Materia TPI: CSS Parte II

Contenido

Tema 5: Posicionamiento y Display

□ 1. Propiedad `display`

La propiedad `display` define **cómo se comporta un elemento en el flujo del documento**.

Valor	Descripción
`block`	El elemento ocupa todo el ancho disponible. Inicia en una nueva línea.
`inline`	El elemento ocupa solo el ancho de su contenido. No inicia nueva línea.
`inline-block`	Como `inline`, pero permite usar `width`, `height`, `margin`, `padding`.
`none`	Oculta el elemento completamente (no ocupa espacio).
`flex`	Convierte el contenedor en un **flexbox**, ideal para layouts responsivos.
`grid`	Convierte el contenedor en una **rejilla**, útil para diseños bidimensionales.

Ejemplo:

```
nav {
  display: flex;
  justify-content: space-between;
}
```

📌 2. Propiedad `position`

La propiedad `position` define **cómo se posiciona un elemento respecto a su contenedor o al documento**.

Valor	Descripción
`static`	Valor por defecto. El elemento sigue el flujo normal del documento.
`relative`	Se posiciona **relativo a su posición original**. Se puede mover con `top`, `left`, etc.
`absolute`	Se posiciona **respecto al contenedor más cercano con** `position: relative`.
`fixed`	Se posiciona **respecto a la ventana del navegador**. No se mueve al hacer scroll.
`sticky`	Se comporta como `relative` hasta que se alcanza un punto de scroll, luego se fija.

Ejemplo:

```
.header {
  position: fixed;
  top: 0;
  width: 100%;
}
```

□ 3. Propiedades complementarias

Estas propiedades se usan junto con `position` para ajustar la ubicación:

- `top`, `right`, `bottom`, `left`: definen desplazamientos desde los bordes.
- `z-index`: controla la **profundidad** (qué elemento está encima).

Ejemplo:

```
.modal {
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
  z-index: 1000;
}
```

🧠 4. Buenas Prácticas

- Usa `relative` en el contenedor padre si vas a posicionar hijos con `absolute`.
- Evita abusar de `fixed` en móviles; puede causar problemas de usabilidad.
- Usa `z-index` con cuidado para evitar conflictos visuales.
- Prefiere `flex` o `grid` para layouts modernos y responsivos.

□ . 5. Ejercicios

Ejemplo de Cajas:

```
<style>
    .contenedor {
      display: flex;
      justify-content: space-around;
      align-items: center;
      height: 200px;
      background-color: #f9f9f9;
      border: 2px solid #ccc;
```

```
      padding: 10px;
    }

    .caja {
      width: 100px;
      height: 100px;
      color: white;
      display: flex;
      justify-content: center;
      align-items: center;
      font-weight: bold;
      border-radius: 8px;
      box-shadow: 2px 2px 5px rgba(0,0,0,0.2);
    }

    .caja1 { background-color: #e74c3c; }  /* Rojo */
    .caja2 { background-color: #3498db; }  /* Azul */
    .caja3 { background-color: #2ecc71; }  /* Verde */
  </style>
```

```
<body>
  <div class="contenedor">
    <div class="caja caja1">Rojo</div>
    <div class="caja caja2">Azul</div>
    <div class="caja caja3">Verde</div>
  </div>
</body>
```

6. Flexbox —

1. Activar Flexbox

Para usar Flexbox, se aplica `display: flex` al **contenedor padre**. Los elementos hijos se convierten en "flex items".

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

📐 2. Propiedades del Contenedor (Padre)

Propiedad	Descripción	Ejemplo
`flex-direction`	Dirección principal: `row`, `column`, `row-reverse`, `column-reverse`	`flex-direction: row;`
`flex-wrap`	Permite que los elementos se ajusten en varias líneas	`flex-wrap: wrap;`
`justify-content`	Alineación horizontal en la dirección principal	`justify-content: space-between;`
`align-items`	Alineación vertical en el eje cruzado	`align-items: center;`
`align-content`	Alineación de múltiples líneas (cuando hay `wrap`)	`align-content: space-around;`

☑ Ejemplo completo:

```
.contenedor {
  display: flex;
  flex-direction: row;
  flex-wrap: wrap;
  justify-content: center;
  align-items: center;
}
```

📦 3. Propiedades de los Elementos (Hijos)

Propiedad	Descripción	Ejemplo
`flex-grow`	Cuánto puede crecer el elemento si hay espacio disponible	`flex-grow: 1;`
`flex-shrink`	Cuánto puede reducirse el elemento si hay poco espacio	`flex-shrink: 1;`
`flex-basis`	Tamaño inicial del elemento antes de distribuir el espacio	`flex-basis: 200px;`
`flex`	Atajo para `flex-grow`, `flex-shrink` y `flex-basis`	`flex: 1 1 200px;`
`align-self`	Alineación individual en el eje cruzado (sobrescribe `align-items`)	`align-self: flex-end;`
`order`	Cambia el orden visual del elemento	`order: 2;`

