{less}

Introducción

qué matrioshkas es LESS?

¿Qué es LESS?

LESS (todo con mayús) es un preprocesador de CSS

```
//añadir tus hojas de estilo .less
//siempre va antes
<link rel="stylesheet/less"
      type="text/css" href="estilo.
      less">

//añadir less.js
<script src="less.js"
      type="text/javascript">
      </script>
```

Variables

EN CSS?!?!?! ESO ES POSIBLE?!?!

¿Y cómo declaro variables?

```
/* variable @hotpink que almacena un color hexadecimal*/
@hotpink: #FF69B4;
```

¿cómo las uso?

```
/* aplica el color @hotpink al elemento con id elem */
#elem{
    background-color: @hotpink;
}
```

Práctica 1

Crear un archivo .less con 7 variables que contengan colores.

Anidación

como los ifs que están dentro de ifs que están dentro de ifs ...

Sin LESS

```
div{
    color: black;
}
div . stuff{
    font-size: 30px;
}
div .more-stuff{
    width: 100px;
}
```

Con LESS

```
div{
    color: black;;
    .stuff{
        font-size: 30px;;
    }
    .more-stuff{
        width: 100px;;
    }
}
```

También se puede usar con pseudo-selectores

```
div{
    background-color: black;
    /* estilo del div en hover, el
& se refiere al selector parent */
    &:hover{
        background-color: blue;
    }
}
```

Práctica 2

Organizar el estilo utilizando LESS.

Mixins

Podemos agrupar propiedades para utilizarlas en varios elementos.

```
// declara mixin “bluebg”
.bluebg{
      background-color: blue;;
}

//aplica las reglas de bluebg
#elemento{
      .bluebg;
      width: 100px;
}
```

Los mixins pueden contener selectores.

```
#hover(){
    background-color: blue;
    &:hover{
        color: cyan;;
    }
}

#elemento{
    #hover();
}
```

Práctica 3

Importar los colores de la práctica 1 y utilizarlos para darle estilo a las clases de prac4.html
Todos las clases deben de compartir propiedades excepto el font y el bg-color. Aplicar mixins

Mixins

(con parámetros)

```
// recibe un color y se puede utilizar en el mixin
.mixi-mixin(@color){
    background-color: @color;
}

#elem{
    mixi-mixin(blue);
}
```

Se pueden tener valores por default

```
// si no se especifica, @color tendrá el valor green
.mixi-mixin(@color: green){
    background-color: @color;
}

#elem{
    // toma el valor por default
    mixi-mixin();
}
```

Mixins con muchos parámetros

```
// los parámetros se separan con “;”
.colores(@texto; @bg){
    background-color: @bg;
    color: @texto;;
}
#elemento{
    .colores(cyan; white);
}
// o así, podemos referirnos a los
parámetros por nombre
#otro{
    .colores(@bg: cyan; @texto:
white);
}
```

@argument

se refiere a todos los argumentos llamados con el mixin

```
.borde(@w: default; @style:
      solid; @color: default){
      border: @arguments;
}
#elemento{
      .borde(5px; dotted; white);
}
```

Práctica 4

Utilizar mixins con parámetros para mejorar la práctica 3.

Funciones matemáticas

los diseñadores saben matemáticas?

ceil()

redondea al siguiente integer más alto

Parámetros

entrada: número[float]

salida: integer

```
ceil(2.4);
//devuelve 3
```

floor()

redondea al siguiente integer más bajo

Parámetros

entrada: número[float]

salida: integer

```
ceil(2.4);
//devuelve 2
```

percentage()

convierte un float a su equivalente en porcentaje

Parámetros

entrada: número[float]

salida: string

```
percentage(0.5);
//devuelve 50%
```

round()

redondea un float

Parámetros

entrada: número[float]

salida: integer

```
round(9.5);
//devuelve 10
```

pow()

eleva un número a la n potencia

Parámetros

entrada: # base, # exp

salida: número

el valor que devuelve tendrá las dimensiones de la base

```
pow(2, 3); //8
pow(3cm, 3px); //27 cm
```

sqrt()

devuelve la raíz cuadrada
de un número

Parámetros

entrada: número

salida: número

se mantienen las dimensiones
del número ingresado

```
sqrt(36px);
//devuelve 6px
```

abs()

devuelve el valor absoluto
de un número

Parámetros

entrada: número

salida: número

se mantienen las dimensiones
del número ingresado

```
abs(-18%);
//devuelve 18%
```

pi()

