HyperText Markup Language

Recomienda nuevamente Visual Studio Code.

Plugins recomendados:g

1. Open In Browser.
2. Html end tag labels.
3. Auto rename tags
4. Indent Rainbow
5. Live Server.
6. Prettier. Con SHIFT ALT F te pone todo bonito

Etiquetas habituales:

* Div. Divide
* Span. Bloques de texto que no son párrafos.
* A (anchor o ancla). Permiten poner enlaces. Href es su gran atributo, donde se pone el enlace.
* Ol y ul (lista ordenada y desordenada). Por dentro tienen los Li.
* Img. Con sus atributos src y alt

Form

Form. Dentro del mismo hay otras etiquetas, como label, la cual tiene el atributo for (para qué sirve esa etiqueta). Hace referencia a un input que tenga un atributo Id igual. Atributo Action de form: Ahí puse el enlace a otra página html. Método =”Post” => hace algo desde el servidor. Otro es el método =”Get”.

Hacer que los campos sean obligatorios. Atributo required.

Se les puede poner el tipo a los campos. Text y Password (para que no se ven las constraseñas), puede ser numérico =”number”, “email”, “date”.

A los numéricos se les puede poner min=”0” así como max =”200” son los mínimos y máximos.

Botones. Tipos: submit (enviar), reset.

Table

Con tr se define una fila (table row). Con th (table header) las columnas. Con td se introduce la info (table data).

Multimedia

Img para imágens. Src para linkear. Alt como texto por si falla imagen. Width (ancho), Height.

Audio. Src + controls

Video. Con etiqueta Video. Hay 3 tipos de videos que soporta html5: webm , mp4 y Ogg.

Se le puede poner witdh y height también.



 <video width="300"height="300" controls src="Vicentico Freak.mp4" >

    </video>

El muchacho agrego esa “propiedad” Source src y luego type, también controls. Yo solo src. Funca bien.

Dice que el atributo autoplay es peligroso. Reproduce automáticamente. Puede llevar el atributo muted también. Para que arranque, pero en silencio. Loop es para que se repita al finalizar.

Controls agrega la posibilidad de poner play, pausa, controles.

Custom Style Sheets

Selectores. Id y class.

Id=> #blabla

Class => .blabla

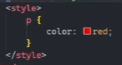
\* es selector universal.

Con la , podés agregar a varios juntos.



3 formas de insertar estilos:

En el HTML:



Desde PureCss.io se pueden linkear estilos.

Luego, al elemento en el HTML se le pone la clase que corresponda al elemento linkeado.

Estos son estilos inline.



Coolors.com :

Permite crear paletas aleatorias.



Copiando el código de abajo podés apropiarte de los mismos.

Rgb!=rgba la a sobrante refiere a la opacidad, es un cuarto valor, que va del 0 al 100.

Para pintar un fondo, p ej, se puede utilizar un selector en un div.

También está eso, con atributo url para usar una imagen por fondo.

background-image:



Ahí se ve la imagen partida, y tras partirse se repite.

background-size: 100%;

Con eso se arregla

Trabajar con pixeles no es responsive friendly

Fuentes

Font-family.

Font-Weight con esto se pone en negrita.

Fonts.google.com

Se puede copiar el link con su referencia al head o al css.

Al css no se le clickea link sino import. Se copia todo meno <style> y </style>

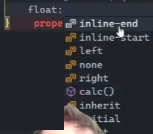
Disposiciones y alineaciones

Div van uno por debajo del otro. Por defecto tienen disposición (display) de bloque.

Span, se acomoda uno al lado del otro. Por defecto tiene display inline.

Las imágenes tienen inline-block, el cual permite cambiar height y width.

Propiedad float para los inline: (Así se mueve de cierta manera en la visual)

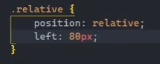


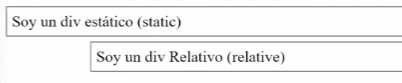
5 posicionamientos en Css:

Static. Este viene por defecto. No se ve afectado por cambio de posiciones que uno le codee.

Al resto de la modifica con propiedad position.

Relative. Se posiciona en relación a la posición original.



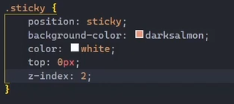


Fix. Se usa mucho en barras de navegación, que siempre tienen que quedar en el mismo lugar de la pantalla, independientemente del Scroll. Se posiciona en relación a la pantalla.

