Unidad 4. Creación de interfaces web utilizando estilos

ESTRUCTURA DEL SITIO WEB

Contenido

1. EL MÓDULO FLEXIBLE BOX LAYOUT....3

1.1. OBJETIVO DEL MÓDULO....3
1.2. LAS CAJAS FLEXIBLES....3
1.2.1. La orientación y el sentido de las cajas....4
1.2.2. El desborde de las cajas....6
1.2.3. La alineación horizontal de las cajas....7
1.2.4. La alineación vertical de las cajas....9
1.2.5. Las propiedades de flexibilidad....10
1.2.6. Las propiedades flex-grow, flex-shrink y flex-basis....12

2. GRID LAYOUT....16

2.1 ¿QUÉ ES CSS GRID?....16
2.2 VENTAJAS DE CSS GRID....16
2.3 PRIMEROS PASOS....16
2.3.1. Crea un bloque contenedor....16
2.3.2. Activa CSS Grid....17
2.3.3. Define tu rejilla....17
2.4. COLOCAR LOS ELEMENTOS EN LA REJILLA....18

3. EJEMPLOS DE COMPOSICIONES DE PÁGINAS....20

3.1. OBJETIVO....20
3.2. COMPOSICIÓN DE PÁGINA CON CAJAS FLOTANTES....20
3.2.1. Objetivo....20
3.2.2. El contenedor general....21
3.2.3. El encabezado....22
3.2.4. El logo, el eslogan y el campo de búsqueda....22
3.2.5. La navegación....26
3.2.6. La zona central....27
3.2.7. El pie de página....28
3.2.8. El código completo de este ejemplo....29
3.2.9. Los inconvenientes....32
3.3 COMPOSICIÓN DE PÁGINA CON FLEX....32

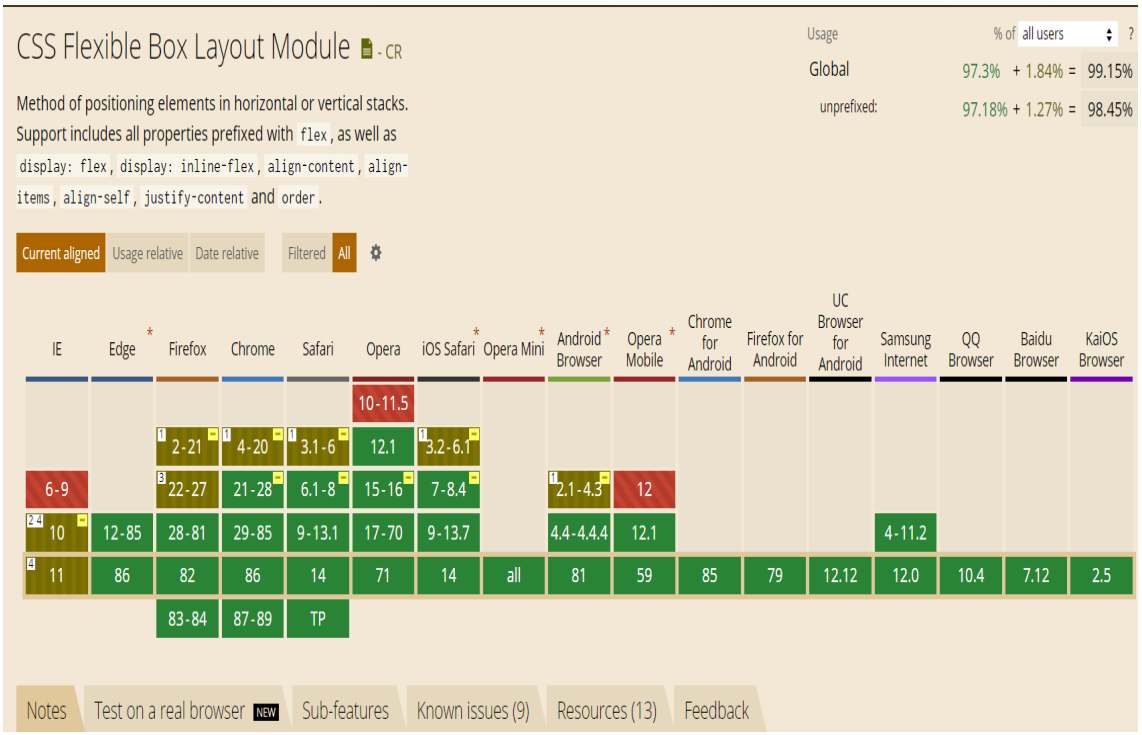
1. EL MÓDULO FLEXIBLE BOX LAYOUT

1.1. OBJETIVO DEL MÓDULO

Hemos visto en el capítulo sobre las cajas cómo ubicarlas donde queramos con las propiedades float y position. En el módulo **Flexible Box Layout**, en **Candidate Recommendation.**

(http://www.w3.org/TR/css3-flexbox/) en noviembre de 2018, el W3C ofrece una nueva posibilidad para colocar elementos en una composición de página elaborada.

En lo que respecta al reconocimiento de este módulo por parte de los navegadores, esto es lo que nos indica el sitio **Can I use** (http://caniuse.com/#search=Flexible%20Box%20Layout):



La imagen es de 27 de octubre de 2020, fecha en la cual ningún navegador necesita de prefijos para ejecutar el módulo Flexbox.

1.2. LAS CAJAS FLEXIBLES

El principio en el que se basa la utilización de las cajas flexibles es muy simple: basta con definir un contenedor padre como caja flexible para que todos los elementos que contiene, los elementos hijos, adopten su comportamiento.

La propiedad display acepta dos valores: flex para una visualización flexible en bloque e inline-flex para una visualización en línea.

Veamos a continuación un primer ejemplo muy sencillo (**10_05.html**). Definimos una caja contenedor padre con una visualización en caja flexible: display: flex;. Automáticamente, todos

sus elementos hijos, tres cajas con fondo de color en este ejemplo, se muestran unas al lado de otras.

He aquí el código utilizado:

```
<!DOCTYPE html>
<html lang="es">
<head>
   <title>Cajas flexibles</title>
   <meta charset="UTF-8" />
   <style>
   #caja {
      display: flex;
   }
   .azul {
      background-color: lightblue;
   }
   .verde {
      background-color: lightgreen;
   }
   .amarillo {
      background-color: lightyellow;
   }
   </style>
</head>
<body>
<div id="caja">
   <div class="azul">
      <p>Vivamus sagittis...</p>
   </div>
   <div class="verde">
      <p>Sed posuere consectetur...</p>
   </div>
   <div class="amarillo">
      <p>Maecenas faucibus...</p>
   </div>
</div>
</body>
</html>
```

Esto es lo que se obtiene:

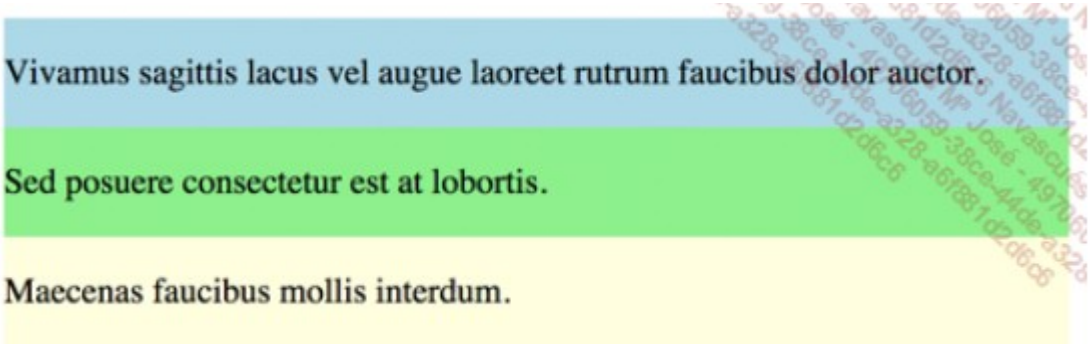
1.2.1. LA ORIENTACIÓN Y EL SENTIDO DE LAS CAJAS

Por defecto, las cajas hijas se muestran unas al lado de otras, en una línea. Es la propiedad **flex-direction** la que determina esta orientación de las cajas hijas con los valores row (comportamiento por defecto) y column.

