UNIDAD DIDÁCTICA 2: EFECTOS ESPECIALES EN PÁGINAS WEB

Exploraremos diferentes técnicas para enriquecer la experiencia de usuario mediante el uso de imágenes, textos, marcos, ventanas y otros recursos interactivos. Se incluyen explicaciones, ejemplos prácticos y ejercicios para fomentar la asimilación de los conceptos.

---

1. TRABAJAR CON IMÁGENES: IMÁGENES DE SUSTITUCIÓN E IMÁGENES MÚLTIPLES

1.1. IMÁGENES DE SUSTITUCIÓN (ROLLOVER IMAGES)

Explicación

Las imágenes de sustitución son aquellas que cambian cuando el usuario realiza una acción, como pasar el ratón por encima (hover) o hacer clic. Esto se utiliza a menudo para menús interactivos o para resaltar visualmente algún elemento al que se quiere dar importancia.

- **Técnicas**:
    - **HTML + CSS**: Usar :hover en una imagen de fondo (background-image) o pseudo-elementos.
    - **JavaScript**: Cambiar dinámicamente el atributo src de la imagen al detectar un evento (mouseover, mouseout, click, etc.).

Ejemplo Sencillo (JavaScript)

```
<!DOCTYPE html>
<html>
<body>
  <img
    id="imagenRollover"
    src="imagen1.jpg"
    alt="Imagen inicial"
    width="200"
  />

  <script>
    const img = document.getElementById("imagenRollover");
    img.addEventListener("mouseover", () => {
      img.src = "imagen2.jpg"; // Cambia la imagen
    });
    img.addEventListener("mouseout", () => {
```

```
      img.src = "imagen1.jpg"; // Vuelve a la imagen original
    });
  </script>
</body>
</html>
```

1.2. IMÁGENES MÚLTIPLES (GALERÍAS Y SLIDERS)

Explicación
Para mostrar múltiples imágenes de manera dinámica, se suelen emplear **galerías** (grids) o **sliders/carousels** (pases de diapositivas).

- **Galerías**: Muestran varias imágenes simultáneamente, a menudo con posibilidad de agrandarlas (lightbox) al hacer clic.
- **Sliders**: Ciclan imágenes en un carrusel con transiciones (deslizamiento, fundido, etc.).

Ejemplo de Slider (JavaScript básico)

```
<!DOCTYPE html>
<html>
<body>
  <div>
    <img id="slider" src="foto1.jpg" alt="Slider" width="300">
  </div>
  <button onclick="anterior()">Anterior</button>
  <button onclick="siguiente()">Siguiente</button>

  <script>
    const imagenes = ["foto1.jpg", "foto2.jpg", "foto3.jpg"];
    let indice = 0;
    const slider = document.getElementById("slider");

    function mostrarImagen() {
      slider.src = imagenes[indice];
    }

    function siguiente() {
      indice = (indice + 1) % imagenes.length;
      mostrarImagen();
    }

    function anterior() {
      indice = (indice - 1 + imagenes.length) % imagenes.length;
```

```
      mostrarImagen();
    }
  </script>
</body>
</html>
```

Ejercicio Propuesto

1. **Rollover con clic**: Implementar un efecto de sustitución de imagen cuando el usuario haga clic (en lugar de mouseover), y que la imagen regrese al original con un segundo clic.

```
<!DOCTYPE html>
<html>
<head>
    <title>Efecto de Sustitución de Imagen</title>
    <style>
        #imagen {
            width: 300px;
            height: auto;
            border: 2px solid black;
        }
    </style>
</head>
<body>
    <h1>Haz clic en la imagen para cambiarla</h1>
    <img id="imagen" src="imagen1.jpg" alt="Imagen Original">

    <script>
        const imagen = document.getElementById('imagen');
        let estaActiva = false;

        imagen.addEventListener('click', () => {
            if (estaActiva) {
                // Restaurar la imagen original
                imagen.src = 'imagen1.jpg';
                imagen.alt = 'Imagen Original';
            } else {
                // Cambiar a la nueva imagen
                imagen.src = 'imagen2.jpg';
                imagen.alt = 'Imagen de Clic';
            }
            estaActiva = !estaActiva;
```

```
        });
    </script>
</body>
</html>
```

Explicación:

1. **HTML**:
    - Se define una imagen con el id="imagen". La imagen inicial es imagen1.jpg.
2. **CSS**:
    - Se añade un estilo básico a la imagen para definir su tamaño y borde.
3. **JavaScript**:
    - Se selecciona la imagen usando document.getElementById() y se define una variable estaActiva para llevar un registro del estado de la imagen (si está en su estado original o alternativo).
    - Se añade un evento de clic a la imagen. Cada vez que se hace clic en la imagen, se cambia la fuente de la imagen (src) y el texto alternativo (alt) entre dos valores: imagen1.jpg y imagen2.jpg.
    - La variable estaActiva se alterna entre true y false para determinar qué imagen mostrar.
4. **Funcionalidad**:
    - Al hacer clic en la imagen, cambia a la segunda imagen (imagen2.jpg).
    - Al hacer clic nuevamente, regresa a la primera imagen (imagen1.jpg).

2. **Slider con transición**: Añadir un efecto de transición (por ejemplo, un fundido con CSS opacity) cada vez que cambie la imagen en el slider.

```
<!DOCTYPE html>
<html>
<head>
    <title>Slider con Transición</title>
    <style>
        .slider-container {
            position: relative;
            width: 300px;
            height: 200px;
            overflow: hidden;
        }
```

```
        .slider img {
            width: 100%;
            height: auto;
            position: absolute;
            top: 0;
            left: 0;
            opacity: 0;
            transition: opacity 1s ease-in-out;
        }

        .slider img.active {
            opacity: 1;
        }
    </style>
</head>
<body>
    <div class="slider-container">
        <img src="imagen1.jpg" alt="Imagen 1" class="active">
        <img src="imagen2.jpg" alt="Imagen 2">
        <img src="imagen3.jpg" alt="Imagen 3">
    </div>
    <br>
    <button id="prevButton">Anterior</button>
    <button id="nextButton">Siguiente</button>

    <script>
        const images = document.querySelectorAll('.slider img');
        let currentIndex = 0;

        function showImage(index) {
            images.forEach((img, i) => {
                img.classList.remove('active');
                if (i === index) {
                    img.classList.add('active');
                }
            });
        }
```

```
document.getElementById('nextButton').addEventListener('click', ()
=> {
            currentIndex = (currentIndex + 1) % images.length;
            showImage(currentIndex);
        });


document.getElementById('prevButton').addEventListener('click', ()
=> {
            currentIndex = (currentIndex - 1 + images.length) %
images.length;
            showImage(currentIndex);
        });
    </script>
</body>
</html>
```

Explicación:

1. **HTML**:
    - El contenedor .slider-container alberga las imágenes del slider.
    - Cada imagen está posicionada de manera absoluta una sobre la otra.
    - Se añaden botones "Anterior" y "Siguiente" para navegar entre las imágenes.
2. **CSS**:
    - .slider-container: Define el tamaño del contenedor y utiliza overflow: hidden para ocultar las imágenes que están fuera del área visible.
    - .slider img: Establece todas las imágenes como no visibles inicialmente con opacity: 0 y añade una transición de opacity que dura 1 segundo.
    - .slider img.active: La clase active establece opacity: 1 para la imagen actualmente visible.
3. **JavaScript**:
    - Se seleccionan todas las imágenes del slider y se mantiene un índice currentIndex para controlar la imagen actual.
    - La función showImage(index) recorre todas las imágenes, quitando la clase active y añadiéndola solo a la imagen en el índice especificado.
    - Los botones "Anterior" y "Siguiente" cambian currentIndex y llaman a showImage() para mostrar la siguiente o anterior imagen, respectivamente.
4. **Funcionalidad**:

- Al hacer clic en los botones, la imagen actual se desvanece y la nueva imagen se desvanece hacia adentro debido a la transición de opacity.

Asegúrate de tener las imágenes imagen1.jpg, imagen2.jpg, y imagen3.jpg en el mismo directorio que tu archivo HTML para que el ejemplo funcione correctamente. Puedes ajustar la duración de la transición y otros estilos según lo necesites.

---

2. TRABAJAR CON TEXTOS: EFECTOS ESTÉTICOS Y DE MOVIMIENTO

2.1. EFECTOS ESTÉTICOS

Explicación
Mediante **CSS** y **JavaScript**, podemos dar estilos y animaciones al texto que mejoren su legibilidad o que sirvan como elemento de atracción visual:

- **Transiciones**: Cambios suaves de color, tamaño, etc.
- **Sombras**: Uso de text-shadow en CSS.
- **Degradados de texto**: Con background-clip: text; (efectos modernos).
- **Animaciones con @keyframes**: Incluir movimientos o cambios de color progresivos.