Absolute. Se posiciona en relación al padre.



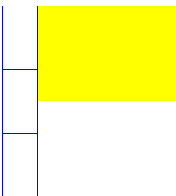
Sticky. Mezcla de relativo y de fijo. Tiene su posición, pero cuando se scrollea queda en algún lugar. Comun usarlo para menúes que siempre debene estar visibles.



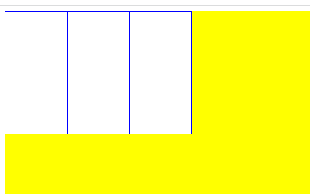
z-index es si va por debajo o por delante.

FlexBox

Sobre todo para vistas responsive.



El amarillo es el div, que contendría a los 3 otros divs. Aquí no le puse al div padre display flex. A continuación si, y se ve como los contiene. Por defecto se ubica uno al lado del otro.



Justify-content es práctico. Con space between queda piola.

Flex-wrap. Con Wrap intenta no redimensionar, sino que manda elementos para abajo. Con nowrap redimensiona.

Align-items: por defecto es stretch. El hijo ocupa todo el alto del padre. Pero si le ponés a los divs hijos alguna altura, tomará la de los hijos. Con center lo centra.

Flex-direction. Establece el eje principal. Por defecto es row. Si se le pone row reverse. Se invierte. Para que el eje principal sea vertical es con flex direction columna.

En codepen.io hay algunos ayudines para ver.

Aconseja evitar sobre utilizar los flex, cuando no hagan falta.

Overflow

(Desbordamiento)

Sucede si tenemos un contenedor fijo y un contenido muy grande.

Overflow hidden: Todo lo que sobresale, se oculta. No hay manera de acceder a esa info.

Overflow Scroll. Se crea una barra para scrollear.

Overvlow visible. Por defecto. Sobresalen los excesos.

Overflow auto. Optimiza. Pone barras solo donde es necesario.

Por defecto css viene con un margin y un padding. Si se quiere eliminar eso se hace con el asterisco y llevando los valores a 0

**Hover**

 .imagen-galeria:hover{

    opacity: 0.3;

    cursor: pointer;

 }

Con hover cuando el cursor pasa por arriba de algo, sucede el código de adentro. Cambia la opacidad y el mouse adviene manito.

Video que ocupe toda la pantalla, sin scroll y fijo:

#videoZ{

    width: 100vw;

    height: 100vh;

    object-fit: cover;

    position:fixed;

}

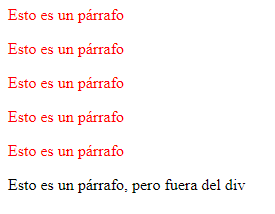
Anidar selectores:

.contenedor p {

    color:red;

}

Primero puso .+clase y luego puso p (por los párrafos):



Ahí se ve como cambia el color de los p, que están dentro de “contenedor” pero no el otro.



Ahí agrego un botón con un Id, y así lo codea.

Pseudoclases (aquí va el hover) (:)

Aportan un estado diferente.

Hover. Afecta cuando el mouse se posa por arriba.

First child. Afecta a la primera aparición de determinado tipo de elemento.

Luego agrega otros. Para el manejo de los links (de a):

A:link Es el color que le da al enlace no visitado.

A:visited. Es el color que le da al enlace ya visitado.

A:hover. Cuando el mouse se posa encima.

A:active. Cuando se le hace click.

Pseudoelementos (::)

::after y ::before

span.frase::before{

    content: '<< ';

    font-size: xx-small;

}

span.frase::after{

    content: ' >>';

    font-size: xx-small;

}

Agrega ese contenido, antes y después.



First-letter afecta solo a la primera letra de un p, en este caso



Similarmente, está first-line.

Selection refiere a cuando se selecciona algo, se suele por default poner el fondo blanco.

Specificity:



Afecta a los miembros de la clase .listado que sean ítems, de listados sin ordenar.

Especificidad

Si un elemento forma parte de 2 clases y ambas le dan color a la letra, va a prevalecer la que esté escrita después, ya que pisa a la primera. Sin embargo, hay cierta jerarquía:

El id es lo mas específico, sigue la clase y luego la etiqueta. El mas específico prevalece.

Si el estilo se define en línea, inline, dentro del html es mas específico.



