Cheat sheet

**CSS**

**Vinculación externa con HTML:**

Esta es la más utilizada ya que podemos escribir todos nuestros estilos en un archivo CSS y vincularlos al HTML usando la etiqueta dentro del de nuestro documento.

<link href=”css/styles.css” rel=”stylesheet”>

El atributo **href** + el **valor** es la ruta de la ubicación de mi hoja de estilos.

**Selectores:**

|  |  |  |
| --- | --- | --- |
| **Sintaxis HTML** | **Sintaxis CSS** | **Uso** |
| <h3 **id**=”Saludo”>Hola!</h3> | **#**saludo {  Color:blue;  } | **Selector Id.** Este selector va a atrapar al elemento HTML que tenga asignado el atributo **id** con el valor correspondiente. Para llamarlo desde el CSS usamos el **#** seguido del nombre del ID. |
| <h3 **class**=”noticia destacada”>Una noticia</h3> | Para llamarlo desde el CSS usamos el **.** seguido del nombre de la clase.  **.**noticia { font-size: 22px; } | **Selector de clase.** Este selector va a atrapar al elemento HTML que tenga asignado el atributo **class** con el valor correspondiente. Podemos asignarle la cantidad de clases que queramos a un mismo elemento. Para hacerlo, solo hace falta separarlas con un espacio. |
| <p>Hola mundo como están?</p> | Para llamarlo desde el CSS usamos el nombre de la etiqueta.  P {color:gray;} | **Selector de etiqueta.** Este selector va a atrapar al elemento HTML con el mismo nombre de etiqueta que llamemos desde nuestro CSS. |
| <h2 class=”subtitulo”>Un subtítulo</h2> | Para llamarlos desde el CSS solo hace falta agregar un selector al lado del otro, cada uno con la sintaxis que le corresponda. Recordemos que van sin espacios.  h2.subtitulo {color: yellow;} | **Selectores combinados.** Son la combinación de todos los anteriores, también podemos usar varios selectores del mismo tipo. En este ejemplo combinamos un selector de etiqueta con uno de clase. |
| <ul id=”lista”>  <li>Primer item</li>  </ul> | Para llamarlos desde el CSS escribimos los selectores separados por un espacio (el de la derecha siempre será el que está dentro del de la izquierda).  ul#lista li {text-align:center;} | **Selectores descendentes.** Estos selectores sirven para agregar especificidad. En el ejemplo vamos a atrapar al elemento li que esté dentro del ul con el id lista. |

**Propiedades tipográficas:**

|  |  |
| --- | --- |
| **Sintaxis** | **Uso** |
| P {  Font-family: Arial, sans-serif;  } | **Font-family.** Permite elegir la familia tipográfica que queremos usar. Como valor recibe el nombre de la tipografía que queramos usar. Para que funcione, la tipografía debe existir en la computadora del usuario, o bien debemos usar una webfont. Podemos poner más de una tipografía, separando las adicionales por comas. En caso de que la primera no esté disponible, se cargará la segunda y así sucesivamente. |
| P {  Font-size: 23px;  } | **Font-size.** Permite definir el tamaño tipográfico. Recibe un valor numérico acompañado de la unidad de medida. Las unidades de medida más habituales suelen ser: px, em y rem. |
| P {  Font-style: normal;  } | **Font-style.** Define el estilo de la tipografía. Recibe los valores **italic, normal y oblique.** Para algunos elementos, como <em>, el valor por defecto será italic. |
| P {  Font-weight: 500;  } | **Font-weight.** Define el peso de la tipografía. Recibe los valores **bold, lighter, normal**, entre otros. También puede recibir un valor numérico que se irá incrementando de 100 en 100. Para algunos tags el valor por defecto será bold. |
| P {  Text-align: center;  } | **Text-align.** Permite definir la alineación del texto. Los valores que recibe son **center, left, right, inherit y justify**. El valor por defecto para todos los elementos es left. |
| P {  Text-decoration: underline;  } | **Text-decoration.** Permite elegir un tipo de decoración para el texto. Recibe los valores **line-through, underline, overline y none**. Para algunos elementos, como los enlaces, el valor por defecto será underline. |
| P {  Line-height: 20px;  } | **Line-height.** Permite definir el alto de cada línea de textos. Esto también suele llamarse interlineado. Recibe un valor numérico acompañado de la unidad de medida y, por lo general, está relacionado con el tamaño de la letra o font-size. Tiene que ser 8 píxeles mayor que el Font-size. |

**Colores:**

**Formatos de color:**

* Nombre. Ejemplo: purple. <https://htmlcolorcodes.com/color-names/>
* Hexadecimal. Ejemplo: #f05331.
* RGB. Ejemplo: rgb(255, 100, 50).
* RGBA. Ejemplo: rgba(122, 50, 125, 0.5). El último número representa la opacidad que tendrá el elemento. Va del 0 al 1 y mientras menor el número, mayor la transparencia.

