Table of Contents

Creación Visual a través de la Programación

Curso Optativo. Área de los Lenguajes Computarizados, IENBA, UdelaR. 2017.

Este curso optativo plantea una introducción a la generación de imágenes digitales desde el código y la programación. Generando un enfoque práctico a los lenguajes de programación más usados en el campo artístico como herramientas fundamentales para la generación de imágenes de carácter creativo a partir de procesos informáticos. Tiene como objetivo principal el crear un acercamiento entre los estudiantes y la programación como herramienta para la creación.

Contenido temático del curso.

- Introducción a la generación de imágenes a través de la programación.
  - Historia.
  - Referencias Actuales.
- Introducción al dibujo a través de la programación.
  - Pixeles.
  - Sistema de coordenadas.
- Introducción a Processing
  - Interfaz
- Dibujo Estático
  - Fornas básicas.
  - Formas irregulares.
  - Color
- Color
  - RGB
  - Modos de color
  - Alpha
- Imágenes dinámicas
  - setup()
  - loop()
  - Datos
    - Variables de sistema
    - Variables de usuario
- Textos y tipografía
- Repetición
  - for
  - while
- Condicionales
  - if
  - case
- Aleatoriedad
  - Random
  - Noise
- Imágenes externas

Aspectos Formales del Curso

El curso tiene una duración total de **30 horas**, divididas en 10 encuentros de 3 horas cada uno.

> La aprobación del curso requiere como mínimo un total de asistencia del 80% y la entrega de todas las premisas domiciliarias solicitadas, junto con una premisa de carácter final que englobe todo lo visto en el curso.
>
> Clase / 1
>
> Clase / 2
>
> Clase / 3

Creación de imágenes a través de programación.

En el arte.

Breve reseña histórica.

> A comienzos de los 60, personas provenientes de varias disciplinas, tales como la ingeniería, las matemáticas, las artes o la filosofía comienzan a investigar en el uso de computadoras en la creación de piezas y procesos artísticos. Las investigaciones en este campo abarcan desde la creación de imágenes abstractas o representativas hasta ploteos generativos, pasando por complejos algoritmos de representación.

Ben Laposky

Harold Cohen

Manfred Mohr

Michael Noll

George Nees

Edward Zajec

Hiroshi Kawano

Vera Molnar

Herbert Franke

Frieder Nake

> **Actualidad**

Aaron Koblin

Casey Reas

Algoritmos y Arte

> ***Un algoritmo es una secuencia no ambigua, finita y ordenada de instrucciones que han de seguirse para resolver un problema.***
>
> Características de un algoritmo:
>
> - Tiene que ser preciso.
> - Tiene que estar bien definido.
> - Tiene que ser finito.
> - La programación es adaptar el algoritmo al ordenador.
> - El algoritmo es independiente de donde se implemente.

Manifiesto Algorista

the algorist

Grupo fundado en 1995 que nuclea artistas que trabajaban en la creación artística desde la practica y la formulación de algoritmos.

```
if (creation && object of art && algorithm && one's own algorithm) {
    include * an algorist *
} elseif (!creation || !object of art || !algorithm || !one's own algorithm) {
    exclude * not an algorist *
}
```

Hello_World!

¿Porque aprender a programar, a crear nuestro propio software?

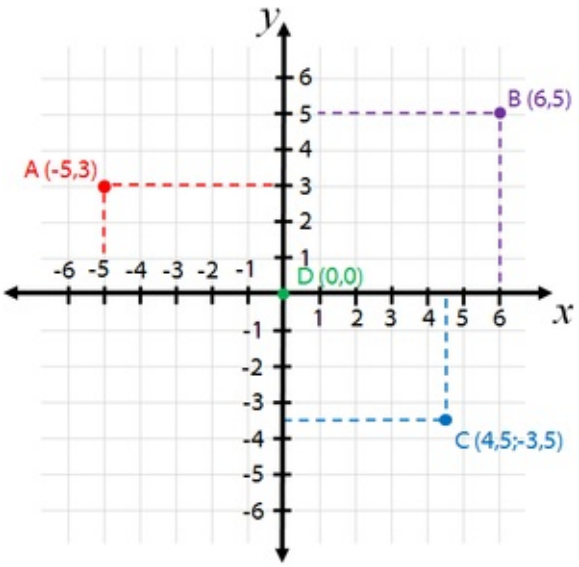
- Las herramientas que encontramos en el mercado, puede que estén limitadas.
- Para poder desarrollar nuestras ideas sin limitaciones, necesitamos crear herramientas que se ajusten exactamente a nuestras necesidades.

Elementos y practicas a tener en cuenta en la programación:

> - La computadora no tiene intuición.
> - Necesitamos una sintaxis correcta.
> - Pseudo-Código.
> - Proceso de "Desarrollo en aumento".

Computación Gráfica, Pixeles y Coordenadas

El pixel, la unidad gráfica mínima dentro de una imagen debe ser colocado en su posición a través de coordenadas. Este sistema de coordenadas tiene similitudes y diferencias con el de Ejes Cartesianos que estamos mas habituados a ver.



En el sistema de Ejes Cartesianos tenemos una representación gráfica compuesta por dos ejes, uno horizontal llamado eje "x" y uno vertical llamado eje "y". La intersección de los mismos es lo que conocemos como punto de origen y sus coordenadas son x = 0 e y = 0. En este caso el origen es el punto `D(0,0)` .

```
A(-5, 3)
B(6, 5)
C(4.5, -3.5)
```

Por otro lado, el sistema de coordenadas en processing (y en general otros softwares de producción de imágenes) se maneja de manera diferente.

La diferencia radica en que no existen pixeles negativos en pantalla, por lo cual el punto de origen, es decir donde se cruza el eje x y el eje y, se encuentra en la esquina superior izquierda de la ventana. Los valores de x aumentan a medida que nos movemos a la derecha y los valores de y aumentan a medida que nos desplazamos hacia abajo.

Puntos y Lineas

El elemento gráfico mas simple que podemos dibujar con pixeles (dejando afuera el punto) es la linea, un segmento de recta que marca la distancia mas corta entre un punto A y un punto B.

El pseudo-código para indicar como dibujar una linea, definiendo la posición del punto `A(30, 100) y punto B(110, 120)` :

```
Dibujar linea desde (30, 100) a (110, 120).
```

Sin embargo, cuando traducimos nuestro pseudo-codigo a la sintaxis correcta en processing el resultado es:

```
line(30, 100, 110, 120);
```

```
line(Ax, Ay, Bx, By);
```

`line` es el nombre de la función y los elementos dentro de los paréntesis son conocidos como `argumentos` , los cuales son valores numéricos separados entre si por comas.

Podemos pensar en la función como un comando y en los argumentos como los parámetros en los que ese comando se va a ejecutar.

`Ax - Posision x del punto A`

`Ay - Posision y del punto A`

`Bx - Posision x del punto B`

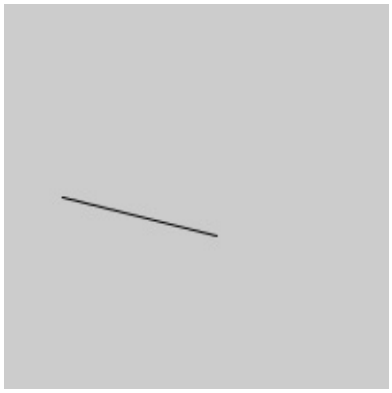
`By - Posision y del punto B`

Sin embargo, para que el sistema de coordenadas en processing funcione como queremos, debemos establecer el tamaño de la ventana.

Por defecto, Processing usa un tamaño de 100 pixeles por 100 pixeles. En este caso podemos usar una ventana de 200 por 200 para poder visualizar correctamente la linea.

```
size(200, 200);
line(30, 100, 110, 120);
```

Resultado:



Ejercicio:

En papel cuadriculado dibujar o diseñar una composición utilizando solo formas primitivas, estas son: rectángulos, elipses, lineas, puntos y triángulos.

En una segunda hoja, escribir el pseudo-código para esa composición, tratando de descubrir los parámetros para todas las formas primitivas.

Relleno y bordes

En Processing las figuras primitivas como las vistas hasta ahora están compuestas por dos elementos, un elemento de borde `stroke` y un elemento de interior, o relleno `fill`

`stroke();`

Define el color de la linea de borde de una figura.

fill `();`

Define el color de relleno de una figura.

> Debemos tener en cuenta que el orden en el que colocamos estos comandos importa. Deben ir siempre por delante de la función que dibuja la figura a la cual queremos aplicarle estas propiedades, ya que lo que hacen es decirle a Processing como debe preparase para dibujar esa figura.

Color

RGB

El modo por defecto en el que se tratan los colores en Processing es en base a valores numéricos correspondientes a los canales RGB.

Estos canales (Red, Green, Blue) corresponden a los tres colores primarios de la luz y se combinan para formar el resto.

En Processing, cada canal se representa con un valor numérico entre 0 y 255 y la sintaxis para definirlo, por ejemplo, en la función fill() es la siguiente.

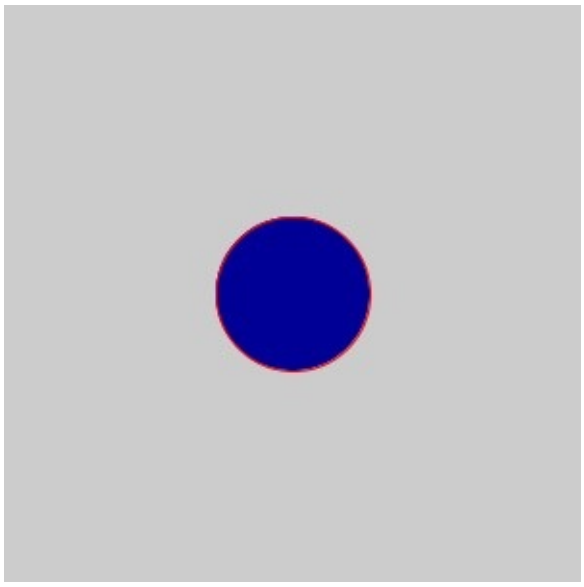
```
fill(R, G, B);

De esa manera, para obtener un color rojo pleno definimos:

fill(255, 0, 0);

Damos todo el valor posible al canal rojo y dejamos en 0 los demas.

fill(0, 255, 0); //Verde
fill(0, 0, 255); //Azul

Y de la misma manera:
fill(255, 255, 255); //Blanco
fill(0, 0, 0); //Negro
```

Si todos los canales están en el valor 255 obtenemos el color blanco, al igual que las propiedades de la luz cuando sumamos los tres canales. De la misma manera, si dejamos todos los canales en 0, marcamos la ausencia de luz, por lo que el color resultante es negro.

Ejemplo:

```
size(300,300);

stroke(255, 0, 0);
```

```
fill(0, 0, 150);
ellipse(150, 150, 80, 80);
```



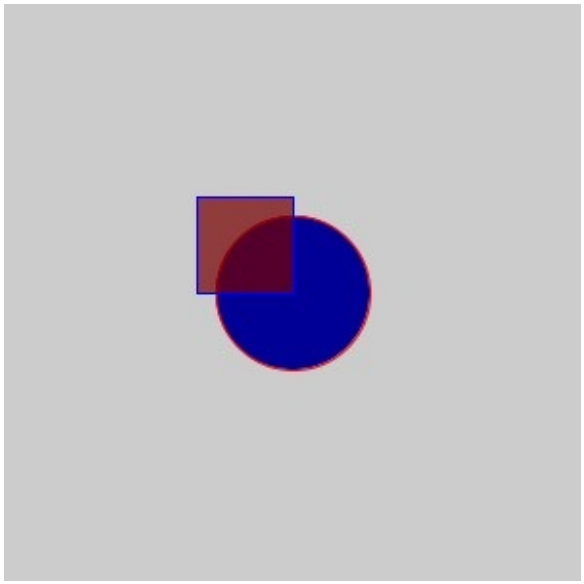
Posibilidades de relleno en fill() y stroke()

Dependiendo de la cantidad de argumentos que le demos a estas dos funciones tendremos varias opciones disponibles.

```
fill(RGB);
fill(255);
Con un parámetro, Processing asume que el resto de los parámetros son iguales.
Es equivalente a fill(255,255,255);

fill(RGB, Alfa);
fill(255, 255);
Con dos parámetros, el primero corresponde a los tres valores de RGB.
El segundo canal corresponde al valor de 'Alfa' o transparencia. 0 es totalmente transparente, 255 opaco.

fill(R, G, B);
fill(0, 120, 255);
Cada valor corresponde a uno de los canales.

fill(R, G, B, Alfa);
fill(0, 120, 255, 200);
Cuatro parámetros corresponden a cada uno de los canales RGB, sumado el canal de transparencia.
```

Ejemplo:

```
size(300,300);

stroke(255, 0, 0);
fill(0, 0, 150);
ellipse(150, 150, 80, 80);

stroke(0, 0, 255);
fill(120, 0, 0, 180);
rect(100, 100, 50, 50);
```

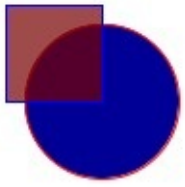


> **Para poder trabajar en otros modos de color ver: colorMode()**

background();

La función background() funciona de manera similar a la función fill(), pero en lugar de rellenar el interior de una figura rellena toda la ventana con el color marcado.

```
size(300,300);
background(255);

stroke(255, 0, 0);
fill(0, 0, 150);
ellipse(150, 150, 80, 80);

stroke(0, 0, 255);
fill(120, 0, 0, 180);
rect(100, 100, 50, 50);
```



Flow, Flujo del programa

Hasta el momento, venimos ordenando funciones e instrucciones en Processing para ir dibujando elementos en pantalla, pero realmente aun no estamos programando en el sentido mas amplio del concepto.

Estamos dejando afuera uno de los elementos basicos: el flujo.

El flujo del software define la manera en la que este funciona y se ejecuta.

Podemos encontrar dentro del software dos tipos de flujo:

- **Basado en Eventos - Por ejemplo, aplicaciones web. Un evento se dispara cuando el usuario hace click, luego otro evento le sigue. Si el usuario no hace nada, el software no hace nada.**
- **Basado en Loops - Un videojuego es un ejemplo de Flujo basado en Loop. Los enemigos, personajes, entorno se siguen manifestando y ejecutando en base a sus parámetros en una repetición constante hasta que algo los detenga.**

En processing podremos hacer uso de ambos.

Con los elementos de flujo entra un concepto conocido como "Bloques de Código".

```
{ //Abrimos bloque de código

    //Nuestro flamante programa

}//Cerramos bloque de codigo
```

En Processing, cuando entramos en el proceso de crear gráficos de manera dinámica, es decir con bloques de código y animaciones, debemos separar el programa en dos bloques principales.

```
setup{

//Este bloque de código se ejecuta solo cuando comienza el programa y no vuelve a ejecutarse.

}

draw{

//Este bloque de código se ejecuta de manera indefinida o hasta que el software se detenga.

}
```

El bloque `setup` corresponde a la parte de nuestro programa que no necesitamos que se ejecute mas de una vez, es la instancia de configuración y preparación.

El bloque `draw` lleva en su interior todos elementos que necesitamos cambien o se modifiquen durante la ejecución del software, es un bloque dinámico. Se repite de manera indefinida, es decir, cuando el programa termina de ejecutar la ultima linea del bloque `draw` , vuelve a la primera del mismo.

Sin embargo, la sintaxis para usar estos bloques en Processing es un poco diferente y debemos definirlo de la siguiente manera:

```
void setup() {

    //Se ejecuta una sola vez.

}

void draw(){
```

```
    //Se ejecuta por siempre.
}
```

Variables de Sistema

Las variables, de manera rápida (ya que profundizaremos mas adelante sobre el concepto) son elementos capaces de almacenar información dinámica en la memoria de nuestra computadora, permitiéndonos acceder a esos datos desde cualquier momento del programa.

Por ejemplo, pensemos que queremos dibujar un circulo cuyo valor de posición en X sea el mismo al valor de posición de X del mouse.

Diríamos:

```
ellipse(Posision X del mouse, 150, 40, 40);
```

Evidentemente, Processing no conoce esa expresión, pero si conoce una variable integrada que nos da ese valor.

```
ellipse(mouseX, 150, 40, 40);
```

`mouseX` es una variable de sistema, es decir, integrada en el núcleo de Processing y nos entrega la posicion X de nuestro mouse dentro de la ventana.

> La correspondiente al eje Y es la función `mouseY`.
>
> Otras variables de sistema son: pmouseX, pmouseY, width y heigth

Eventos

Un evento en Processing es un bloque de código que se ejecuta solo una vez, no en el sentido del bloque `setup`, sino en el sentido de que se ejecuta una vez cuando el evento se dispara y espera hasta que sea disparado de nuevo.

```
void mousePressed() {


}
void keyPressed() {


}
```

Por ejemplo, el evento `mousePressed` se dispara cada vez que el mouse es presionado y el evento `keyPressed` se dispara cada vez que una tecla es presionada.