Contenidos a Trabajar

1. Selectores avanzados
2. Positions
3. Transiciones
4. Transformaciones
5. Animaciones
6. Diseño Responsive

Agencia de Aprendizaje a lo largo de la vida

Selectores Avanzados

Pseudoclases

Las **pseudoclases** se utilizan para hacer referencia a ciertos comportamientos de los elementos HTML. Mediante este tipo de selectores vamos a poder manejar los estilos de nuestros elementos dependiendo esos comportamientos.

Las pseudoclases se definen añadiendo dos puntos al selector antes de la pseudoclase concreta.

Agencia de Aprendizaje a lo largo de la vida

Pseudoclases de enlaces

Son las que tienen en cuenta los comportamientos de los links de nuestra página.

:link	Aplica estilos cuando el enlace no ha sido visitado todavía.
:visited	Aplica estilos cuando el enlace ha sido visitado anteriormente.

Pseudoclases del mouse

Responden a eventos realizados con el mouse sobre los elementos de nuestra página. Muy utilizado para animaciones o cuando queremos resaltar algo en particular.

:hover	Aplica estilos cuando pasamos el mouse sobre un elemento.
:active	Aplica estilos cuando estamos pulsando sobre el elemento.

*estas no tienen el mismo comportamiento al ver el sitio desde un móvil o tablet.

Pseudoclases de formularios

Muy importantes al momento de trabajar con nuestros formularios ya que nos permiten capturar eventos sobre los distintos inputs y gracias a ello dar información adicional al usuario.

:focus	Aplica estilos cuando ingresamos a un input.
:checked	Aplica estilos cuando seleccionamos input checkbox o radio.

Además de estas pseudoclases orientadas a manejar distintos eventos o comportamientos sobre los elementos de nuestras páginas, existen otro tipo que están vinculadas a ampliar el abanico de opciones al momento de seleccionar nuestros elementos para darles estilos.

Estas pseudoclases se basan principalmente en el orden de la etiquetas HTML o su estructura.

Pseudoclases para Elementos Hijos de diferente tipo.

Estas se aplican sobre los elementos que queremos seleccionar y no sobre su contenedor padre. Si bien la selección se hace por etiqueta, clase o id específico, tiene en cuenta TODOS los elementos dentro del contenedor, por ejemplo si queremos seleccionar el primer hijo **<a>** de un contenedor pero resulta que ese primer hijo es un **<p>** entonces esa selección no tendrá efecto.

:first-child	Primer elemento hijo de un contenedor.
:last-child	Último elemento hijo de un contenedor.
:nth-child(n)	N-elemento de un contenedor.
:nth-last-child(n)	N-elemento hijo de un contenedor partiendo desde el final.

:first-child - :last-child

```html
<article>
    <h2>Primer elemento</h2>
    <p>Segundo elemento</p>
    <p>Tercer elemento</p>
</article>
```

```css
h2:first-child {
    color: crimson;
}
```

En este ejemplo con :first-child seleccionamos TODAS etiquetas **<h2>** que sean primer hijo de su contenedor. Por el contrario, si usaramos :last-child con la misma etiqueta, no tendría ningún efecto ya que ese selector no aplica, pero por ejemplo si usamos p:last-child entonces aplicaría los estilos a todos los **<p>** que sean últimos hijos de sus contenedores.

:nth-child(n) - :nth-last-child(n)

```html
<article>
    <h2>Primer elemento</h2>
    <p>Segundo elemento</p>
    <p>Tercer elemento</p>
    <p>Cuarto elemento</p>
    <p>Quinto elemento</p>
    <p>Sexto elemento</p>
</article>
```

```css
p:nth-child(1) {
    /* El primer elemento que
    es párrafo (no aplica) */
    color: crimson;
}

p:nth-child(2) {
    /* El segúndo elemento que
    es párrafo */
    color: crimson;
}
```

```css
p:nth-child(even) {
    /* Los párrafos en
    posición par */
    color: crimson;
}

p:nth-child(3n+1) {
    /* Los párrafos
    de 3 en 3*/
    color: crimson;
}
```

En los casos anteriores tenemos diversos ejemplos de cómo funciona la pseudoclase :nth-child() donde es posible seleccionar un elemento por su posición dentro del contenedor, o varios dependiendo de su condición de si es par o impar o por intervalos definidos.

En el caso de :nth-last-child() funciona igual pero comenzando desde la el último.

Pseudoclases para Elementos Hijos del mismo tipo.

En este caso si bien funcionan igual que las anteriores, no importa el resto de etiquetas dentro del contenedor padre, va a realizar las selecciones tomando en cuenta la cantidad y orden de etiquetas del mismo tipo.

:first-of-type	Primer elemento hijo del mismo tipo.

Agencia de Aprendizaje a lo largo de la vida

:last-of-type	Último elemento hijo del mismo tipo.
:nth-of-type(n)	N-elemento hijo del mismo tipo.
:nth-last-of-type(n)	N-elemento hijo del mismo tipo partiendo desde el final.

:first-of-type - :last-of-type

```html
<article>
    <h2>Primer elemento</h2>
    <p>Segundo elemento</p>
    <p>Tercer elemento</p>
    <p>Cuarto elemento</p>
    <p>Quinto elemento</p>
    <p>Sexto elemento</p>
    <a href="">Séptimo Elemento</a>
</article>
```

```css
p:first-of-type {
    /* Selecciona el primer
    párrafo de ese contenedor */
    color: crimson;
}

p:last-of-type {
    /* Selecciona el último
    párrafo de ese contenedor */
    color: crimson;
}
```

Tal como muestra la imágen, no importa que otras etiquetas hay dentro del contenedor, con :first-of-type o :last-of-type estaremos seleccionado por el tipo de etiqueta, en este caso el segundo elemento y el sexto elemento.