**Atributos de colores:**

|  |  |
| --- | --- |
| **Sintaxis** | **Uso** |
| h2 {**color**: blue;}  h2 {color: #3459ff;}  p {color: rgb(12, 34, 32);} | **Color de texto.** El atributo **color** nos permite asignarle un color al texto de un elemento. Recibe como valor cualquiera de los formatos de color permitidos. |
| P {**background-color**: tomato;}  P {background-color: #3459ff;}  P {background-color: rgb(12, 34, 32);} | **Color de fondo.** El atributo **background-color** nos permite asignarle un color de fondo a un elemento. Recibe como valor cualquiera de los formatos de color permitidos. |
| P {**opacity**: 0.5;} | **La opacidad.** Mediante el atributo **opacity** le otorgamos transparencia a todo el elemento. El valor representa el porcentaje de opacidad que le queremos dar al elemento. Los valores van desde 0 a 1. Eso quiere decir que 0.5 representa una opacidad del 50%. |

**Fondos:** **colores e imágenes**

|  |  |
| --- | --- |
| **Sintaxis** | **Uso** |
| P {**background-color**: tomato;}  P {background-color: #3459ff;}  P {background-color: rgb(12, 34, 32);} | **Background-color.** Nos permite asignarle un color de fondo a un elemento. Recibe como valor cualquiera de los formatos de color permitidos. |
| Body {  Background-image: url (“../img/bici.jpg”)  } | **Background-image.** Nos permite asignarle una imagen de fondo al elemento, definiendo la ruta a través de la URL. |
| Body {  Background-repeat: repeat-x;  } | **Background-repeat.** Nos permite controlar si se va a repetir, y de qué manera, la imagen dispuesta. Recibe los valores **repeat, no repeat, repeat-x, repeat-y, round y space.** |
| Body {  Background-position: right top;  } | **Background-position.** Nos permite mover la imagen dentro del elemento y decidir dónde colocarla. Recibe como valores tamaños en **pixeles y porcentajes**, así como también **right, bottom, left, etcétera**. Podemos asignarle uno o dos valores. El primero para especificar la posición en el eje x y el segundo, la posición en el eje y. |
| Body {  Background-attachment: fixed;  } | **background-attachment**. Nos permite establecer si la imagen de fondo se va a mover junto con la página al hacer scroll o si se va a quedar fija. Recibe como valor **fixed, scroll, inherit e initial**. |
| Body {  Background-size: 130px;  } | **background-size**. Nos permite establecer el tamaño de la imagen de fondo. Recibe como valor **contain, cover, inherit**; así como también tamaños en pixeles y porcentajes, indicando con el primer valor el ancho y con el segundo el alto. |