Con !important se lo hace el mas prevalecedor de todos.

Acceder a selectors mediante atributos:

.input[type="password"]{

    color: red;

}

Input moderno

.input-moderno{

    border: none;

    border-bottom: 1px solid #c2c2c2;

    outline: none;

    padding: 5px;

    color:#949393;

}

.input-moderno:focus{

    border-bottom: 1px solid #5f5f5f;;

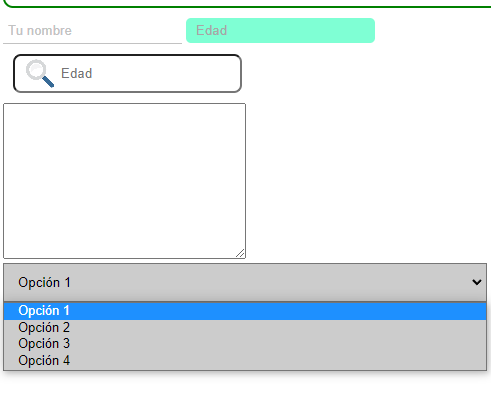
}

Le saca todos los bordes, luego agrega el de abajo. A ese le da un gris clarito. Con el focus, cuando se lo selecciona se le da un color mas oscuro.

Con el outline none, al ser seleccionado no aparece un recuadro bastante feo.

Placeholder escribe un texto de ayuda. Se pone en html y luego en Css se le da estilo.

Select+Options



Estas 4 opciones del final son las option del select

LoginForm

Recomienda inspirarse en otros que si sepan.

Dribbble.com hay muchos diseños hechos ya por otros.

Extensión de Google Chrome para capturar colores de la pantalla: color pick eyedropper.

Descarga letras de Google fonts. Cada fuente hace que tarde algo en cargar. Por eso, si no se la utiliza, lo mejor es borrarla.

 <style>

        @import url('https://fonts.googleapis.com/css2?family=Ubuntu:wght@300;400;500;700&display=swap');

    </style>

En este caso, se puede importar todo eso al html o solamente la url al css.

Centrar contenido

Recomienda usar el display flex:

body{

    display: flex;

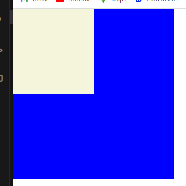
    justify-content: center;

    align-items: center;

    min-height: 100vh;

}

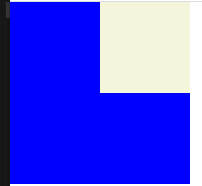
Transiciones



.padre:hover .hijo{

    transform: translateX(100%);

}



Al posar el click sobre el padre (cuadrado azul), se traslada el cuadrado hijo.

.hijo{

    width: 50%;

    height: 50%;

    background-color: beige;

    transition: all 1s;

}

.padre:hover .hijo{

    transform: translateX(100%);

}

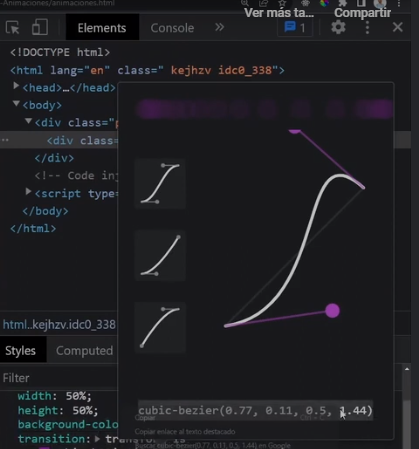
Luego agrega el transition al hijo y se mueve super suave hacia la derecha: al final se ralentiza el movimiento, choca lentamente (ese es el valor por defecto, que se escribe “ease”. Si se quiere que todo el movimiento se haga a la misma velocidad se pone transition:all 1s linear.

Dentro del hover también podrían cambiarse otros atributos del hijo.

El all hace referencia a que todo cambio va a hacerse en un segundo. Se puede especificar, por ejemplo, poniendo “transition: transform 1s;”.

Podría ir hasta el final del movimiento y rebotar, o suceder otras cosas.

Dice que se puede probar, abirendo el editor para programadores dentro de Chrome, y luego copiar el código al css.



También se le puede dar delay.

 transition: all 2s 1s;

Se ponen 2 valores; el segundo representa al delay.