□ 4. Ejemplo

Ejemplo Cajas con formularios:

```
<style>
    body {
      font-family: Arial, sans-serif;
    }

    .contenedor {
      display: flex;
      justify-content: space-around;
      align-items: flex-start;
      padding: 20px;
      background-color: #f4f4f4;
    }

    .caja {
      width: 250px;
      background-color: #ffffff;
      border-radius: 10px;
      box-shadow: 0 4px 8px rgba(0,0,0,0.1);
      padding: 20px;
      display: flex;
      flex-direction: column;
      gap: 10px;
    }

    .caja h3 {
      margin-top: 0;
      color: #333;
    }

    input, textarea, button {
      width: 100%;
      padding: 8px;
      border: 1px solid #ccc;
      border-radius: 5px;
      font-size: 14px;
    }

    button {
      background-color: #3498db;
      color: white;
      border: none;
```

```
      cursor: pointer;
    }

    button:hover {
      background-color: #2980b9;
    }
  </style>
```

```
<div class="contenedor">
    <div class="caja">
      <h3>Contacto</h3>
      <input type="text" placeholder="Nombre">
      <input type="email" placeholder="Correo">
      <textarea placeholder="Mensaje"></textarea>
      <button>Enviar</button>
    </div>

    <div class="caja">
      <h3>Registro</h3>
      <input type="text" placeholder="Usuario">
      <input type="password" placeholder="Contraseña">
      <button>Registrarse</button>
    </div>

    <div class="caja">
      <h3>Suscripción</h3>
      <input type="email" placeholder="Tu correo">
      <button>Suscribirse</button>
    </div>
```

📱 5. Flexbox y Responsividad

Flexbox se adapta muy bien a pantallas pequeñas:

```
@media (max-width: 768px) {
  .contenedor {
    flex-direction: column;
  }
}
```

Esto convierte una fila horizontal en una columna vertical en móviles.

🧠 6. Buenas Prácticas

- Usa `flex: 1` para que los elementos compartan espacio equitativamente.
- Usa `flex-wrap: wrap` para evitar desbordamientos.
- Usa `align-items: stretch` para que los elementos tengan la misma altura.
- Usa `order` para cambiar el orden visual sin alterar el HTML.

🧮 7. Responsive Design

◇ 1. ¿Qué es Responsive Design?

Es una técnica de diseño web que permite que una página **se adapte automáticamente** al tamaño de pantalla del dispositivo del usuario. Esto se logra mediante:

- **Unidades flexibles** (`%`, `vw`, `vh`, `em`, `rem`)
- **Media queries**
- **Layouts fluidos** (Flexbox, Grid)
- **Imágenes adaptativas**
- **Tipografía escalable**

□ 2. Unidades Responsivas

Unidad	Uso	Ejemplo
`%`	Porcentaje del contenedor padre	`width: 80%;`
`vw`	Porcentaje del ancho de la ventana	`width: 50vw;`
`vh`	Porcentaje del alto de la ventana	`height: 100vh;`
`em`	Relativo al tamaño de fuente del padre	`font-size: 1.2em;`
`rem`	Relativo al tamaño de fuente raíz	`font-size: 1.5rem;`

📐 3. Media Queries

Permiten aplicar estilos específicos según el ancho del dispositivo.

☑️ Ejemplo básico:

```
@media (max-width: 768px) {
  .menu {
    flex-direction: column;
  }
}
```

📊 Ejemplo con rangos:

```
@media (min-width: 768px) and (max-width: 1024px) {
  .contenido {
    grid-template-columns: 1fr 1fr;
  }
}
```

□ 4. Layouts Responsivos

◇ Flexbox

```
.contenedor {
  display: flex;
  flex-wrap: wrap;
}

@media (max-width: 600px) {
  .contenedor {
    flex-direction: column;
  }
}
```

◇ Grid

```
.grid {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
}
```

Esto crea columnas que se ajustan automáticamente según el espacio disponible.

🖼️ 5. Imágenes Responsivas

Usa `max-width: 100%` para que las imágenes no se desborden.