**Modelos de caja:**

|  |  |
| --- | --- |
| **Sintaxis** | **Uso** |
| Div {  **width**: 120px;  } | **Width**. Si un elemento de bloque no tiene declarada la propiedad width, el ancho será igual al 100% de su padre contenedor. Podemos asignarle un valor a esta propiedad usando cualquier unidad de medida, como porcentajes (%) o píxeles (px). |
| Div {  **height**: 120px;  } | **Height.** Si un elemento no tiene declarado la propiedad height, el alto será igual a la altura que le proporcione su contenido. Sea un elemento de bloque o de línea. Podemos asignarle un valor a esta propiedad usando cualquier unidad de medida, como pixeles (px). Para la altura de los elementos no se recomienda usar porcentaje. |
| Div {  **Padding**: 12px;  }  Al darle un solo valor se aplicará a todos los lados. | **Padding.** Es el espacio de relleno que podemos agregar entre el contenido del elemento y su borde. Podemos asignarle un valor a esta propiedad usando cualquier unidad de medida, como píxeles (px), indicando 1 valor para los 4 lados de la caja. |
| div {  **padding**: 12px 20px;  }  12px de arriba y abajo.  20px en ambos costados. | **Padding.** También podemos hacerlo con 2 valores. El primero va a indicar el padding de arriba y abajo, y el segundo el de la izquierda y la derecha. |
| div {  **padding**: 12px 20px 18px;  }  12px de relleno para arriba.  20px de relleno para izquierda y derecha. 18px de relleno para abajo. | **Padding.** También podemos hacerlo con 3 valores. El primero va a indicar el padding de arriba, el segundo el de la izquierda y la derecha y el tercero será el de abajo. |
| div {  **padding**: 12px 15px 18px 21px;  }  12px de relleno para arriba.  15px de relleno para la derecha.  18px de relleno para abajo.  21px de relleno para la izquierda. | **Padding.** Por último, podemos utilizar 4 valores que representarán los cuatro costados individualmente. Empezaremos por el valor de arriba y seguiremos (en el sentido de las agujas del reloj) por derecha, abajo e izquierda. |
| div {  **border**: solid 3px yellow;  } | **Border.** Hace referencia al borde del elemento. Se ubica entre el contenido y el margen. Podemos asignarle un valor a esta propiedad definiendo el **estilo de línea, su espesor y su color**. El estilo de línea puede ser s**olid, dotted, dashed o double**. El espesor de línea puede ser cualquier unidad de medida de CSS. El color puede ser cualquier color válido de CSS. |
| div { **margin**: 15px;}  div { margin: 20px 15px }  div { margin: 20px 15px 30px }  div { margin: 20px 15px 30px 25px } | **Margin.** Hace referencia al margen exterior del elemento. Sirve para separar una caja de la otra. Podemos asignarle valor a esta propiedad usando cualquier unidad de medida, como los píxeles (px), indicando 1 valor para los 4 lados de la caja. Se aplica a cada lado de igual manera que el **padding**. |
| **Box-sizing: content-box.** | Por defecto su valor será content-box, y el comportamiento será el de aplicarle el ancho y alto que definamos al contenido del elemento. |
| **Box-sizing: border box** | Si le asignamos el valor border-box, el ancho y alto que indiquemos tomará en cuenta no solo el contenido del elemento, sino también el padding y el borde, dejando solo el margen por fuera. |

Es una práctica muy común asignarle box-sizing: border-box a todos los elementos del sitio con la siguiente línea de código: \* {box-sizing: border-box}

**Etiqueta viewport:**

La etiqueta <meta> viewport da al browser instrucciones de cómo se debe dimensionar y escalar la página web al cargarse. Esta es la estructura básica de este tag. En ocasiones, puede que lo veamos con más información.

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

**Medidas relativas:**

Las medidas relativas son aquellas que tienen en cuenta el contexto donde se encuentran. Si el contexto cambia, estas medidas cambiarán con él.

* **Porcentajes:**

Cualquier medida expresada en porcentaje siempre estará relacionada con la medida (en ese mismo eje) del elemento padre que la contiene. Si el contenedor padre mide 300px de ancho y le asignamos un ancho del 50% al elemento interior, este medirá 150px (el 50% del ancho padre).

.elementoContenedor { width: 300px }

.elementoInterior { width: 50% } // Será 150px

* **Em y Rem:**

Los **ems** son siempre relativos al elemento padre. Tomarán como valor de referencia la propiedad **font-size**. El punto de referencia inicial es el valor de font-size del elemento <html> que **por defecto es 16px**. El resto de los elementos tendrán 1em de font-size que equivale al mismo valor que tenga el padre. Si le asignamos **1.5em** al font-size de cualquier elemento, el tamaño resultante será el valor del padre multiplicado por el valor en ems → 16px \* 1.5 = **24px**.

p { font-size: 1.5em } // 16px \* 1.5 = 24px

Si utilizamos ems en una propiedad que no sea font-size, se tomará para el cálculo el font-size que tenga el elemento que estemos modificando. Por lo general, se utilizan este tipo de unidades para todo lo que sea relacionado a la tipografía, pero también podemos aplicarlo a otras propiedades como margin y padding para que varíen en función de los tamaños de las fuentes.

p {font-size: 20px;

line-height: 2em; // 20px \* 2 = 40px

padding: 1.5em; // 20px \* 1.5 = 30px

}

Los **rems** funcionan muy parecidos a los ems, con la diferencia de que siempre tomarán de base el tamaño de Font-size del elemento <html>. Eso quiere decir que el tamaño expresado en rems no modificará el de los elementos hijos y tampoco se verá afectado por el del elemento padre. Por lo general es mejor usar rems en lugar de ems ya que conservamos las ventajas de una unidad relativa, pero nos evitamos hacer cálculos complejos y estar pendientes de cómo se afectan los elementos entre sí.

p {font-size: 1.5rem;}

* **vw y vh: medidas del viewport**

El viewport es el espacio visible que tiene el navegador para mostrar el sitio. Eso quiere decir que se pueden utilizar medidas relativas a este espacio para poder determinar el tamaño de ciertos elementos.