Este ejemplo (**10_06.html**) muestra las cajas hijas unas debajo de otras:

```
<!DOCTYPE html>
<html lang="es">
<head>
   <title>Cajas flexibles</title>
   <meta charset="UTF-8" />
   <link href="#" rel="stylesheet" />
   <style>
   #caja {
      display: flex;
      flex-direction: column;
   }
   .azul {
      background-color: lightblue;
   }
   .verde {
      background-color: lightgreen;
   }
   .amarillo {
      background-color: lightyellow;
   }
   </style>
</head>
<body>
<div id="caja">
   <div class="azul">
      <p>Vivamus sagittis lacus vel augue laoreet rutrum
faucibus dolor auctor.</p>
   </div>
   <div class="verde">
      <p>Sed posuere consectetur est at lobortis.</p>
   </div>
   <div class="amarillo">
      <p>Maecenas faucibus mollis interdum.</p>
   </div>
</div>
</body>
</html>
```

Esto es lo que se obtiene:



Los valores **column-reverse** y **row-reverse** permiten invertir el sentido en que se visualizan los elementos.

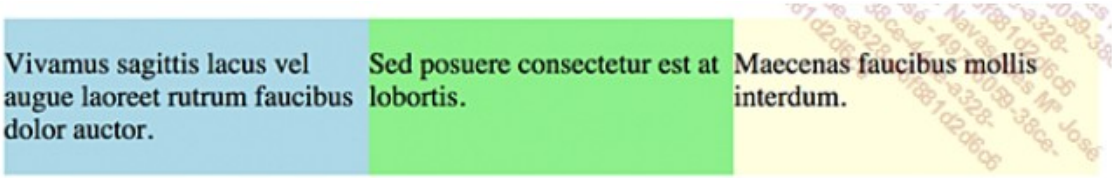
1.2.2. EL DESBORDE DE LAS CAJAS

Podemos gestionar el comportamiento de las cajas hijas cuando estas sobrepasan o desbordan la caja flexible padre con la propiedad **flex-wrap**. El valor **nowrap** impone una visualización en una sola línea aun cuando los elementos hijos se desborden, incluso aunque sean más anchos que la caja flexible padre. La anchura de las cajas hijas se modificará para que estas quepan en la línea de la caja padre.

Este es el código utilizado (**10_07.html**):

```
<!DOCTYPE html>
<html lang="es">
<head>
   <title>Cajas flexibles</title>
   <meta charset="UTF-8" />
   <style>
   #caja {
      width: 600px;
      display: flex;
      flex-direction: row;
      flex-wrap: nowrap;
   }
   .azul {
      width: 250px;
      background-color: lightblue;
   }
   .verde {
      width: 250px;
      background-color: lightgreen;
   }
   .amarillo {
      width: 250px;
      background-color: lightyellow;
   }
   </style>
</head>
<body>
<div id="caja">
   <div class="azul">
      <p>Vivamus sagittis lacus vel...</p>
   </div>
   <div class="verde">
      <p>Sed posuere consectetur...</p>
   </div>
   <div class="amarillo">
      <p>Maecenas faucibus mollis...</p>
   </div>
</div>
</body>
</html>
```

Esto es lo que se obtiene:

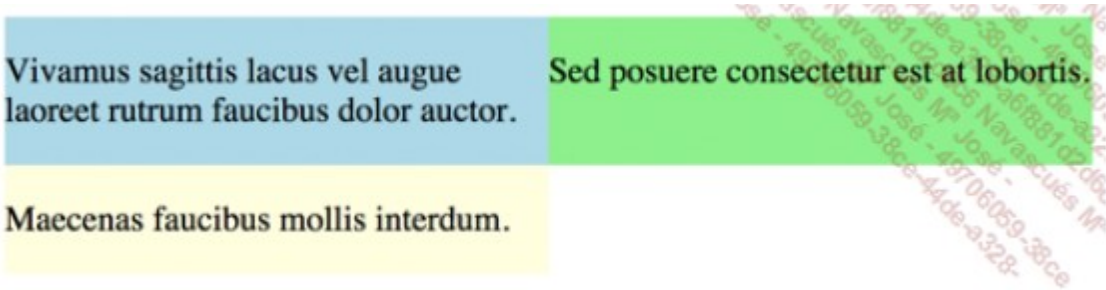


Con valor **wrap**, les cajas hijas conservan la anchura asignada y se encajan en la línea para que su visualización sea correcta.

Este es el selector padre modificado:

```
#caja {
   width: 600px;
   display: flex;
   flex-direction: row;
   flex-wrap: wrap;
}
```

Esto es lo que se obtiene:



1.2.3. LA ALINEACIÓN HORIZONTAL DE LAS CAJAS

Podemos gestionar la alineación horizontal de las cajas hijas con la propiedad **justify-content**. Esta propiedad acepta como valor:

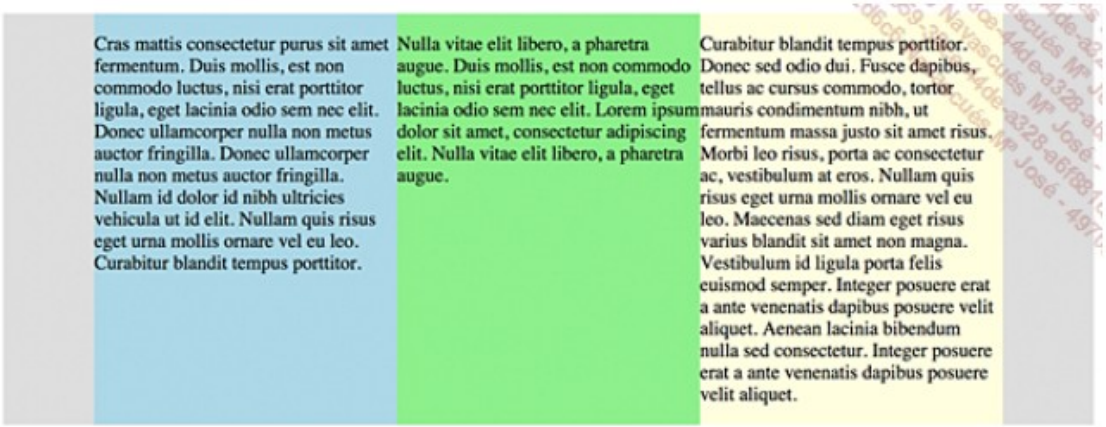
- **flex-start**: los elementos hijos se sitúan a la izquierda dentro del elemento padre. Es el valor por defecto.
- **flex-end**: los elementos hijos se sitúan a la derecha dentro del elemento padre.
- **center**: los elementos hijos se centran horizontalmente dentro del elemento padre.
- **space-between**: los elementos hijos se justifican dentro del elemento padre usando un espacio idéntico entre ellos.
- **space-around**: los elementos hijos se justifican dentro del elemento padre usando un espacio idéntico entre ellos y medio espacio a la izquierda del primero y a la derecha del último.

He aquí un ejemplo (**10_08.html**) con elementos hijos centrados:

```
<!DOCTYPE html>
<html lang="es">
<head>
   <title>Cajas flexibles</title>
   <meta charset="UTF-8" />
```

```
   <style>
   #caja {
      width: 900px;
      background-color: #dedede;
      display: flex;
      flex-direction: row;
      flex-wrap: nowrap;
      justify-content: center;
   }
   .azul {
      width: 250px;
      background-color: lightblue;
   }
   .verde {
      width: 250px;
      background-color: lightgreen;
   }
   .amarillo {
      width: 250px;
      background-color: lightyellow;
   }
   </style>
</head>
<body>
<div id="caja">
   <div class="azul">
      <p>Cras mattis consectetur...</p>
   </div>
   <div class="verde">
      <p>Nulla vitae elit libero...</p>
   </div>
   <div class="amarillo">
      <p>Curabitur blandit...</p>
   </div>
</div>
</body>
</html>
```

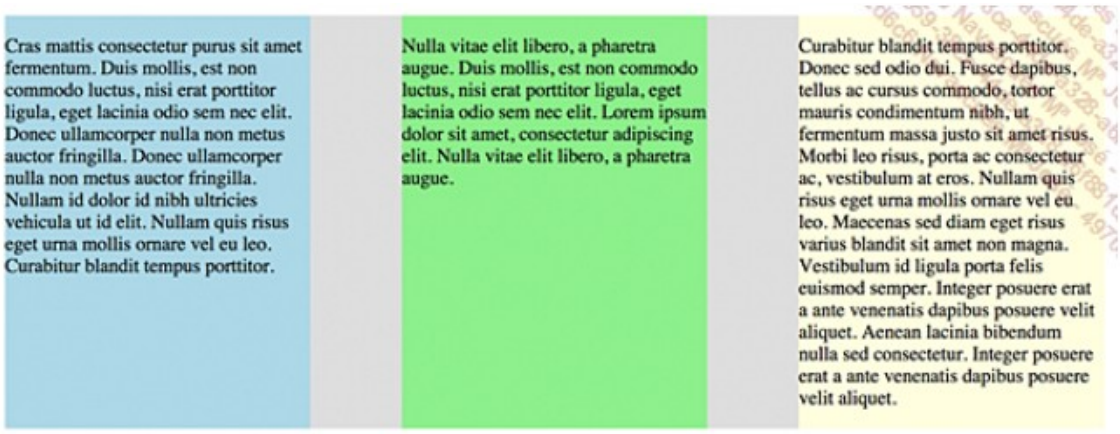
Esto es lo que se obtiene:



