ThePower business school

Display: Flex CSS 💪🎨

Flex o flexbox es un sistema de elementos flexibles que se creó para dar más posibilidades a los desarrollos de páginas web. Lo primero que hay que saber a la hora de usar flex son sus características y elementos básicos:

- Contenedor: será el padre del resto de ítems. El contenedor o padre será el elemento fundamental en flex a la hora de establecer propiedades.
    - Eje principal: por defecto este eje será el horizontal o eje x.
    - Eje secundario: será el eje perpendicular al eje principal. Si el principal hemos dicho que por defecto es el horizontal, el eje secundario por defecto será el vertical o y.
- Elemento hijo: será cada uno de los ítems que se encuentre “encerrado” en el elemento padre o caja contenedora.

Os ponemos un ejemplo sencillo de código:

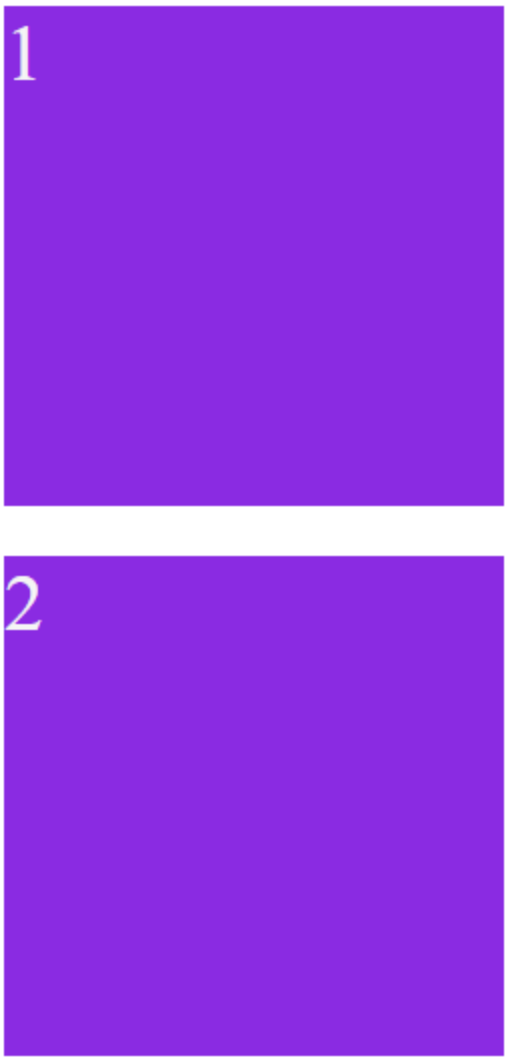
```
<div class="container"> <!-- Elemento padre -->
  <div class="hijo item-1">1</div> <!-- Elementos hijos -->
  <div class="hijo item-2">2</div> <!-- Elementos hijos -->
</div>
```

```
.hijo{
    width: 100px;
    height: 100px;
    background-color: blueviolet;
    color: whitesmoke;
    margin: 10px;
}
```

Si lo que queremos es colocar estos elementos hijos mediante flexbox, debemos establecer el valor de la propiedad display en flex para activar las posibilidades. Solo con esto ya veremos que los elementos se posicionan distinto al comportamiento por defecto.

Siguiendo el ejemplo de código anterior, si no aplicamos el display en flex, veremos en nuestro navegador lo siguiente:

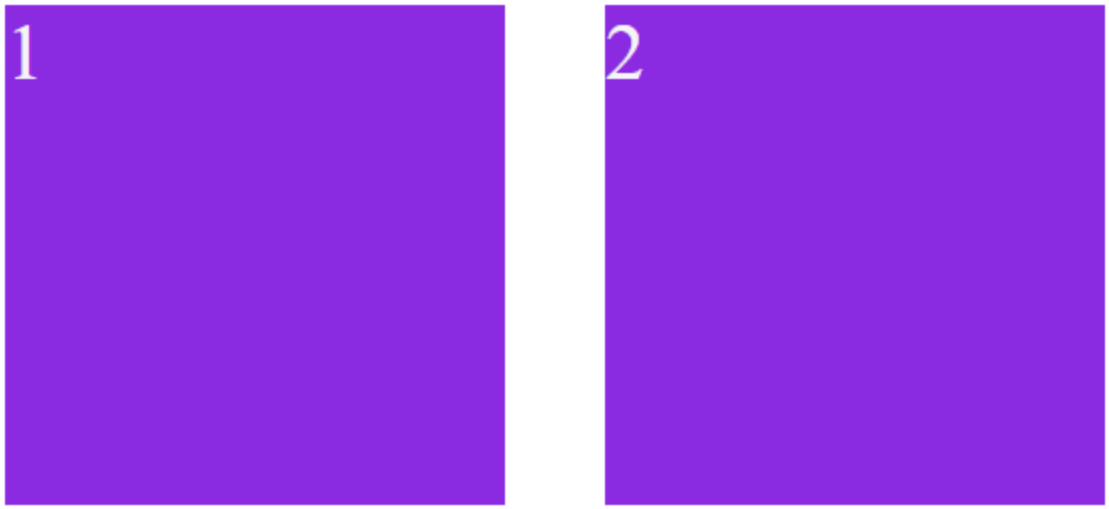
```
.hijo{
    width: 100px;
    height: 100px;
    background-color: blueviolet;
    color: whitesmoke;
    margin: 10px;
}
```



Lo que veremos SIN FLEX

En cambio, si en vez de el css que os pusimos más arriba, aplicamos este:

```
.container {
  display: flex
}
.hijo{
    width: 100px;
    height: 100px;
    background-color: blueviolet;
    color: whitesmoke;
    margin: 10px;
}
```



Lo que veremos en el navegador usando FLEX:

Para establecer cuál es el eje principal y cuál es el secundario, tenemos la propiedad flex-direction que se habrá desbloqueado al haber establecido el display en flex.

Vale bien, tenemos la propiedad flex-direction disponible pero… ¿cómo funciona? esta propiedad establecerá la dirección del eje principal, y sus valores disponibles son:

- row, que establece el eje principal en horizontal.

- row-reverse, que establece el eje principal en horizontal INVERTIDO, es decir, colocará los items contenedores en el orden inverso.
- column, que establece el eje principal en vertical.
- column-reverse, que hace del eje principal el vertical INVERTIDO, es decir, colocará los items contenedores en el orden inverso.

Vamos a verlo con código:

```
<div class="container">
        <div class="hijo item-1">1</div>
        <div class="hijo item-2">2</div>
        <div class="hijo item-3">3</div>
    </div>
```

```
.container {
  display: flex;
  flex-direction: column
}
.hijo{
    width: 100px;
    height: 100px;
    background-color: blueviolet;
    color: whitesmoke;
    margin: 10px;
}
```

Si jugáis con el valor de la propiedad que os hemos resaltado en rojo y cambiáis column por row, row-reverse o column-reverse podréis ver cómo cambian los elementos!

Como veis, hasta ahora hemos trabajado con pocos elementos, pero puede darse el caso que un elemento contenedor al que le queremos dar flex contenga una cantidad muy grande de elementos hijos… esto puede hacer que los hijos reduzcan tanto su tamaño para entrar en el ancho disponible que no se visualicen correctamente.

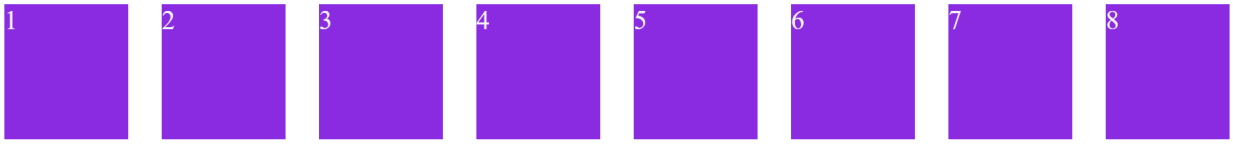
Para arreglar esto tenemos una propiedad adicional que se llama flex-wrap, y lo que hace es que convierte nuestro contenedor en multilínea, desbordando el contenido.