devuelve el valor absoluto de pi

Parámetros

entrada:

salida: número

```
pi();
//devuelve
3.141592653589793
```

min()

devuelve el menor de uno o más valores

Parámetros

entrada: uno o + valores

salida: el valor menor

```
min(5, 10); // 5
min(-18%, -3%, 20%); //-18%
```

max()

devuelve el máximo de uno o más valores

Parámetros

entrada: uno o + valores

salida: el valor mayor

```
max(5, 10); // 10
max(18%, -3%, 20%);//20%
```

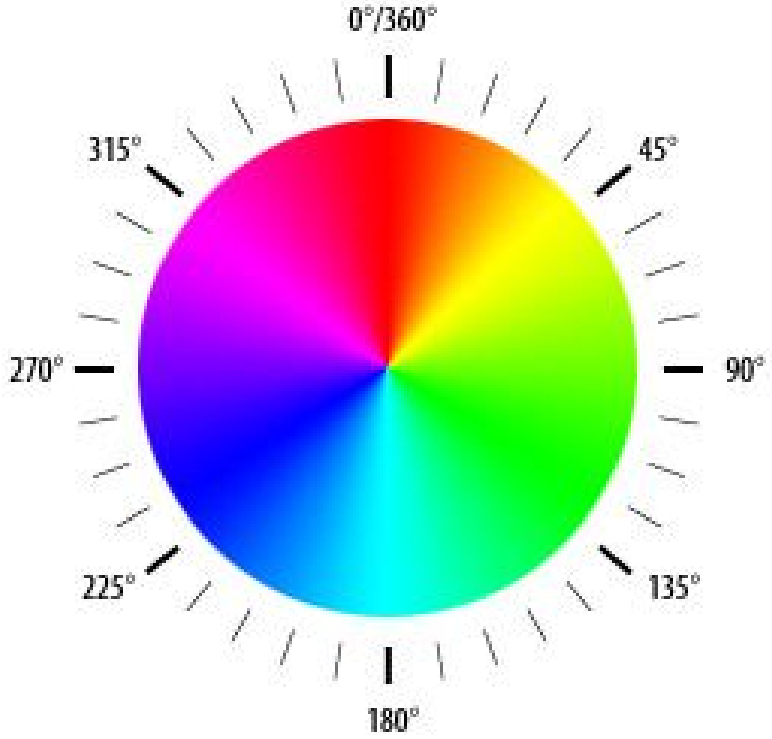
Funciones de color

Propiedades del color

lo que debieron de haber visto en pintura

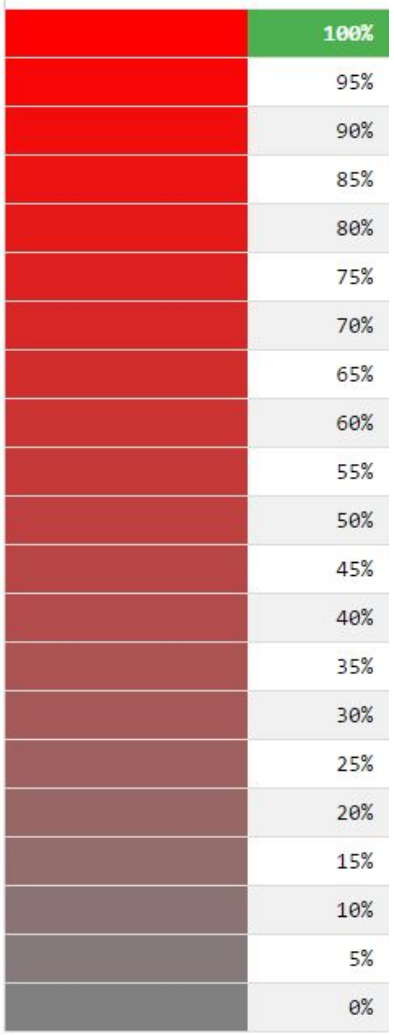
Tono

o Hue, define un color puro en términos de verde, rojo o magenta



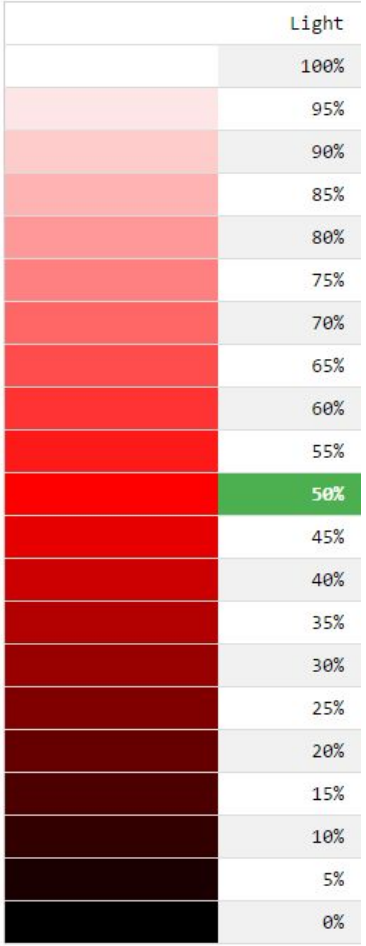
Saturación

define un rango del color puro (100%) a gris (0%)



Luminosidad

define un rango de oscuro (0%) a completamente iluminado (100%)



Definición de colores

rgb()

crea un objeto de color a partir de valores red, green y blue

Parámetros

entrada: valores red, green, blue 0-255 ó 0-100%

salida: color

```
rgb(255, 10, 10);
// #FF0a0a
```

rgba()

crea un objeto de color a partir de valores red, green, blue y alpha

Parámetros

entrada: como el rgb y 0-1 ó 0-100% para alpha

salida: color

```
rgba(255, 10, 10, 0.5);
// #FF0a0a con 0.5 de transparencia
```

hsl()

crea un objeto de color a partir de valores hue, saturation y lightness

Parámetros

entrada: 0-360 para hue, 0-1 ó 0-100% para saturation/lightness

salida: color

```
hsl(0, 100%, 52%);
// #FF0a0a
```

hsla()

crea un objeto de color a partir de valores hue, saturation, lightness y alpha

Parámetros

entrada: lo mismo que hsl y 0-1 ó 0-100% para alpha

salida: color

```
hsla(0, 100%, 52%, 0.5);
// #FF0a0a con 0.5 de transparencia
```

Práctica 5

Definir los colores de la práctica 1 en valores hsl y rgb. Comentar a un lado de cada color su valor hexadecimal.

Funciones con colores

hue()

devuelve el valor del canal hue en un objeto de color

Parámetros

entrada: color

salida: integer[0-360]

```
@blood: hsl(0, 100%, 52%);
hue(@blood);
