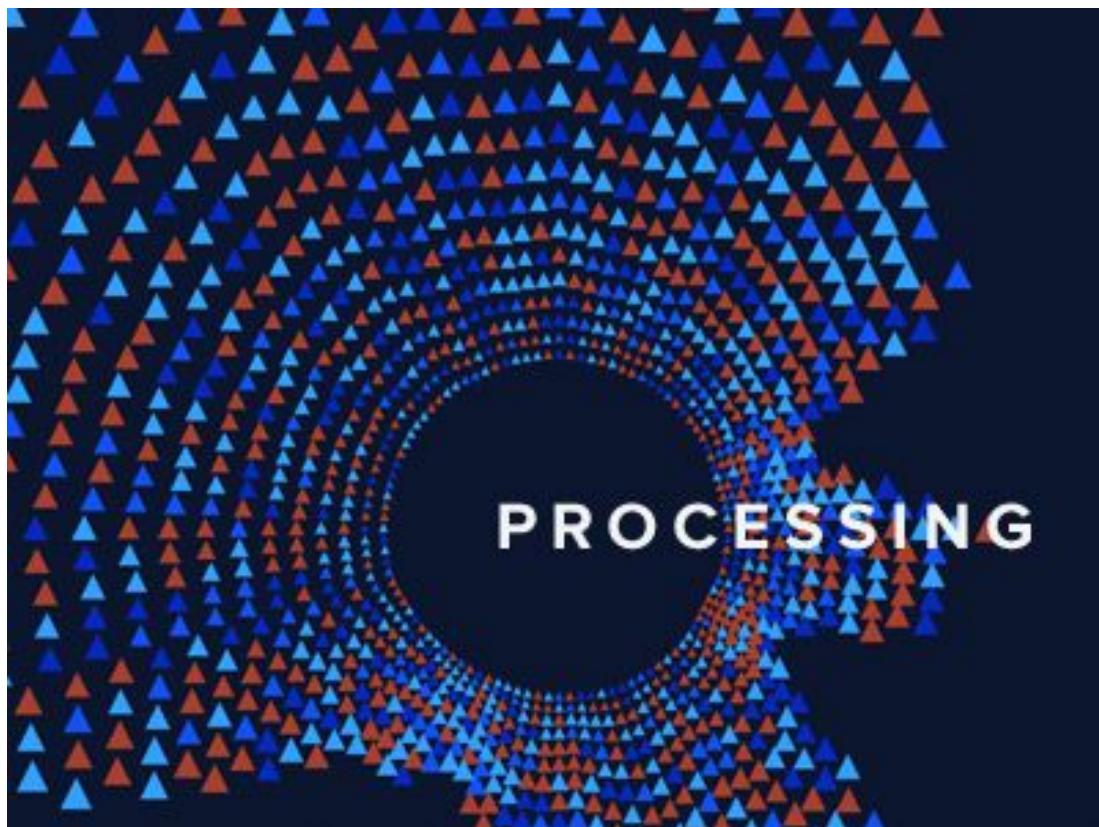


# Programando con Processing



*Tecnologías de la Información y la Comunicación  
II*



BY NC SA

---

Javier Esquiva Mira  
@javiesmi



<https://enclaseconlastic.wordpress.com>

## Índice de Contenido

<b>1. Introducción</b>	<b>4</b>
<b>2. ¿Por qué utilizar Processing?</b>	<b>4</b>
<b>3. Interfaz de Processing.</b>	<b>5</b>
<b>4. Estructura de los programas de Processing y Hello World.</b>	<b>6</b>
<b>5. Prácticas I.</b>	<b>9</b>
<b>6. Formas básicas en Processing.</b>	<b>9</b>
6.1. El punto.	10
6.2. La línea.	10
6.3. Cuadrado y rectángulo.	10
6.4. Círculo y elipse.	11
<b>7. Prácticas II.</b>	<b>11</b>
7.1. Tu nombre	11
7.2. Extraterrestre.	11
7.3. Animalitos.	12
<b>8. Propiedades de la forma en Processing.</b>	<b>12</b>
8.1. Fondo.	13
8.2. Relleno.	14
8.3. Borde.	15
<b>9. Prácticas III.</b>	<b>16</b>
9.1. Aros olímpicos.	16
9.2. Diana.	16
<b>10. Variables Processing.</b>	<b>17</b>
10.1. Enteros (INT).	17
10.2. Reales (FLOAT).	17
10.3. Caracteres (CHAR).	18
10.4. Cadena de caracteres (STRING).	18
10.5. Valores lógicos (BOOLEAN).	18
10.6. Variables del Sistema	19
<b>11. Mostrando información en Processing.</b>	<b>19</b>



---

11.1. En la consola del IDE.	19
11.2. En la ventana o lienzo.	20
<b>12. Iteraciones: bucle for ()</b>	<b>25</b>
<b>13. Prácticas IV.</b>	<b>27</b>
13.1. Líneas horizontales.	27
13.2. Malla sobre fondo verde.	27
13.3. Círculos concéntricos.	28
13.4. Matriz de círculos rojos.	28
<b>14. Iteraciones: bucle While () .</b>	<b>29</b>
<b>15. Interacciones con el ratón.</b>	<b>30</b>
<b>16. Generación de números aleatorios.</b>	<b>31</b>
<b>17. Prácticas V.</b>	<b>32</b>
17.1. Dibujando con el ratón.	32
17.2. Ventana con color de fondo cambiante.	32
<b>18. Estructuras condicionales.</b>	<b>32</b>
<b>19. Interacción con el teclado.</b>	<b>34</b>
<b>20. Prácticas VI.</b>	<b>37</b>
20.1. Jugando con el círculo.	38
20.2. Estela.	38
<b>21. Control del tiempo.</b>	<b>38</b>
<b>22. Prácticas VII.</b>	<b>39</b>
22.1. El coche fantástico.	39
22.2. Semáforo.	39
<b>23. Cargar una Imagen.</b>	<b>40</b>
23.1. La imagen a cargar está en la carpeta de nuestro proyecto (ubicación local).	40
23.2. La imagen está en Internet (ubicación on-line).	41
<b>24. Prácticas VIII.</b>	<b>42</b>
24.1. Imagen local.	42
24.2. Imagen web.	42
<b>25. Control de sonido.</b>	<b>42</b>
<b>26. Práctica IX.</b>	<b>44</b>



---

<b>27. Control de fecha y hora.</b>	<b>44</b>
<b>28. Prácticas X.</b>	<b>46</b>
<b>29. Giros.</b>	<b>46</b>
<b>30. Prácticas XI.</b>	<b>48</b>
30.1. Mueve un ojo.	48
30.2. Mueve los dos ojos.	48
<b>31. Proyectos.</b>	<b>49</b>
31.1. Contador de tiempo.	49
31.2. Artista.	49
31.3. Adapta el código.	49
<b>32. ¿Te interesa y quieres ampliar?</b>	<b>50</b>
<b>33. Webgrafía.</b>	<b>50</b>



## 1. Introducción

Como dicen sus creadores, Casey Reas y Ben Fry en la web del proyecto: Processing is an open source programming language and environment for people who want to program images, animation, and sound. It is used by students, artists, designers, architects, researchers, and hobbyists for learning, prototyping, and production. It is created to teach fundamentals of computer programming within a visual context and to serve as a software sketchbook and professional production tool. Processing is developed by artists and designers as an alternative to proprietary software tools in the same domain. Y para más detalles: [Processing FAQ.](#)

Así, Processing es:

- Un lenguaje de programación.
- Un entorno de programación.
- Un proyecto Open-Source.
- Multiplataforma.

¿Dónde está?

Está aquí: <http://processing.org/download/>

Instalación:

La instalación es extremadamente fácil:

1. Bajar la versión correspondiente.
2. Descomprimir el archivo.
3. Clicad “Processing”... y ya está.

## 2. ¿Por qué utilizar Processing?

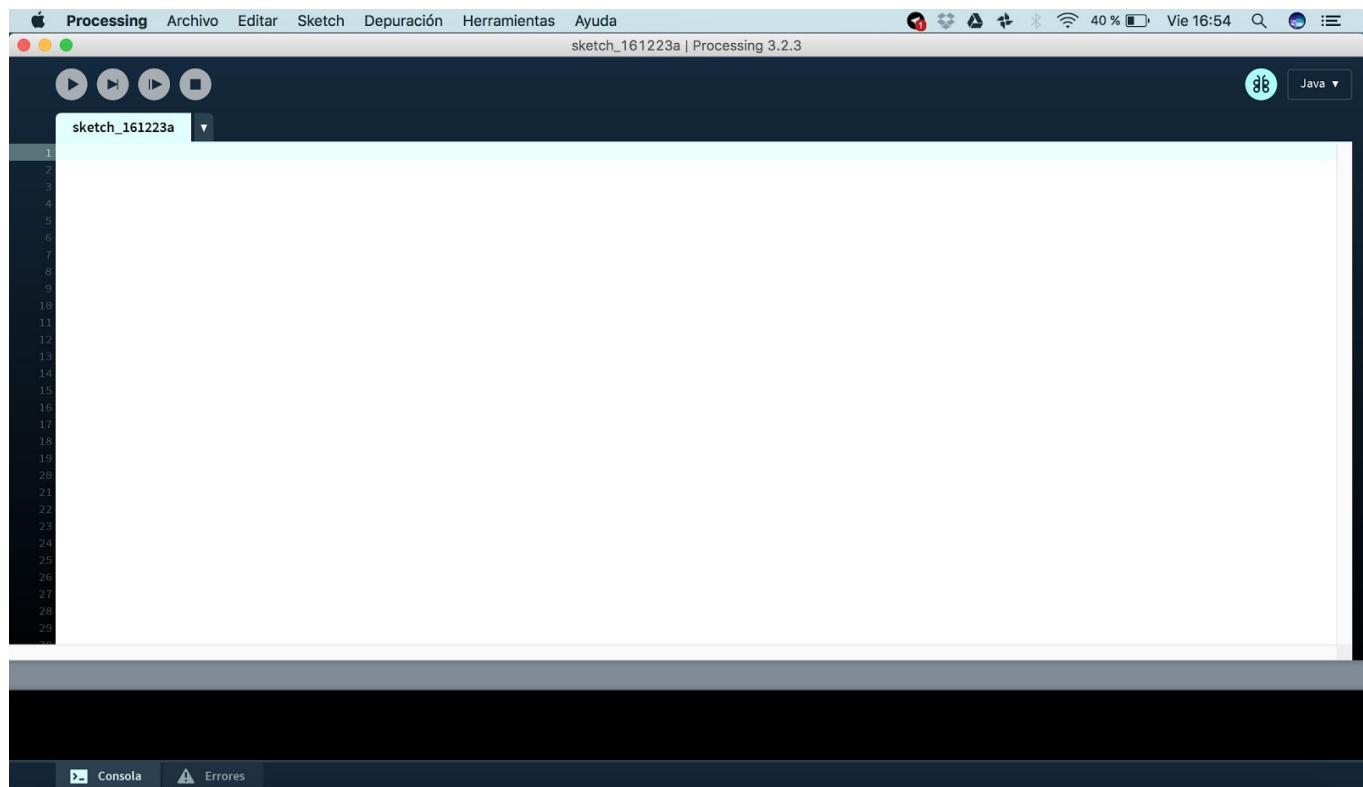
- Porque es un proyecto pensado para aprender a programar, y enfocado al uso de la programación con fines artísticos y de diseño.
- Por la simplicidad de la interfaz, que permite centrarse en el código.
- Porque es un proyecto open-source.
- Porque, sin ser un competidor de Flash, puede ser una alternativa en según qué proyectos, y por eso vale la pena conocerlo y escoger entre uno y otro, en cada caso, según el tipo de proyecto.
- Porque aunque esté pensado para aprender a programar, es suficientemente flexible y potente para desarrollar proyectos muy complejos.
- Porque aunque la programación multimedia en el máster se va a realizar sobretodo en Flash, el código en uno y otro programa son en realidad muy parecidos en los puntos más importantes.



### 3. Interfaz de Processing.

La Interfaz de Processing es sencilla. Esto permite que uno pueda familiarizarse con ella muy rápidamente. Si la comparamos con la complejidad de interfaces como las de Flash o Director, la diferencia es muy considerable.

En la siguiente imagen se pueden distinguir las diferentes partes del IDE de Processing:



(Versión Mac, en el resto de plataformas es prácticamente igual)

Para empezar, nos fijaremos en el siguiente menú:



4 opciones: **Depurar**, **Saltar**, **Continuar**, **Detener**

No hace falta explicarlos. Sólo saber que cuando creamos un proyecto nuevo en Processing, creamos un **Sketch**, y el **Sketch folder** es la carpeta donde éste se guarda. En principio, el Sketch consta tan solo de la misma carpeta y de un archivo **.pde** que contiene el código. Luego se pueden añadir carpetas para archivos de imagen, video, etc.

**Depurar:** Sirve para ejecutar el código y abre una nueva ventana donde se muestra el resultado. Con la magia de este simple botón nos ahorramos lo que en programación se conoce como compilación.

**Detener:** Detiene la ejecución del código y cierra la ventana que abre con el resultado.

Los otros dos botones de esta barra (Saltar y Continuar), así como el menú superior, puedes encontrar una

---

explicación sencilla (en inglés) en el menú Ayuda/Entorno.

También puede ser de interés consultar el menú Ayuda/Referencia donde encontrarás ayuda sobre los comandos que existen en Processing.

## 4. Estructura de los programas de Processing y Hello World.

En su casi totalidad se componen de dos funciones o bloques cuyos nombres son siempre así:

### **void setup() y void draw()**

Por otra parte, es obligado comentar que en todo lenguaje de programación es muy recomendable comentar el código. Esto es debido a que en muchas ocasiones no es muy intuitivo, se le puede olvidar ciertos detalles al propio programador al cabo del tiempo, o bien para facilitar la lectura del código a otros programadores (cosa muy normal ya que los grandes programas suelen ser fruto del trabajo en equipo de numerosas personas a lo largo del tiempo, con las habituales mejoras y actualizaciones).

Para hacer comentarios de una línea se debe iniciar el comentario con //

### **Ejemplo:**

```
//variable media: media de notas de alumno
```

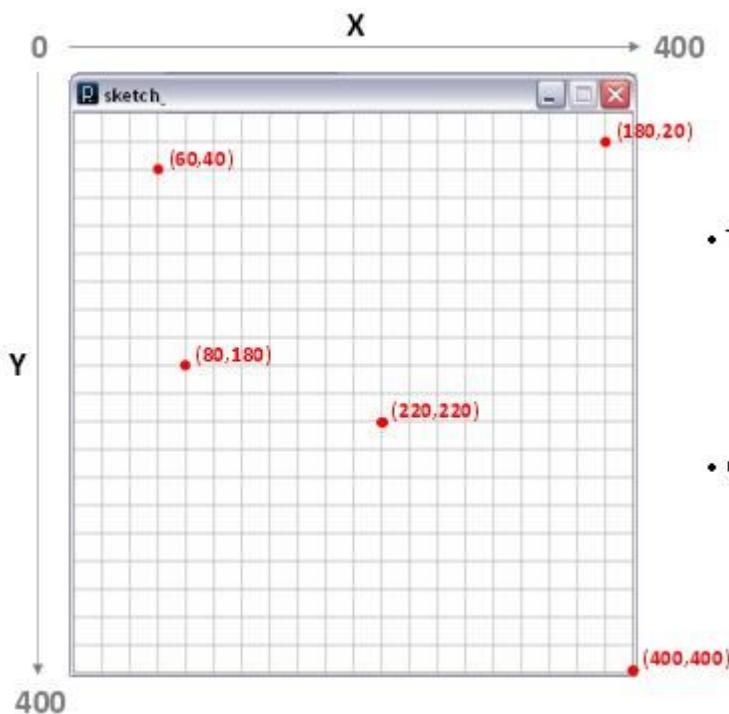
Para hacer comentarios de más de una línea, se debe comenzar el comentario con /\*, y finalizar con \*/

### **Ejemplo:**

```
/* Esta línea de código genera un número aleatorio entre 1000 y 100.000 */
```

Processing es un lenguaje de programación muy visual (de hecho es muy utilizado por profesionales del diseño y las Artes Plásticas precisamente por ello) por lo que debemos tener bien claro cómo utiliza las coordenadas en el plano a la hora de dibujar. Veámoslo en la siguiente imagen:





- Tamaño del lienzo o superficie de dibujo

Para establecer el tamaño del lienzo es necesario definir el ancho (eje x) y el alto (eje y) de la superficie.

```
size(ancho,alto);
```

Ej. size(400,400);

- Coordenada

Posición (punto) en el espacio.

Luego como podemos ver, todo programa en Processing donde utilicemos el entorno gráfico, pasa por definir una ventana o lienzo donde va a transcurrir la acción del programa. Para ello ya hemos visto que debemos utilizar la función:

```
size(ancho,alto);
```

Bueno, una vez visto un poco de teoría es hora de ponernos manos a la obra y empezar a escribir código y generar **sketches**, que es como llama Processing a cada uno de los programas que se generan.