Los valores de esta propiedad son: wrap, nowrap y wrap-reverse. Estos valores evitarán o no el desbordamiento del contenido. Hay que aclarar que el valor por defecto será nowrap.

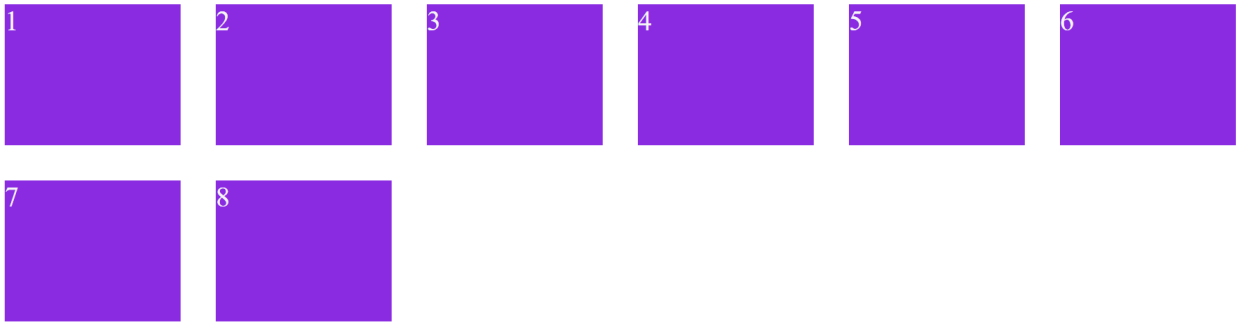
Vamos a ver un ejemplo:

```
<div class="container">
        <div class="hijo item-1">1</div>
        <div class="hijo item-2">2</div>
        <div class="hijo item-3">3</div>
        <div class="hijo item-4">4</div>
        <div class="hijo item-5">5</div>
        <div class="hijo item-6">6</div>
        <div class="hijo item-7">7</div>
        <div class="hijo item-8">8</div>
</div>
```

```
.container {
  display: flex;
  flex-wrap: nowrap;
}
.hijo{
    width: 100px;
    height: 80px;
    background-color: blueviolet;
    color: whitesmoke;
    margin: 10px;
}
```



Si lo que queremos es que cada caja mantenga su valor de width y se pueda desbordar el contenido, establecemos la propiedad flex-wrap: wrap y veremos lo siguiente:



Una vez vistas estas dos propiedades y sus valores, os damos una buena noticia y es que existe una tercera propiedad que sirve como atajo, y podremos poner los valores de las anteriores dentro de ella. Esto nos ahorra código por lo que nos vendrá genial. Esta propiedad se llama flex-flow y os dejamos con un ejemplo de código.

```
.container{
  flex-flow: column nowrap;
}
```

Será lo mismo que si escribiésemos:

```
.container{
  flex-direction: column;
  flex-wrap: nowrap;
}
```

Os podéis fijar que en los ejemplos que os hemos ido dejando, para establecer espacios entre items o elementos hijos hemos usado la propiedad margin, pero existe una que se creó con flex para facilitar precisamente esto. Esta propiedad se llama row/column-gap.

Estas propiedades establecen un hueco o gap entre items y previene posibles errores que puedan surgir del uso de margin o padding.

- row-gap: establece espacio entre filas, y solo funcionará si tenemos flex-direction en column.
- column-gap: espacio entre columnas, y solo funcionará con el flex-direction en row.

Existe un atajo que sería usar una única propiedad gap y darle de una sola vez valor a los gaps tanto en row como en column:

```
.container {
  gap: 4px 8px;
  /* Equivalente a */
  row-gap: 4px;
  column-gap: 8px;
}
```

```
.container{
  gap: 4px;
  /* Equivalente a */
  row-gap: 4px;
  column-gap: 4px;
}
```

Ahora que os hemos introducido a flex y a sus propiedades más básicas, podemos pasar a algo más avanzado y enseñar en más profundidad cómo alinear elementos. Para esta tarea tenemos varias propiedades disponibles:

- justify-content: Actúa sobre el eje principal, y puede tener valores de:
  - flex-start: agrupa los elementos hijos al inicio del eje principal.
  - flex-end: agrupa los hijos al final del eje.
  - center: los agrupa en el centro.
  - space-between: distribuye los hijos dejando espacio entre ellos.
  - space-around: deja espacio alrededor de los hijos.
  - space-evenly: dejará el mismo espacio alrededor de cada elemento hijo.

```
<div class="container">
        <div class="hijo item-1">1</div>
        <div class="hijo item-2">2</div>
        <div class="hijo item-3">3</div>
</div>
```


```
.container {
  display: flex;
    justify-content: center;
}
.hijo{
    width: 100px;
    height: 80px;
    margin: 10px;
    background-color: blueviolet;
    color: whitesmoke;
}
```

Podéis ir cambiando los valores de la propiedad resaltada en rojo y veréis los distintos comportamientos.

- Align-items: alineará los hijos respecto al eje secundario, según hayamos establecido nosotros.
  - flex-start: alinea los hijos al inicio.
  - flex-end: alinea los hijos al final.
  - center: alinea los hijos al centro.
  - stretch: alinea los hijos estirándolos para que ocupen la totalidad del contenedor.
  - baseline: alinea los hijos en relación a la base del contenido de los hijos del contenedor.

```
<div class="container">
        <div class="hijo item-1">1</div>
        <div class="hijo item-2">2</div>
        <div class="hijo item-3">3</div>
</div>
```


```
.container {
    height: 100vh;
    display: flex;
    justify-content: center;
    align-items: center;
}
.hijo{
    width: 100px;
    margin: 10px;
    background-color: blueviolet;
    color: whitesmoke;
}
```

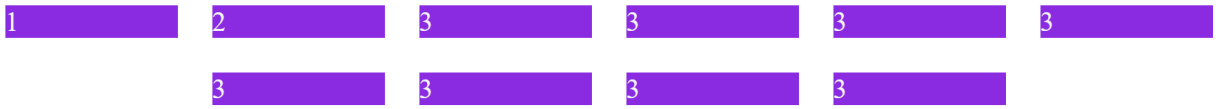
Podéis ir jugando con los diferentes valores para ver cómo actuarán.

- Align-content: será un caso especial de la anterior propiedad align-items. Esta propiedad nos servirá cuando tengamos un contenedor multilínea creado con flex-wrap. Así, la propiedad align-content establecerá cómo queremos ordenar cada una de las líneas. Sus valores pueden ser:
    - flex-start: agrupa los ítems al **inicio** del eje principal.
    - flex-end: agrupa los ítems al final del eje principal.
    - center: agrupa los ítems al centro del eje principal.

- space-between: agrupa los de inicio a final.
- space-around: agrupa los ítems dejando el mismo espacio a los lados de cada uno.
- stretch: alinea los elementos estirándolos para que ocupen la totalidad del espacio.

```
<div class="container">
        <div class="hijo item-1">1</div>
        <div class="hijo item-2">2</div>
        <div class="hijo item-3">3</div>
        <div class="hijo item-3">3</div>
        <div class="hijo item-3">3</div>
        <div class="hijo item-3">3</div>
        <div class="hijo item-3">3</div>
        <div class="hijo item-3">3</div>
        <div class="hijo item-3">3</div>
        <div class="hijo item-3">3</div>
    </div>
```

```
.container {
    height: 100vh;
    display: flex;
    justify-content: center;
    align-items: center;
    flex-wrap: wrap;
    align-content: center;
}
.hijo{
    width: 100px;
    margin: 10px;
    background-color: blueviolet;
    color: whitesmoke;
}
```

