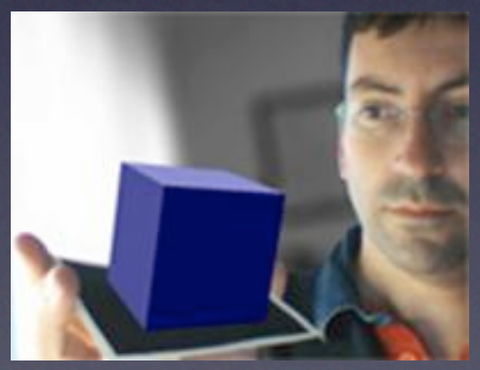
Gráficos por Computador

Introducción a *processing*



Jordi Linares i Pellicer
Escola Politècnica Superior d'Alcoi
Dep. de Sistemes Informàtics i Computació
jlinares@dsic.upv.es
http://www.dsic.upv.es/~jlinares

¿ Qué es *processing* ?

- *Processing* es un dialecto de Java específicamente diseñado para el desarrollo de arte gráfico, animaciones y aplicaciones gráficas de todo tipo
- Desarrollado por artistas y para artistas
- *Processing* es software libre, disponible en las plataformas donde Java está disponible (Mac OS, Linux, Windows etc.)
- Desarrollado a partir del 2001 en el MIT por Casey Reas y Ben Fry. Inspirado en DBN (*Design By Numbers*) de John Maeda
- http://processing.org -> portal de descarga y centro neurálgico del mundo *processing*

¿ Por qué *processing* ?

- Muy, pero que muy fácil de aprender... primeras pruebas en sólo unos minutos
- Dispone de su propio IDE de desarrollo (fácil)
- Mejor curva de aprendizaje que OpenGL + GLUT u otras alternativas
- Potente, pero que muy potente. Permite desarrollar aplicaciones desde muy sencillas a muy complejas.
- Escalable. Se puede combinar programación con *processing* con Java puro y las librerías propias de Java. Una aplicación Java 'pura' también puede usar las librerías de *processing*.

¿ Por qué *processing* ?

- Permite 3 formas de programar: básica, procedural/ estructurada y orientada a objetos
- Las primeras pruebas y 'sketches' sencillos se pueden desarrollar en el modo básico (sin funciones, código directo)
- 'sketches' y programas más ambiciosos pueden hacer uso de funciones (programación a la 'C')
- Programas más complejos pueden desarrollarse aplicando orientación a objetos (clases de Java)
- Finalmente, se puede desarrollar en java 'puro' y usar librerías de *processing* y otros entornos (eclipse, netbeans etc.)

¿ Por qué *processing* ?