:nth-of-type() - :nth-last-of-type()

```html
<article>
    <h2>Primer elemento</h2>
    <p>Segundo elemento</p>
    <p>Tercer elemento</p>
    <p>Cuarto elemento</p>
    <p>Quinto elemento</p>
    <p>Sexto elemento</p>
    <a href="">Séptimo Elemento</a>
</article>
```

```css
p:nth-of-type(1) {
    /* Selecciona el primer
    párrafo de ese contenedor
    (Segundo Elemento) */
    color: crimson;
}

p:nth-of-type(4) {
    /* Selecciona el cuarto
    párrafo de ese contenedor
    (Quinto Elemento) */
    color: crimson;
}
```

```css
p:nth-of-type(even) {
    /* Los párrafos en
    posición par */
    color: crimson;
}

p:nth-of-type(3n+1) {
    /* Los párrafos
    de 3 en 3*/
    color: crimson;
}
```

Del mismo modo, en este caso las selecciones se van a centrar en el tipo de etiqueta, su cantidad y orden dentro del contenedor, sin importar que otras etiquetas de distinto tipo existan en el mismo.

Pseudoelementos

Los **pseudoelementos** son otra de las características de CSS que permiten hacer referencias a «comportamientos virtuales no tangibles», o lo que es lo mismo, se le puede dar estilo a elementos que no existen realmente en el HTML, y que se pueden generar desde CSS.

Dentro de la categoría de los pseudoelementos CSS, como punto central, se encuentra la propiedad content. Esta propiedad se utiliza en selectores que incluyen los pseudoelementos ::before o ::after, para indicar que vamos a crear contenido antes o después del elemento en cuestión:

content	Propiedad para generar contenido. Sólo usable en ::before o ::after.
::before	Aplica los estilos **antes** del elemento indicado.
::after	Aplica los estilos **después** del elemento indicado.

::before y ::after permiten hacer referencia a «justo antes del elemento» y «justo después del elemento», respectivamente. Se trata de un pseudoelemento que se escribe de forma posterior a la clave o elemento en cuestión, precedido por doble : y que, por ejemplo, ::before se encargará de añadir contenido antes del inicio de la etiqueta de apertura.

```
p::before {
    content: '❤';
}
```

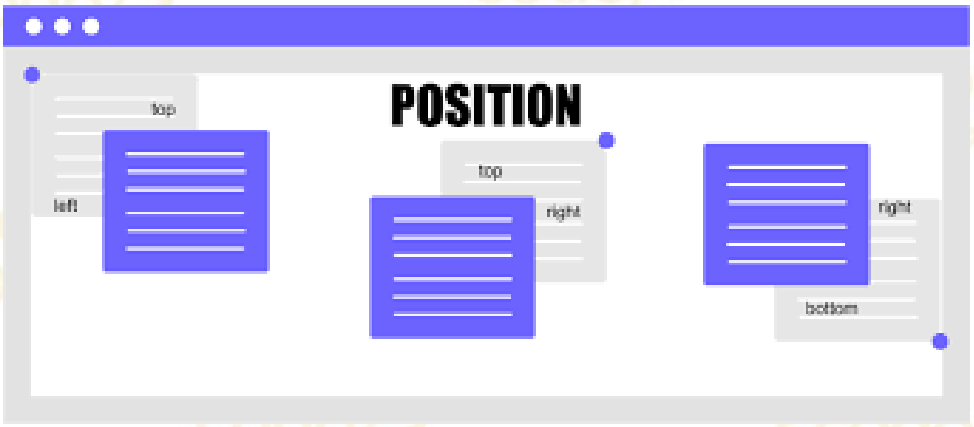
Agencia de Aprendizaje a lo largo de la vida

Además de ::after y ::before existen otros pseudoelementos que puede resultar de mucha utilidad:

::first-letter	Aplica los estilos en la primera letra del texto.
::first-line	Aplica los estilos en la primera línea del texto.
::marker	Aplica estilos a los elementos de cada ítem de una lista.
::backdrop	Aplica estilos al fondo exterior de un elemento (sin que afecte a este).
::placeholder	Aplica estilos a los textos de sugerencia de los campos <input>.
::selection	Aplica estilos al fragmento de texto seleccionado por el usuario.

Agencia de Aprendizaje a lo largo de la vida

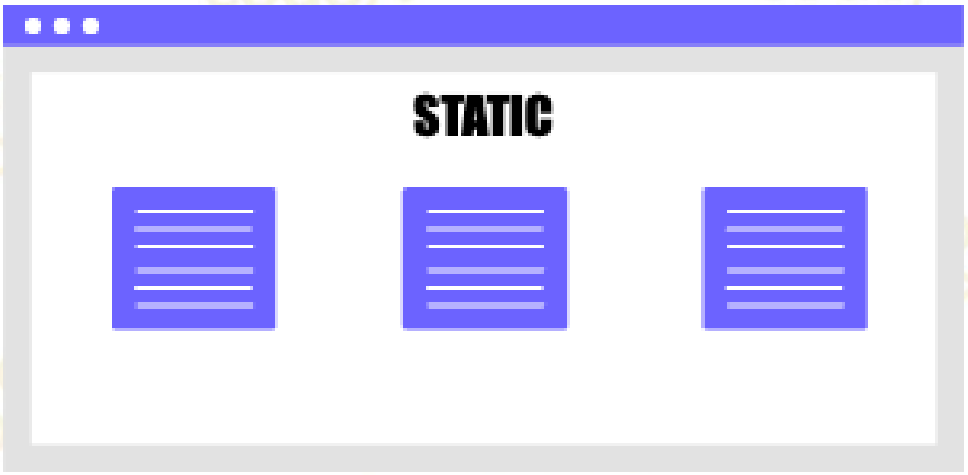
Positions



Hasta ahora, hemos estado trabajando sin saberlo en lo que se denomina posicionamiento estático (static), donde todos los elementos aparecen con un orden natural según donde estén colocados en el HTML. Este es el modo por defecto en que un navegador renderiza una página.

static	Posicionamiento estático. Utiliza el orden natural de los elementos HTML.

