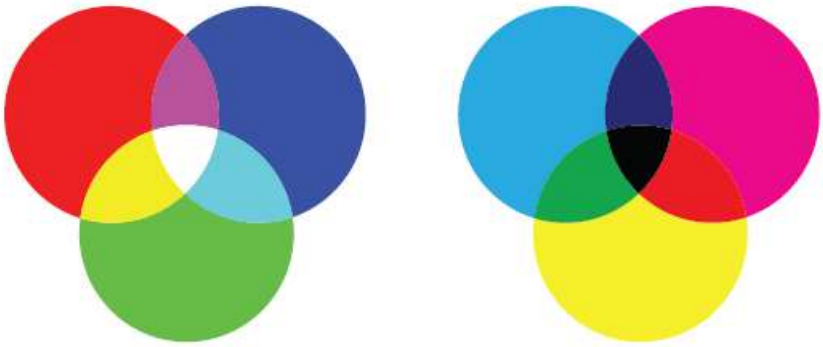
Unidad 9

Color: Color por Números

Elementos que se introducen en esta Unidad:
`color, color(), colorMode()`

Trabajar con colores a través de una pantalla es diferente a hacerlo en papel o lienzo. Mientras que se apliquen las mismas reglas, el conocimiento de los pigmentos para pintar (rojo cadmio, azul prusiano, ocre tostado) y para imprimir (cian, amarillo, magenta) no necesitan ser traducidos para crear colores digitales. Por ejemplo, unir todos los colores en un monitor de ordenador tiene como resultado el color blanco, mientras que unir todos los colores con pintura tiene como resultado el color negro (o un marrón extraño). Un monitor de ordenador mezcla los colores con luz. La pantalla es una superficie negra y se añade luz coloreada. Esto se conoce como colores aditivos, en contraste con el modelo de color sustractivo para tinta en papel y lienzo. Esta imagen expone la diferencia entre los dos modelos:



`Color aditivo` `Color sustractivo`

La forma más común de especificar un color en el ordenador es mediante valores RGB. Un valor RGB ajusta la cantidad de luz roja, verde y azul en un píxel de la pantalla. Si miras de cerca la pantalla de tu televisor o el monitor de tu ordenador, verás que cada píxel se compone de tres elementos independientes de luz roja, verde y azul; pero debido a que nuestros ojos sólo pueden ver una cantidad limitada de detalle, los tres colores se mezclan para formar uno solo. La intensidad de cada elemento de color está especificada con valores entre 0 y 255, donde 0 es el mínimo y 255 el máximo.

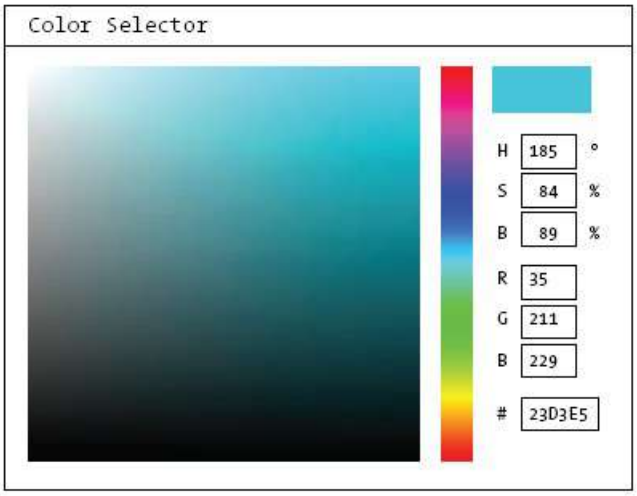
-Ajuste de color

En Processing, los colores están definidos por los parámetros de las funciones `background()`, `fill()` y `stroke()`:

```
background(valor1, valor2, valor3)
fill(valor1, valor2, valor3)
fill(valor1, value2, valor3, alfa)
stroke(valor1, valor2, valor3)
stroke(valor1, valor2, valor3, alfa)
```

Por defecto, el parámetro *value1* define el componente de color rojo, *valor2* el componente verde y *value3* el azul. El parámetro opcional alfa para `fill()` o `stroke()` define la opacidad. El valor 255 del parámetro *alfa* indica que el color es totalmente opaco, y el valor 0 indica que es totalmente transparente (no será visible).

