# DISEÑO DE INTERFACES WEB
# TEMA2 : DESARROLLO DE INTERFACES WEB

CFGS – DAM 2

JESUITAS

Sagrado Corazón
Jesuitas · Logroño

# Indice



- Interfaces web
- Tecnologías
- CSS3 Flex y Grid
- Diseño Web Responsive

# Interfaces web



- La interfaz web son elementos gráficos que permiten al usuario acceder a los contenidos, navegar e interactuar. Para lograr que un usuario se quede y vuelva, el diseño de la interfaz es importante.
- Para que un diseño web sea efectivo, debe lograr que los usuarios del sitio puedan acceder con facilidad a los contenidos, interactuar con eficacia con todos los componentes y sentirse cómodo en forma permanente, y todo ello sin siquiera pensarlo.

# Interfaces web



- Principios básicos sobre el diseño de interfaces de usuario
  - Interfaces claras y precisas
    - Antes que cualquier otra cosa el usuario debe reconocer bien lo que está viendo, saber para qué se usa, y entender cómo la interfaz le ayudará a interactuar con la aplicación.
    - Al diseñar elementos de manera clara creamos confianza en el usuario, pero si en vez de eso le entregas una página desordenada y además con una interfaz poco familiar, el usuario puede tener problemas para usar tu página web.

# Interfaces web



- Principios básicos sobre el diseño de interfaces de usuario
  - Da el control a los usuarios
    - Las personas se sienten mucho más seguras cuando tienen el control de una situación, si el entorno de tu sitio no es seguro para el usuario, este optará por abandonar el lugar.
    - Debemos evitar forzar a la gente a realizar interacciones imprevistas. Evita elementos que salten en pantalla sin aviso, que se muevan o suenen sin la opción de ser suprimidos por el usuario.

# Interfaces web



- Principios básicos sobre el diseño de interfaces de usuario
  - Da el control a los usuarios
    - Para evitar esta clase de situaciones, debemos garantizar que el usuario sienta el control de la situación, para ello podemos añadir en varias partes de la interfaz los estatus y pasos que conlleva el proceso que están llevando.
    - Como complemento, trata de crear interfaces para cubrir todos los casos de uso, si el usuario ha llegado a un punto donde tú ya has obtenido lo que requerías, no te olvides de él y proporciona alguna clase de mensaje que le permita saber que sus metas en el proceso han sido alcanzadas.

# Interfaces web



- Principios básicos sobre el diseño de interfaces de usuario
  - Define bien tus acciones primarias y secundarias
    - Debemos definir bien cuales son las acciones principales que el usuario debe llevar a cabo en nuestro sitio, para que después, en base a eso, enfoquemos el diseño de la interfaz. Para lograr un proceso exitoso, debemos evitar que este tipo de acciones se vean interrumpidas por distracciones.
    - **Trata de mantener una acción primaria por pantalla**, no confundas al usuario con diversos caminos y mantén un proceso secuencial.

# Interfaces web



- Principios básicos sobre el diseño de interfaces de usuario
  - Define bien tus acciones primarias y secundarias
    - El utilizar una sola acción primaria por página, permitirá al usuario aprender de manera más fácil el proceso, además de que será más sencillo su uso evitando la confusión.
    - En caso de necesitar varias acciones dentro de una misma pantalla, estas se catalogarán como acciones secundarias. Una acción secundaria siempre dependerá de la acción primaria que se eligió previamente, por lo que su diseño no debe de resaltar más que la acción principal.

# Interfaces web



- Principios básicos sobre el diseño de interfaces de usuario
  - Da importancia a la consistencia
    - Siguiendo con la base de los principios anteriores, debemos establecer que el diseño de elementos debe realizarse de manera consistente, tratando siempre de utilizar diseños genéricos y a partir de ahí ir especificando lo necesario para cada parte.
    - Tenemos que evitar diseñar cada página o módulo de nuestro sitio como si fuera algo totalmente distinto a lo que fue desarrollado anteriormente. Una interfaz que no respete esto, se verá parchada y llena de inconsistencias lo que puede afectar en gran manera la experiencia del usuario.

# Interfaces web



- Principios básicos sobre el diseño de interfaces de usuario
  - Da importancia a la consistencia
    - No debemos tratar de relacionar o emparentar elementos que no cuenten con un comportamiento coherente o relación lógica con los otros del grupo establecido.
    - Es decir, si agrupamos un conjunto de elementos dándoles una característica de diseño específica, no debemos añadir esta misma característica a otro elemento que no tenga absolutamente ninguna relación con los anteriores.

# Interfaces web



- Principios básicos sobre el diseño de interfaces de usuario
  - Utiliza la jerarquía visual
    - Cuando un usuario ve las mismas cosas, en el mismo orden y sin ninguna referencia, su interés se irá rápidamente. Para evitar esto debemos crear una visualización clara de los elementos que componen nuestra interfaz, **estableciendo niveles de importancia**que nos ayuden a determinar los estilos a aplicar.
    - Al establecer una correcta jerarquía entre los elementos que componen tu interfaz, darás un respiro a los usuarios que navegan por tu sitio y automáticamente tu estructura lucirá mucha más ordenada.

# Interfaces web



- Principios básicos sobre el diseño de interfaces de usuario
  - Organiza tus elementos de manera adecuada
    - Una interfaz bien organizada permite al usuario aprender más rápido y de manera sencilla, lo que tú estás tratando de ilustrar.
    - Para poder llevar nuestra interfaz a la organización, **debemos aprender a agrupar elementos**, relacionarlos, para que de esa manera podamos identificar la orientación y colocación adecuada.

# Interfaces web



- Principios básicos sobre el diseño de interfaces de usuario
  - Organiza tus elementos de manera adecuada
    - De esta manera, el usuario no tendrá que ponerse a averiguar cuál es la relación que existe entre un elemento y otro, porque la misma interfaz le estará proporcionando esa información.
    - Además de esto, debes tratar de **mostrar en pantalla únicamente lo que sea necesario**, no trates de desplegar todo en unos cuantos píxeles, el usuario te lo agradecerá.

# Interfaces web



- Técnicas para mejorar el diseño de nuestra interfaz
  - El color atrae la atención
    - El utilizar el color como herramienta para llamar la atención es una de las técnicas más fundamentales en el diseño de interfaces.
    - Debes tener mucho **cuidado con los colores que eliges**para los elementos de tu página, ya que si no haces una elección correcta puedes terminar dando más relevancia a ciertas partes que realmente no lo ameritan, y dejando en segundo plano secciones donde muestras contenido que tú consideras relevante para el usuario.
    - Los colores en tonos cálidos como el rojo, amarillo y naranja son llamativos por naturaleza, por lo que tienden a atraer más la atención y el ojo del usuario tiende a buscarlos. Además de esto, cuando se ponen en contraste con colores fríos como el azul o el verde, tienden a crear la ilusión de ampliar el elemento que cuenta con colores cálidos.

# Interfaces web



- Técnicas para mejorar el diseño de nuestra interfaz
  - Contraste para administrar la atención
    - Lo que nos permitirá crear una sensación de jerarquía, será el uso de colores en tonalidades similares, dando colores en escala mayor a los que son más relevantes, e ir disminuyendo la escala según la importancia del elemento.
    - Una de las maneras más adecuadas de resaltar elementos, es añadiendo **contraste** mediante el uso de colores oscuros para resaltar los elementos importantes, y utilizar colores más claros para los que siguen en relevancia.

# Interfaces web



- Técnicas para mejorar el diseño de nuestra interfaz
  - Contraste para administrar la atención
    - Si por ejemplo, tuviéramos un blog donde publicamos artículo a diario, nuestra portada lucirá sobrecargada de información si no aplicamos un contraste adecuado. Para ello, podemos optar por asignar un color negro para los títulos, mientras que las fecha de publicación, el número de comentarios, el nombre del autor, o algún otro meta data, puede ir de color gris; esto indicará que el título es mucho más relevante que el resto de la información.
    - Este efecto se logra gracias a que el color oscuro crea la ilusión de que el título esta enfocado, mientras que el resto de la información parece desvanecerse en el fondo. Si por ejemplo quisiéramos agregar más relevancia al nombre del autor, podemos utilizar un color gris más fuerte, que sea intermedio entre el color del título y el resto de los datos.
    - El nivel de contraste es una herramienta muy poderosa, pero debe usarse con moderación porque si caemos en la tentación de sobre utilizarla, al final no tendrá efecto y sólo generará ruido, causando el efecto contrario al deseado.

# Interfaces web



- Técnicas para mejorar el diseño de nuestra interfaz
  - El espacio en blanco para relacionar
    - Uno de los elementos que tiene más influencia sobre la apariencia de una interfaz, es el espacio en blanco que existe entre los elementos de la página. Ese tramo que dejamos libre, y que se encarga de separar los elementos es muy importante para que el usuario pueda intuir relaciones.
    - Mediante la manipulación de los espacios en blanco, podemos indicar las relaciones entre ciertos elementos o grupos de elementos. Por ejemplo, si ponemos un título arriba de un párrafo, y estos se encuentran separados por un espacio normal, estamos dándole a entender al usuario que ese título corresponde al texto que le sigue en la parte inferior.

# Interfaces web



- Técnicas para mejorar el diseño de nuestra interfaz
  - El espacio en blanco para relacionar
    - Pero si repetimos este proceso para otras secciones compuestas por títulos y párrafos, puede resultar confuso para el usuario si no agregamos un espacio adecuado.
    - Por ello, debemos agregar un espacio considerado entre cada sección, para que de esa manera nuestro contenido sea más legible.
    - De esta manera definimos bloques de secciones únicamente modificando pequeños detalles de la interfaz, esto permitirá que las relaciones entre elementos sean identificadas más fácilmente por el ojo del usuario.

# Interfaces web



- Técnicas para mejorar el diseño de nuestra interfaz
  - Espacio entre letras
    - Si tu eres un amante de la tipografía puede que el diseño web te resulte una profesión bastante frustrante, ya que por defecto son pocas las opciones con las que se cuenta para elegir la fuente con la que se mostrará el contenido de nuestro sitio.
    - Pocas fuentes web son seguras, ya que habrá algunas que se muestren bien en alguna máquina y otras no, dependiendo de varios factores que van desde el navegador hasta el sistema operativo.
    - A eso, tenemos que aunarle el hecho de que no existen muchas variantes para modificar su estilo, pero a pesar de eso tenemos la ventaja de que cada letra esta bajo nuestro nivel de control, por lo que podemos utilizar el espacio como aliado para tratar de dar un estilo más decente, y por consecuencia una mejor presentación.

# Interfaces web



- Técnicas para mejorar el diseño de nuestra interfaz
  - Estilos a los elementos enfocados
    - Es común que manejamos varios tipos de elementos a lo largo de un formulario, por ello es fácil que la atención del usuario se pierda si no incluimos una ayuda visual.
    - Podemos agregar un estilo individual para el campo donde actualmente se encuentra colocado el puntero, esto ayudará al usuario a identificar la parte que sigue en el proceso, y donde debe de introducir la información.
      - .

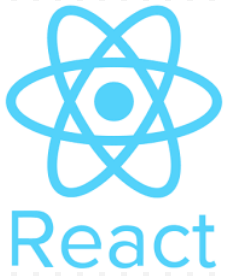
# Tecnologías



## Aplicaciones web











# CSS3 / HTML5



- REPASO CSS
  - https://www.arkaitzgarro.com/css3/
- REPASO HTML 5
  - https://www.arkaitzgarro.com/html5/

# CSS3 Flex y Grid



- “ FLEX y GRID son dos nuevos valores (HTML 5) que vamos a poder dar a la propiedad CSS display y que nos van a permitir, junto con otras propiedades, maquetar nuestras páginas web de una manera más fácil a la que se usaba tradicionalmente….

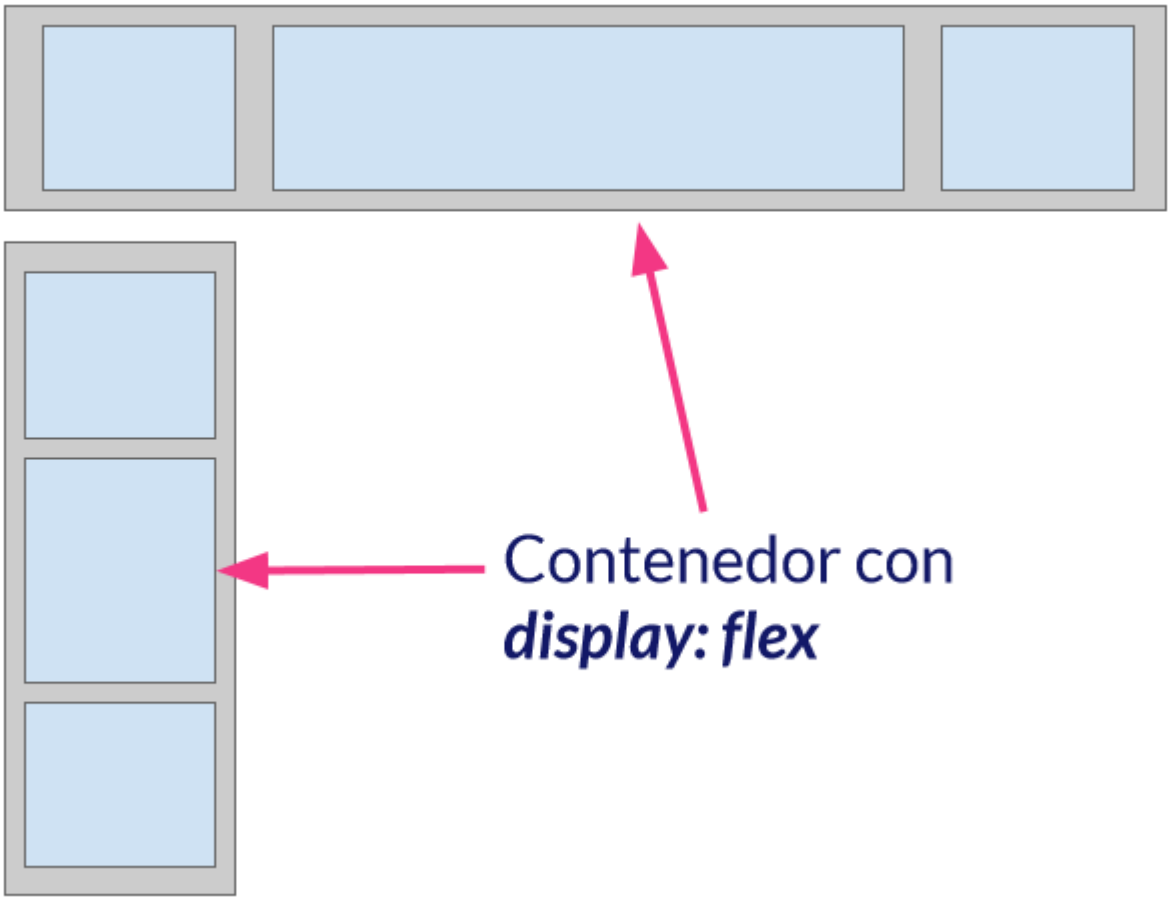
# CSS3 Flex y Grid



- PARA FLEX
  - Elemento contenedor con display:flex
- Podremos:
  - Alinear
  - Dar tamaño
  - Distribuir el espacio restante
  - En una sóla dirección
- A todos los elementos que estén dentro sin saber ni siquiera el tamaño que tienen.

# CSS3 Flex y Grid





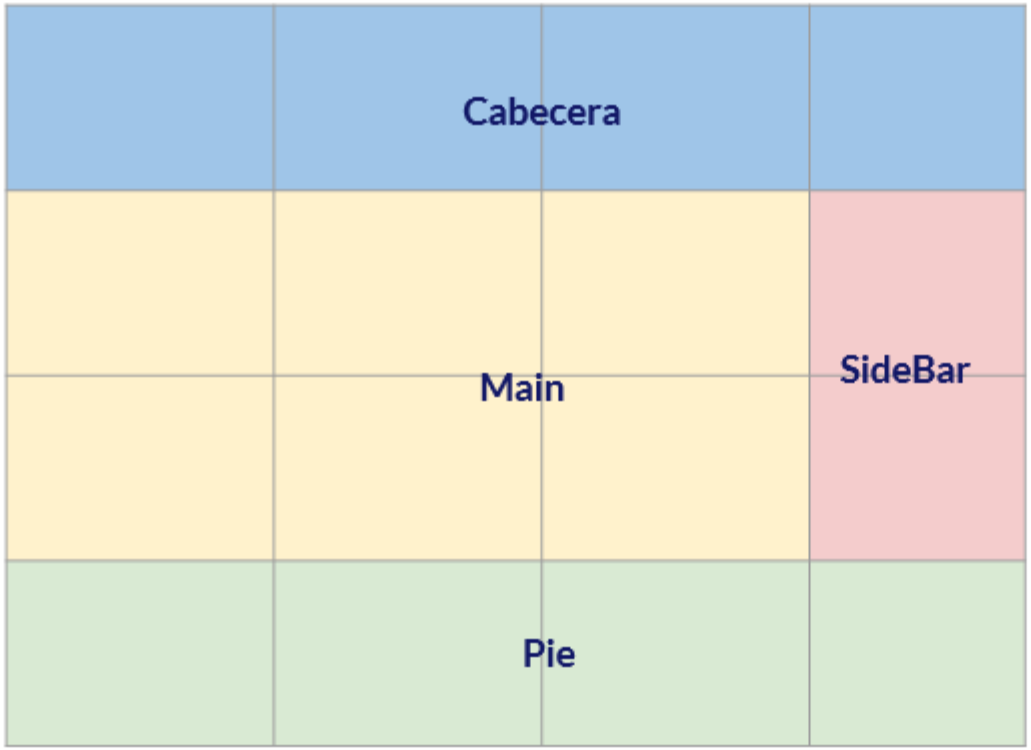
# CSS3 Flex y Grid



- PARA GRID
  - Elemento contenedor con display:flex
  - Es el sistema para maquetación más potente
  - Me permite trabajar con filas y columnas (no una sóla dimensión)

# CSS3 Flex y Grid






Contenedor con
***display: grid***

# CSS3 Flex y Grid



- NO SON SOPORTADOS POR TODOS LOS NAVEGADORES
- (pero sí en lo más nuevos, que no cunda el pánico)
- https://caniuse.com/

