

Estructuras: Continuidad

Elementos que se introducen en esta Unidad:

`draw()`, `frameRate()`, `frameCount`, `setup()`, `noLoop()`

Todos los programas, mostrados en las unidades anteriores, ejecutan el código y, posteriormente, lo detienen. Los programas con animaciones o que modifican su contenido en vivo, requieren estar ejecutándose continuamente. Dichos programas, se ejecutan de manera continua y permiten, por lo tanto, animaciones o bien conseguir datos a través de dispositivos de entrada.

-Evaluación Continua

Un programa que estará ejecutándose continuamente requiere de la función `draw()`. El código dentro de un bloque `draw()` es ejecutado en un continuo loop (repetición) hasta que se detenga de ejecutar el programa o se cierre la ventana de representación. Un programa puede tener **solo un** bloque `draw()`. Cada vez que el código dentro del bloque `draw()` es terminado de leer, lo muestra en la ventana de representación y vuelve a leer desde la primer línea (siempre dentro del bloque `draw()`).

Por defecto, las imágenes son dibujadas a 60 cuadros por segundo (fps). La función `frameRate()` nos permite cambiar la cantidad de esos cuadros por segundo. El programa se ejecutará a la velocidad que se le dé al `frameRate()`, aún así, muchos programadores ambiciosos suelen exceder la cantidad de cuadros por segundos en relación al poder de su ordenador. Se recomienda no exceder los 60 cuadros que ya vienen por defecto.

La variable `frameCount` siempre contiene el número de cuadros que se **está ejecutando** desde que el programa fue iniciado. Un programa que posee un bloque `draw()` siempre variará el número de cuadros (1, 2, 3, 4, 5, 6...) hasta que el programa sea detenido, la computadora agote su capacidad de procesamiento o se pierda la fuente de tensión que la alimenta.

//Imprime cada número de cuadro en la consola

```
void draw() {
    println(frameCount);
}
```

//Se ejecuta a 4 cuadros por segundo, imprime cada cuadro en la consola

```
void draw() {
    frameRate(4);
    println(frameCount);
}
```

Con cambiar algún parámetro dentro de la estructura `draw()`, produciremos un simple efecto de movimiento. Por ejemplo, a través de una variable, cambiar la posición en la que se encuentra una línea.



```
float y = 0.0;
void draw() {
    frameRate(30);
    line(0, y, 100, y);
    y = y + 0.5;
}
```

Cuando este código es ejecutado, las variables son reemplazadas con los valores actuales y vuelve a correr las acciones en este orden:

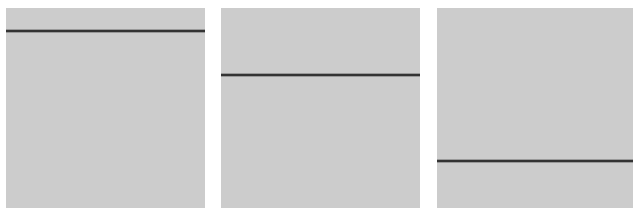
```

float y = 0.0
frameRate(30)
----- Entra al draw()
line(0, 0.0, 100, 0.0)
y = 0.5
frameRate(30)
----- Entra al draw() por
segunda vez
line(0, 0.5, 100, 0.5)
y = 1.0
frameRate(30)
----- Entra al draw() por
tercera vez
line(0, 1.0, 100, 1.0)
y = 1.5
Etc...

```

La variable de control mostrada anteriormente (llamada *y*) debe ser declarada fuera del bloque `draw()`. De otro modo, cada vez que el bloque se actualice volverá a crear la variable, es decir, es como si es valor se *resteará*.

Como se puede comprobar anteriormente, el fondo del programa no se actualiza automáticamente con cada vuelta del bloque `draw()`. Esto produce un efecto de barrido. Sin embargo, ¿qué pasa si lo que queremos es solo una línea moviéndose, en lugar de una especie de barrido?. Recurrirémos a declararlo como **una de las primeras líneas** dentro del `draw()` y con un color en específico. La solución se piensa como si se estuviese dibujando cada cuadro a mano. Si queremos que la posición antigua de una figura desaparezca, volvemos a pintar el fondo, este pintará la figura y, posteriormente, se dibujará la figura en la posición actualizada.




```

float y = 0.0;
void draw() {
    frameRate(30);
    background(204);
    y = y + 0.5;
    line(0, y, 100, y);
}