Processing, como cualquier entorno de programación que se precie, tiene una **consola**. Un espacio donde, vía texto, podemos recibir mensajes del programa y, cuando los haya, ver y descifrar los errores. La consola de Processing se encuentra debajo del todo en la interfaz de Processing, con fondo negro.



Aquí es donde veremos los resultados de nuestro programa, lo que en el argot de los programadores se conoce como el "Hello World!".

Ejecutarlo, en Processing, es tan sencillo como escribir:

```
print("hello world!");
```

dentro la interfaz de Processing, darle a "Depurar" y... voilà! ¡Habéis creado el primer programa!

Si habéis visto "Hello World!" escrito en blanco en la consola, el primer programa ha sido compilado y ejecutado con éxito.

**print** es una instrucción que le dice a java que escriba el parámetro (lo que va entre paréntesis) a la consola. Si cambiáis "Hello World!" por cualquier otra cosa y ejecutáis el programa veréis que cambia el resultado.

Notad que hay un punto y coma al final de la línea.

### **EN JAVA, HAY QUE ACABAR CADA LÍNEA DE CÓDIGO CON PUNTO Y COMA;**

Eso significa que hace falta un ";" al final de cada instrucción, de hecho no necesariamente de cada línea tal y como la escribiremos a la interfaz de Processing. Es un error muy común olvidarse punto y comas.

También es MUY IMPORTANTE al programar EL ORDEN en el que escribimos las instrucciones.

Así, el código:

```
println("Hello World!");
println("¡Hola mundo!");
```

escribirá primero la frase en inglés y luego en español. Obviamente aquí el ejemplo es trivial, pero la importancia es crucial a medida que el código se torna complejo.

También habréis visto que en este segundo ejemplo la instrucción es **println**, con "In". Esto significa "print line", y ejecuta un salto de línea después de escribir el parámetro.

Probad el código:

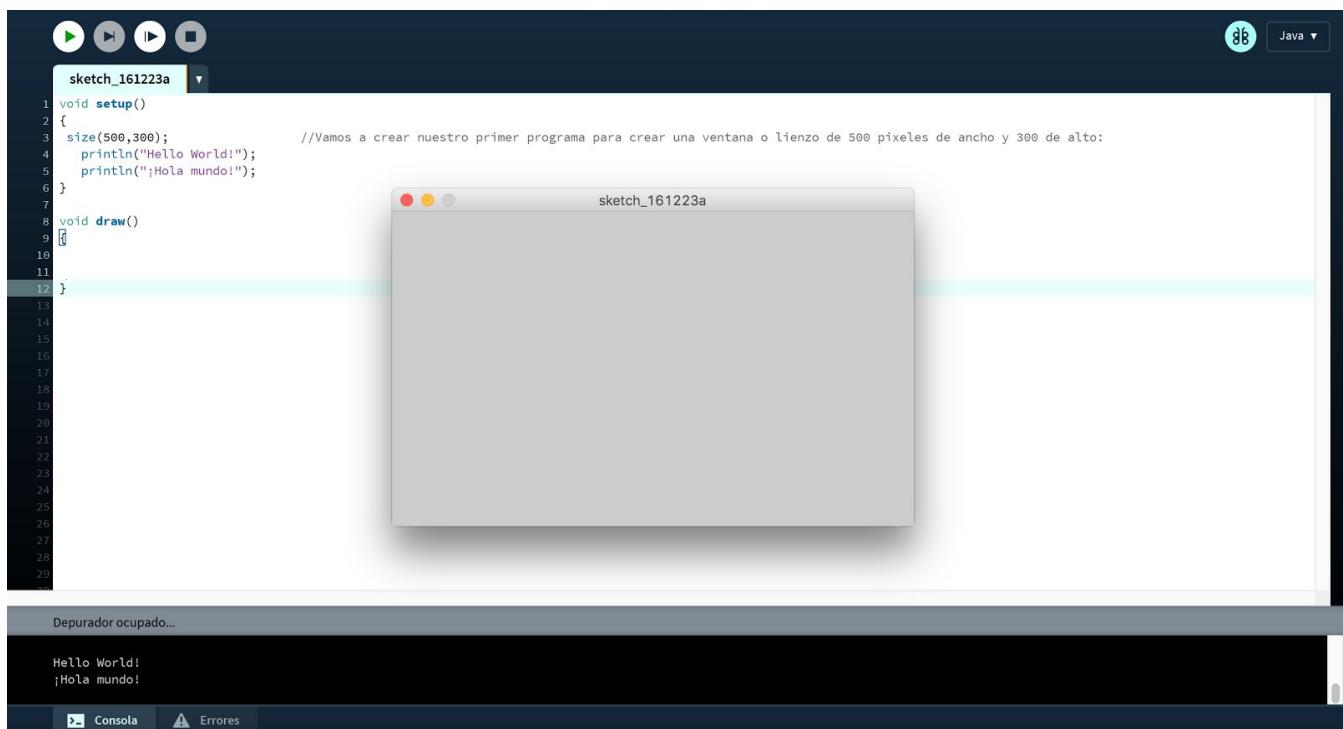
```
print("Hello World!");
print("¡Hola mundo!");
```

y veréis la diferencia.

Finalmente, habréis notado que cada vez que ejecutábamos el programa se abría una pequeña ventana en blanco (en gris, de hecho). Se trata de la ventana que Processing crea y es donde nos centraremos, sobretodo, a partir de ahora. Sin dejar nunca de tener un ojo a la consola, herramienta esencial para el programador, evidentemente lo que nos interesará es la ventana, ya que allí estará el resultado final de nuestra aplicación. Lo que el público o el visitante verá.

Vamos a crear nuestro primer programa para crear una ventana o lienzo de 500 pixeles de ancho y 300 de alto y en el que se pueda leer en la consola Hello World en una línea y Hola Mundo en otra debajo. Debe quedar algo similar a la imagen que hay a continuación:





## 5. Prácticas I.

Genera 3 programas diferentes con los siguientes datos:

1. Dibujar un lienzo de 500 de ancho y 500 de alto.
2. Dibujar un lienzo de 100 de ancho y 400 de alto.
3. Dibujar un lienzo de 400 de ancho y 200 de alto.

Añade comentarios detrás de las líneas con comandos (por ejemplo la de tamaño de lienzo).

Saca una captura de pantalla en el que se vea el lienzo generado y de fondo del IDE de Processing (tal y como se ve la imagen anterior). Pega cada una de las imágenes en un [Documento de texto de Google](#) y súbelo al espacio correspondiente que hay en Google Classroom.

## 6. Formas básicas en Processing.

En Processing existen 4 formas básicas: punto, línea, cuadrado-rectángulo y círculo-elipse.

Hay que tener en cuenta que trabajamos en una cuadrícula de píxeles, por lo que a la hora de dibujar las formas tendremos que dar valores en píxeles.

Con el fin de recoger los diferentes ejemplos que tienes que crear en este apartado, crea un [Documento de texto de Google](#) y pega en él cada uno de los **ejemplos** que se te proponen en los subapartados que vienen a continuación (como el de la práctica anterior).

## 6.1. El punto.

Para ello utilizaremos la función:

```
point(x,y);
```

donde "x" es la coordenada X o ancho, e "y" es la coordenada Y o alto (recordad el sistema de coordenadas del apartado anterior).

Veámoslo con un ejemplo.

Diseña un programa cuyo lienzo sea de 500 por 300 y un punto situado en la coordenada 250, 250.

## 6.2. La línea.

Para ello utilizaremos la función:

```
line(x1,y1,x2,y2);
```

x1: posición en el eje X donde se inicia la línea.

y1: posición en el eje Y donde se inicia la línea.

x2: posición en el eje X donde finaliza la línea.

y2: posición en el eje Y donde finaliza la línea.

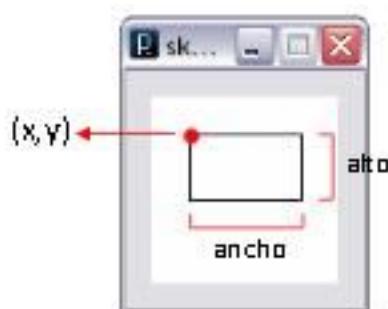
Veámoslo con un ejemplo.

Diseña un programa cuyo lienzo sea de 500 por 300 y una línea desde la coordenada 100, 100 hasta la 200, 200.

## 6.3. Cuadrado y rectángulo.

Para ello utilizaremos la función:

```
rect(x,y,ancho,alto);
```



**x**: posición inicial en el eje X.

**y**: posición inicial en el eje Y.

**ancho**: ancho del cuadrado o rectángulo.

**alto:** alto del cuadrado o rectángulo.

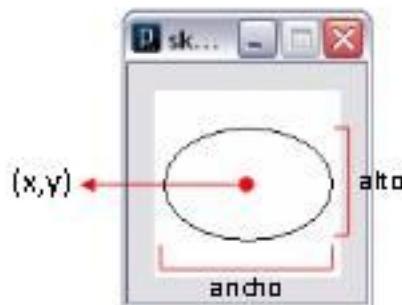
Veámoslo con un ejemplo.

Diseña un programa cuyo lienzo sea de 500 por 300 y dibuja un cuadrado desde la coordenada 50, 50 y de ancho y alto 70. Dibuja también un rectángulo desde la coordenada 200, 50 y de ancho 200 y alto 70.

## 6.4. Círculo y elipse.

Para ello utilizaremos la función:

```
ellipse(x,y,ancho,alto);
```



**x:** posición en el eje X donde se inicia la elipse o círculo.

**y:** posición en el eje Y donde se inicia la elipse o círculo.

**ancho:** ancho del círculo o elipse.

**alto:** alto del círculo o elipse.

Veámoslo con un ejemplo.

Diseña un programa cuyo lienzo sea de 500 por 300 y dibuja un círculo con centro en la coordenada 50, 50 y de ancho y alto 70. Dibuja también una elipse desde la coordenada 200, 50 y de ancho 200 y alto 70.

## 7. Prácticas II.

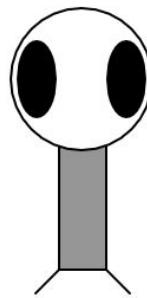
Vamos a realizar dos programas muy sencillos. Guarda la captura de pantalla de cada uno de ellos en un [Documento de texto de Google](#).

### 7.1. Tu nombre

Escribe tu nombre en un lienzo de 500 por 300, utilizando las figuras básicas vistas en el apartado anterior.

### 7.2. Extraterrestre.

Utilizando las figuras básicas, realiza un muñeco similar al de la siguiente imagen.



### 7.3. Animalitos.

Realiza un programa en el que dibujes una rana y un osito como en la imagen. Ordena el código por partes (cara, ojos, boca, etc.), empieza por lo más importante y continua con los detalles.



## 8. Propiedades de la forma en Processing.

Hasta ahora, el fondo de nuestro lienzo era siempre gris y las líneas que generábamos siempre negras. Pues bien, vamos a aprender a dar color al fondo, al relleno de las figuras y a las líneas.

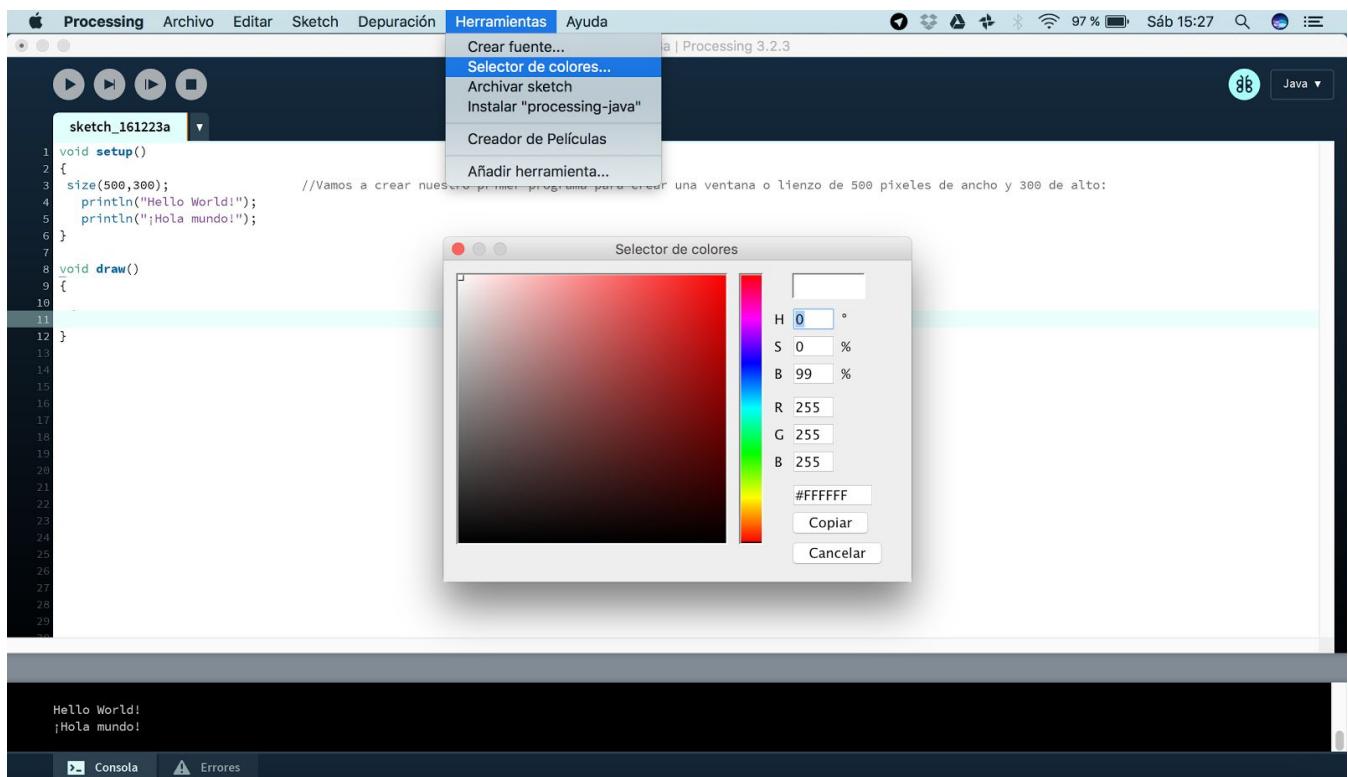
Lo primero que debemos saber es cómo trabaja el color Processing.

Hay tres sistemas:

- El RGB, en el que se parte de que cualquier color se puede conseguir mezclando el R (red-rojo), el G (green-verde) y el B (blue-azul); en tanto pongamos más de uno que de otros. El valor que toma cada uno de ellos va de 0 a 255. Si se utiliza un cuarto parámetro es para referirnos a la transparencia.
- El HSB, en el que se utilizan los parámetros tono, saturación y brillo. El valor que toma cada uno de ellos va de 1% a 100%. Si se utiliza un cuarto parámetro es para referirnos a la transparencia.
- #000000 (notación hexadecimal), donde cada dos cifras de dicho número representa el valor en código hexadecimal de la componente Red, Green o Blue del color.

Para obtener cualquiera de estos parámetros de cualquier color es muy sencillo con una herramienta de la que dispone Processing.

Pinchamos en el menú principal en “Herramientas -> Selector de colores...”.



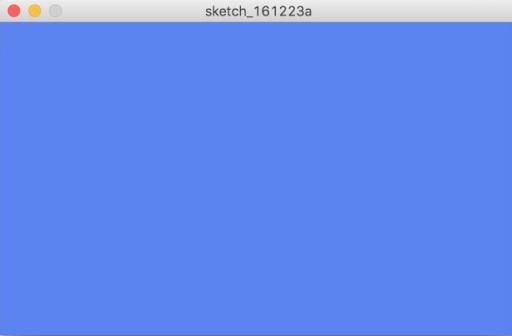
Pinchamos sobre el color deseado y aparecerán sus diferentes valores en la parte derecha de la ventana. Copiamos el código del color (tipo HTML) y lo utilizamos en nuestro programa.

## 8.1. Fondo.

La instrucción con la que modificamos el fondo del lienzo es:

**background(red, green, blue);**

En la siguiente imagen vemos un ejemplo para poner el fondo azul:



```

sketch_161223a
1 void setup()
2 {
3   size(500,300);           //Vamos a crear nuestro primer programa para crear una ventana o lienzo de 500 pixeles de ancho y 300 de alto:
4   background(92, 132, 240);
5 }
6
7 void draw()
8 {
9
10}
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29

```

Depurador ocupado...

