








Flexbox Avanzado y Posicionamiento CSS

Dominando layouts modernos

Fundamentos de la Web - Clase 5

Objetivos de Hoy

¿Qué aprenderemos?

-  Dominar la propiedad `flex` y sus componentes
-  Crear layouts flexibles con proporciones personalizadas
-  Mantener estilos consistentes entre navegadores
-  Usar la propiedad `position` para control preciso
-  Aplicar diseño modular (filosofía de Legos)
-  Construir tarjetas de usuario profesionales
-  Desarrollar páginas de perfil completas




Propiedad Flex

La clave de los diseños flexibles

¿Qué es la propiedad `flex`?

La magia de los espacios flexibles

Imagina que tienes una **torta** y quieres repartirla entre tus amigos:

-  Si todos toman **partes iguales** → cada uno recibe lo mismo
-  Si uno quiere **el doble** → ese amigo recibe 2 porciones
-  El tamaño se **adapta automáticamente** según cuántos amigos haya

Eso mismo hace `flex` con el espacio disponible en un contenedor



Anatomía de la propiedad `flex`

`flex: grow shrink basis`

La propiedad `flex` es un **atajo** que combina 3 propiedades:

```
.elemento {  
  flex: 1 1 0%; /* Forma completa */  
}
```

Esto es equivalente a:

```
.elemento {  
  flex-grow: 1; /* ¿Cuánto crece? */  
  flex-shrink: 1; /* ¿Cuánto se encoge? */  
  flex-basis: 0%; /* ¿Tamaño base? */  
}
```

Desglosando **flex**

1. **flex-grow** → Capacidad de **crecer** para llenar espacio
 - **0** = No crece
 - **1** = Crece para tomar espacio disponible
 - **2** = Crece el doble que los de valor 1
2. **flex-shrink** → Capacidad de **encogerse** si falta espacio
 - **0** = No se encoge
 - **1** = Se puede encoger
3. **flex-basis** → Tamaño **base** antes de distribuir espacio
 - **0%** = Parte de cero (común con flex)
 - **auto** = Usa el tamaño del contenido



Ejemplo: Columnas Iguales

3 columnas del mismo tamaño

```
.row {  
  display: flex; /* Activar flexbox */  
}  
.col {  
  flex: 1; /* Cada columna toma el mismo espacio */  
  background-color: lightblue;  
  padding: 20px;  
  margin: 10px;  
}
```

```
<div class="row">  
  <div class="col">Columna 1</div>  
  <div class="col">Columna 2</div>  
  <div class="col">Columna 3</div>  
</div>
```

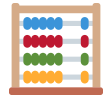
Resultado: Todas las columnas se reparten el espacio equitativamente



Ejemplo: Columnas con Proporciones

```
.row {  
  display: flex;  
}  
.col {  
  flex: 1; /* Tamaño normal */  
  background-color: lightblue;  
  padding: 20px;  
}  
.col-2 {  
  flex: 2; /* El DOBLE de grande */  
  background-color: lightcoral;  
}
```

```
<div class="row">  
  <div class="col">1/3</div>  
  <div class="col col-2">2/3</div>  <!-- Toma el doble de espacio -->  
</div>
```

La Matemática Flex

¿Cómo se calculan los tamaños?

Si tienes columnas con `flex: 1`, `flex: 2`, `flex: 3`:

1. **Suma todos los valores:** $1 + 2 + 3 = 6$ partes totales
2. **Divide el espacio:**
 - Primera columna = $1/6$ del espacio (16.66%)
 - Segunda columna = $2/6$ del espacio (33.33%)
 - Tercera columna = $3/6$ del espacio (50%)

¡El espacio se distribuye proporcionalmente!



Ejemplo Completo: Diferentes Proporciones

```
.row {  
  display: flex;  
  margin-bottom: 20px;  
  gap: 10px; /* Espacio entre columnas */  
}
```

```
.col { flex: 1; background-color: lightblue; padding: 20px; }  
.col-2 { flex: 2; background-color: lightcoral; }  
.col-3 { flex: 3; background-color: lightgreen; }
```

```
<!-- 4 columnas iguales (1/4 cada una) -->  
<div class="row">  
  <div class="col">1/4</div>  
  <div class="col">1/4</div>  
  <div class="col">1/4</div>  
  <div class="col">1/4</div>  
</div>
```



Ejemplo Completo: Diferentes Proporciones (cont.)

```
<!-- 1/3 y 2/3 -->
<div class="row">
  <div class="col">1/3</div>
  <div class="col col-2">2/3</div>
</div>

<!-- 2/5 y 3/5 -->
<div class="row">
  <div class="col col-2">2/5</div>
  <div class="col col-3">3/5</div>
</div>
```

La belleza de flex: No necesitas calcular píxeles. Todo se ajusta automáticamente.