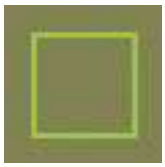
```
background(174, 221, 60);
```



Selector de color

Podemos seleccionar el color con el cursor simplemente introducir los valores que deseamos.

```
background(129, 130, 87);
noStroke();
fill(174, 221, 60);
rect(17, 17, 66, 66);
```



```
background(129, 130, 87);
noFill();
strokeWeight(4);
stroke(174, 221, 60);
rect(19, 19, 62, 62);
```

```
background(116, 193, 206);
noStroke();
fill(129, 130, 87, 102);      //Menos opacidad
rect(20, 20, 30, 60);
fill(129, 130, 87, 204);      //Más opacidad
rect(50, 20, 30, 60);
```

```
background(116, 193, 206);
int x = 0;
noStroke();
for (int i = 51; i <= 255; i += 51) {
      fill(129, 130, 87, i);
      rect(x, 20, 20, 60);
      x += 20;
}
```

```
background(56, 90, 94);
smooth();
strokeWeight(12);
stroke(242, 204, 47, 102);    //Menos opacidad
line(30, 20, 50, 80);
stroke(242, 204, 47, 204);    //Más opacidad
line(50, 20, 70, 80);
```

```
background(56, 90, 94);
smooth();
int x = 0;
strokeWeight(12);
```

```
for (int i = 51; i <= 255; i += 51) {
      stroke(242, 204, 47, i);
      line(x, 20, x+20, 80);
       x += 20;
}
```

La opacidad se puede usar para crear nuevos colores mediante superposición de formas. Los colores originados dependen del orden en que se dibujen las formas.

```
background(0);
noStroke();
smooth();
fill(242, 204, 47, 160);      //Amarillo
ellipse(47, 36, 64, 64);
fill(174, 221, 60, 160);      //Verde
ellipse(90, 47, 64, 64);
fill(116, 193, 206, 160);     //Azul
ellipse(57, 79, 64, 64);
```

```
background(255);
noStroke();
smooth();
fill(242, 204, 47, 160);      //Amarillo
ellipse(47, 36, 64, 64);
fill(174, 221, 60, 160);      //Verde
ellipse(90, 47, 64, 64);
fill(116, 193, 206, 160);     //Azul
ellipse(57, 79, 64, 64);
```

-Dato de color

El tipo dato de color se usa para almacenar colores en un programa, y la función `color()` se usa para asignar una variable de color. La función `color()` puede crear valores de gris, valores de gris con opacidad, valores de color y valores de color con opacidad. Las variables del tipo dato de color puede almacenar todas estas configuraciones:

```
color(gris)
color(gris, alfa)
color(valor1, valor2, valor3)
color(valor1, valor2, valor3, alfa)
```

Los parámetros de la función `color()` define un color. El parámetro *gris* usado sólo o con *alfa*, define un rango de tonos de blanco a negro. El parámetro *alfa* define la opacidad con valores de entre 0 (transparente) a 255 (opaco). Los parámetros *valor1*, *valor2* y *valor3* definen valores para los distintos componentes. Las variables del tipo de dato de color se definen y se asignan de la misma forma que los tipos de dato `int` y `float` discutidos en la unidad Datos: Variables.

```
color c1 = color(51);              //Crea gris.
color c2 = color(51, 204);         //Crea gris con opacidad.
color c3 = color(51, 102, 153);    //Crea azul.
color c4 = color(51, 102, 153, 51);//Crea azul con opacidad.
```

Después de que una variable *color* se haya definido, se puede usar como parámetro para las funciones `background()`, `fill()` y `stroke()`.

```
color ruby = color(211, 24, 24, 160);
color pink = color(237, 159, 176);
background(pink);
noStroke();
fill(ruby);
rect(35, 0, 20, 100);
```

