Tema 2. Ejercicios con soluciones y ejercicios

Notas

Basado en el temario de CSS (selectores, unidades/colores, modelo de caja, visualización) y extendido con técnicas modernas (variables CSS, `box-sizing`, media queries, transiciones, Flexbox y Grid).

Ejercicio 1 Especificidad y colisión de estilos

Enunciado.
Dado el siguiente HTML y CSS, determina el color final del texto del párrafo y justifica por especificidad:

```
<body>
  <p id="aviso" class="destacado nota">Hola CSS</p>
</body>

<style>
  p             { color: black; }
  .destacado    { color: green; }
  p.nota        { color: orange; }
  #aviso        { color: blue; }
  p#aviso       { color: purple; }
</style>
```

Solución. Especificidad (aprox.): `p`=(0,0,0,1); `.destacado`=(0,0,1,0); `p.nota`=(0,0,1,1); `#aviso`=(0,1,0,0); `p#aviso`=(0,1,0,1). Gana `p#aviso` por mayor especificidad ⇒ color **morado**. (En el temario: reglas de selectores y colisión por especificidad/orden).

Ejercicio 2 Selectores combinados y herencia

Enunciado.
Estiliza sólo los `span` dentro de `a` que estén a su vez dentro de un `p` con clase `nota`. Deben ser *rojos*, subrayados y no afectar a otros `span`.
Solución.

```
p.nota a span {
  color: red;
  text-decoration: underline;
}
```

El combinador descendiente y la clase concreta limitan el alcance. (Temario: combinaciones de selectores y descendencia).

Ejercicio 3 Modelo de caja y `box-sizing`

Enunciado.
Crea dos cajas de 300px de ancho visual idéntico: una usando el box model clásico y otra con `box-sizing: border-box`. Cada una debe mostrar borde de 10px y `padding` de 20px. Explica la diferencia.
Solución.

```
<div class="clasico">Clásico</div>
<div class="borderbox">Border-box</div>

<style>
div { border:10px solid #333; padding:20px; margin:8px 0; background:#f7f7f7; }
.clasico   { width:300px;               /* ancho contenido: +padding+borde => mayor total */ }
.borderbox { width:300px; box-sizing:border-box; /* total fijo a 300px */ }
</style>
```

Con el modelo clásico, el ancho total = `width + padding + border`. Con `border-box`, el total se mantiene en 300px. (Temario: modelo de caja, márgenes, padding, borde).

Ejercicio 4 Unidades y accesibilidad tipográfica

Enunciado.
Define tipografías escalables: tamaño base en `16px` en `html`, parrafos a `1rem`, titulares H1 a `clamp(1.8rem, 4vw, 3rem)`. Justifica el uso de `rem` y `vw`.
Solución.

```
html { font-size:16px; }
p    { font-size:1rem; line-height:1.6; }
h1   { font-size:clamp(1.8rem, 4vw, 3rem); }
```

`rem` hereda de raíz (consistente con preferencias del usuario); `vw` permite que el título escale con el ancho de la ventana. (Temario: unidades absolutas/relativas y porcentajes).

Ejercicio 5 Paleta y contraste

Enunciado.
Define una paleta con variables CSS y aplica contraste adecuado para texto sobre fondo. Usa dos temas (claro/oscuro) con una clase en `body`.
Solución.

```
:root{
  --bg:#ffffff; --fg:#222222; --prim:#2D6CDF;
}
body.dark{
  --bg:#0f1115; --fg:#e8eaed; --prim:#8ab4f8;
}
body{ background:var(--bg); color:var(--fg); }
a{ color:var(--prim); text-decoration:underline; }
```

Variables facilitan mantener coherencia cromática y alternar temas. (Temario: colores y paletas; ampliado con variables CSS).

Ejercicio 6 `display` vs `visibility` y flujo

Enunciado.
Oculta el tercer elemento de una cuadrícula de 9 bloques *sin* recolocar el resto; luego ocúltalo recolocando el flujo.
Solución.

```
.grid{ display:grid; grid-template-columns:repeat(3,1fr); gap:12px; }
.item{ background:#ddd; padding:12px; }
.item.hidden-space{ visibility:hidden; } /* ocupa sitio */
.item.hidden-flow { display:none; }      /* no ocupa sitio */
```

`visibility:hidden` reserva espacio; `display:none` retira del flujo. (Temario: diferencias display vs visibility).

Ejercicio 7 Responsive con media queries

Enunciado. Haz que una barra de navegación horizontal pase a vertical bajo 640px. **Solución.**

```
<nav class="nav">
  <a>Inicio</a><a>Blog</a><a>Contacto</a>
</nav>

<style>
.nav{ display:flex; gap:12px; }
@media (max-width:640px){
  .nav{ flex-direction:column; }
}
</style>
```

Combina Flexbox + media query mínima. (Extiende visualización/unidades del temario).