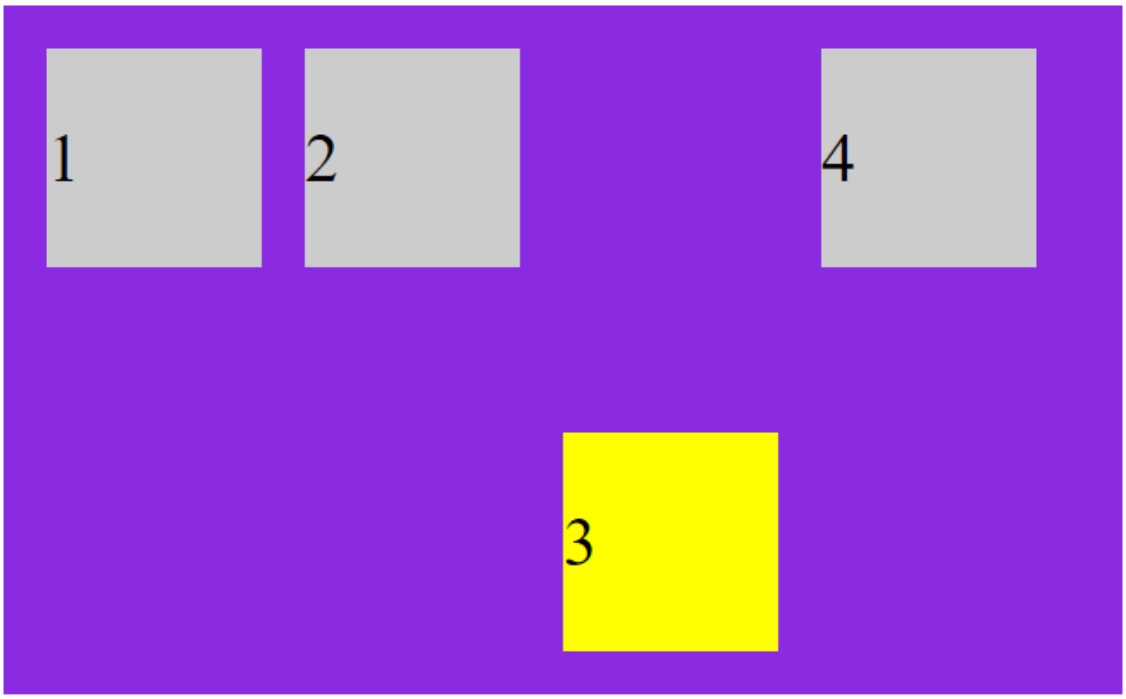


Podéis jugar con los distintos valores de la propiedad align-content y ver cómo cambia el resultado.

Hasta ahora hemos visto únicamente propiedades de alineación de contenedores padres, pero, ¿qué pasa con los hijos? pues para los hijos existen propiedades concretas que nos permitirán alinearlos por separado de manera muy sencilla. Vamos a ver estas propiedades.

Dentro de lo que hemos visto anteriormente, si lo que queremos es alinear un elemento hijo concreto, tenemos disponible la propiedad align-self, que actuará exactamente igual que align-items pero solo en un hijo concreto. Veamos un ejemplo:

```
<div class="flex-container">
  <div class="flex-item"><p>1</p></div>
  <div class="flex-item"><p>2</p></div>
  <div class="flex-item flex-start"><p>3</p></div>
  <div class="flex-item"><p>4</p></div>
</div>
```