RGB, HSB

Processing usa el formato de color RGB como predeterminado para trabajar, pero se puede usar en su lugar la especificación HSB para definir colores en función del matiz, saturación y brillo. El matiz de un color es lo que normalmente piensa la gente del nombre del color: amarillo, rojo, azul, naranja, verde, violeta. Un matiz puro es un color sin diluir muy intenso. La saturación es el grado de pureza en un color. Va desde el matiz sin diluir hasta el matiz completamente diluido, donde el color no tiene brillo. El brillo de un color es su relación de luz y oscuridad. La función `colorMode()` establece el formato de color para un programa:

```
colorMode(modo)
colorMode(modo, rango)
colorMode(modo, rango1, rango2, rango3)
```

Los parámetros de `colorMode()` cambian la forma en que Processing interpreta el dato de color. El parámetro *modo* puede ser tanto RGB como HSB. El rango de parámetros permite a Processing usar valores diferentes al predeterminado (0 a 255). En los gráficos de ordenador se usa frecuentemente un rango de valores entre 0.0 y 1.0. Tanto si un parámetro de un sólo rango establece el rango para todos los colores, como si los parámetros *rango1*, *rango2* y *rango3* establecen el rango de cada uno – tanto rojo, verde, azul como matiz, saturación, brillo, dependiendo del valor del parámetro *modo*.

```
//Establece el rango para los valores rojo, verde y azul de 0.0 a 1.0
colorMode(RGB, 1.0);
```

Un ajuste útil para el formato HSB es establecer los parámetros de *rango1*, *rango2* y *rango3* respectivamente a 360, 100 y 100. Los valores de matiz desde 0 a 360 son los grados alrededor de la rueda de color, y los valores de saturación y brillo desde 0 a 100 son porcentajes. Este ajuste coincide con los valores usados en muchos selectores de color y, por tanto, hace más sencilla la transición de dato de color entre otros programas y Processing:

```
//Establece el rango para el matiz con valores desde 0 a 360 y la
//saturación y el brillo con valores entre 0 y 100
colorMode(HSB, 360, 100, 100);
```

Los siguientes ejemplos revelan la diferencia entre matiz, saturación y brillo.

```
//Cambia el matiz; saturación y brillo constante
colorMode(HSB);
for (int i = 0; i < 100; i++) {
      stroke(i*2.5, 255, 255);
      line(i, 0, i, 100);
}
```

```
//Cambia la saturación; matiz y brillo constante
colorMode(HSB);
for (int i = 0; i < 100; i++) {
      stroke(132, i*2.5, 204);
      line(i, 0, i, 100);
}
```

```
//Cambia el brillo; matiz y saturación constante
colorMode(HSB);
for (int i = 0; i < 100; i++) {
      stroke(132, 108, i*2.5);
      line(i, 0, i, 100);
}
```

```
//Cambia la saturación y el brillo; matiz constante
colorMode(HSB);
for (int i = 0; i < 100; i++) {
      for (int j = 0; j < 100; j++) {
            stroke(132, j*2.5, i*2.5);
            point(i, j);
      }
}
```

Es fácil hacer transiciones suaves entre colores cambiando los valores para `color()`, `fill()` y `stroke()`. El formato HSB tiene una enorme ventaja sobre el formato RGB cuando trabajamos con códigos, porque es más intuitivo. Cambiando los valores de los componentes rojo, verde y azul, a menudo obtenemos resultados inesperados, mientras que estimando los resultados de los cambios del matiz, saturación y brillo seguimos un camino más lógico. Los siguientes ejemplos muestran la transición del verde al azul. El primer ejemplo hace la transición usando el formato RGB. Requiere calcular los tres valores de color, y la saturación del color cambia inesperadamente en el centro. El segundo ejemplo hace la transición usando el formato HSB. Sólo es necesario alterar un número, y el matiz cambia suavemente e independientemente de las otras propiedades del color.

```
//Cambio del azul al verde en formato RGB
colorMode(RGB);
for (int i = 0; i < 100; i++) {
      float r = 61 + (i*0.92);
      float g = 156 + (i*0.48);
      float b = 204 – (i*1.43);
      stroke(r, g, b);
      line(i, 0, i, 100);
}
```

```
// Cambio del azul al verde en formato HSB
colorMode(HSB, 360, 100, 100);
for (int i = 0; i < 100; i++) {
      float newHue = 200 – (i*1.2);
      stroke(newHue, 70, 80);
      line(i, 0, i, 100);
}
```

