LAS VARIABLES

La variable es un elemento frecuentemente utilizado en la computación y, en general, en todos los lenguajes de programación. El concepto básico de la variable se relaciona con la capacidad de poder almacenar un valor dentro del programa. Este valor podrá utilizarse y modificarse en la ejecución de este.

Una variable está formada por un espacio físico en la memoria del ordenador, su valor, y un nombre simbólico, comúnmente llamado identificador, que está asociado a dicho espacio. El valor y el identificador de la variable permiten que el identificador sea usado independientemente de la información exacta que representa. El identificador, en el programa, puede estar ligado a un valor durante el tiempo de ejecución y el valor de la variable puede por lo tanto cambiar durante el curso de la ejecución del programa.

El nombre del identificador será proporcionado por el programador de manera libre y deberá ser lo suficientemente descriptivo para poder entender qué es lo que se almacenará. La única restricción es la utilización de alguna palabra reservada por Processing.

El valor que se almacenará en el identificador tiene que ser definido, en otras palabras, debemos de seleccionar qué tipo de dato a utilizar.

LOS TIPOS DE DATOS EN PROCESSING

El tipo de dato es un atributo que indica al ordenador (y/o al programador) los rangos de valores posibles que se le puede asignar a los datos que se van a procesar. En el caso de las variables, indica qué tipo de información podemos almacenar en el identificador y qué operaciones podemos realizar.

Processing permite utilizar dos tipos de datos: los primitivos y los tipos referencia. En este apartado estudiaremos los primeros, mientras que los tipos referencia se reservarán para más adelante. Los primitivos reservan espacio de memoria y por lo tanto definen explícitamente el rango de valores que puede adoptar una variable definida con ese tipo de datos. La ventaja de esto es que el lenguaje se independiza de la plataforma donde se ejecute y por lo tanto el rango de valores se mantendrá independientemente de si se ejecuta en Windows, Linux, etc. Como desventaja, como reserva memoria, es posible que se esté desperdiciando este recurso si la variable no es utilizada de manera inmediata a su declaración.

A continuación, se indican los primitivos que se pueden usar:

- int: El tipo de dato int representa un conjunto de enteros de 32 bits, así como las operaciones que se pueden realizar con los enteros, como la suma, la resta y la multiplicación. Su rango va del -2.147.483.648 al 2.147.483.647.
- float: El tipo de dato float se refiere a la utilización de números con punto decimal. El rango de valores va del -3.40282347E+38 al 3.40282347E+38 y está almacenado en 32 bits.
- char: El tipo de dato char permite almacenar cualquier tipo de carácter (letras o símbolos) en el formato Unicode.
- boolean: El tipo de dato boolean permite únicamente el almacenamiento de dos valores: TRUE y FALSE. Este tipo de dato es muy útil para el determinar el flujo de los programas.
- color: El tipo de dato color permite el almacenamiento de cualquier color codificado en números hexadecimales.

DECLARACIÓN E INICIALIZACIÓN

Processing es un lenguaje fuertemente tipado, esto significa que al momento de declarar una variable es requisito fundamental indicar su tipo de datos. La declaración de las variables se realiza de la siguiente manera:

```
int variable1;       // para una variable entera
float diametro;    // para un variable flotante
char letra;           // para una variable carácter
boolean valor;      // para una variable booleana
color arcoiris;  // para una variable color
```

- Si las variables van a ser utilizadas en todo el programa (**variable global** a todo el programa), se deberán declarar al inicio del programa, antes de la función ***setup()***, y serán globales en todo el ámbito del programa.
- Si las variables se utilizan únicamente en alguna función, entonces deberán declararse al inicio de la función deseada y se denominarán **variables locales**. Estas variables únicamente existirán cuando la función esté siendo utilizada.
- Si ud define una variable dentro de un bloque de código (los bloques de código se expresan mediante las llaves {}. Es decir, todo lo que se escriba entre esa llaves es un bloque de código) entonces la variable existirán únicamente dentro de ese bloque.

La "vida" de la variable definida en términos de los párrafos indicados anteriormente se denomina **ámbito** de la variable.

Las variables deben de tomar su valor mediante una asignación del valor. El símbolo para realizar esta asignación es el =. La asignación se realiza modificando el valor de la variable por el valor o por el resultado de la parte derecha del símbolo =.

De este modo, algunas asignaciones comunes son las siguientes:

```
int count =0;              //Asignamos el valor 0 a la variable count
char letter = 'a';         //Asignamos el valor 'a' a la variable letter
float d= 132.32;           //Asignamos el valor 132.32 a la variable d
float x = 4.0;             //Asignamos el valor 4.0 a la variable x
float y = x + 5.2;         //Asignamos el valor 9.2 a la variable y
float z = x*y + 15.0;      //Asignamos el valor 51.8 a la variable z
```

Es importante hacer notar que, por ejemplo, en algunos casos los valores de las variables se adquieren mediante el resultado de una operación realizada en la parte derecha del símbolo =. Eso pasa, por ejemplo, en la asignación del valor de la variable z.

Una vez asignado un valor en la variable, entonces esta se encuentra lista para ser utilizada dentro del ámbito donde ha sido definida. La idea de la variable es generar un dinamismo en el programa y que no sea completamente estático. Para lograrlo, la variable tendrá que ser utilizada en el programa, sustituyendo normalmente algún valor estático utilizado.