Para que se muestren justificadas, usaremos el valor space-between:

```
#caja {
   width: 900px;
   background-color: #dedede;
   display: flex;
   flex-direction: row;
   flex-wrap: nowrap;
   justify-content: space-between;
}
```

Esto es lo que se obtiene:



1.2.4. LA ALINEACIÓN VERTICAL DE LAS CAJAS

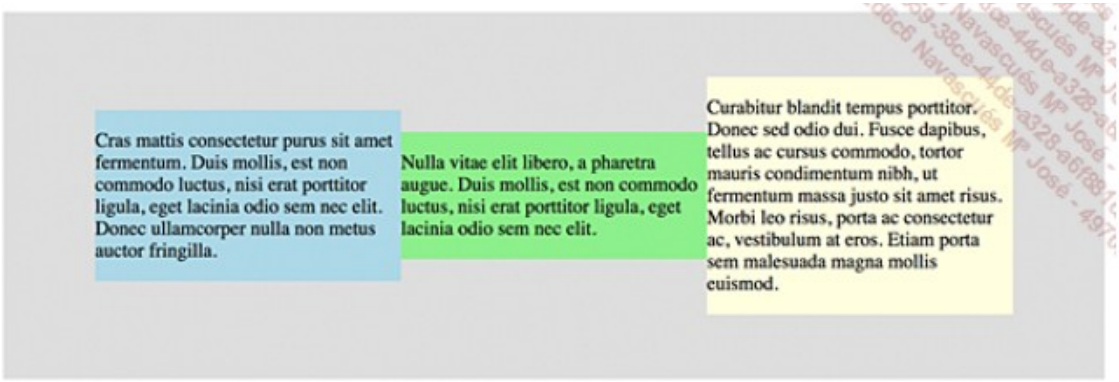
La propiedad **align-items** es la que permite gestionar la alineación vertical de los elementos hijos dentro de la caja padre. Esta propiedad acepta como valor:

- **flex-start**: los elementos hijos se colocan en la parte superior dentro del elemento padre.
- **flex-end**: los elementos hijos se colocan en la parte inferior dentro del elemento padre.
- **center**: los elementos hijos se centran verticalmente dentro del elemento padre.
- **stretch**: los elementos hijos se «estiran» dentro del elemento padre, de forma que utilizan toda la altura de este. Es el valor por defecto.
- **baseline**: los elementos hijos se alinean en la parte superior del padre, sobre cada primera línea.

Este es el selector de la caja padre para elementos hijos centrados horizontal y verticalmente:

```
#caja {
   width: 900px;
   height: 300px;
   background-color: #dedede;
   display: flex;
   flex-direction: row;
   flex-wrap: nowrap;
   justify-content: center;
   align-items: center;
}
```

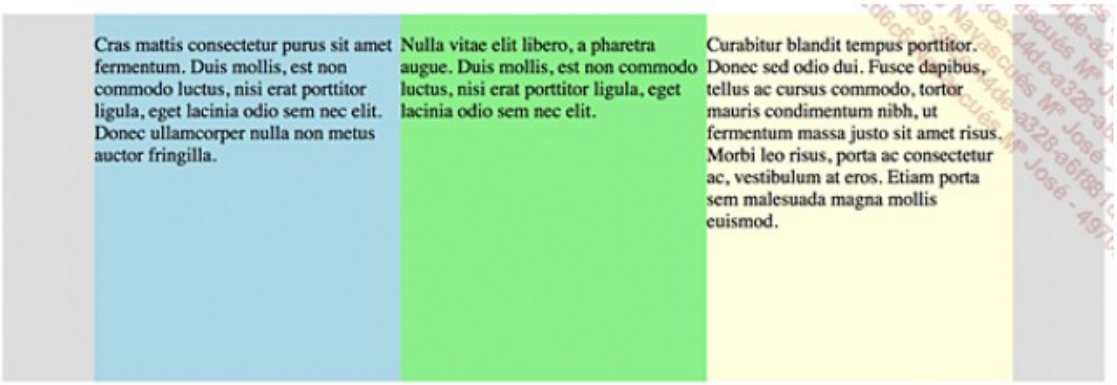
Esto es lo que se obtiene:



Y este es el selector de la caja padre para una visualización vertical justificada:

```
#caja {
   width: 900px;
   height: 300px;
   background-color: #dedede;
   display: flex;
   flex-direction: row;
   flex-wrap: nowrap;
   justify-content: center;
   align-items: stretch;
}
```

Esto es lo que se obtiene:



1.2.5. LAS PROPIEDADES DE FLEXIBILIDAD

Recordemos que este módulo se denomina **Flexbox Layout**; la noción de flexibilidad está incorporada en el nombre. Esta noción de flexibilidad se aplica con la propiedad flex, que es la sintaxis corta de tres propiedades: **flex-grow, flex-shrink y flex-basis**.

Esta propiedad determina el comportamiento de los elementos hijos en lo que respecta a la adaptación al espacio disponible dentro del elemento padre. Estos elementos serán flexibles según el espacio disponible.

La propiedad flex permite definir el factor con el que se «estiran» los elementos, es decir, la facultad de ocupar el espacio disponible dentro del elemento padre. El valor 1 permite indicar que el elemento ocupa **1 parte** del total de partes disponibles, 2 indica que el elemento ocupa **2 partes**.

Veamos un ejemplo sencillo: en el contenedor (#caja) tenemos tres cajas que van a ocupar, respectivamente:

- 1 parte (flex: 1;),
- 2 partes (flex: 2;),
- 1 parte (flex: 1;).

Obtendremos, por lo tanto, 1 + 2 + 1 = 4 partes en el contenedor.

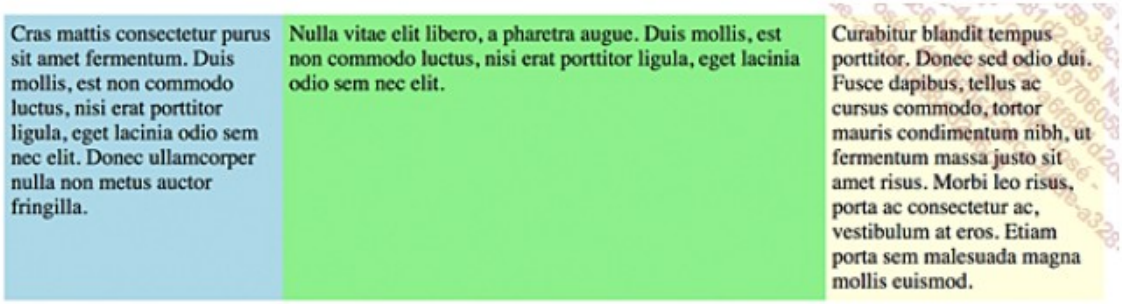
He aquí el código de este ejemplo (**10_10.html**):

```
<!DOCTYPE html>
<html lang="es">
<head>
   <title>Cajas flexibles</title>
   <meta charset="UTF-8" />
   <style>
   #caja {
      width: 800px;
      display: flex;
      flex-direction: row;
      flex-wrap: nowrap;
   }
   #caja > div {
      padding: 5px;
   }
   .azul {
      background-color: lightblue;
      flex: 1;
   }
   .verde {
      background-color: lightgreen;
      flex: 2;
   }
   .amarillo {
      background-color: lightyellow;
      flex: 1;
   }
   p {
      margin: 0;
   }
   </style>
</head>
<body>
<div id="caja">
   <div class="azul">
      <p>Cras mattis consectetur...</p>
   </div>
   <div class="verde">
```

```
      <p>Nulla vitae elit libero...</p>
   </div>
   <div class="amarillo">
      <p>Curabitur blandit...</p>
   </div>
</div>
</body>
</html>
```

El contenedor padre mide 800 píxeles de ancho; la primera y la tercera cajas ocuparán cada una un 25 % (1/4), es decir, 200 píxeles, y la segunda, el 50 % (2/4), es decir, 400 píxeles.

