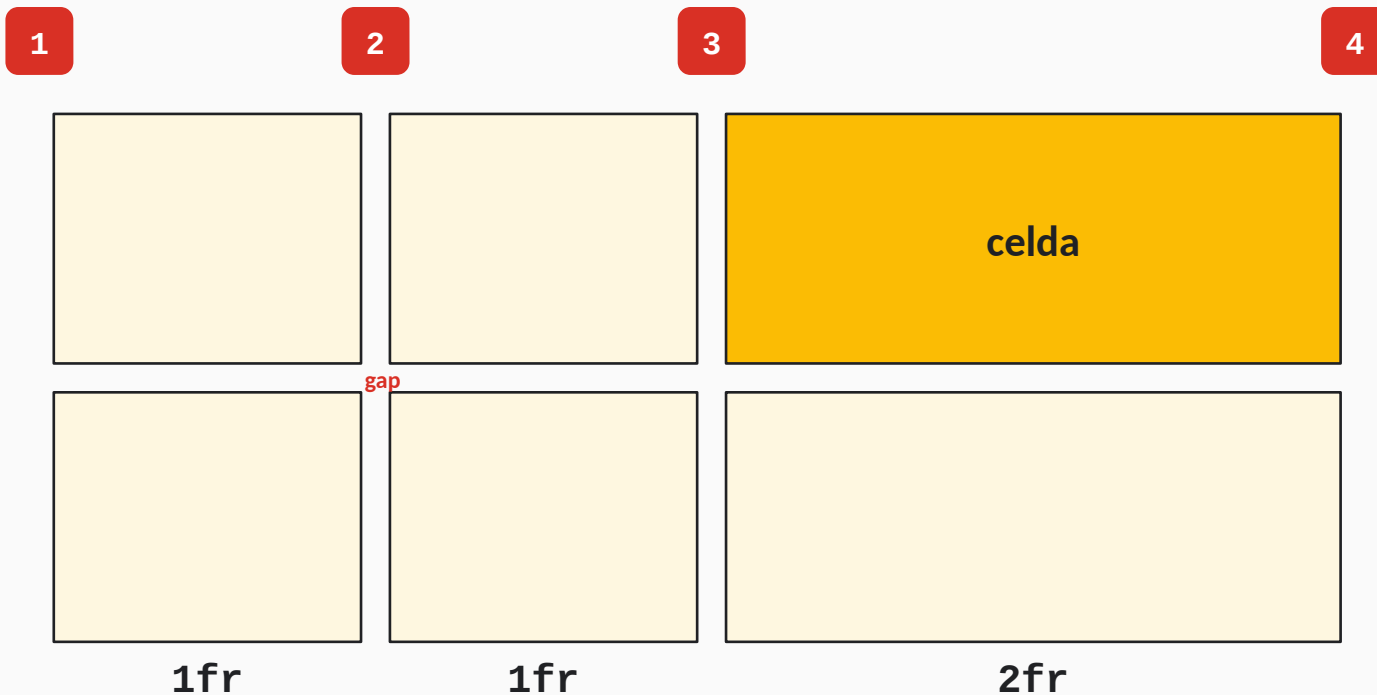
Penpot genera CSS real (Inspect → Code). Nosotros lo reescribimos con nuestras variables de SASS, para que los valores vengan de los tokens y no queden «mágicos».



CLAVE — Lo que en Penpot era un panel con despleables, en CSS son cinco propiedades. Mismo modelo mental.



Grid: líneas, pistas, celdas y la unidad fr



← los números rojos son las líneas de columna

```
_layout.scss

.rejilla {
  display: grid;
  grid-template-columns: 1fr 1fr 2fr;
  gap: 16px;
}
```

pista (track)

Una fila o una columna entera

fr

Fracción del espacio libre: 1fr 1fr 2fr = $\frac{1}{4}$ $\frac{1}{4}$ $\frac{1}{2}$

repeat(3, 1fr)

