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.

Tipos de datos en Processing

El tipo de dato es un atributo que indica al ordenador (y/o al programador) qué datos 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.
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

Las variables que se van a utilizar deben de ser declaradas en todo programa. En otras palabras, necesitamos avisarle al programa que vamos a utilizarlas y que, por lo tanto, hay que crearlas. 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, 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.

Las variables deben de tomar su valor mediante una llamada asignación del valor. El símbolo para realizar esta asignación es el =. En computación, este símbolo no significa igual que y no estamos haciendo ningún tipo de comparación entre dos valores. La asignación se realiza modificando el valor de la variable por el valor o por el resultado de la parte derecha del símbolo = .

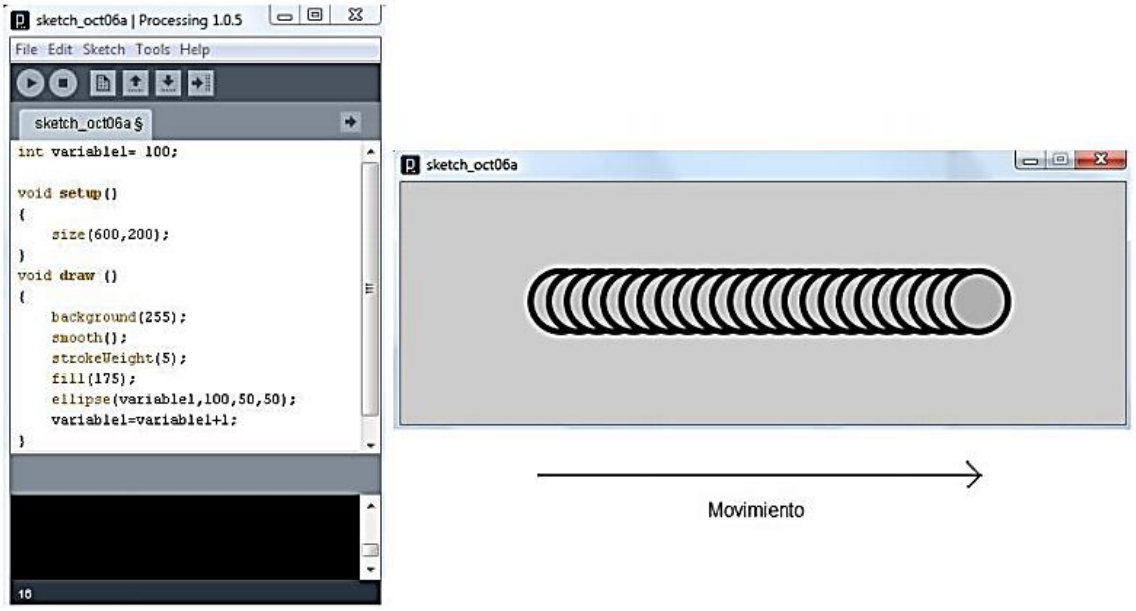
De este modo, algunas asignaciones comunes son las siguientes:

```