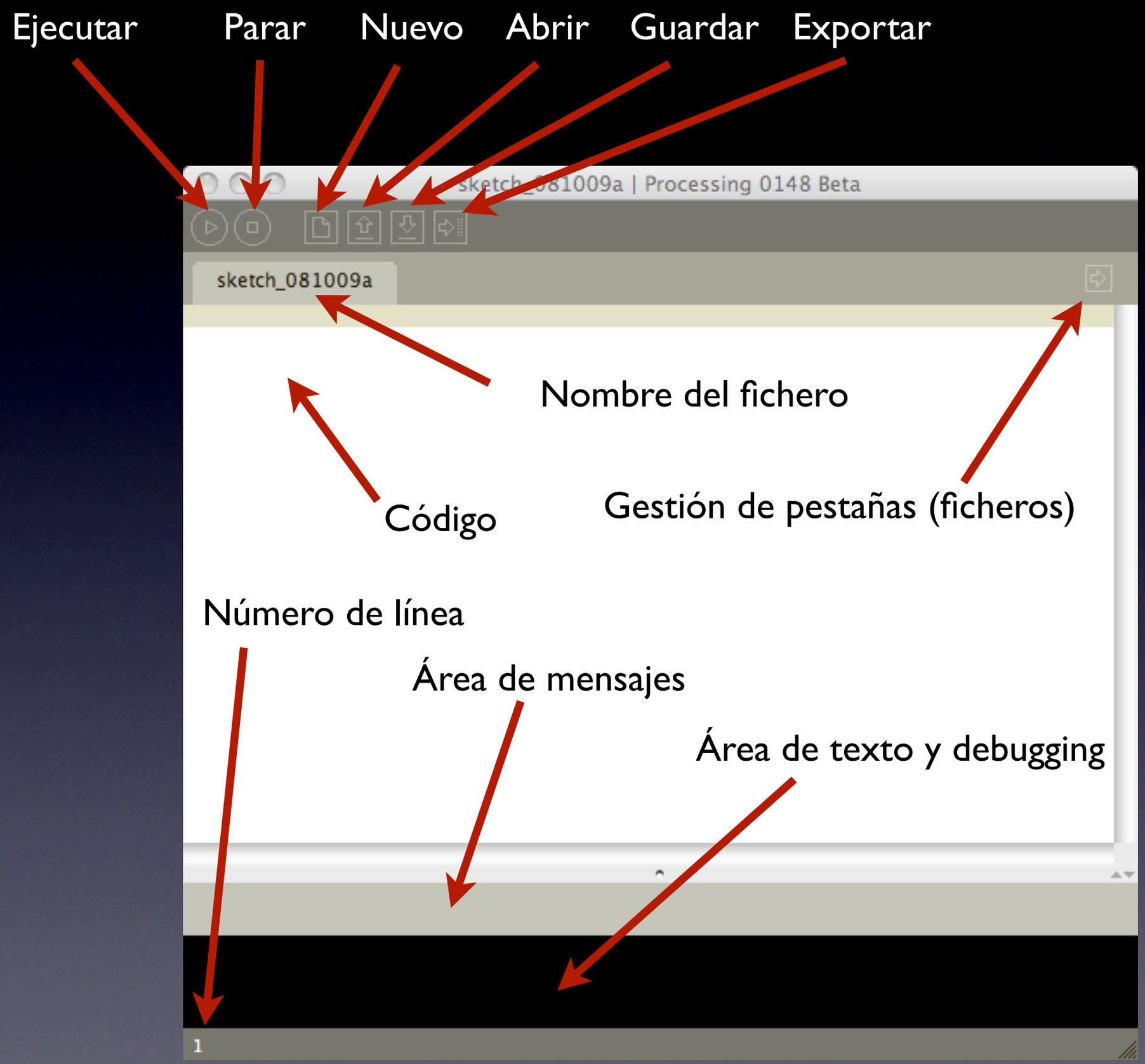
- Genera aplicaciones listas para ser ejecutadas en las tres principales plataformas: Mac OS, Linux y Windows
- Las aplicaciones *processing* también pueden generarse para su ejecución en internet (como un *applet* de Java)
- Es posible desarrollar aplicaciones para dispositivos móviles (http://mobile.processing.org)
- Conexión con dispositivos y prototipos electrónicos: proyectos Arduino y Wiring (http://hardware.processing.org)

Paquetes gráficos. Historia

- Estándares:
  - 3D Core Graphics System. ACM y SIGGRAPH (1977)
  - GKS (Graphical Kernel System). 2D. ISO-ANSI (1985)
  - GKS-3D (1988)
  - PHIGS (Programmer's Hierarchical Interactive Graphics System). 3D. (1988)
  - SRGP y SPHIGS (Foley)
- Otros
  - OpenGL
  - XWindows, Microsoft Windows, Mac OS ...
  - VRML, X3D (orientados a internet)
  - POV-Ray, Renderman
  - Java2D, Java3D
  - DirectX (Direct-3D)
  - etc.

El IDE de *processing*

- *Processing* dispone de un IDE (Integrated Developement Environment) propio desarrollado en Java
- Sencillo y fácil. Suficiente para la mayoría de aplicaciones. Se puede migrar a Eclipse, p.e., ante aplicaciones de mayor envergadura
- Se conoce por PDE (*processing* development environment)


Ejecutar
Parar
Nuevo
Abrir
Guardar
Exportar
sketch_081009a | Processing 0148 Beta
sketch_081009a
Nombre del fichero
Código
Gestión de pestañas (ficheros)
Número de línea
Área de mensajes
Área de texto y debugging
1