A continuación, en la figura 1 se puede observar un ejemplo del uso de una variable. En este ejemplo podemos identificar tres características importantes. La primera es la declaración de la variable, la cual se encuentra en la parte superior del programa. La segunda es el uso de la variable, donde se sustituye un valor estático de la función ***ellipse()***, por esa variable, en este caso, la posición x de la elipse. La tercera es el uso de la variable, que en este caso es incrementada en 1. Estas características permiten que la elipse tenga un desplazamiento constante hacia la derecha.

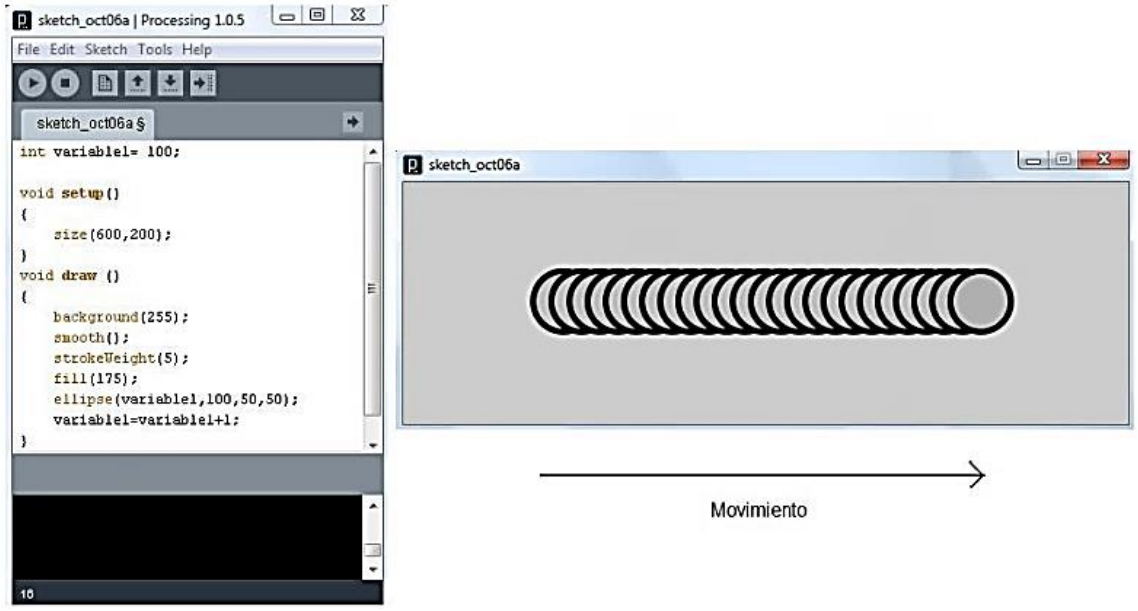


Figura 1. Ejemplo de uso de variable

Así, se deduce que esta variable puede ser muy útil para generar cambios en el comportamiento del programa. Podríamos utilizar la variable para sustituirla por cualquier valor estático (argumento) que se encuentre dentro de la función ***draw()***, lo que generaría cambios en el comportamiento del programa. Se pueden generar tantas variables como sean necesarias, y actualizarlas según los requerimientos del programa.

LAS VARIABLES PREDEFINIDAS (O DEL SISTEMA)

Existen un conjunto de variables que están predefinidas por Processing. Estas variables tienen la ventaja de que no necesitan ser declaradas, simplemente son utilizadas directamente por los programas. Todas estas variables predefinidas son de mucha utilidad para la ejecución y el seguimiento de los programas. Las más importantes son:

- width: Esta variable almacena el ancho de la ventana de trabajo y se inicializa utilizando el primer parámetro de la función *size*().
- height: Esta variable almacena el alto de la ventana de trabajo y se inicializa por el segundo parámetro definido en la función *size*().
- frameRate: En la variable frameRate se guarda la velocidad con la que se está ejecutando el programa o sketch. Se puede definir la velocidad de ejecución mediante la función *frameRate*();
- frameCount: La variable frameCount contiene el número de frames que el programa ha ejecutado desde su inicio. Esta variable se inicializa con el valor de cero y al ingresar a la función *draw*() obtiene el valor de 1, y a partir de ese momento aumenta constantemente.
- displayHeight: Esta variable almacena el alto de la pantalla completa a ser desplegada. Es utilizada para crear una ventana, independientemente del dispositivo de ejecución.
- displayWidth: Esta variable almacena el ancho de la pantalla completa a ser desplegada. Funciona de igual forma que displayHeight pero para el ancho.
- key: La variable del sistema key contiene el valor reciente de la tecla oprimida desde el teclado.
- keyCode: Esta variable es utilizada para detectar teclas especiales como las flechas
- UP, DOWN, LEFT o RIGHT, o teclas como ALT, CONTROL, o SHIFT.