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 en cualquier sección de nuestro programa. 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, se muestra 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 en 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 el incremento de ella en 1. Estas características permiten que la elipse tenga un desplazamiento constante hacia la derecha.

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 1 se pueden observar el efecto del uso de estas variables.

El 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*().

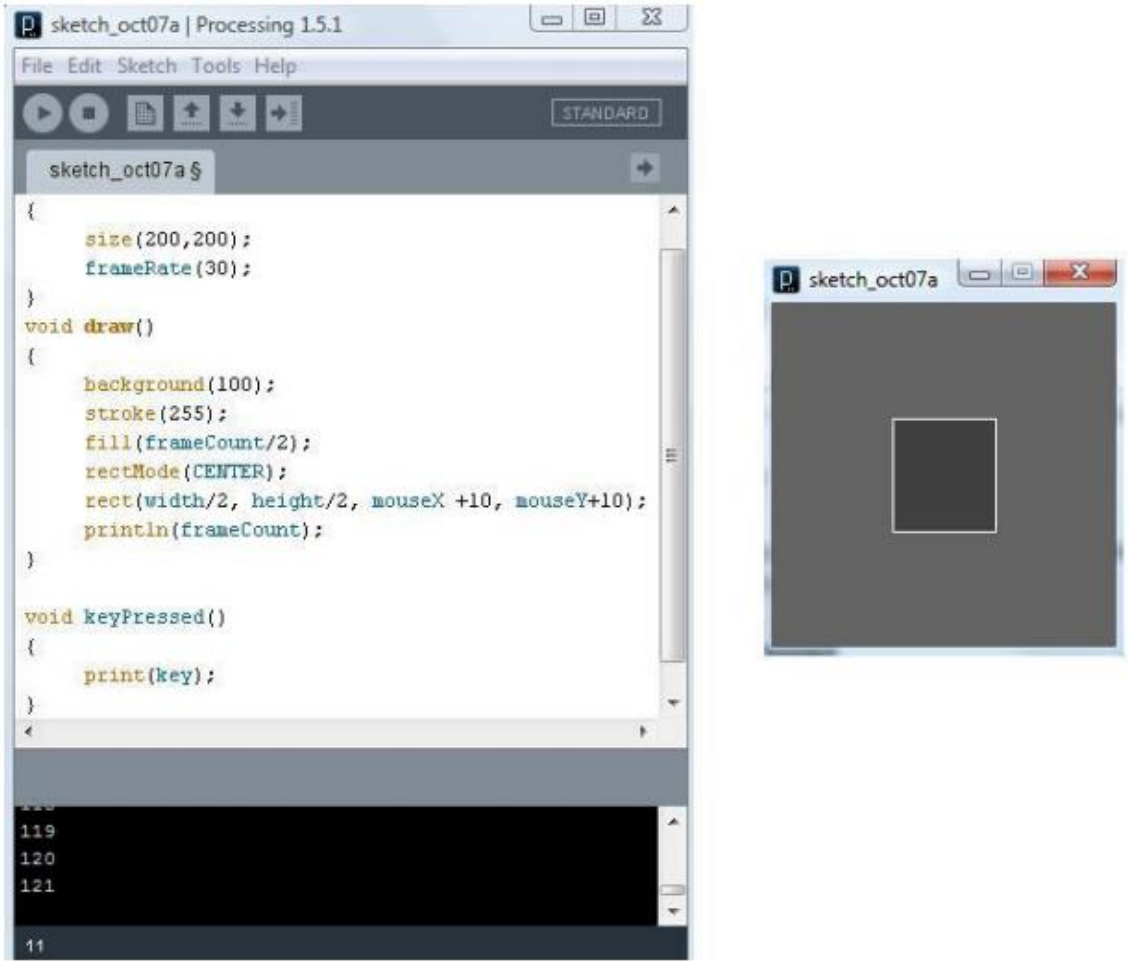


Figura 1. Ejemplo de uso de variables del sistema.

En la figura 2 se muestra un ejemplo de uso de esta función, donde todos los elementos se generan de manera aleatoria (color, tamaño, ubicación)

Las condicionales

Es momento de estudiar uno de los pilares de la programación imperativa: las condicionales. Una condicional, en la programación, es una sentencia o grupo de sentencias que puede ejecutarse, o no, en función del resultado que arroja una condición. La estructura condicional compara una variable contra otro(s) valor(es), para que, con base en el resultado de esta comparación, se siga 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 utilizan los llamados operadores relacionales para verificar el resultado y saber si el resultado cumple o nó la condición, por lo cual se dice que la condición es verdadera o falsa. Los operadores que podemos utilizar son los siguientes:

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

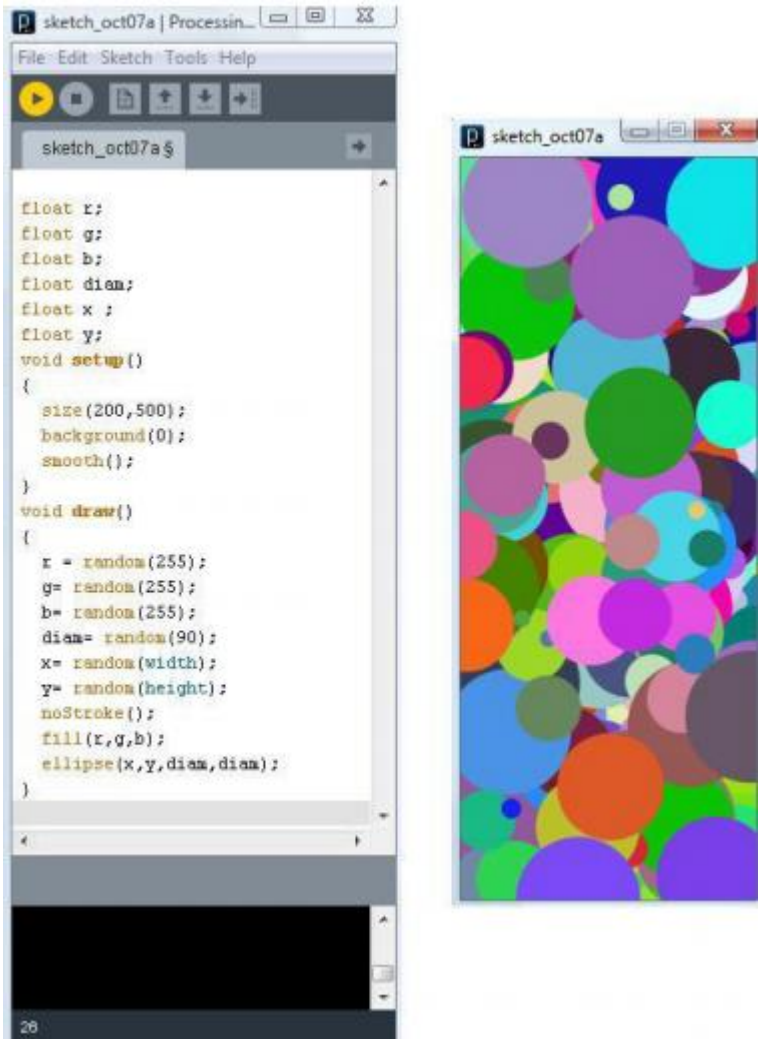


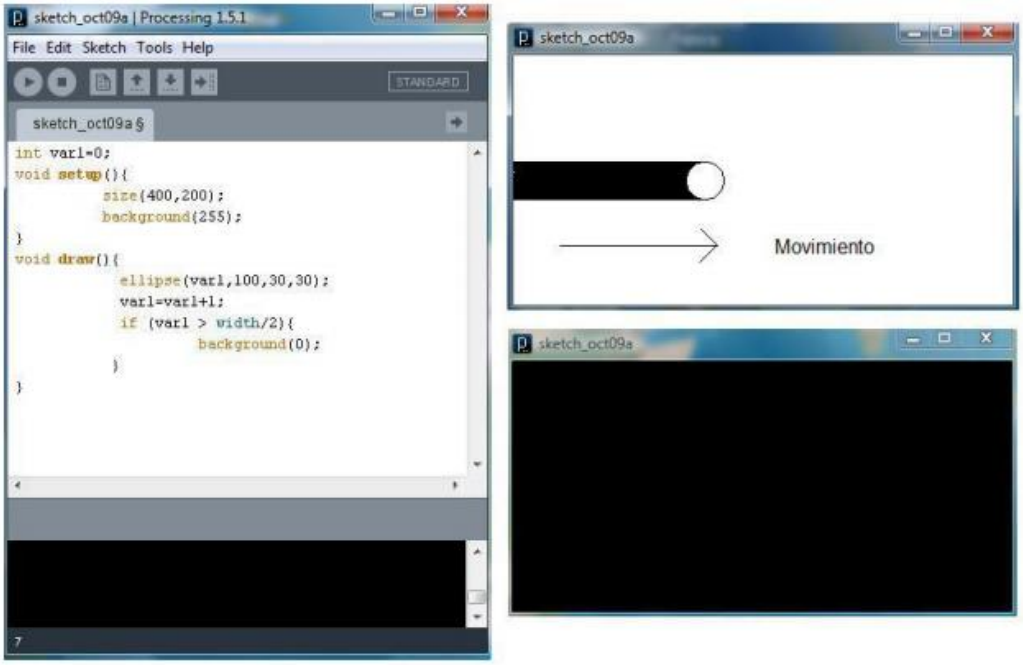
Figura 2. Uso de la función *random*()

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, cuando empezamos a utilizar estructuras de control múltiples, el poder seguir algunas reglas de estilo. Estas reglas nos ayudarán a comprender mejor el programa. En el ejemplo anterior, podemos observar que existe una indentación 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 al él (en la misma línea) debe de ser ignorado. Esto sirve para agregar comentarios internos en el programa y poder hacerlo más legible a cualquier otro usuario que interactúe con el mismo. Del mismo modo, podemos utilizar el (/*) y el (*/) para definir todo un bloque que queramos sea ignorado por Processing. A continuación, se muestra un ejemplo muy simple acerca de la utilización de la estructura condicional simple:

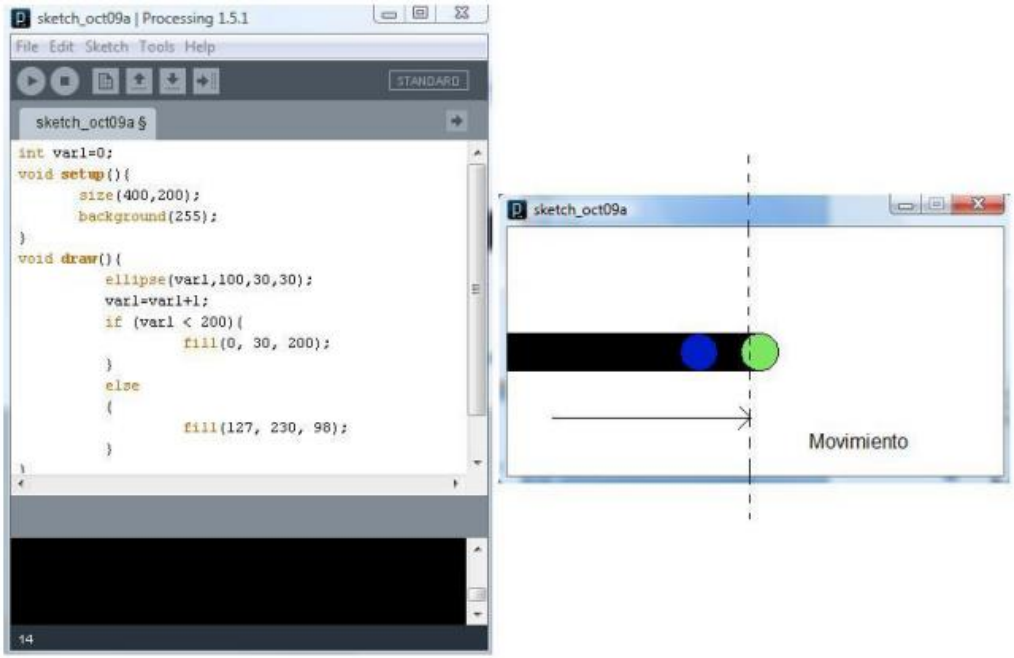