El IDE de *processing*

- Cuando guardamos una aplicación, se genera un directorio con ese nombre y el fichero de la aplicación se almacena dentro con la extensión *.pde*
- Los nombres de los ficheros máximo 64 caracteres sin espacios
- Se pueden crear más ficheros asociados a la misma aplicación (gestión de pestañas). Estos ficheros por defecto tendrán la misma extensión *.pde*, pero también pueden tener la extensión *.java* (y serán tratados como tales).
- Los ficheros adicionales corresponderán a nuevas funciones o clases (en tal caso habrá que definir en el principal la función *setup*)
- Las pestañas visibles serán las que se consideraran al generar la aplicación
- Para abrir un proyecto bastará con abrir cualquier *.pde* del interior del directorio

El IDE de *processing*

- *Run* y *Stop* permiten iniciar la ejecución de la aplicación y su finalización
- *Export* permite generar un '*applet*' de la aplicación. La opción del menú '*Export Application*' permite generar una aplicación lista para ejecutarse en Mac OS, Linux o Windows. Todo esto se generará en directorios dentro del directorio principal de la aplicación
- El directorio '*Skecthbook*' es el directorio por defecto para el usuario pero se puede almacenar las aplicaciones en cualquier directorio

El IDE de *processing*

- Otras opciones interesantes:
  - *Import Library*. Nos facilita la inclusión de las sentencias *import* de las librerías más útiles
  - *Add File*. Nos permite añadir imágenes, fuentes u otra información multimedia a un directorio especial en nuestra carpeta de aplicación: la carpeta 'data'. En esta carpeta debemos almacenar cualquier elemento que queramos utilizar en nuestra aplicación (mediante la función '*loadImag*e', p.e.). La carpeta 'data' se creará si no existía previamente.
  - *Create font*. Facilita la creación de *applets* que utilicen fuentes de nuestro sistema y que queramos garantizar que aparezcan correctamente en cualquier navegador.
  - *Help*. Nos permite consultar la información de *processing,* incluso ver la referencia de la función que tengamos seleccionada en el código

El lenguaje *processing*

- *processing* está basado en Java 1.4.2 (sólo unas cuantas modificaciones con objeto de simplificar la programación)
- Se puede utilizar 1.5 y posteriores pero utilizando otro entorno de desarrollo (aplicación Java pura y utilizando las clases de la librería gráfica de *processing*)
- Documentación completa en:
  - http://java.sun.com/j2se/1.4.2/docs/api/index.html
- *processing* permite programar a la 'C' (programación estructurada, a base de funciones). Pero es conveniente aprovechar su enfoque orientado a objetos (ligeras modificaciones de las clases Java)

El lenguaje *processing*

Código processing
(.pde)

Preprocesador
(*processing*)

Código Java
(.java)

Compilación
(javac)

bytecode
(.class)

Ejecución

JVM
(Java Virtual Machine)

El lenguaje *processing*

- Variables

  `type name;`
- Tipos básicos

  `byte`, `short`, `int` (178), `long` (8864L), `float` (37.266F), `double` (37.266/26.77e3), `char` (‘c’), `boolean` (true/false)
- Arrays

  `byte[ ] array;`
- Inicialización

  `type name = value;`
- Constantes

  `final type variable = value;`

El lenguaje *processing*

- Aritméticos: `+, -, *, /, %, ++, --`
- Relacionales: `>, >=, <, <=, ==, !=`
- Condicionales (logic): `&&, ||, !, &, |, ^`
- Operaciones con bits: `<<, >>, &, |, ^, ~`
- Asignación: `=, +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=`
- Otros: `?:, [], ., (type), new, instanceof`
- Precedencia:

1. `expr++, expr--`
2. `++expr, --expr, ~, !`
3. `*, /, %`
4. `+, -`
5. `<<, >>`
6. `<, >, <=, >=, instanceof`
7. `==, !=`
8. `&`
9. `^`
10. `|`
11. `&&`
12. `||`
13. `?:`
14. `=, +=, -, =, *=, /=, %=, &=, ^=, |=, <<=, >>=`

El lenguaje *processing*

- Delimitadores de bloque: { instrucciones; }
- while:

```
while ( expr ) { instrucciones }
do { instrucciones } while ( expr )
```

- for

```
for ( begin; end; inc ) {instrucciones }
```

- if/else

```
if ( expr ) {instrucciones }
if ( expr ) {instrucciones } else { instrucciones }
```