// Devuelve 0
```

saturation()

devuelve el valor del canal saturation en un objeto de color

Parámetros

entrada: color
salida: porcentaje[0-100%]

```
@blood: hsl(0, 100%, 52%);
saturation(@blood);
// Devuelve 100%
```

lightness()

devuelve el valor del canal lightness en un objeto de color

Parámetros

entrada: color

salida: porcentaje[0-100%]

```
@blood: hsl(0, 100%, 52%);
lightness(@blood);
// Devuelve 52%
```

luma()

devuelve el valor del brillo perceptual en un objeto de color

Parámetros

entrada: color

salida: porcentaje[0-100%]

```
@blu: hsl(210, 100%, 20%);
luma(@blu);
// Devuelve 72.30551289%
```

red()

devuelve el valor del canal red en un objeto de color

Parámetros

entrada: color

salida: integer[0-255]

```
@blood: rgb(255, 10, 10);
red(@blood);
// Devuelve 255
```

green()

devuelve el valor del canal green un objeto de color

Parámetros

entrada: color

salida: integer[0-255]

```
@blood: rgb(255, 10, 10);
green(@blood);
// Devuelve 10
```

blue()

devuelve el valor del canal blue en un objeto de color

Parámetros

entrada: color

salida: integer[0-255]

```
@blood: rgb(255, 10, 10);
blue(@blood);
// Devuelve 10
```

Mixin guards

porque LESS no podía quedarse sin condicionales

Se utiliza “when” para establecer un guard

```
// si el color es blanco, el fondo es
      morado
.mixin(@color) when (@color = white){
      color: @color;
      background-color: purple;
}
// aplica el mixin
#elemento{
      .mixin(#FFFFFF);
}
```

Los guard aceptan los operadores de comparación

```
// si el número es >= a 15, aplica un
      padding de ese tamaño
.mixin(@num) when (@num >= 15){
      padding: @num;
}
```

Se utiliza “not” para negar condiciones

```
// si el color no es blanco, el fondo es
    azul
.mixin(@color) when not (@color =
white){
    color: @color;
    background-color: blue;
}
```

Se utiliza “and” para combinar guards

```
// si el color es blanco y padd >= 15,
.mixin(@color; @padd) when (@color =
white) and (@padd >= 15){
        color: @color;
        padding: @padd;
}
```