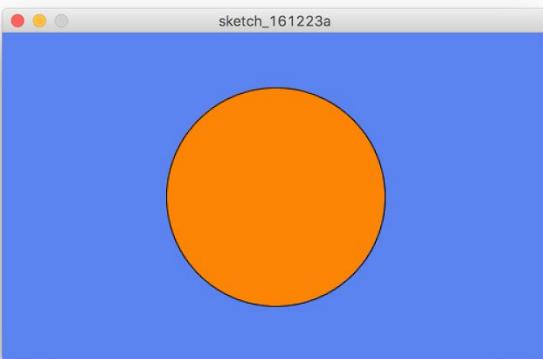
Consola | Errores

## 8.2. Relleno.

Añadimos un círculo de un determinado color. Esto lo haremos con la función:

**fill(red,green,blue);**

colocándola antes de llamar a la función de la forma **[muy importante]**.



```

sketch_161223a
1 void setup()
2 {
3   size(500,300);           //Vamos a crear nuestro primer programa para crear una ventana o lienzo de 500 pixeles de ancho y 300 de alto:
4   background(92, 132, 240);
5   fill(252, 133, 5);
6   ellipse(250, 150, 200, 200);
7 }
8
9 void draw()
10{
11
12}
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29

```

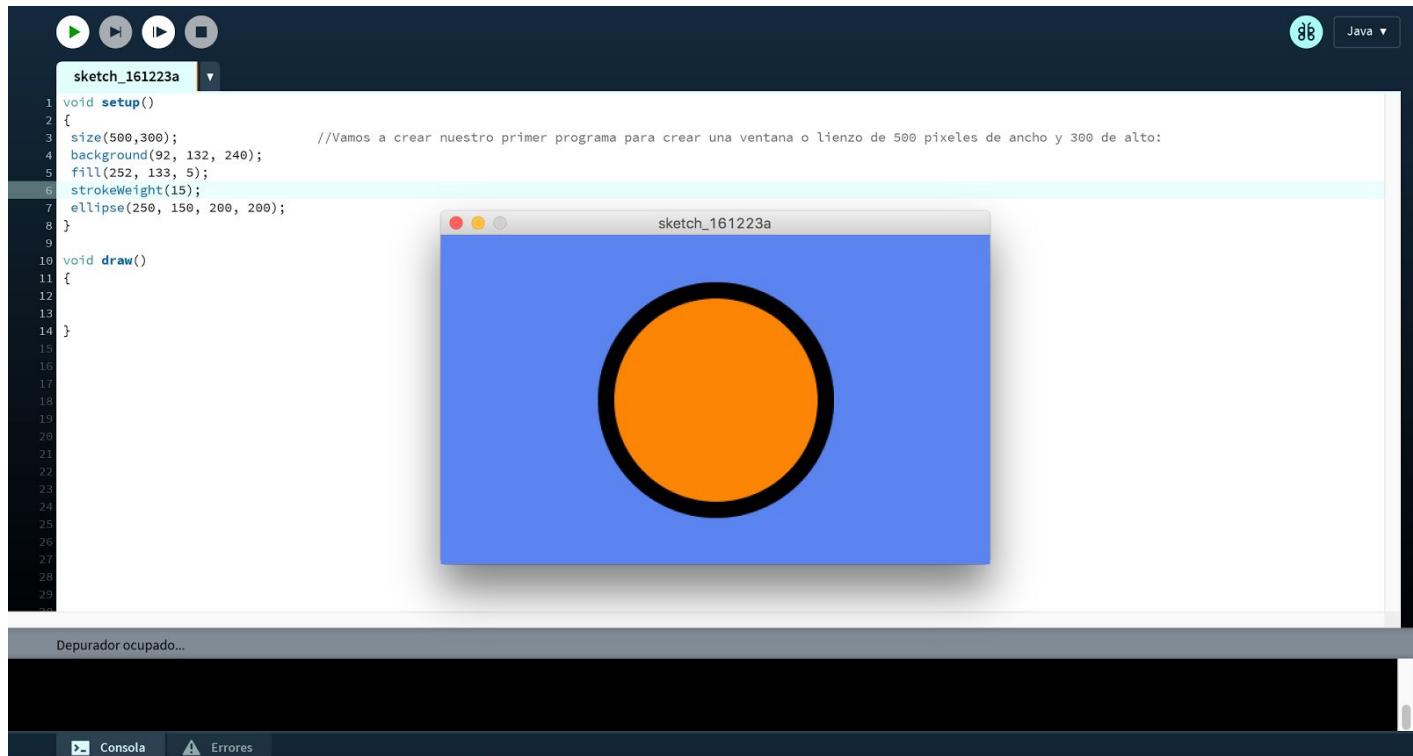
Depurador ocupado...

Consola | Errores

### 8.3. Borde.

Vamos a ponerle el borde al círculo más ancho. Eso lo hacemos con la función:

**strokeWeight(x);**



```
sketch_161223a
1 void setup()
2 {
3     size(500,300);           //Vamos a crear nuestro primer programa para crear una ventana o lienzo de 500 pixeles de ancho y 300 de alto:
4     background(92, 132, 240);
5     fill(252, 133, 5);
6     strokeWeight(15);
7     ellipse(250, 150, 200, 200);
8 }
9
10 void draw()
11 {
12
13
14 }
```

Depurador ocupado...

Consola    Errores

Fíjate, lo ponemos antes de definir la forma a dibujar.

Y ahora le cambiamos el color al borde con la función:

**stroke(red, green, blue);**

sketch\_161223a

```

1 void setup()
2 {
3   size(500,300);           //Vamos a crear nuestro primer programa para crear una ventana o lienzo de 500 pixeles de ancho y 300 de alto:
4   background(92, 132, 240);
5   fill(252, 133, 5);
6   strokeWeight(15);
7   stroke(208, 255, 3);
8   ellipse(250, 150, 200, 200);
9 }
10
11 void draw()
12 {
13
14 }
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
  
```

Depurador ocupado...

Consola    Errores

## 9. Prácticas III.

Realiza los siguientes programas que se proponen a continuación. Guarda la captura de pantalla de cada uno de ellos en un [Documento de texto de Google](#) y súbelo donde corresponda.

### 9.1. Aros olímpicos.

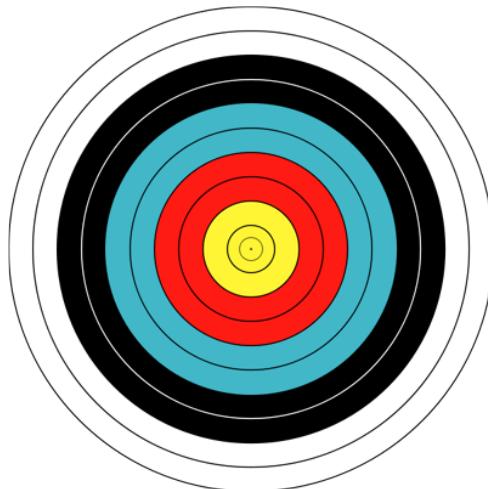
Realiza los aros olímpicos con la forma, distribución y colores correctos.



**Aclaración:** No tienen porqué estar los aros entrelazados, pueden estar los aros superiores por detrás de los inferiores.

### 9.2. Diana.

Realiza un programa en el que se genera la siguiente imagen.



## 10. Variables Processing.

Como en la mayoría de los lenguajes de programación, es imprescindible el uso de datos. Para ello deberemos conocer los tipos de datos que nos ofrece Processing según sea su naturaleza (entero, real, carácter, cadena de caracteres, etc).

En Processing vamos a utilizar los siguientes tipos de datos:

### 10.1. Enteros (INT).

Permiten representar valores numéricos de tipo entero.



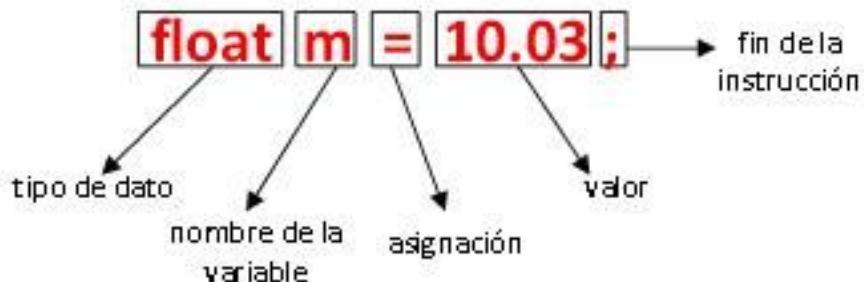
Ejemplos:

```
int x = 60;
```

```
int y = -22;
```

### 10.2. Reales (FLOAT).

Permiten representar valores numéricos de tipo real.



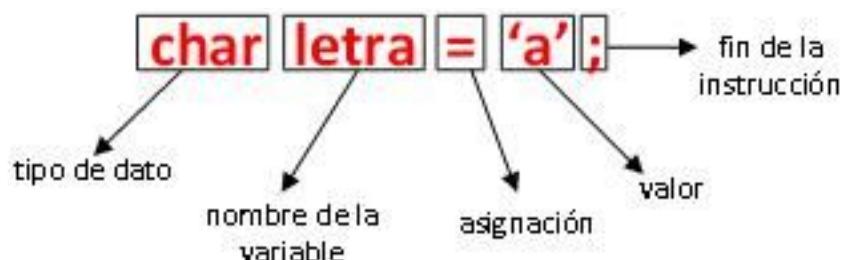
Ejemplos:

`float masa = 45,60;`

`float temperatura = 23,50;`

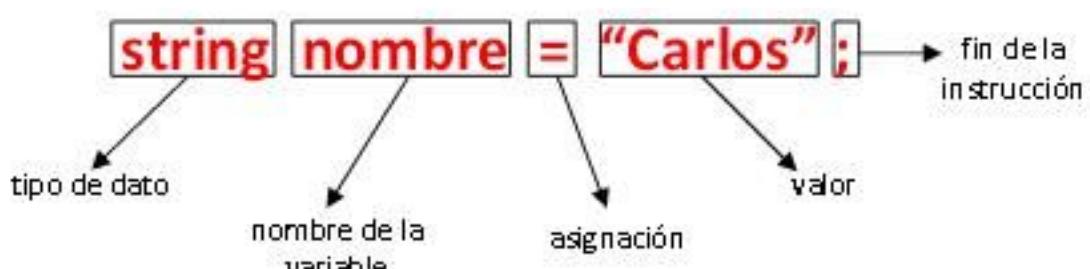
### 10.3. Caracteres (CHAR).

Nos permiten representar caracteres.



### 10.4. Cadena de caracteres (STRING).

Nos permiten representar cadenas de caracteres como palabras, frases, etc.



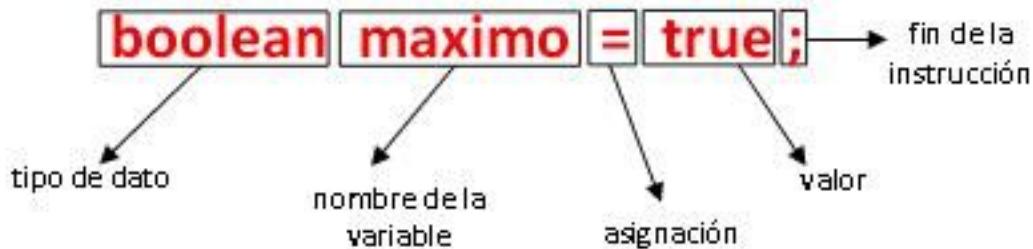
Ejemplos:

`string nombre = "Luis";`

`string apellido1 = "Torreño";`

### 10.5. Valores lógicos (BOOLEAN).

Nos permiten representar valores de tipo lógico (TRUE/FALSE).



Ejemplos:

```
boolean SensorActivado = true;
```

```
boolean AlarmaMotor = false;
```

## 10.6. Variables del Sistema

Además de las variables que hemos visto hasta ahora, que son definidas y creadas por nosotros según necesidades, existen unas variables asociadas a parámetros del sistema que nos serán muy útiles en determinadas circunstancias. Ejemplos de éstas son:

**width**: ancho de la ventana.

**height**: alto de la ventana.

**mouseX**: posición del ratón en el eje X.

**mouseY**: posición del ratón en el eje Y.

**key**: variable char que almacena la última tecla pulsada.

**keyPressed**, **keyReleased**, **mousePressed**, **mouseButton**, **mouseReleased**: Variables Boolean que reaccionan a las acciones del teclado o el ratón como presionar una tecla o el ratón o soltarlos.

## 11. Mostrando información en Processing.

Para mostrar información en Processing lo podemos hacer de dos maneras:

- En la propia consola del IDE.
- En la ventana o lienzo.

### 11.1. En la consola del IDE.

Para ello utilizaremos las funciones:

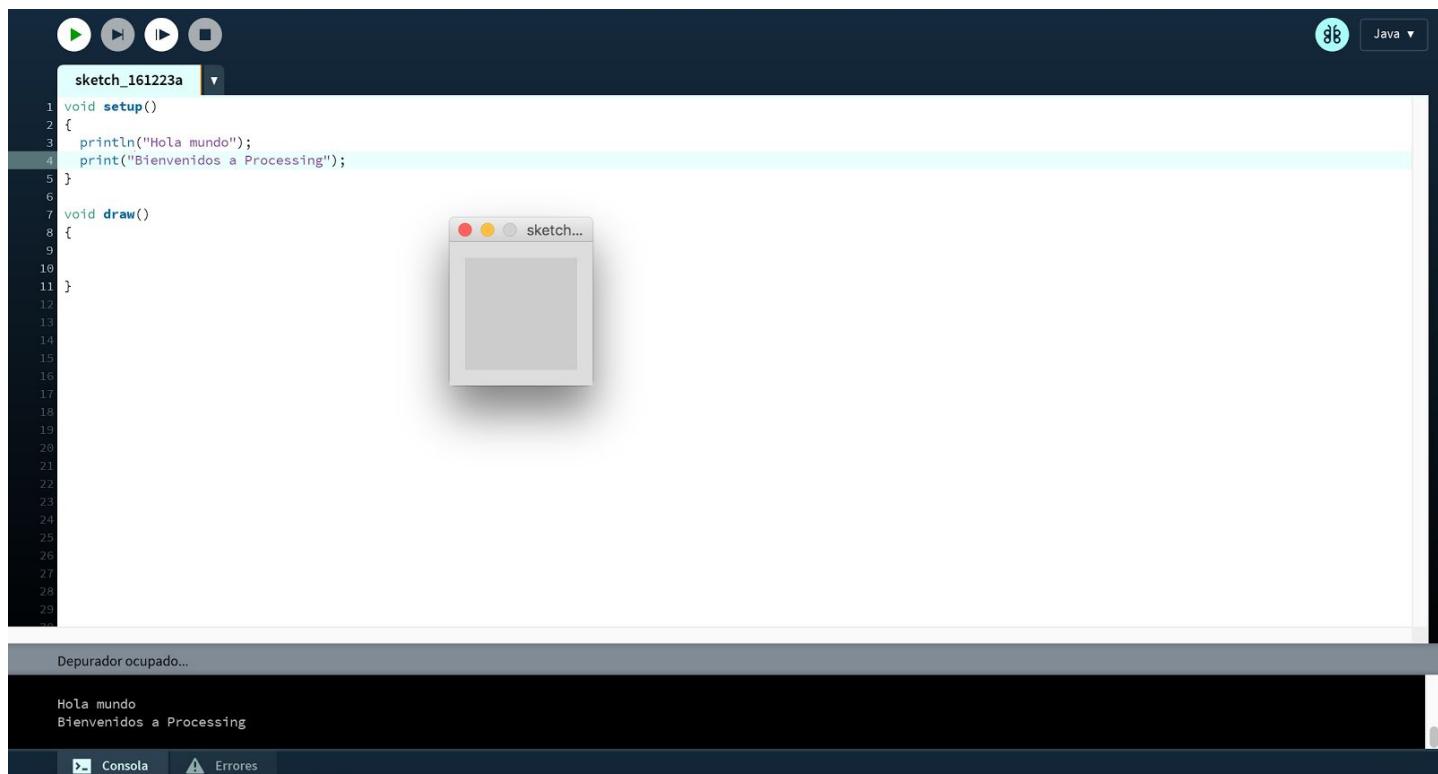
```
print(); y println();
```

Ambas imprimen la cadena que pongamos dentro de los paréntesis en la consola del IDE de Processing. La diferencia entre ambas es que **println()** inserta un salto de línea después de la cadena a imprimir.

Ejemplo:

```
println("Hola mundo");
```

```
print("Bienvenidos a Processing");
```



Este método nos puede servir para mostrar mensajes de confirmación o chequeo de cara al programador o al usuario, pero este formato no es muy estético.

Cuando queramos mostrar una información y poder darle un formato más complejo (estilo, tamaño, color, etc...) utilizaremos los siguientes métodos.

## 11.2. En la ventana o lienzo.

De esta manera vamos a poder **cambiar el tamaño de la fuente**.