# CSS3 Flex y Grid



- ALGUNAS  EXTENSIONES PARA VSCODE
  - Auto Close Tag
  - Auto Rename Tag
  - GitLens
  - Intellisense for CSS
  - Prettier - Code Formatter
- REFERENCIAS!
  - https://developer.mozilla.org/es/docs/Web/CSS/CSS_Flexible_Box_Layou t
    https://developer.mozilla.org/es/docs/Web/CSS/CSS_Grid_Layout

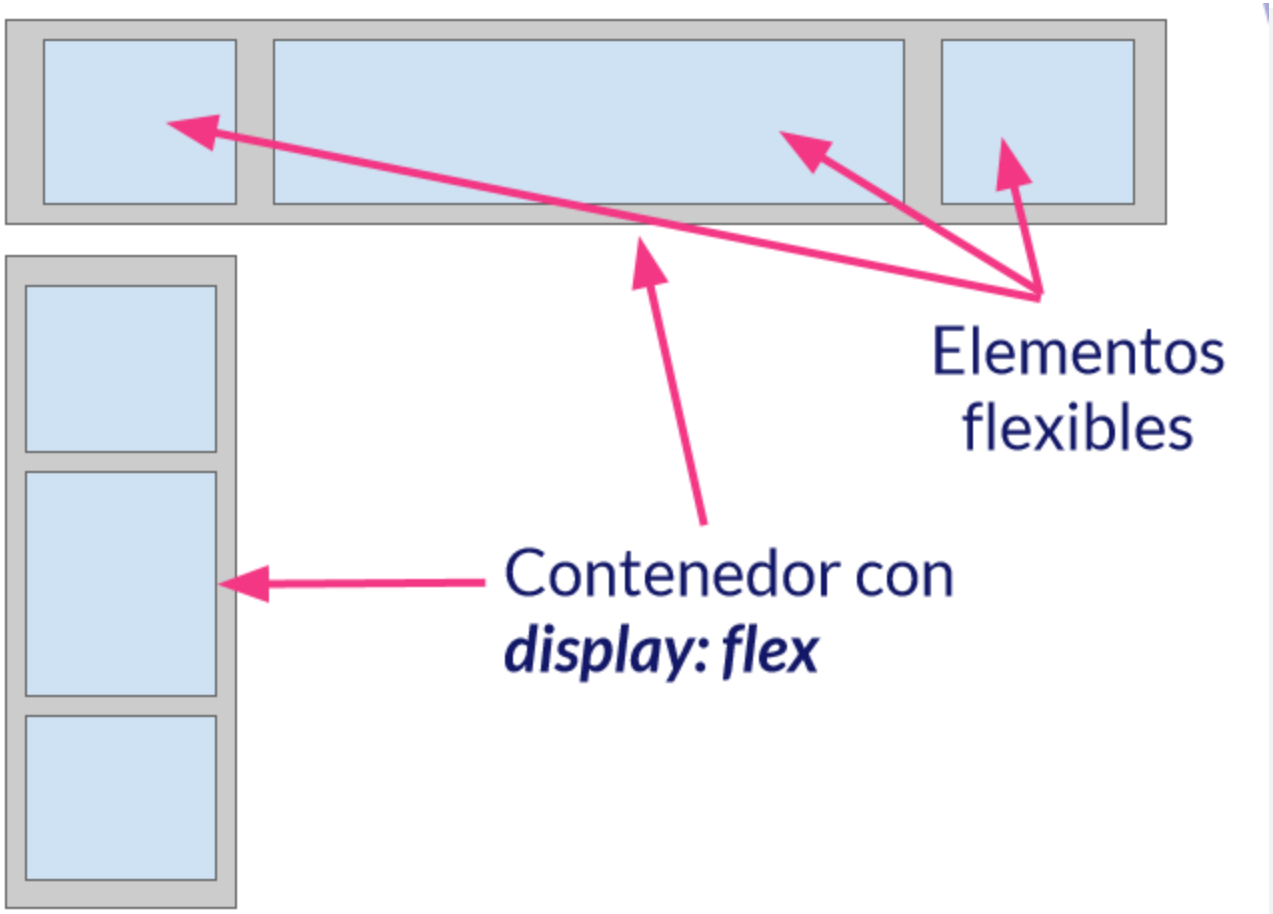
# CSS3 Flex



- “ La idea principal  de la maquetación con elementos FLEX es que vamos a tener un elemento (una etiqueta) que va a poder controlar propiedades de los elementos que contiene.
- ELEMENTOS DE MAQUETACIÓN FLEX
  - El contenedor FLEX (display:flex)
  - Los elementos flexibles que están dentro y cuyas propiedades modificaremos

# CSS3 Flex





# CSS3 Flex - Contenedor



- PROPIEDADES MODIFICABLES (de los elementos flexibles)
  - La altura
  - La anchura
  - El orden
  - La alineación vertical
  - La alineación horizontal
  - La distribución a lo largo del contenedor
- Es decir propiedades que usamos para maquetar y de una forma más fácil a la forma tradicional.

# CSS3 Flex - Contenedor



- El primer paso para maquetar usando elementos FLEX es añadir la propiedad FLEX al contenedor. En cuanto hagamos esto vamos a poder comprobar que empiezan a pasar cosas:
- Si tengo este HTML:
  - 
  - <header>
  - Primera frase
  - Segunda frase
  - Tercera frase
  - </header>
  - 

# CSS3 Flex - Contenedor



- Simplemente añadiendo la propiedad ***display:flex*** podremos ver que de manera automática los elementos ajustan su anchura a su contenido y flotan a la izquierda. Y todo es simplemente poniendo una sóla regla CSS.
- Además de con esta regla, desde el contenedor flex puedo modificar de manera directa varias propiedades de los elementos hijos. Por ejemplo:
  - La dirección en la que se van a mostrar.
  - Cómo se van a ajustar dentro del contenedor.
  - La alineación horizontal de los elementos flexibles.
  - La alineación vertical de los elementos flexibles cuando solo ocupan una línea.
  - La alineación vertical de los elementos flexibles cuando ocupan más de una línea.

# CSS3 Flex - Contenedor



- **Dirección de los elementos flexibles.**
- La ajustaremos mediante la propiedad **flex-direction** que podrá tomar los siguientes valores:
  - **row:** es la opción por defecto y ajustará los elementos flexibles de izquierda a derecha.
  - **row-reverse:** igual que la anterior pero de derecha a izquierda.
  - **column:** ajustará los elementos flexibles en columna, de arriba a abajo.
  - **column-reverse:** igual que la de arriba pero de abajo a arriba.
- **NOTA:** Estamos dando por supuesto que trabajamos con sistemas de escritura occidentales donde el flujo de lectura es de izquierda a derecha y de arriba a abajo. En otros sistemas de escritura los valores por defecto en nuestro navegador podrían ser otros y deberíamos revisar todo con cuidado.

# CSS3 Flex - Contenedor

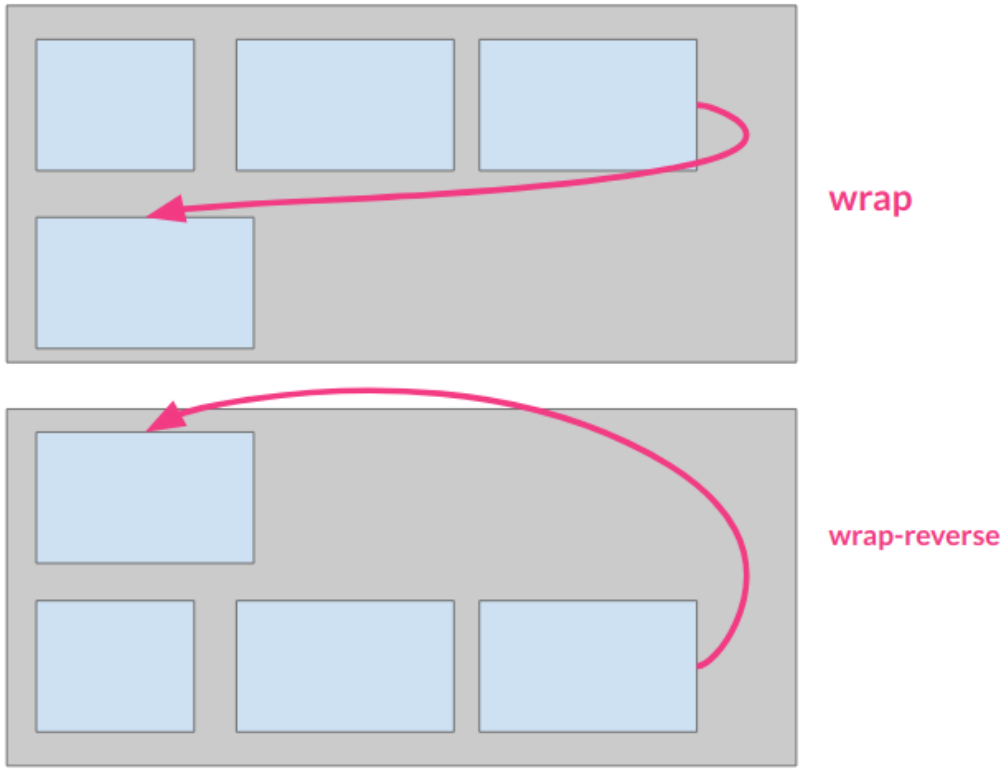


- **El ajuste de los elementos flexibles**
- Hemos visto como antes, por defecto y sin indicar ninguna anchura, los elementos flexibles adecuaban su tamaño a su contenido (si no les dábamos anchura) y se ponían todos a la izquierda permaneciendo siempre así aunque el ancho de la pantalla sea muy pequeño. Esto puede provocar desajustes en pantallas muy pequeñas. Este contratiempo podemos controlarlo usando la propiedad **flex-wrap** y eligiendo uno de los siguientes valores:
  - **no-wrap:** es el valor por defecto y fuerza para que siempre los elementos estén en la misma línea aunque esto suponga que se salgan del contenedor (les haya dado o no les haya dado anchura).
  - **wrap:** provoca un salto de línea si la anchura de los elementos (fijada por nosotros o por el contenedor) es superior a la del contenedor.
  - **wrap-reverse:** lo mismo que arriba pero de abajo a arriba.

# CSS3 Flex - Contenedor



- Podemos ver el efecto de los últimos valores en la siguiente imagen.



# CSS3 Flex - Contenedor



- Las dos propiedades, flex-direction y flew-wrap podemos juntarlas en la propiedad flex-flow con dos partes como por ejemplo:
  - flex-flow: row wrap;

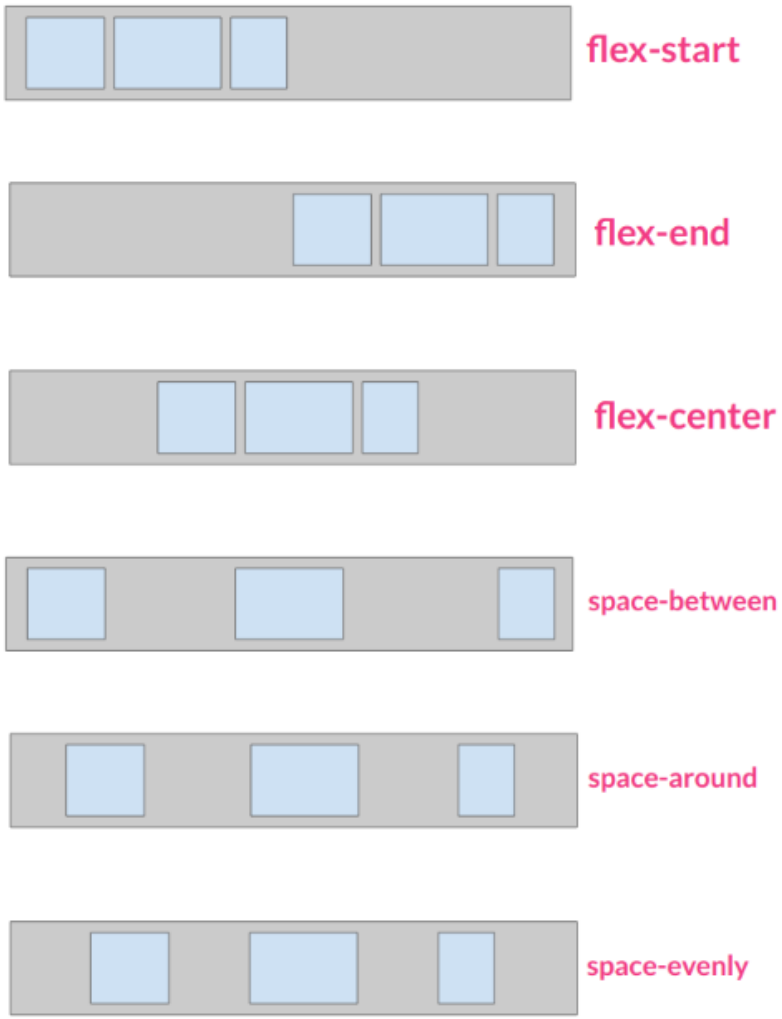
# CSS3 Flex - Contenedor



- **Alineación horizontal de los elementos flexibles**
- Podemos alinear horizontalmente los elementos flexibles, tengan o no tengan establecida una anchura, añadiendo la propiedad **justify-content** al elemento contenedor. Esta propiedad puede tener 6 valores distintos:
  - **flex-start:** Los elementos flexibles se sitúan al principio.
  - **flex-end:** Los elementos flexibles se situán al final.
  - **center:** Los elementos se centran horizontalmente
  - **space-beetween:** Distribuye el espacio restante entre los elementos pero el primero y el último están en los bordes.
  - **space-around:** Distribuye el espacio restante entre los elementos pero no tiene en cuenta la distancia a los bordes.
  - **space-evenly:** Distribuye el espacio restante entre los elementos y tiene en cuenta la distancia a los bordes.

# CSS3 Flex - Contenedor





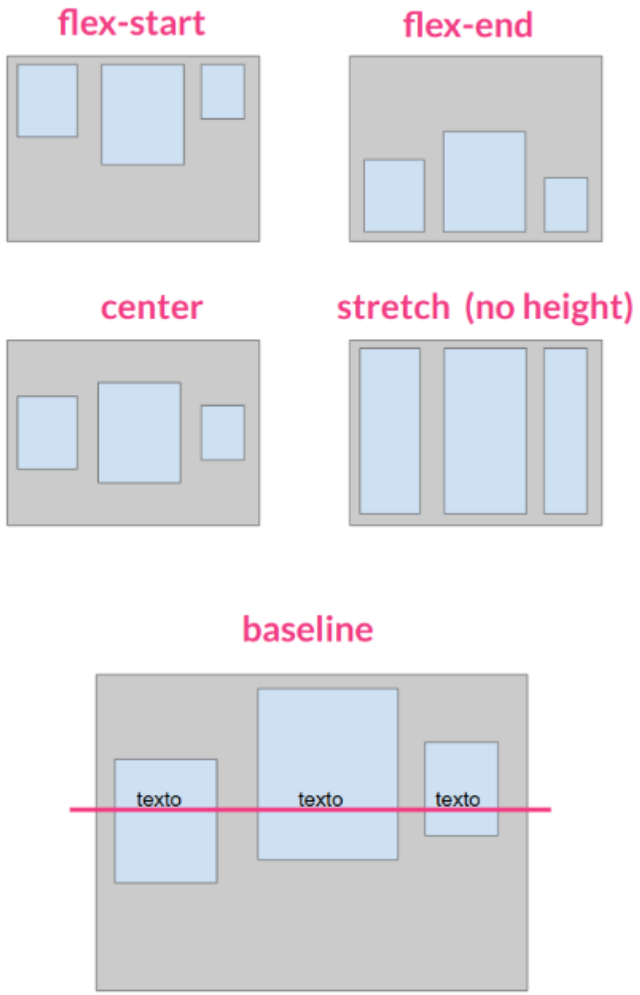
# CSS3 Flex - Contenedor



- **Alineación vertical de los elementos flexibles**
- Podemos alinear verticalmente los elementos flexibles añadiendo la propiedad **align-items** que puede tomar los siguientes valores:
  - **flex-start:** Los elementos se ponen junto al borde superior.
  - **flex-end:** Los elementos se ponen junto al borde inferior.
  - **center:** Los elementos flexibles se centran verticalmente.
  - **stretch:** Los elementos crecen en altura para ocupar toda la altura del contenedor flexible. No deben tener altura fija establecida.
  - **baseline:** Los elementos se alinean en relación con la primera línea de texto que posean los elementos flexibles.

# CSS3 Flex - Contenedor





# CSS3 Flex – Elementos flexibles



- Además de modificando las propiedades del contenedor FLEX, podemos actuar sobre los elementos flexibles modificando otra serie de propiedades relacionadas.
- En este apartado nos vamos a ocupar de cómo modificar el orden en el que aparecerán los elementos flexibles y de cómo podemos controlar que se encojan o crezcan.

# CSS3 Flex – Elementos flexibles



- **El orden de los elementos flexibles.**
- Los elementos flexibles se muestran dentro del contenedor flex en el mismo orden en que están escritos en nuestro código HTML.
- Si queremos modificar esto debemos añadir la propiedad CSS ***order*** a los elementos cuyo orden queremos modificar.
- Por defecto este valor es 0 y se mostrarán primeros aquellos elementos que tenga un mayor orden. En caso de empate se muestra antes el que primero estuviera en el código.

# CSS3 Flex – Elementos flexibles



- **Ajustando el tamaño de los elementos flexibles.**
- Para controlar el tamañó de los elementos flexibles disponemos de varias propiedades relacionadas interesantes:
  - **flex-grow:** que sirve para indicar, mediante un número, el factor de crecimiento de un elemento flexible cuando se distribuye entre los elementos flexibles el espacio restante. Por defecto es 1 pero si quiero que un elemento participe en el reparto debo añadirle esta propiedad.
  - **flex-shrink:** que sirve para indicar, mediante un número el factor de contracción de un elemento flexible cuando el tamaño de todos sobrepasa el tamaño del contenedor. Por defecto es 1 pero si quiero que un elemento participe en la contracción debo añadirle esta propiedad.
  - **flex-basic:** que sirve para indicar el tamaño de un elemento antes de que el espacio restante (negativo positivo) se distribuya. Por defecto el valor de esta propiedad es *auto* y hace que la anchura del elemento flexible se ajusta a su contenido.