? Pregunta Rápida

¿Qué pasará con estas columnas?

```
.row {  
  display: flex;  
}  
.col-a { flex: 3; }  
.col-b { flex: 1; }
```

```
<div class="row">  
  <div class="col-a">Columna A</div>  
  <div class="col-b">Columna B</div>  
</div>
```

¿Qué proporción de espacio tomará cada columna?

Piensa unos segundos... 🤔

✓ Respuesta

```
<div class="row">
  <div class="col-a">Columna A</div> <!-- flex: 3 -->
  <div class="col-b">Columna B</div> <!-- flex: 1 -->
</div>
```

Respuesta correcta: A toma 75% y B toma 25%

¿Por qué?

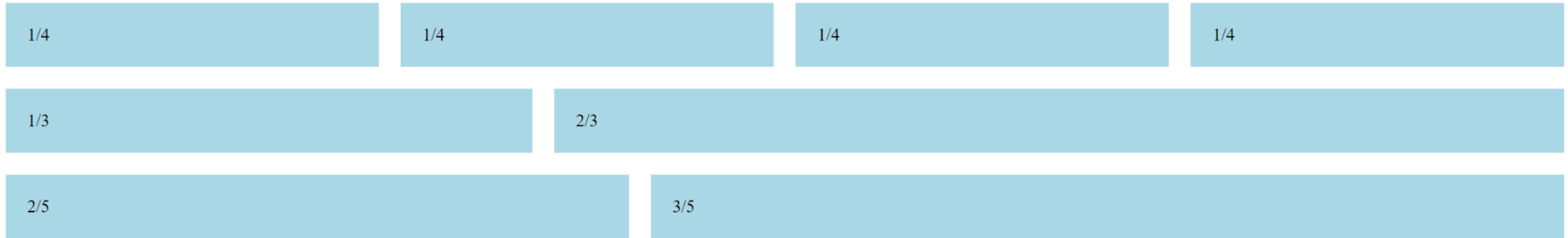
- Total: $3 + 1 = 4$ partes
- Columna A: $3/4 = 75\%$
- Columna B: $1/4 = 25\%$

¡Excelente! Ahora entiendes las proporciones flex 🎉



Ejercicio: Columnas Flex

¡Hora de practicar!: Crea un archivo `columnas-flex.html` que replique:



Requisitos:

1. 4 filas con diferentes proporciones
2. Usar `display: flex` en los contenedores
3. Usar la propiedad `flex` para las columnas



Tiempo: 15 minutos

Ejercicio: Columnas Flex (Pistas)

Pistas:

- Primera fila: 4 columnas de `flex: 1` (1/4 cada una)
- Segunda fila: 3 columnas de `flex: 1` (1/3 cada una)
- Tercera fila: `flex: 1` y `flex: 2` (1/3 y 2/3)
- Cuarta fila: `flex: 2` y `flex: 3` (2/5 y 3/5)

Estructura base:

```
<div class="row">  
  <div class="col">Contenido</div>  
</div>
```



Recursos: Profundiza en Flex

Para aprender más



Guía completa sobre flex:

[CSS-Tricks: Understanding flex-grow, flex-shrink, and flex-basis](#)

Conceptos clave que encontrarás:

- Cómo funcionan los tres valores de flex
- Casos de uso comunes
- Trucos y patrones útiles
- Ejemplos interactivos

Estilos Consistentes en CSS

Normalize y Reset

El Problema de los Navegadores

Cada navegador es diferente

Imagina que diseñas un sitio web precioso en Chrome, pero cuando lo abres en Firefox o Safari... ¡se ve diferente!

¿Por qué pasa esto?