Esto es lo que se obtiene:



1.2.6. LAS PROPIEDADES FLEX-GROW, FLEX-SHRINK Y FLEX-BASIS

Obtenido de https://www.desarrollolibre.net/blog/css/las-propiedades-flex-grow-flex-shrink-y-flex-basis#flex-grow

Lo que caracteriza un diseño flexible es su habilidad para alterar el ancho y alto de sus elementos para ajustarse lo mejor posible al espacio disponible en cualquier dispositivo. Un contenedor flexible expande sus elementos para rellenar el espacio libre, o los comprime para evitar que rebasen el área prevista.

Cosas como emplear la propiedad: `display: flex` para especificar que un contenedor es "flexible", se explicó el uso de la propiedad `flex-direction` para alterar el orden de la columna y `flex-direction` para especificar si los flex se deben mostrar en columnas o filas; entre otras cosas como ejemplos y consideraciones; para más información, consulte en enlace anterior.

En esta entrada emplearemos algunas propiedades que permiten personalizar más el uso de los flex según la situación; ellas son:

`flex-grow`

`flex-shrink`

`flex-basis`

Antes de realizar la explicación de cada una de ellas, se presenta un pequeño ejemplo desde el cual podrás alterar algunos valores y ver el comportamiento en Flexbox Tester

Como muchas otras propiedades como `margin` o `padding` cuenta con una forma resumida o abreviada desde la cual podemos ahorrarnos varias líneas de código al trabajar con propiedades de la misma naturaleza; los flex también cuentan con dicha forma.

La propiedad `flex` es una abreviatura para `flex-grow`, `flex-shrink` y `flex-basis`.

En otras palabras; para especificar las anteriores propiedades lo podemos hacer desde:

flex: flex-grow, flex-shrink, flex-basis

La propiedad `flex-basis`: tamaño básico

Comenzando por la última la cual es la que define el tamaño base (tal como indica su nombre) de los flex, la misma puede venir dada en pixeles porcentajes, etc; esta propiedad indica un punto de partida desde el cual el navegador se guiaría para calcular el tamaño del flex (aunque como dependiendo de las otras propiedades, este tamaño podría variar).

La propiedad `flex-grow`: tamaño sobrante

Esta propiedad especifica cuánto debe crecer nuestros flexs al rellenar el espacio sobrante; se especifican números; para el siguiente CSS:

```
.simple{
flex-grow:1
}
.doble{
flex-grow:2
}
```

Siguiente el ejemplo anterior, el flex con la propiedad `doble` tendría el doble de espacio disponible del padre que el flex con la clase `simple`; si quisiéramos que tuviera el triple del espacio disponible se establecería el número a establecer sería 3; en resumen:

```
unit grow = espacio sobrante/suma de flex-grow
```

dado el siguiente CSS:

```
.contenedor { width: 500px; }
.item-1 { flex-basis: 150px ; flex-grow: 1 }
.item-2 { flex-basis: 150px ; flex-grow: 2 }
.item-3 { flex-basis: 150px ; flex-grow: 1 }
```

Como vemos en el ejemplo anterior, la suma de los flex es menor al tamaño del padre (sobran 50px), en ese escenario el flex de mayor tamaño sería el item-2 siguiendo la fórmula presentada anteriormente quedaría:

```
unit grow = 50/4 = 12.5
```

Sería de 12.5px por unidad:

```
.item-1: 12.5px
.item-2: 25px
.item-3: 12.5px
```

Dando el tamaño total:

```
.item-1: 150px + 12.5px = 170.5px
.item-2: 150px + 25px = = 185px
.item-3: 150px + 12.5px = 170.5px
```

En donde "espacio sobrante" es el espacio sobrante del contenedor y "suma de flex-grow" es la suma de todos los flexs en una columna.

La propiedad `flex-shrink`: tamaño faltante

Al contrario que la propiedad anterior, con `flex-shrink` se especifica el comportamiento de los flexs cuando el tamaño del contenedor es menor al de los flex que lo componen; se especifican de igual manera números:

```
.simple{
flex-shrink:1
}
.doble{
flex-shrink:2
}
```

La fórmula quedaría igual:

```
unit shrink = espacio sobrante/suma de flex-shrink
```

Dado el siguiente CSS:

```
.contenedor { width: 400px; }
.item-1 { flex-basis: 150px ; flex-shrink: 1 }
.item-2 { flex-basis: 150px ; flex-shrink: 2 }
.item-3 { flex-basis: 150px ; flex-shrink: 1 }
```

Siguiente el ejemplo anterior, el flex con la propiedad `doble` se encogería el doble de espacio faltante del padre que el flex con la clase `simple`; si quisiéramos que tuviera se encogiera el triple del espacio faltante del padre se establecería el número a establecer sería 3; en resumen:

```
unit shrink = 50/4 = 12.5
```

Sería de 12.5px por unidad:

```
.item-1: 12.5px
.item-2: 25px
.item-3: 12.5px
```

Dando el tamaño total:

```
.item-1: 150px - 12.5px = 137,5px
.item-2: 150px - 25px = = 124px
.item-3: 150px - 12.5px = 137,5px
```

Para obtener un ejemplo preciso de estructura de página flexible con diseño web adaptativo, consulte el sitio Mozilla Developer Network en esta URL: https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Flexible_boxes#Examples

Para ver más ejemplos consulte el siguiente artículo:
https://www.yunbitsoftware.com/blog/2017/03/30/flexbox-css3-tutorial-descripcion-ejemplos-html/

https://webdesign.tutsplus.com/es/tutorials/exercises-in-flexbox-simple-web-components--cms-28049

https://www.desarrollolibre.net/blog/css/creando-una-web-basica-con-flex-en-html#.XcmCjZNKh0t

https://programacion.net/articulo/5_tecnicas_flexbox_que_deberias_conocer_1418

https://developer.mozilla.org/es/docs/Web/CSS/CSS_Flexible_Box_Layout/Casos_de_uso_tipicos_de_Flexbox.

https://webdesign.tutsplus.com/es/tutorials/building-responsive-forms-with-flexbox--cms-26767

https://developer.mozilla.org/es/docs/Web/CSS/CSS_Flexible_Box_Layout/Casos_de_uso_tipicos_de_Flexbox.#Controles_de_formulario

https://developer.mozilla.org/es/docs/Web/CSS/CSS_Flexible_Box_Layout/Usando_las_cajas_flexibles_CSS

Aplicación web para practicar con Flex:

https://flexboxfroggy.com/#es

2. GRID LAYOUT

2.1 ¿QUÉ ES CSS GRID?

CSS Grid es un sistema de maquetación web que divide la página en una cuadrícula o rejilla (*grid*) a partir de la cual se pueden posicionar los diferentes elementos de manera más sencilla, versátil y coherente.

Su practicidad y sus múltiples ventajas lo han convertido en un estándar. Es decir, casi cualquier navegador soporta e interpreta este tipo de código.

La irrupción de CSS Grid, junto con Flexbox, supuso una revolución en el mundo de la programación web, ya que permitía realizar con mucho menos código elementos y estructuras que resultaban muy complejas o directamente imposibles.

2.2 VENTAJAS DE CSS GRID

- **Mucha más flexibilidad:** te permite controlar los elementos en las 2 dimensiones y con total libertad.
- **Menos código y menos bugs:** se reduce considerablemente el código empleado, lo que conlleva revisiones más sencillas y menor probabilidad de bugs.
- **Optimización de recursos:** al ser código más simple y consistente se economiza tanto en tiempo como en recursos necesarios para mostrar la página.
- **Responsive más sencillo:** permite crear elementos dinámicos que se adaptan a diferentes tamaños o resoluciones sin complicaciones.

2.3 Primeros pasos

Te resumimos el funcionamiento de CSS Grid en 4 sencillos pasos.