# CSS3 Flex – Elementos flexibles

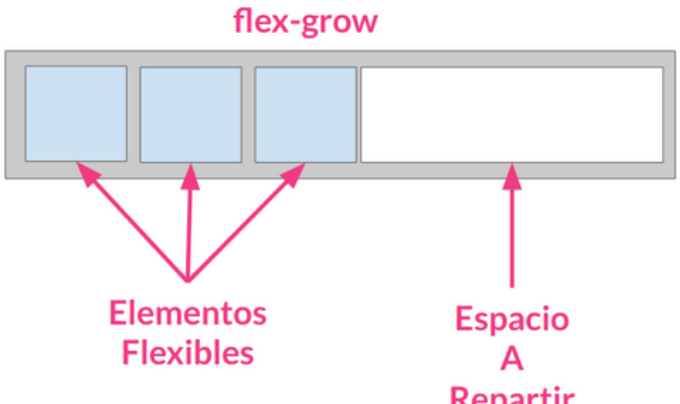


- **Ajustando el tamaño de los elementos flexibles.**
- Debemos de tener en cuenta que para una correcta maquetación debemos considerar las tres propiedades de manera conjunta. Además, se puede expresar de manera unitaria con la propiedad CSS **flex**. Por ejemplo:
  - flex: grow-factor shrink-factor flex-basis-value;

# CSS3 Flex – Elementos flexibles



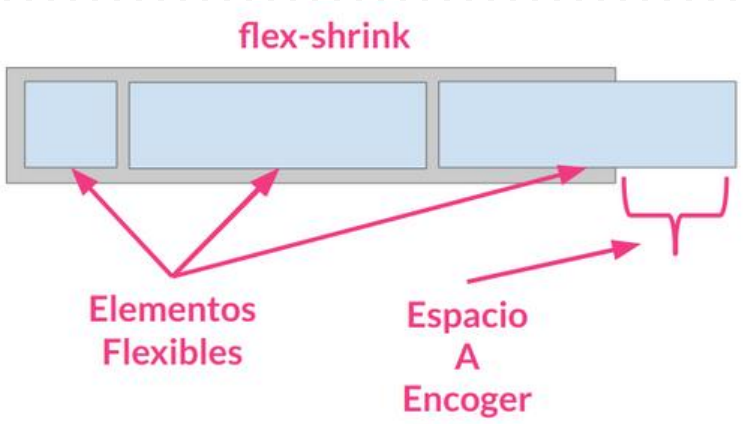
- **Ajustando el tamaño de los elementos flexibles.**
- En caso de que los elementos deban crecer (*grow*).
- Ese espacio a crecer se repartirá atendiendo al valor que tengan los elementos flexibles en la propiedad ***flex-grow***.
- Así si esos valores fueran: 2,2 y 4 el espacio tendría 8 partes y cada uno crecería la cantidad de partes de su valor.



# CSS3 Flex – Elementos flexibles



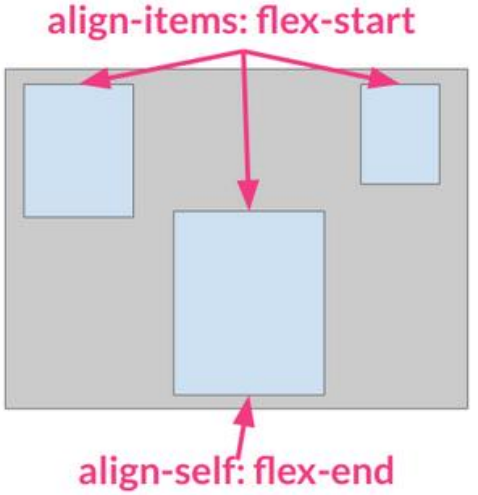
- **Ajustando el tamaño de los elementos flexibles.**
- En caso de que los elementos deban encoger (*shrink*). Ese espacio a encoger se repartirá atendiendo al valor que tengan los elementos flexibles en la propiedad ***flex-shrink***.
- Así si esos valores fueran: 2,2 y 4 el espacio tendría 8 partes y cada uno encogería la cantidad de partes de su valor.



# CSS3 Flex – Elementos flexibles



- En apartados anteriores hemos visto que desde el contenedor FLEX se puede establecer la alineación vertical de todos los elementos flexibles que contiene. Esto lo hacíamos con la propiedad CSS **align-items**.
- En ocasiones puedo necesitar que un elemento flexible tenga una alineación vertical diferente al resto. En este caso, en el elemento para el que quiero una alineación diferente, debo añadir la propiedad CSS **align-self** que puede tomar los mismo valores (y con el mismo significado) que la propiedad ***align-items***.
  - **flex-start**
  - **flex-end**
  - **center**
  - **strecth** (no debe tener altura establecida)
  - **baseline**



# It's your turn



A2.1.1 Ejercicio FLEX

# CSS3 Grid



- La idea principal que hay detrás de la maquetación **GRID** es que vamos a tener un elemento, es decir, un etiqueta que nos va a permitir *controlar* propiedades de los elementos que contiene y establecer **estructuras complejas** para distribuirlos.

# CSS3 Grid

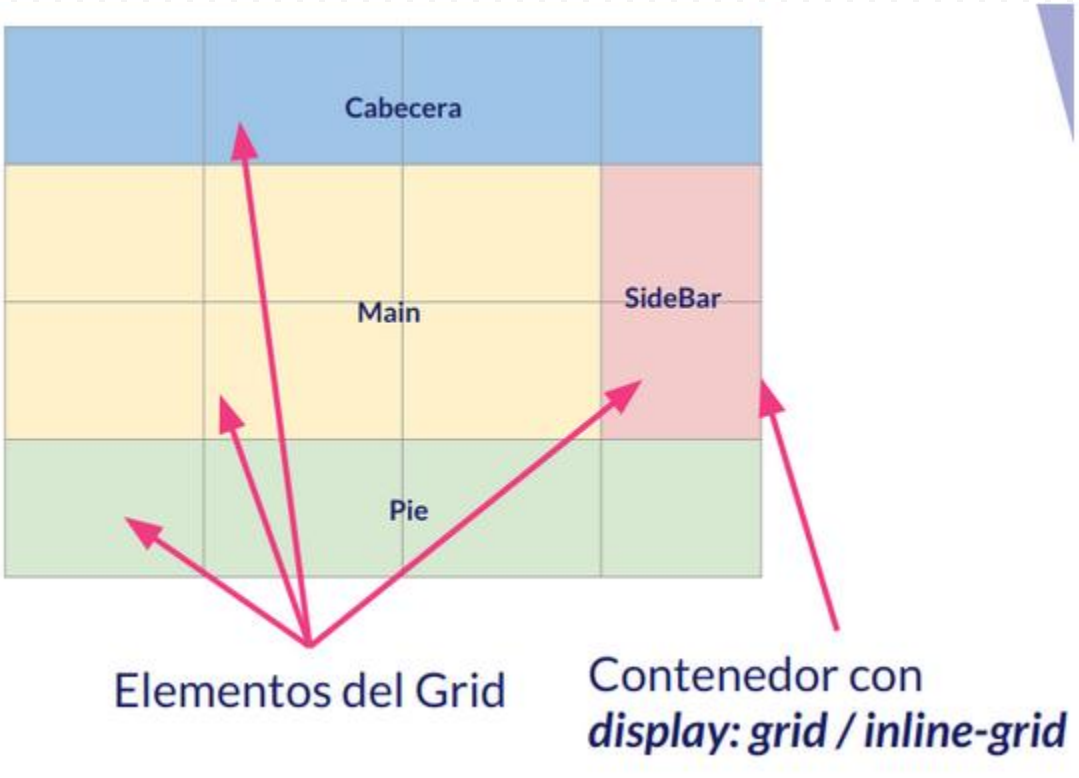


- Por lo tanto, en esta situación, vamos a poder distinguir dos tipos de elementos:
- El **contenedor GRID** que tendrá asignada la propiedad CSS *display:grid* o *display:inline-grid* la estructura y las propiedades de los elementos que contiene.
- Los **elementos GRID** que son los elementos que están dentro del contenedor, elementos que distribuiremos y cuyas propiedades modificaremos.

# CSS3 Grid



- De manera visual podemos ver una distribución de los elementos en la siguiete imagen:



# CSS3 Grid



- Pero, ¿qué podremos modificar desde el contenedor?:
  - La estructura en filas y columnas y la separación entre ellas.
  - Definir áreas del GRID con nombre.
  - La alineación horizontal y vertical de los elementos del GRID y del propio GRID dentro del elemento que lo contiene.
  - Es decir, vamos a poder controlar propiedades que usamos para maquetar y además, vamos a poder maquetar de manera mucho más ágil a lo que lo hacemos con las técnicas tradicionales de maquetado.

# CSS3 Grid



- Para empezar a maquetar usando GRID lo primero que tenemos que hacer es definir cuál de nuestras etiquetas HTML se va a convertir en el **contenedor GRID**. Una vez lo hemos decidido le daremos una de estas propiedades:
  - **display:grid** si queremos que nuestra rejilla (nuestro grid) sea un elemento de bloque.
  - **display:inline-grid** si queremos que nuestro grid sea un elemento en línea.
- Para este curso, con el objeto de que los ejemplos se muestren de manera más clara, usaremos la primera de ellas.
- Una vez hemos asignado esta propiedad al contenedor, todos los elementos que contiene pasan a convertirse de manera automática en elementos del GRID cuya colocación y propiedades podremos empezar a modificar desde el contenedor.

# CSS3 Grid



- **Definición de la estructura del GRID**
  - **grid-template-columns:** Para definir el número y tamaño de las diferentes columnas de mi estructura. Debo de poner tantos valores de anchura como columnas quiero que tenga el GRID.
  - **grid-template-rows:** Para definir el número y tamaño de las diferentes filas de mi estructura. Debo de poner tanto valores de altura como filas quiero que tenga el GRID.
  - **grid-row-gap:** Para establecer la separación entre las diferentes columnas.
  - **grid-column-gap:** Para establecer la separación entre las diferentes filas.

# CSS3 Grid



- **Ejemplos con columnas:**

```
/* Tres columnas que se reparten el 100% del contenedor*/
grid-template-columns: 20% 50% 30%;

/* Cuatro columnas. Tres de tamaño fijo 100px y la otra ocupa el
espacio libre restante */
grid-template-columns: 100px auto 100px 100px;

/* Cuatro columnas. Todas con un tamaño igual */
grid-template-columns: auto auto auto auto;

/* Tres columnas cada una con nombre (entre []). Dos con tamaño
fijo y la otra ocupando el espacio restante */
grid-template-colums: [id] 100px [nombre] 300px [apellidos] auto;
```

# CSS3 Grid



- **Ejemplos con filas:**

```
    /* Tres filas que se reparten toda la altura del contenedor
(la que sea) */
    grid-template-rows: 20% 50% 30%;

    /* Cuatro filas. Tres de altura fija 100px y la otra ocupará
el resto del espacio libre hasta llenar todo el contenedor en
altura.*/
    grid-template-rows: 100px auto 100px 100px;

    /* Cuatro filas que se reparten de manera equitativa el alto
del contenedor */
    grid-template-rows auto auto auto auto;

    /* Tres filas (todas con nombre, entre corchetes) Dos de
ellas con tamaño fijo y la restante ocupará todo el alto libre.
*/
    grid-template-rows: [uno] 100px [dos] 300px [tres] auto;
```

# CSS3 Grid



- En estas dos propiedades también puedo repetir valores y usar la unidad fr que me sirve para establecer ratios para que los elementos se repartan el espacio restante. Podemos verlo mejor con un par de ejemplos.

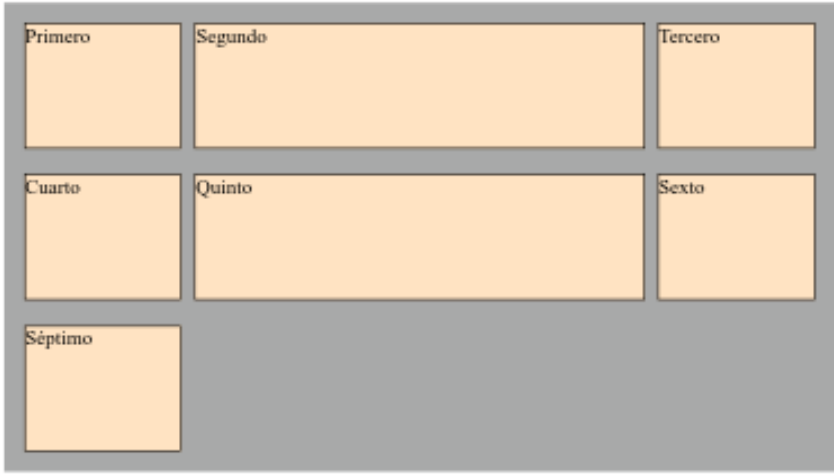
```
/* Cuatro columnas. Tres de 20% con nombre col-start. Y la último
que ocupará el resto del espacio libre */
grid-template-columns: repeat(3, 20% [col-start]) auto;
/* Cuatro columnas. Una de tamaño fijo y las demás se reparten el
espacio libre en 5 partes de la siguiente manera (2+1+2) */
grid-template-columns: 2fr 100px 1fr 2rf;
```

# CSS3 Grid



- Vemos cómo se han distribuido los 7 elementos en una cuadrícula, en un grid de 3x3 siguiendo la estructura que le hemos dicho.

```
<div class="container">
  <div>Primero</div>
  <div>Segundo</div>
  <div>Tercero</div>
  <div>Cuarto</div>
  <div>Quinto</div>
  <div>Sexto</div>
  <div>Séptimo</div>
</div>
```




```
.container {
  background-color: #aaa;
  display: grid;
  grid-column-gap: 10px;
  grid-row-gap: 20px;
  grid-template-columns: 20% auto 20%;
  grid-template-rows: repeat(3,
100px);
  margin: 20px auto;
  padding: 1em;
  width: 80%;
}

.container > div {
  background-color: bisque;
  border: 1px solid black;
}
```

# CSS3 Grid

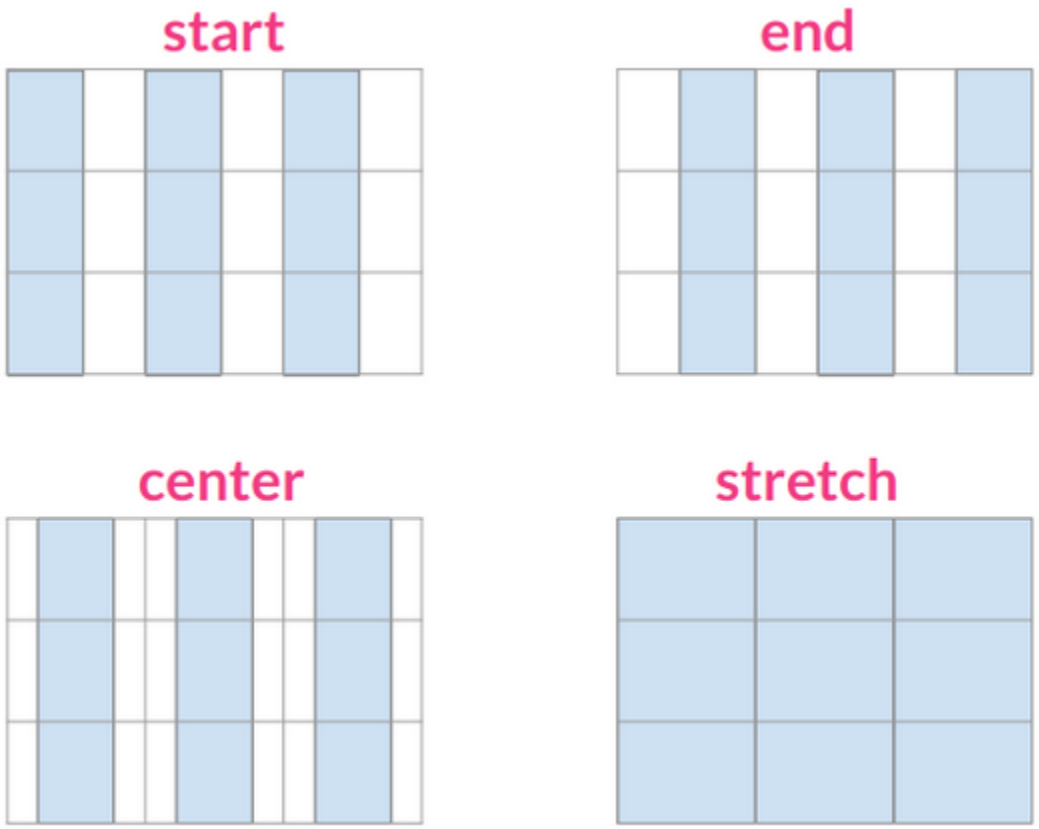


- **Alineación Horizontal**
- Por defecto los elementos del GRID ocupan todo el ancho de la celda que le corresponde pero podemos optar por otro tipo de alineaciones horizontales dando valores a la propiedad **justify-items.** Los diferentes valores que puede tomar son los siguientes:
  - **start**
  - **end**
  - **center**
  - **stretch** que es la opción por defecto.