```

La variable que controla la posición de la línea puede ser usada para otros propósitos. Por ejemplo, alterar el color del fondo:



```

float y = 0.0;
void draw() {
    frameRate(30);
    background(y * 2.5);
    y = y + 0.5;
    line(0, y, 100, y);
}

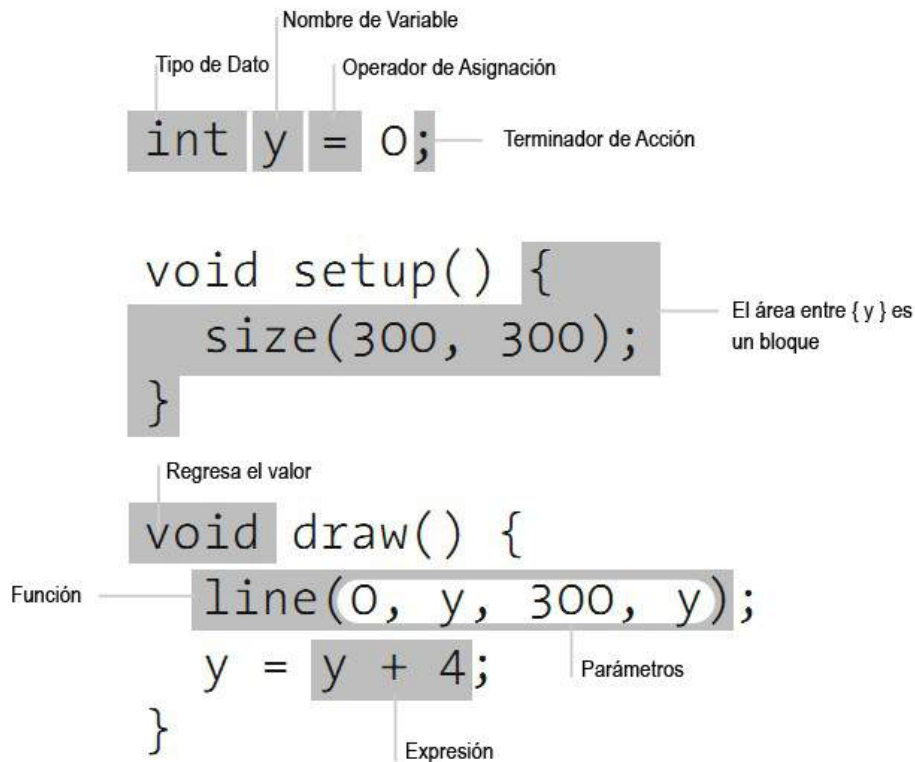
```

Pasado unos pocos segundos del programa, la línea desaparece por debajo. Con una simple estructura `IF` solucionamos ese problema:



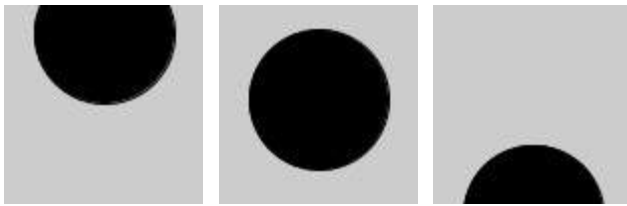
```
float y = 0.0;
void draw() {
    frameRate(30);
    background(y * 2.5);
    y = y + 0.5;
    line(0, y, 100, y);

    if (y > 100) {
        y = 0;
    }
}
```



-Controlando el Flujo

Hay funciones que, incluso en programas dinámicos, solo requieren ser ejecutadas una vez. El bloque `setup()` permite que cualquier función que se escriba dentro de él se ejecute solo una vez. Muy útil para funciones como `size()` o `loadImage()`, que suelen ser ejecutadas únicamente la primera vez. El bloque `setup()` se escribe siempre antes que el bloque `draw()` y, al igual que con el bloque `draw()`, puede haber **solo un** bloque `setup()` por programa. Cuando un programa es ejecutado, primero se lee lo que está fuera de los bloques `setup()` y `draw()`, luego se lee todo lo que está dentro del `setup()` y, finalmente, lo que está dentro del `draw()`. En el siguiente ejemplo, el tamaño de la ventana, el filtro de suavizado y el relleno no cambiarán, así que se incluirán en el bloque `setup()`.