En este ejemplo, dentro del programa solo hay un `if`. El programa verificará el valor de la variable y lo comparará con el valor `width`/2 (que corresponde al centro de la pantalla). Cuando el valor de `var1` es menor (en este caso, su valor inicial es 0) a la mitad de la pantalla, el resultado de la evaluación es `falso` y no se ejecutarán las instrucciones dentro del `if`. Cuando el valor de `var1` 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.

- 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 comparación es verdadero, entonces ejecutará un conjunto de instrucciones, y si el resultado es falso, entonces ejecutará otro conjunto diferente de instrucciones. Una vez realizado este, 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 `var1` sea menor a 200, entonces la pelota se pintará de color azul, pero cuando la variable `var1` obtenga el valor de 200, entonces se pintará automáticamente de color verde. Esta estructura tiene únicamente dos condiciones a verificar y está manejado por la estructura `if` y su correspondiente `else`.

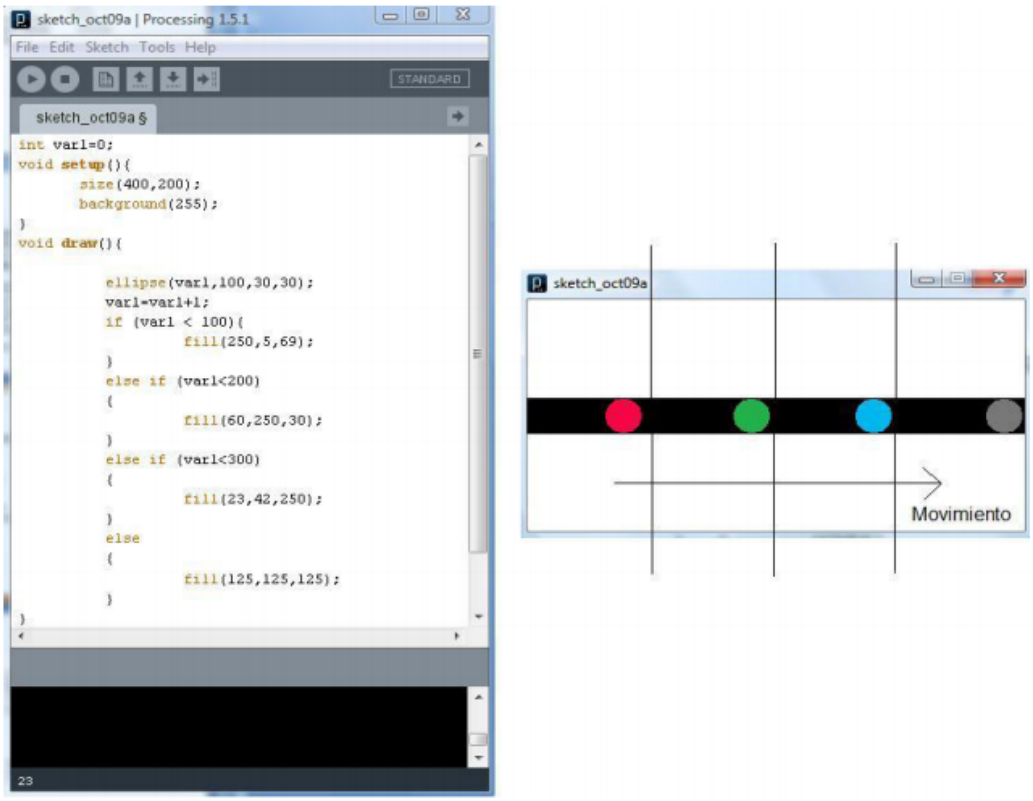
- Múltiples: Las estructuras de comparación múltiples son tomas de decisión especializadas que permiten comparar una variable contra distintos posibles resultados, ejecutando para cada caso una serie de instrucciones específicas. La forma común es la siguiente:

```
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`.



Condicionales compuestos y operadores lógicos

Hasta ahora en los múltiples ejemplos de condicionales, se han evaluado exclusivamente condiciones simples para poder aplicar o no la estructura de la condicional. Sin embargo, en algunas ocasiones necesitaremos comparar un conjunto de condiciones (dos o más) para verificar si una acción se ejecuta o no.
Para solucionar este problema, tendríamos dos posibles soluciones:

1. Generar un conjunto de condiciones independientes disyuntivas, las cuales se verificaría una a una, y donde la acción a ejecutar sería siempre la misma, por ejemplo:

```
if (variable1 == 1){
      variable2=100;
}

if (variable1 == 2){
      variable2=100;
}

if (variable1 == 3){
      variable2=100;
}
```

2. Generar un llamado `if` anidado para verificar un conjunto de condiciones conjuntivas; así, cuando se cumpla una de las condiciones, entonces se verificará otra condición. Si ambas condiciones son verdaderas, entonces finalmente se ejecuta la acción. Un ejemplo de `if` anidado es el siguiente:

```
if (variable1 == 1){
        if(variable2 ==5){
            ellipse(10,10,10,10);
        }
}
```

Para simplificar estas dos estructuras, existen los llamados operadores lógicos, que permiten hacer una evaluación unificada, dado un conjunto de condiciones. Los operadores lógicos utilizados son:

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

De esta manera, se puede apreciar que la condicional es más concisa y se entiende mejor, además de que no se tiene la necesidad de repetir instrucciones. La evaluación de un conjunto de condiciones simultáneas requiere que, al final de la verificación global, se tenga un veredicto para ejecutar, o no, un bloque de código. Para estos casos, se deberán de seguir las denominadas tablas de verdad, para finalmente poder verificar si un conjunto de condiciones es verdadero o no.

- 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, se muestra un ejemplo del uso de estas condicionales compuestas. En este ejemplo, podemos apreciar el funcionamiento del operador lógico `&&`. En la estructura del `if` se realizan un par de comparaciones, al igual que en los `else if`.
En este caso, se ubica la posición del mouse para determinar en qué sector se va a dibujar un rectángulo de color aleatorio. El código del ejemplo se visualiza en la figura 3.

Construcción de botones

Los botones, en computación, son una metáfora utilizada en las interfaces gráficas para simular el funcionamiento de un botón corriente. Los botones suelen representarse como rectángulos con una leyenda o ícono dentro. La principal funcionalidad de los botones es realizar una selección. Esta selección se realiza mediante dos estados que puede tener el botón (encendido

o apagado). Con esta primicia, las variables booleanas pueden ser de gran ayuda. Recordemos que las variables booleanas solo aceptan un par de valores (true y false).