-Hexadecimal

La notación hexadecimal (hex) es una notación alternativa para definir el color. Este método es popular entre diseñadores que trabajan en la Web porque estándares como HyperText
Markup Language (HTML) y Cascading Style Sheets (CSS) usan esta notación. La notación Hex para el color codifica cada número desde el 0 al 255 en un valor de dos dígitos usando los números de 0 a 9 y las letras de A a F. De esta forma, tres valores RGB desde 0 a 255 se pueden escribir como un solo valor hex de seis dígitos. Unas pocas conversiones de ejemplo demuestran esta notación:

RGB	Hex
255, 255, 255	#FFFFFF
0, 0, 0	#000000

```
102, 153, 204                    #6699CC
195, 244, 59                     #C3F43B
116, 206, 206                    #74CECE
```

La conversión de los valores RGB a notación hex no es intuitiva. A menudo, el valor se toma del selector de color. Por ejemplo, puedes copiar y pegar un valor hex desde el selector de color de Processing en tu código. Cuando se usan valores de color codificados en notación hex, debemos colocar un # antes del valor para distinguirlo dentro del código.

```
// Código del tercer ejemplo reescrito usando números hex
background(#818257);
noStroke();
fill(#AEDD3C);
rect(17, 17, 66, 66);
```

-Valores de grises

Los ejemplos vistos anteriormente han usado el fondo por defecto de color gris claro, líneas negras, y figuras blancas. Para cambiar estos valores, es necesario introducir sintaxis adicional. La función `background()` establece el color de la ventana de representación con un número entre 0 y 255. Este rango puede ser incómodo si no estás familiarizado con programas de dibujo en el ordenador. El valor 255 es blanco y el valor 0 es negro, con un rango de valores de grises en medio. Si no se define un valor para el fondo, se usa el valor por defecto 204 (gris claro).
La función `fill()` define el valor del relleno de las figuras, y la función `stroke()` define el valor del contorno de las figuras dibujadas. Si no se define un valor de relleno, se usa el valor por defecto 255 (blanco). Si no se define un valor de contorno, se usa el valor por defecto 0 (negro).

```
rect(10, 10, 50, 50);
fill(204);
stroke(102);
rect(20, 20, 50, 50);
fill(153);
stroke(153);
rect(30, 30, 50, 50);
fill(102);
stroke(204);
rect(40, 40, 50, 50);
```

Cuando se ha definido un valor de relleno o contorno, se aplica a todas las figuras dibujadas después. Para cambiar el valor de relleno o contorno, usamos la función `fill()` o `stroke()` de nuevo.
Un parámetro opcional adicional para `fill()` o `stroke()` regula la transparencia. Definiendo el parámetro a 255 hace que la figura sea totalmente opaca, y a 0 totalmente transparente:

```
background(0);
fill(255, 220);
rect(15, 15, 50, 50);
rect(35, 35, 50, 50);
```

El relleno y el contorno de una figura se puede eliminar. La función `noFill()` detiene a Processing de rellenar figuras, y la función `noStroke()` detiene la creación de líneas y contornos de las figuras. Si usamos `noFill()` y `noStroke()` no dibujaremos nada en la pantalla.