- switch

```
switch ( var ) { case val: instrucciones default: }
```

- Saltos: `break`, `continue`, `return`

El lenguaje *processing*

- Comentarios:

```
// Este comentario va hasta el final de línea
/* Este comentario es
multilínea */
```

- Funciones:

```
tipo_retorno nombreFunción(lista de parámetros)
```

Si se definen funciones propias en *processing*, se está obligado a dar implementación al menos a la función predefinida `setup()`

El lenguaje *processing*

- Ejemplo:

```
void setup() // Función que se ejecuta al principio
{
   size(100, 100);
   dibujaLinia(5);
}
void dibujaLinia(int x)  // Función propia
{
   line(x, 0, x, 99);
}
```

El lenguaje *processing*

- Strings:
  - `String mensaje = “hello”;`
  - `String mensaje = “hello” + “bye”;`
- Gran cantidad de métodos:
  - `length()`
  - Acceso a un carácter: `charAt()`
  - `substring()`
  - Modificación: `concat()`,`replace()`,`trim()`,`toLowerCase()`,etc.
  - Búsqueda: `indexOf()`, `lastIndexOf()`, etc.
  - Comparación: `startsWith()`, `endsWith()`, `compareTo()`, etc.
  - etc.
- Ejemplos:

```
String mensaje = “hello”+ “bye”;
int len = mensaje.length();
int len = “hello”.length();
```

El lenguaje *processing*

- Un array en *processing* (Java) es un objeto:
  - Su longitud puede conocerse mediante el atributo *length*
  - Ejemplos:

```
int[] vector; // vector es 'null'
vector = new int[3]; // creamos 3 componentes
int len = vector.length; // Longitud = 3
int item = vector[2]; // acceso a una componente

int [][] matriz = new int[4][4];
matriz[0][0] = 12;  // Matrices
```

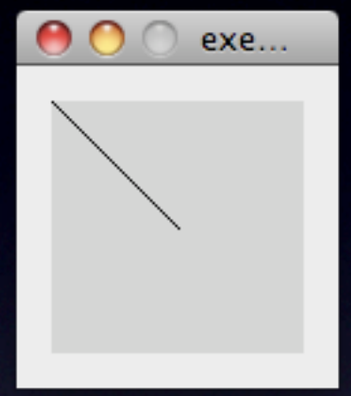
El lenguaje *processing*

- ¿ Cuales son las diferencias de *processing* como lenguaje respecto a Java ?
  - El preprocesador de *processing* nos permite usar en general una sintaxis más relajada
  - Permite el modo básico: sin funciones propias, únicamente variables globales y directamente líneas de código
  - Permite programar a la 'C', definiendo nuestras propias funciones, sin tener que definir clases y métodos
  - Permite definir y usar clases, con una sintaxis de clases más sencilla que después es transformada a completas clases Java
  - Los modos básico y a la 'C', las variables globales etc. son posibles en *processing* haciendo que el preprocesador las integre en una clase (de forma transparente al programador)

El lenguaje *processing*

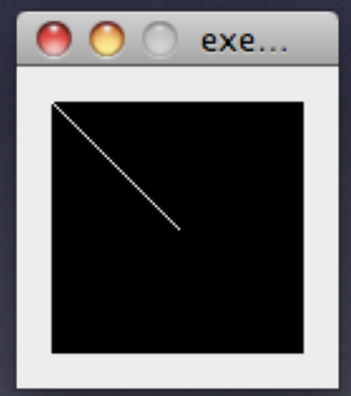
- Ejemplo en modo básico (I)

```
// Programa mínimo en processing
line(0,0,50,50);
```



- Ejemplo en modo básico (II)

```
// También variables y otros elementos ...
int i;
background(0); // pantalla a negro
stroke(255);   // trazo de color blanco
for (i = 0; i <= 50; i++)
      point(i, i);
```