Cada navegador tiene su propia **hoja de estilos incorporada** llamada `user agent stylesheet` . Esto hace que:

- Un `<h1>` tenga diferente margen en IE vs Chrome
- Los botones se vean distintos
- Los espacios varíen entre navegadores

Resultado: Tu diseño se ve inconsistente 😓

Soluciones: Reset vs Normalize

Hay dos estrategias principales para lograr consistencia:

1. Reset CSS

- Borra TODOS los estilos predeterminados
- Empiezas desde cero
- Control total pero más trabajo

2. Normalize CSS






- Homogeniza los estilos entre navegadores
- Mantiene los estilos útiles
- Corrige bugs comunes

¿Cuál usar? Depende de tu proyecto, pero **Normalize** es más popular.

Normalize.css

La opción recomendada

¿Qué hace Normalize?

-  Preserva los estilos útiles (no los borra todos)
-  Normaliza los estilos entre navegadores
-  Corrige bugs y inconsistencias comunes
-  Mejora la usabilidad con sutiles mejoras
-  Es mantenido activamente

Normalize.css (cont.)

La opción recomendada

¿Cuándo usarlo?

En **casi todos tus proyectos**, especialmente si quieres:

- Mantener semántica HTML
- Evitar resetear todo desde cero
- Tener una base sólida y moderna



Cómo agregar Normalize.css

Opción 1: CDN (Recomendado para aprender)

Agrega este `<link>` en el `<head>` de tu HTML **ANTES** de tu propio CSS:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Mi Sitio</title>

  <!-- 1. Primero Normalize -->
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/normalize/8.0.1/normalize.min.css" />

  <!-- 2. Luego tus estilos -->
  <link rel="stylesheet" href="styles.css">
</head>
```



Obtener el enlace: <https://cdnjs.com/libraries/normalize>

Reset CSS

El enfoque minimalista

¿Qué hace Reset CSS?

- ✗ Borra TODOS los márgenes
- ✗ Borra TODOS los paddings
- ✗ Borra TODOS los estilos predeterminados

Ejemplo del reset más simple:

```
* {  
  margin: 0;  
  padding: 0;  
  box-sizing: border-box;  
}
```

Reset CSS (cont.)

El enfoque minimalista

¿Cuándo usarlo?

- Cuando quieres control TOTAL desde cero
- Proyectos muy específicos
- Cuando sabes exactamente qué estilos quieres



Ejemplo: Reset CSS Completo

```
* { /* Reset de todos los elementos */  
  margin: 0;  
  padding: 0;  
  box-sizing: border-box;  
}  
ul, ol { /* Reset de listas */  
  list-style: none;  
}  
a { /* Reset de enlaces */  
  text-decoration: none;  
  color: inherit;  
}  
img { /* Reset de imágenes */  
  max-width: 100%;  
  display: block;  
}
```

Coloca esto al inicio de tu archivo CSS

Reset vs Normalize: ¿Cuál elegir?

Característica	Reset	Normalize
Filosofía	Borra todo	Homogeniza
Estilos útiles	✗ Los elimina	✓ Los preserva
Control	📈 Total	🎯 Equilibrado
Trabajo extra	✍ Más	✍ Menos
Recomendado para	Expertos	Principiantes
Popularidad	★ ★ ★	★ ★ ★ ★ ★

Consejo: Empieza con **Normalize**, es más amigable.

Tip para Entrevistas

Demuestra que sabes de consistencia

Pregunta común de entrevista:

"¿Cómo haces que tu sitio se vea igual en todos los navegadores?"

Respuesta profesional:

"Uso Normalize.css para homogenizar los estilos predeterminados entre navegadores. También valido mi HTML y CSS con el validador de W3C para detectar errores que pueden causar renderizados inconsistentes. Adicionalmente, pruebo en múltiples navegadores durante el desarrollo."

Bonus: Menciona <https://validator.w3.org/> para validar código

Recursos: Profundiza en Reset/Normalize

 Colección de CSS Resets:

[Global CSS Reset Styles](#)

 Guía de Normalize:

[Normalize CSS - GeeksforGeeks](#)

Nota importante: 

Mucha documentación está en inglés. Si no estás acostumbrado/a a leer en inglés, ¡es buen momento para empezar a practicar! La mayoría de recursos de programación están en ese idioma.

Propiedad Position

El GPS de tus elementos



¿Qué es `position`?

Control preciso sobre la ubicación

La propiedad `position` es como el **GPS de tus elementos**. Te permite decirle a cada elemento exactamente dónde debe ir y cómo debe comportarse.

Analogía: Imagina que estás poniendo **stickers en tu laptop**:

- Puedes pegarlos en el flujo normal (uno al lado del otro)
- Puedes ponerlos encima de otros
- Puedes hacer que uno siempre esté visible aunque muevas la tapa
- Puedes posicionarlos exactamente donde quieras

Eso mismo hace `position` con tus elementos HTML

Los 4 Valores de Position

Cada uno con un comportamiento único