Para utilizar “or” se usa una coma

```
// si el color es blanco o el padd >= 15,
.mixin(@color; @padd) when (@color =
white), (@padd >= 15){
    color: @color;
    padding: @padd;
}
```

También se pueden usar guards con los selectores

```
@color: #FFFFFF;
// si el color es blanco
#elem when (@color = white){
      color: @color;
      background-color: black;
}
```

Y podemos combinar guards con el selector &

```
@color: #FFFFFF;
#elem{
    color: @color;
    // si el color es blanco
    & when (@color = white){
        background-color: black;
    }
}
```

Práctica 6

Se tienen 4 divs de clase out con los siguientes ID's: color, saturation, lightness y luma. En el .less, declarar 1 variable @col con cualquier color. El color de fondo de #color es @col.

Se utilizará la propiedad “content” para mostrar los mensajes.

Para #saturation, se comprobará que la saturación de @col sea mayor o igual a 50%. Si se cumple la condición mostrar el mensaje “Muy saturado”. De otra manera mostrar el mensaje “Poco saturado”.

Para #lightness, se comprobará que la luminosidad de @col sea mayor o igual a 50%. Si se cumple la condición mostrar el mensaje “Casi blanco”. De otra manera mostrar el mensaje “Casi negro”.

Para #luma, se comprobará que el valor de la propiedad luma de @col sea mayor o igual a 50%. Si se cumple la condición mostrar el mensaje “Muy luminoso”. De otra manera mostrar el mensaje “No es lo suficientemente luminoso”.

Práctica 7

Se tiene un div. En el .less, declarar @col con cualquier color. El color del div será @col.
Si la propiedad luma de @col es menor o igual a 50% el color del texto es blanco. De otra manera el color del texto es negro.

Más funciones de color

ya merito acabamos?

saturate()

incrementa la saturación de un color

Parámetros

entrada: color, aumento[0-100%]

salida: color

```
@green: hsl(90, 80%, 50%);
saturate(@green, 20%);
// Devuelve hsl(90, 100%, 50%)
```

desaturate()

disminuye la saturación de un color

Parámetros

entrada: color, decremento [0-100%]

salida: color

```
@green: hsl(90, 80%, 50%);
desaturate(@green, 20%);
// Devuelve hsl(90, 60%, 50%)
```

lighten()

incrementa la luminosidad de un color

Parámetros

entrada: color, aumento[0-100%]

salida: color

```
@green: hsl(90, 80%, 50%);
lighten(@green, 20%);
// Devuelve hsl(90, 80%, 70%)
```

darken()

disminuye la luminosidad de un color

Parámetros

entrada: color, decremento [0-100%]

salida: color

```
@green: hsl(90, 80%, 50%);
darken(@green, 20%);
// Devuelve hsl(90, 80%, 30%)
```

fadeout()

incrementa la transparencia de un color

Parámetros

entrada: color, aumento[0-100%]

salida: color

```
@green: hsla(90, 80%, 50%, 0.5);
fadeout(@green, 20%);
// Devuelve hsla(90, 80%, 50%, 0.7)
```

fadein()

disminuye la transparencia de un color

Parámetros

entrada: color, decremento [0-100%]

salida: color

```
@green: hsla(90, 80%, 50%, 0.5);
fadein(@green, 20%);
// Devuelve hsla(90, 80%, 50%, 0.3)
```

fade()

establece la transparencia absoluta de un color

Parámetros

entrada: color, decremento [0-100%]

salida: color

```
@green: hsl(90, 80%, 50%);
fade(@green, 20%);
// Devuelve hsla(90, 80%, 50%, 0.2)
```

mix()

combina dos colores en una proporción variable

Parámetros

entrada: color1, color2, proporción(por default es 50%)

salida: color

```
@red: #FF0000;
@blue: #0000FF;
@purple: mix(@red,
@blue);
// Devuelve #800080
```

greyscale()

quita toda la saturación de un color

Parámetros

entrada: color

salida: color

```
@green: hsla(90, 80%, 50%, 0.5);
greyscale(@green);
// Devuelve hsl(90, 0%, 50%)
```

contrast()

escoge cuál de dos colores hace mejor contraste con otro

Parámetros

entrada: color
salida: color

```
@bludarks: #FF0000;
@cremita: #F1D4AF;
contrast(@bludarks);
// white
contrast(@cremita);
// black
```

spin()

rota el ángulo del hue de color en cualquier dirección

Parámetros

entrada: color

salida: color

```
@orange: hsl(10, 90%, 50%);
spin(@orange, 30%);
//devuelve hsl(40, 90%, 50%)
```

Práctica 8

Cambiar los guards de la práctica 7 por la función contrast().