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

También puedes usar el elemento `<picture>` para servir diferentes imágenes según el dispositivo:

```
<picture>
  <source media="(max-width: 600px)" srcset="imagen-movil.jpg">
  <img src="imagen-desktop.jpg" alt="Ejemplo">
</picture>
```

🔡 6. Tipografía Escalable

Usa `em` o `rem` para que el texto se adapte al tamaño base del navegador.

```
body {
  font-size: 1rem; /* base */
}

h1 {
  font-size: 2.5rem;
}
```

🧠 7. Buenas Prácticas

- Diseña **mobile-first**: comienza con estilos para móviles y escala hacia pantallas grandes.
- Usa `min-width` en media queries para construir progresivamente.
- Evita fijar tamaños en `px` para elementos clave.
- Prueba tu diseño en múltiples dispositivos y navegadores.
- Usa herramientas como Chrome DevTools para simular pantallas.

□ Ejercicio

```
<style>
        @media (min-width: 768px) {
            .contenedor {
                flex-direction: row;
                justify-content: space-between;
            }

            img {
                max-width: 100%;
                height: auto;
            }

        }
    </style>
```

```html
<body>
    <div class="contenedor">
        <div style="background: coral; padding: 20px;">Elemento 1</div>
        <div style="background: teal; padding: 20px;">Elemento 2</div>
        <div style="background: gold; padding: 20px;">Elemento 3</div>
    </div>
    <div style="background: rgb(196, 185, 181); padding: 20px;">
        <picture>
            <source media="(max-width: 600px)" srcset="imagen/movil.jpeg">
            <img src="imagen/desktop.jpeg">
        </picture>
    </div>
</body>
```

🎬 8. Transiciones y Animaciones

◇ 1. ¿Qué son?

- **Transiciones**: Cambios suaves entre dos estados de una propiedad CSS (por ejemplo, al pasar el mouse).
- **Animaciones**: Secuencias definidas de cambios que pueden repetirse, pausarse o controlarse con mayor precisión.

⚡ Transiciones CSS

□ 2. Propiedades clave

Propiedad	Descripción	Ejemplo
`transition`	Atajo para definir duración, propiedad, tiempo y curva	`transition: all 0.3s ease;`
`transition-property`	Propiedad que se va a animar	`transition-property: background;`
`transition-duration`	Duración del cambio	`transition-duration: 0.5s;`
`transition-timing-function`	Curva de aceleración (`ease`, `linear`, etc.)	`transition-timing-function: ease-in-out;`

Propiedad	Descripción	Ejemplo
`transition-delay`	Retraso antes de iniciar la transición	`transition-delay: 0.2s;`

☑ Ejemplo práctico de transición

```html
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Botón Animado</title>
  <style>
    .boton {
      background-color: coral;
      color: white;
      padding: 10px 20px;
      border: none;
      border-radius: 5px;
      cursor: pointer;
      font-size: 16px;
      transition: background-color 0.3s ease;
    }

    .boton:hover {
      background-color: darkred;
    }

    body {
      display: flex;
      justify-content: center;
      align-items: center;
      height: 100vh;
      margin: 0;
      background-color: #f4f4f4;
    }
  </style>
</head>
<body>
  <button class="boton">Haz clic aquí</button>
</body>
</html>
```

🎞 Animaciones CSS

□ 3. Propiedades clave

Propiedad	Descripción	Ejemplo
`@keyframes`	Define los pasos de la animación	`@keyframes mover { ... }`
`animation-name`	Nombre de la animación definida	`animation-name: mover;`
`animation-duration`	Duración total de la animación	`animation-duration: 2s;`
`animation-timing-function`	Curva de aceleración	`animation-timing-function: ease-in-out;`
`animation-delay`	Retraso antes de iniciar	`animation-delay: 1s;`
`animation-iteration-count`	Número de repeticiones (`infinite`, `1`, etc.)	`animation-iteration-count: infinite;`
`animation-direction`	Dirección (`normal`, `reverse`, `alternate`)	`animation-direction: alternate;`

☑ Ejemplo práctico de animación

```html
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Animación con CSS</title>
  <style>
    @keyframes mover {
      0%   { transform: translateX(0); }
      50%  { transform: translateX(100px); }
      100% { transform: translateX(0); }
    }

    .caja {
      width: 100px;
      height: 100px;
      background: teal;
      animation: mover 2s ease-in-out infinite;
    }
```

```
    body {
      display: flex;
      justify-content: center;
      align-items: center;
      height: 100vh;
      margin: 0;
      background: #f0f0f0;
    }
  </style>
</head>
<body>
  <div class="caja"></div>
</body>
</html>
```

🧠 Buenas Prácticas

- Usa transiciones para **interacciones simples** (hover, focus, active).
- Usa animaciones para **efectos más complejos** (entradas, salidas, ciclos).
- No abuses de animaciones infinitas: pueden distraer o afectar el rendimiento.
- Usa `will-change` para optimizar animaciones en propiedades como `transform` y `opacity`.
- Considera la accesibilidad: permite desactivar animaciones si el usuario lo solicita (`prefers-reduced-motion`).