```
position: static;    /* Por defecto - flujo normal */  
position: relative; /* Relativo a su posición original */  
position: absolute; /* Relativo al padre posicionado */  
position: fixed;     /* Fijo en la ventana del navegador */
```

Vamos a ver cada uno en detalle 🙌

1 Position: Static, comportamiento por defecto

```
.elemento {  
    position: static; /* Valor predeterminado */  
}
```

- Es el valor **por defecto** de todos los elementos
- Los elementos siguen el **flujo normal** del documento
- No puedes usar `top`, `right`, `bottom`, `left`
- No puedes usar `z-index`

¿Cuándo usarlo?

- Casi nunca lo escribes explícitamente
- Es útil para **resetear** un position que habías cambiado

2 Position: Relative, relativa a sí mismo

```
.div-relativo {  
  position: relative;  
  top: 10px;    /* Se mueve 10px hacia ABAJO */  
  left: 20px;   /* Se mueve 20px hacia la DERECHA */  
}
```

Analogía: Es como hacer espacio en el sofá sin echar a tus amigos al suelo.

Características:

- Se mueve desde su **posición original**
- **Deja su espacio reservado** (los demás elementos no se mueven)
- Puedes usar `top`, `right`, `bottom`, `left`
- Sirve como **referencia para position: absolute**



Ejemplo: Position Relative

```
<div class="caja">Caja Normal</div>  
<div class="caja caja-relativa">Caja Relativa</div>  
<div class="caja">Caja Normal</div>
```

```
.caja {  
  width: 200px;  
  height: 100px;  
  background-color: lightblue;  
  margin: 10px;  
}  
  
.caja-relativa {  
  position: relative;  
  top: 20px;      /* Se mueve 20px hacia abajo */  
  left: 30px;     /* Se mueve 30px hacia la derecha */  
  background-color: lightcoral;  
}
```

Observa: El espacio original sigue reservado

3 Position: Absolute, elemento flotante

```
.div-absoluto {  
  position: absolute;  
  top: 50px;      /* 50px desde el BORDE SUPERIOR */  
  right: 30px;    /* 30px desde el BORDE DERECHO */  
}
```

Analogía: Es como si el elemento pudiera **flotar en el aire** en la posición exacta que quieres, sin importar lo que haya debajo.

Características:

- Se **sale del flujo normal** (los demás elementos se mueven como si no existiera)
- Se posiciona **relativo al padre posicionado más cercano**
- Si no hay padre posicionado, se posiciona relativo al `<body>`



Ejemplo: Position Absolute

```
<div class="contenedor">  
  Contenedor  
  <div class="hijo-absoluto">Hijo Absoluto</div>  
</div>
```

```
.contenedor {  
  position: relative; /* Padre posicionado */  
  width: 400px;  
  height: 300px;  
  background-color: lightgray;  
}  
.hijo-absoluto {  
  position: absolute;  
  top: 50px; /* 50px desde el top del CONTENEDOR */  
  right: 30px; /* 30px desde el right del CONTENEDOR */  
  background-color: lightcoral;  
  padding: 20px;  
}
```



Regla de Oro: Absolute + Relative

El patrón más común

Padre: `position: relative` (sin moverse)

Hijo: `position: absolute` (posicionado exactamente)

```
.card {  
  position: relative; /* Padre de referencia */  
  width: 300px;  
  height: 400px;  
}  
  
.badge {  
  position: absolute; /* Se posiciona relativo a .card */  
  top: 10px;  
  right: 10px;  
  /* Este badge estará en la esquina superior derecha de .card */  
}
```

Caso de uso: Insignias, iconos de notificación, botones de cerrar

4 Position: Fixed, siempre visible

```
.navbar {  
  position: fixed;  
  top: 0;  
  width: 100%;  
  /* Esta navbar siempre estará en la parte superior */  
}
```

Analogía: Es como **pegar algo en tu pantalla**. No importa cómo te desplaces por la página, ese elemento se queda fijo en el mismo lugar.

Características:

- Se fija a la ventana del navegador
- No se mueve al hacer scroll
- Ideal para barras de navegación, botones flotantes, modales



Ejemplo: Navbar Fixed