Viewport Width y Viewport Height:

**vw** o viewport width es relativo al ancho total del viewport.

**vh** o viewport height es relativo al alto total del viewport.

Ambas medidas están expresadas como porcentajes del total, eso quiere decir que 50vh será equivalente al 50% del alto disponible en el viewport.

div {

width: 25vw; // 25% del ancho disponible

height: 50vh; // 50% del alto disponible

}

**Media queries:**

Son un conjunto de reglas de CSS que nos permiten cambiar los estilos de los elementos en función de las características del dispositivo que esté visualizando nuestro sitio. La mayor parte de nuestros estilos estarán fuera de las media queries. Dentro de cada media query escribiremos solo aquello que necesitemos ajustar para ese tamaño de viewport. Por lo general, se escriben al final de nuestra hoja de CSS.

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

|  |  |
| --- | --- |
| **Sintaxis** | **Uso** |
| @media (min-width: 460px) {  body {  background: red;  }  } | **min-width.** Al especificar min-width estamos diciendo: “si como mínimo el viewport tiene N píxeles de ancho, apliquemos estas reglas”. Similar a decir: “Desde este ancho, hacia arriba”. |
| @media (max-width: 768px) {  body {  background: yellow;  }  } | **max-width.** Al especificar max-width estamos diciendo: “si como máximo viewport tiene N píxeles de ancho, apliquemos estas reglas”. Similar a decir: “Desde este ancho, hacia abajo”. |
| @media (max-width: 460px) and (orientation: landscape) {  body {  background: blue;  }  } | **Orientación.** Al especificar la orientación **(portrait o landscape)** estamos diciendo: “si como máximo el viewport tiene N píxeles de ancho y además el dispositivo está en posición vertical/horizontal, apliquemos estas reglas” |

**Flexbox:**

Para empezar a trabajar con Flexbox tenemos que definir un **flex-container**. Para eso usamos la propiedad **display** con el valor **flex**. De esta forma, estamos habilitando un contexto flex, para trabajar con los hijos directos del elemento. La propiedad display también puede recibir el valor **inline-flex**.

Para aplicar un Flexbox dentro de un elemento debemos escribir lo siguiente:

* Display: flex;

Por defecto, los elementos hijos de un contenedor flex van a tratar de entrar todos en una misma línea. Para aclararle al contenedor que debe respetar el ancho definido de sus hijos usamos la propiedad **flex-wrap** con el valor **wrap**:

* Display:flex;
* Flex-wrap: wrap;

flex-wrap también puede recibir los valores **nowrap** y **wrap-reverse**.

**Ejes en Flexbox:**

Flexbox trabaja con dos ejes para desarrollar todo su flujo interno: el eje X y el eje Y. Según cómo decidamos ordenar los elementos, llamaremos **main axis** a uno y **cross axis** al otro.

Definiendo el eje principal de nuestro contenedor flex estamos determinando el flujo que tendrán los elementos dentro del contenedor. En función de cuál es el eje principal, los elementos se distribuyen en filas horizontales o en columnas verticales.

* **Flex-direction:**

Con esta propiedad definimos el main axis (eje principal) del contenedor, que puede ser tanto horizontal como vertical. El cross axis (eje transversal) será la dirección perpendicular al main axis.

* **flex-direction: row**

Los ítems se disponen en el eje x, de izquierda a derecha. Si no le aclaramos la propiedad flex-direction al contenedor, row es el valor por defecto.

* **flex-direction: row-reverse**

Los ítems se disponen en el eje x, de derecha a izquierda. En este caso, estamos invirtiendo el inicio y fin del main-axis.

* **flex-direction: column**

Los ítems se disponen en el eje y, de arriba hacia abajo.

* **flex-direction: column-reverse**

Los ítems se disponen en el eje y, de abajo hacia arriba. En este caso, estamos invirtiendo el inicio y fin del main-axis.

**Justify-content:**

Con esta propiedad alineamos los ítems a lo largo del main axis. Si es horizontal, se alinearán en función de la fila. Si es vertical, se alinearán en función de la columna.

* **justify-content: flex-start**

Los ítems se alinean respecto del inicio del main axis que hayamos definido. Si no le aclaramos el justify-content al contenedor, flex-start es el valor por defecto.