2.3.1. CREA UN BLOQUE CONTENEDOR

Para alojar los elementos que quieras colocar es necesario crear un bloque contenedor, como por ejemplo:

```
<section class="contenedor">
      <div class="item-1">1</div>
      <div class="item-2">2</div>
      <div class="item-3">3</div>
      <div class="item-4">4</div>
```

```
      <div class="item-5">5</div>
      <div class="item-6">6</div>
</section>
```

2.3.2. ACTIVA CSS GRID

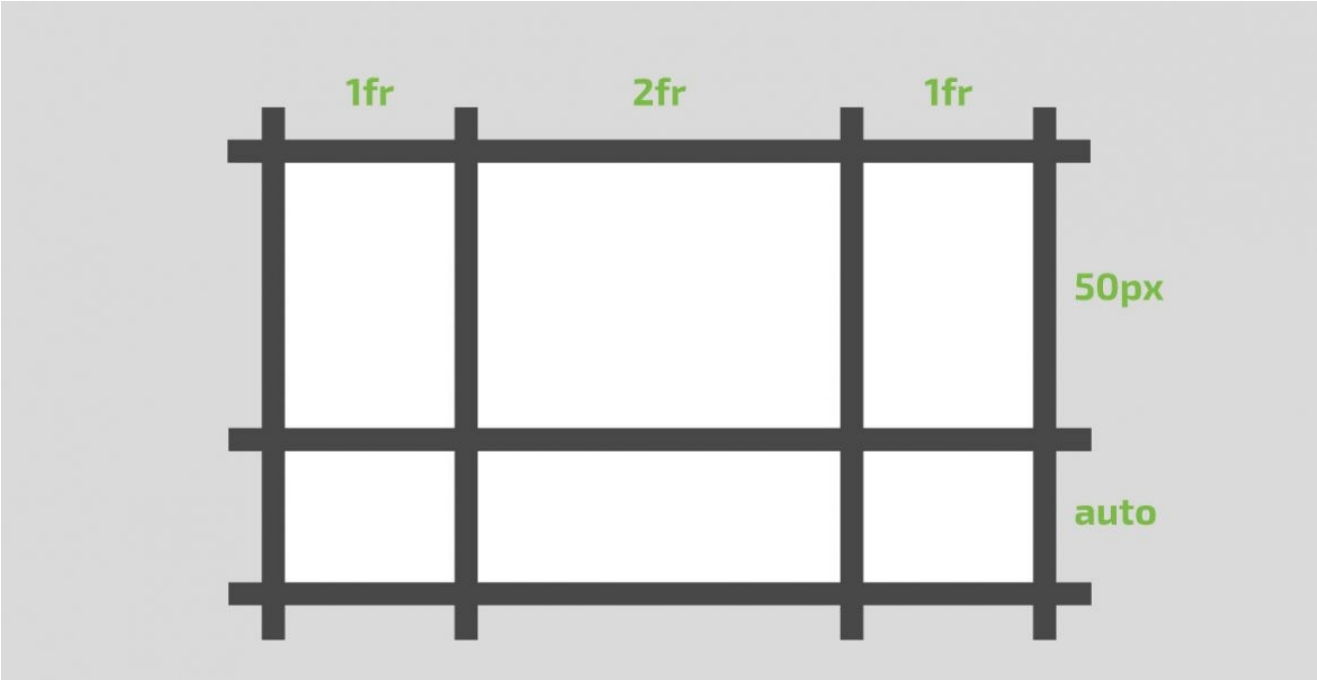
Para que Grid comience a funcionar debes activarlo mediante la propiedad *display: grid*.

```
.contenedor {
    display: grid;
 }
```

2.3.3. Define tu rejilla

Una vez esté activo, toca definir la rejilla mediante las siguientes propiedades: *grid-template-rows*, *grid-template-columns* y/o *grid-template-areas*.

Por ejemplo, si queremos hacer este esquema:



Utilizaremos el siguiente código:

```
.contenedor{
     display: grid;
     grid-template-columns: 1fr 2fr 1fr;
     grid-template-rows: 50px auto;
 }
```

Puedes definir tus elementos utilizando diferentes unidades fijas como los píxeles, o utilizar la unidad propia del sistema Grid: los *fr*.

La unidad *fr* indica la fracción del espacio restante que ocupará un elemento.

Por ejemplo, si tienes 3 columnas y le asignas un valor 1fr a cada una, esto repartirá el espacio a partes iguales entre las 3. Si en cambio, le das el valor 2fr a una de ellas, esa ocupará el doble de espacio que las otras.

El espaciado entre items puedes marcarlo con *grid-row-gap* y *grid-column-gap*, o bien definirlo previamente en tu grid como si fueran una columna/fila más.

2.4. Colocar los elementos en la rejilla

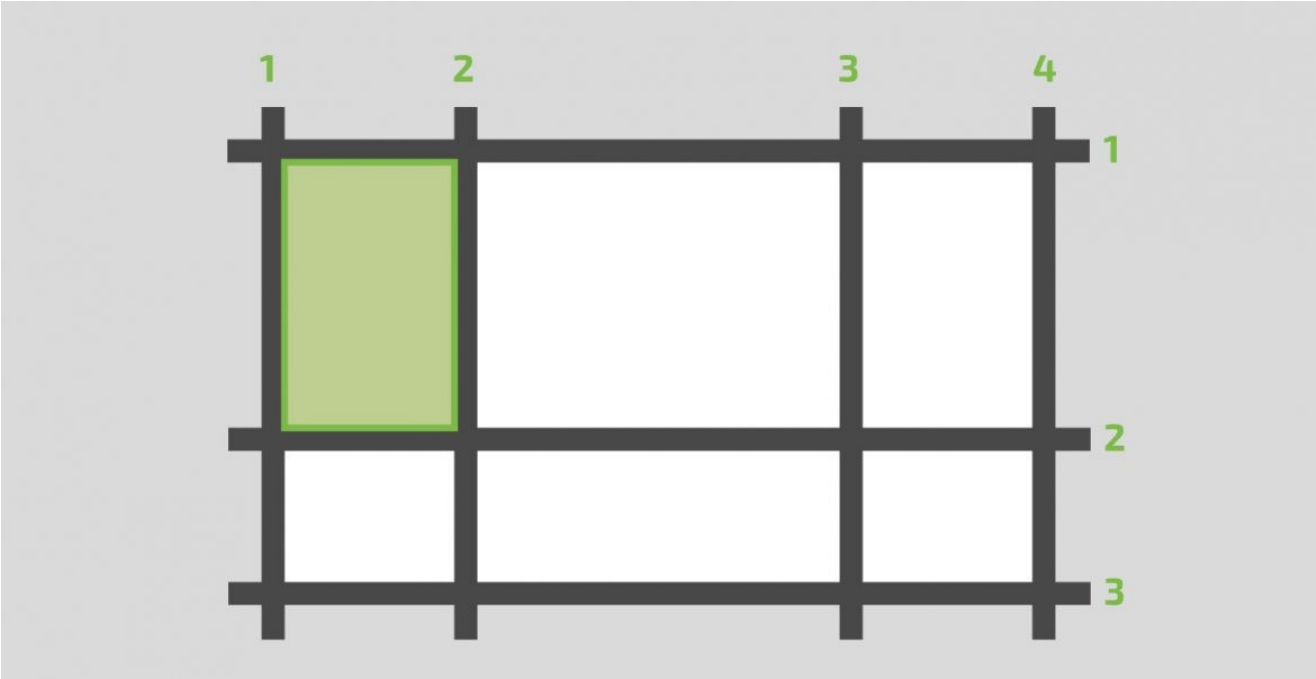
Si no marcas la posición de los elementos estos se colocarán por defecto de izquierda a derecha y de arriba abajo.

Si quieres personalizar la ubicación de los ítems puedes indicarle donde quieres que vaya cada uno tal que así:

```
.item-1 {
    grid-row-start: 1;
    grid-row-end: 2;
    grid-column-start: 1;
    grid-column-end:2;
 }
```