Ejemplo de Texto con Degradado (CSS)

```
.titulo-degradado {
  font-size: 2em;
  background: linear-gradient(to right, #ff7e5f, #feb47b);
  -webkit-background-clip: text;
  color: transparent;
}
```

2.2. Efectos de movimiento

Explicación
Podemos lograr que el texto se desplace, aparezca o desaparezca de manera dinámica:

- **Scrolling o "marquee"**: Aunque la etiqueta <marquee> está obsoleta, podemos usar CSS + JavaScript para simularlo.
- **GSAP o librerías similares**: Permiten animaciones más complejas (desplazamientos, giros, rebotes, etc.).

Ejemplo de Animación sencilla con keyframes

```
@keyframes moverTexto {
  0% {
    transform: translateX(100%);
  }
  100% {
```

```
    transform: translateX(-100%);
  }
}
.texto-animado {
  display: inline-block;
  animation: moverTexto 5s linear infinite;
}
<div class="texto-animado">¡Este texto se mueve de derecha a
izquierda!</div>
```

Ejercicio Propuesto

1. **Fondo animado**: Crear un texto con una animación de fondo degradado que cambie de color de manera continua (usando @keyframes).

```
<!DOCTYPE html>
<html>
<head>
    <title>Texto con Fondo Animado</title>
    <style>
        .animated-background {
            font-size: 2em;
            font-family: Arial, sans-serif;
            background: linear-gradient(45deg, #ff6f61, #ffcc5c,
#68c8c2);
            background-size: 200% 200%;
            animation: gradientAnimation 5s ease infinite;
            -webkit-background-clip: text;
            -webkit-text-fill-color: transparent;
        }

        @keyframes gradientAnimation {
            0% { background-position: 0% 50%; }
            50% { background-position: 100% 50%; }
            100% { background-position: 0% 50%; }
        }
    </style>
</head>
<body>
    <h1 class="animated-background">Texto con Fondo Animado</h1>
</body>
</html>
```

Explicación:

1. **CSS**:
    - .animated-background: Esta clase se aplica al texto que deseamos animar.
        - background: Define un degradado lineal con tres colores (#ff6f61, #ffcc5c, #68c8c2). Puedes cambiar estos colores según tus preferencias.
        - background-size: 200% 200%: Esto hace que el degradado sea el doble del tamaño del contenedor, lo que permite que se mueva más allá de sus límites.
        - animation: Aplica la animación gradientAnimation durante 5 segundos en un bucle infinito con una curva de suavizado (ease).
        - -webkit-background-clip: text y -webkit-text-fill-color: transparent: Estas propiedades aseguran que el fondo se aplique solo al texto y que el texto en sí sea transparente, permitiendo que el degradado sea visible.
2. **@keyframes**:
    - gradientAnimation: Define la animación del degradado.
        - 0% y 100%: La posición del degradado comienza y termina en 0% 50%.
        - 50%: La posición del degradado se mueve al 100% 50%, creando un efecto de movimiento continuo.
3. **HTML**:
    - Se aplica la clase animated-background a un elemento <h1> que contiene el texto que queremos animar.

2. **Entrada gradual**: Implementar un efecto donde el texto aparezca con un fundido (opacity) al cargar la página, usando transition o animation.

```
<!DOCTYPE html>
<html>
<head>
    <title>Texto con Entrada Gradual</title>
    <style>
        .fade-in {
            opacity: 0;
            animation: fadeInAnimation 2s ease-in-out forwards;
        }

        @keyframes fadeInAnimation {
            from { opacity: 0; }
            to { opacity: 1; }
```

```
        }
    </style>
</head>
<body>
    <h1 class="fade-in">Texto con Entrada Gradual</h1>
</body>
</html>
```

Explicación:

1. **CSS**:
    - .fade-in: Esta clase aplica la animación de fundido al texto.
        - opacity: 0: Establece la opacidad inicial a 0, haciendo que el texto sea inicialmente invisible.
        - animation: Define una animación llamada fadeInAnimation que dura 2 segundos, utiliza una curva de suavizado (ease-in-out) y se detiene en el último fotograma de la animación (forwards).
2. **@keyframes**:
    - fadeInAnimation: Define la animación de fundido.
        - from { opacity: 0; }: Al inicio de la animación, la opacidad es 0 (completamente transparente).
        - to { opacity: 1; }: Al final de la animación, la opacidad es 1 (completamente visible).
3. **HTML**:
    - La clase .fade-in se aplica a un elemento <h1> que contiene el texto que deseamos que aparezca gradualmente al cargar la página.