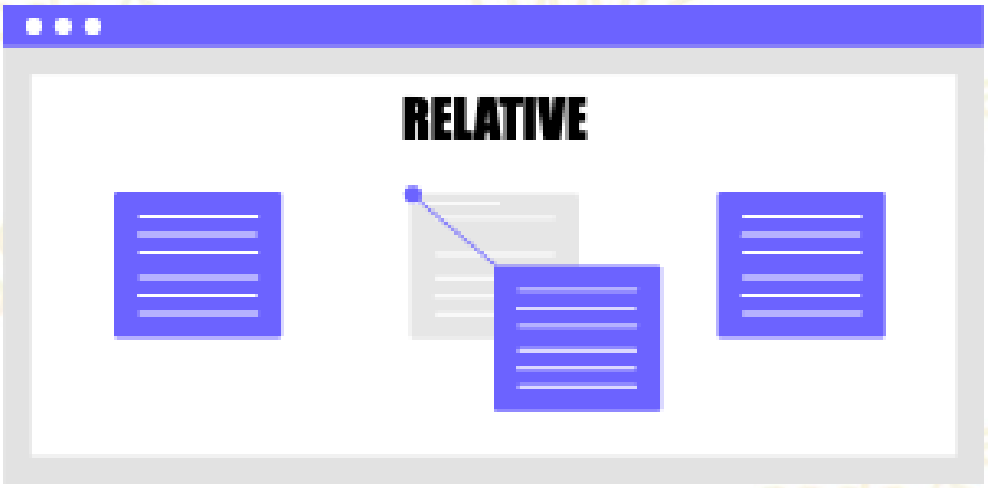
Agencia de Aprendizaje a lo largo de la vida



Sin embargo, existen otros modos alternativos de posicionamiento, que podemos cambiar mediante la propiedad **position**, que nos pueden interesar para modificar la posición en donde aparecen los diferentes elementos y su contenido.

relative	Posicionamiento relativo. Los elementos se mueven ligeramente en base a su posición estática.
absolute	Posicionamiento absoluto. Los elementos se colocan en base al contenedor padre.
fixed	Posicionamiento fijo. Idem al absoluto, pero aunque hagamos scroll no se mueve.
sticky	Posicionamiento «pegado». Similar al relativo, usado para pegar menús a la parte superior.

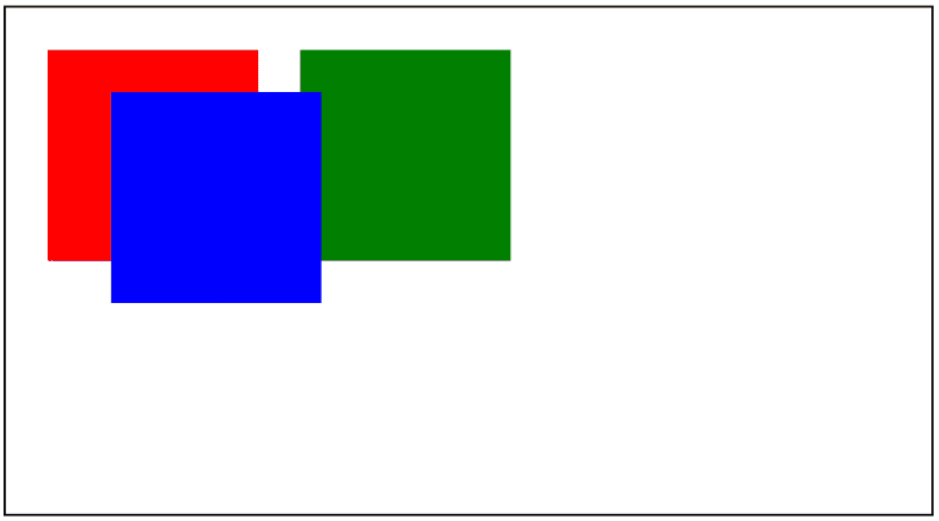
Position Relative



Si utilizamos la palabra clave relative activaremos el modo de posicionamiento relativo, que es el más sencillo de todos. En este modo, los elementos ocupan el espacio que ocuparían de forma estática solo que con las propiedades **top**, **right**, **bottom** y **left** podemos moverlo "visualmente" sin quitar el espacio que le corresponde originalmente.

Position Absolute

La posición de una caja se establece de forma absoluta respecto de su elemento contenedor y el resto de elementos de la página ignoran la nueva posición del elemento.

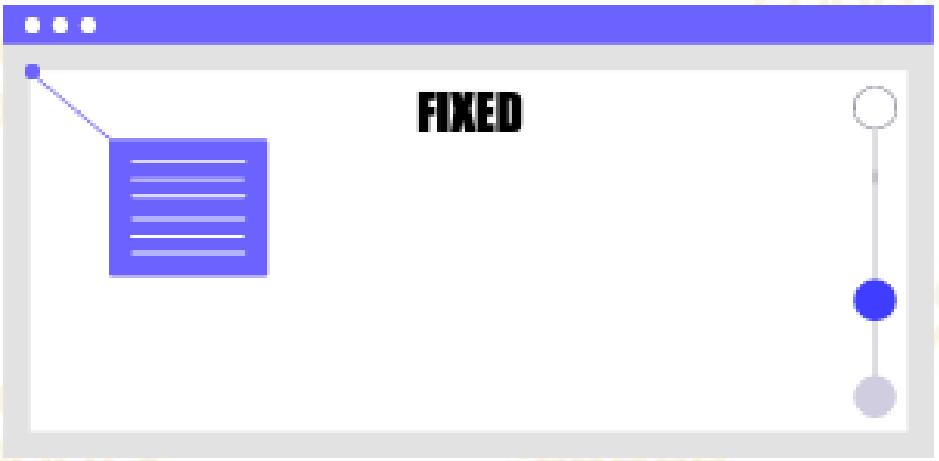


Aunque este es el funcionamiento del posicionamiento absoluto, hay algunos detalles más complejos en su modo de trabajar. Realmente, este tipo de

posicionamiento coloca los elementos utilizando como punto de origen el primer contenedor con posicionamiento diferente a estático.

Por este motivo, si no definimos un position relative a cualquier elemento padre o abuelo del que posee position absolute, entonces este se posicionará de forma absoluta respecto al body de la página. En cambio, si por el ejemplo al contenedor de este elemento se le aplica un position relative, entonces el hijo con position absolute se posicionará de forma absoluta dentro de los límites del elemento padre.