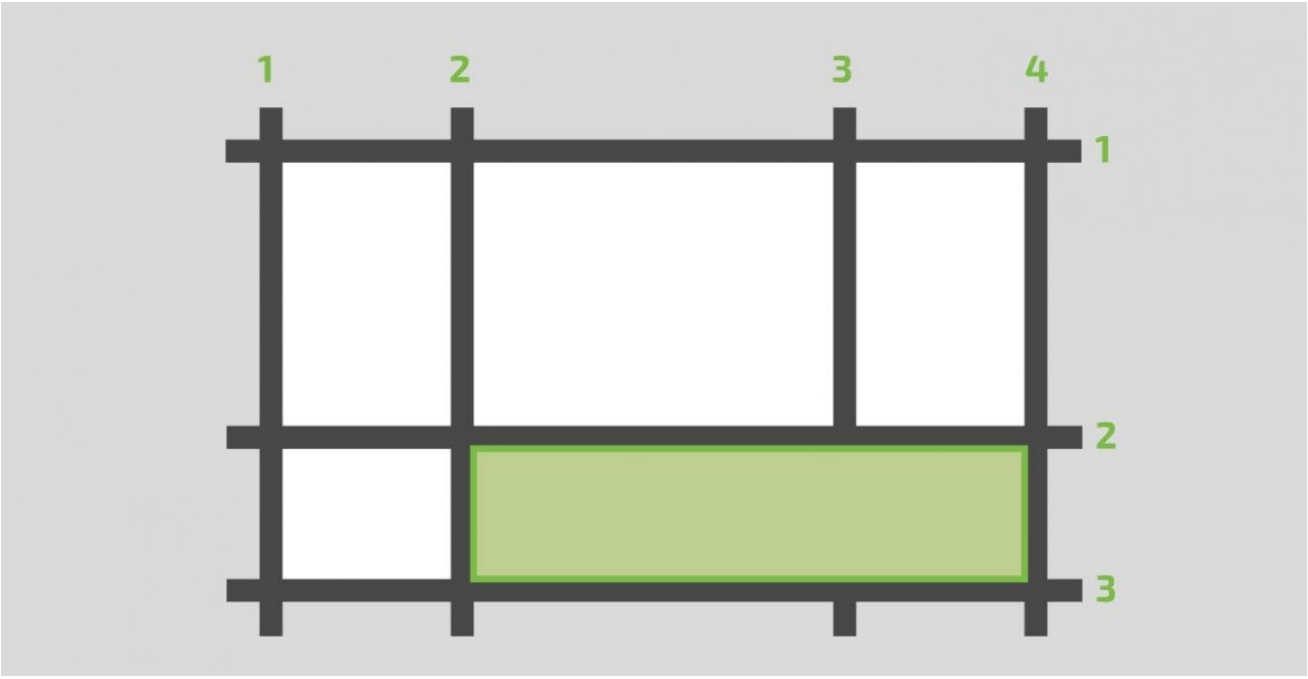
O de forma acortada:

```
.item-1 {
    grid-row: 1;
    grid-column: 1;
 }
```



Otro modo acortado sería utilizando *grid-row: X / Y* o *grid-column: X / Y*. Siendo X el número de fila o columna inicial e Y el número de fila o columna final.

```
.item-1 {
    grid-row: 2;
    grid-column: 2 / 4;
 }
```



De esta forma puedes repartir los elementos a tu gusto, incluso variando el espacio que ocupa cada uno para hacerlo más dinámico.

Esto es solo un avance de lo que puedes realizar con este sistema. Si quieres saber más, puedes pasarte por aquí y echarle un ojo a la guía.

Con CSS Grid ahorrarás tiempo picando código y tus diseños web serán más flexibles y funcionales. Si todavía no tienes hosting para montar tu página, pásate por nuestra web y obtén la mejor base para crear tus diseños. ¡A qué esperas para montar tu rejilla!

3. EJEMPLOS DE COMPOSICIONES DE PÁGINAS

3.1. OBJETIVO

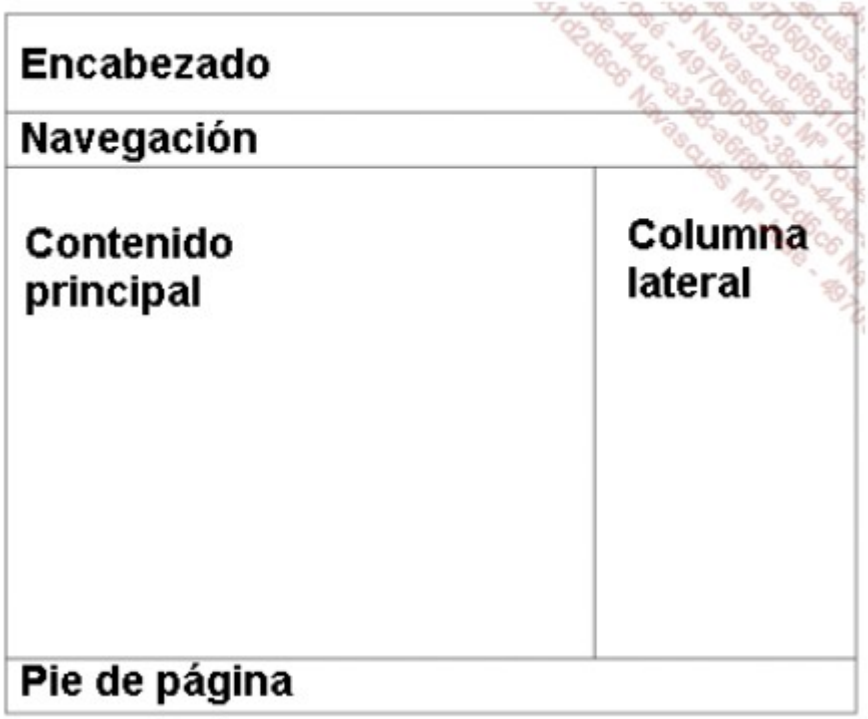
En este capítulo vamos a estudiar tres ejemplos de composición de página realizados con técnicas diferentes: con cajas flotantes, con una visualización en modo tabla y usando Responsive Web Design (diseño adaptativo).

3.2. COMPOSICIÓN DE PÁGINA CON CAJAS FLOTANTES

3.2.1. OBJETIVO

Vamos a componer una página usando cajas flotantes, una técnica que, aunque comienza a estar en declive, aún se utiliza mucho en la actualidad. Nuestro objetivo es que comprenda sus ventajas e inconvenientes, de manera que, si se lanza a crear un diseño de este tipo, lo haga con conocimiento de causa.

Este es el borrador de la composición de página que queremos crear.



- Esta página tendrá un **tamaño fijo**: una anchura fija de 960 píxeles.
- En el **encabezado**, colocaremos un logo a la izquierda, debajo un eslogan y a la derecha un campo de búsqueda de texto.
- Debajo del encabezado aparecerá un **menú de navegación**.
- A continuación, tendremos el **contenido** principal de esta página, con una **columna lateral** a la derecha (sidebar en inglés).
- Debajo del todo situaremos el **pie de página**, que contendrá diversos datos.
- Cada una de las cinco zonas tendrá un color de fondo diferente.

Puede encontrar este ejemplo en el archivo **12_01.html**.

3.2.2. EL CONTENEDOR GENERAL

Toda la estructura está comprendida en un contenedor general, en una caja identificada con el nombre contenedor. Esta caja tiene, por lo tanto, una anchura de 960 píxeles y está centrada en la ventana del navegador.

He aquí la estructura HTML:

```
<!DOCTYPE html>
<html lang="es">
<head>
   <title>Mi empresa</title>
   <meta charset="UTF-8" />
   <style>
      #contenedor {
         width: 960px;
         margin: 0 auto;
      }
   </style>
</head>
<body>
   <div id="contenedor">

   </div>
</body>
</html>
```

Para que resulte más cómodo, los estilos CSS están incluidos en el archivo HTML.

3.2.3. EL ENCABEZADO

El encabezado debe incluir tres elementos: un logo y un eslogan a la izquierda y un campo de búsqueda a la derecha.

Utilizaremos el elemento de HTML5 <header> para crear esta parte. En lo que respecta al formato del <header>, le aplicaremos una altura fija (height), un color de fondo (background-color) y un color de texto (color):

```
header {
   background-color: #000;
   height: 150px;
   color: #fff;
}
```

3.2.4. EL LOGO, EL ESLOGAN Y EL CAMPO DE BÚSQUEDA

En este <header> crearemos dos cajas : una para el logo y el eslogan, y otra para el campo de búsqueda.

```
<div id="contenedor">
   <header>
      <div id="logo">
         ...
      </div>
      <div id="busqueda">
         ...
      </div>
   </header>
</div>
```

El logo está insertado en un elemento <img>, en un párrafo <p>.

El eslogan es el texto de un elemento <h1>.

```
<div id="logo">
   <p><img src="tierra.jpg" alt="Logo" /></p>
   <h1>Planeta azul</h1>
</div>
```

El campo de búsqueda está situado en un elemento <form> con un campo <input>.

```
<div id="busqueda">
   <form method="post" action="#">
      <input type="text" name="search"
placeholder="buscar" value="" />
   </form>
</div>
```

Como hemos señalado al principio del capítulo, el logo y el eslogan deben aparecer en la parte izquierda del encabezado, mientras que el campo de búsqueda se mostrará a la derecha.