Atajo de 1fr 1fr 1fr

grid-column: 1 / 3

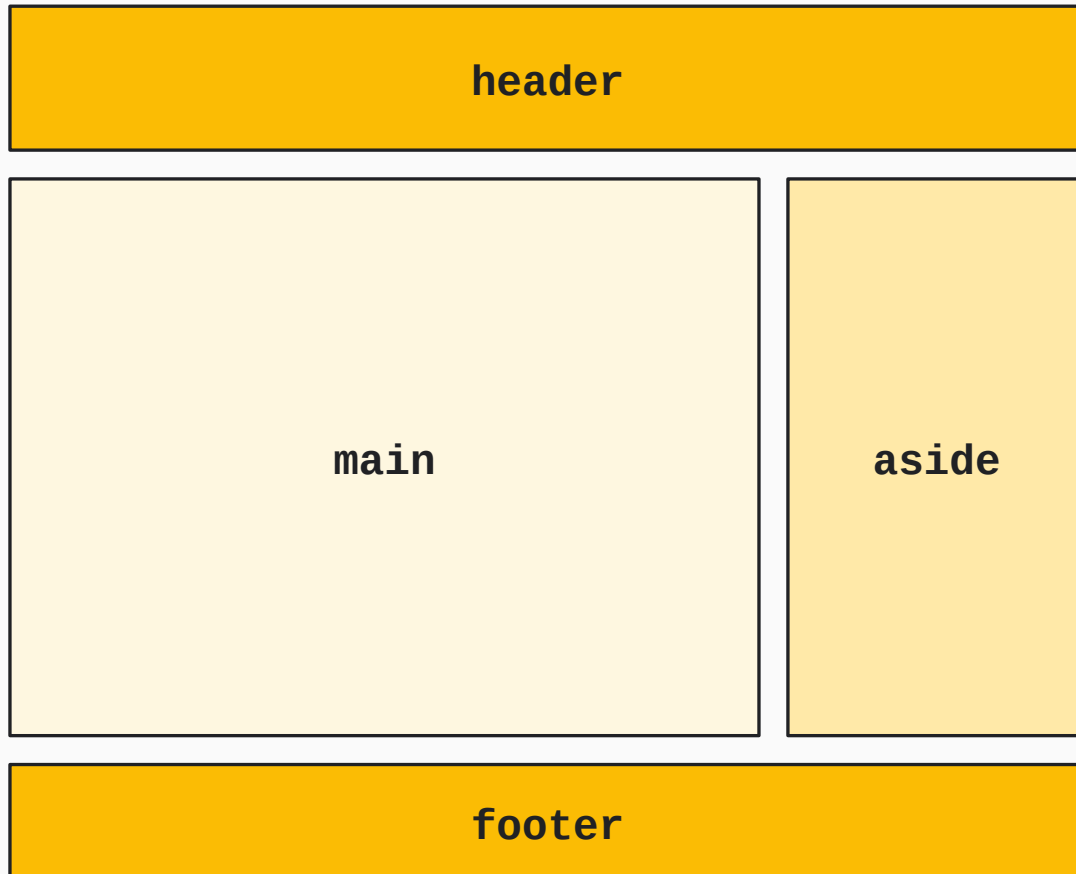
Ocupa de la línea 1 a la 3 (dos columnas)



DEMO — F12 → Elements → botón «grid» junto al elemento: dibuja las líneas numeradas encima de la página.



grid-template-areas: dibujar el layout con palabras



_layout.scss

```
.pagina {  
  display: grid;  
  grid-template-columns: 3fr 1fr;  
  grid-template-areas:  
    "header header"  
    "main aside"  
    "footer footer";  
}  
header { grid-area: header; }
```