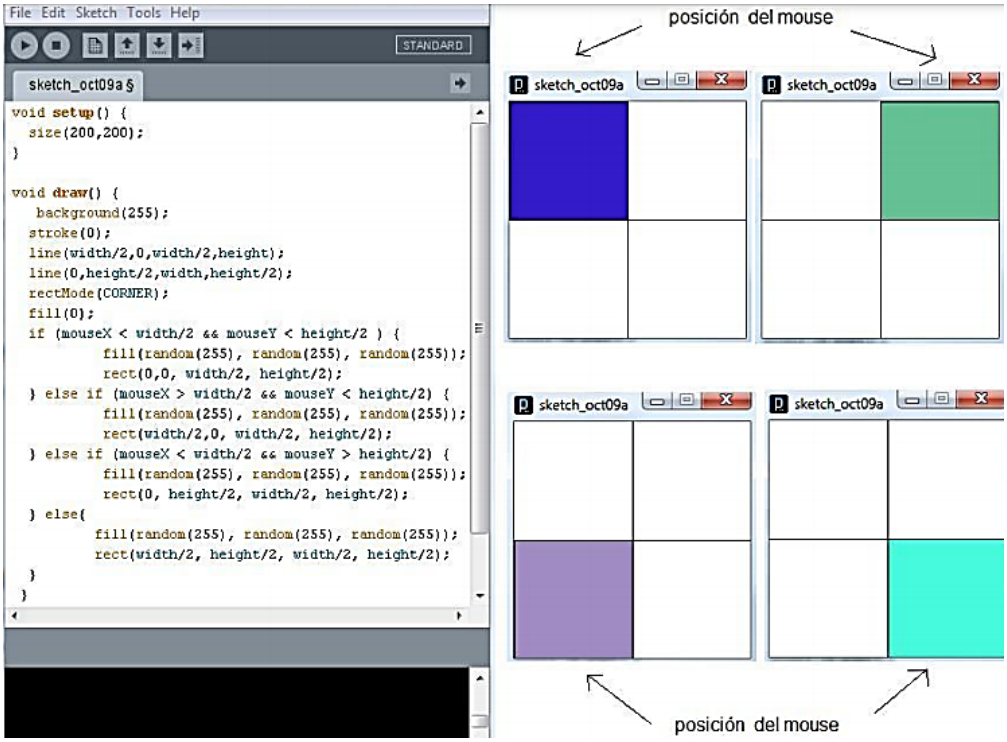
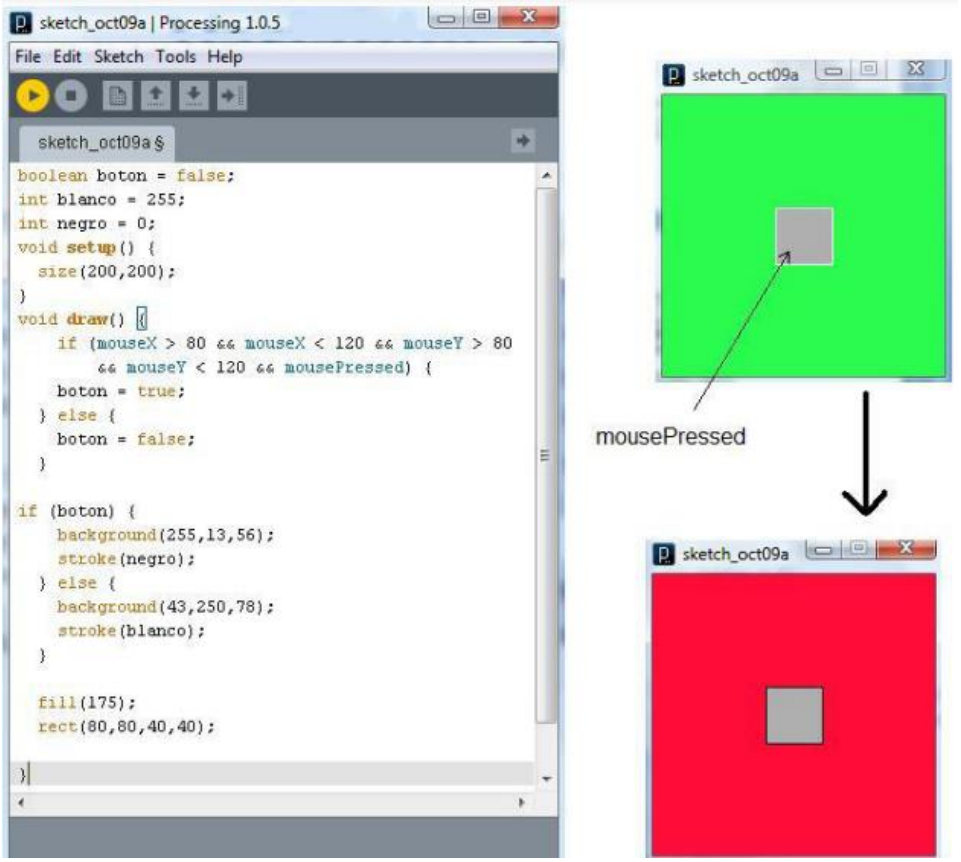
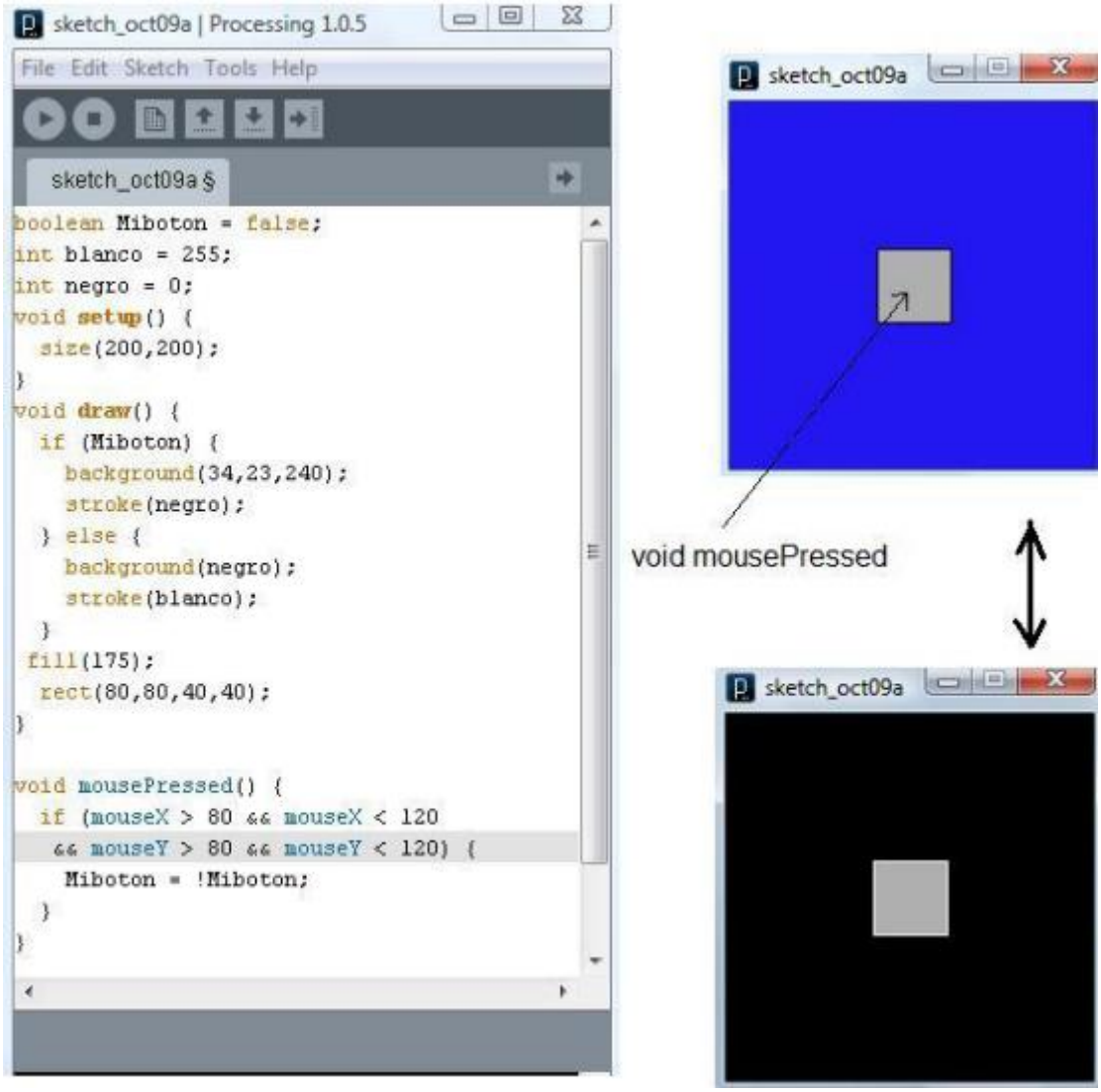