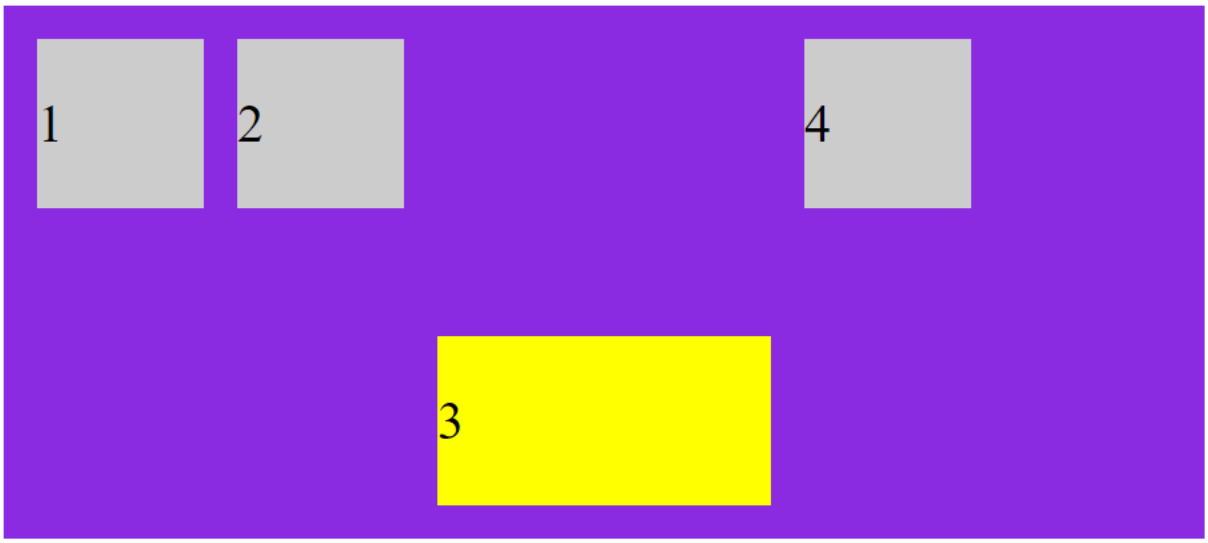


Como siempre, podéis ir jugando con los valores de la propiedad resaltada para ver cómo cambia el resultado.

```
.flex-container{
  width: 250px;
  height:150px;
  padding:5px;
  margin: 10px auto;
  background-color:blueviolet;
  display:flex;
  display: flex;
  align-items: flex-start;
}
.flex-item{
  display: inherit;
  width:50px;
  background-color:#ccc;
  margin:5px;
}

.flex-container .flex-start{
    background-color: yellow;
  align-self: flex-end;
}
```

Tenemos la propiedad flex-basis que definirá el tamaño base por defecto de los hijos. Este valor se puede aplicar en unidades, porcentajes, etc, pero también se puede usar la palabra content para indicar que se ajuste el tamaño al contenido del elemento. Este valor content será el valor por defecto de la propiedad flex-basis.

```
<div class="flex-container">
  <div class="flex-item"><p>1</p></div>
  <div class="flex-item"><p>2</p></div>
  <div class="flex-item flex-start"><p>3</p></div>
  <div class="flex-item"><p>4</p></div>
</div>
```



Podéis jugar con diferentes valores y ver los resultados!

```
.flex-container{
  width: 350px;
  height:150px;
  padding:5px;
  margin: 10px auto;
  background-color:blueviolet;
  display:flex;
  display: flex;
  align-items: flex-start;
```

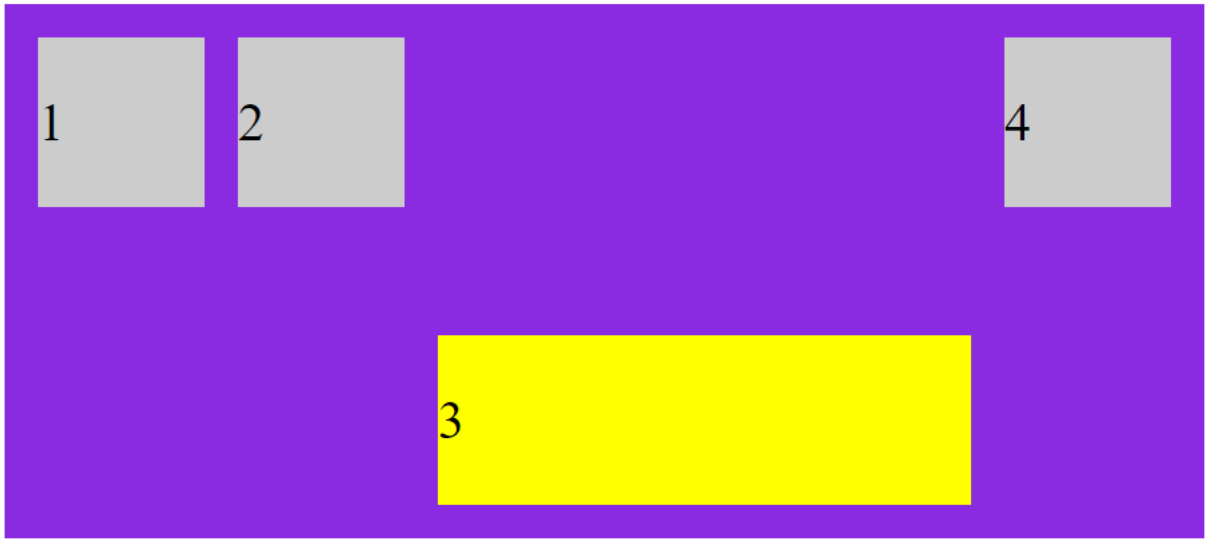
```
}
.flex-item{
  display: inherit;
  width:50px;
  background-color:#ccc;
  margin:5px;
}

.flex-container .flex-start{
    background-color: yellow;
    align-self: flex-end;
    flex-basis: 100px;
}
```