Position Fixed

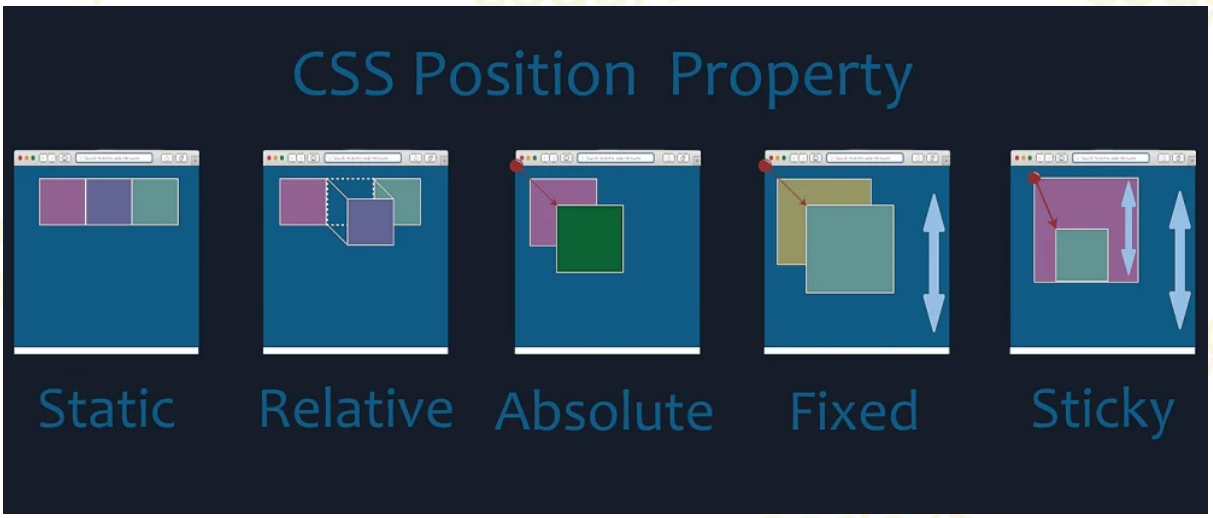


Hace que la caja esté posicionada con respecto a la ventana del navegador, lo que significa que se mantendrá en el mismo lugar incluso al hacer scroll en la página.

Un ejemplo de estos casos, es cuando tenemos un header que al realizar scroll vertical, nos va siguiendo en todo lo alto de la página de forma fija.

Position Sticky

Las cajas con este valor se posicionan según el estado de desplazamiento del usuario. Se "pega" en su lugar al igual que con position fixed, pero solo después de llegar a ese elemento en flujo de la página. Una vez que eso sucede, la caja se pega y nos acompaña con el scroll hasta el final de su contenedor.



Agencia de Aprendizaje a lo largo de la vida

Transiciones

Las transiciones permiten cambiar los valores de las propiedades (de un valor a otro), durante una duración determinada.

Para crear un efecto de transición, hay que especificar dos cosas:

- La propiedad CSS a la que desea agregar un efecto.
- La duración del efecto.

Si la duración no se especifica, la transición no tendrá ningún efecto, ya que el valor predeterminado es 0.

Las transiciones se basan en un principio muy básico: conseguir un efecto suavizado entre un estado inicial y un estado final al realizar una acción.

Propiedades de Transition

transition-property	Propiedades CSS afectadas por la transición.
transition-duration	Tiempo de duración.
transition-timing-function	Ritmo de la transición.
transition-delay	Tiempo de retardo inicial.

Además podemos utilizar el shorthand property **transition** declarando todas estas propiedades en una misma línea en este orden.

Agencia de Aprendizaje a lo largo de la vida

```css
div {
    width: 200px;
    transition: width 2s ease-in 0s;
}

div:hover {
    width: 500px;
}
```

En este ejemplo, vemos como al pasar el mouse por encima de la caja, el elemento cambiará su ancho de 200px a 500px. Usando la propiedad transition con estos valores, lograremos que ese efecto sobre el ancho no sea instantáneo, que tome 2 segundos, siendo suave al comienzo y con un retraso de 0 segundos.

Cabe destacar que al colocar la propiedad transition sobre el selector principal logramos que el efecto se realice tanto al momento de entrada del mouse como de su salida. En caso que lo coloquemos como propiedad del selector con la pseudoclase :hover, entonces solo tendremos el efecto al momento de salida del mouse.

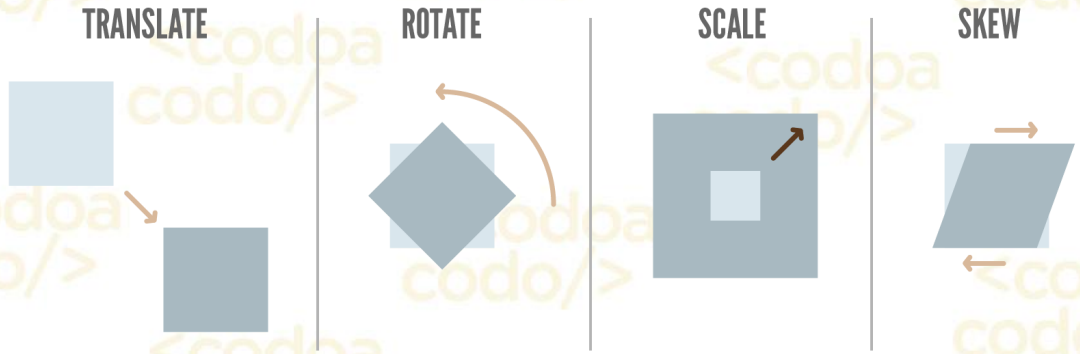
Otro punto importante, es que si bien el efecto trabaja sobre el ancho de la caja podemos afectar diversas propiedades con una transición. Para aplicar el mismo efecto a todas en lugar de colocar **width** tendríamos que poner **all** y en caso que queramos afectar varias propiedades pero con distintos valores de transition, entonces podemos utilizar la misma propiedad transition, con cada declaración separada por comas.

```css
div {
    width: 200px;
    height: 200px;
    transition: all 2s ease-in 0s;
}

div:hover {
    width: 500px;
    height: 200px;
}
```

```css
div {
    width: 200px;
    height: 200px;
    transition: width 2s ease-in 0s,
                heigth 5s ease-out 5s;
}

div:hover {
    width: 500px;
    height: 200px;
}
```

Agencia de Aprendizaje a lo largo de la vida

Transformaciones



Las transformaciones son una de las características de CSS más interesantes y potentes que se introducen en el lenguaje para convertir las hojas de estilo en un sistema capaz de realizar efectos visuales 2D y 3D. Con ellas podemos hacer cosas como mover elementos, rotarlos, aumentarlos o disminuirlos y otras transformaciones relacionadas o combinadas.