- keyPressed: Esta variable del sistema es de tipo booleano, por lo que tendrá el valor de TRUE si alguna tecla es oprimida y FALSE para el caso contrario.
- mouseX: Esta variable siempre conserva la coordenada horizontal del mouse. Esta información es únicamente utilizada cuando el mouse está dentro de la ventana de trabajo. Inicialmente el valor está definido en 0.
- mouseY: La variable mouseY funciona exactamente de la misma forma que mouseX, solo que almacena la coordenada y la posición actual del mouse.
- pmouseX: La variable pmouseX contiene la posición horizontal del mouse en el frame anterior al actual. En otras palabras, almacena la posición anterior del mouse.
- pmouseY: Al igual que la variable pmouseX, esta variable almacena la posición vertical del mouse en el frame anterior al actual.
- mousePressed: Esta variable booleana se encarga de almacenar si el botón del mouse está actualmente oprimido o no. El valor de TRUE es cuando el botón está siendo oprimido y el valor de FALSE cuando se libera el botón.
- mouseButton: La variable mouseButton detecta qué botón del mouse fue seleccionado. Adquiere entonces los valores de LEFT, RIGHT o CENTER, dependiendo del botón seleccionado. Inicialmente tiene el valor de 0.

En la figura 2 se pueden observar el efecto del uso de estas variables.

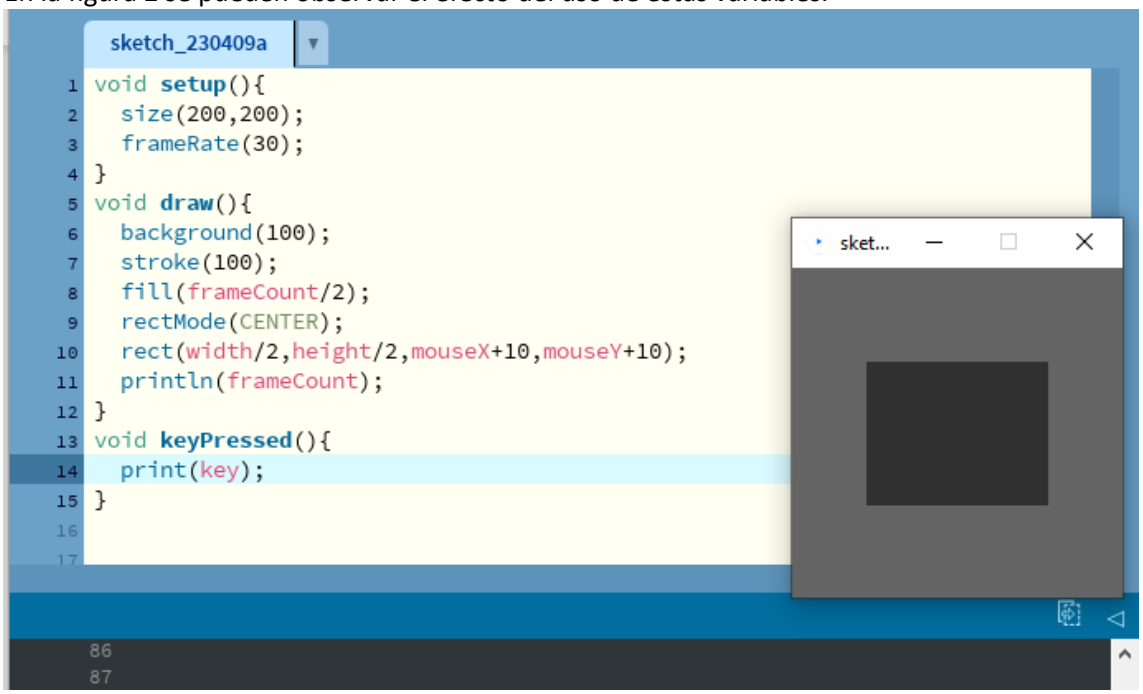


Figura 2. Ejemplo de uso de variables del sistema.

En este ejemplo se crea un programa conformado por tres funciones: setup(), draw() y keyPressed(). El primero se ejecuta primero de forma automática por única vez, y tiene por objetivo establecer la configuración inicial del programa. En este caso se define el tamaño de la ventana y luego se establece la cantidad de fotogramas que se ejecutarán por segundo.

El método draw() es un bucle infinito, es decir se irá ejecutando constantemente hasta que el programa finalice. Establece el color del fondo, el grosor de la línea de la figura que se va a dibujar, y el color de relleno de esa figura, que en este caso es un rectángulo. Gracias a las variables tanto el color de relleno, como el ancho y el alto de este rectángulo cambia constantemente.

Por último la función keyPressed() se encarga de detectar que si se ha presionado una tecla e imprime su valor en la pantalla. En la imagen se puede observar que no se ha presionado ninguna tecla, ya que el valor que aparece indica la cantidad de frames que se han ejecutado.

LAS ESTRUCTURAS DE CONTROL CONDICIONALES

La estructura condicional compara una variable contra otro(s) valor(es), para que, con base en el resultado de esta comparación, se determine un curso de acción dentro del programa. Cabe mencionar que la comparación se puede hacer contra otra variable o contra una constante, según se necesite.
Estas comparaciones pueden utilizar los operadores relacionales para verificar si el resultado cumple o no la condición, por lo cual se dice que la condición es verdadera o falsa. Los operadores que podemos utilizar en Processing son los siguientes:

```
   > Mayor que
   < Menor que
>= Mayor o igual
<= Menor o igual
   == Igualdad
   ! Diferente
```

Los tipos de condicionales

- **Simples**: Las estructuras condicionales simples son conocidas como tomas de decisión. Estas tomas de decisión lo único que hacen es verificar una única comparación. Si el resultado es verdadero, entonces se ejecutará el conjunto de instrucciones que esté dentro del bloque. Si el resultado es falso, simplemente se ignora todo el bloque. El flujo del programa continúa después de hacer esta simple comparación. La estructura simple es:

```
if (expresión booleana){
// Este conjunto de instrucciones se ejecutan
            //si la condición es verdadera
      }
```