-Atributos de dibujo

Además de cambiar los valores de relleno y contorno de las figuras, también es posible cambiar atributos de

la geometría. Las funciones `smooth()` y `noSmooth()` activan y desactivan el suavizado (conocido como filtro *antialiasing*). Cuando usamos una de estas funciones, afectará a todas las funciones dibujadas después. Si usamos primero `smooth()`, usar `noSmooth()` cancelará el ajuste, y viceversa.

```
smooth();
ellipse(30, 48, 36, 36);
noSmooth();
ellipse(70, 48, 36, 36);
```

Los atributos de la línea, están controlados por las funciones `strokeWeight()`, `strokeCap()` y `strokeJoin()`. La función `strokeWeight()` tiene un parámetro numérico que define el grosor de todas las líneas dibujadas después de usar esta función. La función `strokeCap()` requiere un parámetro que puede ser `ROUND`, `SQUARE` o `PROJECT`. `ROUND` redondea los puntos finales, y `SQUARE` los cuadra. `PROJECT` es una mezcla de ambos: redondea las esquinas cuadradas suavemente. Esta función se usa en líneas. La función `strokeJoin()` tiene un parámetro que puede ser `BEVEL`, `MITTER` o `ROUND`. Estos parámetros determinan la forma del contorno de la figura. `BEVEL` corta en diagonal las esquinas cuadradas, `MITTER` cuadra las esquinas (es el valor por defecto) y `ROUND` redondea las esquinas.

```
smooth();
strokeWeight(12);
strokeCap(ROUND);
line(20, 30, 80, 30);   //Línea superior
strokeCap(SQUARE);
line(20, 50, 80, 50);   //Línea central
strokeCap(PROJECT);
line(20, 70, 80, 70);   //Línea inferior

smooth();
strokeWeight(12);
strokeJoin(BEVEL);
rect(12, 33, 15, 33);   //Figura izquierda
strokeJoin(MITER);
rect(42, 33, 15, 33);   //Figura central
strokeJoin(ROUND);
rect(72, 33, 15, 33);   //Figura derecha
```

Modos de dibujo

Por defecto, los parámetros para `ellipse()` define la coordenada x del centro, la coordenada y del centro, la anchura y la altura. La función `ellipseMode()` cambia la forma en que se usan estos parámetros para dibujar elipses. La función `ellipseMode()` requiere un parámetro que puede ser `CENTER`, `RADIUS`, `CORNER` o `CORNERS`. El modo por defecto es `CENTER`. El modo `RADIUS` también usa el primer y segundo parámetro de `ellipse()` para establecer el centro, pero el tercer parámetro deber ser la mitad del ancho, y el cuarto parámetro debe ser la mitad del alto. El modo `CORNER` hace que `ellipse()` funcione de manera parecida a `rect()`. Causa que el primer y segundo parámetro se trasladen a la esquina superior izquierda del rectángulo que circunscribe la elipse y usa el tercer y cuarto parámetro para definir la anchura y la altura. El modo `CORNERS` tiene un efecto similar a `CORNER`, pero causa que el tercer y cuarto parámetro de `ellipse()` se definan en la esquina inferior derecha del rectángulo.

```
smooth();
noStroke();
ellipseMode(RADIUS);
fill(126);
ellipse(33, 33, 60, 60);      //Elipse gris
fill(255);
```

```
ellipseMode(CORNER);
ellipse(33, 33, 60, 60);      //Elipse blanca
fill(0);
ellipseMode(CORNERS);
ellipse(33, 33, 60, 60);      //Elipse negra
```

Con un estilo similar, la función `rectMode()` afecta a cómo se dibujan los rectángulos. Requiere un parámetro que puede ser `CORNER`, `CORNERS` o `CENTER`. El modo por defecto es `CORNER`, y `CORNERS` causa que el tercer y cuarto parámetro de `rect()` se definan en la esquina contraria a la primera. El modo `CENTER` causa que el primer y segundo parámetro de `rect()` definan el centro del rectángulo y usa el tercer y cuarto parámetro para la anchura y la altura.

```
noStroke();
rectMode(CORNER);
fill(126);
rect(40, 40, 60, 60);         //Rectángulo gris
rectMode(CENTER);
fill(255);
rect(40, 40, 60, 60);         //Rectángulo blanco
rectMode(CORNERS);
fill(0);
rect(40, 40, 60, 60);         //Rectángulo negro
```