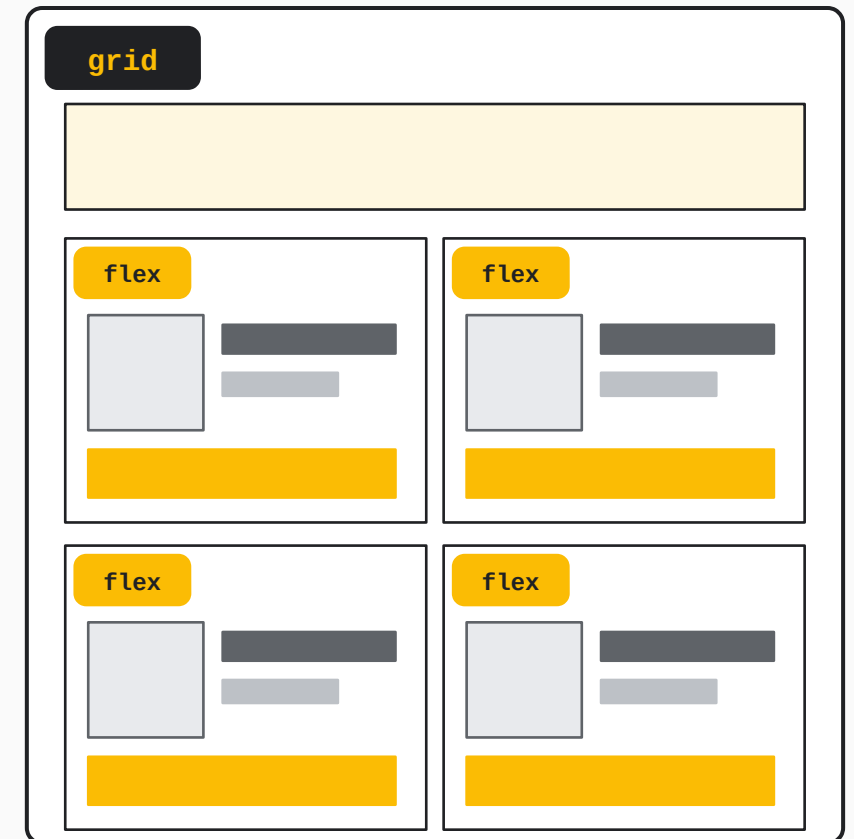
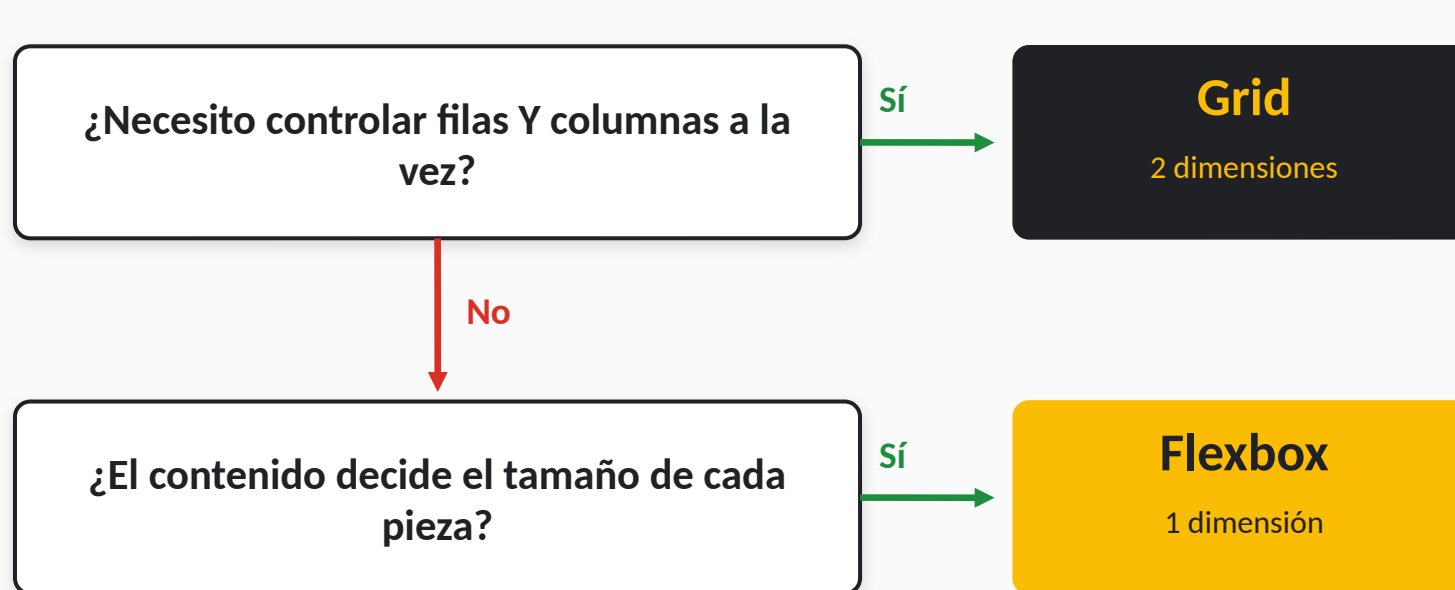
En móvil, una sola columna: basta con redibujar el mapa: **"header"** **"main"** **"aside"** **"footer"**. El HTML no se toca.



CLAVE — Cada cadena es una fila; cada palabra, una columna. Las áreas tienen que ser rectángulos.



¿Flexbox o Grid? Grid fuera, Flexbox dentro



Página en Grid • cada tarjeta en Flexbox



CLAVE — No es uno u otro: en una web real conviven. Grid organiza la página; Flexbox, el interior de cada pieza.



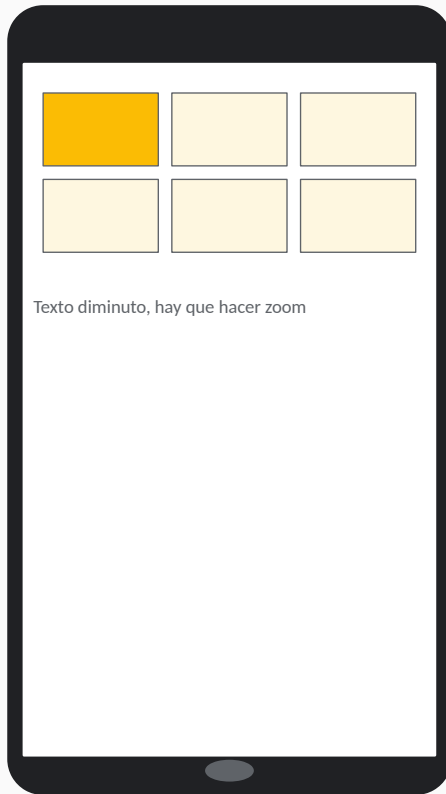
BLOQUE 3

Responsive y Container Queries

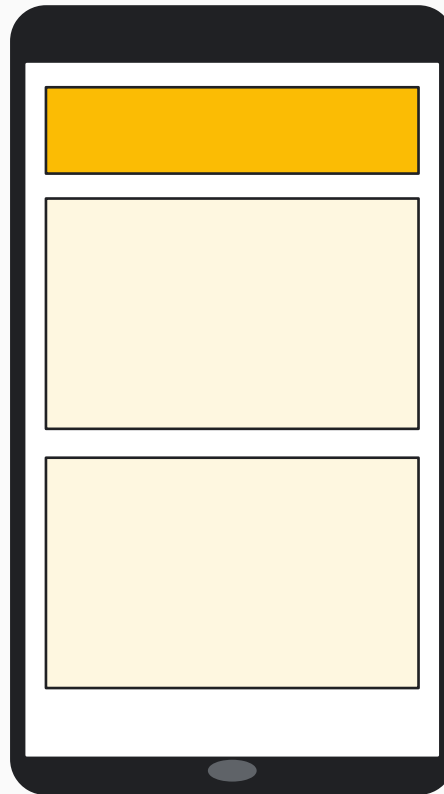
Que la página y cada componente se adapten a cualquier ancho