  Resulta importante poder seguir algunas reglas de estilo. En el ejemplo anterior, podemos observar que existe una identación importante que denota el conjunto de instrucciones que están dentro de la estructura del if. Otro elemento que podemos observar es la utilización del doble slash (//). Este slash le indica a Processing que todo lo que esté posterior a él (en la misma línea) debe de ser ignorado; es decir es un comentario interno en el programa que ayuda a su legibilidad. Del mismo modo, podemos utilizar el (/*) y el (*/) para definir todo un bloque que queramos sea ignorado por Processing. En la figura 3 se muestra un ejemplo muy simple acerca de la utilización de la estructura condicional simple.
  En este ejemplo, dentro del programa solo hay un `if`. El programa verificará el valor de la variable y lo comparará con el valor `width`/2 (que corresponde al centro de la pantalla). Cuando el valor de `variable` es menor (en este caso, su valor inicial es 0) a la mitad de la pantalla, el resultado de la evaluación es `falso` y no se ejecutarán las instrucciones dentro del `if`. Cuando el valor de `variable` es mayor a la mitad del ancho de la pantalla, entonces automáticamente se ejecutarán las instrucciones que están dentro del `if`, en este caso el *background*(0), lo que generará que el fondo de la pantalla adquiera el color negro. Observe que para auentar el valor de la variable se utiliza un operador de incremento, esto quiere decir que Processing lo posee.

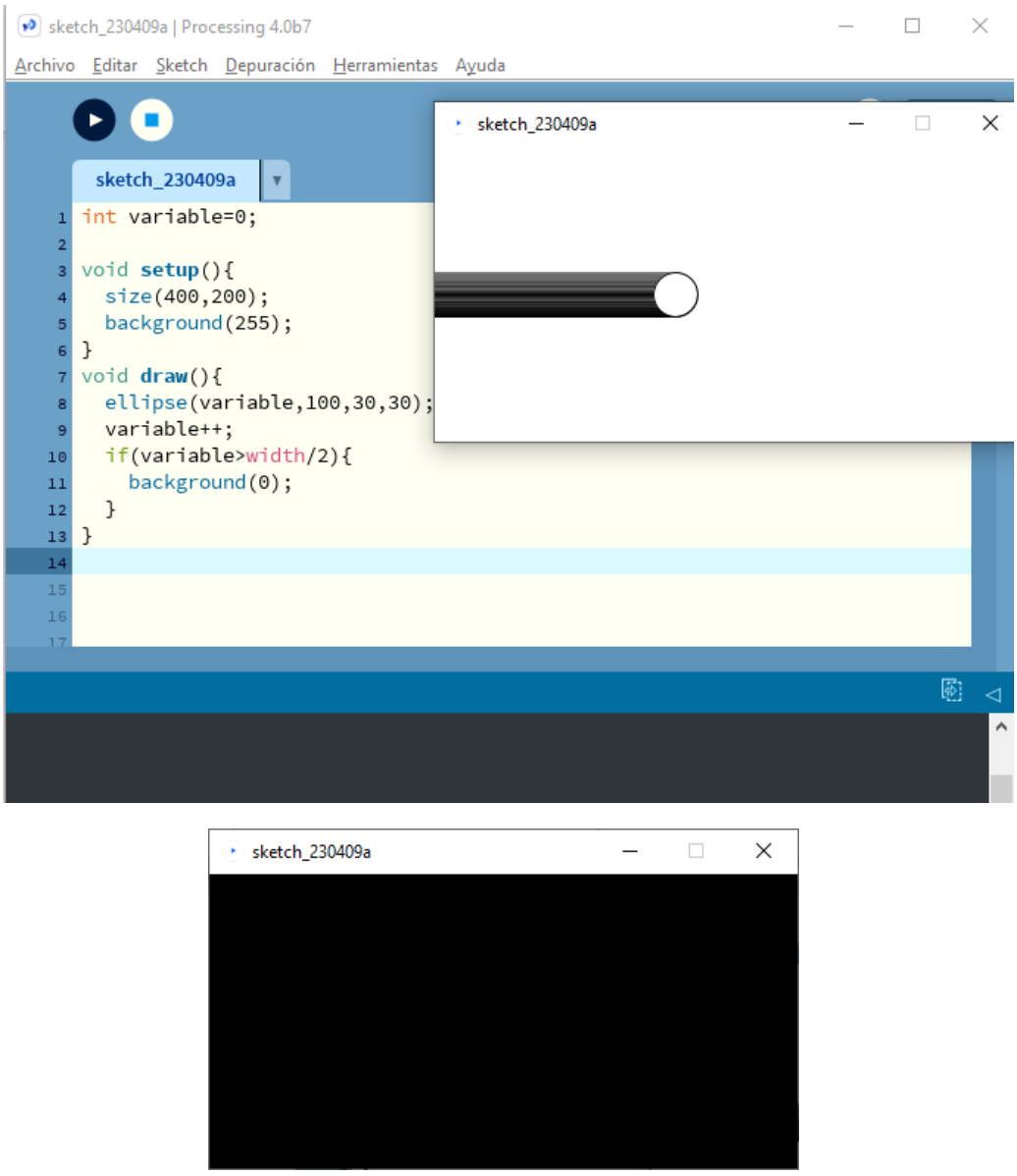
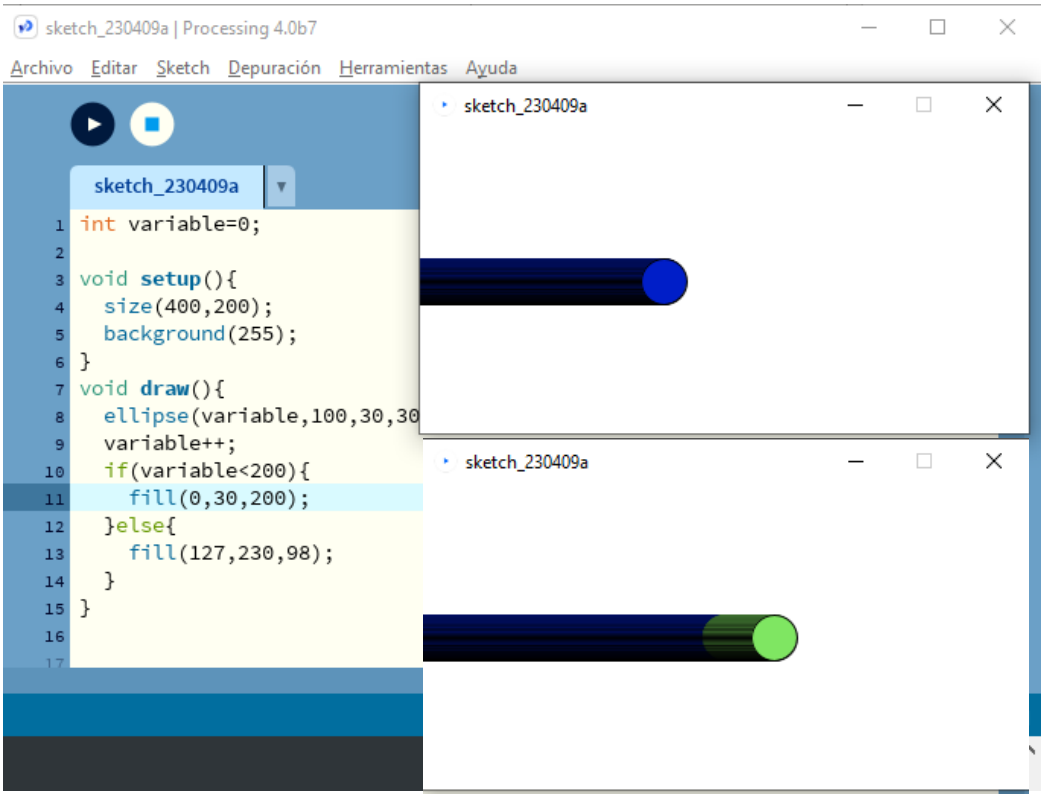