Para ello, usaremos la técnica de las cajas flotantes, con la propiedad float.

```
#logo {
   float: left;
}
#busqueda {
```

```
   float: right;
}
```

Esto es lo que obtenemos por ahora:

Añadiremos un margen izquierdo al logo y al eslogan, y un margen derecho al campo de búsqueda, para que estos elementos no aparezcan pegados a los bordes del encabezado.

```
#logo {
   float: left;
   margin-left: 20px;
}
#busqueda {
   float: right;
   margin-right: 20px;
}
```

Ahora queremos que el logo y el eslogan estén centrados verticalmente en el encabezado. Para ello, indicamos márgenes de 0, con objeto de suprimir los márgenes por defecto de los párrafos y poder colocar los elementos más fácilmente.

```
#logo p, #logo h1 {
   margin: 0;
}
```

A continuación, indicamos un margen superior al párrafo que contiene el logo y el eslogan, con objeto de que bajen un poco. Este margen lo calculamos visualmente, «a ojo».

```
#logo {
   float: left;
   margin: 25px 0 0 20px;
}
```

En este ejemplo, el margen superior es de 25 píxeles; el derecho y el inferior son de 0, y el de la izquierda, de 20 píxeles, como al principio.

Esto es lo que tenemos ahora:

En el caso del campo de búsqueda, esto es más fácil, ya que solo hay un elemento. Basta con indicar una altura de línea igual a la altura del <header>.

```
#busqueda {
   float: right;
   margin-right: 20px;
   line-height: 150px;
}
```

Este es el aspecto que presenta ahora:

Para terminar con el encabezado, aplicaremos un formato tipográfico al eslogan usando una fuente de Google:

```
<link href='http://fonts.googleapis.com/css?family=Oswald'
rel='stylesheet' type='text/css'>
```

Y

```
#logo h1 {
   font-family: 'Oswald', sans-serif;
   font-size: 2em;
}
```

El encabezado ahora queda así:

He aquí el código completo de esta primera parte:

```
<!DOCTYPE html>
<html lang="es">
<head>
   <title>Mi empresa</title>
   <meta charset="UTF-8" />
   <link href='http://fonts.googleapis.com/css?family=Oswald'
rel='stylesheet' type='text/css'>
   <style>
      #contenedor {
         width: 960px;
         margin: 0 auto;
      }
      header {
         background-color: #000;
         height: 150px;
         color: #fff;
```

```
      }
      #logo {
         float: left;
         margin: 25px 0 0 20px;
      }
      #logo h1 {
         font-family: 'Oswald', sans-serif;
         font-size: 2em;
      }
      #logo p, #logo h1 {
         margin: 0;
      }
      #busqueda {
         float: right;
         margin-right: 20px;
         line-height: 150px;
      }
   </style>
</head>
<body>
   <div id="contenedor">
      <header>
         <div id="logo">
            <p><img src="tierra.jpg" alt="Logo" /></p>
            <h1>Planeta azul</h1>
         </div>
         <div id="busqueda">
            <form method="post" action="#">
               <input type="text" name="search"
placeholder="buscar" value="" />
            </form>
         </div>
      </header>
   </div>
</body>
</html>
```

3.2.5. LA NAVEGACIÓN

La navegación está situada en un elemento <nav> que posee un color de fondo y una altura fija.

```
nav {
   background-color: lightblue;
   height: 30px;
}
```

La barra de navegación es un elemento <ul>. Cada vínculo <a> está situado en elementos <li>.

```
<nav>
   <ul id="menu">
      <li><a href="#">Inicio</a></li>
      <li><a href="#">Proyectos</a></li>
      <li><a href="#">Acciones</a></li>
      <li><a href="#">Reportajes</a></li>
   </ul>
```

```
</nav>
```

Los elementos <li> se muestran en modo de bloque en línea, a fin de que aparezcan alineados horizontalmente y de poder utilizar todas las propiedades de las cajas. Cada elemento <li> tiene un margen derecho de 20 píxeles para que quede separado del siguiente.

```
#menu li {
   display: inline-block;
   margin-right: 20px;
}
```

El elemento <ul> tiene los márgenes a 0 y un relleno a la izquierda de 20 píxeles. La altura de línea es igual a la altura del <nav>, a fin de obtener un centrado vertical.

```
#menu {
   margin: 0;
   padding: 0 0 0 20px;
   line-height: 30px;
}
```

Finalmente, los vínculos <a> no tendrán decoración, para que no aparezcan las líneas de subrayado.

```
#menu a {
   text-decoration: none;
}
```

Este es el resultado:

3.2.6. LA ZONA CENTRAL

La zona central incluye, a la izquierda, el contenido principal de la página y, a la derecha, la columna lateral.

En el contenedor general ( ), utilizaremos el elemento de HTML5 <main>. En este elemento vamos a crear una caja y usaremos el elemento de HTML5 <aside>.

```
<div id="contenedor">
   <header>
      ...
   </header>
   <nav>
      ...
   </nav>
   <main>
      <div id="content">
```

```
            ...
        </div>
        <aside>
            ...
        </aside>
    </main>
</div>
```

Los elementos y <aside> incorporan texto con los elementos <h2>, <h3> y <p>.

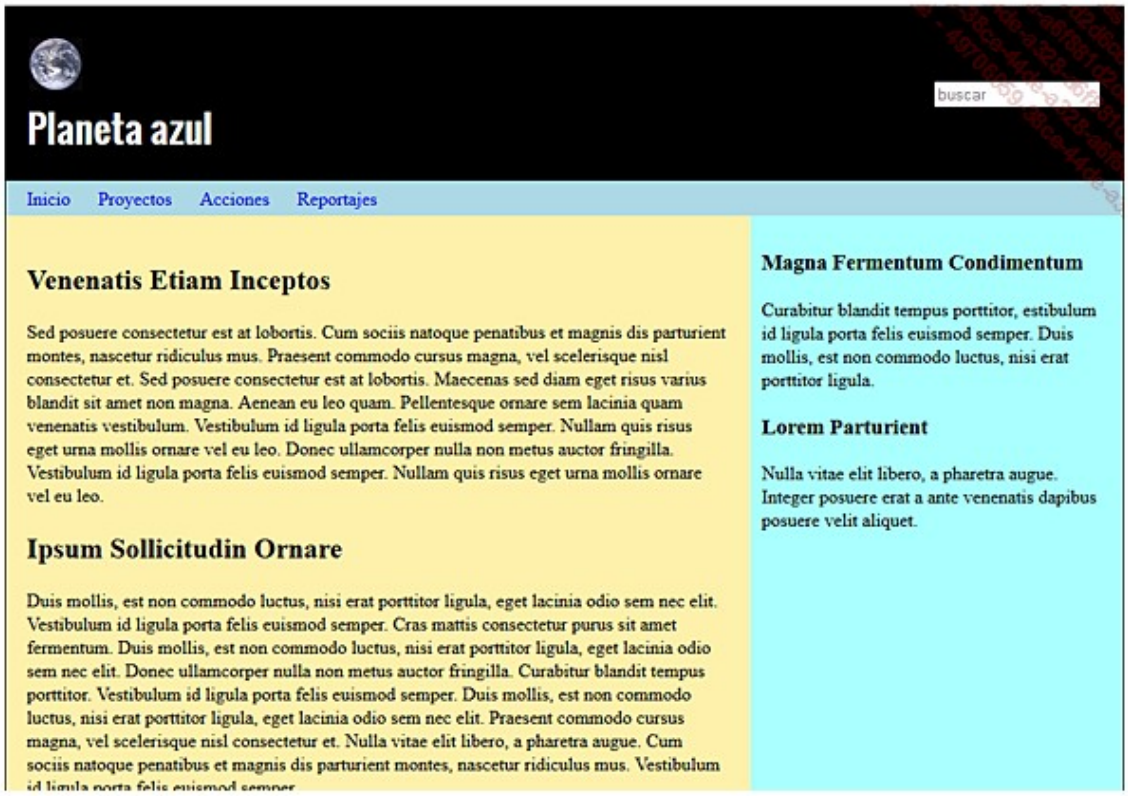
En lo que respecta al diseño de esta área, hemos decidido aplicar una anchura de 640 píxeles al contenido, con un relleno de 20 píxeles, y una anchura de 320 píxeles para la columna lateral derecha, con un relleno de 10 píxeles. Para que resulte más fácil hacer los cálculos, utilizamos la propiedad box-sizing: padding-box;. Estas dos cajas deben flotar sobre su izquierda.

```
#content, aside {
    float: left;
    box-sizing: padding-box;
}
#content {
    width: 640px;
    padding: 20px;
}
aside {
    width: 320px;
    padding: 10px;
}
```