Siguiendo con el ejemplo anterior, hemos usado flex-basis para indicar que el elemento con el número 3 queremos que tenga un flex-basis de 100px, lo que sustituye al valor de 50px que tenía por defecto cada elemento.

Así, podemos entender que flex-basis será similar a la propiedad width, con la diferencia que flex-basis se usará en los hijos de un contenedor exclusivamente.

Otra propiedad que tenemos disponible es flex-grow, que tendrá efecto únicamente cuando se haya definido un flex-basis. Esta propiedad hace que los hijos cubran el tamaño total del contenedor padre y su definición también puede ser que es una propiedad relacionada con el factor de crecimiento de los elementos. Por defecto todos los hijos tendrán un flex-grow: 0. Esta propiedad es algo compleja pero vamos a ver un ejemplo:

```
<div class="flex-container">
  <div class="flex-item"><p>1</p></div>
  <div class="flex-item"><p>2</p></div>
  <div class="flex-item flex-start"><p>3</p></div>
  <div class="flex-item"><p>4</p></div>
</div>
```



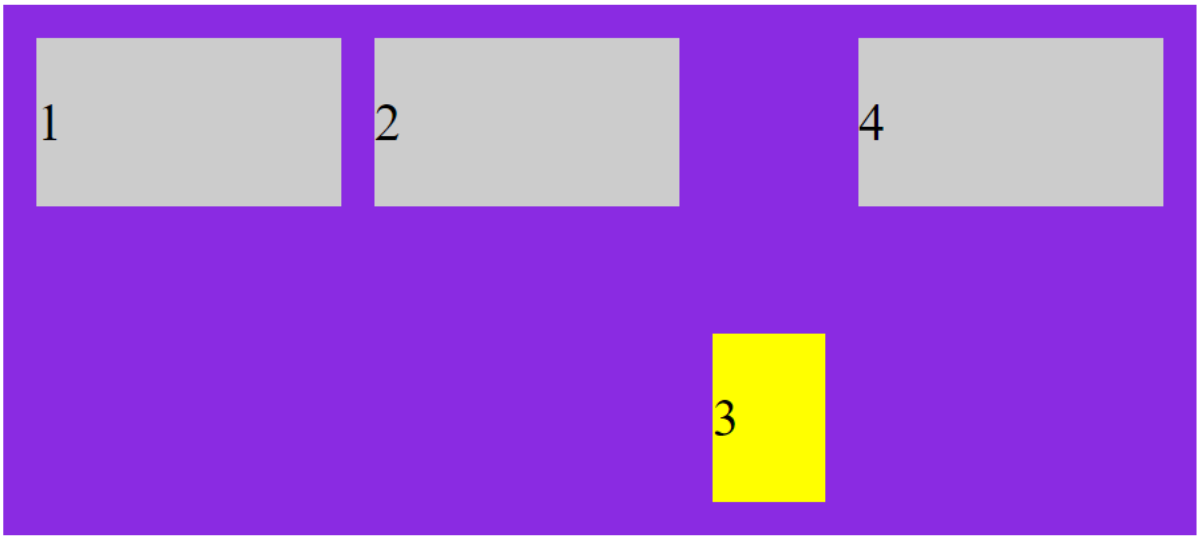

```
.flex-container{
  width: 350px;
  height:150px;
  padding:5px;
  margin: 10px auto;
  background-color:blueviolet;
  display:flex;
  display: flex;
  align-items: flex-start;
}
.flex-item{
  display: inherit;
  flex-basis: 50px;
  background-color:#ccc;
  margin:5px;
}

.flex-container .flex-start{
    background-color: yellow;
    align-self: flex-end;
    flex-grow: 1;
}
```