Este código crea un efecto de entrada gradual en el texto, haciendo que aparezca suavemente al cargar la página. Puedes ajustar la duración de la animación y otros parámetros según tus necesidades.

---

3. TRABAJAR CON MARCOS

Explicación

Históricamente, se usaban los **frames** (<frameset>) en HTML para dividir la ventana del navegador en varias secciones independientes. Actualmente, esta técnica está **obsoleta** y se desaconseja en favor de:

- **Layouts con HTML + CSS**: Usar flexbox, grid, etc.
- **<iframe>**: Para incrustar contenido externo (videos, documentos, otras páginas).

Aunque no se recomienda el uso de frames tradicionales, sí es útil conocer **iframe**s para insertar contenido externo de forma segura y aislada.

Ejemplo de iframe

```
<!DOCTYPE html>
<html>
<body>
```

```
  <iframe
    width="400"
    height="300"
    src="https://www.youtube.com/embed/VIDEO_ID"
    frameborder="0"
    allowfullscreen
  ></iframe>
</body>
</html>
```

Este ejemplo incrusta un video de YouTube dentro de la página.

Ejercicio Propuesto

1. **Insertar contenido externo**: Crear una página con un iframe que muestre el sitio web oficial de una organización (por ejemplo, la W3C). Asegurarse de indicar un tamaño fijo o responsivo.
2. **Comparar soluciones**: Hacer un breve ejercicio comparando el uso de marcos antiguos (si se dispone de material legacy) con un layout moderno usando HTML/CSS.

---

4. TRABAJAR CON VENTANAS

Explicación

En JavaScript, podemos trabajar con **nuevas ventanas o pestañas** del navegador usando métodos como window.open(). También podemos crear **ventanas modales** (pop-ups) simuladas con HTML/CSS/JS dentro de la misma página.

- **Pop-ups**: Alertas, confirmaciones y prompts nativos (alert(), confirm(), prompt()). Muy básicos y bloqueantes.
- **Ventanas nuevas**: window.open("url", "nombreVentana", "caracteristicas").
- **Modales**: Efectos superpuestos que muestran una ventana flotante dentro de la misma página, más usado que abrir una pestaña aparte.

Ejemplo de window.open()

```
function abrirVentana() {
  window.open("https://www.google.com", "ventanaNueva",
"width=600,height=400");
}
<button onclick="abrirVentana()">Abrir Google</button>
Ejemplo de ventana modal (con CSS)
<style>
  .modal {
    display: none;
```

```
    position: fixed;
    top: 0; left: 0;
    width: 100%; height: 100%;
    background-color: rgba(0,0,0,0.5);
  }
  .modal-contenido {
    background: #fff;
    margin: 10% auto;
    padding: 20px;
    width: 300px;
  }
</style>

<div id="miModal" class="modal">
  <div class="modal-contenido">
    <p>Esto es una ventana modal</p>
    <button onclick="cerrarModal()">Cerrar</button>
  </div>
</div>

<button onclick="abrirModal()">Abrir Modal</button>

<script>
  function abrirModal() {
    document.getElementById("miModal").style.display = "block";
  }
  function cerrarModal() {
    document.getElementById("miModal").style.display = "none";
  }
</script>
```