Deseamos aplicar un color de fondo diferente a cada una de estas dos zonas, o de otro modo no sabremos de antemano la cantidad de texto que pueden contener estas dos áreas. No podemos utilizar la propiedad background-color. Estamos obligados a usar la técnica de las «falsas columnas» en el contenedor <main>. Vamos a utilizar una imagen de 960 píxeles de ancho por 1 píxel de alto. Esta imagen contendrá los dos colores que queremos aplicar y la usaremos de fondo de <main> con una repetición únicamente vertical, desde el ángulo superior izquierdo. Además, para poder visualizar siempre el color, es preciso emplear la propiedad overflow: auto;.

```
main{
    background: url('columnas.png') repeat-y left top;
    overflow: auto;
}
```

Esto es lo que obtenemos:



3.2.7. EL PIE DE PÁGINA

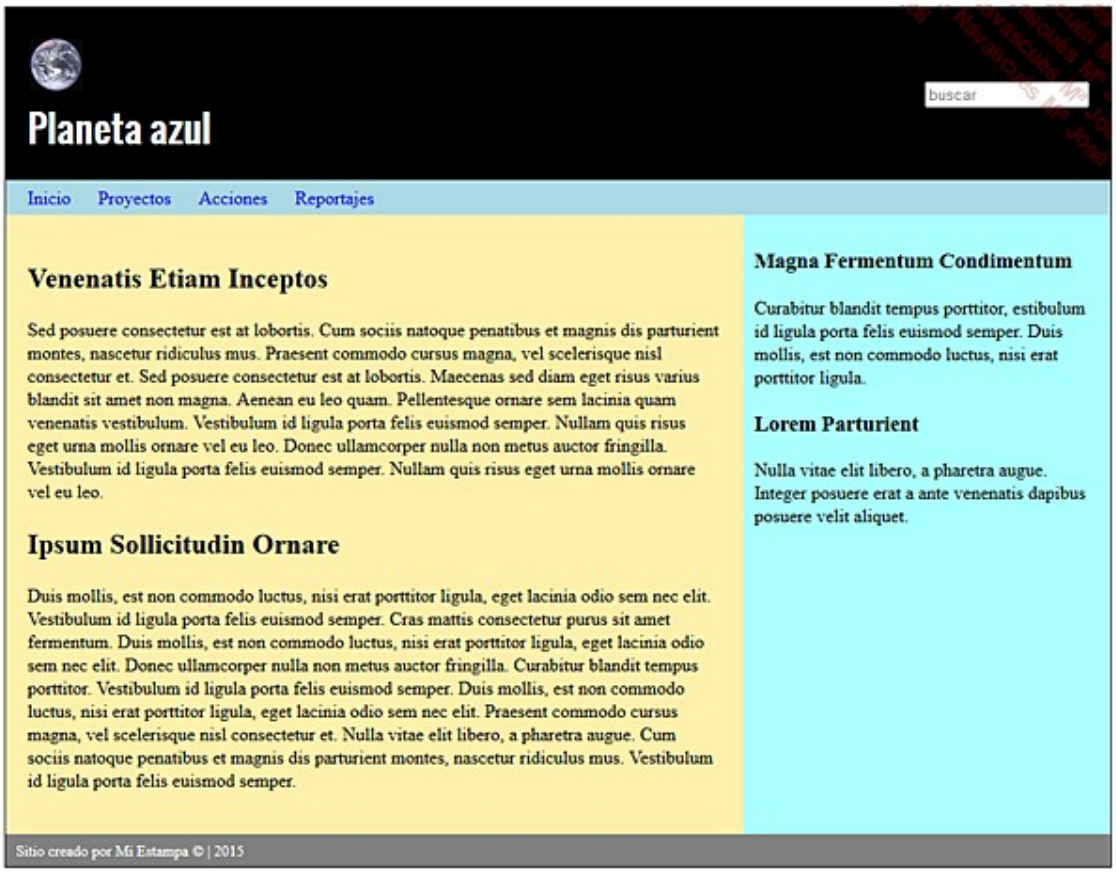
El pie de página, el elemento de HTML5 <footer>, no contiene más que un párrafo.

```
<footer>
   <p>Sitio creado por Mi Estampa &copy; | 2015</p>
</footer>
```

Incluirá un fondo de color, una altura fija y su contenido estará centrado verticalmente y será de color blanco.

```
footer {
   height: 30px;
   background-color: gray;
   font-size: small;
}
footer p {
   margin: 0 0 0 10px;
   line-height: 30px;
   color: #fff;
}
```

Este es el resultado:



3.2.8. EL CÓDIGO COMPLETO DE ESTE EJEMPLO

```
<!DOCTYPE html>
<html lang="es">
<head>
   <title>Mi empresa</title>
   <meta charset="UTF-8" />
   <link href='http://fonts.googleapis.com/css?family=Oswald'
rel='stylesheet' type='text/css'>
   <style>
      #contenedor {
         width: 960px;
         margin: 0 auto;
      }
      /* El encabezado */
      header {
         background-color: #000;
         height: 150px;
         color: #fff;
      }
      #logo {
         float: left;
         margin: 25px 0 0 20px;
      }
      #logo h1 {
         font-family: 'Oswald', sans-serif;
         font-size: 2em;
      }
```

```
      #logo p, #logo h1 {
         margin: 0;
      }
      #busqueda {
         float: right;
         margin-right: 20px;
         line-height: 150px;
      }
      /* La barra de navegación */
      nav {
         background-color: lightblue;
         height: 30px;
      }
      #menu {
         margin: 0;
         padding: 0 0 0 20px;
         line-height: 30px;
      }
      #menu li {
         display: inline-block;
         margin-right: 20px;
      }
      #menu a {
         text-decoration: none;
      }
      /* La parte central */
      main{
         background: url('columnas.png') repeat-y left top;
         overflow: auto;
      }
      #content, aside {
         float: left;
         box-sizing: padding-box;
      }
      #content {
         width: 640px;
         padding: 20px;
      }
      aside {
         width: 320px;
         padding: 10px;
      }
      /* El pie de página */
      footer {
         height: 30px;
         background-color: gray;
         font-size: small;
      }
      footer p {
         margin: 0 0 0 10px;
         line-height: 30px;
         color: #fff;
      }
   </style>
</head>
<body>
```

```
   <div id="contenedor">
      <header>
         <div id="logo">
            <p><img src="tierra.jpg" alt="Logo" /></p>
            <h1>Planeta azul</h1>
         </div>
         <div id="busqueda">
            <form method="post" action="#">
               <input type="text" name="search"
placeholder="buscar" value="" />
            </form>
         </div>
      </header>
      <nav>
         <ul id="menu">
            <li><a href="#">Inicio</a></li>
            <li><a href="#">Proyectos</a></li>
            <li><a href="#">Acciones</a></li>
            <li><a href="#">Reportajes</a></li>
         </ul>
      </nav>
      <main>
         <div id="content">
            <h2>Venenatis Etiam Inceptos</h2>
            <p>Sed posuere consectetur est at lobortis. Cum
sociis natoque penatibus et magnis dis pa...</p>
            <h2>Ipsum Sollicitudin Ornare</h2>
            <p>Duis mollis, est non commodo...</p>
         </div>
         <aside>
            <h3>Magna Fermentum Condimentum</h3>
            <p>Curabitur blandit tempus...</p>
            <h3>Lorem Parturient</h3>
            <p>Nulla vitae elit libero...</p>
         </aside>
      </main>
      <footer>
         <p>Sitio creado por Mi Estampa &copy; | 2015</p>
      </footer>
   </div>
</body>
</html>
```

3.2.9. LOS INCONVENIENTES

Veamos los inconvenientes de esta técnica:

- En lo que respecta al encabezado, la alineación vertical del logo y del eslogan se hacen a ojo, sin precisión.
- En el encabezado y el pie de página, la alineación vertical se obtiene con la propiedad line-height, que se desvía de su uso estándar y se realiza por cálculo.
- La altura de las cajas con contenido textual no puede conocerse de ninguna manera ni fijarse de forma previa.

- En cuanto a los colores para el fondo de la parte central, estamos obligados a usar una imagen de fondo que se repite. Ello no es práctico y aumenta ligeramente el tiempo de carga.

3.3 COMPOSICIÓN DE PÁGINA CON FLEX

Puedes encontrar diferentes ejemplos en las URLs:

https://codepen.io/enriquearkas/pen/XJwBEr

https://codepen.io/cchambers/pen/KdBpa