Ejercicio 8 Transición y estado :hover/:focus-visible

Enunciado.
Crea botones con transición suave de color y elevación (`transform`) al interactuar con ratón/teclado.
Solución.

```
.button{
  background:#2D6CDF; color:#fff; padding:.75rem 1rem;
  border-radius:.5rem; display:inline-block;
  transition: background .25s ease, transform .2s ease, box-shadow .2s
     ease;
}
.button:hover,
.button:focus-visible{
  background:#1f4fa3; transform: translateY(-2px);
  box-shadow:0 6px 18px rgba(0,0,0,.15);
}
```

Transiciones mejoran UX; `:focus-visible` mantiene accesibilidad. (Amplía temario con pseudo-clases y transiciones).

Ejercicio 9 Flexbox: alineación de tarjetas

Enunciado.
Tres tarjetas en fila, con misma altura, botón pegado abajo, y separación uniforme.
Solución.

```
.container{ display:flex; gap:16px; align-items:stretch; }
.card{ flex:1; display:flex; flex-direction:column; border:1px solid #ddd;
    padding:16px; }
.card .content{ flex:1; }
.card .actions{ margin-top:auto; }
```

`align-items:stretch` iguala alturas; `margin-top:auto` empuja las acciones abajo. (Extensión moderna).

Ejercicio 10 Grid: layout holy-grail simple

Enunciado.
Construye un layout con cabecera, barra lateral, contenido y pie; que la barra se sitúe a la izquierda en escritorio y abajo en móvil.
Solución.

```
.layout{
  display:grid; gap:12px;
  grid-template-areas:
    "header header"
    "aside  main"
    "footer footer";
  grid-template-columns: 240px 1fr;
}
.header{ grid-area: header; background:#eee; }
.aside { grid-area: aside;  background:#f2f2f2; }
.main  { grid-area: main;   background:#fff; }
.footer{ grid-area: footer; background:#eee; }

@media (max-width:700px){
  .layout{
    grid-template-areas:
      "header"
      "main"
      "aside"
      "footer";
    grid-template-columns: 1fr;
  }
}
```

Grid define áreas semánticas y reordena sin cambiar el HTML (responsive limpio).

Referencias al temario (conceptos base): selectores y combinaciones; colisión/especificidad; unidades/colores; modelo de caja; `display`/`visibility`. (Ver páginas señaladas en la introducción.) :contentReference[oaicite:1]index=1

Ejercicios propuestos

1. **Selector :nth-child complejo Enunciado:** Diseña una tabla donde las filas pares tengan fondo gris claro, las filas múltiplo de 3 tengan fondo azul claro y las celdas en columnas pares estén en negrita. **Ayuda:** El pseudo-clase `:nth-child(an+b)` permite seleccionar elementos en función de patrones matemáticos. Por ejemplo, `:nth-child(2n)` selecciona los pares y `:nth-child(3n)` los múltiplos de tres. Ten en cuenta que los selectores pueden combinarse y que en una tabla se suele aplicar sobre `tr` o sobre `td` según lo que quieras estilizar. Revisa cómo interactúan varios selectores de este tipo en conjunto y cómo se resuelven los conflictos de estilos mediante la cascada y la especificidad.

2. **Animaciones con @keyframes Enunciado:** Crea una animación que mueva un cuadrado de izquierda a derecha de manera infinita, acelerando al inicio y frenando al final. **Ayuda:** Las animaciones en CSS se definen con `@keyframes`, donde se describen los estados inicial, intermedio y final de una propiedad. Para lograr un movimiento suave debes usar propiedades transformables, como `transform: translateX()`. El control de la velocidad depende de la función de tiempo (`animation-timing-function`); por ejemplo, `ease-in-out` acelera y luego desacelera. Recuerda que `animation-iteration-count: infinite` repite el ciclo sin detenerse.

3. **Pseudo-elementos avanzados Enunciado:** Diseña un botón que al pasar el ratón muestre una sombra interna animada usando `::after`. **Ayuda:** Los pseudo-elementos `::before` y `::after` son útiles para generar contenido adicional sin alterar el HTML. Para este ejercicio debes crear un pseudo-elemento posicionado absolutamente dentro del botón, con dimensiones ajustadas a su caja. Con `opacity` o `box-shadow` puedes simular la sombra. Añade una transición a esa propiedad y activa el efecto con `:hover`. Piensa en la importancia de usar `position: relative` en el botón para que el pseudo-elemento se posicione correctamente.