Figura 3. Ejemplo de programa usando if

- **Dobles**: Las estructuras condicionales dobles permiten elegir entre dos opciones o alternativas posibles en función del cumplimiento o no de una determinada condición. Si el resultado de la expresión lógica es verdadero, entonces ejecutará un conjunto de instrucciones, y si el resultado es falso, entonces ejecutará otro conjunto diferente de instrucciones. Finalizado cualquiera de los caminos alternativos, se continuará con la ejecución del programa.

```
if (expresión booleana){
/* El código que se ejecuta si la condición es verdadera*/
} else {
// El código que se ejecuta si la condición es falsa
}
```

A continuación, se muestra un ejemplo de su funcionamiento.



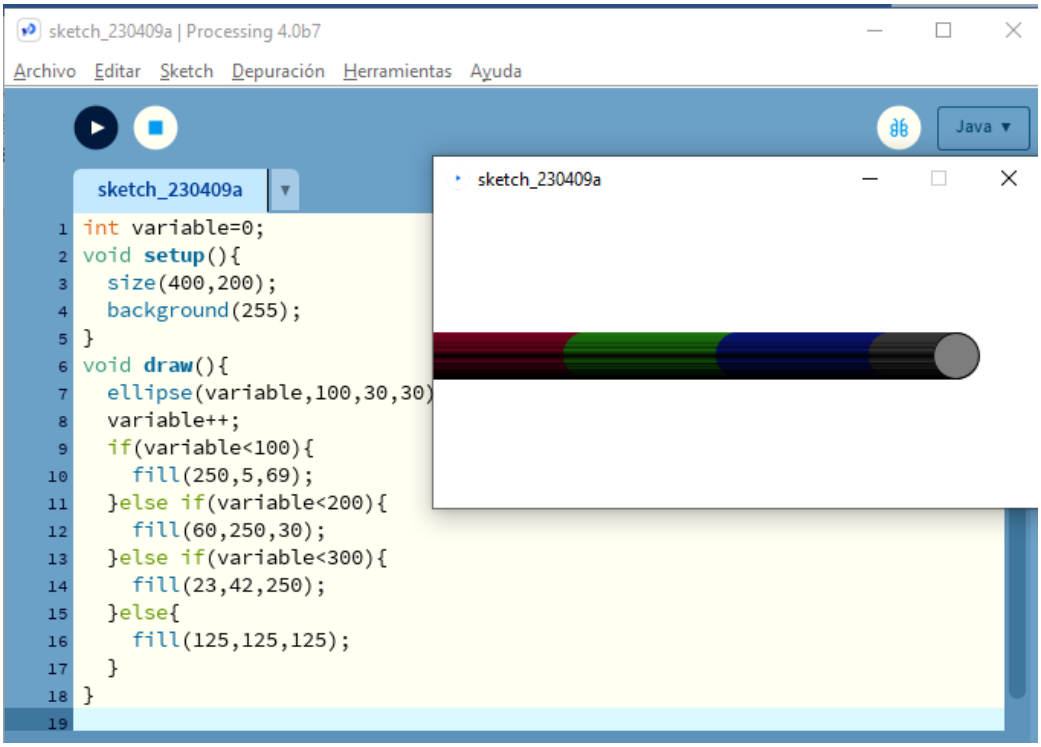
En este ejemplo, cuando la variable `variable` sea menor a 200, entonces la pelota se pintará de color azul, pero cuando la variable `variable` obtenga el valor de 200, entonces se pintará automáticamente de color verde (lo cual se puede ver en la imagen de la ventana superpuesta). Esta estructura está manejada por la estructura `if` y su correspondiente `else`.