```
<nav class="navbar">Navegación Fija</nav>
<div class="contenido">
  <!-- Mucho contenido aquí -->
</div>
```

```
.navbar {
  position: fixed;
  top: 0;           /* Pegado arriba */
  left: 0;          /* Pegado a la izquierda */
  width: 100%;      /* Ancho completo */
  background-color: #333;
  color: white;
  padding: 15px;
  z-index: 1000;    /* Por encima de todo */
}
.contenido {
  margin-top: 60px; /* Espacio para la navbar */
}
```

? Pregunta Rápida, ¿Qué position usarías?

Necesitas crear un botón de "Volver arriba" que:

- Siempre esté visible en la esquina inferior derecha
- No se mueva cuando hagas scroll
- Esté por encima de todo el contenido

¿Qué valor de `position` usarías?

- a) `static`
- b) `relative`
- c) `absolute`
- d) `fixed`

Piensa unos segundos... 🤔

✓ Respuesta correcta: d) **fixed**

```
.btn-volver-arriba {  
  position: fixed;  
  bottom: 20px;      /* 20px desde abajo */  
  right: 20px;       /* 20px desde la derecha */  
  z-index: 1000;     /* Por encima de todo */  
  background-color: #2196F3;  
  color: white;  
  padding: 15px;  
  border-radius: 50%;  
  cursor: pointer;  
}
```

- ✓ **fixed** mantiene el elemento visible durante el scroll
- ✓ Se posiciona relativo a la ventana del navegador
- ✓ Perfecto para elementos que deben estar siempre accesibles



Comparación: Position en Acción

Valor	Flujo Normal	Usa top/left/etc	Referencia	Uso Común
static	✓ Sí	✗ No	-	Predeterminado
relative	✓ Sí*	✓ Sí	Sí mismo	Padre para absolute
absolute	✗ No	✓ Sí	Padre posicionado	Overlays, badges
fixed	✗ No	✓ Sí	Ventana	Navbars, botones

*Reserva su espacio original



Ejercicio: Tarjeta con Badge

¡Aplica lo aprendido!

Crea una tarjeta de producto con un badge "NUEVO" en la esquina superior derecha

Requisitos:

1. Tarjeta con `position: relative`
2. Badge con `position: absolute`
3. Badge posicionado en top-right
4. Imagen de producto
5. Título y precio



Tiempo: 10 minutos



Ejercicio: Tarjeta con Badge (Pista)

```
<div class="producto">
  <div class="badge">NUEVO</div>
  
  <h3>Nombre del Producto</h3>
  <p class="precio">$99.99</p>
</div>
```

```
.producto {
  position: relative; /* Padre de referencia */
  /* ... más estilos ... */
}

.badge {
  position: absolute;
  top: 10px;
  right: 10px;
  /* ... más estilos ... */
}
```



Recursos: Position en Profundidad

 Documentación oficial MDN:

[CSS Position](#)

Temas que encontrarás:

- Explicaciones detalladas de cada valor
- Ejemplos interactivos
- Casos de uso avanzados
- Z-index y stacking context
- Position: sticky (¡un bonus!)

Diseñando con Legos

Filosofía de diseño modular




El Concepto de Legos

Construye de lo grande a lo pequeño

Imagina que estás armando algo con **bloques de Lego**:

1. **Primero** colocas los bloques grandes (la base)
2. **Luego** añades bloques medianos (las secciones)
3. **Finalmente** agregas los detalles pequeños (los acabados)

Lo mismo aplica al diseño web:

1.  Define la **estructura principal** (header, main, footer)
2.  Divide en **secciones** (navbar, hero, about, services)
3.  Agrega los **detalles** (botones, textos, imágenes)

Metodología: De Grande a Pequeño

Paso a paso

✗ Error común:

Empezar por los detalles (colores de botones, tipografías)

✓ Enfoque correcto:

1. Wireframe (estructura general)
↓
2. Bloques principales (contenedores)
↓
3. Secciones (componentes)
↓
4. Detalles (estilos finales)

Beneficio: Menos errores, más organización, mejor código



Ejemplo: Página Principal

1. Bloques Grandes:

```
<header></header>  
<main></main>  
<footer></footer>
```

2. Secciones Medianas:

```
<header>  
  <nav></nav>  
</header>  
<main>  
  <section class="hero"></section>  
  <section class="about"></section>  
  <section class="services"></section>  
</main>
```



Ejemplo: Página Principal (cont.)

3. Detalles Pequeños:

```
<header>
  <nav>
    <div class="logo"></div>
    <ul class="menu"></ul>
    <button class="cta"></button>
  </nav>
</header>
```






Ahora sí: Con la estructura clara, podemos añadir estilos



Planificación Visual

Antes de codificar

Aunque sea tentador empezar a codificar inmediatamente, es mejor:

1.  **Dibuja** la estructura en papel o pizarra
2.  **Identifica** los bloques principales
3.  **Nombra** cada sección
4.  **Decide** qué componentes necesitas
5.  **Codifica** con claridad

Resultado: Menos errores, código más limpio, trabajo más rápido

Estrategia de Desarrollo

De afuera hacia adentro

Para la Tarjeta de Usuario:

1. Contenedor principal (.profile-card)
↓
2. Header azul (.header)
↓
3. Contenido (.profile-content)
↓
4. Imagen de perfil (.profile-img)
↓
5. Textos y botón (detalles)

Ventaja: Cada paso es pequeño y manejable

Checklist de Diseño

Antes de empezar a codificar

- ☐ ¿Identifiqué todos los bloques principales?
- ☐ ¿Tengo claro qué secciones necesito?
- ☐ ¿Sé qué componentes son reutilizables?
- ☐ ¿Planeé la estructura HTML?
- ☐ ¿Tengo un sistema de nombres claro?
- ☐ ¿Entiendo el diseño visual?

Si respondiste **SÍ** a todo: ¡Estás listo para codificar! 🚀

Ejercicios Core

¡Hora de la acción!






Ejercicio Core 1: Tarjeta de Usuario

Tu primer reto importante

Descripción:

Usa HTML y CSS para recrear una tarjeta de presentación profesional.

Lo que practicarás:

-  Estructura HTML semántica
-  Position (absolute + relative)
-  Flexbox básico
-  Border-radius para círculos
-  Box-shadow para profundidad

Tiempo estimado: 60-90 minutos



Ejercicio Core 1: Especificaciones

Elementos requeridos:

1. Contenedor de tarjeta con bordes redondeados
2. Header azul (#00BCD4)
3. Imagen de perfil circular (position: absolute)
4. Nombre en fuente grande y bold
5. Ubicación y título profesional
6. Descripción breve
7. Botón "edit profile"



Ejercicio Core 1: Imagen de referencia

Imagen de referencia:



Linn Olsen

Bergen, Noruega

Diseñadora de UX/UI | Desarrolladora Frontend | Amante de la naturaleza

Apasionada por crear experiencias digitales hermosas y funcionales. Cuando no está codificando, disfruta explorando los fiordos noruegos y capturando la belleza de la naturaleza en fotografías.

[edit profile](#)

Tips para el Ejercicio

Paso 1: HTML

- Estructura básica
- Contenedores y elementos

Paso 2: CSS Básico

- Colores y fuentes
- Tamaños y espaciados

Paso 3: CSS Avanzado

- Posicionamiento de la imagen
- Border-radius para círculo
- Sombras y detalles finales







Ejercicio Core 2: Página de Perfil

El desafío completo

Descripción:

Crea una página de perfil completa tipo LinkedIn/Facebook.

Lo que practicarás:

-  Layout con Flexbox (columnas 2:1)
-  Múltiples componentes
-  Navbar con position: fixed
-  Listas de conexiones
-  Diseño responsive
-  Organización de código

Tiempo estimado: 3-4 horas



Ejercicio Core 2: Componentes

1. Navbar superior fija

- Logo/título
- Menú de navegación
- Botón de logout

2. Columna izquierda (flex: 2)

- Tarjeta de perfil
- Sección de educación


3. Columna derecha (flex: 1)

- Solicitudes de conexión
- Tus conexiones



Ejercicio Core 2: Imagen de Referencia

Perfil[Inicio](#)[Mi Perfil](#)[Encontrar Conexiones](#)[Cerrar Sesión](#)



Linn Olsen

Bergen, Noruega
Diseñadora de UX/UI | Desarrolladora Frontend | Amante de la naturaleza


Apasionada por crear experiencias digitales hermosas y funcionales. Cuando no está codificando, disfruta explorando los fiordos noruegos y capturando la belleza de la naturaleza en fotografías.