Ejercicio Propuesto

1. **Pop-up controlado**: Crear un script que abra una ventana con dimensiones específicas y la posicione en la esquina superior izquierda de la pantalla.

```
<!DOCTYPE html>
<html>
<head>
    <title>Ventana Emergente Controlada</title>
    <style>
        #abrirPopup {
```

```
            display: inline-block;
            padding: 10px 20px;
            background-color: #007BFF;
            color: white;
            text-decoration: none;
            border-radius: 5px;
        }

        #abrirPopup:hover {
            background-color: #0056b3;
        }
    </style>
</head>
<body>
    <a href="#" id="abrirPopup">Abrir Pop-up</a>

    <script>

document.getElementById('abrirPopup').addEventListener('click',
function(event) {
            event.preventDefault();

            // Dimensiones de la ventana emergente
            const width = 400;
            const height = 300;

            // Posición en la esquina superior izquierda
            const left = 0;
            const top = 0;

            // Opciones de la ventana emergente
            const options =
`width=${width},height=${height},left=${left},top=${top},resizable=
yes,scrollbars=yes`;

            // Abrir la ventana emergente
            window.open('', '_blank', options);
        });
    </script>
</body>
</html>
```

Explicación:

1. **HTML**:
    - Se crea un enlace <a> con el texto "Abrir Pop-up". Este enlace disparará la apertura de la ventana emergente cuando se haga clic en él.
2. **CSS**:
    - Se aplican estilos al enlace para hacerlo visualmente atractivo, con un fondo azul y texto blanco.
3. **JavaScript**:
    - Se añade un evento click al enlace que previene el comportamiento por defecto del enlace (event.preventDefault()).
    - Se definen las dimensiones de la ventana emergente (width y height) y la posición en la esquina superior izquierda (left y top).
    - Las opciones de la ventana emergente (options) se configuran como una cadena que incluye las dimensiones, la posición y otros parámetros (resizable y scrollbars).
    - window.open() se utiliza para abrir una nueva ventana con las opciones especificadas. El primer argumento es una URL vacía (''), lo que crea una ventana en blanco. El segundo argumento ('_blank') indica que la ventana debe abrirse en una nueva pestaña o ventana.

2. **Modal con animación**: Añadir una ligera transición fade-in/fade-out al mostrar y ocultar la ventana modal, usando CSS opacity y transiciones de 0.3s o 0.5s.

```
<!DOCTYPE html>
<html>
<head>
    <title>Modal con Animación</title>
    <style>
        /* Estilo del botón */
        #abrirModal {
            display: inline-block;
            padding: 10px 20px;
            background-color: #007BFF;
            color: white;
            text-decoration: none;
            border-radius: 5px;
        }

        #abrirModal:hover {
            background-color: #0056b3;
```

```
}

/* Estilo del modal */
.modal {
    display: none;
    position: fixed;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
    width: 300px;
    padding: 20px;
    background-color: white;
    border: 2px solid #ccc;
    box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
    opacity: 0;
    transition: opacity 0.5s ease-in-out;
    z-index: 1000;
}

.modal.active {
    display: block;
    opacity: 1;
}

/* Overlay */
.modal-overlay {
    display: none;
    position: fixed;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background: rgba(0, 0, 0, 0.5);
    opacity: 0;
    transition: opacity 0.5s ease-in-out;
    z-index: 999;
}

.modal-overlay.active {
    display: block;
```

```
            opacity: 1;
        }

        /* Botón de cerrar */
        .close-button {
            position: absolute;
            top: 10px;
            right: 10px;
            background: none;
            border: none;
            font-size: 1.2em;
            cursor: pointer;
        }
    </style>
</head>
<body>
    <a href="#" id="abrirModal">Abrir Modal</a>

    <div class="modal-overlay" id="modalOverlay"></div>
    <div class="modal" id="modal">
        <button class="close-button"
id="cerrarModal">&times;</button>
        <h2>Título del Modal</h2>
        <p>Contenido del modal.</p>
    </div>

    <script>
        const modal = document.getElementById('modal');
        const modalOverlay =
document.getElementById('modalOverlay');
        const abrirModalBtn =
document.getElementById('abrirModal');
        const cerrarModalBtn =
document.getElementById('cerrarModal');

        abrirModalBtn.addEventListener('click', function(event) {
            event.preventDefault();
            modal.classList.add('active');
            modalOverlay.classList.add('active');
        });
```

```
        cerrarModalBtn.addEventListener('click', function() {
            modal.classList.remove('active');
            modalOverlay.classList.remove('active');
        });

        modalOverlay.addEventListener('click', function() {
            modal.classList.remove('active');
            modalOverlay.classList.remove('active');
        });
    </script>
</body>
</html>
```

Explicación:

1. **HTML**:
    - Un botón con el texto "Abrir Modal" que, al hacer clic, muestra el modal.
    - Un contenedor div para el modal que contiene un botón de cerrar y contenido de ejemplo.
    - Un div de overlay (modal-overlay) que oscurece el fondo cuando el modal está activo.
2. **CSS**:
    - .modal: Define el estilo del modal, incluyendo su posición fija en el centro de la pantalla, tamaño, y transición de opacity.
    - .modal.active: Hace que el modal sea visible y cambia la opacidad a 1 cuando se activa.
    - .modal-overlay: Define el estilo del overlay con una transición de opacidad para el efecto de fundido.
    - .modal-overlay.active: Hace que el overlay sea visible y opaco cuando el modal está activo.
    - .close-button: Estilo del botón de cerrar dentro del modal.
3. **JavaScript**:
    - Añade eventos click a los botones para abrir y cerrar el modal.
    - Añade un evento click al overlay para cerrar el modal si se hace clic fuera del contenido del modal.
    - Usa classList.add('active') y classList.remove('active') para controlar la visibilidad y la animación del modal y el overlay.

---

5. OTROS EFECTOS

La creatividad no tiene límites en la web. Algunas posibilidades adicionales:

1. **Parallax Scrolling**: Fondos que se mueven a velocidad distinta que el contenido, creando sensación de profundidad.

2. **Canvas / SVG**: Gráficos y animaciones con API Canvas o SVG (dibujos vectoriales), permitiendo efectos muy personalizados.

3. **Efectos 3D y WebGL**: Con librerías como Three.js, se pueden crear experiencias 3D interactivas dentro del navegador.

4. **Filtrado y transformaciones CSS**: Aplicar filtros (blur, grayscale, sepia) y transformaciones 2D/3D (rotate, scale, translate) con animaciones.

Ejemplo Canvas Básico

```
<canvas id="miCanvas" width="300" height="200"></canvas>
<script>
  const canvas = document.getElementById("miCanvas");
  const ctx = canvas.getContext("2d");
  // Dibuja un rectángulo rojo
  ctx.fillStyle = "red";
  ctx.fillRect(10, 10, 100, 50);
</script>
```

Ejercicio Propuesto

1. **Efecto parallax**: Crear una sección con imagen de fondo y texto encima, de modo que al hacer scroll, la imagen se mueva a diferente velocidad que el texto.

```
<!DOCTYPE html>
<html>
<head>
    <title>Efecto Parallax</title>
    <style>
        body, html {
            margin: 0;
            padding: 0;
            height: 200%; /* Para permitir el desplazamiento */
            overflow-x: hidden;
        }

        .parallax-section {
            position: relative;
            height: 100vh;
            display: flex;
            align-items: center;
            justify-content: center;
            color: white;
            text-align: center;
            font-size: 2em;
        }

        .parallax-section::after {
```

```
            content: '';
            position: absolute;
            top: 0;
            left: 0;
            width: 100%;
            height: 100%;
            background-image: url('fondo.jpg'); /* Ruta a tu imagen
de fondo */
            background-size: cover;
            background-position: center;
            z-index: -1;
        }

        .parallax-text {
            position: relative;
            z-index: 1;
        }
    </style>
</head>
<body>
    <div class="parallax-section">
        <div class="parallax-text">
            Texto con Efecto Parallax
        </div>
    </div>

    <script>
        const parallaxSection = document.querySelector('.parallax-
section');

        window.addEventListener('scroll', function() {
            const scrollPosition = window.pageYOffset;
            parallaxSection.style.backgroundPositionY =
scrollPosition * 0.5 + 'px';
        });
    </script>
</body>
</html>
```

Explicación:

1. **HTML**:

- Se crea un contenedor .parallax-section que alberga el texto y aplica el efecto parallax.
- Dentro del contenedor, el texto está dentro de un div con la clase .parallax-text.

2. **CSS**:
    - body, html: Asegura que la página tenga suficiente altura para permitir el desplazamiento.
    - .parallax-section: Define el contenedor del parallax con una altura del 100% de la ventana (100vh) y centra el texto.
    - .parallax-section::after: Utiliza un pseudo-elemento para aplicar la imagen de fondo. La imagen se posiciona de manera absoluta detrás del texto.
    - .parallax-text: Asegura que el texto esté en primer plano utilizando z-index.
3. **JavaScript**:
    - Añade un evento scroll a la ventana para capturar la posición de desplazamiento.
    - Ajusta la propiedad backgroundPositionY del fondo del contenedor .parallax-section para crear el efecto parallax. La imagen de fondo se mueve a la mitad de la velocidad de desplazamiento (scrollPosition * 0.5).