Modificar variables

super~easy

```
/* obtiene el valor de un input y
      lo guarda en una variable
var color = document.
getElementById("dato").value;
/* cambia la variable color en
      LESS */
less.modifyVars({
      "@color": color;

});

// actualiza el estilo
less.refreshStyle();
```

Práctica 9

Agregar al archivo de la práctica 7 un input que le permita al usuario ingresar un color para cambiar @col.

Funciones de tipo

isnumber()

devuelve true si el valor es número

```
isnumber(9001); //true
isnumber(56px); //true
isnumber(7.8%); //true

isnumber("Koala"); //false
isnumber(#660000);
//false
```

isstring()

devuelve true si el valor es un string

```
isstring("Koala"); //true
isstring("#660000"); //true

isstring(9001); //false
isstring(#006600); //false
isstring(true); //false
```

iscolor()

devuelve true si el valor es un color

```
iscolor(#006600); //true
iscolor(hsl(0, 100%, 52%));
//true
iscolor(rgb(0, 200, 5));
//true

iscolor(9001); //false
iscolor(true); //false
```

ispixel()

devuelve true si el valor es un número en pixeles

```
ispixel(9001px); //true

ispixel(9001); //false
ispixel(true); //false
ispixel(#006600); //false
ispixel(9001em); //false
```

ispercentage()

devuelve true si el valor es un porcentaje

```
ispercentage(50%); //true

ispercentage(9001); //false
ispercentage(true); //false
ispercentage(#006600);
//false
ispercentage(9001em);
//false
```

isunit()

devuelve true si el valor es un número en las unidades especificadas

```
isunit(50px, px); //true
isunit(50px, "px"); //true
isunit(50%, %); //true

isunit(50px, %); //false
isunit(50px, em); //false
isunit(50, px); //false
```

validación con las funciones de tipo

```
// si el parámetro es un color
.mixin(@col) when (iscolor(@col)){
      background-color: @col;
}

@morao:  purple;
@doge:  "Says wowe";
//aplica el estilo, porque es un color
#el{
      .mixin(@morao);
}
//no aplica el estilo
#elem{
    .mixin(@doge);
}
```

if... else kinda

```
// si el parámetro es un color
.mixin(@col) when (iscolor(@col)){
      background-color: @col;
}
//si el parámetro no es un color
.mixin(@col) when (@default){
      background-color: gray;
}
@doge: "Says wowe";

//el fondo será gris
#elem{
      .mixin(@doge);
}
```

Práctica 10

Generar un html con un div y un input para que el usuario ingrese un valor.
Si el valor que se ingresó es un color cambia el fondo del div a ese color.
Si el valor es una medida, cambia el width del div a la medida.

Matemáticas... con colores???

Área I para Área IV

multiply()

multiplica dos colores dando como resultado un color más oscuro

```
multiply(#FF6600, CCC);
//Devuelve #CC5200
```

screen()

hace lo opuesto a multiply, resultando un color más claro

```
multiply(#FF6600, #CCC);
//Devuelve #FFE0CC
```

difference()

sustrae el segundo color del primer color

```
multiply(#FF6600, #CCC);
//Devuelve #3366CC
```

average()

obtiene el promedio de dos colores

```
average(#FF6600, #CCC);
//Devuelve #E69966
```

negation()

hace lo opuesto a difference(), resultando un color más claro
esto no quiere decir que los sume, hace lo opuesto no lo contrario

```
negation(#FF6600, #CCC);
//Devuelve #33CCCC
```

negation()

hace lo opuesto a difference(), resultando un color más claro
esto no quiere decir que los sume, hace lo opuesto no lo contrario

```
negation(#FF6600, #CCC);
//Devuelve #33CCCC
```

Práctica 11

Generar un html con un 7 divs. En un .less hay dos variables de color, @col1 y @col2.
Cada uno de los divs contendrá el color resultante de una operación. Los colores de fondo de los div serán:
1 div @col1
2 div @col2
3 div @col1*@col2
4 div screen(@col1,@col2)
5 div @col1-@col2
6 div average(@col1,@col2)
7 div negation(@col1,@col2)
Incluir dos inputs que permitan cambiar los valores de @col1 y @col2, validar los datos con las funciones de tipo