[Edit profile](#)

Education


Coding Dojo - Bootcamp de Desarrollo Full Stack
Dic 2019 - Mar 2020
Triple cinturón negro en Python, MERN y C#


Solicitudes de Conexión (2)


 Kai Schröder ☒ ☐


 Amira El-Masry ☒ ☐


Tus Conexiones (500+)


 Adrien Desplatz

 Aarav Patel

 Lina Kovačević

 Mateo Hernández

 Yuna Kim

 Liam O'Brien

Consejo importante:

✨ ¡Empieza de arriba hacia abajo y de afuera hacia adentro!

Estrategia para el Ejercicio 2

Divide y conquistarás

Fase 1: Estructura

Navbar → Container → Left Column → Right Column

Fase 2: Componentes individuales

Profile Card → Education → Connections → Requests





Fase 3: Detalles

Estilos finales → Ajustes → Perfeccionamiento

Recuerda: Trabaja en equipo, CSS puede ser complicado

Trabajo en Equipo

Beneficios de trabajar en grupo:

-  Diferentes perspectivas
-  Detectar errores más rápido
-  Aprender de los demás
-  Mantenerse motivado



Sugerencias:

- Divide las tareas (uno hace navbar, otro las cards)
- Comparte código y ayuda
- Revisa el código de tus compañeros
- Pregunta cuando tengas dudas



Recursos Adicionales

Para seguir aprendiendo



Documentación:

-  [MDN Web Docs](#) - Referencia completa
-  [CSS-Tricks](#) - Guías y tutoriales

Juegos:

-  [Flexbox Froggy](#) - Aprende Flexbox
-  [Grid Garden](#) - Aprende CSS Grid

Herramientas:

-  [Flexbox Cheatsheet](#) - Referencia visual
-  [Can I Use](#) - Compatibilidad de navegadores



Resumen de Conceptos Clave

1. Propiedad `flex` :

- Atajo de `flex-grow` , `flex-shrink` , `flex-basis`
- Distribuye espacio proporcionalmente

2. Normalize/Reset:

- Normalize: homogeniza estilos
- Reset: borra todos los estilos

3. Position:

- `relative` : relativo a sí mismo
- `absolute` : relativo al padre
- `fixed` : relativo a la ventana

Llévate esto a casa

1. Planifica antes de codificar

- Dibuja la estructura
- Identifica componentes
- Piensa en bloques (Legos)

2. Usa las herramientas correctas

- Flexbox para layouts
- Position para control preciso
- Normalize para consistencia




3. Practica, practica, practica

- Los ejercicios Core son fundamentales
- Experimenta y rompe cosas




Próximos Pasos

Continúa tu aprendizaje

Esta semana:

-  Completa los ejercicios Core
-  Juega Flexbox Froggy (todos los niveles)
-  Experimenta con position en tus proyectos

Próxima clase:

-  CSS Grid (layouts bidimensionales)
-  Diseño responsive
-  Media queries

Proyecto sugerido:

Crea tu propia tarjeta de presentación personal usando lo aprendido

Para convertirte en mejor desarrollador

1. Lee documentación en inglés

- La mayoría está en inglés
- Practica poco a poco
- Usa traductores al principio

2. Valida tu código

- HTML: <https://validator.w3.org/>
- CSS: <https://jigsaw.w3.org/css-validator/>
- Buena práctica para entrevistas

3. Inspecciona sitios web

- F12 en tu navegador
- Estudia cómo están hechos

Criterios de Éxito

Nivel Principiante:

- [] Entiendo qué hace `flex: 1`, `flex: 2`, etc.
- [] Sé la diferencia entre `normalize` y `reset`
- [] Puedo usar `position: relative` y `absolute`

Nivel Intermedio:

- [] Creo layouts flexibles con proporciones custom
- [] Uso `position` para casos complejos
- [] Planifico estructura antes de codificar

Criterios de Éxito (cont.)

Nivel Avanzado:

- ☐ Completo los ejercicios Core sin ayuda
- ☐ Aplico diseño modular (Legos)
- ☐ Mi código es limpio y organizado

 ¡Felicidades!

Ya dominas Flexbox avanzado y Position

Estás listo para crear layouts profesionales ✨

? Preguntas

¿Alguna duda sobre Flexbox o Position?

 ¡Excelente Trabajo!

¡Nos vemos en la siguiente clase con CSS Grid!

No olvides completar los ejercicios Core 