- **Múltiples anidadas**: Las estructuras condicionales múltiples son tomas de decisión especializadas que permiten evaluar el resultado de una expresión contra distintos posibles resultados, ejecutando para cada caso una serie de instrucciones específicas. La forma común es la siguiente:

```
if (expresión booleana # 1){
    // El código que se ejecuta si la condición # 1 es verdadera
} else if (expresión booleana # 2){
    // El código que se ejecuta si la condición # 2 es verdadera
} else if (expresión booleana # n){
    // El código que se ejecuta si la condición # n es verdadera
} else {
    // Ejecución si ninguna de las condiciones anteriores fueron
    // verdaderas
}
```

Para evaluar múltiples condiciones, utilizamos el `else if`. Las condicionales son evaluadas en el orden presentadas. Cuando una expresión es verdadera, se ejecuta el código de manera inmediata y todo lo demás es ignorado.
Esta estructura es muy útil cuando tenemos múltiples opciones y lo que se debe de evaluar es la misma variable. Podemos incluir cuantos `else if` queramos. La ventaja es que cuando se cumpla la condición, automáticamente se ignorarán todas las comparaciones subsecuentes.

A continuación, se muestra un ejemplo donde la pelota cambia de color al verificar la variable `var1` en los puntos 100, 200 y 300, todo utilizando la estructura `else if`. Se puede comprobar que la pelota ha cambiado de color por el color de la estela que deja al avanzar hacia la derecha.



LOS OPERADORES LÓGICOS

Los operadores lógicos permiten hacer una evaluación unificada, dado un conjunto de expresiones lógicas. Los operadores lógicos que ofrece Processing son:

```
||   (O lógico)
&&   (Y lógico)
   !  (NOT lógico)
```

- El `&&`: La conjunción es un operador que opera sobre dos valores de verdad. Típicamente los valores de verdad de dos condiciones, devolviendo el valor de verdad verdadero cuando ambas proposiciones son verdaderas, y falso en cualquier otro caso. Es decir que es verdadera cuando ambas son verdaderas. La tabla de verdad de la conjunción es la siguiente:

```
V && V -> V
V && F -> F
F && V -> F
F && F -> F
```

- El `||`: La disyunción es un operador que funciona sobre dos valores de verdad, típicamente los valores de verdad de dos condiciones, devolviendo el valor de verdadero cuando una de las proposiciones es verdadera, o cuando ambas lo son, y falso cuando ambas son falsas.

```
V | | V -> V
V | | F -> V
F | | V -> V
F | | F -> F
```

A continuación, en la figura 4, se muestra un ejemplo donde se aprecia el funcionamiento del operador lógico `&&`. En este caso, se ubica la posición del mouse para determinar en qué sector se va a dibujar un rectángulo de color aleatorio.