Estas transformaciones se pueden efectuar en CSS mediante la propiedad transform que permite recibir una función de transformación determinada, la cuál será aplicada en el elemento HTML en cuestión seleccionado mediante CSS. Dicho elemento HTML se verá transformado visualmente.

Propiedad transform

transform	Aplica una (o varias) transformaciones CSS al elemento.

Agencia de Aprendizaje a lo largo de la vida

Tipos de transformaciones

Si bien existen muchos (ver), vamos a conocer los tipos de transformaciones más comunes:

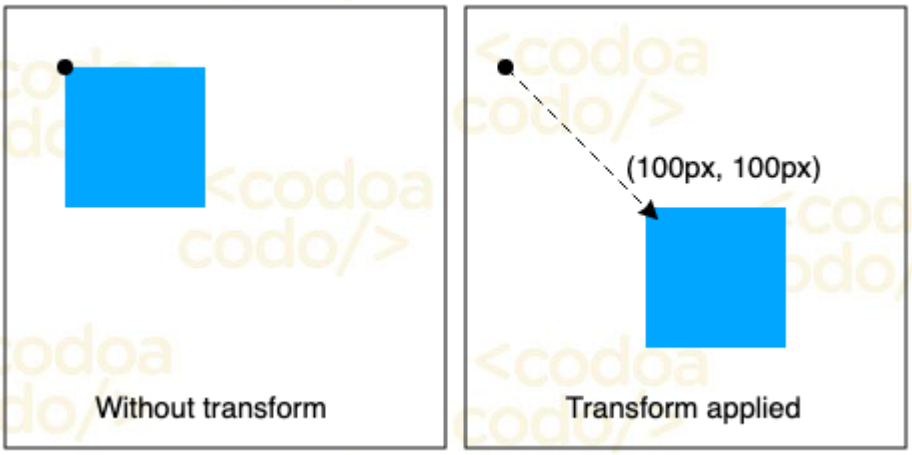
Translate

Nos permite trasladar un objeto de un lugar a otro sin modificar su estructura en el documento (parecido a position relative).

translate(x,y)	Desplaza un elemento en el **eje X** (*izquierda, derecha*) y/o en el **eje Y** (*arriba, abajo*)
translateX(x)	Desplaza un elemento en el **eje X**
translateY(y)	Desplaza un elemento en el **eje Y**

```
div {
    /*
    Valores positivos mueven a la derecha/abajo.
    Valores negativos mueven a la izquierda/arriba.
    */
    transform: translate(10, 30);
}
```

Agencia de Aprendizaje a lo largo de la vida

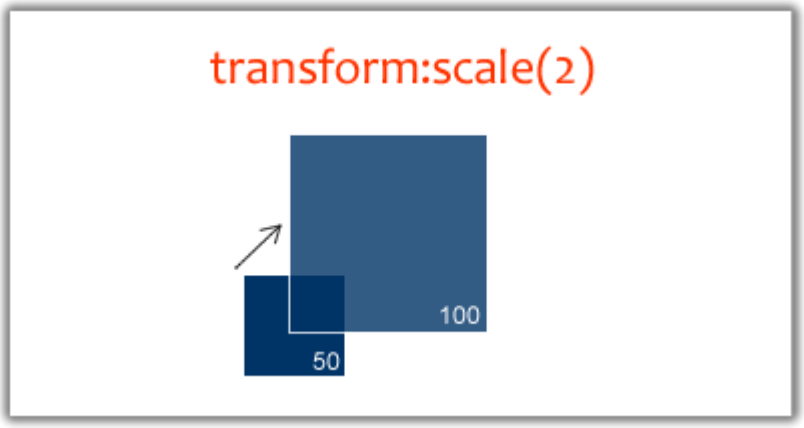


Scale

Permite cambiar la escala del elemento, donde los valores de 0 a 1 achican su tamaño y de 1 en adelante lo agrandan.

scale(x, y)	Escala el elemento una determinada cantidad más grande o más pequeña.
scaleX()	Escala el elemento solo de forma horizontal.
scaleY()	Escala el elemento solo de forma vertical.

```
div {
    /* 2,5 - X,Y */
    /* 2 - X/Y */
    transform: scale(2,2);
}
```



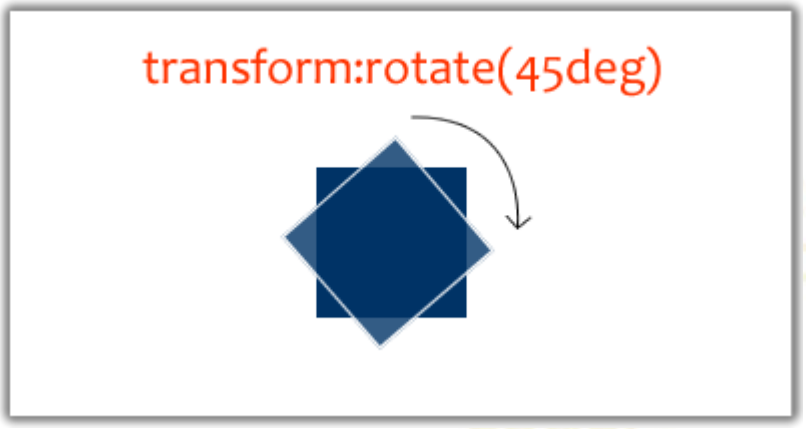
Rotate

Nos da la posibilidad de rotar un elemento sobre su eje o sobre un punto de origen determinado. Acepta valores de 0 a 360 grados y la unidad de medida utilizada es **deg** (degrees).

rotate(deg)	Gira el elemento sobre su **eje X** o sobre su **eje Y**.
rotateX(deg)	Gira el elemento sobre su **eje X**

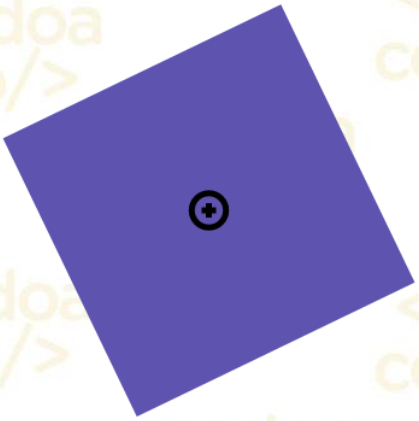
rotateY(deg)	Gira el elemento sobre su **eje Y**

```
div {
    transform: rotate(230deg);
}
```

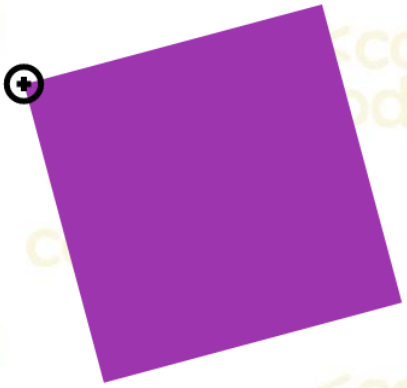


transform-origin	Cambia el punto de origen del elemento en una transformación.

Agencia de Aprendizaje a lo largo de la vida



transform-origin: center center;



transform-origin: left top;

Skew

Este valor nos permite sesgar o torcer un elemento desde sus ejes X o Y.

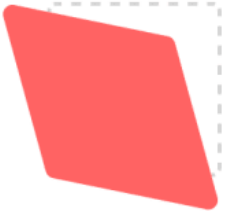
skew(x, y)	Inclina el elemento sobre su **eje X** o sobre su **eje Y**.
skewX(x)	Inclina el elemento sobre su **eje X**
skewY(y)	Inclina el elemento sobre su **eje Y**

Agencia de Aprendizaje a lo largo de la vida

```css
div {
    transform: skew(20deg,10deg);
}
```

transform: skewX(10deg) ;

transform: skewY(10deg) ;



transform: skew(10deg, 10deg) ;

Agencia de Aprendizaje a lo largo de la vida

Animaciones

Ahora que conocemos las transiciones y las transformaciones podemos utilizar estas y otras propiedades CSS para lograr animaciones en nuestros sitios.

Para crear animaciones CSS es necesario realizar 2 pasos:

- Utilizar la propiedad animation (o derivadas) para indicar que elemento HTML vamos a animar.
- Definir mediante la regla @keyframes la animación en cuestión y su comportamiento.

Propiedad animation

Esta propiedad es la que nos va a permitir asignar a un elemento una animación en particular y manejar como será su comportamiento.

La propiedad animation funciona como shorthand property de las siguientes propiedades:

animation-name	Nombre de la animación a aplicar.
animation-duration	Duración de la animación.
animation-timing-function	Ritmo de la animación.
animation-delay	Retardo en iniciar la animación.

animation-iteration-count	Número de veces que se repetirá.
animation-direction	Dirección de la animación.
animation-fill-mode	Como se quedará la animación al terminar.
animation-play-state	Estado de la animación.

Estas se pueden utilizar todas juntas en el shorthand property animation o pueden usarse por separado. Las más importantes son animation-name y animation-duration ya que sin ellas no podemos especificar que animación se desea utilizar ni cuánto va a durar.

@keyframes

Una animación está formada por varios fotogramas, una secuencia de imágenes mostradas una detrás de otra que generan el efecto de movimiento que conocemos de una animación.

Para crear esta secuencia de fotogramas en CSS se utiliza la regla conocida como @keyframes.

```
@keyframes nombre-animation {
    time-selector {
        propiedad : valor ;
        propiedad : valor
    }
}
```

Esta regla no se usa dentro de un selector, sino que se declara como una estructura independiente donde crearemos una secuencia de sucesos aplicando distintas propiedades en momentos de tiempo y luego usaremos el nombre asignado a este bloque de código como valor de la propiedad animation-name que vimos anteriormente.

Cómo **time-selector** podemos acudir a distintos métodos que nos permitirán crear nuestra escala de tiempo o secuencia de momentos.

Uno de ellos son los valores **from** y **to**:

from	valores iniciales de la animación
to	valores con los que deberá finalizar la animación

Veamos un ejemplo:

```css
@keyframes agrandar {
    from {
        width: 100px;
        height: 100px;
        background-color: orange;
    }

    to {
        width: 500px;
        height: 500px;
        background-color: crimson;
    }
}
```

En este caso tenemos una animación llamada ***agrandar*** cuyo estado inicial es un ancho y alto de 100px y un color de fondo orange y su estado final es de ancho y alto de 500px y color de fondo crimson.

Dada esta animación, luego tenemos que utilizarla en un elemento de nuestro sitio, para eso la invocamos en la propiedad animation dentro del selector de ese elemento.

```
div {
    animation: agrandar 5s linear infinite;
}
```

De esta manera logramos animar nuestra caja con la animación **agrandar** cuyo tiempo de duración para pasar de 100px a 500px y de orange a crimson será de 5 segundos, teniendo un avance lineal y que se repetirá de forma infinita.

La diferencia entre este ejemplo y el que hicimos con :hover en transitions es que ese efecto, se disparaba cuando el mouse pasaba por encima de nuestra caja, mientras que en este caso la animación comienza a funcionar desde que carga el sitio.

Otra forma además de from/to para crear nuestras secuencias es indicando los valores en porcentaje donde se debe modificar una propiedad.

```
@keyframes change-color {
    0% {
        background: red;          /* Primer fotograma */
    }
    50% {
        background: yellow;       /* Segundo fotograma */
        width: 400px;
    }
    100% {
        background: green;        /* Último fotograma */
    }
}

.animated {
    background: grey;
    color: #FFF;
    width: 150px;
    height: 150px;
    animation: change-color 2s ease 0s infinite;
}
```

Agencia de Aprendizaje a lo largo de la vida

Diseño Responsive

Cada vez es más frecuente acceder a Internet con diferentes tipos de dispositivos, que a su vez tienen diferentes pantallas y resoluciones, con distintos tamaños y formas, que hacen que se consuman las páginas webs de formas diferentes, apareciendo por el camino también diferentes necesidades, problemas y soluciones.

El **responsive web design** es un enfoque que se centra en el entorno del usuario dentro de un sitio web que dependerá del dispositivo que tenga conectado a internet y de algunas otras características como:

- Conexión de red
- Tamaño de la pantalla
- Tipos de interacción (pantallas táctiles, track pads)

- Resolución gráfica.

Antes que el diseño web responsive fuese popular, muchas compañías manejaban un sitio web completamente separado, normalmente en un subdominio diferente que comenzaba con la letra m (de mobile), seguido del dominio del sitio original.

En estos casos, el trabajo era doble ya que había que diseñar, desarrollar y mantener al menos 2 sitios web diferentes al mismo tiempo.

Por otra parte, en el diseño web responsive, el servidor siempre manda el mismo código HTML a todos los dispositivos y se hace uso de CSS para alterar la renderización de la página en el dispositivo.

Independientemente de las dos estrategias previas, el primer paso en la creación de un sitio web para teléfonos o tabletas, es asegurarse que el navegador conozca la intención. Aquí es donde el meta tag viewport entra en juego.

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

El meta tag viewport le indica al navegador cómo ajustar la página al ancho de cada dispositivo teniendo en cuenta la resolución de la pantalla de cada dispositivo y manteniendo la escala con la que se representan en un entorno de escritorio.

Cuando el elemento meta viewport está ausente, los navegadores móviles mostrarán las páginas con su configuración de escritorio por defecto.

Una vez que ya establecimos la escala que debe mantener nuestro sitio independientemente de la resolución de la pantalla es que podemos comenzar a trabajar sobre el diseño responsivo.

Finalmente, para poder tener diseños que se adapten a los distintos tamaños de pantallas, siempre es recomendable el uso de unidades de medida relativas como em, rem, porcentajes, vh, vw, entre otras.

Estrategias de diseño

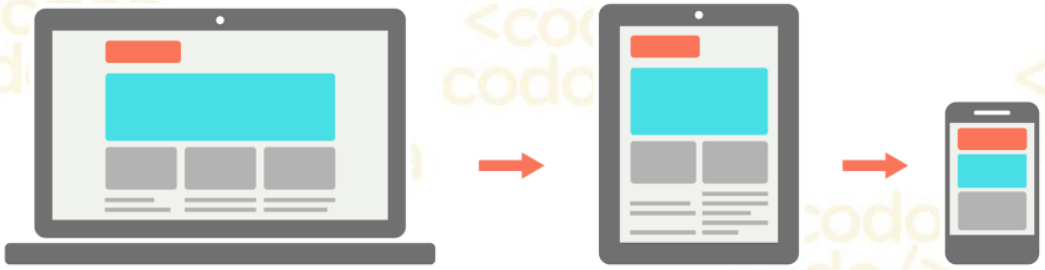
Es aconsejable decidirse por una estrategia de diseño antes de comenzar. Aunque existen otras estrategias, las dos vertientes principales más populares son mobile-first y desktop-first.

Mobile-first

La estrategia Mobile-first es la que utilizan los diseñadores de sitios webs en las que su público objetivo es mayoritariamente usuario de móvil. Ejemplos

como una web para comprar billetes de transporte, la web de un juego o aplicación móvil o una web para pedir cita en un restaurante podrían ser, a priori, una buena elección para utilizar Mobile-first.

Desktop-first



Responsive Web Design

Por otro lado, la estrategia Desktop-first suele interesar más a los diseñadores de sitios webs en las que el público objetivo son usuarios de escritorio. Por ejemplo, una página de una aplicación para PC/Mac o similares, podría ser una buena opción para la estrategia Desktop-first. En ella, hacemos justo lo contrario que en la anterior, lo primero que diseñamos es la versión de escritorio y luego vamos descargando detalles o acomodando información hasta tener la versión para dispositivos móviles.

Media-queries

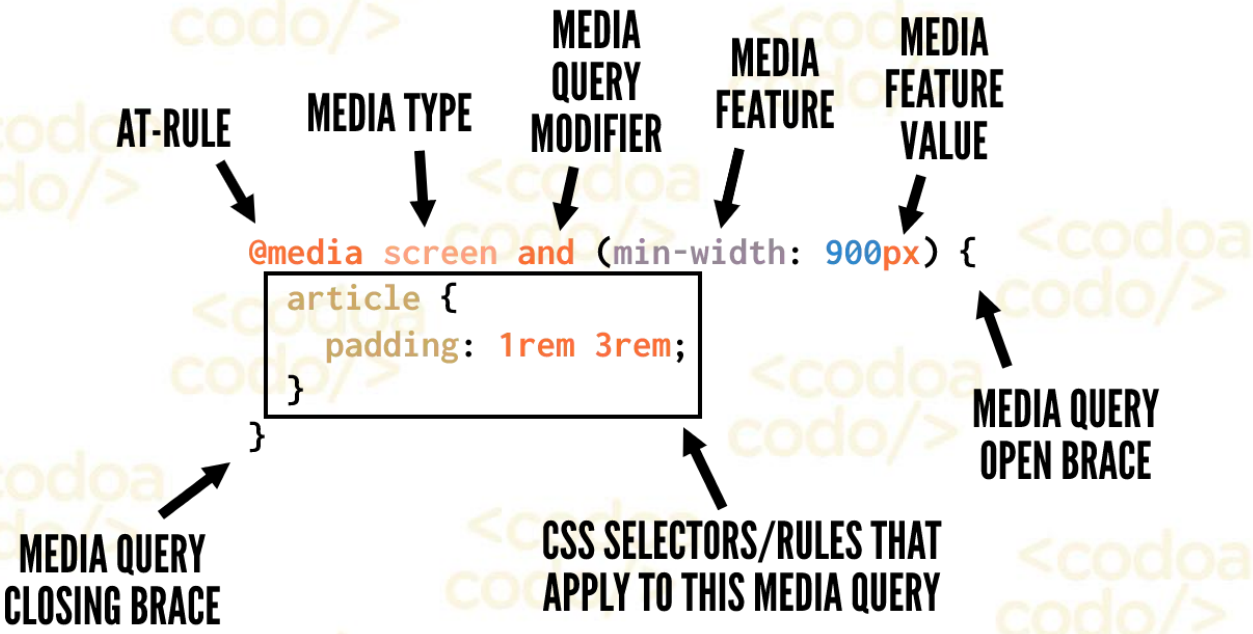
Una vez nos adentramos en el mundo del Responsive Design, nos damos cuenta en que hay situaciones en las que determinados aspectos o

componentes visuales deben aparecer en un tipo de dispositivos, o deben existir ciertas diferencias.

Para ello, utilizaremos un concepto denominado **media queries**, con los que podemos crear ciertas condiciones para que sólo se apliquen a un tipo de diseño concreto.

Ahora que tenemos una idea básica de lo que es un media query, veamos cómo funciona realmente esta particular característica de CSS.

Un media query básico luce así:



Esto significa que los estilos que se escriban dentro de las reglas de media queries anteriores sólo funcionarán o serán efectivos en las propiedades de

ancho especificadas anteriormente, en este caso desde un mínimo de 900px en adelante.

El número de bloques de reglas @media que se utilicen depende del desarrollador web, ya que no es obligatorio utilizar un número concreto. Se pueden utilizar desde un sólo media query, hasta múltiples de ellos a lo largo de todo el documento CSS.

Propiedades max-width y min-width

Hay dos cosas que debemos tener en cuenta al crear media queries para diferentes tamaños de pantalla: las propiedades max-width y min-width.

Cuando se pasa una propiedad max-width a un media query, CSS interpreta una propiedad que debe tomarse hasta ese punto o tamaño máximo de pantalla.

Una vez que la propiedad max-width tiene un valor asignado, todos los estilos dentro de esa media query particular se aplicarán a cualquier dispositivo cuyo tamaño de pantalla abarque desde 0px hasta el ancho máximo especificado.

En cambio para min-width sucede justamente al revés, donde los estilos definidos dentro funcionarán desde la medida asignada en adelante, salvo que otra media querie con estilos que sobreescriben los anteriores tomen lugar.

```css
@media screen and (max-width: 480px) {
    .element {
        width: 100%;
    }
}

@media screen and (max-width: 768px) {
    .element {
        width: 50%;
    }
}

@media screen and (max-width: 1024px) {
    .element {
        width: 25%;
    }
}
```

En el ejemplo anterior, nos encontramos con la declaración de 3 media queries diferentes. En este caso todas poseen estilos aplicados al mismo elemento y la cascada de CSS hace que se vayan aplicando en orden ascendente, justo como fueron declaradas y acompañando el incremento en los tamaños de pantalla.

De esta manera, vemos que de 0px a 480px, el elemento va a contar con un ancho del 100% el cual será sobrescrito por la regla siguiente a partir de los

768px con un ancho de 50% y que podría durar hasta cualquier ancho de la pantalla, salvo porque tenemos una tercera regla que vuelve a pisar los estilos y cambiar el ancho de nuestro elemento, en este caso a partir de los 1024px con un ancho de 100% y donde ahora sí conservará este estilo desde esa medida en adelante.

Cabe destacar, que si bien las distintas reglas de media queries pueden pisar a otras reglas, en los casos donde para un mismo elemento existan propiedades aplicadas en media queries de distintos tamaños y estos no se pisen, entonces se complementarán estos estilos hasta que la regla que los define no siga siendo válida.

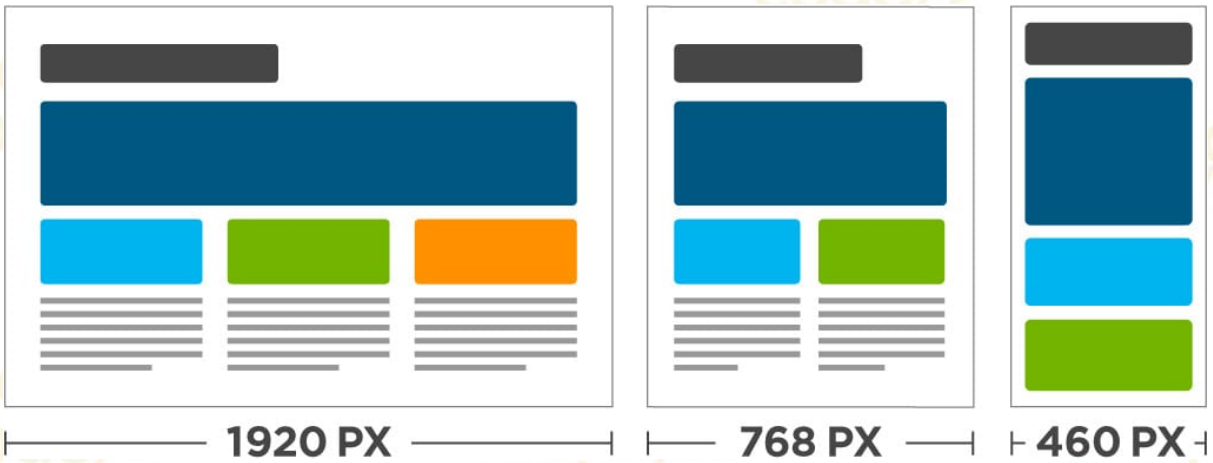
Tipos de Medios

Estos son valores que se incluyen en la declaración de la media querie para especificar a qué tipo de medio se le debe aplicar la regla. Los valores posibles son:

screen	Monitores o pantallas de ordenador. Es el más común.
print	Documentos de medios impresos o pantallas de previsualización de impresión.
speech	Lectores de texto para invidentes (Antes **aural**, el cuál ya está obsoleto).
all	Todos los dispositivos o medios. El que se utiliza **por defecto**.

Breakpoints

Es común cuando trabajamos con media queries, definir ciertos puntos de quiebre o breakpoints por defecto que normalmente obedecen a los tamaños de pantallas más tradicionales de los distintos dispositivos.



Si bien, este tema lo abordaremos más adelante cuando veamos Bootstrap, trabajar con un sistema de tamaños a veces puede resultar de utilidad a la hora de establecer cómo deberían fluir nuestros elementos al agrandar o achicar la pantalla.

Por otra parte, es posible que en algunos casos necesitemos aplicar reglas de @media en tamaños poco convencionales debido a que detectamos algún elemento que no cuadra o se adapta del todo bien en nuestro sitio. Estas situaciones son completamente normales y no hay una regla estricta que nos impida realizarlo.