Nota:

- Asegúrate de tener una imagen llamada fondo.jpg en el mismo directorio que tu archivo HTML o ajusta la ruta según sea necesario.
- Puedes ajustar el factor de velocidad (0.5) para cambiar cuánto se mueve la imagen en relación con el desplazamiento. Valores más bajos crearán un efecto más pronunciado.

2. **Primeros pasos en Canvas**: Dibujar un círculo y un texto sobre un <canvas>, e intentar animarlos moviéndose lentamente por la pantalla (usar requestAnimationFrame o un setInterval).

```
<!DOCTYPE html>
<html>
<head>
    <title>Animación en Canvas</title>
    <style>
        canvas {
            border: 1px solid black;
        }
    </style>
</head>
<body>
    <canvas id="myCanvas" width="800" height="600"></canvas>
```

```
<script>
    const canvas = document.getElementById('myCanvas');
    const ctx = canvas.getContext('2d');

    // Posiciones iniciales
    let circleX = 50;
    let circleY = 50;
    let textX = 200;
    let textY = 100;

    // Velocidades de movimiento
    const circleSpeedX = 1;
    const circleSpeedY = 1.5;
    const textSpeedX = 0.8;
    const textSpeedY = 1.2;

    function draw() {
        // Limpiar el canvas
        ctx.clearRect(0, 0, canvas.width, canvas.height);

        // Dibujar el círculo
        ctx.beginPath();
        ctx.arc(circleX, circleY, 30, 0, Math.PI * 2);
        ctx.fillStyle = '#0095DD';
        ctx.fill();
        ctx.closePath();

        // Dibujar el texto
        ctx.font = '30px Arial';
        ctx.fillStyle = '#333';
        ctx.fillText('Canvas Texto', textX, textY);

        // Actualizar posiciones
        circleX += circleSpeedX;
        circleY += circleSpeedY;
        textX += textSpeedX;
        textY += textSpeedY;

        // Volver al inicio si se sale del canvas
```

```
            if (circleX > canvas.width || circleY > canvas.height)
{
                circleX = 0;
                circleY = 0;
            }
            if (textX > canvas.width || textY > canvas.height) {
                textX = 0;
                textY = 0;
            }

            // Solicitar el siguiente fotograma de la animación
            requestAnimationFrame(draw);
        }

        // Iniciar la animación
        draw();
    </script>
</body>
</html>
```

Explicación:

1. **HTML**:
    - Se define un elemento <canvas> con un tamaño de 800x600 píxeles y un borde para ver sus límites.
2. **JavaScript**:
    - Se obtiene el contexto del <canvas> usando getContext('2d'), que permite dibujar en 2D.
    - Se definen las posiciones iniciales y las velocidades de movimiento para el círculo y el texto.
    - La función draw() se encarga de dibujar el círculo y el texto en el <canvas>, actualizando sus posiciones en cada llamada.
    - clearRect() se utiliza para limpiar el <canvas> antes de dibujar los nuevos fotogramas, evitando así el rastro de los elementos.
    - Si los elementos salen del <canvas>, se restablecen sus posiciones al inicio.
    - requestAnimationFrame(draw) se usa para llamar a la función draw() en el próximo ciclo de animación, creando una animación fluida.

Este ejemplo muestra cómo animar un círculo y un texto moviéndose por el <canvas>. Puedes ajustar las velocidades y las posiciones iniciales para obtener diferentes efectos de animación.

CONCLUSIÓN

En esta **Unidad Didáctica 2: Efectos Especiales en Páginas Web**, hemos explorado múltiples técnicas y recursos para hacer las páginas más atractivas e interactivas:

1. **Imágenes de sustitución y múltiples**: rollovers, sliders y galerías.

2. **Textos con efectos estéticos y animaciones**: transiciones, degradados, movimientos.
3. **Marcos e iframes**: aunque los frames antiguos están obsoletos, conocemos la utilidad de los iframes para incrustar contenido externo.
4. **Ventanas**: pop-ups tradicionales o modales modernos para una mejor experiencia.
5. **Otros efectos**: parallax, canvas, transformaciones 2D/3D, entre otros.