* **justify-content: flex-end**

Los ítems se alinean respecto del final del main axis que hayamos definido.

* **justify-content: center**

Los ítems se alinean en el centro del main axis.

* **justify-content: space-between**

Los ítems se distribuyen de manera uniforme. El primer ítem será enviado al inicio del main axis, y el último ítem, al final. El espacio libre se repartirá para separar los ítems.

* **justify-content: space-around**

Los ítems se distribuyen de manera uniforme. El espacio libre disponible se repartirá entre todos los elementos. Del espacio que le toque a cada elemento, la mitad irá a la derecha y la otra a la izquierda (o arriba y abajo en caso de que sean columnas).

**Align-items:**

Con esta propiedad alineamos los ítems a lo largo del cross axis. Si no aclaramos esta propiedad, el valor por defecto es stretch, en otras palabras, los ítems ocuparán todo el espacio disponible en el cross axis.

* **align-items: stretch**

Los ítems se ajustan para abarcar todo el contenedor. Si el cross axis es vertical, se ajustan en función de la columna. Si el cross axis es horizontal, se ajustan en función de la fila.

* **align-items: flex-start**

Los ítems se alinean al inicio del cross-axis.

* **align-items: flex-end**

Los ítems se alinean al final del eje transversal.

* **align-items: center**

Los ítems se alinean al centro del eje transversal.

**Ítems en Flexbox:**

|  |  |
| --- | --- |
| **Sintaxis** | **Uso** |
| .caja {  order: 1;  } | **Order.** Con esta propiedad controlamos el orden de cada ítem, sin importar el orden original que tengan en la estructura HTML. Esta propiedad recibe un número entero, positivo o negativo, como valor. Por defecto, todos los ítems flex tienen un order: 0 implícito, aunque no se especifique. |
| .caja-q { order: 1; }  Q-M-D // quedará M-D-Q | **Order: número positivo.** Si le asignamos a la caja Q (que posee la clase caja-q) la propiedad order con valor 1, esta pasará al final de la fila por ser el número más alto. Recordemos que, por defecto, el valor del orden de cada ítem es 0. |
| .caja-d { order: -1; }  M-D-Q // quedará D-M-Q | **Order: número negativo.** Si ahora le asignamos a la caja D la propiedad order con un -1 como valor, esta pasará al principio de la fila. Colocando al ítem con el orden más pequeño primero. |
|  | **Flex-grow.** Con esta propiedad definimos cuánto puede llegar a crecer un ítem en caso de disponer de espacio libre en el contenedor. Configura un crecimiento flexible para el elemento. |
| .caja-a, .caja-b {  flex-grow: 1;  } | **Flex-grow.** Si ambos ítems tienen la propiedad flex-grow con valor 1, a medida que el contenedor se agrande, irán abarcando el espacio disponible en partes iguales. |
| .caja-b {  flex-grow: 1;  } | **Flex-grow.** Si un solo ítem tienen la propiedad flex-grow, este intentará ocupar el espacio libre disponible, a medida que el contenedor se agrande, según la proporción que definamos con el valor. |
| El número que le asignamos a flex-grow determina qué cantidad de espacio disponible dentro del contenedor flexible tiene que ocupar ese ítem. 1 equivale al 100% del espacio disponible, y 0 al 0%. Podemos usar cualquier valor en el medio, como 0.25 para el 25%. | |
|  | **Align-self.** Nos permite alinear, sobre el cross axis, a cada ítem al que le apliquemos esta propiedad, independientemente de la alineación que se haya definido en el contenedor flex con align-items. |
| .contenedor-padre {  align-items: flex-start;  }  .caja-dos {  align-self: flex-end;  } | **Align-self: flex-end**. El ítem se alinea al final del eje transversal. |
| .contenedor-padre {  align-items: flex-start;  }  .caja-dos {  align-self: center;  } | **Align-self: center.** Con center, el ítem se alinea al centro del eje transversal. |
| .contenedor-padre {  align-items: flex-end;  }  .caja-dos {  align-self: flex-start;  } | **align-self: flex-start.** Con flex-start el ítem se alinea al inicio del eje transversal. |
| .caja-uno, .caja-tres {  align-self: flex-start;  }  .caja-dos {  align-self: stretch;  } | **align-self: stretch**. Con stretch, el ítem se ajusta hasta abarcar todo el cross axis, es el comportamiento por defecto. Funciona siempre que el elemento no tenga definida una altura. |