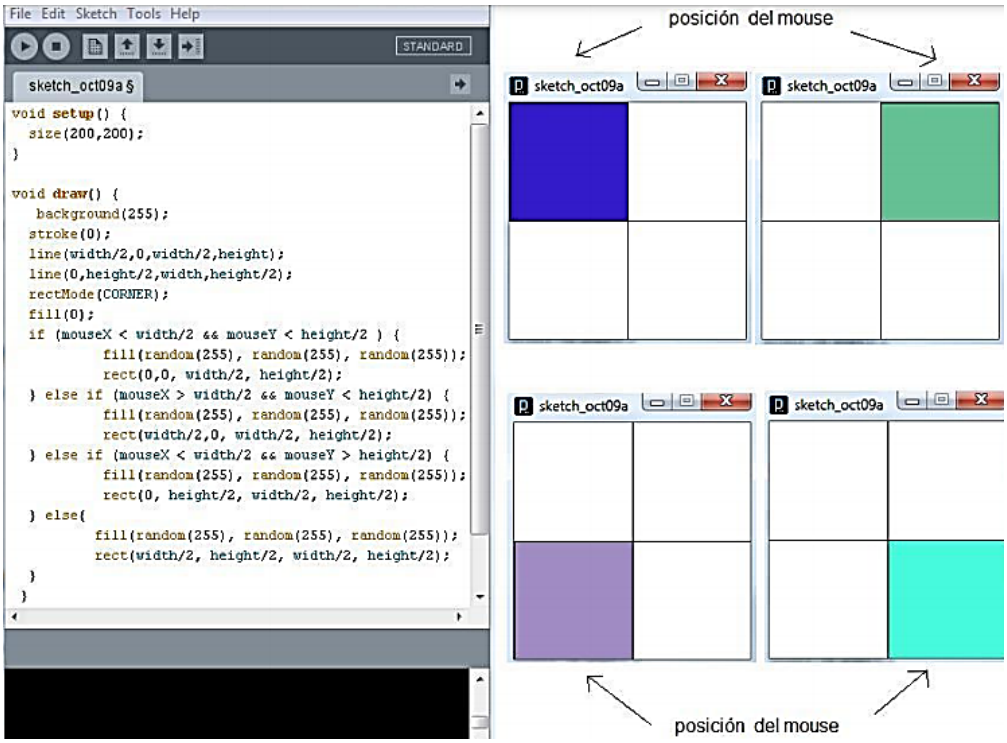


Figura 4. Ejemplo del uso de los operadores lógicos

LA FUNCIÓN RANDOM

La función *random*() se refiere al proceso de aleatoriedad. Este término se asocia a todo proceso cuyo resultado no es previsible más que por azar. El resultado de todo suceso aleatorio no puede determinarse en ningún caso antes de que este se produzca. En computación, la función *random*() es capaz de generar un número flotante aleatorio dado un rango de valores proporcionado. Se utiliza de la siguiente manera:

```
float w;
w=random(0,10);
```

donde la variable `w` es definida como flotante y después a esa variable le asignamos el valor de la aplicación de la función *random*(). En la figura 5 se muestra un ejemplo de uso de esta función, donde todos los elementos se generan de manera aleatoria (color, tamaño, ubicación)

LAS ESTRUCTURAS DE CONTROL ITERATIVAS

En la vida diaria existen situaciones que frecuentemente se resuelven por medio de realizar una secuencia de pasos que puede repetirse muchas veces mientras no se logre la meta trazada. A este tipo de algoritmo se le conoce como algoritmos iterativos o repetitivos.

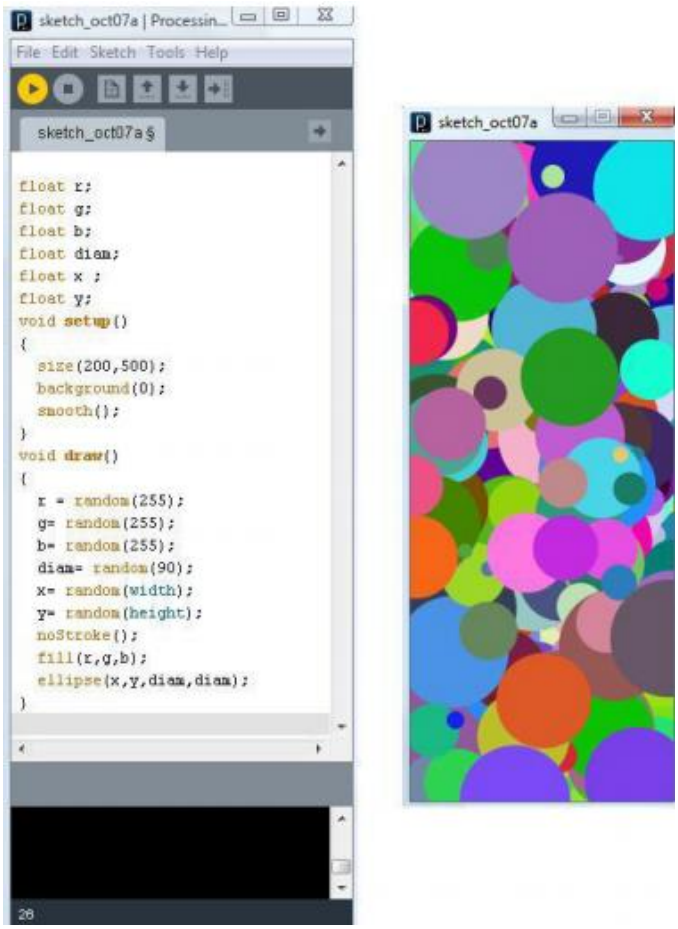


Figura 5. Uso de la función *random*()

Supongamos que nos piden que realicemos el dibujo de cinco rectángulos de las mismas características, pero en posición diferente. En realidad, no tendríamos problemas para escribir cinco veces la función *rect*():

```
rect(20,20,20,20);
rect(50,20,20,20);
rect(80,20,20,20);
rect(110,20,20,20);
rect(140,20,20,20);
```

Básicamente, este programa cumple con el objetivo. Pero ¿qué podemos apreciar? Se repitió cinco veces la misma instrucción. Ahora, ¿qué pasaría si quisiéramos dibujar no solo cinco sino cien rectángulos? Tendríamos que agregar más líneas de código y seguramente tardaríamos mucho tiempo en escribirlo. Sin embargo, existe un recurso para solucionar este tipo de problemas, nos referimos a las estructuras de control iterativas. **Una iteración** consiste en una repetición de un bloque de sentencias un número determinado de veces o hasta que se cumpla una condición. De esta manera, el código puede simplificarse.