Figura 3. Ejemplo del uso de los operadores lógicos

A continuación, se muestra un ejemplo de la elaboración de un botón.



En este ejemplo, primero se define botón como una variable booleana y se asigna el valor de falso. Posteriormente, se debe verificar la posición del mouse en donde se quiere ubicar nuestro botón y delimitar con las coordenadas x e y. Además, es necesario verificar que la variable `mousePressed` esté activa. Si toda esta condición se cumple, entonces daremos el valor de `true` a nuestro botón. En el segundo bloque del `if`, únicamente se verifica el estado del botón. Si es `true`, entonces asignamos un color rojo al fondo de la pantalla, y si es `false`, entonces el fondo se quedará en verde. Cabe hacer notar que, para activar el botón, el mouse debe de estar siempre presionado. A continuación, se muestra una manera alternativa –y probablemente más útil– de programar botones en Processing.



Este ejemplo funciona de manera un poco diferente al anterior. Al oprimir el botón, provoca que este quede activo todo el tiempo, y se tendrá que desactivar hasta volver a dar clic una vez más en el botón. En este ejemplo, se definió la función *mousePressed* () de manera separada. Al hacer un clic, se entra a la función *mousePressed*() y se verifican las coordenadas del mouse. Si se está dentro de las coordenadas definidas, entonces se debe de cambiar el valor a `Miboton`, si estaba con el valor de `false`, ahora será `true` y viceversa. El `if` que se encuentra en la función *draw*() lo único que hace es verificar el estado del botón, y si es verdadero, entonces dibuja el fondo de pantalla de color azul. En caso contrario, lo dibuja de color negro.