Esto lo haremos con la función:

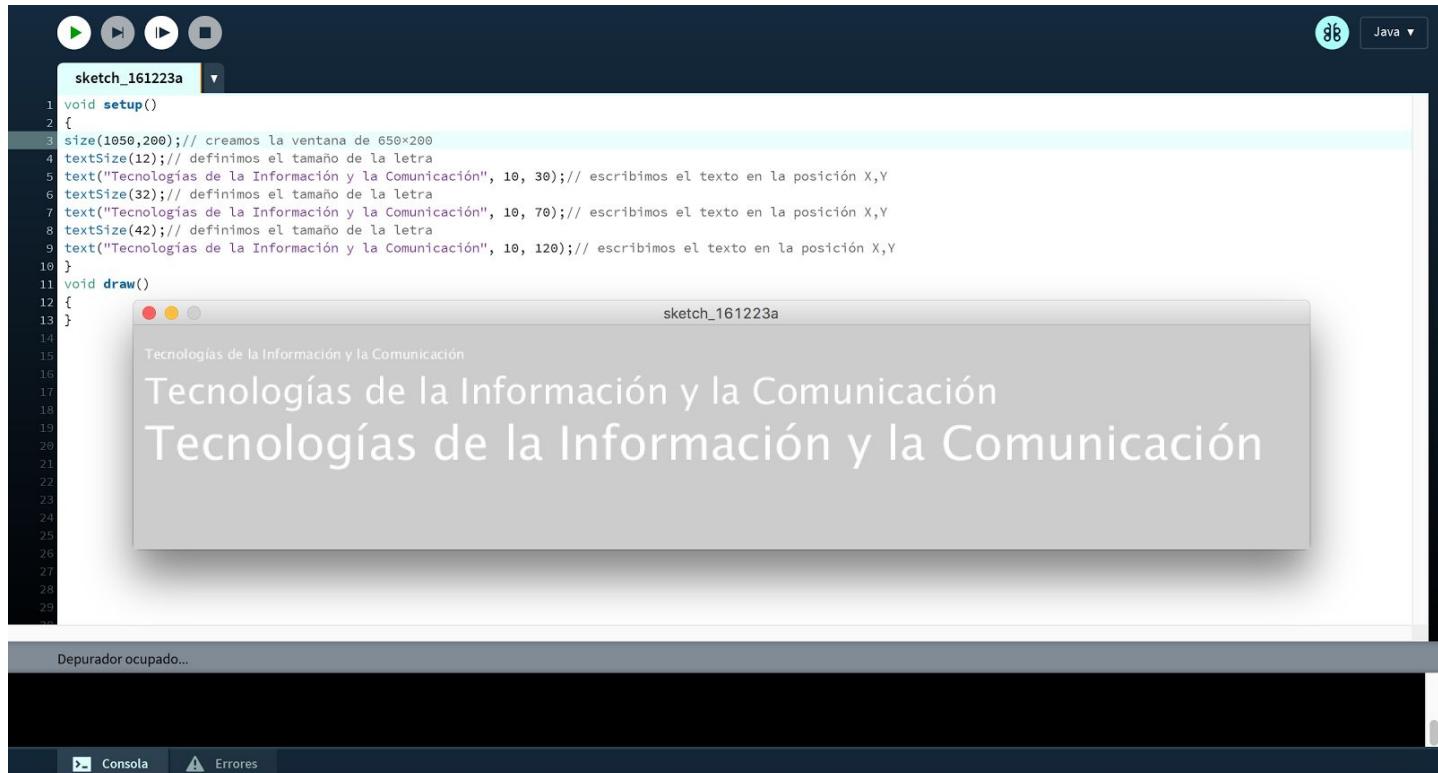
```
textSize(x);
```

Probemos el siguiente ejemplo:

```
void setup()
{
size(650,200); // creamos la ventana de 650x200
textSize(12); // definimos el tamaño de la letra
text("Tecnologías de la Información y la Comunicación", 10, 30); // escribimos el texto en la posición X,Y
textSize(32); // definimos el tamaño de la letra
text("Tecnologías de la Información y la Comunicación", 10, 70); // escribimos el texto en la posición X,Y
textSize(42); // definimos el tamaño de la letra
text("Tecnologías de la Información y la Comunicación", 10, 120); // escribimos el texto en la posición X,Y
}
```

{  
}

...y obtenemos el siguiente resultado:

**Cambiar posición del texto.**

Esto lo haremos con los parámetros X e Y de la función:

**text("texto", X, Y);**

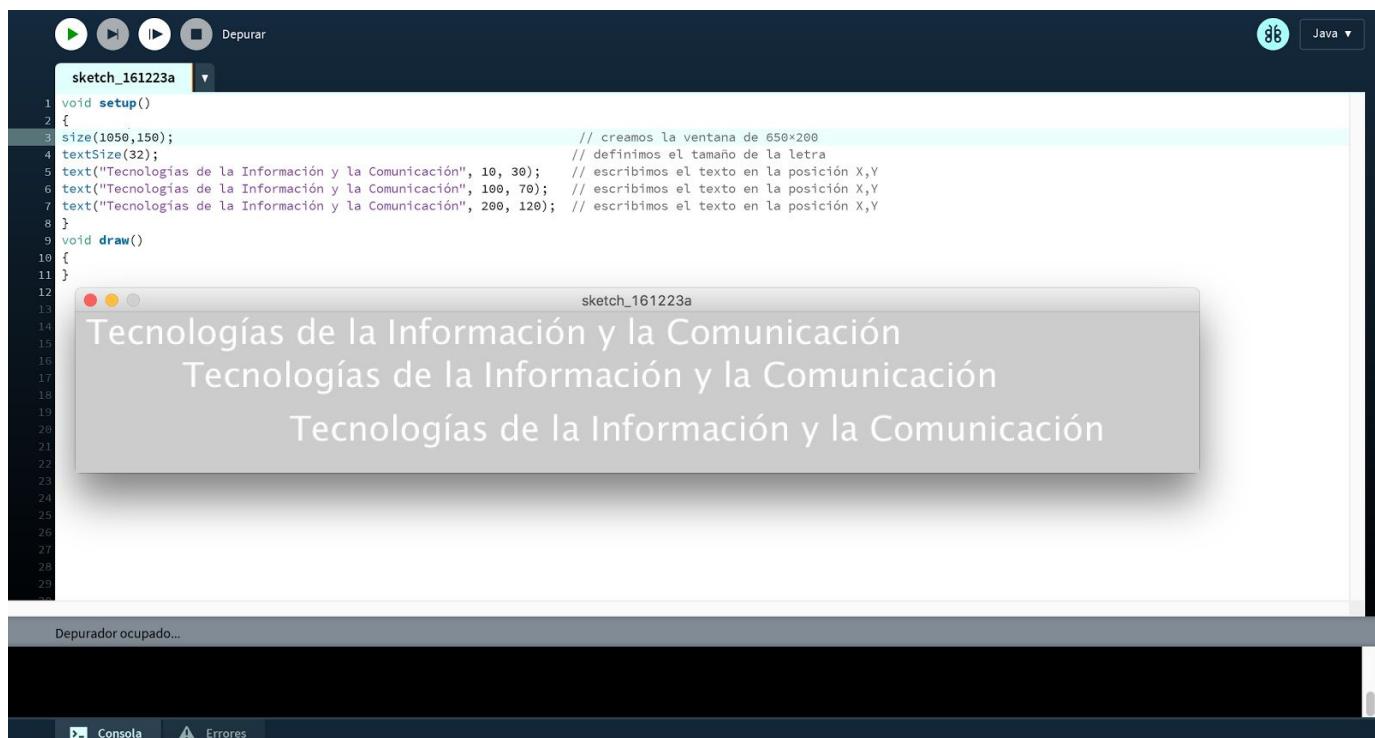
Probemos el siguiente ejemplo:

```

void setup()
{
size(700,150); // creamos la ventana de 650x200
textSize(32); // definimos el tamaño de la letra
text("Tecnologías de la Información y la Comunicación", 10, 30); // escribimos el texto en la posición X,Y
text("Tecnologías de la Información y la Comunicación", 100, 70); // escribimos el texto en la posición X,Y
text("Tecnologías de la Información y la Comunicación", 200, 120); // escribimos el texto en la posición X,Y
}
void draw()
{
}

```

Y obtenemos el siguiente resultado:



### Cambiar el color de la fuente.

Esto lo haremos con la función:

**fill(R, G, B); o fill(R, G, B, transparencia);**

R: componente RED (de rojo)

G: componente GREEN (de verde)

B: componente BLUE (de azul)

Probemos el siguiente ejemplo:

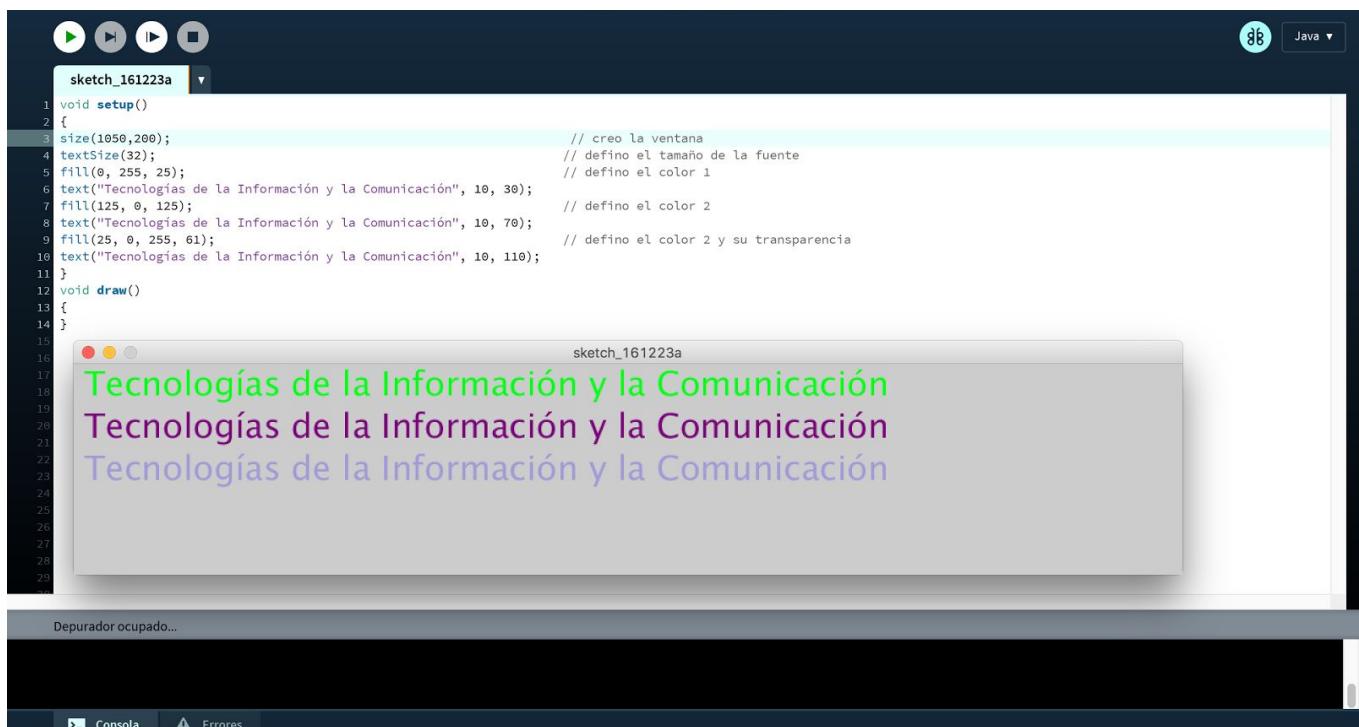
```

void setup()
{
size(650,200); // creo la ventana
textSize(32); // defino el tamaño de la fuente
fill(0, 255, 25); // defino el color 1
text("Tecnologías de la Información y la Comunicación", 10, 30);
fill(125, 0, 125); // defino el color 2
text("Tecnologías de la Información y la Comunicación", 10, 70);
fill(25, 0, 255, 61); // defino el color 2 y su transparencia
text("Tecnologías de la Información y la Comunicación", 10, 110);
}
void draw()
{
}

```

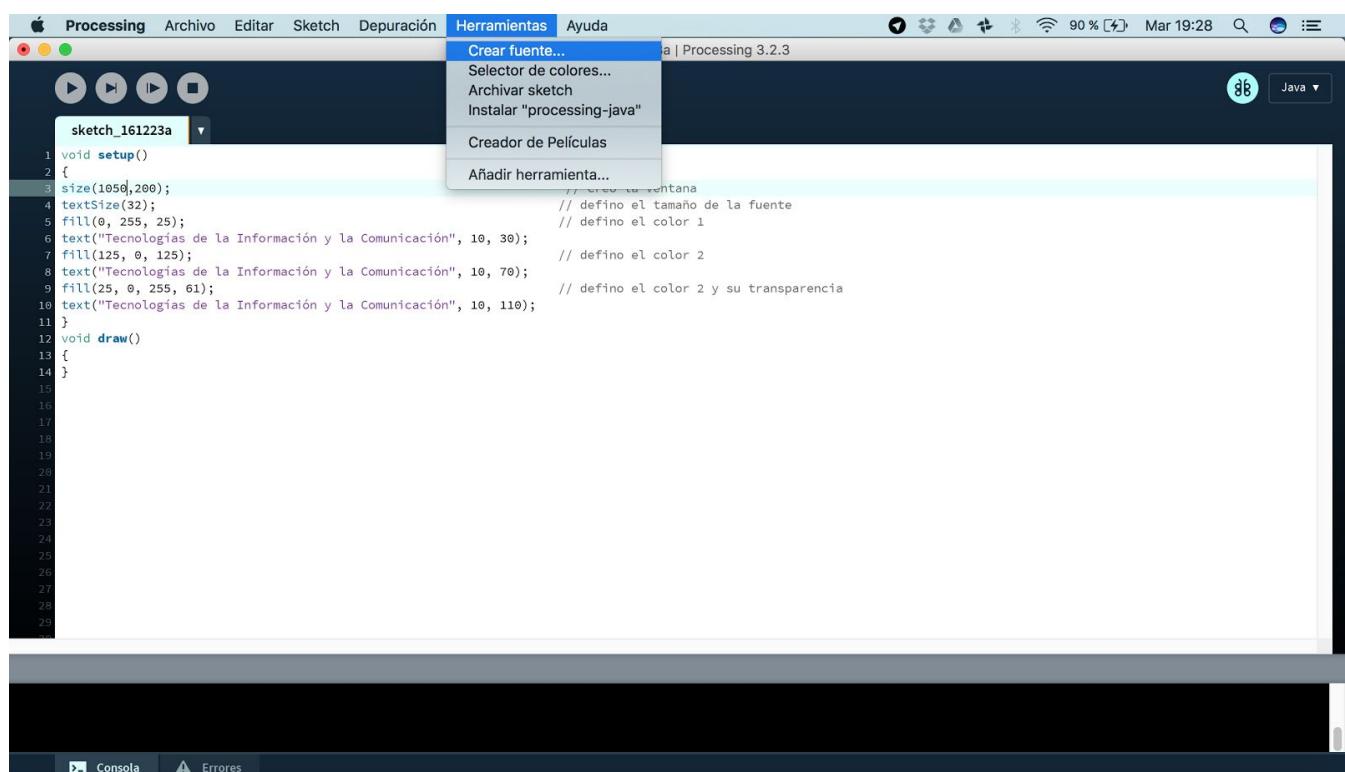
Y obtenemos el siguiente resultado:





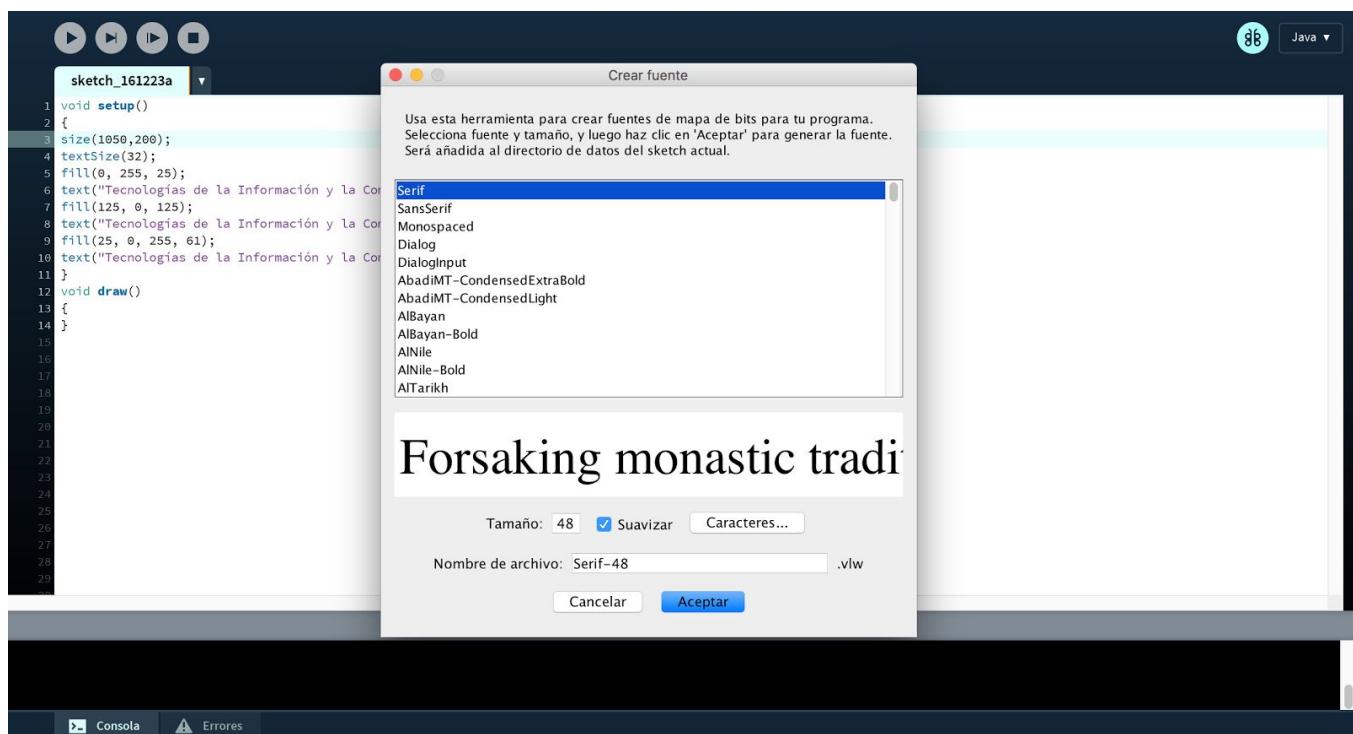
### Cambiar el estilo o fuente.