Vamos a explicar un poco este ejemplo para que lo entendáis. Lo que hemos hecho ha sido fijar un flex-basis de 50px en cada uno de los elementos hijos, por eso veréis que cada uno tiene ese tamaño excepto el elemento 3, que tiene un valor de 160px debido al valor de flex-grow de 1 que os hemos resaltado. Esto implica que su factor de

crecimiento es 1 y debido a esto el ancho de ese elemento concreto se ha adaptado para rellenar el total del ancho del elemento contenedor padre.

Por último tenemos la propiedad flex-shrink, que actuará de forma contraria a la que acabamos de ver flex-grow, ya que si flex-grow lo que hacía era indicar el factor de crecimiento, flex-shrink indicará lo contrario, el factor de **DECRECIMIENTO**. Las condiciones para que esta propiedad actúe deben ser que haya un flex-basis definido y que los elementos hijos NO cubran el acho total del contenedor padre.

```
<div class="flex-container">
  <div class="flex-item"><p>1</p></div>
  <div class="flex-item"><p>2</p></div>
  <div class="flex-item flex-start"><p>3</p></div>
  <div class="flex-item"><p>4</p></div>
</div>
```



Como vemos, el elemento 3 ha decredido debido a la propiedad flex-shrink con respecto al resto de elementos hijos.

```
.flex-container{
  width: 350px;
  height:150px;
  padding:5px;
  margin: 10px auto;
```

```
  background-color:blueviolet;
  display:flex;
  display: flex;
  align-items: flex-start;
}
.flex-item{
  display: inherit;
  flex-basis: 150px;
  background-color:#ccc;
  margin:5px;
}

.flex-container .flex-start{
    background-color: yellow;
    align-self: flex-end;
    flex-shrink: 2;
}
```

ATAJO: Mediante la propiedad flex podremos establecer los valores de las tres propiedades que acabamos de ver para alinear elementos hijos, por lo que:

```
.item {
  /* flex: <flex-grow> <flex-shrink> <flex-basis> */
  flex: 1 3 35%;

  /* Equivalente a: */
  flex-grow: 1;
  flex-shrink: 3;
  flex-basis: 35%;
}
```

Os dejamos para terminar unos juegos y recursos con los que podréis practicar flex y convertiros en unos auténticos expertos:

Flexbox Froggy

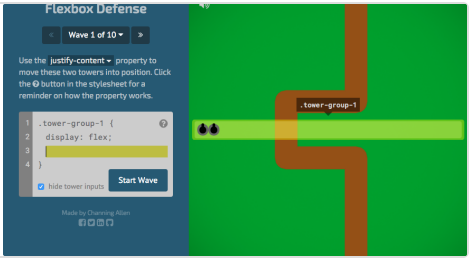
A game for learning CSS flexbox

https://flexboxfroggy.com/#es

Flexbox Defense

Your job is to stop the incoming enemies from getting past your defenses. Unlike other tower defense games, you must position your towers using CSS!

http://www.flexboxdefense.com/



Play Flex Box Adventure - CSS Game to Learn Flexbox

1. You often stumble and try to figure out which combination of Flex Box properties makes the browser do what you want it to do. 2. You want to create complex web layouts without constantly looking at

https://codingfantasy.com/games/flexboxadventure

A Complete Guide to Flexbox | CSS-Tricks

Our comprehensive guide to CSS flexbox layout. This complete guide explains everything about flexbox, focusing on all the different possible properties for the parent element (the flex container) and

https://css-tricks.com/snippets/css/a-guide-to-flexbox/

¿Qué hemos aprendido?

1. Qué es flex, qué nos ofrece y sus propiedades más esenciales.
2. Propiedades específicas para alinear elementos.
3. Las propiedades con las que podremos maquetar elementos hijos concretos dentro de los contenedores padres.
4. Recursos extras con los que podéis completar el aprendizaje de manera divertida!