4. **Variables CSS y herencia Enunciado:** Crea un esquema de colores donde un contenedor padre defina `-primary` y sus hijos lo usen para fondo o borde. Cambia el valor en un sub-contenedor y observa el efecto. **Ayuda:** Las variables CSS (custom properties) se definen con `-nombre` y se invocan con `var(-nombre)`. Se comportan como valores heredables y permiten que un valor definido en un contenedor esté disponible en todos sus descendientes. Este ejercicio busca que entiendas cómo funciona el concepto de herencia y sobreescritura: si un sub-contenedor redefine la variable, todos los elementos dentro de él usarán el nuevo valor. Así puedes construir sistemas temáticos escalables.

5. **Grid con áreas solapadas Enunciado:** Diseña un grid 2Œ2 donde un elemento abarque las cuatro celdas como fondo (con opacidad), y los otros tres aparezcan encima en distintas posiciones. **Ayuda:** CSS Grid permite asignar un mismo elemento a varias celdas mediante `grid-area` o `grid-column/row`. Si un elemento cubre todo el grid, los demás pueden solaparse ajustando el orden de apilamiento con `z-index`. Este ejercicio te ayudará a entender que Grid no solo organiza en filas y columnas, sino que también puede crear capas de contenido. Usa opacidad para ver la superposición y experimentar con cómo se reparten los espacios.

6. **Flexbox con order y grow Enunciado:** Diseña un layout con 5 cajas que: la tercera ocupe el doble de espacio que las demás, y la última se muestre primera en pantallas pequeñas (`max-width:600px`). **Ayuda:** Flexbox reparte espacio con `flex-grow`, donde el valor asignado determina la proporción de espacio extra que cada elemento ocupa. Para cambiar el orden visual puedes usar `order`, que no altera el HTML pero sí la posición en el flujo flex. Junto a media queries, es posible crear reordenamientos responsivos. Este ejercicio une varias capacidades: control del crecimiento, reordenamiento y adaptabilidad.

7. **Clipping y máscaras Enunciado:** Crea una tarjeta con imagen circular usando `clip-path: circle()`, y añade una transición que cambie a forma hexagonal al pasar el ratón. **Ayuda:** La propiedad `clip-path` permite recortar la región visible de un elemento en formas básicas o personalizadas (con polígonos). Al animar entre formas compatibles (igual número de puntos), se logran transiciones fluidas. Si usas `clip-path: circle()`, puedes pasar a un `polygon()` con seis puntos para simular un hexágono. Recuerda que para animar se debe aplicar una transición a la propiedad y que algunos navegadores aún requieren prefijos.

8. **Tipografía responsiva avanzada Enunciado:** Crea un título que se mantenga entre 2rem y 4rem, pero escale proporcionalmente con el ancho de la pantalla usando `clamp()`. **Ayuda:** La función `clamp(min, preferido, max)` limita un valor entre un mínimo y un máximo, adaptándose al espacio disponible. Es útil en tipografía responsiva porque evita textos demasiado pequeños en pantallas grandes o demasiado grandes en pantallas pequeñas. Usando `vw` como valor preferido,

el tamaño crece de forma fluida, pero gracias a `clamp` nunca sale de los límites establecidos. Esto permite un control más preciso que depender solo de unidades relativas al viewport.

9. **Animación con variables CSS Enunciado:** Define un gradiente con ángulo en una variable `-angle`, y anima ese ángulo para crear un fondo con gradiente rotatorio infinito. **Ayuda:** Las variables CSS pueden usarse dentro de funciones como `linear-gradient(var(-angle), ...)`. Al animar la variable con `@keyframes`, el valor del ángulo cambia gradualmente y produce la sensación de movimiento. La clave es que los navegadores interpolan valores numéricos de las variables, lo que hace posible animar gradientes sin necesidad de imágenes externas ni JavaScript. Es una técnica moderna que aprovecha el poder de las custom properties.

10. **Accesibilidad y media queries avanzadas Enunciado:** Diseña un esquema de colores que se adapte a `prefers-color-scheme: dark`, y otro que reduzca las animaciones si el usuario tiene `prefers-reduced-motion`. **Ayuda:** CSS ofrece media queries que consultan preferencias del sistema operativo o navegador. Con `prefers-color-scheme` puedes definir estilos diferentes para tema claro u oscuro, respetando la configuración del usuario. Con `prefers-reduced-motion` es posible desactivar o simplificar animaciones que podrían resultar molestas. Este ejercicio enfatiza la importancia de la accesibilidad y la adaptación, no solo al tamaño de pantalla, sino también a las necesidades del usuario final.