Para incorporar una nueva fuente a nuestro proyecto debemos hacerlo desde el menú principal: "Herramientas → Crear fuente..." .



Nos aparecerá el listado con todas las fuentes posibles. **La seleccionamos y le damos a “Aceptar”**. De esta manera la incorporará en la carpeta “Data” de nuestro proyecto. **Deberemos hacer ésto por cada fuente distinta que utilicemos en nuestro proyecto.**

Además, tenemos que fijarnos en el nombre del fichero con extensión .vlw que será el que deberemos invocar en nuestro programa (como veremos a continuación).



Veamos un ejemplo a continuación para alternar entre varios tipos de fuente:

```
void setup()
{
PFont mono; // creo una variable de tipo fuente de letra
mono = loadFont("Copperplate-48.vlw"); // cargo el primer estilo de fuente
textFont(mono); // creamos la ventana
size(650,200); // definimos el tamaño de la fuente
textSize(32); // definimos el color de la fuente
fill(255, 0, 0); //escribimos el texto con la fuente por
text("Tecnologías de la Información y la Comunicación", 10, 30); //defecto
mono = loadFont("Bauhaus93-48.vlw"); // cargo el primer estilo de fuente
textFont(mono);
text("Tecnologías de la Información y la Comunicación", 10, 70);
mono = loadFont("CurlzMT-48.vlw"); // cargo el segundo estilo de fuente
textFont(mono);
text("Tecnologías de la Información y la Comunicación", 10, 110);
}

void draw()
{}
```

```

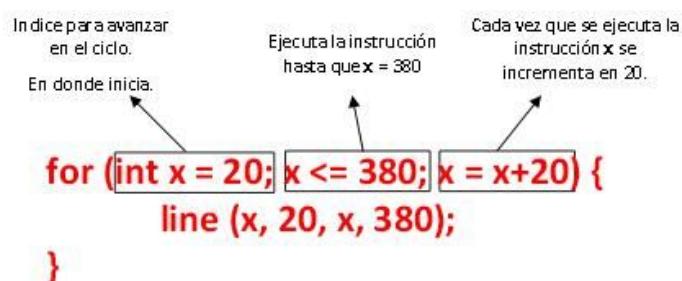
1 void setup()
2 {
3     PFont mono;
4     mono = loadFont("Copperplate-48.vlw");
5     textSize(32);
6     fill(255, 0, 0);
7     text("Tecnologías de la Información y la Comunicación", 10, 30);
8     mono = loadFont("Bauhaus93-48.vlw");
9     textSize(110);
10    text("Tecnologías de la Información y la Comunicación", 10, 110);
11    mono = loadFont("CurlzMT-48.vlw");
12    textSize(70);
13    text("Tecnologías de la Información y la Comunicación", 10, 70);
14    mono = loadFont("Bauhaus93-48.vlw");
15    textSize(110);
16 }
17 void draw()
18 {
19 }

```

## 12. Iteraciones: bucle for ()

En ésta y en la siguiente práctica vamos a ver unas estructuras iterativas o repetitivas que se suelen utilizar en todos los lenguajes de programación para realizar una determinada instrucción que se repite un determinado número de veces (bajo unas determinadas condiciones) mediante una única función dentro de un bucle o loop.

Veamos cómo funciona el bucle for():



Y comprobamos el resultado que se obtiene al insertarlo en el siguiente programa:

```

void setup()
{
    size(400,400);
    for(int x=20; x<=380; x=x+20)
    {
        line(x,20,x,380); //dibuja una línea vertical
    }
}

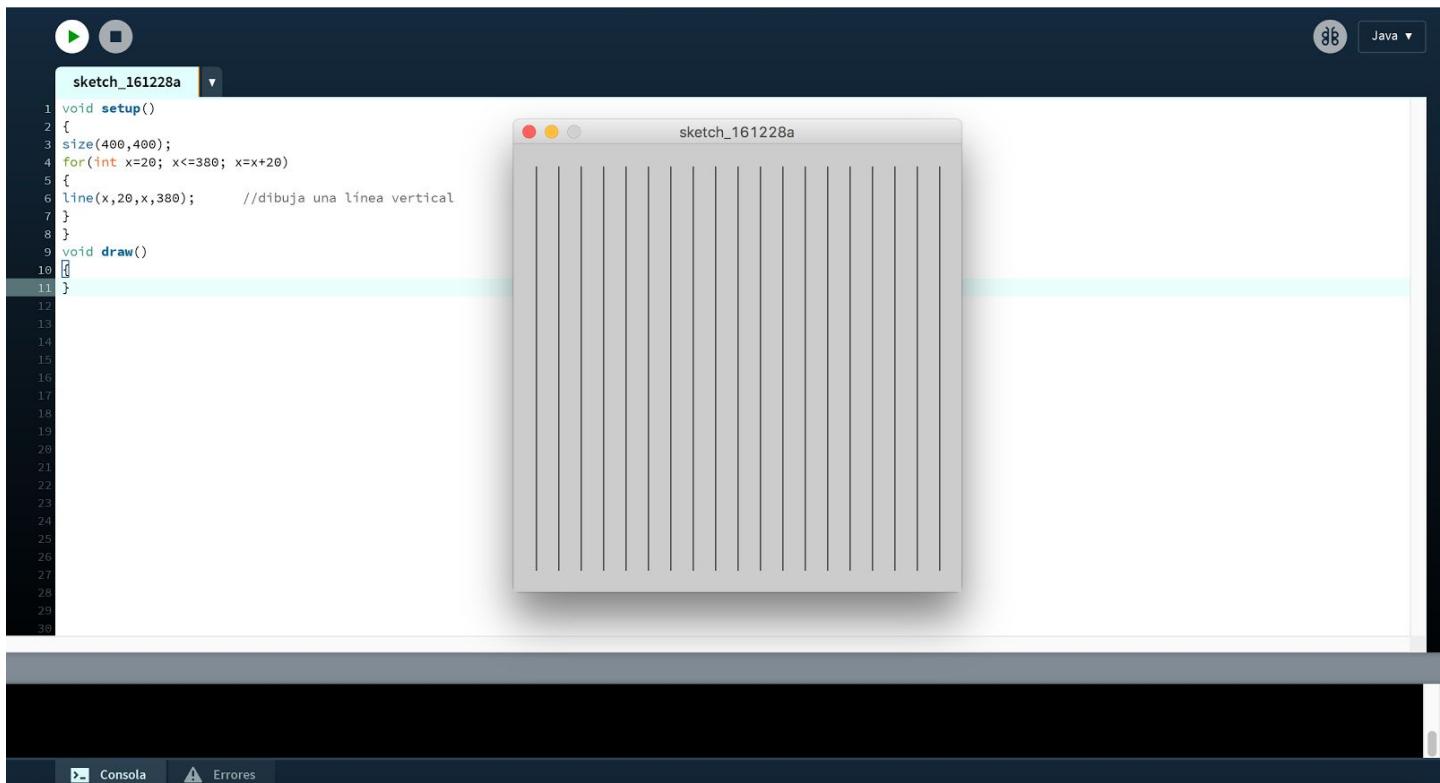
```

```

}
}

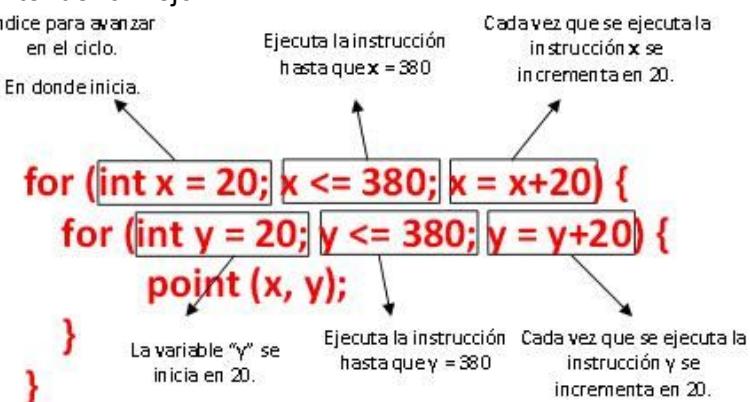
void draw()
{
}

```



Además, contamos con la posibilidad de anidar varios bucles **for()**, lo cual nos permite no sólo trabajar en una dimensión, sino en dos o más. Ésto nos será muy útil cuando trabajemos con matrices en próximas prácticas.

Veámos un ejemplo para entenderlo mejor...



Lo metemos dentro de un programa y obtenemos el siguiente resultado:

The screenshot shows the Processing IDE interface. On the left, the code editor displays a sketch named 'Ejemplos\_for' with the following code:

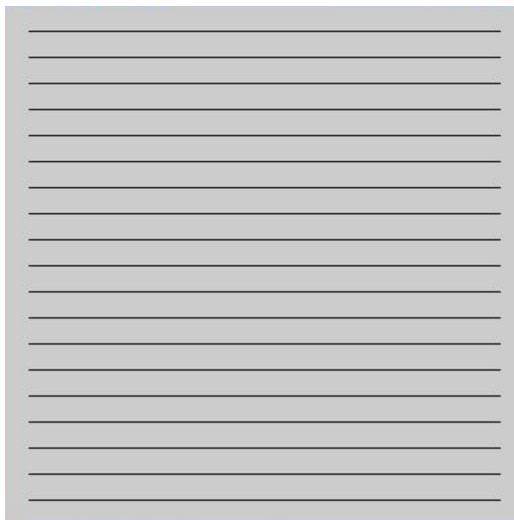
```
1 void setup()
2 {
3     size(400,400);
4     for(int x=20; x<=380; x=x+20)      // con este for() recorro las columnas
5     {
6         for(int y=20; y<=380; y=y+20)    // con este for() recorro las filas
7         {
8             point(x,y);
9         }
10    }
11 }
12 void draw()
13 {
14 }
15 }
```

The code uses nested loops to draw a grid of points. The outer loop iterates over columns (x from 20 to 380), and the inner loop iterates over rows (y from 20 to 380). A preview window titled 'Ejemplos\_for' shows a 20x20 grid of small black dots on a white background.

## 13. Prácticas IV.

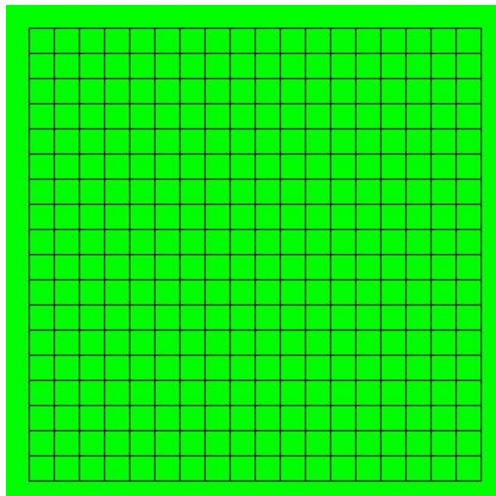
### 13.1. Líneas horizontales.

Realiza un programa utilizando el bucle **for()** para obtener el siguiente resultado:



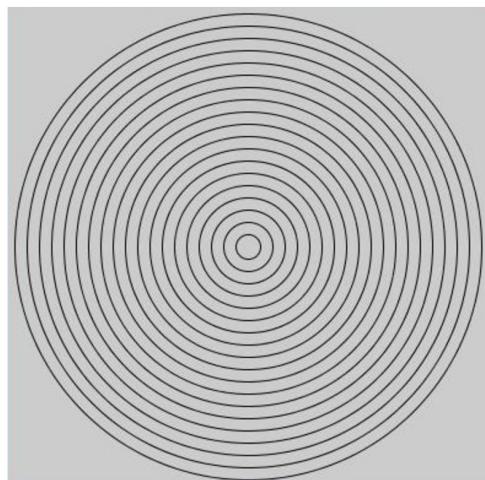
### 13.2. Malla sobre fondo verde.

Realizar un programa utilizando el bucle **for()** para obtener la siguiente rejilla sobre fondo verde:



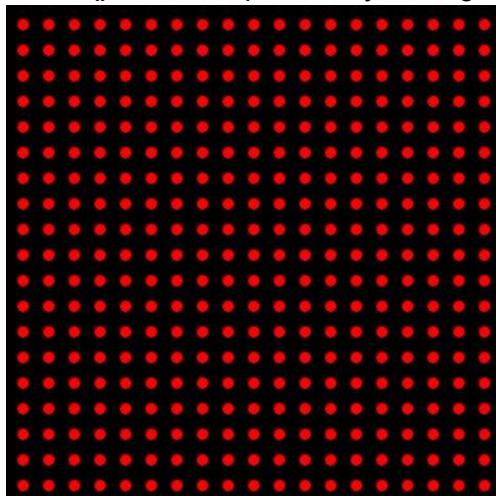
### 13.3. Círculos concéntricos.

Realizar un programa utilizando el bucle **for()** para obtener círculos concéntricos separados una determinada distancia y que genere el siguiente resultado:



### 13.4. Matriz de círculos rojos.

Realizar un programa utilizando bucles **for()** anidados para dibujar la siguiente matriz de círculos rojos:



## 14. Iteraciones: bucle While () .

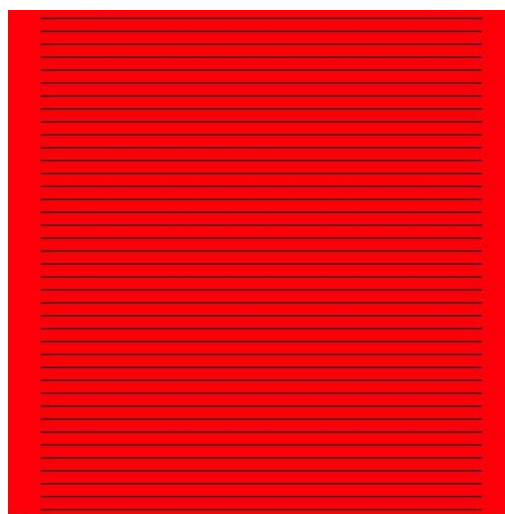
El bucle while() también nos permite realizar una serie de operaciones similares según que se cumpla una determinada condición.

La estructura es la siguiente:

```
int i = 0; //declaración de la variable contador
while (condición que se debe cumplir)
{
    instrucciones a realizar;
    i = i + 10; //incremento de la variable contador
}
```

Veámos un ejemplo concreto:

```
void setup()
{
    size(400,400); //Creamos la ventana
    background(255,0,10); //definimos color de fondo
    int i = 0; // creamos la variable que va a hacer de contador
    while (i < 400) { //evaluamos la condición
        line(30, i, 370, i); //dibuja una línea horizontal
        i = i + 10; //incrementa la variable contador en 10 unidades
    }
}
void draw()
{}
```



## 15. Interacciones con el ratón.

En esta práctica vamos a ver cómo Processing nos permite ejecutar acciones al detectar determinados eventos del ratón (como cuando es pulsado, soltado, arrastrado, obtener sus coordenadas sobre pantalla, etc).

Para ello disponemos de las siguientes funciones:

**mouseX, mouseY**

Nos da la coordenada X e Y (respectivamente) de donde se encuentra situado el cursor del ratón.

Ejemplo:

```
void setup()
{
    size(400,400);
    background(255,0,10);
}
void draw()
{
    ellipse(mouseX, mouseY, 80, 80);
}
```

Resultado:

[Vídeo con el resultado.](#)

En el vídeo podemos apreciar cómo se va formando un halo a lo largo de la trayectoria del puntero del ratón. Esto se debe a que se van superponiendo todos los círculos dibujados, y esto es porque no refrescamos el fondo con la función **background()** cada vez que dibujamos un círculo.

La solución está pasando la función **background()** del bucle **setup()** al bucle **draw()**. De esta manera se limpia el fondo antes de dibujar cada círculo.

```
void setup()
{
    size(400,400);
}
void draw()
{
    background(255,0,10);
    ellipse(mouseX, mouseY, 80, 80);
}
```

Resultado:

[Vídeo con el resultado.](#)



Se puede apreciar ahora cómo el círculo va siguiendo la trayectoria del ratón pero sin dejar rastro de los círculos dibujados anteriormente.

## 16. Generación de números aleatorios.

En esta práctica vamos a ver cómo podemos generar números de forma aleatoria. Para ello utilizaremos la función:

**random(x);**

La función random(x) nos devuelve un valor comprendido entre el rango 0 y x-1. Por ejemplo, si pusiéramos random(256); nos devolvería cada vez que la llamáramos un valor entre el 0 y el 255.

Además, si le pasamos dos parámetros:

**random(x,y);**

el rango pasaría a ser entre el valor x y el y-1.

Por ejemplo, si escribiéramos random(25,101); nos devolvería un valor entre el 25 y el 100.

Analicemos estos dos ejemplos y veamos sus resultados:

### Ejemplo 1:

```
void setup()
{
    size(400,400);
}
void draw()
{
    float ancho = random(80);
    ellipse(random(400),random(400),ancho,ancho);
}
```

Resultado:

[Vídeo con el resultado.](#)

### Ejemplo 2:

```
void setup()
{
    size(400,400);
}
void draw()
{
    float ancho = random(80);
    fill(random(255),random(255),random(255));
    ellipse(random(400),random(400),ancho,ancho);
}
```



---

Resultado:

[Vídeo con el resultado.](#)

## 17. Prácticas V.

### 17.1. Dibujando con el ratón.

Realiza un programa en una ventana de 480x270 en la que se pueda dibujar con el ratón, es decir, que al mover el ratón dentro de ella vaya dibujando el trazo.

### 17.2. Ventana con color de fondo cambiante.

Haz un programa que genere una ventana de 500x500 y cuyo fondo vaya cambiando de color de forma aleatoria.

## 18. Estructuras condicionales.

Es muy habitual en programación tener que tomar decisiones en función del valor de una variable. Para eso existen bloques de código llamados condicionales los cuales son la sentencia if y la switch. En esta práctica veremos el funcionamiento y la sintaxis de estas sentencias.

```
if (condición)
{
    código a ejecutar si es verdadero
}
else
{
    código a ejecutar si es falso
}
```

La sentencia if evalúa una expresión y en caso de que sea verdadera se ejecuta el código entre las llaves del if, en caso contrario se ejecuta el código entre llaves después del else. Si solo necesitamos la parte del if es posible ignorar la parte del else y no escribirla. Además, si el código a ejecutar tanto en el if como en el else se compone de una única instrucción, también nos podemos ahorrar las llaves de apertura y cierre.

Para evaluar las condiciones podemos usar los siguientes operadores:

- == igual a
- != diferente de



- > mayor que
- < menor que
- >= mayor o igual que
- <= menor o igual que

por ejemplo tenemos el siguiente código:

---

```
int x=7;
if (x>=5)
{
    println(x+" es mayor o igual a 5");
}
else
{
    println(x+" es menor a 5");
}
```

---

En este caso el valor de x es 7 por lo que la condición es verdadera, entonces se imprimirá en pantalla “7 es mayor o igual a 5”, intenta cambiar el valor de x y observa los resultados.

También es posible usar if's anidados para evaluar distintas condiciones, vamos a modificar el ejemplo anterior añadiendo otra condición:

---

```
int x=3;
if (x>=10)
{
    println(x+" es mayor o igual a 5");
}
else if(x>3)
{
    println(x+" es mayor a 3");
}
else
{
    println(x+" es menor que o igual a 3");
}
```

---

En este caso nuestra variable x vale 3, entonces llegamos al if pero como  $3>=10$  es falso, saltamos al else if donde ahora compara si  $3>3$  de nuevo esta expresión es falsa por lo que al final se ejecutará el código dentro del else donde sabemos que 3 es menor o igual a 3.

También existe otra forma de evaluar condiciones, usando la sentencia switch que es muy parecida al else if, se recomienda el uso de switch cuando se tienen 3 o más alternativas, también se usa comúnmente para hacer menús, sus sintaxis es la siguiente.

switch(var)



```

{
    case etiqueta1:
        código a ejecutar
    case etiqueta2:
        código a ejecutar
    default:
        código a ejecutar
}

```

El switch funciona de la siguiente manera, se evalúa var entre paréntesis que en este caso puede ser un **int**, **char** o **byte**, dependiendo del valor de var se ejecutará el código entre cada caso, si var coincide con la **etiqueta 1** se ejecutará el código del primer caso, si coincide con la **etiqueta 2** se ejecuta el segundo caso y si no coincide con ninguna se ejecutará el default. Veamos un ejemplo sencillo:

```

int x=1;
switch(x)
{
case 3:
println("Se ejecutó el caso x=3");
break;
case 7:
println("Se ejecutó el caso x=7");
break;
case 9:
println("Se ejecutó el caso x=9");
break;
default:
println("Se ejecutó el caso por default");
break;
}

```

En el ejemplo se evalúa el valor de x que tiene el valor de 1, ya que no coincide con ninguno de los 3 casos se ejecuta el código por default, ¿notaste el comando break? Se usa para cuando se quiere salir de un bloque condicional o ciclo (que veremos después) cuando se termina de ejecutar el código del caso llega al break que hace que el programa salga del switch y continúe con lo siguiente. Para utilizar el switch con un tipo de variable **char** se debe colocar el carácter entre comillas simples por ejemplo: case 'A'.

Para finalizar, cabe destacar que el funcionamiento y sintaxis vistas en este tutorial funcionan no solo para Processing sino también para otros lenguajes de programación.

## 19. Interacción con el teclado.

En esta práctica vamos a aprender cómo podemos actuar en función de eventos que se produzcan en nuestro

teclado.

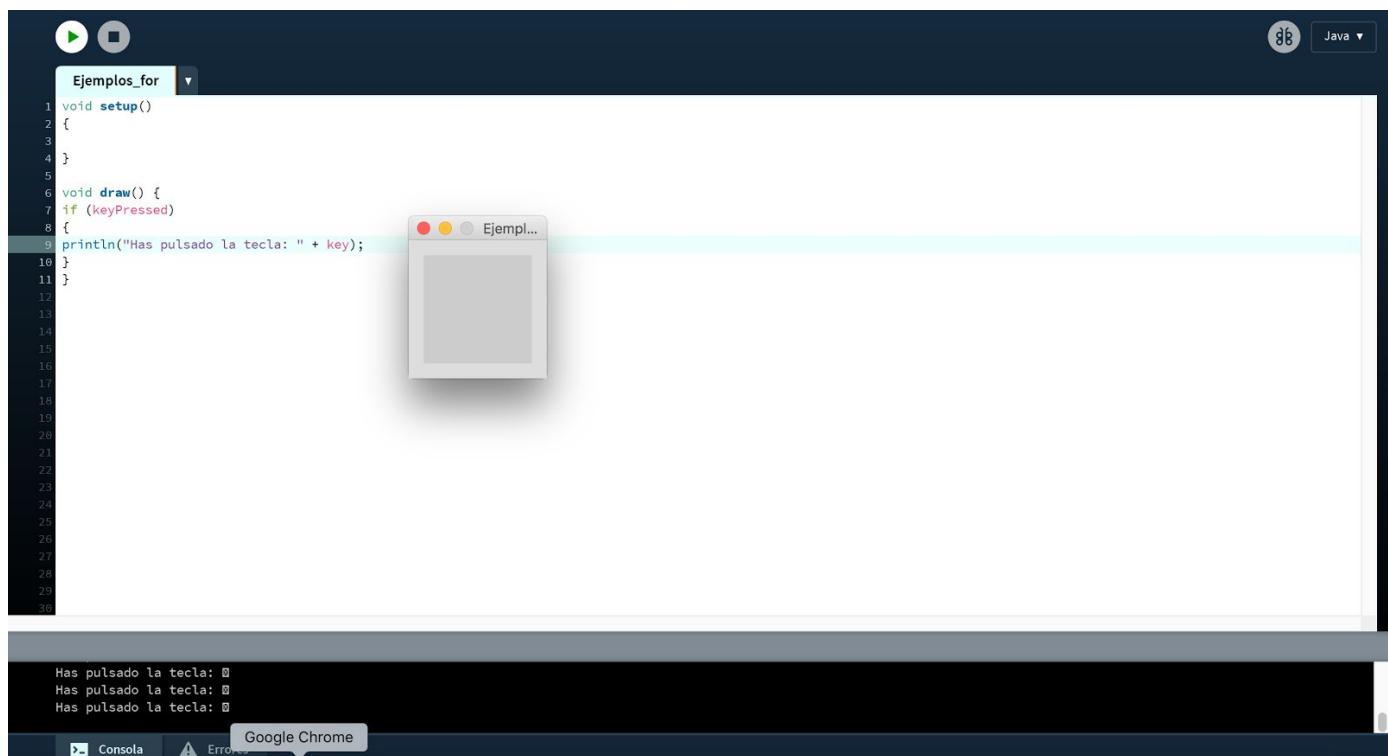
Para saber si una tecla fue presionada podemos utilizar la variable booleana [keyPressed](#), que devuelve el valor **true** si alguna tecla fue pulsada y **false** si no se pulsó ninguna. Por otro lado, tenemos la variable de estado [key](#), que nos devuelve el valor de la última tecla pulsada. La variable key se suele utilizar cuando utilizamos teclas de letras y números, mientras que [keyCode](#) se utiliza cuando utilizamos teclas de símbolos y caracteres especiales (SHIFT, CTRL, UP, DOWN, etc). También es posible implementar las funciones [keyPressed\(\)](#) y [keyReleased\(\)](#) que serán invocadas cada vez que se presione o suelte una tecla respectivamente.

Veamos algunos ejemplos para familiarizarnos con estas funciones:

### Ejemplo 1:

```
void setup()
{
}
void draw() {
if (keyPressed)
{
println("Has pulsado la tecla:" + key);
}
}
```

Este ejemplo nos muestra por la consola del IDE la tecla que vamos pulsando:



[Vídeo con el resultado.](#)

**Ejemplo 2:**

```

void setup()
{
    size(500,500);
}
void draw()
{
    if (key=='r')                                //si pulso la tecla r
    {
        fill(255,0,0);                          //relleno de color ROJO
    }
    if (key=='g')                                //si pulso la tecla g
    {
        fill(0,255,0);                          //relleno de color VERDE
    }
    if (key=='b')                                //si pulso la tecla b
    {
        fill(0,0,255);                         //relleno de color AZUL
    }
    ellipse(250,250,300,300);
}

```

Con este ejemplo, si pulsamos la tecla de la “r”, pintará un círculo rojo, si pulsamos la tecla de la “g”, pintará un círculo verde, y si pulsamos la tecla de la “b”, pintará un círculo azul.

[Vídeo del resultado.](#)

**Ejemplo 3:**

```

void setup()
{
}
void draw() {
    if (keyPressed)
    {
        println("El código de la tecla pulsada es: " + keyCode);
    }
}

```

Con este código averiguamos el código de cualquier tecla de función. Esto nos servirá para el siguiente ejemplo, donde vamos a controlar la posición de un círculo con las teclas de flechas (izquierda, derecha, arriba y abajo).

Si lo ejecutamos, veremos que los códigos son:

- Flecha DERECHA: 39
- Flecha IZQUIERDA: 37



- Flecha ARRIBA: 38
- Flecha ABAJO: 40

#### Ejemplo 4:

```

int x = 300;
int y = 300;
void setup()
{
    size(600,600);
    background(0,0,0);
}
void draw()
{
    if(keyPressed)
    {
        switch(keyCode)
        {
            case 38:           //arriba.....también admite UP
                y--;
                break;
            case 40:           //abajo.....también admite DOWN
                y++;
                break;
            case 39:           //derecha.....también admite RIGHT
                x++;
                break;
            case 37:           //izquierda.....también admite LEFT
                x--;
                break;
            default:
                break;
        }
        background(0,0,0);
        if(x>=575)          //limita la trayectoria hacia la derecha
            x= 575;
        if(x<=25)           //limita la trayectoria hacia la izquierda
            x=25;
        if(y>=575)           //limita la trayectoria hacia abajo
            y= 575;
        if(y<=25)           //limita la trayectoria hacia arriba
            y=25;
        ellipse(x,y,50,50);
        println("x: " + x + ",y: " + y);
    }
}

```

## 20. Prácticas VI.



## 20.1. Jugando con el círculo.

Realizar un programa que dibuje un círculo en el centro de la ventana y que podamos modificar su diámetro mediante las flechas UP (aumentaría diámetro) y DOWN (disminuiría diámetro). Tomar la precaución de que no crezca más que el tamaño de la ventana.

[Vídeo con el resultado.](#)

## 20.2. Estela.

Realiza un programa que dibuje un círculo y con las flechas DERECHA e IZQUIERDA se vaya moviendo pero dejando una estela (otros círculos) detrás de su movimiento.

# 21. Control del tiempo.

En programación es imprescindible muchas veces el controlar cada cuánto tiempo se lanzan eventos o acciones. Para ello Processing utiliza una función que se llama:

**millis()**

Esta función devuelve el tiempo que ha transcurrido desde que se lanza el programa y lo da expresado en milisegundos. De esta manera, si invocamos a esta función dos veces separadas en el tiempo y restamos esos valores, obtendremos el tiempo transcurrido entre esos dos eventos.

Por hacer un simul, es igual a como trabajan los contadores de la electricidad que llega a nuestras viviendas. El contador empezó a contar cuando lo instalaron por primera vez, pero las lecturas de cada factura se hacen restando la lectura actual y la del último mes (de esa manera se calcula lo consumido ese último mes).

Veamos un ejemplo para entenderlo mejor:

---

```
//Se declara una variable que almacenará el tiempo actual (real) transcurrido
//desde que se activa el programa.
```

```
long tiempo = 0;
```

```
//Se declara una variable que almacenará el último valor de tiempo en el que se
//ejecutó la instrucción (delay).
```

```
long t_actualizado = 0;
```

```
//Se declara una variable que almacenará el tiempo que se desea que dure el retardo.
```

```
long t_retardo = 500; // retardo de medio segundo
```

```
void setup()
{
    size(600,600);
    background(0,0,0);
}
```

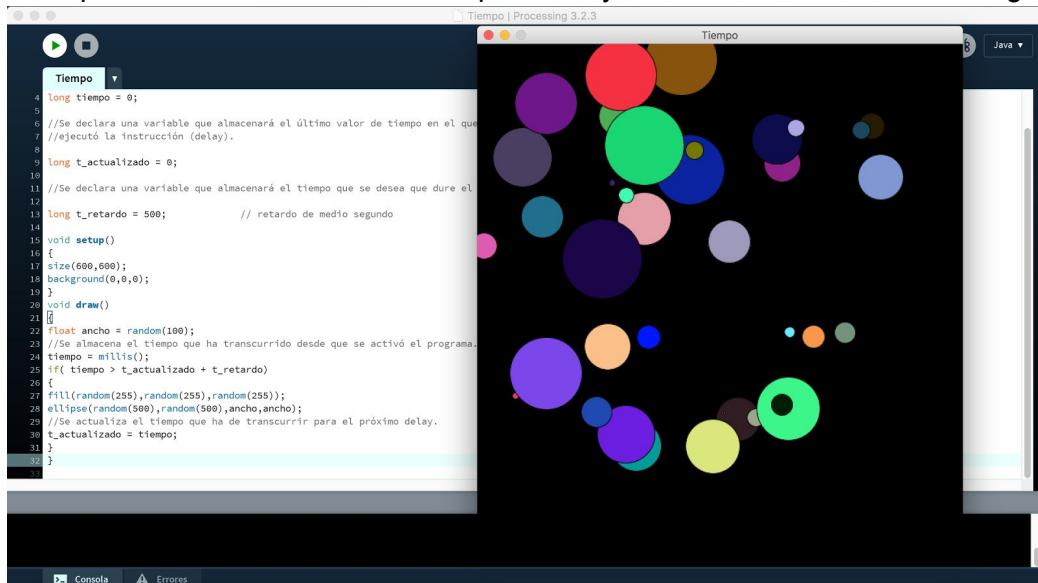


```

void draw()
{
    float ancho = random(100);
    //Se almacena el tiempo que ha transcurrido desde que se activó el programa.
    tiempo = millis();
    if( tiempo > t_actualizado + t_retardo)
    {
        fill(random(255),random(255),random(255));
        ellipse(random(500),random(500),ancho,ancho);
    }
    //Se actualiza el tiempo que ha de transcurrir para el próximo delay.
    t_actualizado = tiempo;
}

```

Vemos cómo van apareciendo círculos de ancho, posición, y color aleatorio cada medio segundo (500 ms):



Probad a modificar el valor de la variable que fija el retardo (`t_retardo`) y observar cómo la animación se acelera o se ralentiza.

## 22. Prácticas VII.

### 22.1. El coche fantástico.

Realiza un programa que simule el movimiento indefinido de un círculo de derecha a izquierda como la característica animación de luces del “Coche Fantástico”.

### 22.2. Semáforo.

Realiza un programa que simule un semáforo. Establece el tiempo que creas conveniente para el cambio de colores. El ámbar debe parpadear varias veces antes de cambiarse a rojo.

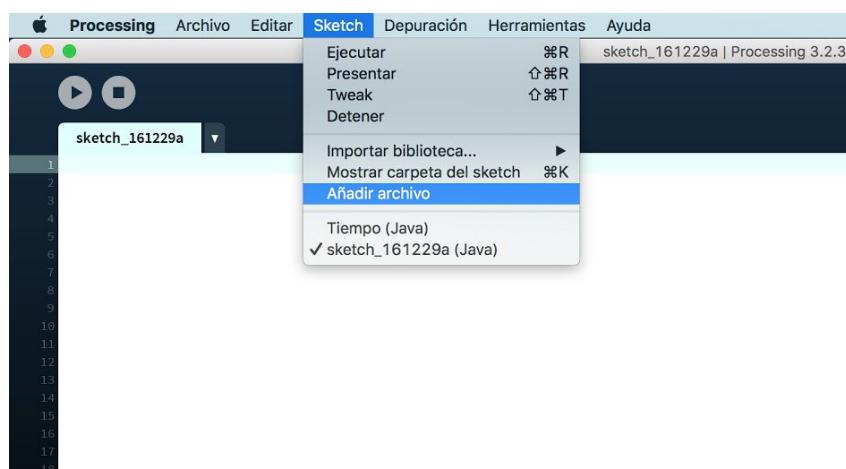
## 23. Cargar una Imagen.

Vamos a aprender a trabajar con imágenes con Processing. Esto puede ser muy útil a la hora de desarrollar un juego o algún Interface de usuario donde queramos una estética más elaborada.

Vamos a ver dos métodos:

### 23.1. La imagen a cargar está en la carpeta de nuestro proyecto (ubicación local).

En este método, lo primero que debemos hacer antes de cargar la imagen, es meterla en la carpeta DATA de nuestro proyecto. Para ello, nos hacemos con la imagen deseada y efectuamos los siguientes pasos:

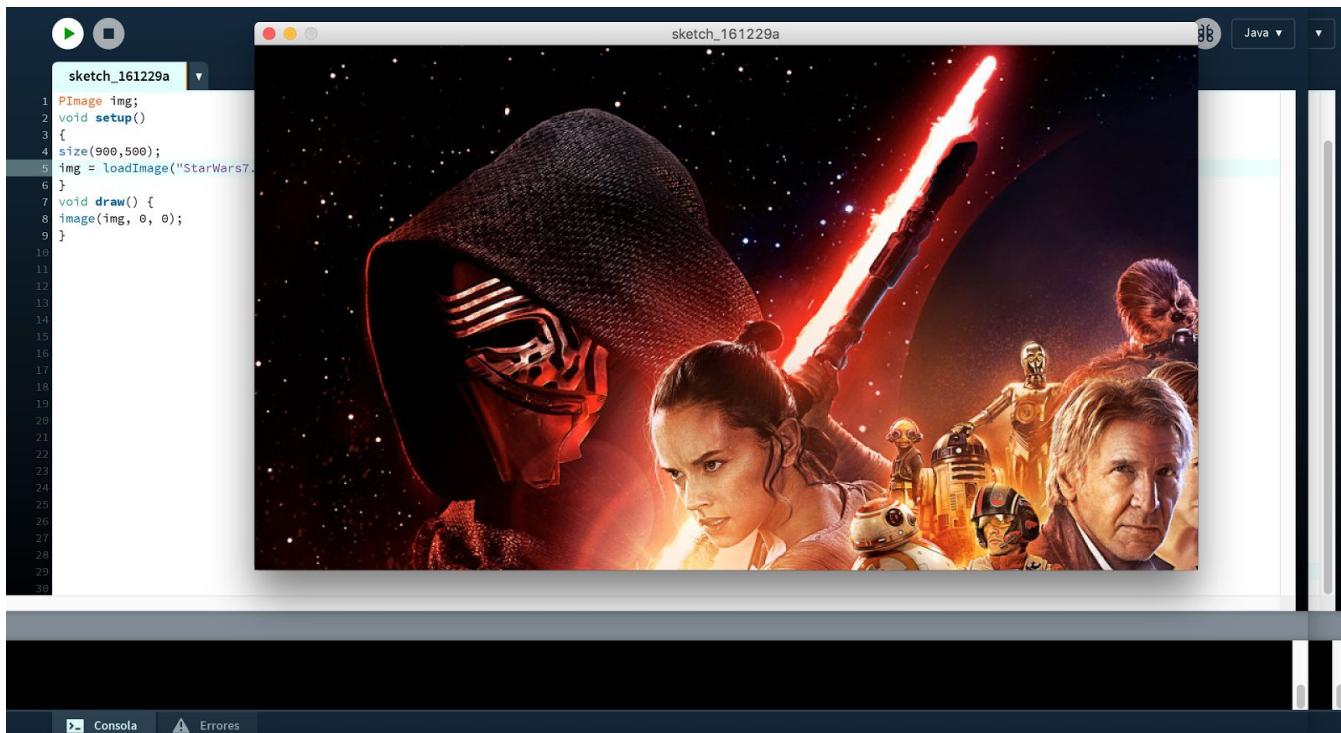


Y seleccionamos de entre las carpetas de nuestro ordenador la imagen que deseamos cargar. Con este paso la imagen ya queda dentro de la carpeta DATA de nuestro proyecto.

Ahora el código a implementar sería el siguiente:

```
PI mage img;
void setup()
{
    size(900,500);
    img = loadImage("robots.jpg");
}
void draw() {
    image(img, 0, 0);
}
```

Lo ejecutamos y obtenemos el siguiente resultado:



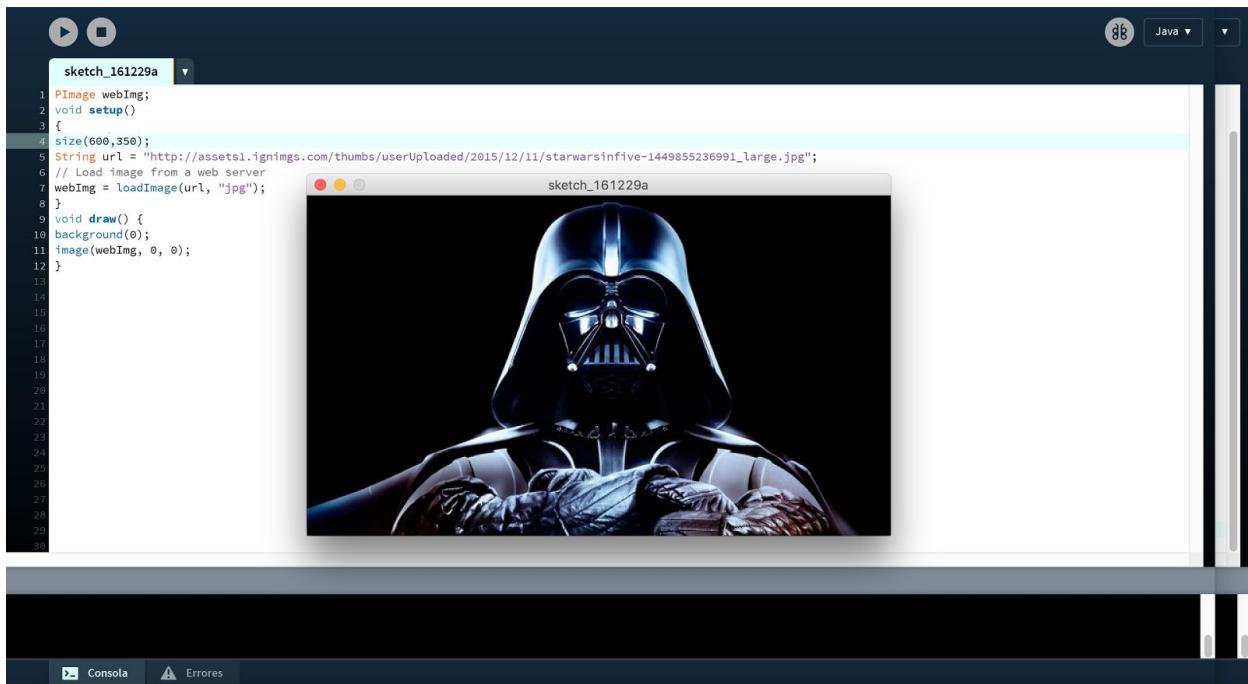
Es importante meternos en las propiedades de la imagen y ver sus dimensiones (ancho y alto) en píxeles, para crear previamente una ventana donde encajar luego la imagen. De no ser así, podría salir la imagen cortada.

## 23.2. La imagen está en Internet (ubicación on-line).

Con este método accedemos a las imágenes que están en Internet a través de su URL. Veámoslo con el siguiente ejemplo:

```
PImage webImg;
void setup()
{
    size(600,350);
    String url =
"http://assets1.ignimgs.com/thumbs/userUploaded/2015/12/11/starwarsinfive-1449855236991_large.jpg";
// Load image from a web server
    webImg = loadImage(url, "jpg");
}
void draw() {
    background(0);
    image(webImg, 0, 0);
}
```

Lo ejecutamos y vemos el resultado:



## 24. Prácticas VIII.

### 24.1. Imagen local.

Realiza un programa que cargue una imagen local.

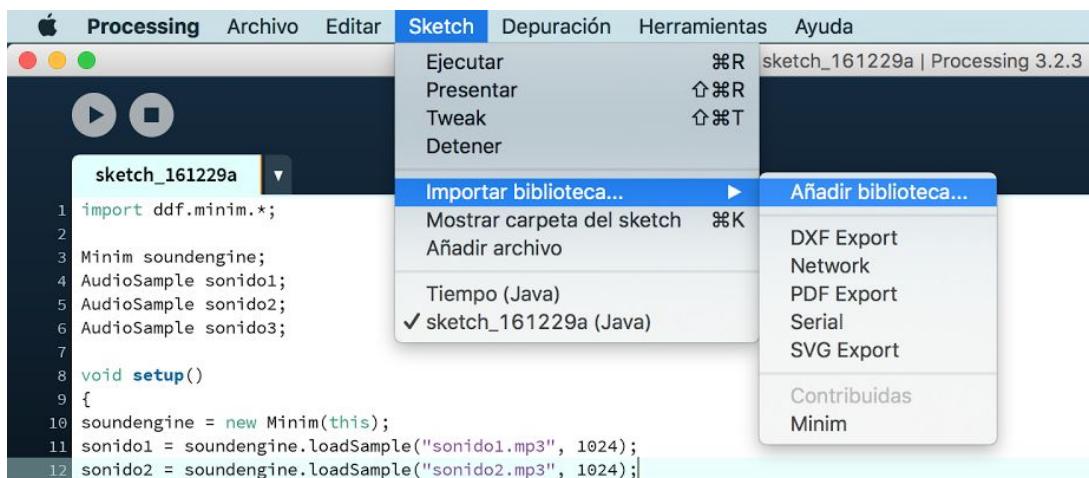
### 24.2. Imagen web.

Realiza un programa que cargue una imagen de Internet.

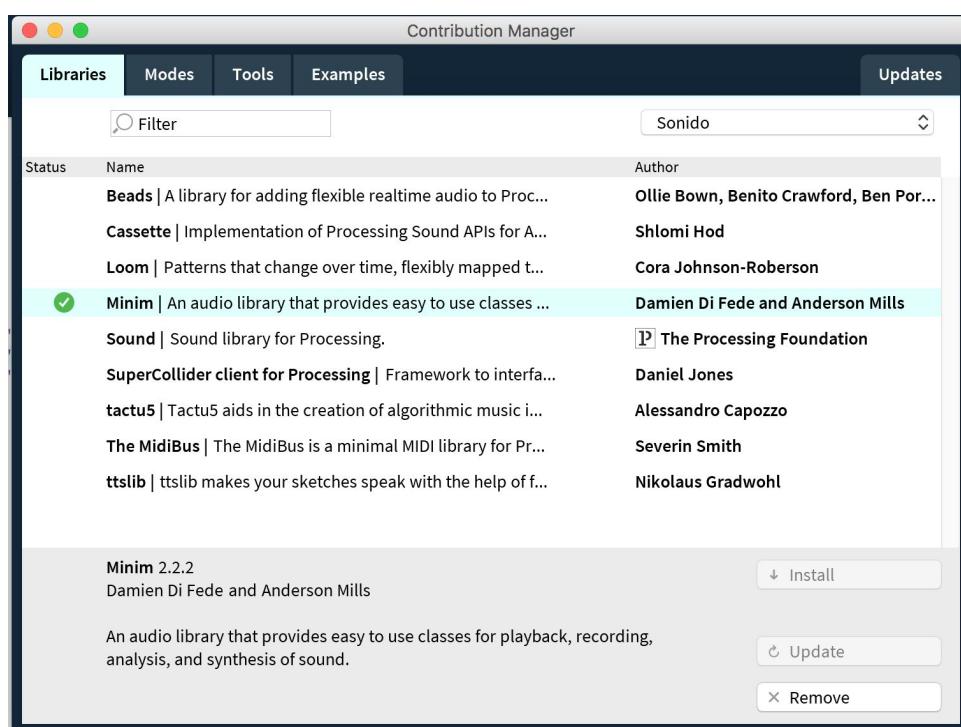
## 25. Control de sonido.

En esta práctica vamos a aprender cómo manejar sonidos con Processing. Existen numerosas librerías que nos proporcionan una gran cantidad de funciones y efectos con sonidos. En esta ocasión vamos a utilizar la librería MINIM.

Lo primero que debemos hacer es iniciar el nuevo proyecto en el IDE de Processing e importar dicha librería. Para ello seguimos los siguientes pasos:



Nos aparece la siguiente pantalla:



En el desplegable de la derecha elegimos **Sonido** y entre las librerías que aparecen seleccionamos **Minim** y pulsamos sobre le botón **Install**.

Así quedaría instalada la librería para poder utilizarla. Lo siguiente que hay que hacer es meter en la carpeta DATA de nuestro proyecto los sonidos con los que vayamos a trabajar. Para ello seguimos los siguientes pasos:



Buscando el archivo de sonido en la carpeta correspondiente, quedando así adjuntado al proyecto. Este paso lo debemos hacer por cada fichero de sonido que queramos utilizar en nuestro proyecto.

Con todo esto ya estamos en condiciones de introducir el código y probar su resultado.

En este caso vamos a trabajar con tres sonidos, que serán lanzados al pulsar las teclas “a”, “s” y “d” respectivamente.

Los tres sonidos utilizados para que los descarguéis son: [sonido1.mp3](#) [sonido2.mp3](#) y [sonido3.mp3](#).

El código es el siguiente:

```
import ddf.minim.*;
Minim soundengine;
AudioSample sonido1;
AudioSample sonido2;
AudioSample sonido3;
void setup()
{
soundengine = new Minim(this);
sonido1 = soundengine.loadSample("sonido1.mp3", 1024);
sonido2 = soundengine.loadSample("sonido2.mp3", 1024);
sonido3 = soundengine.loadSample("sonido3.mp3", 1024);
}
void draw() {
}
void keyPressed() {
if (key == 'a' || key == 'A') {
sonido1.trigger();
}
if (key == 's' || key == 'S') {
sonido2.trigger();
}
if (key == 'd' || key == 'D') {
sonido3.trigger();
}
}
```

## 26. Práctica IX.

Realizar mediante una Processing una melodía donde intervengan al menos tres sonidos.