La iteración

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. 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 iteración. 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 de veces una instrucción o 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
}
```

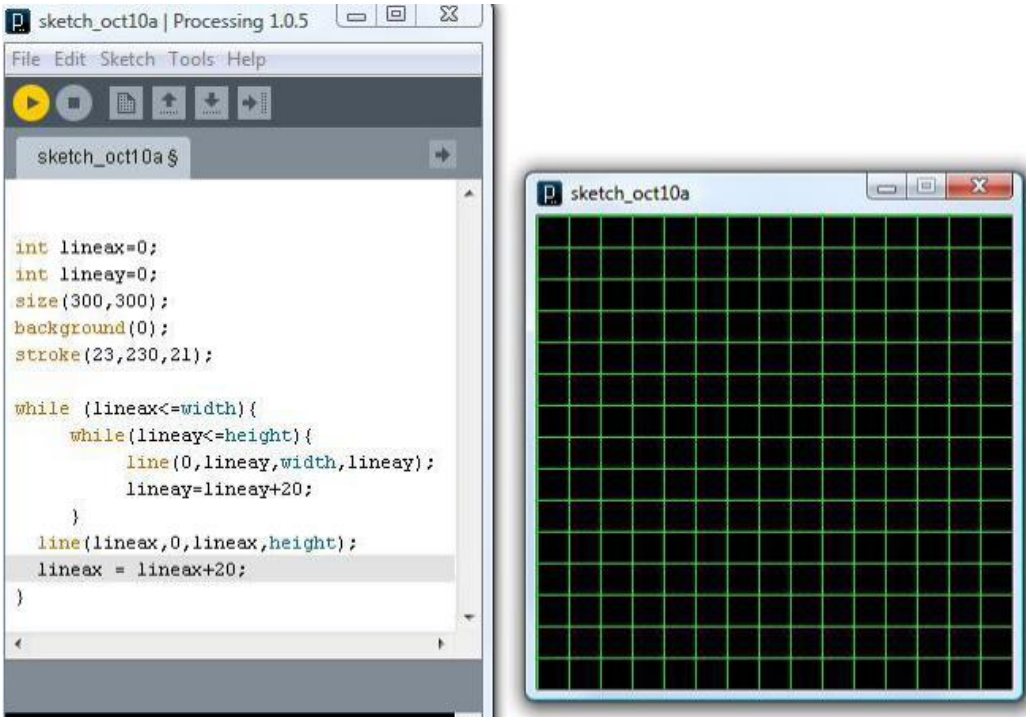
Al cumplirse la expresión booleana, el flujo del programa entrará a la estructura `while` y repetirá la ejecución de las instrucciones de manera indefinida hasta que la condición no se cumple. En ese momento, el flujo del programa continuará con su ejecución normal. Los elementos que deben de estar integrados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



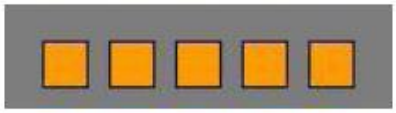
- La instrucción `for`: Es utilizada cuando sabemos de antemano el número de veces que debemos repetir un conjunto de instrucciones. También es un bloque, y está definido de la siguiente manera:

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

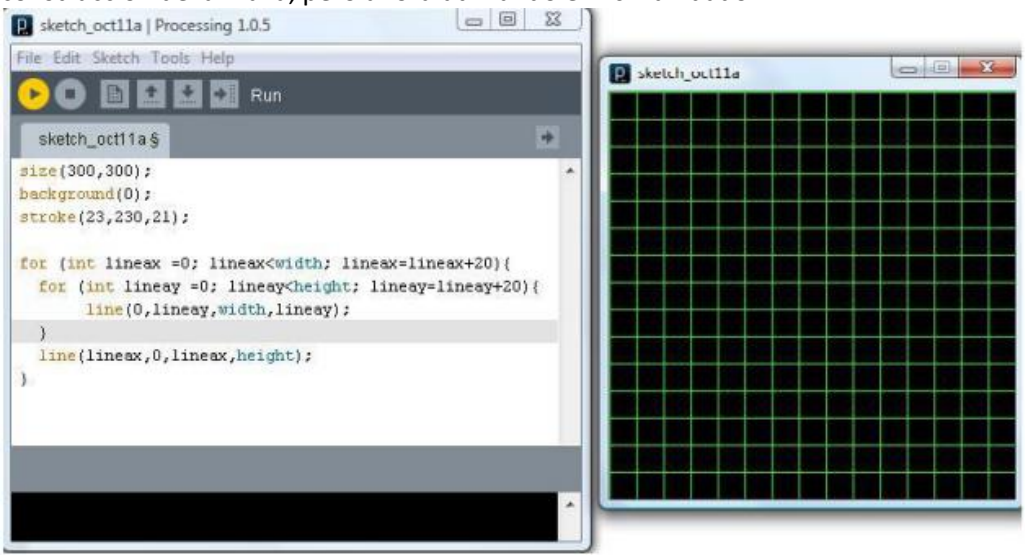
La estructura del `for` debe de tener los tres elementos de la iteración (inicialización, actualización y condición), representadas de manera implícita en la propia estructura iterativa. Así, para el ejemplo de dibujar un número `n` de rectángulos, la estructura `for` sería la siguiente:

```
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:



Otros operadores

Existen los llamados operadores de incremento y decremento, los cuales pueden verse como unos atajos, pueden hacer el código mucho más fácil de escribir y de leer. Es menos complicado entender una línea de código con pocas letras.
Los operadores de incremento y decremento se utilizan a menudo para modificar las variables que controlan el número de veces que se ejecuta alguna instrucción dentro de un ciclo.
El operador de decremento es -- , que significa decrementar una unidad. El operador de incremento es ++ , que significa incrementar una unidad. Los operadores de incremento y decremento producen el valor de la variable como resultado. Algunos ejemplos son:

- `x++; es equivalente a: x = x+1;`
- `x--; es equivalente a: x = x-1;`
- `x+=2; es equivalente a x = x+2;`
- `x*=3; es equivalente a x = x*3;`