Animaciones

@keyframes se usa para especificar fotogramas clave.

@keyframes traslacion{

    0%{

        background-color: red;

    }

    40%{

        background-color: black;

    }

    80%{

        background-color: white;

    }

    100%{

        background-color: red;

    }

}

.padre:hover .hijo2{

    animation: traslacion 2s;

}

Se puede agregar forwards. Hace que de mantenerse el click sobre el objeto hover, la animación permanece en su estado final.

Se puede agregar un número o infinite. Eso da la cantidad de veces que se va a repetir.

Se puede agregar alternate. Una vez que llegue al final va a volver al ppio.

En estos casos, la animación se activa si hoover se activa. Pero, el objeto podría estar por defecto animado y frenarse si hay hoover. Eso se hace con animation-play-state:pause;

Tooltip

Línea de texto, consejo o pista que nos da info de lo que hace otro elemento.

Lo hace agregando un span al botón.

Al botón le pone position relative y al tooltip relative.

Al principio el tooltip no debe estar visible. Eso puede hacerse con visibility: invisible o con opacity: 0% En este segundo caso es mas fácil hacer la transición.

.tooltip{

position: absolute;

opacity: 0%;

top: -50px;

left: -100%;

transform: translateX(25%);

/\*Vieja forma de hacerlo en el renglón de arriba\*/

color: #ddd;

background-color: grey;

padding: 5px 10px;

border-radius: 5px;

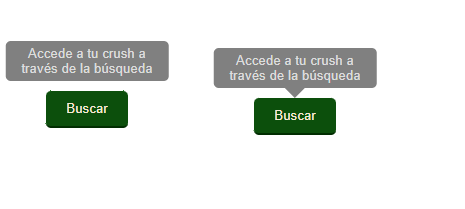
transition: opacity 200ms;

}

.btn:hover .tooltip{

       opacity: 1;

}



.tooltip::before{

content: '';

border:solid 10px transparent;

border-top-color: grey ;

position: absolute;

top: 40px;

left: 50%;

transform: translateX(-50%);

}

Con eso de ahí le agrega un triángulo, como si el tooltrip apuntase al botón.

Con las líneas del borde crea al triángulo, con las líneas restantes, lo acomoda

Diseño responsive

Eyy, hace una cosita parecida al término de herencias. En vez de poner escenario 1 y escenario 2, pone escenario a ambos y les agrega escenario 1 y escenario 2. Todo lo común va a escenario, con el numerito los diferencia.

.escenario{

    margin: 10px 0px;

    border: 1px solid ccc(0, 0%, 80%);

    padding: 10px;

}

.escenario-1{

    width: 100%;

}

.escenario-2{

    width: 50%;

}

Pixel es una unidad fija.

Porcentaje es relativa al padre. En este caso, a escenario 1 o escenario 2.

Vw es relativo al tamaño de la pantalla.

.escenario{

    margin: 10px 0px;

    padding: 10px;

    border: 1px solid grey;

}

.escenario-1{

    width: 100%;

}

.escenario-2{

    width: 70%;

}

.px{

    background-color: green;

    width: 200px;

}

.porcentaje{

    background-color: red;

    width: 50%;

}

Todo eso, funciona así:



Unidades Em y Rem

Hay píxeles físicos y píxeles digitales o lógico.

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

Esa línea del html indica que un pixel físico corresponde con uno lógico. En algún Iphone esa relación es distinta, de 1 a 2.

Hacen referencia al ***tamaño de fuente*** de donde están contenidos.

Em refiere al tamaño de fuente del ***contenedor***. Rem refiere no al contenedor sino a la ***raíz***. Creo que no el body sino el \* o html.

Estas 2 son las que mas se usan a la hora de crear diseños responsive, sobre todo la **Rem.**

html{

    font-size: 12px;

}

.escenario{

    padding: 10px;

    border: 1px solid grey;

    margin: 10px;

}

.escenario-1{

    font-size: 32px;

}

.escenario-2{

    font-size: 16px;

}

.em{

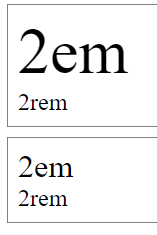
    font-size: 2em;

}

.rem{

    font-size: 2rem;

}



Ambos em se dejan modificar por los escenarios, en tanto que los Rem no, por mas que se defina cierto Font-size a esos contenedores, el que usa es del de html.

Media Queries

@media hace referencia a que la condición que se codeen van a ser afectados solo para los dispositivos que yo especifique. Por defecto es media all. Puede ser screen, print o speech. Print, por ejemplo, es si se va a imprimir. Lo mas común es media all, o también media screen.

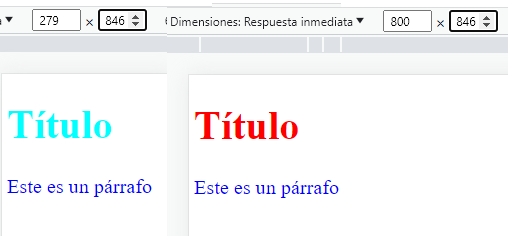
@media screen and (max-width: 576px) {

    h1{

        color:aqua;

    }

}



@media (max-width: 576px)

Se puede simplificar así.

Si se quiere que de 0 a 576 se pone min-width:576px. Una segunda condición iría igual, pero cambiando la pixeleada.

@media (min-width: 576px) {

    h1{

        color:aqua;

    }

}

@media (min-width:980px) {

    h1{

        color:green;

    }

}

También se podría hacer con el height pero no se suele utilizar así.

Landscape es el celular horizontal, portrait es vertical. Si la altura es mayor que la altura es portrait y viceversa.

Se pueden anidar condiciones también.

@media (orientation:landscape)and (min-height:300px) {

    p {

        color:violet;

    }

}

Flexbox

Si hablamos de responsive hay 2 disposiciones mas utilizadas:

Flexbox y GridBox.

Contenedor e ítems.

Flex direction por defecto es row.

Eje principal es el de las X. Utiliza Justify content.

Eje secundario es el de las Y. Utiliza align ítems.

Stretch estira hasta alcanzar el total.

Se puede modificar el comportamiento de un solo elemento. No se toca el contenedor sino el ítem, por medio de align-self o justify-self.

Flex wrap por defecto es no wrap. El wrap intenta preservar el tamaño de los elementos. Si se achica la pantalla, se mandarán elementos para abajo para lograr su cometido.

ViewPort, Grid y Templates

    grid-template-columns: auto auto auto;

En html pongo un contenedor y nuevo ítems. Inicialmente se dispone en forma de lista, todo en la misma columna. Con esa línea que agregué ahí, ya te dispone en un 3 por 3. Otra forma, idéntico resultado:

grid-template-columns: repeat(3,auto);

Con esto te va a hacer 2 columnas, de tamaños distintos:

grid-template-columns: 50px 3rem;

Esta es otra. Fr es fracción o pedazo. Se divide el ancho en 4 y cada columna ocupa lo indicado.

grid-template-columns: 1fr 1fr 2fr;

También se pueden modificar los rows.

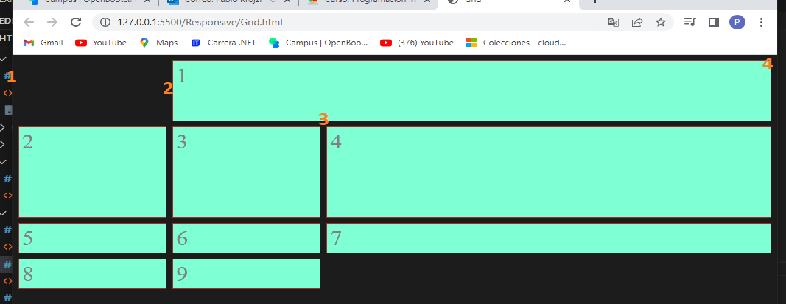
También se puede trabajar con áreas, con grid-template-areas.

Grid-template-areas: ‘head head head’ ‘main main aside’ ‘footer footer footer’.

Esta es otra forma, va desde la columna 2 hasta la 4 (por así decir).

.item-1{

    grid-column: 2/4;}



.item-2{

    grid-column: 3 /span 2;

}

Ahí le dice que empiece en 3, y que ocupe 2 columnas. No aclara hasta donde.

.item-5{

    grid-area: 3/2/4/4;

}

Grid-area: Desde fila/desde columna/hasta fila/hasta columna.

Definir variables y utilizarlas luego

:root{

   --color-principal: rgb(0,187,255) ;

   --ancho-principal: 250px;

}

.titulo{

    color:var(--color-principal);}