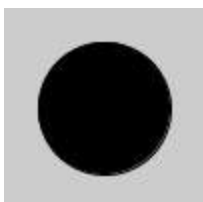
```
float y = 0.0;
void setup() {
  size(100, 100);
  smooth();
  fill(0);
}

void draw() {
  background(204);
  ellipse(50, y, 70, 70);
  y += 0.5;
  if (y > 150) {
    y = -50.0;
  }
}
```

Cuando este código es ejecutado, las variables son reemplazadas con los valores actuales y vuelve a correr las acciones en este orden:

```
float y = 0.0
size(100, 100)
----- Entra al setup()
smooth()
fill(0)
background(204)
----- Entra al draw()
ellipse(50, 0.0, 70, 70)
y = 0.5
background(204)
----- Entra al draw() por segunda vez
ellipse(50, 0.5, 70, 70)
y = 1.0
background(204)
----- Entra al draw() por tercera vez
ellipse(50, 1.0, 70, 70)
y = 1.5
Etc....
```

Cuando el valor de `y` es mayor a 150, el código en la estructura IF convierte el valor a -50. La variable que cambia con cada repetición del bloque `draw()` debe ser declarada **fuera** de los bloques `setup()` y `draw()`. Si está es declarada dentro del `draw()`, se estará re-creando con cada repetición del mismo, por lo tanto su valor nunca cambiaría. Por otro lado, fuera de los bloques `setup()` y `draw()` solo pueden declararse y asignarse variables. Si se declarara una función, causará un error. Si un programa solo dibujará un cuadro, este puede declararse en el `setup()`. La única diferencia entre `setup()` y `draw()` es que el `setup()` se ejecuta **solo una vez**.



```
void setup() {
  size(100, 100);
  smooth();
  fill(0);
  ellipse(50, 50, 66, 66);
}
```

Utilizando la función `noLoop()`, podemos hacer que el `draw()` deje de repetir y simplemente dibuje un cuadro. El siguiente ejemplo es similar al anterior, con la diferencia que la elipse es dibujada en el `draw()`:



```
void setup() {
    size(100, 100);
    smooth();
    fill(0);
    noLoop();
}
void draw() {
    ellipse(50, 50, 66, 66);
}
```

-Extensión de la Variable

Cuando trabajamos con los bloques `setup()` y `draw()` es necesario tomar consciencia de donde declaramos y asignamos cada variable. La localización de la variable determina su *extensión*. La regla para conocer la extensión que posee una variable es muy simple: si la variable está dentro de una estructura, esa variable es local y puede usarse solamente dentro de esa estructura; si la variable está en el programa, fuera de cualquier estructura, esa variable es global y puede usarse en cualquier estructura. Las variables declaradas en el `setup()` pueden usarse **solo** en el `setup()`. Las variables declaradas en el `draw()` pueden usarse **solo** en el `draw()`.

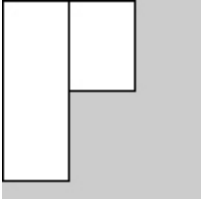
```
int d = 51; //La variable d es global, puede ser usada donde sea
void setup() {
    size(100, 100);
    int val = d * 2; //La variable local val, puede ser usada solo en el
    fill(val); //setup()
}
void draw() {
    int y = 60; //La variable local y, puede ser usada solo en el draw()
    line(0, y, d, y);
    y -= 25;
    line(0, y, d, y);
}
```

Cuando una variable es creada dentro de un bloque, al salir del bloque, esa variable es **destruida**. Esto significa que no puede utilizarse fuera de ese bloque.

```
void draw() {
    int d = 80; //Esta variable puede usarse donde sea en el draw()
    if (d > 50) {
        int x = 10; //Esta variable puede usarse solo en este bloque IF
        line(x, 40, x+d, 40);
    }
    line(0, 50, d, 50);
    line(x, 60, x+d, 60); //ERROR! No se puede leer esta variable fuera del
    //bloque
}

void draw() {
    for (int y = 20; y < 80; y += 6) { //La variable puede usarse
        line(20, y, 50, y); //solo en el bloque FOR
    }
    line(y, 0, y, 100); //ERROR! No se puede acceder a y fuera del FOR
}
```

La implementación de variables locales y globales permite que se pueda tener dos o más variables con el mismo nombre. Aún así, no se recomienda el trabajo con variables del mismo nombre ya que puede producir confusión en quien programa.



```
int d = 45;           //Asigna 45 a la variable d
void setup() {
  size(100, 100);
  int d = 90;         //Asigna 90 a la variable local d
  rect(0, 0, 33, d);  //Usa la variable local d con
                      //valor 90
}
void draw() {
  rect(33, 0, 33, d); //Usa d con valor de 45
}
```

Una variable dentro de un bloque con el mismo nombre que una variable fuera del bloque es, comúnmente, un error muy difícil de encontrar.