- La instrucción `while` Esta instrucción, permite repetir un número indeterminado pero finito de veces un bloque de instrucciones hasta que no se cumpla una condición específica. La estructura del `while` es la siguiente:

```
while (expresión booleana){
      // Este es el conjunto de instrucciones que se ejecutan
      // si la condición es verdadera
}
```

Al cumplirse la expresión booleana, el flujo del programa entrará a la estructura `while` y repetirá la ejecución de las instrucciones de manera indefinida hasta que la condición no se cumple. En ese momento, el flujo del programa continuará con su ejecución normal. Los elementos que deben de estar integrados para que la estructura `while` pueda trabajar de manera correcta son:

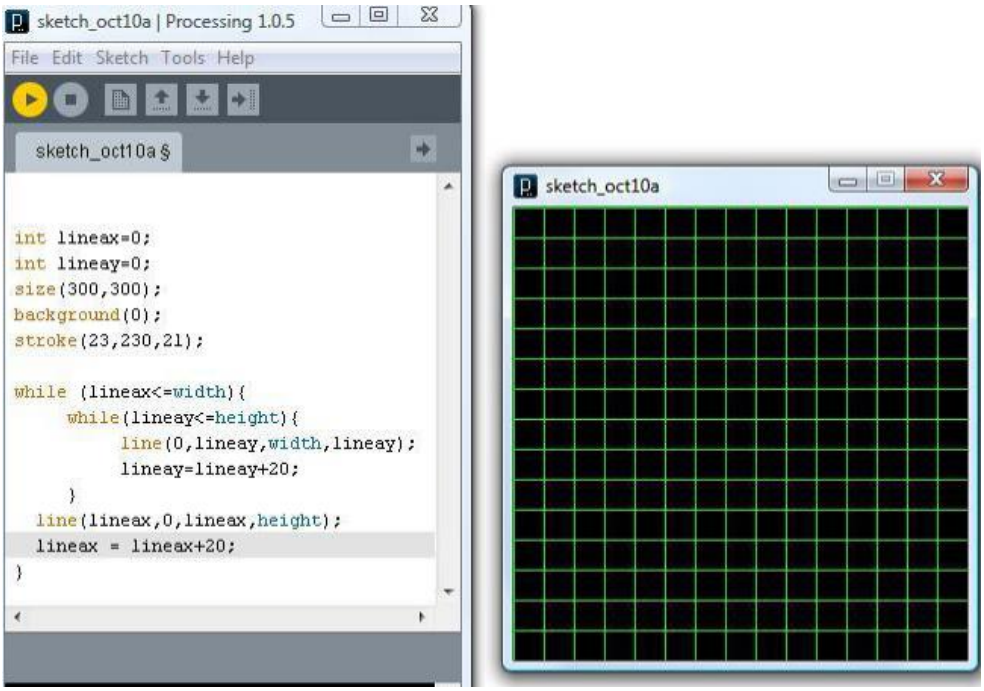
1. La inicialización: establece un valor inicial para aquellas variables que participan en la condición booleana.
2. La condición: es la expresión booleana que se evalúa como verdadero o falso, según el valor de las variables que participan en esa condición con la cual se decide si el cuerpo del ciclo debe repetirse o no.
3. La actualización: es una instrucción que debe de estar en el cuerpo de la estructura `while`, la cual hace cambiar el valor de las variables que forman parte de la condición. Dentro de la estructura del `while` se debe de actualizar la condición. No necesariamente la actualización se realiza en una sola acción, puede darse en varias acciones y ser tan compleja como sea necesario, pero siempre dentro de la estructura.

Entonces, para nuestro ejemplo, integrando la estructura `while` el resultado es el siguiente:

```
int variable1=20;
while(variable1<=140){
      rect(variable1,20,20,20);
      variable1 = variable1 + 30;
}
```

Es importante mencionar que esta estructura podría no ejecutarse, si es que la condición definida nunca se cumple. Otro caso específico que puede ocurrir es que el ciclo sea infinito, esto quiere decir que la condición siempre se cumple.

Al igual que la estructura `if`, al utilizar la estructura `while` también podemos utilizar otra estructura `while` dentro. A esto se le llama un ciclo anidado. A continuación, se muestra un ejemplo de la construcción de una malla en la pantalla utilizando `while` anidado:



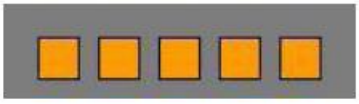
- La instrucción `for`: Es utilizada cuando sabemos de antemano el número de veces que debemos repetir un conjunto de instrucciones. También es un bloque, y está definido de la siguiente manera:

```
for (inicialización, condición, actualización)
{
   // Este es el conjunto de instrucciones que se ejecutan
   // n número de veces
}
```

La estructura del `for` debe de tener los tres elementos de la iteración (inicialización, actualización y condición), representadas de manera explícita en la propia estructura iterativa. Así, si por ejemplo se debe dibujar un número `n` de rectángulos, la estructura `for` sería la siguiente:

```
for(int variable1=20; variable1<=140;
    variable1=variable1+30)
{
    rect(variable1,20,20,20);

}
```



La estructura del `while` y del `for` son equivalentes, la diferencia es que en el `for` concentramos la inicialización, la condición y la actualización en la definición de la estructura, mientras que en el `while` esta definición es más libre. Al igual que en la estructura del `while`, en el ciclo `for` también podemos encontrar anidamientos, es decir, un `for` dentro de otro `for`. A continuación, se muestra el mismo ejemplo de la construcción de la malla, pero ahora utilizando el `for` anidado: