# Layout

# display

- Es la propiedad CSS más importante para controlar el layout.
- Permite especificar:
  - si un elemento HTML se muestra o no.
  - cómo se muestra un elemento HTML.

# display

- Cada elemento HTML tiene un comportamiento por defecto respecto a cómo se muestra en la página, en función del tipo de elemento que sea.
- Los dos comportamientos posibles son:
  - **block**
  - **inline**

# Elementos de tipo bloque (block)

- Un elemento de tipo bloque (block) siempre comienza en una nueva línea y ocupa todo el ancho disponible (se extiende de izquierda a derecha tanto como puede).
- Algunos elementos de tipo bloque:
  - div
  - h1 - h6
  - p
  - form
  - header, footer, section

# Elementos de tipo en línea (inline)

- Un elemento en línea (inline) comienza al lado de otro si tiene espacio suficiente y solo ocupa el ancho necesario.
- Algunos elementos de tipo en línea:
  - span
  - a
  - img

# display: block; & display: inline;

- Se puede modificar el comportamiento block/inline de un elemento HTML.

- Más ejemplos:
  - Elementos  mostrados como elementos de tipo bloque.
  - Elementos <a> mostrados como elementos de tipo bloque.

# display: none;

- Se utiliza habitualmente junto a JavaScript para:
  - ocultar elementos, (sin eliminarlos del DOM), aprovechando el espacio disponible para otros elementos de la página.
  - mostrar elementos (sin necesidad de volver a crearlos).
- Ejemplo.

# display: none; VS visibility: hidden;

- Ocultar un elemento utilizando `display: none;` -> la página se mostrará como si el elemento no existiera. Ejemplo.

- Ocultar un elemento utilizando `visibility: hidden;` -> la página seguirá mostrando el mismo espacio que cuando el elemento estaba visible. Ejemplo.

- Otro ejemplo.

# Actividad 2.7.1

Crea  un documento HTML que contenga un botón de menú que, cuando se haga click sobre él, abra un panel con opciones correspondientes a ese menú. Si el panel está abierto, cuando se haga click sobre el botón de menú, el panel se debe ocultar.

# display: inline-block

- Hasta ahora, sabíamos que los elementos HTML podían mostrarse de dos formas distintas en función del tipo de elemento:
  - display: block;
  - display: inline;

- Pero hay otro modo posible, a medio camino entre esas dos -> display: inline-block;

# display: inline-block

- Diferencias con `display: inline;`
  - Permite establecer ancho y alto.
  - Se respetan los márgenes y rellenos de la parte superior (top) e inferior (bottom).
- Diferencias con `display: block;`
  - Los elementos se renderizan uno a continuación del otro, sin líneas de separación.

# display: inline-block

- Comparativa display: inline; VS display: inline-block; VS display: block;

- display: inline-block; se utiliza habitualmente para mostrar elementos de manera horizontal el lugar de vertical-> Ejemplo de uso de display: inline-block; para crear un menú de navegación.

# Actividad 2.7.2

Crea  un documento HTML que contenga una barra de menú con tres secciones (menú 1, menú 2 y menú3) y que, cuando se haga click sobre alguna de ellas, abra un panel con opciones correspondientes a ese menú. Si un panel está abierto, cuando se haga click sobre otra sección de la barra de menú, el panel se debe ocultar y mostrar el correspondiente.

| menú 1 | menú 2 | menú 3 |
|---|---|---|
| submenú 1.1<br>submenú 1.2<br>submenú 1.3 | submenú 2.1<br>submenú 2.2<br>submenú 2.3 | submenú 3.1<br>submenú 3.2<br>submenú 3.3 |

# Actividad 2.7.2 (solución)

```
<body>
	<nav class="menu">
		<a id="menu_entrie_1" href="">menu 1</a>
		<a id="menu_entrie_2" href="">menu 2</a>
		<a id="menu_entrie_3" href="">menu 3</a>
	</nav>
	<div class="submenu">
		<a href="">submenu 1.1</a>
		<a href="">submenu 1.2</a>
		<a href="">submenu 1.3</a>
	</div>
	<div class="submenu">
		<a href="">submenu 2.1</a>
		<a href="">submenu 2.2</a>
		<a href="">submenu 2.3</a>
	</div>
	<div class="submenu">
		<a href="">submenu 3.1</a>
		<a href="">submenu 3.2</a>
		<a href="">submenu 3.3</a>
	</div>

</body>
```

# Actividad 2.7.2 (solución)

```
<style>
	.menu{
		border: 1px solid black;
		background-color: black;
		padding: 10px;
	}
	.menu a{
		color: white;
		text-decoration: none;
	}
	#menu_entrie_1{
		margin-left: 14%;
	}
	#menu_entrie_2{
		margin-left: 28%;
	}
	#menu_entrie_3{
		margin-left: 28%;
	}
	.submenu{
		border: 1px solid black;
		display: inline-block;
		width: 32.8%;
		text-align: center;
		padding-top: 20px;
	}
	.submenu a{
		display: block;

	}
</style>
```

# position

- Permite especificar el tipo posicionamiento de un elemento.
- Existen cinco tipos:
  - ***static***
  - ***relative***
  - ***fixed***
  - ***absolute***
  - ***sticky***

# position

- Para posicionar un elemento se utilizan las propiedades: ***top***, ***right***, ***bottom*** y ***left***.
- Para que funcionen estas propiedades es necesario establecer previamente la propiedad ***position***.

# position: static;

- Los elementos HTML se posicionan por defecto como `position: static;`
- En este caso, no tienen efecto las propiedades ***top***, ***right***, ***bottom*** y ***left***.
- Un elemento con `position: static;` no está posicionado de ninguna manera especial, simplemente se posiciona de acuerdo al flujo normal de la página.
- Ejemplo.

# position: relative;

- Un elemento con `position: relative;` está posicionado en relación a su posición normal.
- Se toma como base su posición normal, y, en función de los valores de las propiedades ***top***, ***right***, ***bottom*** y ***left*** se mostrará en una posición u otra.

- **IMPORTANTE:** El contenido no se ajustará para encajar en ningún hueco dejado por el elemento.

# position: fixed;

- Un elemento con `position: fixed;` está posicionado en relación a la ventana.
- Siempre se mantiene en el mismo lugar, incluso si se hace scroll.
- Ejemplo.

# position: absolute;

- Un elemento con `position: absolute;` está posicionado en relación con su elemento padre posicionado más cercano.
- Si no existiese un elemento padre posicionado, se posiciona en relación al body y se mueve junto con el desplazamiento de la página.
- **IMPORTANTE**: los elementos con `position: absolute;` dejan de formar parte del flujo normal y pueden superponerse a otros elementos.
- Ejemplo.

# position: sticky;

- Un elemento con `position: sticky;` está posicionado según la posición de scroll realizada por el usuario.
- Combina entre posicionamiento `relative` y `fixed`, dependiendo de la posición del scroll.
- El elemento se coloca en una posición relativa hasta que se alcanza una posición determinada en la ventana, momento en el que pasa a mantener una posición fija.
- Ejemplo.

# position: sticky;

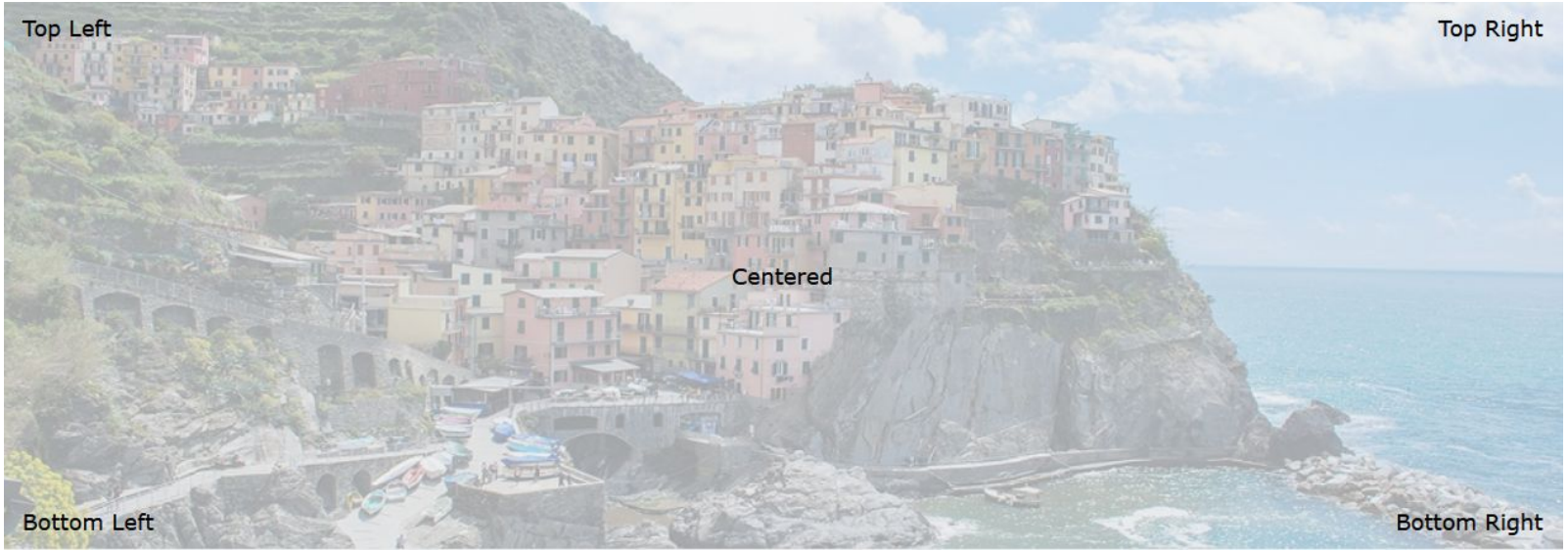
- **TEN EN CUENTA QUE**:
  - Internet Explorer no soporta posicionamiento sticky.
  - Safari requiere declarar el prefijo -webkit-. También se debe especificar al menos una de las propiedades `top`, `right`, `bottom` y `left` para que funcione el posicionamiento sticky.

# Actividad 2.8 (parte 1)

- Enlace a los ejercicios.
- No utilices “Show Answer”.

# Actividad 2.8 (parte 2)

- Crea un documento web con la siguiente distribución de texto:



# Actividad 2.8 (parte 2) (Solución): Posicionando texto en una imagen

- Texto “centrado”.
- Texto arriba a la izquierda.
- Texto arriba a la derecha.
- Texto debajo a la izquierda.
- Texto debajo a la derecha.

# z-index

- Existe un problema: Cuando posicionamos elementos, estos pueden solaparse, formándose una “pila de elementos”.
- Solución: La propiedad `z-index` permite especificar el orden que un elemento HTML tiene en esa “pila de elementos”.

# z-index

- Un elemento puede tener un orden positivo (>=0) o negativo (<0) dentro de la pila de elementos solapados.
- z-index solo funciona con elementos posicionados (absolute, relative, fixed o sticky)

- Si no se utiliza z-index, el último elemento definido en el código HTML será el que se muestre en la cima de la pila.

# overflow

- Permite controlar lo que sucede con el contenido que excede/desborda el área visible de un elemento.
- **Ten en cuenta que**: si en el ejemplo eliminas `overflow: scroll;` recuerda cambiar el color del texto a negro para poder ver el comportamiento por defecto.

- **Importante**: La propiedad `overflow` sólo funciona con elementos de tipo bloque que tienen un alto específico.

# overflow

- Se pueden utilizar los siguientes valores:
  - visible -> El desbordamiento no se recorta. El contenido se renderiza fuera de la caja del elemento. Es el comportamiento por defecto.
  - hidden -> El desbordamiento se recorta y el resto del contenido será invisible.
  - scroll -> El desbordamiento se recorta y se añaden barras de desplazamiento para ver el resto del contenido.
  - auto -> Similar a scroll, pero sólo se añaden barras de desplazamiento si es necesario.

# overflow-x & overflow-y

- overflow-x especifica qué se hace con el desbordamiento a nivel horizontal.
- overflow-y especifica qué se hace con el desbordamiento a nivel vertical.
- Ejemplo.

# float

# float

- Se utiliza para posicionar y formatear contenido.
- Ejemplo: permitir que una imagen “flote” a la derecha de un texto en un contenedor.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor. Maecenas nisl est, ultrices nec congue eget, auctor vitae massa. Fusce luctus vestibulum augue ut aliquet. Mauris ante ligula, facilisis sed ornare eu, lobortis in odio. Praesent convallis urna a lacus interdum ut hendrerit risus congue. Nunc sagittis dictum nisi, sed ullamcorper ipsum dignissim ac...

# float

- La propiedad float puede tomar los siguientes valores:
  - left -> El elemento flota a la izquierda de su contenedor.
  - right -> El elemento flota a la derecha de su contenedor.
  - none -> El elemento no flota (se mostrará justo donde aparece el texto). Es el comportamiento por defecto.
  - inherit -> El elemento hereda el valor de la propiedad float que tiene su padre.

# flotando uno al lado del otro

- Por defecto, dos o más elementos `<div>` se muestran uno encima del otro, con un salto de línea entre ellos (comportamiento `display: block;` que ya conoces).


- Sin embargo, si utilizamos `float: left;` estos `<div>` flotarán uno al lado del otro.

# entendiendo float

- float desliga un elemento de su flujo normal de posicionamiento.
- Un elemento flotante abandona su posición normal y “flota” en la página hasta su límite superior (sea este su contenedor u otros elementos posicionados de forma normal encima del elemento flotante).
- float crea una especie de “nueva capa de profundidad”, (no confundir con z-index), que permite que:
    - el elemento flotante se posicione en primer plano
    - el espacio sobrante a izquierda o derecha que genera el elemento flotante será ocupado por el contenido del elemento que está a continuación del elemento flotante

# clear

- Al utilizar `float`, en algún punto necesitaremos volver a disponer del flujo normal de posicionamiento de los elementos.
- La propiedad `clear` permite restablecer el flujo normal de los elementos que están a continuación del elemento flotante.

# clear

- La propiedad clear puede tomar los siguientes valores:
  - `none` -> El elemento no se empuja hacia abajo de los elementos flotantes. Es el comportamiento por defecto.
  - `left` -> El elemento se empuja hacia abajo de los elementos flotantes izquierdos.
  - `right` -> El elemento se empuja hacia abajo de los elementos flotantes derechos.
  - `both` -> El elemento se empuja hacia abajo de los elementos flotantes izquierdos y derechos.
  - `inherit` -> El elemento hereda el valor de la propiedad `clear` de su padre.

# clear

- **TEN EN CUENTA QUE:**
  - Si un elemento flota hacia la izquierda, debes “limpiar” hacia la izquierda -> `clear: left;` o `clear: both;`
  - Si un elemento flota hacia la derecha, debes “limpiar” hacia la derecha -> `clear: right;` o `clear: both;`
  - El elemento flotante seguirá flotando, pero el elemento que has “limpiado” aparecerá debajo del que flota.

# clearfix hack

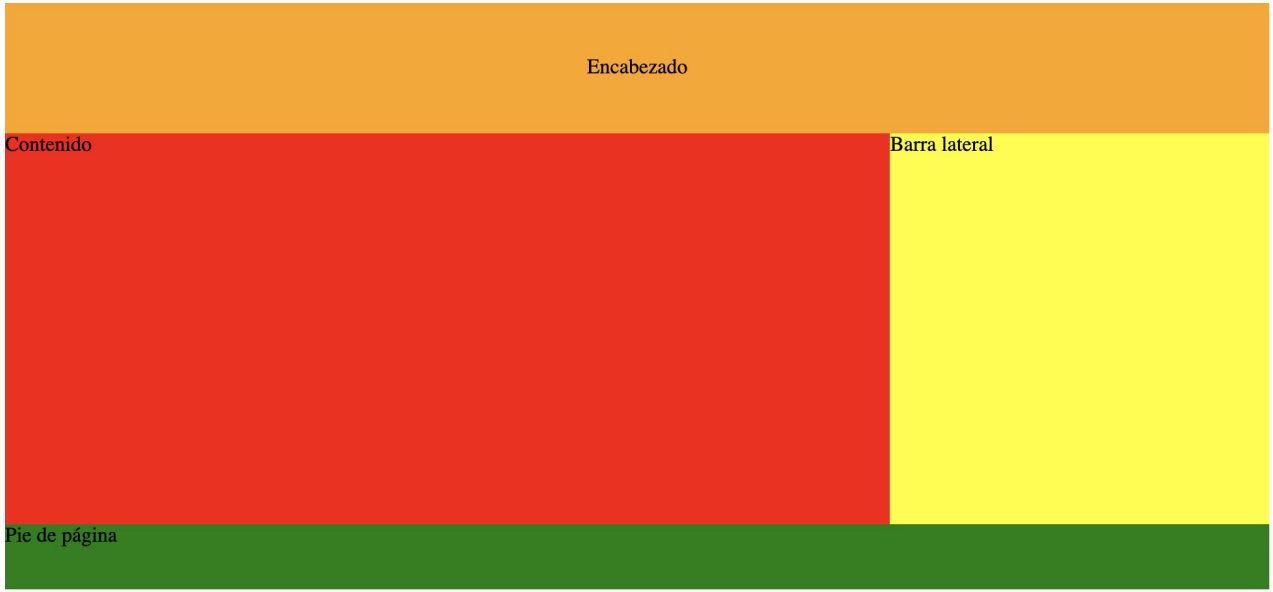
- Existe un **problema**: Si un elemento flotante es más alto que el elemento contenedor, se "desbordará" fuera de su contenedor.

- **Solución 1** (navegadores modernos): agregar `overflow: auto;`
- **Solución 2** (todos los navegadores): agregar un “clearfix hack”

# Actividad 2.9

- Crea un fichero HTML que incluya los estilos CSS necesarios para que se renderice un layout básico con los siguientes elementos:
    - Un **encabezado** de 100% de ancho y 100px de alto. Fondo naranja.
    - Un **contenido** de 70% de ancho y 300px de alto. Color rojo.
    - Una **barra lateral** de 30% de ancho y 300px de alto. Color amarillo.
    - Un **pié de página** de 100% de ancho y 50px de alto. Color verde.

# Actividad 2.9



# Ejemplos de aplicación de float

- Utilizando ***float*** se puede componer un grid de cajas que se muestran una al lado de la otra.

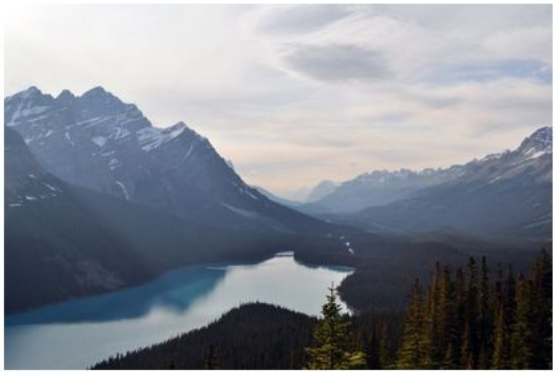
# Ejemplos de aplicación de float

- El ejemplo anterior tiene un **problema:** cuando incrementas el ancho de cada caja con relleno o borde, la última caja se posicionará en otra línea.
- **Solución**: la propiedad `box-sizing` te permite incluir el relleno y el borde en el ancho (y alto) total de la caja, eliminando el comportamiento anterior.

# Ejemplos de aplicación de float

- Utilizando ***float*** se puede componer un grid de cajas para mostrar imágenes.





# Ejemplos de aplicación de float

- Utilizando ***float*** se puede componer un grid de cajas que tienen la misma altura.

# Ejemplos de aplicación de float

- El ejemplo anterior tiene un **problema:** “no es flexible” -> sólo funciona bien si puedes garantizar que siempre existe la misma cantidad de contenido dentro de la caja. En caso contrario, el contenido se mostrará fuera de la caja.
- Vuelve a revisar el ejemplo anterior reduciendo el ancho de la ventana.

# Ejemplos de aplicación de float

- ¿Solución? -> Utilizar Flexbox.

- flex-wrap especifica si los elementos flexibles deben ajustarse (wrap) o no (nowrap).

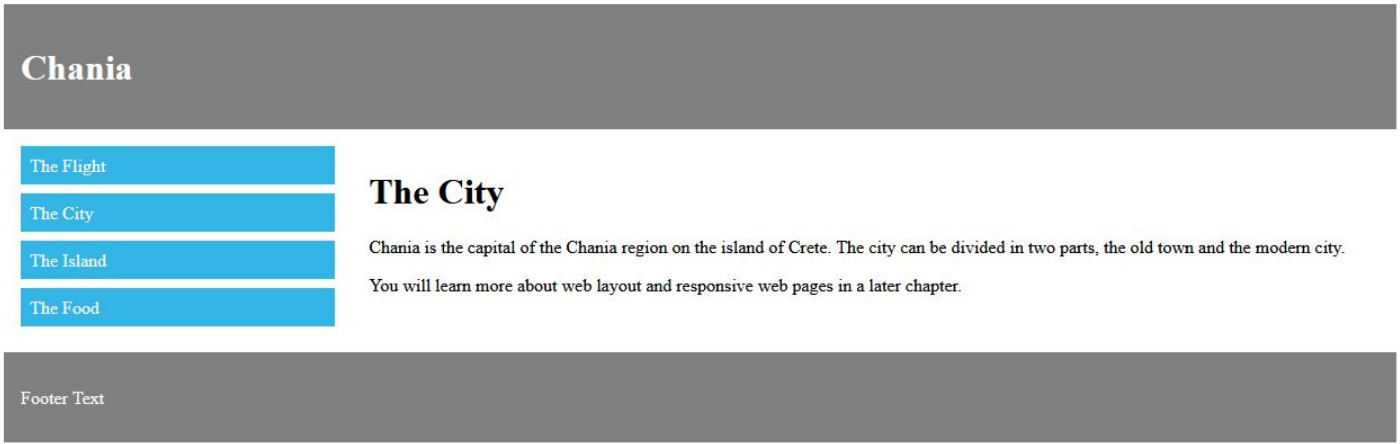
- Más sobre Flexbox próximamente...

# Ejemplos de aplicación de float

- Se puede utilizar ***float*** junto a una lista de enlaces para crear un menú horizontal.

# Ejemplos de aplicación de float

- Y, como ya hemos visto, con float [también podemos componer layouts](#).



# Actividad 2.10 (Entrega obligatoria)

A partir del trabajo realizado en la UD1 (propuesta de diseño y guía de estilos), **implementar los diferentes ficheros HTML y hojas de estilos** teniendo en cuenta:

- Cada una de las páginas diseñadas (login, home (admin) y home (user)) debe ser implementada en un fichero HTML distinto junto a su/s correspondiente/s hoja/s de estilos.
- Los estilos que sean comunes para todas las páginas, pueden ubicarse en un mismo fichero CSS, que será referenciado en cada uno de los ficheros HTML.
- Si se realizan cambios respecto de la propuesta original presentada en la UD1, deben ser mencionados en un apartado adicional de la guía de estilos (Modificaciones v.1.1).

# Actividad 2.10 (Entrega obligatoria)

Se valora:

- Aplicación correcta de las diferentes propiedades CSS estudiadas durante la UD2.
- Aplicación correcta de las diferentes técnicas CSS (referencia de hojas de estilos dentro de ficheros HTML, posicionamiento, uso de float, etc.) estudiadas durante la UD2.
- Adecuación a la propuesta de diseño realizada en la UD1 o a la propuesta de diseño modificada según lo descrito en la guía de estilos.

# Actividad 2.10 (Entrega obligatoria)

Ten en cuenta que:

- Debes entregar el trabajo a través del Aula Virtual dentro de un fichero ZIP (nomenclatura: A2_10_nombreApellido1_nombreApellido2.zip) que incluya los siguientes ficheros:
    - Por cada página (login, home (admin) y home (user)): fichero HTML y hoja/s de estilo utilizadas.
- **Fecha límite de entrega: publicada en el aula virtual.**

# Actividad 2.10 (Entrega obligatoria)

Ten en cuenta que:

- Es obligatorio realizar la defensa del trabajo.
- Se indicará la fecha y hora de defensa para cada grupo a través del Aula Virtual.
- Los trabajos no entregados, entregados fuera de plazo o no defendidos tendrán una calificación de 0 puntos.
- Fechas de defensa: 12 al 15 de diciembre de 2022 (en horario de clase).