# CSS3 Grid



- **Alineación Horizontal**



# CSS3 Grid

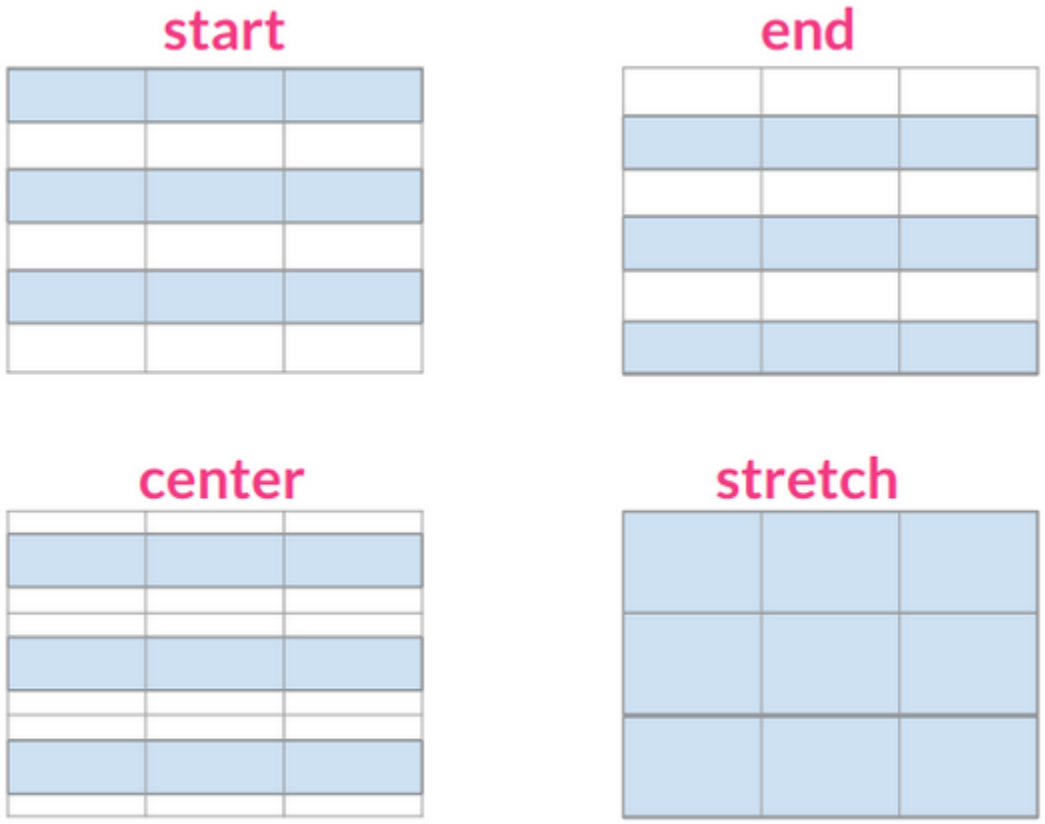


- **Alineación Vertical**
- Muy similar a lo anterior. Por defecto los elementos del GRID ocupan todo el alto de la celda que le corresponde pero podemos optar por otro tipo de alineaciones verticales dando valores a la propiedad **align-items**. Los diferentes valores que puede tomar son los siguientes:
  - **start**
  - **end**
  - **center**
  - **stretch** que es la opción por defecto.

# CSS3 Grid



- **Alineación Vertical**



# CSS3 Grid



- **Alineación Horizontal/Vertical**
- Si quiero juntar estas dos últimas alineaciones usaré la propiedad **place-items** indicando primero el valor para **align-items** y después el valor para **justify-items.**

# CSS3 Grid



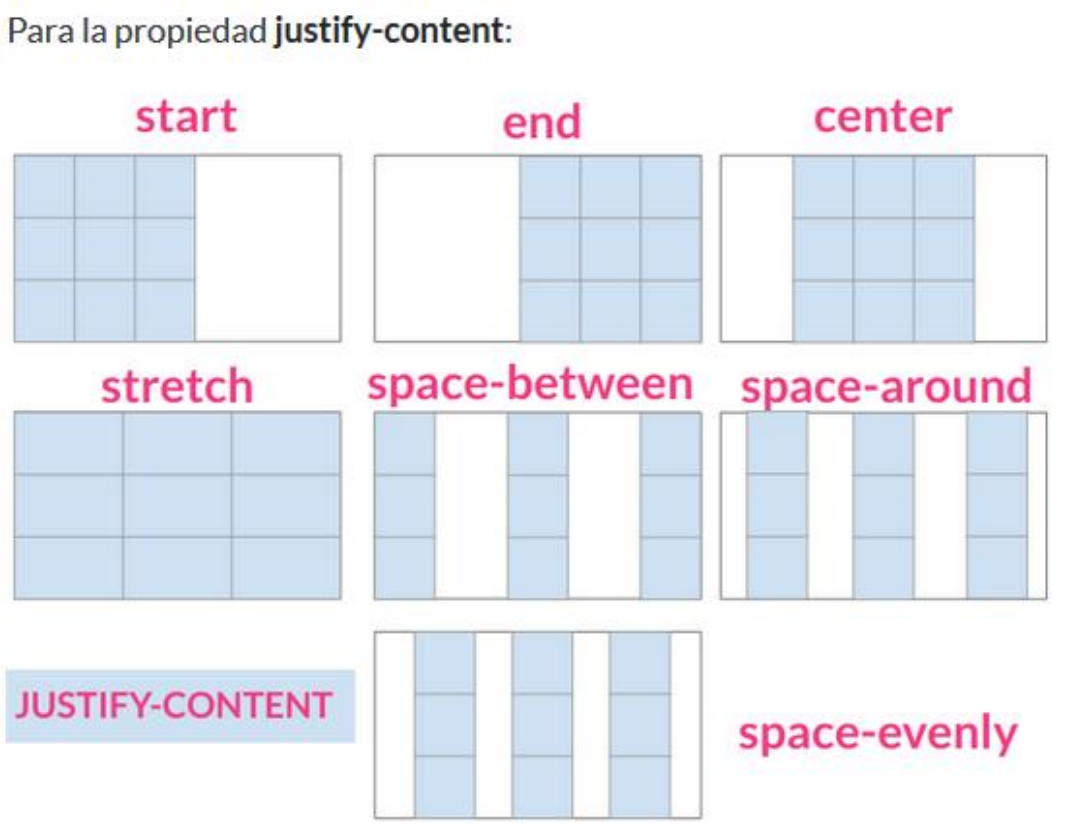
- **Distribución dentro del contenedor**
- En determinados casos puede suceder que los elementos del GRID no ocupen todo el ancho o todo el alto del contenedor GRID.
- En estas ocasiones puedo distribuir las columnas y las filas usando las propiedades **justify-content** (horizontal) y **align-content** (vertical). Ambas puede tomar los mismos valores y para entender mejor esos valores y cómo funcionan vamos a presentar dos imágenes.

# CSS3 Grid



## Distribución dentro del contenedor

- La zona azul representa las columnas que están dentro de un rectángulo que es el contenedor GRID.

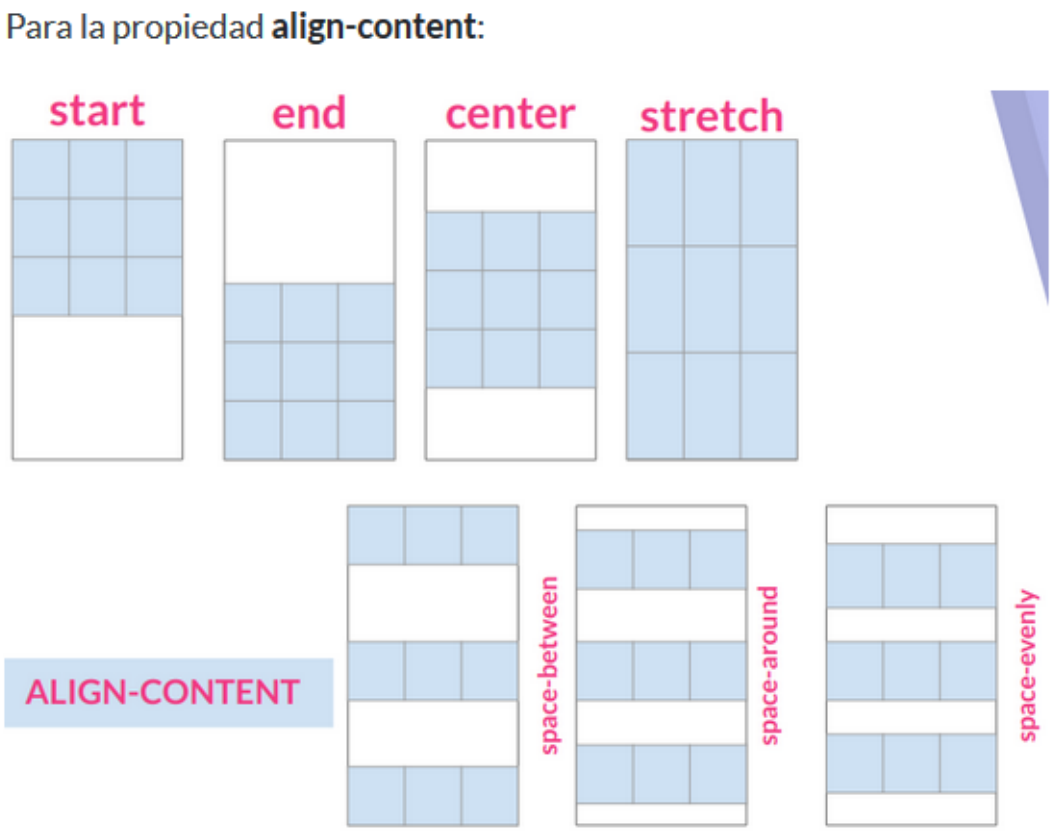


# CSS3 Grid



## Distribución dentro del contenedor

- La zona azul representa las columnas que están dentro de un rectángulo que es el contenedor GRID.



# CSS3 Grid



- **Distribución dentro del contenedor**
- Si quiero juntar estas dos últimas alineaciones usaré la propiedad **place-content** indicando primero el valor para **align-content** y después el valor para **justify-content.**

# CSS3 Grid



- Los **Elementos GRID** son los hijos directos, dentro del árbol DOM de nuestra página HTML, del elemento con la propiedad CSS ***display:grid***.
- De manera individual podemos modificar las propiedades de estos elementos para conseguir lo siguiente:
  - Definir el área o zona del GRID (rejilla) que va a ocupar.
  - Especificar la alineación horizontal del elemento.
  - Especificar la alineación vertical del elemento.
- Y además, existen ciertas reglas de colocación implícita que debemos de conocer.

# CSS3 Grid

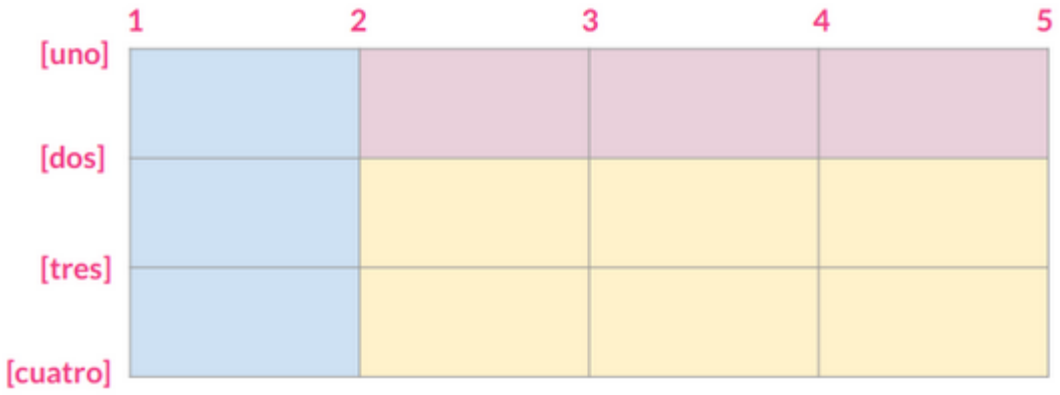


- **Área del elemento GRID**
- Para especificar el área que va a ocupar el elemento GRID lo haremos con las siguiente propiedades:
  - **grid-column-start**
  - **grid-column-end**
  - **grid-row-start**
  - **grid-row-end**

# CSS3 Grid



- **Área del elemento GRID**
- Fijaros que puede especificar el área que ocupa de tres maneras principalmente:
- Indicando el número de línea donde empieza y donde acaba.
- Indicando con nombres de líneas, que habremos definido al definir el contenedor, donde empieza y donde acaba.
- Indicando cuánto ocupa en la dirección en cuestión (fila o columna) y usando **span** y el valor de la extensión.




```css
#azul {
  background-color: blue;
  grid-column-start: 1;
  grid-column-start: 2;
  grid-row-start: uno;
  grid-row-end: cuatro;
}

#rojo {
  background-color: red;
  grid-column-start: 2;
  grid-column-start: 3;
  grid-row-start: uno;
  grid-row-end: uno;
}

#amarillo {
  background-color:
yellow;
  grid-column-start: 2;
  grid-column-start: 5;
  grid-row-start: dos;
  grid-row-end: span 2;
}
```

# CSS3 Grid



- Podemos juntar estas propiedades:

```
/* Para juntar las dos propiedades referentes a columnas */
grid-column: start / end;

/* Para juntar las dos propiedades referentes a filas */
grid-row: start / end;

/* Para junta las cuatro propiedades que nos permiten definir el
área */
grid: row-start column-start row-end column-end;
```

# CSS3 Grid



- **Alineación horizontal.**
- La alineación horizontal de un elemento de manera individual se consigue usando la propiedad **justify-self** que puede tomar los mismos valores y funciona igual que la propiedad **justify-items** que dábamos al contenedor GRID.
- Estos valores son:
  - start
  - end
  - center
  - stretch (por defecto)

# CSS3 Grid



- **Alineación vertical.**
- La alineación vertical de un elemento de manera individual se consigue usando la propiedad **align-self** que puede tomar los mismos valores y funciona igual que la propiedad **align-items** que dábamos al contenedor GRID.
- Estos valores son:
  - start
  - end
  - center
  - stretch (por defecto)

# CSS3 Grid

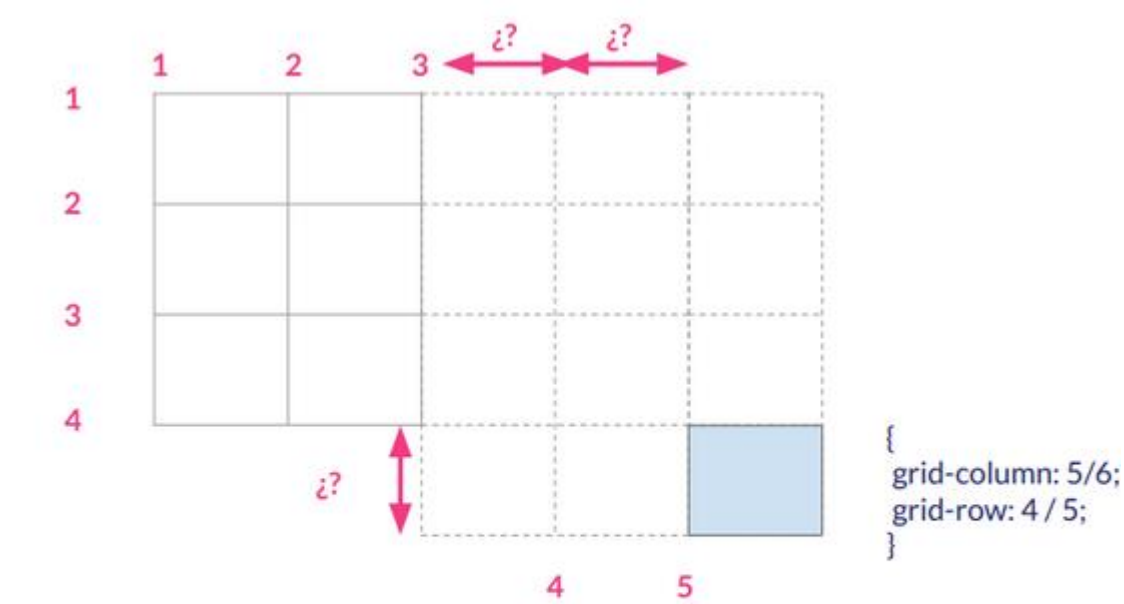


- **Colocación Implícita**
- La **colocación implícita** de los **elementos GRID** es lo que sucede cuando colocamos esos elementos **fuera** de la estructura del contenedor o cuando no les damos posición.
- Cuando colocamos una elemento fuera de la estructura definida para su contenedor GRID podemos mantener cierto control añadiendo al contenedor (que no al elemento) las siguiente propiedades:
  - **grid-auto-columns** Que dará tamaño a las columnas de separación entre los límites del contenedor y la posición donde hemos dejado nuestro elemento.
  - **grid-auto-rows** Que dará tamaño a las filas de separación entre los límites del contenedor y la posición donde hemos dejado nuestro elemento.

# CSS3 Grid



- **Colocación Implícita**



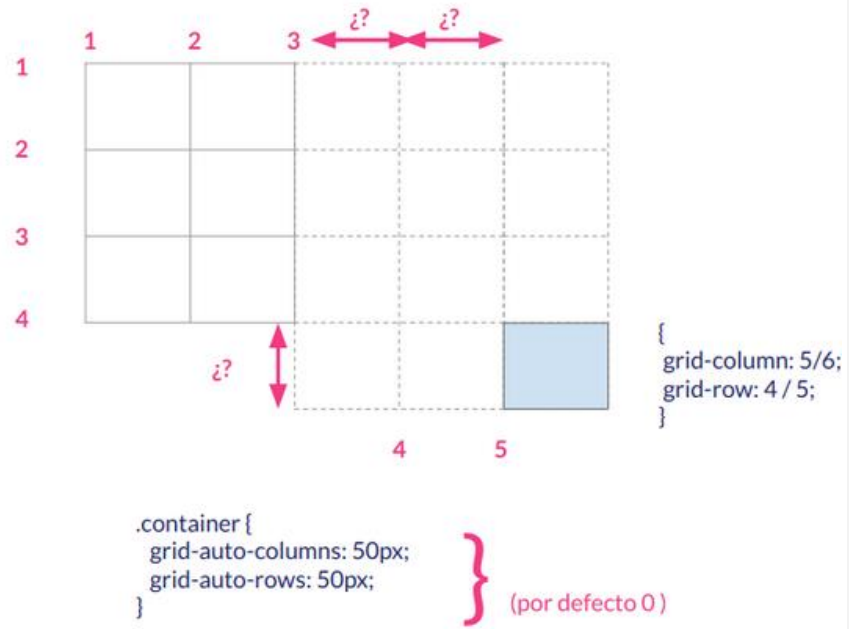
# CSS3 Grid



- **Colocación Implícita**
- Al dar al contenedor

las siguiente propiedades:

```
.container {
  ... grid-auto-columns:
50px;
  grid-auto-rows: 50px;
  ...;
}
```




- Las filas y las columnas con los interrogantes, que están fuera del grid que es 2x3, tendrán unas dimensiones de 50x50. Por defecto estos valores son 0.

# CSS3 Grid



- **Colocación Implícita**
- En el caso de que no hayamos especificado el área que le corresponde a un elemento de GRID podemos establecer ciertas reglas del comportamiento mediante la propiedad **grid-auto-flow** que añadiremos al contenedor.
- Esta propiedad puede tomar varios valores:
  - **row:** Rellena primero las filas. Es la opción por defecto.
  - **column:** Rellena primero las columnas. Es la opción por defecto.
  - **dense:** Intenta rellenar primero los huecos por si viene elementos posteriores más pequeño. Hay que tener cuidado al usar ésta porque puede provocar cambios de orden en los elementos.