## 27. Control de fecha y hora.

Vamos a utilizar la fecha y hora del sistema en nuestros proyectos de Processing. Para ello disponemos de estas tres funciones para la fecha:

**year()**.....nos devuelve el año



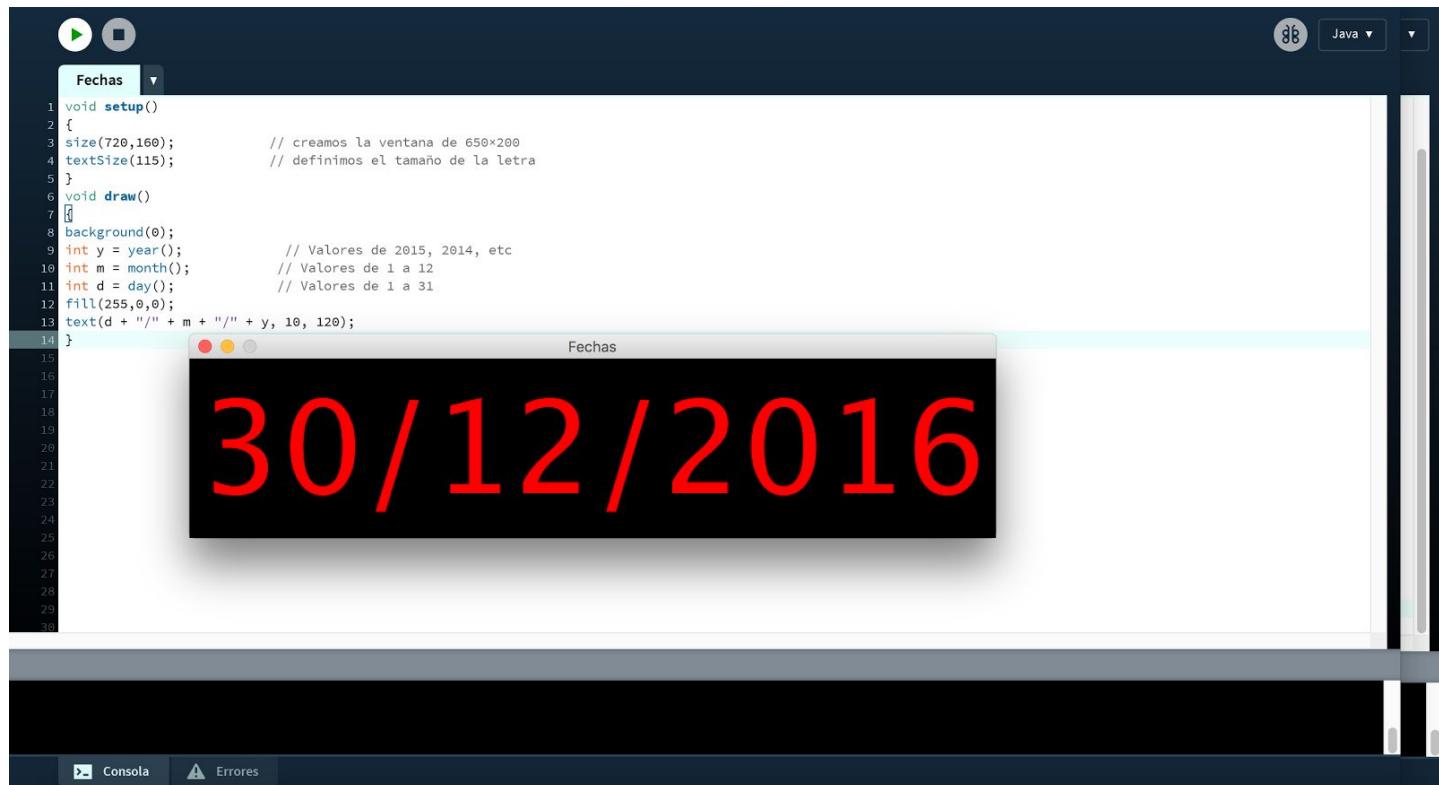
**month()**.....nos devuelve el mes

**day()**.....nos devuelve el día

Ejemplo:

```
void setup()
{
    size(640,160);
    textSize(115); // creamos la ventana de 650x200
    // definimos el tamaño de la letra
}
void draw()
{
    background(0);
    int y = year(); // Valores de 2015, 2014, etc
    int m = month(); // Valores de 1 a 12
    int d = day(); // Valores de 1 a 31
    fill(255,0,0);
    text(d + "/" + m + "/" + y, 10, 120);
}
```

Siendo el resultado:



Processing dispone de las siguientes tres funciones para la hora:

**hour()**.....nos devuelve la hora

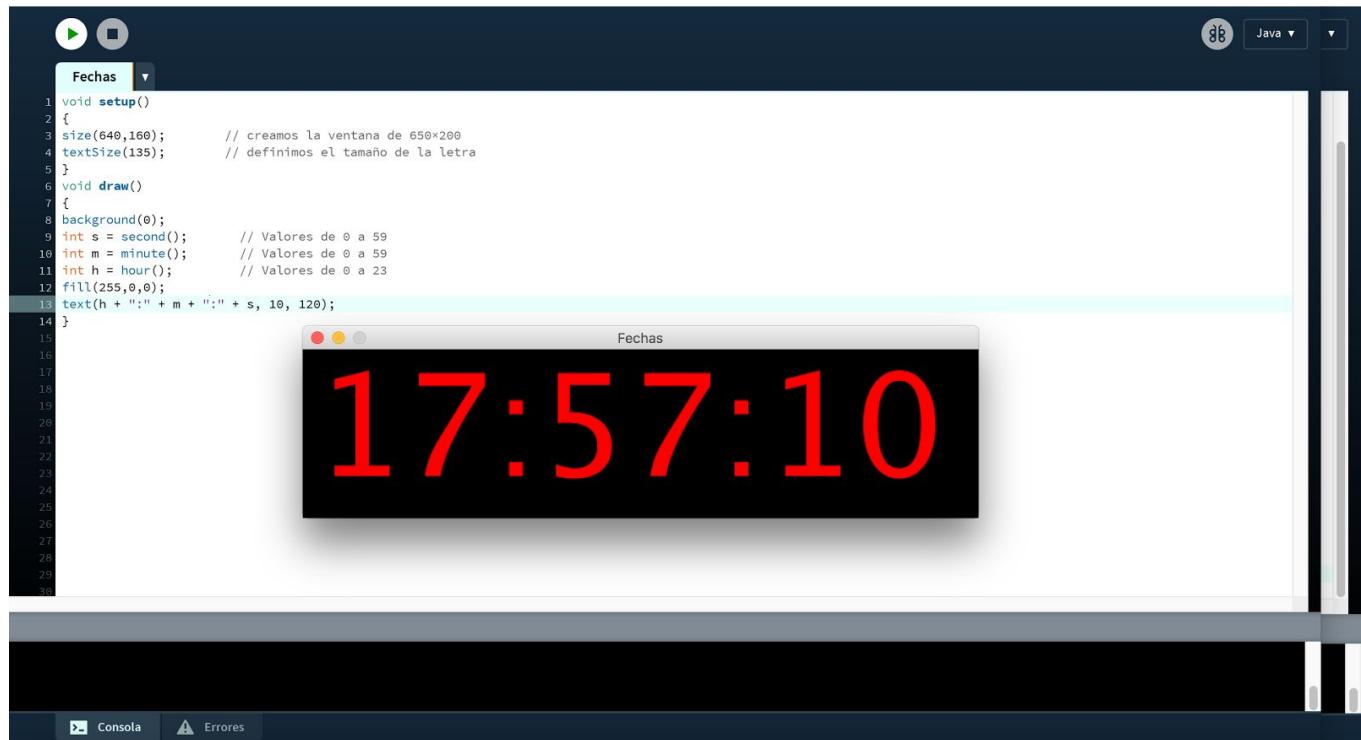
**minute()**.....nos devuelve los minutos

**second()**.....nos devuelve los segundos

Ejemplo:

```
void setup()
{
    size(640,160);           // creamos la ventana de 650x200
    textSize(135);          // definimos el tamaño de la letra
}
void draw()
{
    background(0);
    int s = second();        // Valores de 0 a 59
    int m = minute();        // Valores de 0 a 59
    int h = hour();          // Valores de 0 a 23
    fill(255,0,0);
    text(h + ":" + m + ":" + s, 10, 120);
}
```

Siendo éste el resultado:



## 28. Prácticas X.

Con los conocimientos adquiridos en esta práctica y en la anterior de “control del sonido”, desarrollar un reloj despertador que nos muestre la hora en formato digital y que reproduzca una melodía a una determinada hora para despertarnos.

## 29. Giros.

Vamos a aprender a girar objetos en Processing.

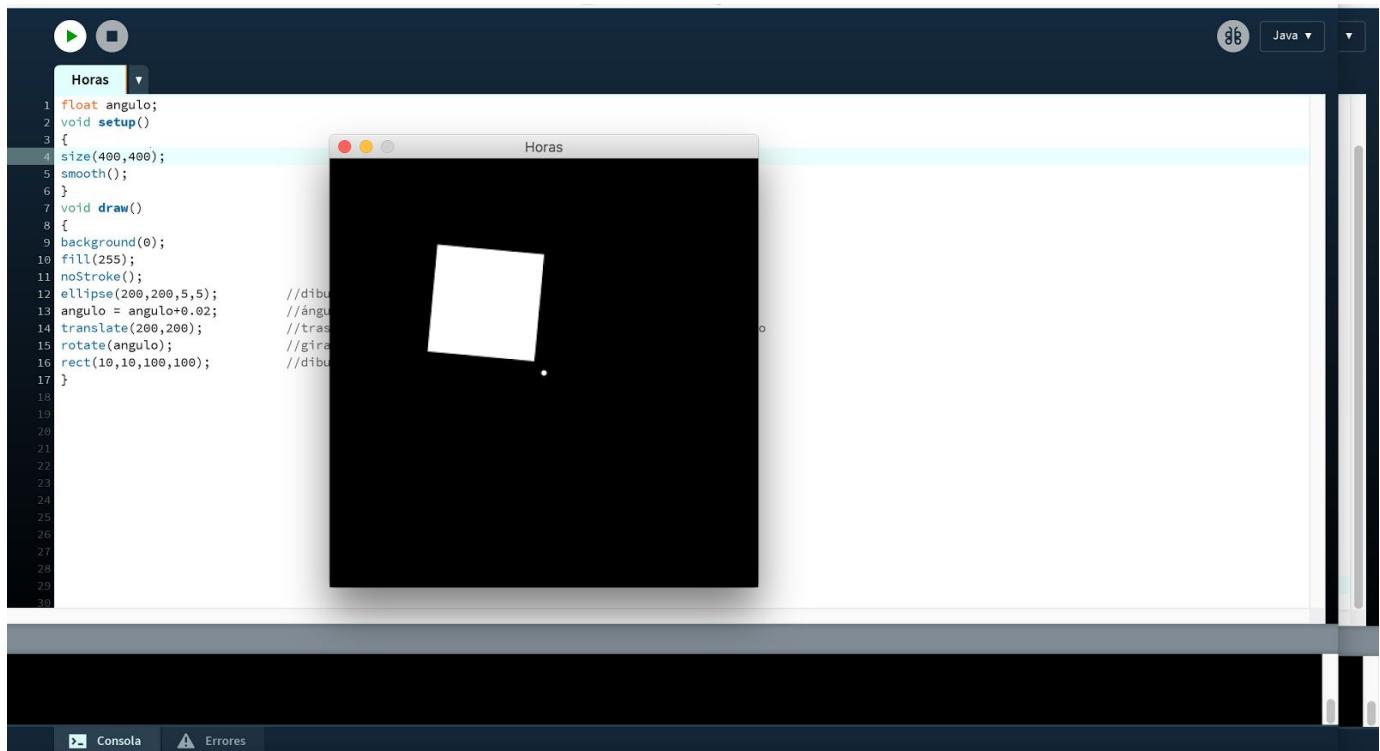
Para ello utilizaremos la función:

**rotate(ángulo);**

Ejemplo:

```
float angulo;  
void setup()  
{  
    size(400,400);  
    smooth();  
}  
void draw()  
{  
    background(0);  
    fill(255);  
    noStroke();  
    ellipse(200,200,5,5);           //dibuja el puntito del centro de giro  
    angulo = angulo+0.02;          //ángulo girado  
    translate(200,200);            //traslada el origen de coordenadas para girar dentro del lienzo  
    rotate(angulo);               //gira el objeto el ángulo girado  
    rect(10,10,100,100);          //dibuja el objeto  
}
```

Obteniéndose el siguiente resultado:



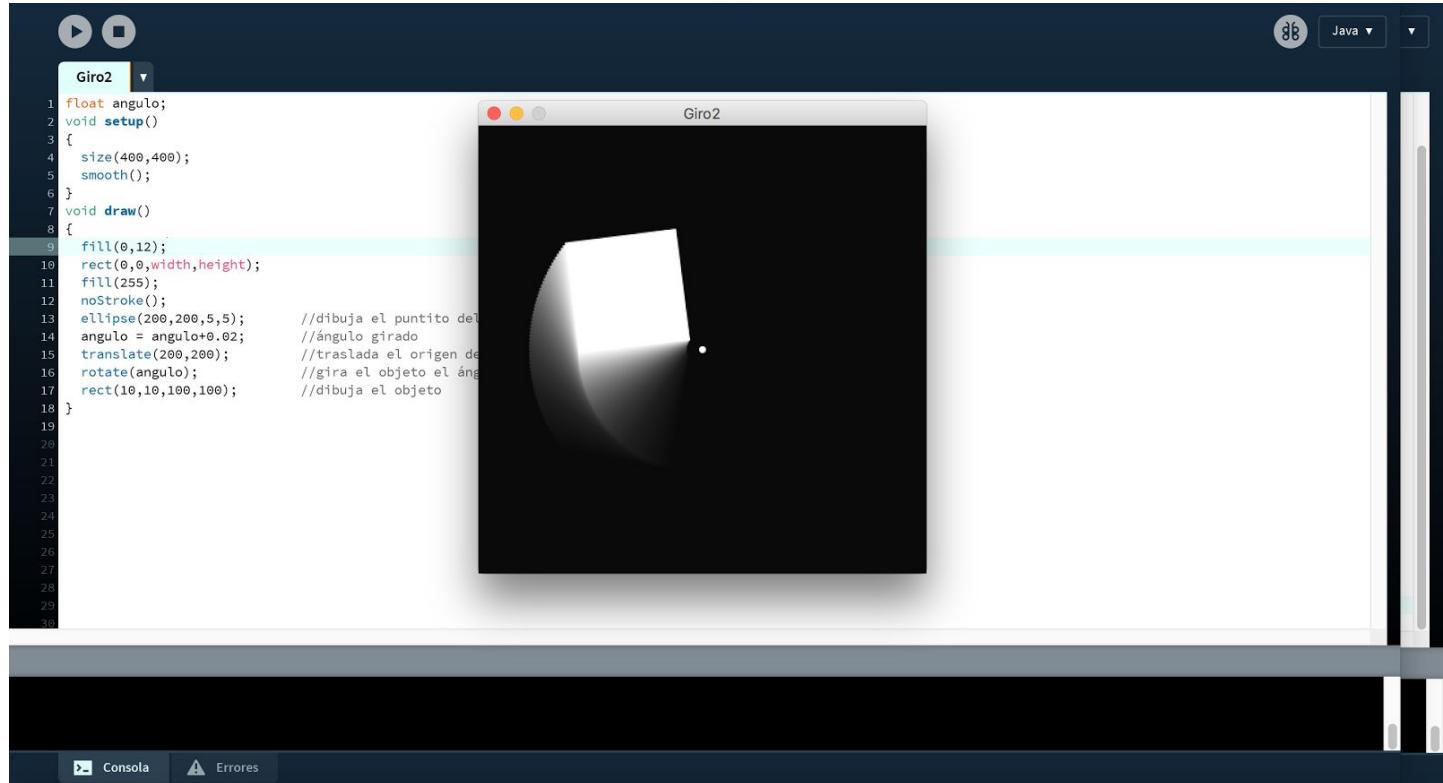
Y si jugamos con la transparencia del cuadrado podemos conseguir el siguiente efecto:

```

float angulo;
void setup()
{
    size(400,400);
    smooth();
}
void draw()
{
    fill(0,12);
    rect(0,0,width,height);
    fill(255);
    noStroke();
    ellipse(200,200,5,5);           //dibuja el puntito del centro de giro
    angulo = angulo+0.02;          //ángulo girado
    translate(200,200);            //traslada el origen de coordenadas para girar dentro del lienzo
    rotate(angulo);                //gira el objeto el ángulo girado
    rect(10,10,100,100);           //dibuja el objeto
}

```

Resultado:



## 30. Prácticas XI.

### 30.1. Mueve un ojo.

A partir de la imagen siguiente [monstruo.jpg](#), conseguir que se mueva la pupila izquierda.

## 30.2. Mueve los dos ojos.

Para lanzar varios giros en la misma aplicación debemos utilizar la función **resetMatrix()** entre ambos. Sabido ésto, modificar el programa de la tarea anterior utilizando la siguiente imagen [monstruo2.jpg](#) para conseguir que se muevan los dos ojos.

# 31. Proyectos.

## 31.1. Contador de tiempo.

Crea un contador que cuente el número de segundos que pasan desde que la aplicación se inicia.

## 31.2. Artista.

Demuestra tu faceta artística y elabora una aplicación original en la que se creen figuras, se muevan, cambien de color o lo que se te ocurra. Debes trabajar con valores aleatorios.

## 31.3. Adapta el código.

Dado el código que hay más abajo, adáptalo para que cada vez que pulsemos sobre el círculo con el ratón sumemos un punto y nos lo muestre en la pantalla.

```
/*
Ejemplo de cómo crear proyecto pelota
sin clases
*/
float xpos; // posición x de la pelota
float ypos; // posición y de la pelota
float vx=random(-5, 5); // velocidad x pelota
float vy=random(-5, 5); // velocidad y pelota
float diameter=40; // diámetro de la pelota

void setup() {
  // tamaño de la pantalla
  size (400, 400);
  // posición inicial de la pelota
  xpos=width/2;
  ypos=height/2;
}

void draw() {
  background(255);

  // mueve la pelota
  xpos=xpos+vx;
  ypos=ypos+vy;
  if ((xpos>width-diameter/2)||((xpos-diameter/2<0)) {
```



```
vx=-vx;  
}  
if ((ypos>height-diameter/2)||((ypos-diameter/2<0)) {  
    vy=-vy;  
}  
  
// dibuja la pelota  
fill(255);  
stroke(28, 201, 16);  
strokeWeight(3);  
ellipse(xpos, ypos, diameter, diameter);  
}
```

## 32. ¿Te interesa y quieres ampliar?

Aquí te dejo algunas webs que pueden ser de interés para ampliar conocimientos con este fascinante lenguaje de programación.

[Cómo programar para android con Processing](#)

[Controlando arduino desde Processing](#)

[Processing for android: usando el gps](#)

[Processing for android: controlando el tamaño del emulador](#)

[Mvoce, Síntesis de voz en Processing](#)

[Capturar webcam en Processing](#)

[Página web de Processing](#)

[Web para probar en tiempo real los sketchs de Processing](#)

[Taller de programación creativa con Processing](#)

[Presentación resumen](#)

[Canal de Youtube de Agustín Casaña](#)

[Tutorial sencillo](#)

[Recursos Processing](#)

[Vídeos en Vimeo](#)

[Tutorial dunadigital](#)

## 33. Webgrafía.

[Programación y Robótica.](#)



[Joan.cat](http://Joan.cat)

[Tecnología, programación y robótica en Secundaria](#)