Paso 0: la etiqueta meta viewport



SIN meta



CON meta

```
index.html · dentro de <head>  
  
<meta name="viewport"  
      content="width=device-width, initial-scale=1">
```

Sin ella, **el móvil finge ser un escritorio de unos 980 px** y encoge la página: tus media queries nunca se activan.

width=device-width: el ancho del layout es el del dispositivo.

initial-scale=1: sin zoom al cargar.

Nunca **user-scalable=no**: impide hacer zoom y es un fallo de accesibilidad.



NOTA — Si «el responsive no funciona en el móvil» pero sí en DevTools, lo primero que se mira es esta línea.



Mobile-first: se empieza por lo pequeño y se suma

@media (min-width: 1024px)

3 columnas



@media (min-width: 600px)

2 columnas



Base (sin media query)

1 columna



ancho de pantalla →

```
_productos.scss

/* base: móvil */
.productos {
  display: grid; gap: 16px;
  grid-template-columns: 1fr;
}
@media (min-width: 600px) {
  .productos { grid-template-columns:
    repeat(2, 1fr); }
}
```

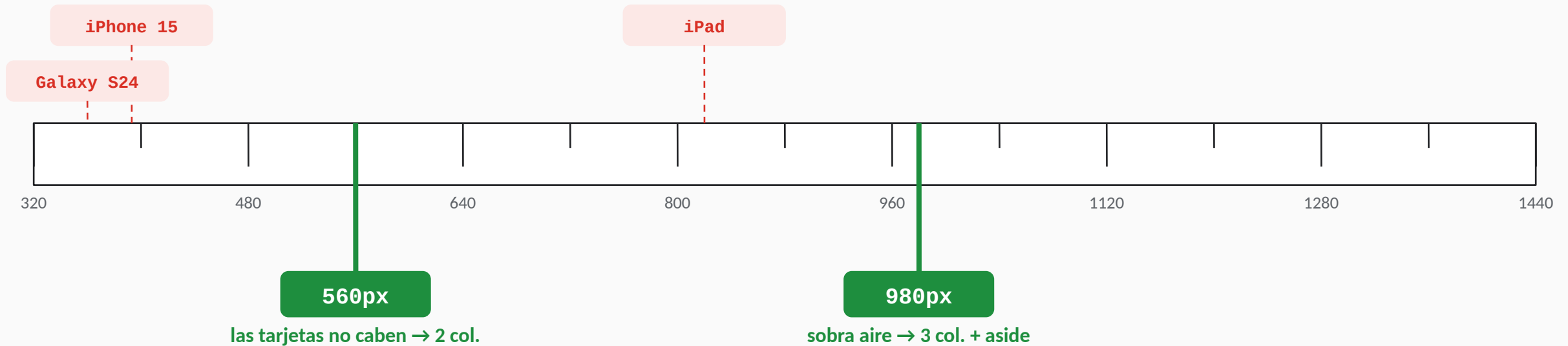
	Mobile-first	Desktop-first
Consulta	min-width	max-width
Se usa en	Proyectos nuevos	Mantener webs antiguas



REGLA — No mezcles min-width y max-width en el mismo proyecto: las reglas se pisan y nadie sabe cuál manda.



Los breakpoints los decide el contenido, no el dispositivo



1

Diseña la base móvil

Sin ninguna media query

2

Ensancha la ventana

Despacio, con DevTools abierto

3

¿Algo se ve mal?

Líneas larguísimas, huecos, cosas apretadas

4

Ahí va el breakpoint

Y vuelta a empezar



CLAVE — Un breakpoint «del iPhone 15» caduca con el siguiente móvil. Uno puesto donde el contenido se rompe no caduca nunca. Con 2-4 suele bastar.



Los breakpoints, en un solo sitio (SASS de la UD3)

```
_mixins.scss

$bp-md: 600px;
$bp-lg: 1024px;

@mixin desde($ancho) {
  @media (min-width: $ancho) {
    @content;
  }
}
```

```
components/_productos.scss

@use '../mixins' as *;
.productos {
  grid-template-columns: 1fr;
  @include desde($bp-md) {
    grid-template-columns: 1fr 1fr;
  }
}
```



Cambiar un breakpoint

Se toca una variable, no 30 media queries



La media query vive con su componente

Todo lo de .productos está junto



@content

Inserta el bloque que escribes dentro del @include



NOTA — Las variables SASS no funcionan dentro de var(): las media queries no admiten custom properties. Por eso aquí se usa \$.



Unidades: menos píxeles fijos, menos media queries

px

Fijo

Uso: Bordes, sombras finas

%

Del contenedor padre

Uso: Anchos de columnas

rem

Del tamaño de letra raíz
(16px)

Uso: Tipografía y espaciados

em

Del tamaño de letra del
propio elemento

Uso: Padding de botones

vw · vh

1 % del ancho / alto de la
ventana

Uso: Secciones a pantalla completa

fr

Fracción del espacio libre
(solo Grid)

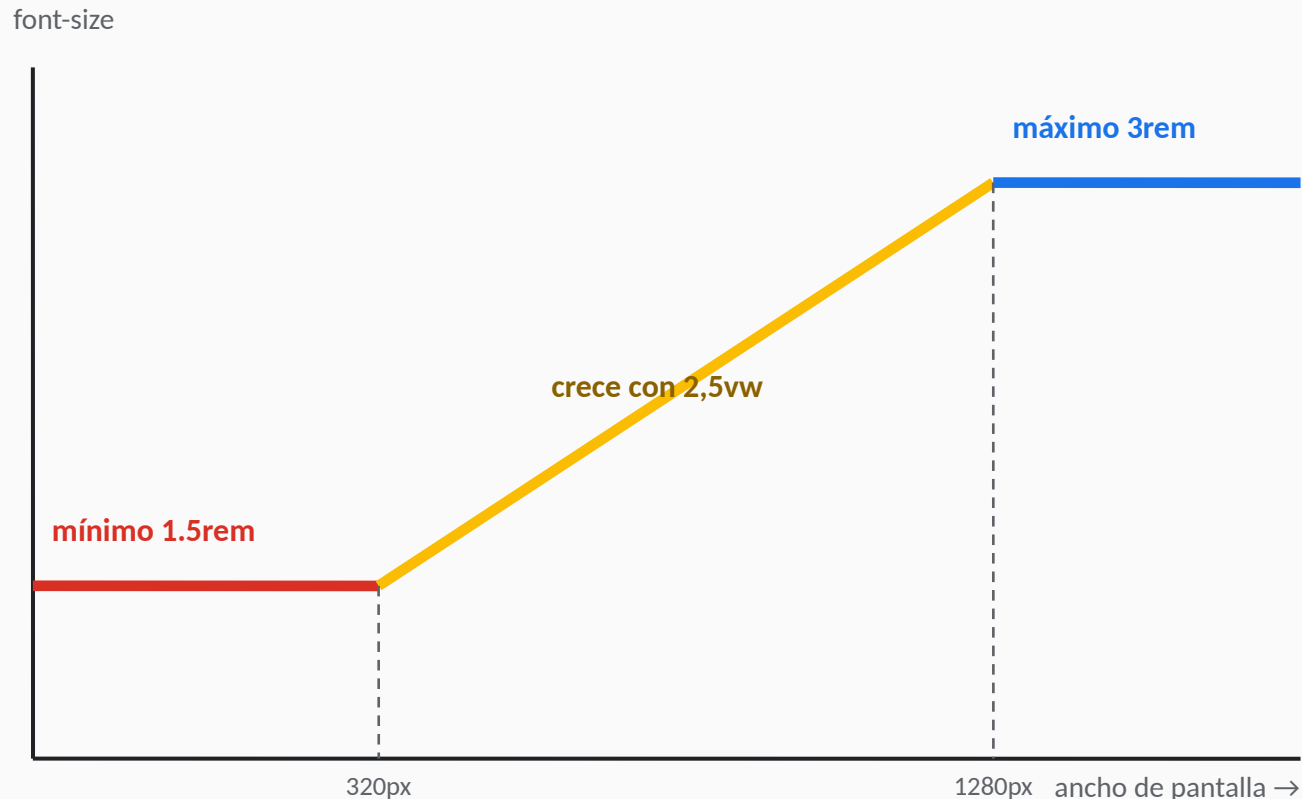
Uso: Columnas de Grid



ACCESIBILIDAD — Si el usuario sube el tamaño de letra del navegador, rem crece con él; px no. Tipografía siempre en rem.



clamp(): tipografía fluida sin media queries



_tipografia.scss

```
h1 {  
  font-size: clamp(1.5rem, 1rem + 2.5vw, 3rem);  
}
```

c **l**amp(**m**ínimo, **p**referido, **m**áximo)

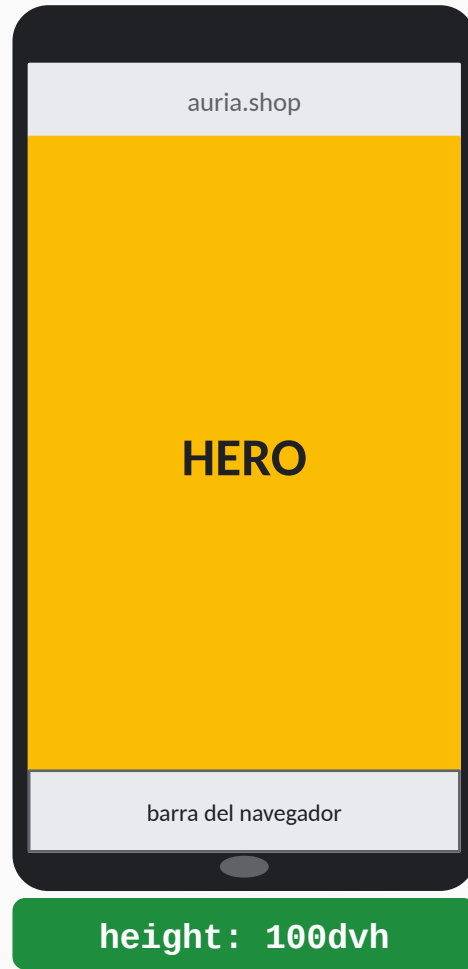
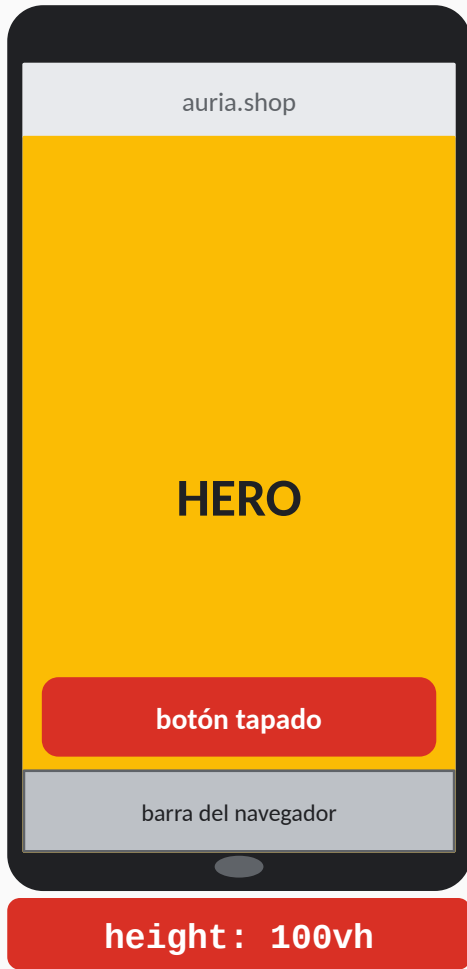
- Nunca baja del mínimo ni sube del máximo
- En medio, crece suavemente con la ventana
- Sumar rem + vw evita que el zoom deje de funcionar
- Sirve también para padding, gap o márgenes



CLAVE — Un clamp() sustituye a tres o cuatro media queries de tamaño de letra, sin saltos bruscos.



100vh miente en el móvil: usa 100dvh



Unidad	Qué mide
vh	Alto de la ventana SIN barras (el más grande)
svh	Alto con barras visibles (el más pequeño)
lvh	Alto sin barras (grande): como vh
dvh	Se ajusta en vivo al alto real visible

```
_hero.scss

.hero {
  min-height: 100dvh;
}
```



NOTA — Funciona en todos los navegadores actuales.



Media query mira la pantalla; container query, la caja

Misma pantalla de 1280px • la misma tarjeta en dos sitios

main (ancho)

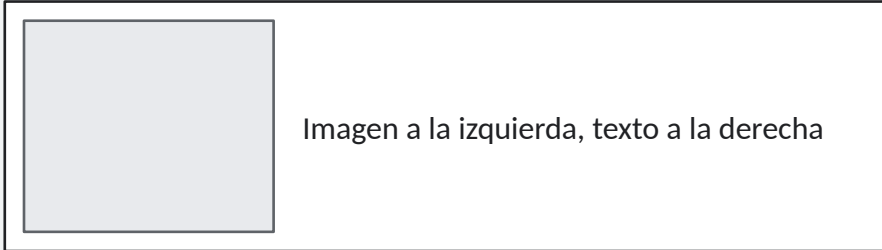


Imagen a la izquierda, texto a la derecha

aside (estrecho)

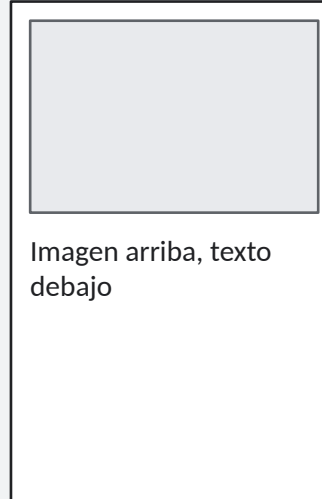


Imagen arriba, texto debajo



@media (min-width: 600px)

Ve 1280px en los dos sitios → pone la tarjeta en horizontal también en el aside, y no cabe



@container (min-width: 400px)

Cada tarjeta mira el ancho de SU contenedor → horizontal en main, vertical en el aside



CLAVE — Con container queries el componente es independiente de dónde lo pongas. Es la idea de los componentes de la UD7.



Container queries en dos pasos

Paso 1 · el PADRE se declara contenedor

```
.producto-slot {  
  container-type: inline-size;  
  container-name: card;  
}
```

Paso 2 · el HIJO pregunta por su contenedor

```
.card { display: grid; }  
@container card (min-width: 400px) {  
  .card {  
    grid-template-columns: 1fr 2fr;  
  }  
}
```

inline-size

Mide el ancho del contenedor (el caso normal)

container-name

Opcional: para elegir contenedor si hay varios anidados

cqi

1 % del ancho del contenedor: font-size: 5cqi

¡Olvidar el paso 1!

Sin container-type, el @container no hace nada y no da error



OJO — Un elemento no puede preguntar por su propio tamaño: la consulta la hace el hijo sobre el padre.



Hojas alternativas hoy: modo oscuro e impresión (RA2.d)

AURIA

Auriculares AURIA Pro

Cancelación de ruido y 40 h de batería.

por defecto

AURIA

Auriculares AURIA Pro

Cancelación de ruido y 40 h de batería.

prefers dark

```
_tema.scss

:root { --bg: #fff; --texto: #1a1a1a; }
@media (prefers-color-scheme: dark) {
  :root { --bg: #121212; --texto: #f0f0f0; }
}
body { background: var(--bg); color: var(--texto); }
```

Antes	Hoy	Para qué
<link rel="alternate stylesheet">	prefers-color-scheme + --custom-properties	Tema claro / oscuro
<link media="print">	@media print { ... }	Estilos al imprimir

```
_print.scss

@media print {
  nav, .btn, footer { display: none; }
  body { color: #000; background: #fff; }
}
```



CLAVE — Aquí es donde las custom properties (--) ganan a las variables SASS (\$): cambian en vivo, con la página ya cargada.



BLOQUE 3 • VERIFICAR

Probar el responsive de verdad



Modo dispositivo

F12 → Ctrl+Mayús+M. Usa «Responsive» y arrastra el borde: no solo 3 tamaños



Un móvil de verdad

El emulador no simula la barra del navegador ni el dedo



Zoom al 200 %

Debe seguir leyéndose sin scroll horizontal (WCAG 1.4.4 y 1.4.10)



Horizontal

Casi todo el mundo prueba solo en vertical



Contenido real

Nombres largos y textos de verdad, no «Lorem ipsum» corto

La prueba del acordeón

Abre la página y ve estrechando la ventana desde 1440px hasta 320px muy despacio.

Si en ALGÚN punto intermedio algo se solapa, se corta o aparece scroll horizontal, falta un breakpoint o sobra un ancho fijo.



ÉXITO — Ningún ancho intermedio rompe el layout, no solo los tres que has probado.



BLOQUE 4

Manos a la obra

Laboratorio AURIA, DevTools, caso real y errores comunes



La página de AURIA, de fea a responsive



Se trabaja sobre el HTML semántico de la UD3 y la carpeta scss/. El HTML solo se toca si hace falta un contenedor para el @container.



ÉXITO — Pasa la prueba del acordeón (1440 → 320 px) sin scroll horizontal y con la tarjeta bien en main y en aside.



DevTools: el CSS se depura, no se adivina

Elements › Styles

```
#comprar {  
  background: green;  
}  
  
.btn.btn--primary {  
  background: #1E88E5;  
}  
  
button {  
  background: gray;  
}
```

Tachado = perdió en la cascada



Styles

Todas las reglas que afectan al elemento, ordenadas por quién gana



Computed

El valor final y el dibujo del modelo de caja



Insignias flex / grid

Pintan ejes, líneas numeradas y huecos encima de la página



Modo dispositivo

Anchos, rotación y simulación de prefers-color-scheme



DEMO — Rendering (menú : → More tools) → «Emulate CSS prefers-color-scheme: dark» para probar el modo oscuro sin cambiar el sistema.



CASO REAL

CSS moderno a escala de producto



Mobile-first indexing

Desde 2023 Google rastrea e indexa todas las webs con su versión móvil.



Tokens que escalan

Los espaciados y tamaños del design system (UD2) también cambian entre breakpoints.



Componentes autónomos

Las container queries permiten que una tarjeta funcione igual en cualquier columna.



Modo oscuro de serie

Las grandes webs lo resuelven con custom properties y prefers-color-scheme.



Presupuesto de rendimiento

En móvil con mala conexión, cada KB cuenta (UD6 y UD12).



Pruebas en dispositivos reales

Laboratorios de móviles o servicios remotos como BrowserStack.



AURIA — Su tráfico es mayoritariamente móvil: si la ficha falla en 360px, falla para la mayoría de clientes.



Trampas de cascada y caja

!important para «ganar»

Abre una escalada que nadie gana

✗ `.btn { color: red !important; }`

✓ `.producto__btn { color: red; }`

Pensar que gana la última regla

Solo decide si hay empate de especificidad

✗ `#nav a {...} ... a.activo {...}`

✓ Comprobarlo en Styles (tachados)

Olvidar el reset de box-sizing

La caja mide más de lo que pone width

✗ `width: 300px; padding: 20px; → 340`

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

Anchos fijos en píxeles

Scroll horizontal en móvil

✗ `.contenedor { width: 1200px; }`

✓ `.contenedor { max-width: 1200px; }`



Trampas de layout y responsive

Sin meta viewport

Las media queries no se activan en el móvil

✗ `<head> ... sin meta viewport`

✓ `<meta name="viewport" content="width=device-width, initial-scale=1">`

Breakpoints de dispositivo

Caducan con el siguiente móvil

✗ `@media (min-width: 393px) /* iPhone */`

✓ `@media (min-width: 560px) /* se rompe aquí */`

Mezclar min-width y max-width

Reglas que se pisan entre sí

✗ `@media (max-width: 767px) + (min-width: 600px)`

✓ Solo min-width (mobile-first)

@container sin contenedor

No pasa nada y no da error

✗ `@container (min-width: 400px) { ... }`

✓ `.slot { container-type: inline-size; }`



Preguntas de control y debate

- 1 Una regla con un id y otra con tres clases: ¿cuál gana y por qué?
- 2 Con width: 300px, padding: 20px y border: 5px, ¿cuánto mide la caja con content-box? ¿Y con border-box?
- 3 ¿Cuándo eliges Grid y cuándo Flexbox? Pon un ejemplo de AURIA para cada uno.
- 4 ¿Por qué un breakpoint puesto donde se rompe el contenido envejece mejor que uno «del iPhone»?
- 5 ¿Qué resuelve una container query que una media query no puede resolver?



MINI-RETO PRÁCTICO

Reto express: 20 minutos, en parejas

1

Especificidad

Calcula el marcador de 5 selectores

2

Grid

Layout de 3 zonas con grid-template-areas

3

Flexbox

Menú horizontal con gap y space-between

4

Responsive

1 breakpoint mobile-first + clamp() en el h1

5

Acordeón

Sin scroll horizontal de 1440 a 320 px

Un layout de la vida real

Periódico · tienda · blog · panel de control



Pareja A



Pareja B

- DevTools → insignias grid / flex activadas
- Modo dispositivo: arrastra de 320 a 1440
- Zoom al 200 %: ¿se sigue leyendo?



ÉXITO — La otra pareja no encuentra ningún ancho intermedio en el que vuestro layout se rompa.



Cascada, caja y layout

cascada

Algoritmo que decide qué regla se aplica: origen, especificidad y orden

especificidad

Peso de un selector: (id, clase, elemento)

!important

Salta la especificidad; abusar de él es deuda técnica

herencia

Valores que los hijos reciben del padre (los de texto)

inherit / initial

Forzar el valor del padre / el valor por defecto de CSS

modelo de caja

content + padding + border + margin

box-sizing

Qué incluye width: solo contenido o también padding y borde

colapso de márgenes

Dos márgenes verticales seguidos se funden en el mayor

Flexbox

Layout en una dimensión: fila o columna

eje principal

El de flex-direction; se alinea con justify-content

eje cruzado

El perpendicular; se alinea con align-items

flex-grow

Cuánto crece un hijo si sobra espacio

Grid

Layout en dos dimensiones: filas y columnas

fr

Fracción del espacio libre de un grid



Responsive y container queries

responsive

Una sola web que se adapta a cualquier ancho

meta viewport

Hace que el móvil use su ancho real, sin encoger la página

media query

Aplica CSS según características de la pantalla

mobile-first

Base para móvil y min-width para ampliar

breakpoint

Ancho en el que cambia el layout porque el contenido lo pide

rem

Relativo al tamaño de letra raíz; respeta el del usuario

vw / vh

1 % del ancho / alto de la ventana

dvh

Alto real visible en móvil, con o sin barras

clamp()

Valor fluido con mínimo y máximo

container query

Aplica CSS según el tamaño del contenedor

container-type

Convierte un elemento en contenedor consultable

custom property

Variable de CSS (--x) que cambia en vivo

prefers-color-scheme

Media query del tema claro u oscuro del sistema

@media print

Estilos que solo se aplican al imprimir



Mañana sale un móvil nuevo. ¿Qué tienes que tocar?

Media queries

Cambian el layout de la PÁGINA en 2-4 puntos

Container queries

Cada COMPONENTE se adapta a su hueco

Unidades fluidas • Flexbox • Grid

%, rem, fr, clamp(), wrap: se adaptan sin preguntar

Si la base es fluida, la respuesta es: nada.

Cuanto más trabajo hace la base de la pirámide, menos media queries necesitas arriba.

Igual que con los tokens: el ahorro se nota el día que algo cambia.



DEBATE — ¿Cuántas media queries tiene vuestra página de AURIA? ¿Podrías quitar alguna con clamp() o con auto-fit?

CHECKPOINT

Resumen de puntos clave



La cascada decide por origen, luego especificidad y solo al final por orden



Con box-sizing: border-box, width es el ancho real de la caja



Flexbox coloca en una dimensión; Grid, en dos. Grid fuera, Flexbox dentro



Mobile-first con min-width, y breakpoints donde se rompe el contenido



rem, fr, clamp() y dvh reducen el número de media queries



Las container queries adaptan el componente a su hueco, no a la pantalla

SIGUIENTE • UD5

Arquitectura CSS: BEM y Tailwind

- BEM: bloque, elemento, modificador
- Utility-first con Tailwind CSS
- Stylelint para validar
- Actualizar la guía de estilo

AURIA ya tiene piel y se adapta a cualquier pantalla. En la UD5 ordenamos ese CSS para que escale en equipo: BEM, Tailwind y Stylelint.