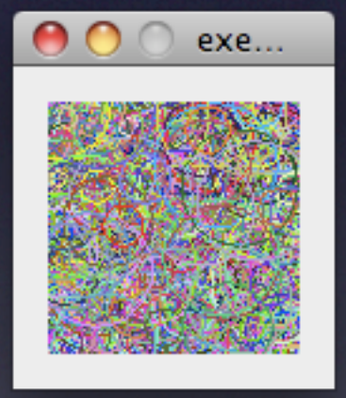
El lenguaje *processing*

- Ejemplo a la 'C':

```
// Si definimos funciones o clases propias, hay que
// dar implementación a setup()
void setup()
{
    size(100,100); // Tamaño de ventana de la aplicación
    noFill(); // Sin relleno en los círculos
}

void draw() // Se repite a cada frame
{
    dibujaCirculoAleatorio();
}

void dibujaCirculoAleatorio()
{
    int r = int(random(50));
    // Color del trazo
    stroke(random(255), random(255), random(255));
    // Círculos con posición, radio y color aleatorios
    ellipse(random(100), random(100), r, r);
}
```



El lenguaje *processing*

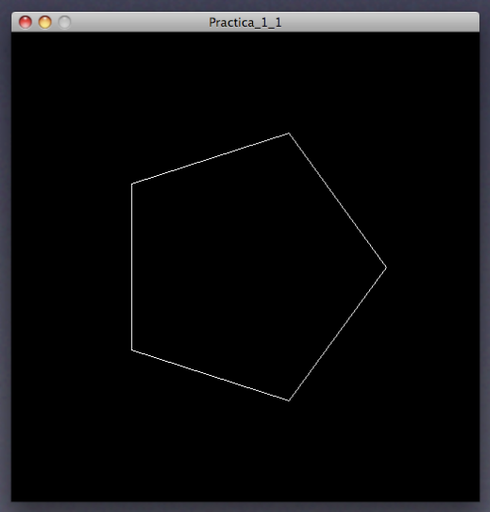
- Clases

```
void setup()
{
    size(100,100); // Tamaño de ventana de la aplicación
    noFill(); }

void draw() // Se repite a cada frame
{
    Circulo c = new Circulo();
    c.dibuja();
}
// Clase círculo
class Circulo
{
    // Atributos
    int r = int(random(50));
    int x = int(random(100));
    int y = int(random(100));
    color c = color(random(255), random(255), random(255));
    // Métodos
    void dibuja()
    {
        stroke(c);
        ellipse(x, y, r, r);
    }
}
```

Práctica 1-1

- Desarrollar un programa en *processing* que permita dibujar polígonos de n lados.
- Para ello hay que implementar una función `poligon` con los siguientes parámetros: centro (dos enteros), radio (entero) y número de lados del polígono (entero)
- Información de interés para su desarrollo:
  - Lo primero es implementar la función `setup`, donde se puede definir el tamaño de ventana, p.e. `size(500,500)`, y a continuación invocar a la función `poligon`
  - La función `poligon` deberá trazar el polígono utilizando:
    - Las funciones trigonométricas `sin()` y `cos()` cuyos argumentos se exigen en radianes
    - La función `line(x1, y1, x2, y2)` que traza una línea de (x1, y1) a (x2, y2)
    - El (0,0) está situado en la esquina superior izquierda (las x positivas hacia la derecha, las y positivas hacia abajo)
    - Constantes predefinidas como `PI` y `TWO_PI`



Práctica 1-2

- Desarrollar un programa en *processing* que permita dibujar un gradiente de color, de forma que la primera fila de la ventana se dibujará de un color de partida y la última fila de un color de destino. Las filas intermedias se dibujaran de forma que llevarán a cabo una graduación lineal de colores entre las dos filas inicial y final.
- Para ello hay que implementar una función `gradient` con los siguientes parámetros: 6 enteros; los primeros 3 son los valores RGB del color de partida (de 0 a 255), los otros tres los valores RGB del color de destino
- Información de interés para su desarrollo:
    - Lo primero es implementar la función `setup`, donde se puede definir el tamaño de ventana, p.e. `size(500,500)`, y a continuación invocar a la función `gradient`
    - La función `gradient` deberá trazar el gradiente utilizando:
        - Interpolación lineal de cada component de color
        - Las variables `width` y `height` nos devuelven en todo momento el ancho y alto de la ventana de la aplicación
        - La función `line(x1, y1, x2, y2)` que traza una línea de (x1, y1) a (x2, y2)
        - La función `stroke` que permite cambiar el color del trazo de las líneas, donde sus 3 argumentos son los valores RGB de la línea