# CSS3 Grid



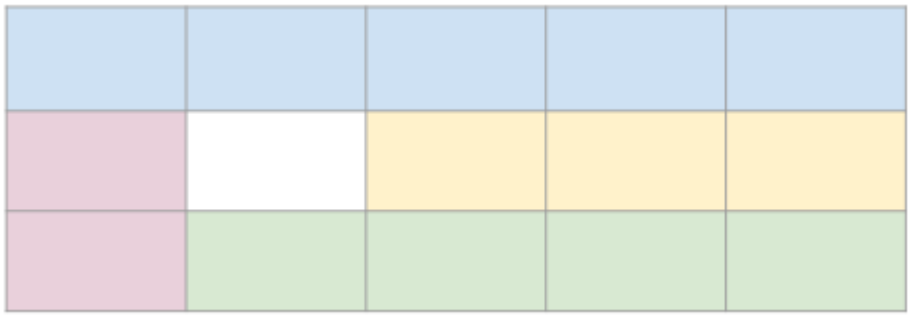
- **Areas Grid**
- Vamos a ver como podemos maquetar únicamente nombrando áreas. Para ello usaremos las siguiente propiedades:
  - **grid-area** en los elementos GRID
  - **gird-template-area** en el contenedor GRID.

# CSS3 Grid



- **Areas Grid**

```
.container {
  grid-template-columns: repeat(5,
20%);
  grid-template-rows: repeat(3, 100px);
}
```



```
.container {
  display: grid;
  grid-template-columns: repeat(5,
20%);
  grid-template-rows: repeat(3, 100px);
  grid-template-areas:
    "cab cab cab cab cab"
    "menu  .  main main main"
    "menu  pie pie pie pie";
}
```

```
#cab {
  background-color: blue;
  grid-area: cab;
}

#pie {
  background-color: green;
  grid-area: pie;
}

#menu {
  background-color: red;
  grid-area: menu;
}

#principal {
  background-color:
yellow;
  grid-area: main;
}
```

# CSS3 Grid



- Una vez visto esto podemos juntar todas las propiedades grid-template-* en una sola propiedad **grid-template**.
- Además, ya no nos quedan por ver más propiedades del contenedor y podríamos juntar todas en la propiedad **grid**.
- Aunque es preferible tener más control y tener todo separado. Son tantas cosas que a veces si las juntamos perdemos el foco.

# CSS3 ¿Flex o Grid?



- Independientemente de qué sistema utilicemos maquetar usando FLEX o usando GRID es mucho más fácil que maquetar usando sistemas tradicionales basados únicamente en float, position, tablas etc..

# CSS3 ¿Flex o Grid?



- FLEX
  - Ventajas
    - Una sola dirección
    - Controlo fácilmente la alineación de los elementos flexibles.
  - Desventajas
    - Para el mismo layout que GRID necesito una estructura HTML más compleja

# CSS3 ¿Flex o Grid?



- GRID
  - Ventajas
    - Dos dimensiones
    - Puedo definir muy fácilmente la estructura general
  - Desventajas
    - Es muy fácil definir la separación (gap) entre la celdasDesventajas
    - La alineación “dentro” de las celdas

# CSS3 ¿Flex o Grid?



- CONCLUSIONES
  - Hay cosas que sólo puedo hacer con Flex.
  - Hay cosas que sólo puedo hacer con Grid.

**USA**

**LAS DOS**

Nada impide que un área de un contenedor GRID sea a su vez un contenedor FLEX.

# It's your turn



A2.1.2 Ejercicio GRID

# Diseño Web Responsive



- Responsive Web Design (o diseño web adaptativo) se trata de una técnica de diseño y desarrollo web por el que se consigue que un único sitio se adapte perfectamente a todos los dispositivos que puedan consumirlo, desde ordenadores de escritorio a netbooks, tablets, teléfonos móviles, televisores, etc.
- En definitiva, se trata de construir una única web para que se vea correctamente y aproveche las particularidades de todo dispositivo que hoy exista, o pueda existir en el futuro, independientemente de la pantalla en la que se muestre.

# Diseño Web Responsive



- Tanto la idea y como el propósito del diseño web adaptativo fueron previamente discutidos y descritos por el consorcio W3C en julio de 2008 en su recomendación *Mobile Web Best Practices* bajo el subtítulo One Web.
- Dicha recomendación, aunque específica para dispositivos móviles, puntualiza que está hecha en el contexto de "One Web", y que por lo tanto engloba no solo la experiencia de navegación en dispositivos móviles sino también en dispositivos de mayor resolución de pantalla como dispositivos de sobremesa.
- El concepto de "One Web" hace referencia a la idea de construir una Web para todos (Web for All) y accesible desde cualquier tipo de dispositivo (Web on Everything).
- Hoy en día, la variedad de dispositivos existente en el mercado ha provocado que la información disponible no sea accesible desde todos los dispositivos, o bien es accesible pero la experiencia de navegación es muy pobre.

# Diseño Web Responsive



- Ventajas de utilizar un diseño web adaptativo:
  - Con **una sola versión** en HTML y CSS se cubren todas las resoluciones de pantalla, es decir, el sitio web creado estará optimizado para todo tipo de dispositivos: PCs, tabletas, teléfonos móviles, etc. Esto mejora la experiencia de usuario a diferencia de lo que ocurre, por ejemplo, con sitios web de ancho fijo cuando se acceden desde dispositivos móviles.
  - **Reducción de costos**. Se reducen los costes ya que hasta hoy se debe hacer un portal para la Web y otro para dispositivos móviles. Esto origina mayores costos de creación y mantenimiento de la información.

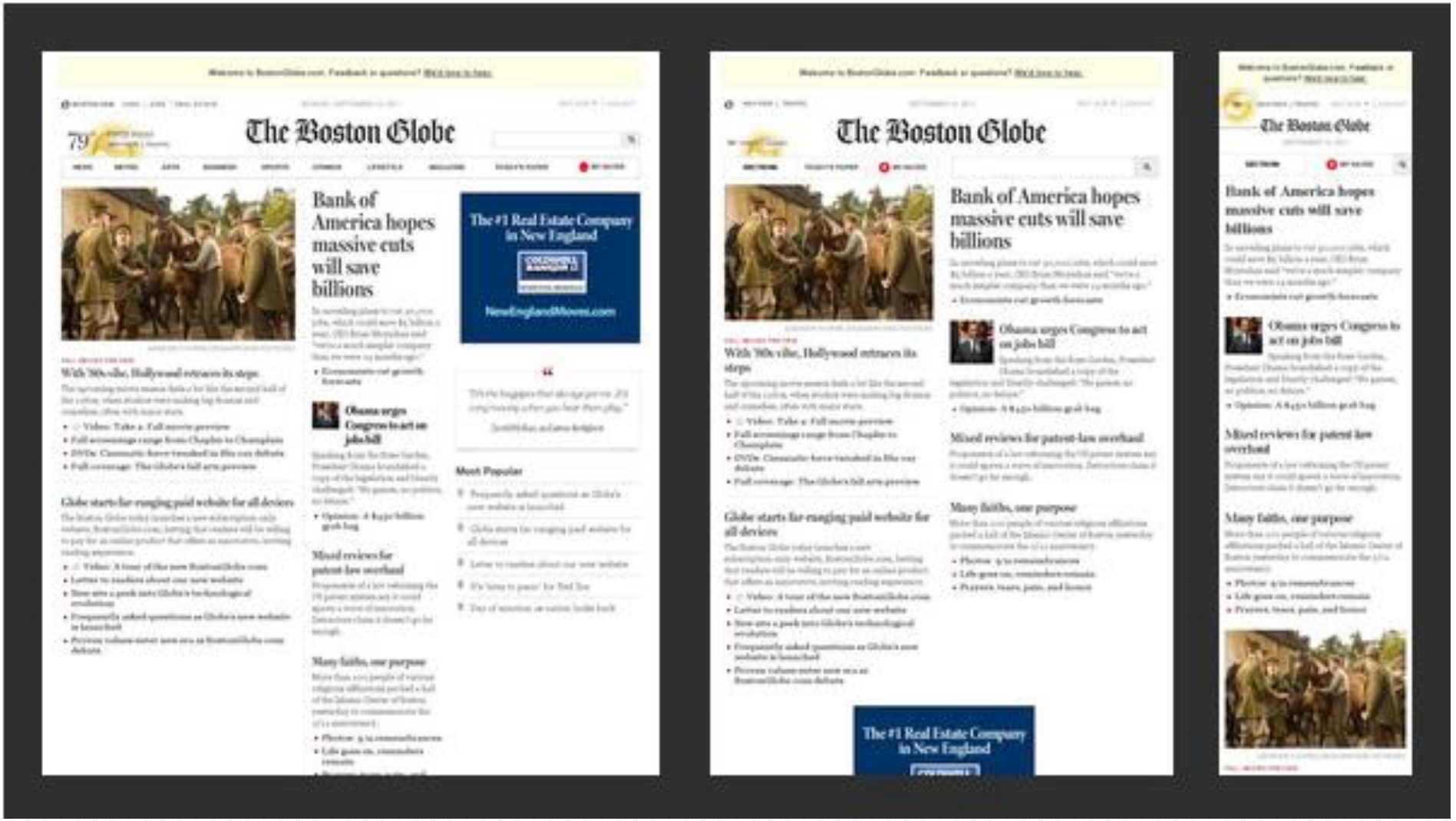
# Diseño Web Responsive



- Ventajas de utilizar un diseño web adaptativo:
  - **Eficiencia en la actualización**. El sitio solo se debe actualizar una vez y se ve reflejada en todas las plataformas. Cuando tenemos los portales independientes para Web y Mobile se debe realizar la actualización dos veces lo que crea la necesidad de mayor cantidad de recursos y posibilidad de error.
  - Desde el punto de vista de la **optimización de motores de búsqueda**, sólo aparecería una URL en los resultados de búsqueda, con lo cual se ahorran redirecciones y los fallos que se derivan de éstas. También se evitarían errores al acceder al sitio web en concreto desde los llamados *social links*, es decir, desde enlaces que los usuarios comparten en medios sociales tales como Facebook, Twitter, etc y que pueden acabar en error dependiendo de qué enlace se copió (desde qué dispositivo se copió) y desde qué dispositivo se accede.

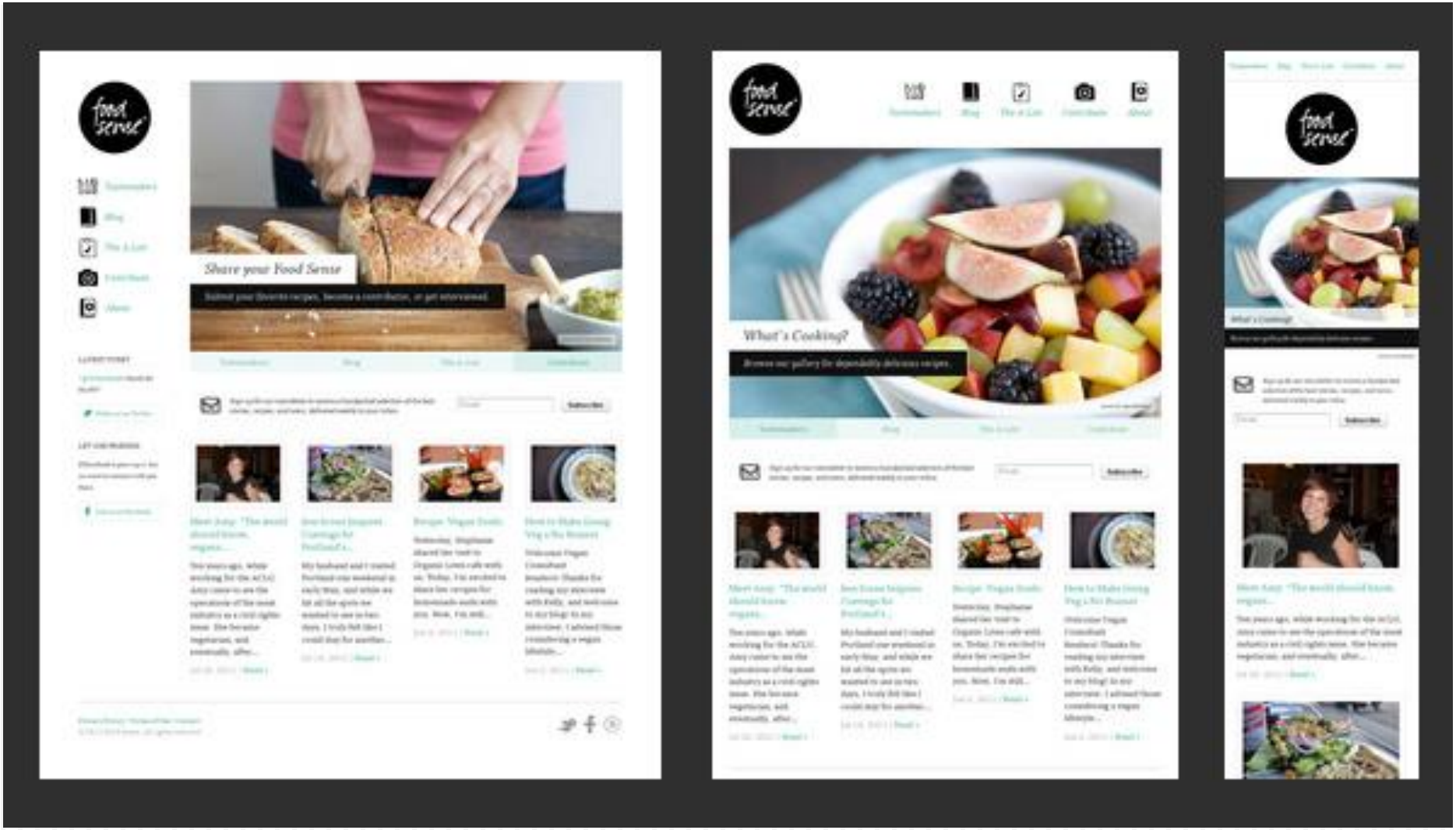
# Diseño Web Responsive





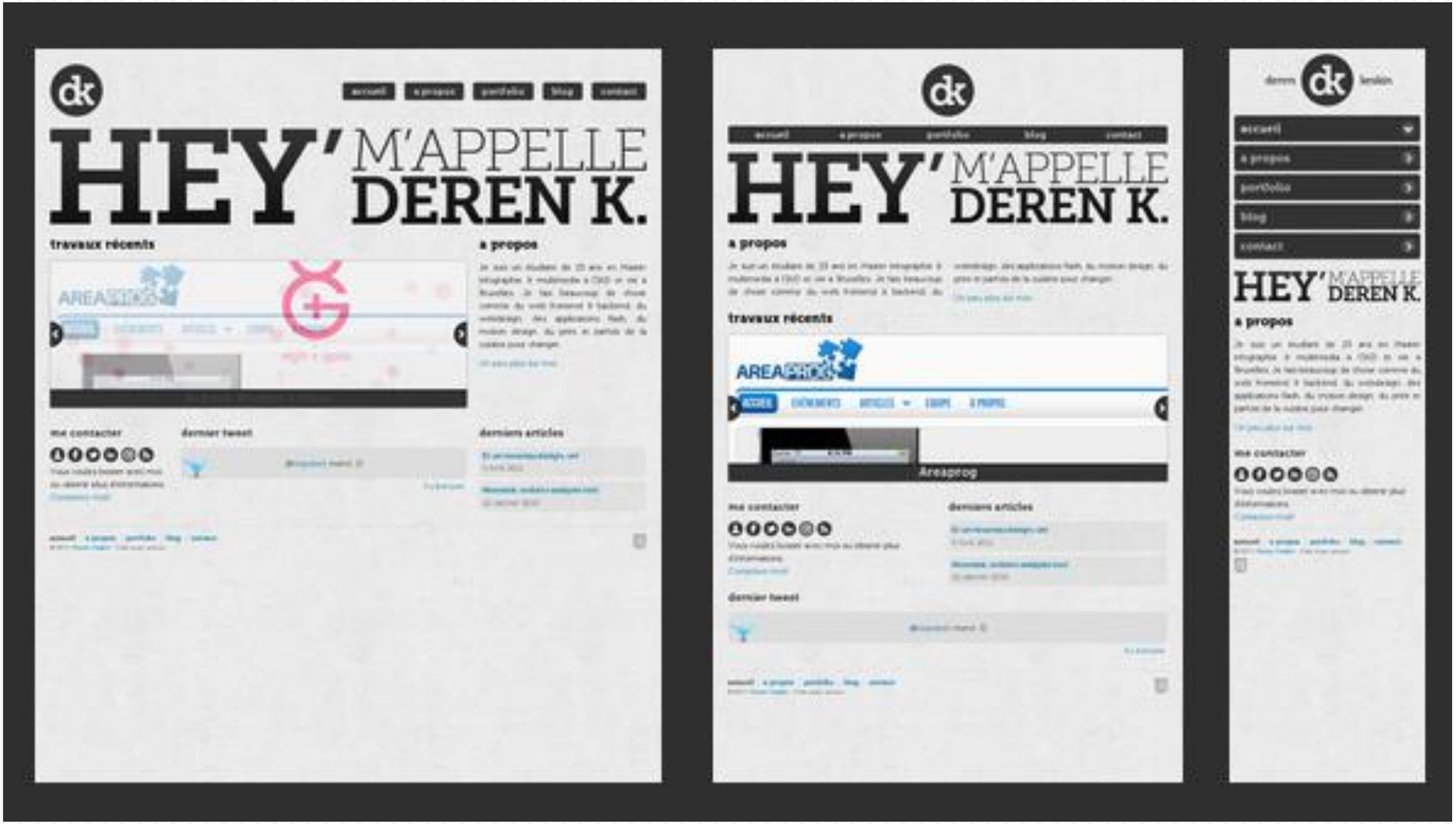
# Diseño Web Responsive





# Diseño Web Responsive





# Diseño Web Responsive

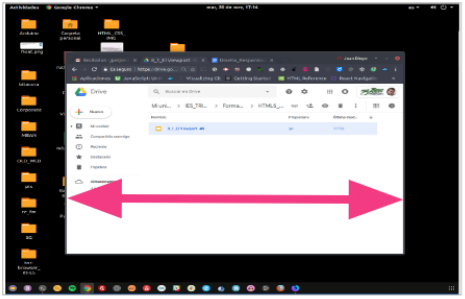


- Opciones para hacer una Web responsive
  - Bibliotecas
    - BootStrap
    - Foundation
    - Angular Material
  - CSS/JS
    - Una estructura flexible, basada en rejilla.
    - Imágenes y multimedia flexible.
    - *Media queries*, como parte de la especificación de CSS 3.

# Diseño Web Responsive



- Uno de los conceptos más importantes que debemos conocer antes de lanzarnos a realizar diseños responsivos es el concepto de ***Viewport***.
- Una posible definición sería:
  - *Área de la pantalla en la que el navegador puede renderizar contenido, es decir, el espacio disponible para mostrar mi página web.*
- Este concepto es muy fácil de comprender cuando nos referimos a él en sistemas de escritorios o en laptops o portátiles. En estos casos el **Viewport** coincide con la pantalla de nuestro navegador.



# Diseño Web Responsive



- Sin embargo, si nos referimos a móviles y tablets estaremos hablando de otra cosa diferente.
- Cuando este tipo de dispositivos irrumpieron las páginas se maquetaban usando normalmente una anchura fija en píxels que solía variar entre 800px y 1000px que, obviamente, era mayor que la anchura de la pantalla de los móviles.
- Ante esta circunstancia los fabricantes hicieron que este tipo de dispositivos tuvieran dos **Viewports:**
- El ***Layout-viewport*** que es el viewport que es tenido en cuenta para la aplicación de estilos.Suele ser de aproximadamente 900px.
- El ***Visual-viewport*** que es el viewport que realmente ve el usuario cuando está navegando.

# Diseño Web Responsive



# Diseño Web Responsive



- Además, sucede que en este tipo de dispositivos podemos hacer Zoom mediante gestos lo que provoca que:
  - No cambie el ***Layout-viewport***
  - Cambie el ***Visual-viewport***.
- Y no acaban ahí los problemas. También, en este mundo de miles de dispositivos diferentes no vamos a encontrar con que tenemos que lidiar con diferentes valores para los:
  - **Hardware Pixels:** que son los pixels de resolución que nos da la tarjeta gráfica.
  - **DPI(Device Independent Pixels):** Que es la unidad de medida del navegador. Se relaciona con la distancia real y ocupan lo mismo independientemente de la densidad de pixels de la pantalla.
  - **DPR(Device Pixel Ratio):** Hw Pixel / DPI (es una dimensión)

# Diseño Web Responsive



- Esto, todo junto, es un jaleo y hay multitud de teoría detrás. Sin embargo, lo que necesitamos para empezar a gestionar toda esta situación es simplemente añadir una línea en la cabecera de mi página Web.
- Con esa simple línea podremos empezar a maquetar diseños responsivos de manera eficiente e incluso haremos que las páginas que no están preparadas para móviles se muestren correctamente aunque tengamos, eso sí, que hacer continuos ***zooms*** para poder usarlas.
- Si estáis interesados en profundizar más hay un vídeo muy interesante y aclaratorio (de 44min) sobre el ***Viewport***.
- **Enlace al vídeo**.

```
<head>
  ...
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  ...
</head>
```

# Diseño Web Responsive



- Hay clasificaciones más complejas pero si condensamos podemos establecer que hay tres tipos de diseños o layouts a la hora de hacer maquetación web.
  - **Fixed:** Donde la anchura de la página es fija y es expresada en pixels.
  - **Elastic:** Donde la anchura de la página es fija y es expresada en la unidad **em** (múltiplos del tamaño de letra).
  - **Fluid/Liquid/Relative:** Donde la anchura de la página depende del tamaño del Viewport del usuario y se expresa en porcentajes (%).
- A continuación vamos a explicar de manera concisa las ventajas y desventajas de cada uno de estos tipos.

# Diseño Web Responsive



- **Fixed Layout**
  - **Ventajas:** Tenemos un control total sobre el layout ya que especificamos dimensiones fijas.
  - **Desventaja:** Puede aparecer un scroll horizontal en pantallas pequeñas (si no escalo).
  - Sin embargo, este tipo de Layouts puede tener sentido si no quiero cambiar nunca el layout ni sus proporciones en ningún dispositivo (no suele ser el caso para las pantallas de los móviles).

# Diseño Web Responsive



- **Elastic Layout**
  - **Ventajas:** Control al hacer zoom-in y zoom-out. Se mantienen las proporciones.
  - **Desventajas:**
  - Al final es igual que fixed.
  - Es difícil calcular las dimensiones reales.
  - Ems se calcula en relación al padre, podemos necesitar rems.
  - No hay fluidez cuando cambia el tamaño de pantalla. No se adapta bien.

# Diseño Web Responsive



- **Breakpoints**
- Una vez hemos establecido que usaremos diseños de tipo ***Fluid*** debemos recordar que el concepto de diseño responsivo, además de ***adaptación*** al tamaño implicaba el ***cambio*** en el diseño atendiendo al tamaño u otras características del dispostivo.
- La anchura de la pantalla en la que se produce el cambio es lo que se conoce como ***Breakpoint***.
- La elección de unos ***breakpoints*** adecuados no es una tarea fácil pero tenemos la suerte de que ya ha habido empresas que han hecho estudios muy serios al respecto. Destacar los Breakpoints elegidos por **Twitter BootStrap**:
  - <576px (pantallas pequeñas)
  - 576px-768px (móviles apaisados)
  - 768px-992px (tablets)
  - 992px-120ppx (desktop)
  - >1200px (pantallas grandes)

# Diseño Web Responsive



- MEDIA QUERIES
  - Desde la especificación de CSS 2.1, ha sido posible modificar el aspecto de los documentos HTML en función del tipo de dispositivo en el que se mostraban. El caso más común es el de crear una hoja de estilos que se aplica al imprimir los documentos:
    - <link rel="stylesheet" type="text/css" href="core.css" media="screen" />
    - <link rel="stylesheet" type="text/css" href="print.css" media="print" />
  - Como no sólo vamos a formatear nuestros documentos para que se muestren de manera correcta, la especificación CSS ofrece una buena cantidad de tipos de medios (media type) para los que podemos aplicar un diseño específico, concretamente los siguientes: all, braille, embossed, handheld, print, projection, screen, speech, tty, y tv..

# Diseño Web Responsive



- MEDIA QUERIES
  - El problema es existe una extensa variedad de dispositivos que comparten el mismo tipo de medio, pero son completamente diferentes entre sí.
  - Por suerte, la W3C introdujo los Media Query como parte de la especificación de CSS 3, mejorando de manera notable el objetivo de los Media Type. Un Media Query no sólo nos permite seleccionar el tipo de medio, sino consultar otras características sobre el dispositivo que esta mostrando la página.

# Diseño Web Responsive



- MEDIA QUERIES
  - Un simple ejemplo de Media Query:

```
<link rel="stylesheet" type="text/css"
      media="screen and (max-device-width: 480px)"
      href="shetland.css" />
```

  - Esta consulta contiene dos componentes:
    - Un tipo de medio (screen)
    - y una consulta concreta sobre la característica del medio (max-device-width) y el valor objetivo (480px).
  - En otras palabras, estamos preguntando al dispositivo si su resolución horizontal (max-device-width) es igual o menor a 480px. Si se cumple la condición, el dispositivo cargará la hoa de estilos shetland.css. De otra manera, el link será ignorado.

# Diseño Web Responsive



- MEDIA QUERIES
  - Los Media Query también pueden ser definidos dentro de la propia hoja de estilos:

```
@media screen and (min-width: 1024px) {
  body  {
  font-size: 100%;
  }
}
```

  - O incluso utilizando la sentencia @import:

```
@import url("wide.css") screen and (min-width: 1024px);
```

  - No importa la manera en la que se defina el Media Query, el resultado debe ser el mismo: si el navegador cumple con el tipo de medio y las condiciones indicadas, se aplicarán las reglas CSS definidas.

# Diseño Web Responsive



- MEDIA QUERIES - SINTAXIS
  - Los Media Queries pueden contener una o más expresiones, expresadas como funciones multimedia, que se resuelven en true o false. El resultado del query o consulta devuelve true si el media type especificado en el Media Query coincide con el tipo de dispositivo en que el documento está siendo mostrado y todas las expresiones en el Media Query devuelven true.
  - Cuando un Media Query devuelve true, la correspondiente hoja de estilo es aplicada, siguiente las reglas habituales de CSS. Las hojas de estilo con Media Queries adjuntos a los tags <link> seguirán descargándose, incluso si sus Media Queries resultan false (sin embargo, no se aplicarán).

# Diseño Web Responsive



- MEDIA QUERIES – SINTAXIS- OPERADORES LÓGICOS O LOGICAL OPERATORS
  - Se pueden crear Media Queries complejos utilizando logical operators, incluyendo not, and y only. El operador and es usado para combinar múltiples media features en un sólo Media Query, requiriendo que cada función devuelve true para que el Query también lo sea. El operador not se utiliza para negar un Media Query completo y el operador only se usa para aplicar un estilo sólo si el Query completo es correcto.
  - Además, se pueden combinar múltiples Media Queries separados por comas en una lista; si alguno de los Queries devuelve true, todo el *media statement devolverá true. Esto es equivalente a un operador or.

# Diseño Web Responsive



- MEDIA QUERIES – SINTAXIS- OPERADORES LÓGICOS O LOGICAL OPERATORS-AND
  - El keyword and se usa para combinar múltiples media features, así como combinar éstos con media types. Un Media Query básico sería:
    - @media (min-width: 700px) { ... }
  - Sin embargo, si quisiéramos que esto sólo se aplicara si la pantalla está en modo landscape, se usaría el operador and:
    - @media (min-width: 700px) and (orientation: landscape) { ... }
  - Si además, sólo quisiéramos que esto se aplicara si el dispositivo fuera un media type TV:
    - @media tv and (min-width: 700px) and (orientation: landscape) { ... }

# Diseño Web Responsive



- MEDIA QUERIES – SINTAXIS- OPERADORES LÓGICOS O LOGICAL OPERATORS - COMMA-SEPARATED LISTS
  - Cuando se utilizan las listas separadas por comas en los Media Queries, si algunas de las Media Queries devuelven true, los estilos se aplican. Cada Media Query separado por comas en la lista se trata como un Query individual, y cualquier operador aplicado a un Media Query no afecta a los demás. Esto significa que los Media Queries separados por comas puede dirigirse a diferentes media features, types o states.
  - Por ejemplo, si quisiéramos aplicar un conjunto de estilos si el dispositivo de visualización tienen un mínimo de 700px o está en modo landscape:
    - @media (min-width: 700px), handheld and (orientation: landscape) { … }

# Diseño Web Responsive



- MEDIA QUERIES – SINTAXIS- OPERADORES LÓGICOS O LOGICAL OPERATORS - COMMA-SEPARATED LISTS
  - Cuando se utilizan las listas separadas por comas en los Media Queries, si algunas de las Media Queries devuelven true, los estilos se aplican. Cada Media Query separado por comas en la lista se trata como un Query individual, y cualquier operador aplicado a un Media Query no afecta a los demás. Esto significa que los Media Queries separados por comas puede dirigirse a diferentes media features, types o states.
  - Por ejemplo, si quisiéramos aplicar un conjunto de estilos si el dispositivo de visualización tienen un mínimo de 700px o está en modo landscape:
    - @media (min-width: 700px), handheld and (orientation: landscape) { … }

# Diseño Web Responsive



- MEDIA QUERIES – SINTAXIS- OPERADORES LÓGICOS O LOGICAL OPERATORS - NOT
  - La keyword not se aplica al Media Query en su totalidad y devuelve true si el Media Query devuelve false (como monochrome en una pantalla a color). Este keyword no se puede utilizar para negar un individual feature query, solamente un entire media query. Por ejemplo:
    - @media not all and (monochrome) { ... }
  - Esto significa que el Query es evaluado de esta manera:
    - @media not (all and (monochrome)) { ... }
  - en vez de así:
    - @media (not all) and (monochrome) { ... }
  - Por ejemplo, este otro Media Query:
    - @media not screen and (color), print and (color)
  - Se evalúa así:
    - @media (not (screen and (color))), print and (color)

# Diseño Web Responsive



- MEDIA QUERIES – SINTAXIS- OPERADORES LÓGICOS O LOGICAL OPERATORS - ONLY
  - El keyword only previene a los navegadores que no soportan Media Queries:
    - <link rel="stylesheet" media="only screen and (color)" href="example.css" />

# Diseño Web Responsive



- MEDIA QUERIES – CARACTERÍSTICAS DE LOS MEDIOS
  - La especificación de los Media Query incluye una larga lista de características que podemos consultar sobre el dispositivo. En este especificación, se hace referencia a dos términos que hay que tener claros:
    - display area: espacio disponible en la ventana del navegador para mostrar el contenido de la página web.
    - rendering sourface: hace referencia al espacio total disponible en el dispositivo.
  - El listado completo de características es el siguiente:

# Diseño Web Responsive



- MEDIA QUERIES – CARACTERÍSTICAS DE LOS MEDIOS

| Característica | Definición | Tiene prefijo `min-` y `max-` |
|---|---|---|
| `width` | El ancho del área de visualización (`display area`) | Sí |
| `height` | El alto del área de visualización (`display area`) | Sí |
| `device-width` | El ancho total del dispositivo (`rendering sourface`) | Sí |
| `device-height` | El alto total del dispositivo (`rendering sourface`) | Sí |
| `orientation` | Acepta los valores `portrait` o `landscape` | No |
| `aspect-ratio` | Relación de aspecto entre el ancho y alto del área de visualización | Sí |
| `device-aspect-ratio` | Relación de aspecto entre el ancho y alto del dispositivo | Sí |
| `color` | El número de `bits` de profundidad de color | Sí |
| `color-index` | El número de entradas en la tabla de colores del dispositivo | Sí |
| `monocrome` | El número de `bits` de profundidad de color, en dispotivos monocromáticos | Sí |
| `resolution` | Densidad de `pixels` en el dispositivo, medido en `dpi` | Sí |

# Diseño Web Responsive



- MEDIA QUERIES – RESUMEN

```
@media mediatype [condiciones] {
  .... /* Estilos para esas condiciones */ ....;;
}
```

- Los distintos tipos de valores que puedo tener para ***mediatype*** son:
  - all
  - screen (el más usado)
  - tty (terminal)
  - print (para impresoras)
  - tv (para pantallas de televisión)
  - projected (para proyectar contenidos)
  - braille (para dispositivos para invidentes)
  - etc…

# Diseño Web Responsive



- MEDIA QUERIES – RESUMEN
- En cuanto a las condiciones que puedo consultar:
  - width | min-width | max-width
  - height | min-height | max-height
  - orientation (landscape / portrait)
  - aspect-ratio | min-aspect-ratio | max-aspect-ratio
  - color | min-color | max-color

# Diseño Web Responsive



- MEDIA QUERIES – RESUMEN
- Esta condiciones se pueden combinar y modificar utilizando claúsulas como:
  - And
  - Not
  - All
  - Only

# Diseño Web Responsive



- MEDIA QUERIES – RESUMEN

```
/* Estilos para todo tipo de pantallas con una anchura máxima
de 576px*/
@media all and (max-width: 576px) {
  .....;
}

/* Estilos para pantallas con al menos 992px de anchura y que estén apaisadas (más ancho que alto)*/
@media screen and (min-width: 992px) and (orientation: landscape) {
  ......;
}

/* Estilos sólo para pantallas que tengan al menos 768px de anchura*/
@media only screen and (min-width: 768px) {
  ......;
}
```

# Diseño Web Responsive



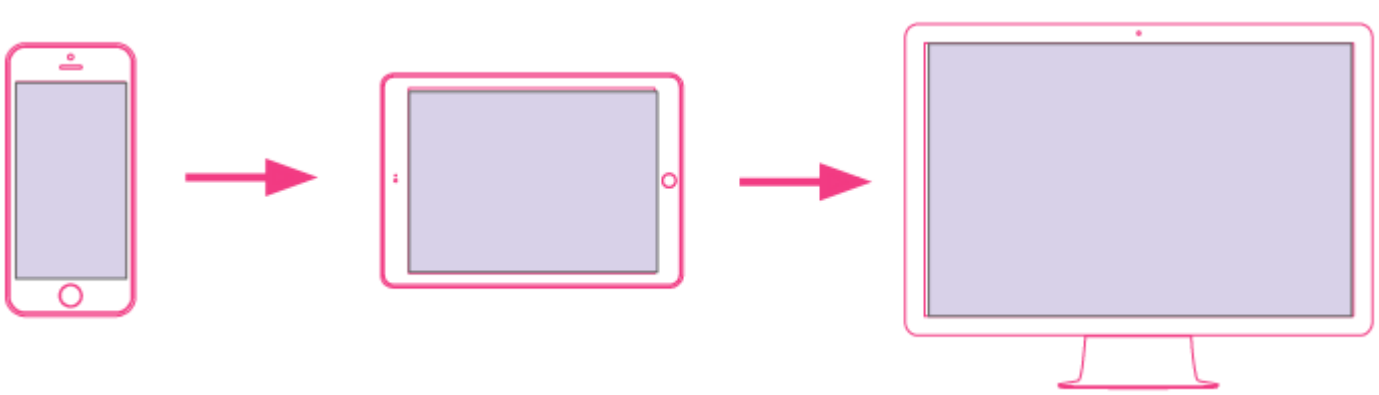
- MEDIA QUERIES – RESUMEN
- Podemos usar media-queries en la etiqueta link para seleccionar hojas de estilos diferentes atendiendo a las características del dispositivo.

```
<!--Para pantallas de hasta 576px de ancho -->
<link rel="stylesheet" media="(max-width: 576px)" href="small.css" />

<!--Para pantallas entre 576px y 768px de ancho -->
<link rel="stylesheet" media="(max-width: 768px)" href="medium.css" />

<!--Para pantallas entre 768px y 992pxde ancho -->
<link rel="stylesheet" media="(max-width: 992px)" href="large.css" />
```

# Diseño Web Responsive



- PATRONES RESPONSIVE
  - Afrontar un proyecto que requiera el desarrollo de una página web responsiva no es fácil y hay varias estrategias posibles para afrontarlo. No obstante ya es comunmente aceptado que lo más recomendable es lo siguiente:



# Diseño Web Responsive



- PATRONES RESPONSIVE
  - Es decir, que el proceso de diseño de nuestra página web debe empezar haciendo que todo quede correctamente en pantallas pequeñas. De este manera:
  - **Priorizo** siempre lo que es importante. Si el proceso fuera al revés es muy fácil que caigamos en el error de quitar cosas que sean realmente importantes.
  - Me pregunto en cada diseño si es **necesario un diseño nuevo** para paltallas más grandes.
  - Elijo los **breakpoints** más adecuados.

# Diseño Web Responsive



- PATRONES RESPONSIVE
- En nuestro proceso de diseñar web responsivas debemos conocer lo que se conoce como ***patrones responsivos***.
- Los ***patrones responsivos*** son soluciones que se han dado por buenas para el problema de diseñar páginas web responsivas. Hay muchos pero los más comunes son lo siguientes:
  - Column Drop
  - Mostly Fluid
  - Layout Shifter
  - Off Canvas
  - …
  - Mezclar varios de ellos
  - Pequeños ajustes (Tiny Tweaks)

# Diseño Web Responsive



- PATRONES RESPONSIVE
  - **Column Drop**
    - Es el patrón más básico y consiste en que en cada breakpoint se va a apilando un elemento
  - **Mostly Fluid**
    - Parecido a Column Drop. Es una cuadrícula fluida y en cada breakpoint hay redimensionamiento de varias columnas.
  - **Layout Shifter**
    - Es el patrón más responsivo, en cada breakpoints cambia el diseño del layout, no únicamente el flujo y la anchura de los elementos.
  - **Off Canvas**
    - En vez de apilar contenidos éstos se colocan fuera de la pantalla cuando el tamaño de pantalla no es lo suficientemente grande.

# Diseño Web Responsive



- CREAR UNA REJILLA FLEXIBLE
  - Mientras que los media query ofrecen la potencia real para desarrollar un sitio web adaptativo, es posible ahorrar trabajo y código utilizando una aproximación en base a rejillas flexibles.
  - Utilizando este tipo de rejillas, podemos asegurar que el sitio web se redimensiona en función del espacio disponible, sin la necesidad de utilizar media query. Posteriormente es posible utilizar media query si es necesario realizar cambios significativos en la estructura de la web.

# Diseño Web Responsive



- CREAR UNA REJILLA FLEXIBLE
- Vamos a ver seix componentes que nos permitirán construir una rejilla flexible, y cómo utilizarlas:
  - Texto flexible
  - Tablas flexibles
  - Contenedores flexibles
  - Márgenes flexibles
  - Rellenos flexibles
  - Imágenes flexibles

# Diseño Web Responsive



- CREAR UNA REJILLA FLEXIBLE - TEXTI FLEXIBLES
  - El método tradicional utilizado para dar estilo a las fuentes es utilizar pixels.
  - Esto da un control exacto a los diseñadores a la hora de definir su tamaño, pero lo ideal es comenzar a pesar de manera proporcional.
  - Utiliza valores proporcionales a la hora de definir los tamaños del texto, nos ahorrará mucho tiempo y líneas de código en nuestros media query.
  - Si deseamos aumentar o reducir el tamaño de nuestro texto, con indicar el tamaño de letra del elemento padre sería suficiente, ya que el resto de elementos se verían afectados de manera inmediata.

# Diseño Web Responsive



- CREAR UNA REJILLA FLEXIBLE - TEXTO FLEXIBLE
  - La manera correcta de conseguir este comportamiento es utilizar medidas basadas en em. Además, debemos aplicar una regla muy sencilla para calcular el tamaño de letra concreto para cada caso:
    - target ÷ context = result
  - Donde target es el tamaño de fuente que queremos mostrar y context es el tamaño de fuente base de nuestro navegador (generalmente son 16px para la mayoría de navegadores).
  - Así pues, si deseamos mostrar un texto con un tamaño de 32px, su tamaño en em se calcularía de la siguiente manera:
    - 32px ÷ 16px = 2
  - Al utilizar esta fórmula, lo normal es obtener un número con mucho decimales. Lo ideal es mantener estos decimales, ya que el navegador es capaz de interpretarlos y siempre será mucho más preciso que no disponer de ellos. Además, es recomendable indicar el cálculo realizado junto a la regla de CSS, como referencia a futuro y en caso de necesitar cambiarlo.
    - font-size: 2em /* 32px/16px */

# Diseño Web Responsive



- CREAR UNA REJILLA FLEXIBLE - TEXTO FLEXIBLE
- Cuando hablamos de diseño responsivo solemos dar más importancia al layout, pero el texto que vamos a presentar es igual de importante para conseguir un buen diseño.
- Si no tenemos cuidado nos podemos en contrar con varios problemas:
  - Líneas cortas con pocos caracteres, lo que dificultan la lectura.
  - Líneas largas con muchas caracteres, lo que también dificulta la lectura.
  - Caracteres muy pequeños.
- Lo ideal, según estudios, es una letra de un tamaño adecuado, para poder leer sin forzar la vista, y que se disponga formando líneas de entre 60-80 caracteres.

# Diseño Web Responsive



- CREAR UNA REJILLA FLEXIBLE - TEXTO FLEXIBLE
- Para intentar solucionar esto con garantías lo más recomendable es utilizar para el texto unidades de tamaño relativas al viewport. Tendremos:
  - **vw** en relación a la anchura del viewport.
  - **vh** en relación a la altura del viewport.
  - **vmin** el valor menor en relación a la dimensión pequeña del viewport (anchura o altura).
  - **vmax** el valor máximo en relación a la dimensión más grande del viewport (anchura o altura).
- Teniendo en cuenta, por ejemplificar que 1vw es un 1% de la anchura del viewport.

# Diseño Web Responsive



- CREAR UNA REJILLA FLEXIBLE - TEXTO FLEXIBLE
- No obstante encontrar siempre el tamaño perfecto de texto es algo que puede ser complicado sino imposible. Sin embargo allí van unas posibles pautas:
  - Calcular el tamaño mínimo y máximo que me puedo permitir usando calc() (función css)
  - No importa si el texto hace “reflow” (pasa a la siguiente línea) en determinadas ocasiones.
  - Mantener el tamaño perfecto de línea puede que sea imposible.
  - Si tenemos que elegir es conveniente priorizar móviles y tablets.
  - En algunos elementos, por ejemplo en un menú de navegación, puede tener sentido usar tamaño fijos de letra en vez de unidades relativas al viewport.

# Diseño Web Responsive



- CREAR UNA REJILLA FLEXIBLE - TABLAS FLEXIBLES
- Las tablas son un elemento problemático a la hora de realizar **Diseño Responsivo** ya que en cuanto tienen un número de columnas considerable pueden provocar la aparición del tan temido *scroll horizontal* en nuestra página web, sobre todo en pantallas pequeñas.
- Para afrontar este tipo de problemas hay tres soluciones que son aceptadas como buenas:
  - Esconder columnas.
  - Convertir las filas en listas
  - Crear un scroll horizontal que solo se aplique a la tabla.

# Diseño Web Responsive



- CREAR UNA REJILLA FLEXIBLE - TABLAS FLEXIBLES
- **Esconder columnas**
- Esta técnica consiste básicamente en esconder ciertas columnas, que debe de ser las menos importantes, cuando el tamaño de la pantalla es menor que un breakpoint establecido.
- Tiene las siguiente ventajas:
  - Conseguimos un diseño responsivo.
  - Priorizo el contenido que quiero mostrar.
- Aunque también tiene desventajas:
  - Pierdo información en determinadas pantallas.

# Diseño Web Responsive



- CREAR UNA REJILLA FLEXIBLE - TABLAS FLEXIBLES
- **Esconder columnas**
  - Un ejemplo para conseguir esto sería una hoja de estilos similar a esta.
  - En ese diseño tenemos dos breakpoints y ocultaremos unas u otras columnas dependiendo del tamaño (la que tienen la clase .primero la clase .segundo)

```
@media screen and (max-width: 576px) {
  .hidden td.primero,
  .hidden th.primero {
    display: none;
  }
}

@media screen and (max-width: 768px) {
  .hidden td.segundo,
  .hidden th.segundo {
    display: none;
  }
}
```

# Diseño Web Responsive



- CREAR UNA REJILLA FLEXIBLE - TABLAS FLEXIBLES
- **Convertir Listas a filas**
- Esta técnica consiste en hacer desaparecer las cabeceras de la tabla cuando la pantalla es menor que una determinada cantidad y hacer que todas las celdas se conviertan en elementos de bloque para que se muestren una debajo de otra y no al lado.
- Tiene las siguiente ventajas:
  - Conseguimos un diseño responsivo.
  - No pierdo información.
- Aunque también tiene desventajas:
  - No priorizo la información.
  - “Desplazo” mucho el resto del layout hacia abajo.

# Diseño Web Responsive



- CREAR UNA REJILLA FLEXIBLE - TABLAS FLEXIBLES
- **Scroll controlado**
- Esta técnica consiste en acotar el scroll horizontal para que si ha de aparecer solo afecte a la tabla y no a la página entera.
- Tiene las siguiente ventajas:
  - Conseguimos un diseño responsivo.
  - No pierdo información.
- Aunque también tiene desventajas:
  - No priorizo la información.
  - Aunque local sigue habien un scroll horizontal.

# Diseño Web Responsive



- CREAR UNA REJILLA FLEXIBLE - TABLAS FLEXIBLES
- **Scroll controlado**
- Para conseguir esto debemos "envolver" la tabla en un contenedor y darle las siguiente propiedades:

```
<div class="localscroll">
  <table>
    .....
  </table>
</div>
```

```
div.localscroll {
  overflow-x: auto;
  width: 100%;
}
```

- No obstante no hay una solución universal. Debemos experimentar según el Layout que tengamos para ver cuál de estas técnicas (o la mezcla de ellas) se adecúa mejor a nuestro diseño.

# Diseño Web Responsive

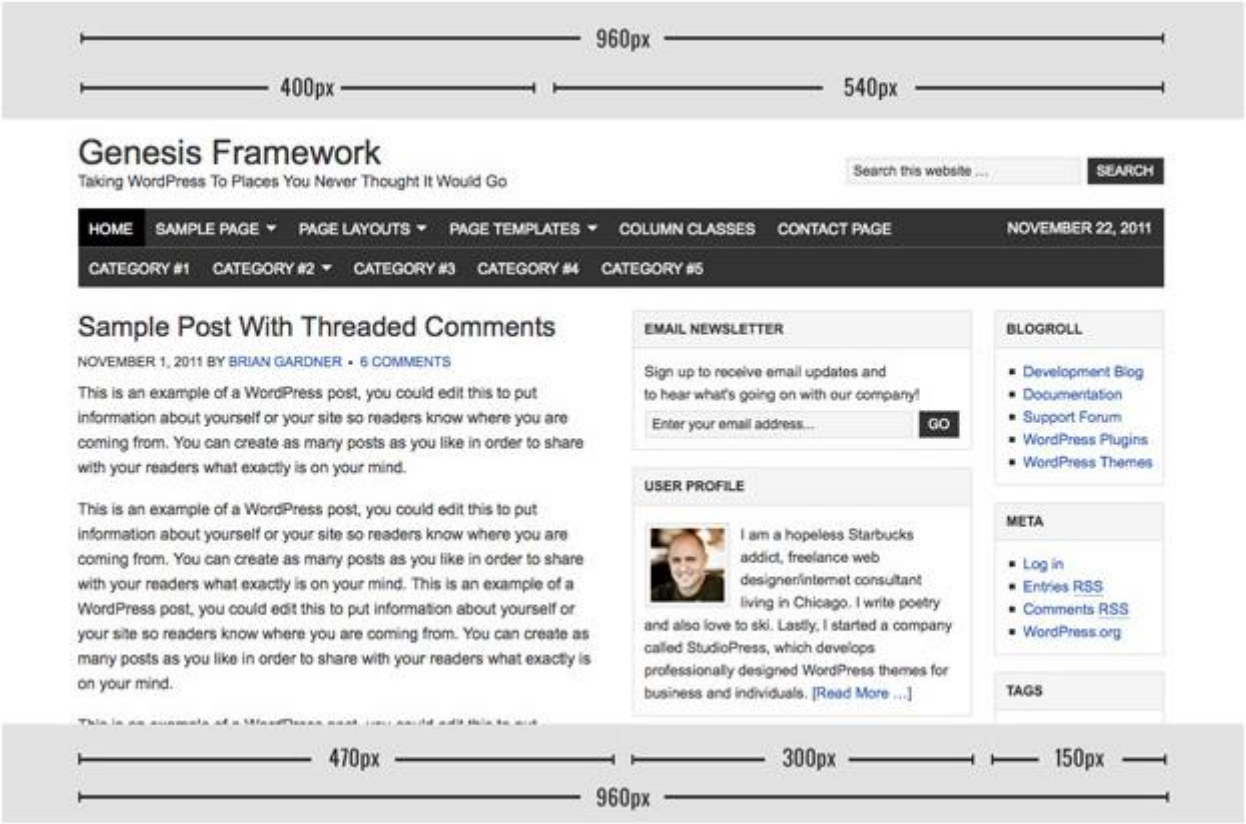


- CREAR UNA REJILLA FLEXIBLE - CONTENEDORES FLEXIBLES
  - La magia de la fórmula anterior es que no sólo se puede aplicar a tamaños de fuente, sino que funciona correctamente para crear la estructura de una web a través de etiquetas div.
  - Utilicemos como ejemplo el tema Genesis de Wordpress. En la siguiente imagen se puede ver que se ha optado por un diseño de 960pxde ancho, dividido en una cabecera con dos contenedores y un cuerpo con tres contenedores, ambos separados por un espacio de 20px.

# Diseño Web Responsive



- CREAR UNA REJILLA FLEXIBLE - CONTENEDORES FLEXIBLES



# Diseño Web Responsive



- CREAR UNA REJILLA FLEXIBLE - CONTENEDORES FLEXIBLES
- Este es el HTML correspondiente a la estructura:

```
<div id="wrap">
    <div id="header">
        <div id="title-area"></div>
        <div class="widget-area"></div>
    </div>
    <div id="inner">
        <div id="content-sidebar-wrap">
            <div id="content"></div>
            <div id="sidebar"></div>
        <div id="sidebar-alt">
    </div>
</div>
```

# Diseño Web Responsive



- CREAR UNA REJILLA FLEXIBLE - CONTENEDORES FLEXIBLES
- Este es el CSS:

```
#wrap {
    width: 960px;
}

#header {
    width: 960px;
}

#title-area {
    width: 400px;
}

#header .widget-area {
    width: 540px;
}

#inner {
    width: 960px;
}
```

```
#content-sidebar-wrap {
    width: 790px;
}

#content {
    width: 470px;
}

#sidebar {
    width: 300px;
}

#sidebar-alt {
    width: 150px;
}
```

# Diseño Web Responsive



- CREAR UNA REJILLA FLEXIBLE - CONTENEDORES FLEXIBLES
  - Conociendo los valores objetivo, la primera tarea es establecer nuestro ancho base. Esta medida hace referencia al ancho que hemos definido a nuestro layout sin aplicar ningún diseño adaptativo. Como medida general, suele adoptarse como ancho base un 90% del área visible de la pantalla, lo que permite seguir manteniendo toda la estructura y contenidos de la web siempre visibles.
  - A la hora de aplicar la fórmula, la única diferencia es que en este caso no vamos a hacer uso de la unida de medida em, sino de porcentajes, por lo que debemos multiplicar el resultados por 100. Si aplicamos la fórmula a la estructura anterior, el resultado sería el siguiente:

# Diseño Web Responsive



- CREAR UNA REJILLA FLEXIBLE - CONTENEDORES FLEXIBLES

```
#wrap {
    width: 960px;
}

#header {
    width: 960px;
}

#title-area {
    width: 400px;
}

#header .widget-area {
    width: 540px;
}

#inner {
    width: 960px;
}
```

```
#wrap {
    width: 90%;
    max-width: 960px;
}

#header {
    width: 100%; /* 960px/960 */
}

#title-area {
    width: 41.666667% /* 400px/960 */;
}

#header .widget-area {
    width: 56.25% /* 540px/960 */;
}

#inner {
    width: 100% /* 960px/960 */;
}
```

# Diseño Web Responsive



- CREAR UNA REJILLA FLEXIBLE - CONTENEDORES FLEXIBLES

```
#content-sidebar-wrap {
    width: 790px;
}

#content {
    width: 470px;
}

#sidebar {
    width: 300px;
}

#sidebar-alt {
    width: 150px;
}
```

```
#content-sidebar-wrap {
    width: 82.291667% /* 790px/960 */;
}

#content {
    width: 48.958333% /* 470px/960 */;
}

#sidebar {
    width: 31.25% /* 300px/960 */;
}

#sidebar-alt {
    width: 15.625% /* 150px/960 */;
}
```

- Ahora que tenemos nuestros contenedores flexibles, el siguiente paso es obtener el mismo comportamiento para los márgenes y rellenos.

# Diseño Web Responsive



- CREAR UNA REJILLA FLEXIBLE - MÁRGENES FLEXIBLES
  - Para determinar la medida de los márgenes, vamos a utilizar la anchura del elemento contenedor.
  - Tomemos como ejemplo la barra lateral. En este caso, el margen lateral izquierdo está definido a 25px, mientras que el ancho del contenedor es de 150px. Por lo tanto, la manera de calcular el nuevo margen es dividir el 25px (target) entre 150px (context), dando como resultado 16.666667%

# Diseño Web Responsive



- CREAR UNA REJILLA FLEXIBLE - MÁRGENES FLEXIBLES

```
.widget-area ul {
    margin: 10px 0 0 25px;
}

.widget-area ul {
    margin: 10px 0 0 16.666667%;
}
```

# Diseño Web Responsive



- CREAR UNA REJILLA FLEXIBLE - RELLENOS FLEXIBLES
  - En este caso, el comportamiento es exactamente igual al utilizado para calcular la media de los márgenes. Dividimos el relleno actual, 10px, entre la anchura del elemento contenedor 300px, dando como resultado un relleno del 3.33333%.

```
.enews p {
    padding: 5px 10px 0;
}

.enews p {
    padding: 5px 3.33333% 0;
}
```

# Diseño Web Responsive



- CREAR UNA REJILLA FLEXIBLE - IMÁGENES FLEXIBLES
- Las imágenes son un elementos **fundamental** de todas las páginas y representan una gran parte del **pseo** de la misma. Esta situación plantea ciertos **retos** a la hora de hacer ***diseño responsivo.**
- Estos retos son principalmente dos:
  - Retos a nivel de **optimización** de la página web.
  - Retos a la hora de **diseñar** la página.

# Diseño Web Responsive

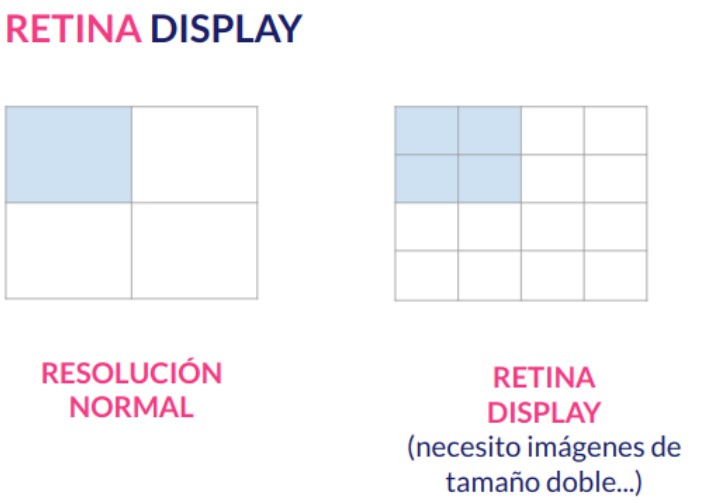


- CREAR UNA REJILLA FLEXIBLE - IMÁGENES FLEXIBLES
- **Optimización de Imágenes en el Diseño Responsivo**
- Cuando nos referimos a optimizar el uso de las imágenes nos referimos a:
  - Consumir el menor ancho de banda posible.
  - Elegir la versión de una misma imagen más adecuada para la resolución.
- En este proceso de optimización debemos tener en cuenta:
  - Ancho del dispositivo.
  - Dimensiones de la imagen.
  - Resolución de la imagen (en especial en los dispositivos RETINA DISPLAY)

# Diseño Web Responsive



- CREAR UNA REJILLA FLEXIBLE - IMÁGENES FLEXIBLES
- **Retina Display**
- Este tipo de dispositivos tiene una densidad de pixels superior a la normal. En la imagen su muestra la explicación para dispositivos con densidad 2x. En estos casos para que las imágenes se muestren bien tendrán que ser de tamaño doble.



# Diseño Web Responsive



- CREAR UNA REJILLA FLEXIBLE - IMÁGENES FLEXIBLES
- **Optimización. Imágenes SVG**
  - Para solucionar este tipo de problemas lo más fácil es usar imágenes **SVG** que son gráficos vectoriales que escalan y encogen sin perder resolución.
  - El problema es que no siempre disponemos de gráficos SVG.

# Diseño Web Responsive



- CREAR UNA REJILLA FLEXIBLE - IMÁGENES FLEXIBLES
- **Optimización. Imágenes PNG/GIF/JPEG**
- En estos casos, si no nos importa la optimización es suficiente con usar una imagen de gran resolución y dimensiones y acotarla dentro de un contenedor.
- Por ejemplo:

```
<div><img src="".......”" /></div>
```

```
div {
  /* dimensiones deseadas */
}

img {
  max-width: XXXXXpx;
  width: 100%;
}
```

# Diseño Web Responsive



- CREAR UNA REJILLA FLEXIBLE - IMÁGENES FLEXIBLES
- **Optimización. Imágenes PNG/GIF/JPEG**
- Pero sin embargo, si nos importa la optimización utilzaremos los atributos **scrset** y/o **sizes** de la imagen que queremos mostrar.
- Por ejemplo:

```
<!-- Considerando resolución: -->
<div class="container">
  <img src="img/small.jpg" srcset="img/big.jpg 2x, img/small.jpg 1x" />
</div>

<!-- Considerando dimensiones: -->
<div class="container">
  <img
    src="img/small.jpg"
    srcset="img/big.jpg 2000w, img/small.jpg 1000w"
    sizes="(min-width: 960px) 960px,100vw"
  />
</div>
```

# Diseño Web Responsive



- CREAR UNA REJILLA FLEXIBLE - IMÁGENES FLEXIBLES
- **Diseño "ART-DIRECTOR"**
  - La técnica de diseño responsivo Art Director consiste en elegir una u otra imagen utilizando la etiqueta source dentro la etiqueta picture y sus atributos srcset para indicar la imagen y media que funciona de manera similar a una media query.
  - Normalmente este diseño consiste en fotos que se acercan al objeto importante.
  - Al final, en casos reales siempre hay que combinar ambas técnicas y probar, en la medida de lo posible, en dispositivos reales, incluidos móviles con Retina Display.

```
<div class="art">
  <picture>
    <source media="(min-width: 576px)" srcset="img/big-art.jpg" />
    <source media="(max-width: 575px)" srcset="img/small-art.jpg" />
    <img src="img/small-art.jpg" />
    <!-- En caso de no soportar picture -->
  </picture>
</div>
```

# Diseño Web Responsive



- CREAR UNA REJILLA FLEXIBLE - IMÁGENES FLEXIBLES
  - El último punto a tener en cuenta es el comportamiento de las imágenes. Por suerte, este es el caso más sencillo. Es suficiente con definir el ancho máximo de las imágenes al 100% (todo el ancho de su contenedor).

```
img {
    max-width: 100%;
}
```

# Diseño Web Responsive



- CREAR UNA REJILLA FLEXIBLE - IMÁGENES FLEXIBLES
  - El último punto a tener en cuenta es el comportamiento de las imágenes. Por suerte, este es el caso más sencillo. Es suficiente con definir el ancho máximo de las imágenes al 100% (todo el ancho de su contenedor).
  - Todas las imágenes del documento ocuparán como máximo el mismo tamaño que su contenedor, independientemente de su tamaño original.
  - background-size de CSS 3 para redimensionar imágenes de fondo
  - Valido videos

```
img {
    max-width: 100%;
}
```

```
img,
embed,
object,
video {
    max-width: 100%;
}
```

# Diseño Web Responsive



- DESARROLLO ADAPTATIVO
  - Hasta ahora, hemos definido todas las herramientas necesarias para crear estructuras adaptativas: diseños basados en rejillas flexibles, estrategias para incorporar elementos multimedia y la potencia de los *Media Query* para adaptar el contenido a nuestras necesidades. Vemos como incorporar estas técnicas a la hora de desarrollar un sitio web adaptativo.

# Diseño Web Responsive



- DESARROLLO ADAPTATIVO - LA IMPORTANCIA DEL CONTEXTO
  - Un diseño adaptativo, implementado de una manera correcta, puede dar a los usuarios de la web, un alto nivel de continuidad entre los diferentes contextos. Esto es así, porque de la manera más simple, un diseño adaptativo es capaz de mostrar un documento HTML correctamente en una infinidad de dispositivos, gracias a una estructura flexible y los Media Query que aseguran un diseño portable y accesible en la manera de lo posible.
  - De alguna manera, es posible identificar el contexto en el que se visita una web, a partir del dispositivo utilizado. De este contexto, podemos definir un tipo de usuario y sus objetivos. En otras palabras, un usuario móvil quiere un acceso rápido a la información y realizar diferentes tareas a las que realizaría sentado en su sofá con su portátil. En este caso, el tiempo y el ancho de banda están en extremos totalmente opuestos.

# Diseño Web Responsive



- DESARROLLO ADAPTATIVO - LA IMPORTANCIA DEL CONTEXTO
  - Por otra parte, si las prioridades y los objetivos del usuario varían según el contexto, entonces puede que disponer de un único documento HTML pueda suponer un problema, dependiendo de la manera en la que se encuentre estructurada la información.
  - De todas formas, es complicado suponer el contexto del usuario únicamente por el tipo de dispositivo, ya que efectivamente, es solamente eso: una suposición. ¿Cómo diferenciar, si mi navegación móvil se realiza desde la entrada del metro o desde el sofá de mi casa? ¿Es posible por tanto suponer un contexto?
  - Por lo tanto, no podemos inferir el contexto del usuario en base a su dispositivo. Así mismo, las palabras mobile o desktop no definen el comportamiento en la que los usuarios acceden a la web: un portátil puede ser un dispositivo móvil (por ejemplo en un tren), al igual que un smartphone o tablet puede estar fijo en un lugar. El desarrollo adaptativo no pretende ser un reemplazo de los actuales sitios móviles, sino que forma parte de una estrategia de desarrollo, donde se pretende evaluar si efectivamente es necesario separar totalmente la experiencia móvil o tiene más sentido mostrar la información de una manera adaptativa..

# Diseño Web Responsive



- DESARROLLO ADAPTATIVO - LA IMPORTANCIA DEL CONTEXTO
  - Por otra parte, si las prioridades y los objetivos del usuario varían según el contexto, entonces puede que disponer de un único documento HTML pueda suponer un problema, dependiendo de la manera en la que se encuentre estructurada la información.
  - De todas formas, es complicado suponer el contexto del usuario únicamente por el tipo de dispositivo, ya que efectivamente, es solamente eso: una suposición. ¿Cómo diferenciar, si mi navegación móvil se realiza desde la entrada del metro o desde el sofá de mi casa? ¿Es posible por tanto suponer un contexto?
  - Por lo tanto, no podemos inferir el contexto del usuario en base a su dispositivo. Así mismo, las palabras mobile o desktop no definen el comportamiento en la que los usuarios acceden a la web: un portátil puede ser un dispositivo móvil (por ejemplo en un tren), al igual que un smartphone o tablet puede estar fijo en un lugar. El desarrollo adaptativo no pretende ser un reemplazo de los actuales sitios móviles, sino que forma parte de una estrategia de desarrollo, donde se pretende evaluar si efectivamente es necesario separar totalmente la experiencia móvil o tiene más sentido mostrar la información de una manera adaptativa..

# Diseño Web Responsive



- DESARROLLO ADAPTATIVO - HACIA UN FLUJO DE TRABAJO ADAPTATIVO
  - Una de las primeras preguntas (si no lo primer) que debemos hacernos, a la hora de plantearnos un diseño adaptativo es la siguiente: ¿En qué medida este contenido o funcionalidad beneficia o aporta valor a nuestros usuarios? De hecho, esto es algo que deberíamos preguntar siempre para cualquier tipo de proyecto, sea web o no.
  - Si partimos de un planteamiento mobile first, hay que asegurarse que la información que mostramos, y las funcionalidades que implementamos sean importantes para el usuario, ya que no hay espacio suficiente para todo. Hay que darse cuenta de lo que realmente importa. Diseñar partiendo de este paradigma, nos obliga a concentrarnos en lo realmente importante.

# Diseño Web Responsive



- DESARROLLO ADAPTATIVO - HACIA UN FLUJO DE TRABAJO ADAPTATIVO
  - *If you design mobile first, you create agreement on what matters most. You can then apply the same rationale to the desktop/laptop version of the web site. We agreed that this was the most important set of features and content for our customers and business; why should that change with more screen space? Luke Wroblewski*
  - En otras palabras, diseñar desde un inicio pensando en dispositivos móviles puede enriquecer la experiencia de los usuarios, proporcionando un elemento que normalmente se pasa por alto: enfocarnos en lo realmente importante. Esto no quiere decir que los diseños sean simples, faltos de contenidos o funcionalidades, sino que debemos concentrarnos en lo realmente importante.

# Diseño Web Responsive



- DESARROLLO ADAPTATIVO - PUNTOS DE RUPTURA
  - El siguiente paso es identificar el número de diseños diferentes que vamos crear, para acomodarnos a los distintos tamaños de dispositivos. La siguiente tabla muestra los anchos más comunes a la hora de identificar los puntos de ruptura

| Punto de ruptura | Dispositivo objetivo |
|---|---|
| `320 pixels` | Para dispositivos pequeños como teléfonos, en disposición vertical |
| `480 pixels` | Para dispositivos pequeños como teléfonos, en disposición horizontal |
| `600 pixels` | Para tabletas de menor tamaño, como Amazon Kindle (`600×800`), en disposición vertical |
| `768 pixels` | Para tabletas de unas 10", como el iPad (`768x1024`), en disposición vertical |
| `1024 pixels` | Para tabletas de unas 10", como el iPad (`768x1024`), y pequeños portátiles o *netbooks*, en disposición horizontal |
| `1200 pixels` | Para pantallas panorámicas, en portátiles o dispositivos de escritorio |
| `1600 pixels` | Para pantallas panorámicas, en portátiles o dispositivos de escritorio |

# Diseño Web Responsive



- DESARROLLO ADAPTATIVO - DESARROLLO ITERATIVO
  - De manera tradicional, el desarrollo web (y en general, el desarrollo de software) ha seguido las siguientes fases para la construcción de un sitio web:
    - Una fase de planteamiento inicial, donde se capturan los requisitos que debe cumplir la solución final, se plantea la estructura de contenidos y se realizan mocksups que serán la base del diseño.
    - En la fase de diseño, los mockups se convierten en pantallas utilizando una herramienta del tipo Photoshop o Fireworks. Estas pantallas deben ser aprobadas por el cliente antes de seguir adelante.
    - Finalmente, una vez que lo diseños se encuentran aprobados, pasan al equipos de desarrollo para crear las páginas estáticas en HTML.

# Diseño Web Responsive



- DESARROLLO ADAPTATIVO - DESARROLLO ITERATIVO
  - En un desarrollo adaptativo, este proceso puede resultar muy pesado y costoso. Cuando diseñamos una página, lo ideal sería diseñar también el aspecto de esta página en los distintos dispositivos, lo que supondría realizar tantos diseños como puntos de ruptura existan. Si tenemos que repetir este proceso con quince o cincuenta páginas, simplemente deja de ser viable.
  - En este punto, una posible mejora puede ser combinar el equipo de diseño y el de desarrollo, para que, partiendo de un diseño fijo y único, discutir de manera conjunta cómo el diseño se va a acomodar a las distintas resoluciones, y las implicaciones que puede tener a la hora de implementarlo. De esta manera, rápidamente surgen preguntas del tipo: ¿Cómo va a funcionar esta galería de imágenes en un dispositivo táctil? ¿De qué manera se va a mostrar este elemento emergente? ¿Qué pasa si no disponemos de JavaScript en el dispositivo?
  - De esta manera podemos comenzar a construir prototipos con un diseño flexible, escalable en los distintos dispositivos, de una manera muy ágil, implicando a ambos equipos.

# Diseño Web Responsive



- DESARROLLO ADAPTATIVO - DESARROLLO ITERATIVO
  - En un desarrollo adaptativo, este proceso puede resultar muy pesado y costoso. Cuando diseñamos una página, lo ideal sería diseñar también el aspecto de esta página en los distintos dispositivos, lo que supondría realizar tantos diseños como puntos de ruptura existan. Si tenemos que repetir este proceso con quince o cincuenta páginas, simplemente deja de ser viable.
  - En este punto, una posible mejora puede ser combinar el equipo de diseño y el de desarrollo, para que, partiendo de un diseño fijo y único, discutir de manera conjunta cómo el diseño se va a acomodar a las distintas resoluciones, y las implicaciones que puede tener a la hora de implementarlo. De esta manera, rápidamente surgen preguntas del tipo: ¿Cómo va a funcionar esta galería de imágenes en un dispositivo táctil? ¿De qué manera se va a mostrar este elemento emergente? ¿Qué pasa si no disponemos de JavaScript en el dispositivo?
  - De esta manera podemos comenzar a construir prototipos con un diseño flexible, escalable en los distintos dispositivos, de una manera muy ágil, implicando a ambos equipos.

# It’s your turn



A2.1.3 Diseño Web Responsive

# Diseño Web Responsive



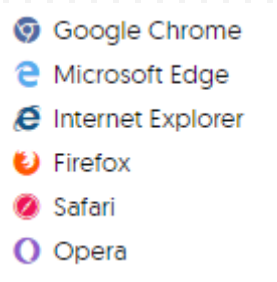
- BOOTSTRAP
  - **Bootstrap** es una biblioteca multiplataforma o conjunto de herramientas de código abierto para diseño de sitios y aplicaciones web. Contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos de diseño basado en HTML y CSS, así como extensiones de JavaScript adicionales. A diferencia de muchos frameworks web, solo se ocupa del desarrollo front-end.
  - Bootstrap es el segundo proyecto más destacado en GitHub[1] y es usado por la NASA y la MSNBC entre otras organizaciones

- https://www.udemy.com/curso-la-biblia-perdida-de-bootstrap-4/

# Diseño Web Responsive



- BOOTSTRAP
  - Bootstrap mejora el tiempo de desarrollo de un sitio web. Permite aplicar metodologías ágiles como **SCRUM** de forma efectiva. Y sobre todo, facilita el diseño responsive para cualquier smartphone o tablet sin perder la navegabilidad de las pantallas de escritorio.
  - Actualmente este framework está totalmente adaptado a **HTML5** y **CSS3**. Es totalmente compatible con los siguientes navegadores



# Diseño Web Responsive




- BOOTSTRAP
  - Bootstrap proporciona una serie de controles web atractivos y muy efectivos para dirigir el tráfico en nuestra web hacia los puntos clave. Se podrían crear componentes SEO como un **Call To Action** con poco esfuerzo.
  - Las **barras de navegación** para la cabecera de nuestra web serán mucho más sencillas de desarrollar, incluso en responsive.
  - La curva de aprendizaje de este framework es bastante rápida, en poco tiempo empezaremos a crear estructuras más complejas.

# Diseño Web Responsive




- BOOTSTRAP
  - Mejora la UX del cliente.
  - Diseño web atractivo, intuitivo y responsive.
  - Curva de aprendizaje rápida y fácil.
  - Se puede combinar con **HTML5**, **CSS3** y **jQuery**, potenciando la web.
  - Su uso es recomendable en metodologías ágiles como SCRUM.
  - Se puede utilizar en **tiendas de afiliación online (TSA)** para hacer un buscador de productos atractivo y eficiente.

# Diseño Web Responsive




- BOOTSTRAP
  - **Donde aprender BootStrap**
    - Su propia documentación:
      - https://getbootstrap.com/docs/4.3/getting-started/introduction/
    - Cursos gratuitos online:
      - https://www.udemy.com/curso-la-biblia-perdida-de-bootstrap-4/

# Preprocesadores CSS





- Son aplicaciones que procesan cierto tipo de archivos para crear hojas de estilos con algunas de las ventajas de un lenguaje de programación común como el uso de variables, funciones, condiciones, la posibilidad de hacer cálculos matemáticos y que además te permiten utilizar una sintaxis mucho más sencilla e intuitiva.
- Hay muchos pre-procesadores, cada uno con sus ventajas, los 2 más populares: **Sass** y **LESS.**
- http://www.wizache.com/blog/sass-vs-less-que-pre-procesador-de-css-es-mejor.html
- Koala o plugins Visual Code.

# JS / Jquery /AJAX



- JavaScript
  - https://www.arkaitzgarro.com/javascript/
- JQuery
  - https://www.arkaitzgarro.com/jquery/
- AJAX
  - https://www.arkaitzgarro.com/ajax/