DISEÑO TIPOGRÁFICO EXPERIMENTAL.

DOMESTIKA . MANOLO GUERRERO

Portada: Basada en Sansita una googlefont de Omnibus- Type

Este manual es parte del Diseño Tipográfico Experimental un curso de Manuel Guerrero (Bluetypo) en Domestika.

BlueTypo 2016

INTRODUCCIÓN

Este manual es un material de apoyo realizado para el curso Diseño Tipográfico Experimental impartido por Manuel Guerrero para Domestika. Este curso es de iniciación, va dirigido a todos aquellos interesados en el desarrollo de tipografía experimental, que quieran iniciarse en la programación.

Este manual es una introducción al uso de Processing, un software opensource que permite programar de una manera muy sencilla y rápida. Aquí encontrarás los conceptos básicos de programación y los ejercicios realizados durante este curso.

1. PROCESSING

Processing es un software libre de código abierto desarrollado por Ben Fry y Casey Reas en el Laboratorio de Medios del MIT en el año 2008.

Processing es un software flexible y un lenguaje para aprender el cómo codificar en el contexto de las artes visuales. Hay decenas de miles de estudiantes, artistas, diseñadores, investigadores y aficionados que utilizan Processing para el aprendizaje y la creación de prototipos.

Processing se encuentra disponible para diversos sistemas operativos, como son: Windows, Linux y Mac OS X. Lo puedes descargar de manera gratuita en la siguiente url:
https://www.processing.org/download/

Si tiene problema para instalarlo, puedes ver este link: https://github.com/processing/processing/wiki/FAQ

En el sitio web de Processing encontraras, tutoriales, ejemplos, el manual de referencia, librerías, libros y muchas cosas más. Te recomiendo ver el video Welcome to Processing 3
https://vimeo.com/140600280

La interface es muy sencilla, en la parte superior aparece el nombre del archivo (Sketch), luego viene la barra de herramientas con botones para ejecutar o detener el programa, un botón para depurar el código y un botó que te permite cambiar modos de programación como Pyton, Android, etc.

Enseguida se encuentra la caja de texto, que es el espacio donde estaremos trabajando. Finalmente se encuentra el área de mensajes, la consola y el marcador de errores, estos espacios te ayudan a cuidar la sintaxis, el uso de variables y te advierte sobre los errores encontrados en la programación.

Fig. 1. Interface

Los documentos de Processing se denominan Sketch (boceto) haciendo alusión al lenguaje utilizado en la creación de prototipos. Por default los documentos se guardan la carpeta Documentos (Mac) o Mis Documentos (Windows) pero puedes guardarlos donde prefieras utilizando Guardar como...

1. PRINCIPIOS BÁSICOS DE PROGRAMACIÓN.

Processing es un programa orientado al desarrollo visual, por lo que el uso de coordenadas X, Y es muy frecuente. Es importante comprender que cada píxel tiene un asignada una posición. El punto 0 está ubicado en la esquina superior izquierda de la pantalla, X corresponde a la distancia en el eje horizontal y Y corresponde a la distancia en el sentido vertical.

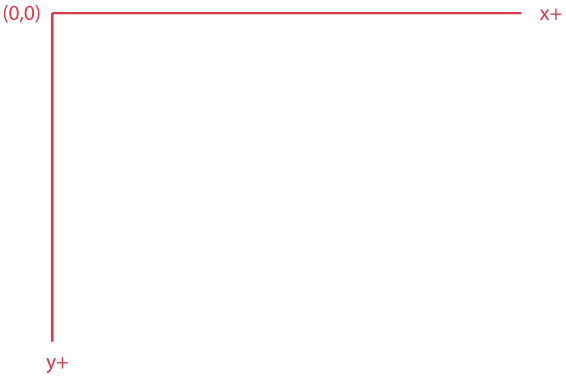


Fig. 2. Coordenadas (x,y)

Existen tres tipos de declaraciones que podemos utilizar, éstas son: Funciones, operadores de asignación y estructuras de control.

Las funciones se componen de nombre, seguido de argumentos descritos entre paréntesis. Un ejemplo de esto es la función `size()` que indica el tamaño de la ventana y los argumentos corresponden al ancho y el alto. `size (ancho,alto);`

```
size(250, 250);
```

Mientras escribes código, es útil utilizar comentarios para recordar o saber que hace cada parte del programa.

```
// Este es un comentario de un renglón.

/* Este es un comentario de más de un renglón.
    Puedes agregar el numero de líneas que
    consideres.
*/
```

Es importante cuidar la sintaxis ya que si existen errores el programa no funcionará. Al programar ten abierta una ventana de tu navegador con la referencia de processing para evitar estos errores.

https://www.processing.org/reference/

1.1 DIBUJANDO FORMAS BÁSICAS.

Dentro de las formas básicas de dibujo 2D en processing podemos encontrar:

Punto

```
point( x, y );
point(30, 20);
```

Linea

Donde x1, y1 corresponden al origen de la línea y x2,y2 a la parte final.

```
line(x1, y1, x2, y2)
line(30, 20, 0, 85, 20, 15);
```

Elipse

Donde a,b corresponden al punto de origen y c,d corresponde al ancho y alto.

```
ellipse(a, b, c, d)
ellipse(56, 46, 55, 55);
```

Rectángulo

Donde a,b corresponden al origen y c,d al ancho y alto, r corresponde a radio y es opcional su uso.

```
rect(a, b, c, d)
rect(a, b, c, d, r)
```

Triángulo

Donde x1, y1 corresponden al primer vértice, x2, y2 al segundo y x3, y3 al tercero.

```
triangle(x1, y1, x2, y2, x3, y3)
triangle(30, 75, 58, 20, 86, 75);
```

más detalles: https://processing.org/reference/

Polígonos rellenos

Para dibujar un polígono con propiedades de color de relleno se utiliza la instrucción beginShape y endShape para indicar los vértices ubicados en x,y. En numero de caras del polígono dependerá del numero de instrucciones vertex.

```
beginShape();
vertex(30, 20);
vertex(85, 20);
vertex(85, 75);
vertex(30, 75);
endShape(CLOSE);
```

1.2 PROPIEDADES DE COLOR Y DE LINEA.

Las propiedades de los objetos que podemos trabajar en processing son:

Opciones de contorno.

`strokeWeight(4); // Grosor de linea de 4 píxels.`
`strokeJoin(opción);` las opciones son MITER angular, BEVEL recortado y ROUND redondeado.

Opciones de color.

El color se puede utilizar en valores de R, G, B o en valores hexadecimales, el valor 255 es el máximo, por lo tanto 255,255,255 corresponde a blanco y el valor 0 a negro. Existe un cuarto valor, este corresponde al nivel de transparencia que va de 0 a 255, siendo 0 la máxima transparencia.

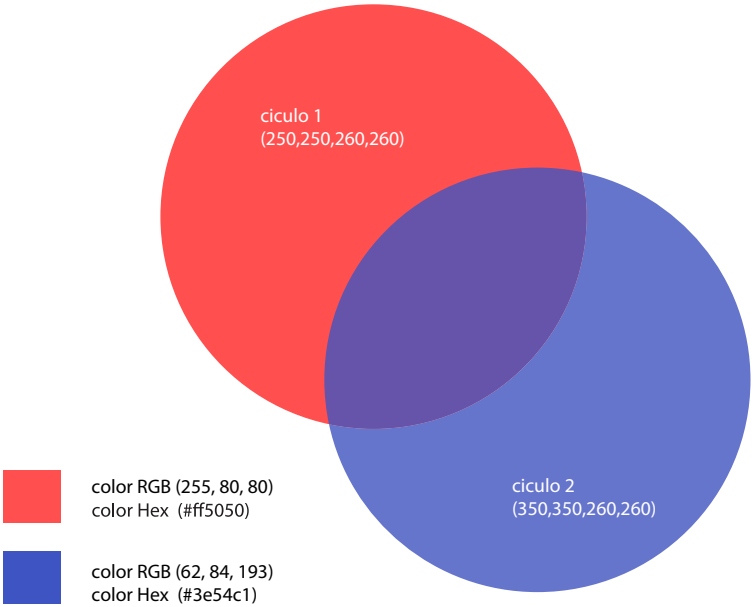
```
background(120,0,0);        //color rojo de fondo.
fill(0, 125,0);             //color azul de relleno.
stroke(0,0,125);            //color verde de borde.
```

más detalles: https://processing.org/reference/beginShape_.html

EJERCICIO

En este primer ejercicio dibujaremos dos figuras básicas, los archivos correspondientes a este ejercicio los encuentras en los recursos de la unidad 3.

Para dibujar en Processing, te recomiendo dibujar primero en papel o en ilustrador antes de hacerlo programando, ya que de este modo puedes previsualizar, asignar tamaños, ubicar elementos en la ventana o definir propiedades de los objetos.



Para dibujar la figura anterior es necesario crear dos elipses, las cuáles ubicaremos en x=250, y=250 y otra en x=350, y=350 con las siguientes propiedades: sin contorno y con colores #ff5050 y #3e54c1 con una transparencia de 50%

```
void setup() {
  size(600,600);    // tamaño de la ventana
  background(255);  // fondo blanco
}

void draw() {
  noStroke();       // sin contorno
  fill(#ff5050);    // color primera elipse
  ellipse(250,250,260,260);      // elipse 1
  fill(#3e54c1,120); // color segunda elipse + alpha
  ellipse(350,350,260,260);      // elipse 2
}
```

1.2. VARIABLES

Una variable es un valor que se almacena en la memoria del ordenador y a lo largo del programa se puede ir actualizando su valor.

```
size(480, 120);            // tamaño de la ventana.
int Y = 60;                // variable Y
int D = 80;                // variable D

ellipse(75, Y, D, D);      // donde Y es 60 y D es 80
ellipse(175, Y, D, D);
ellipse(275, Y, D, D);
```

Entonces aquí se dibujan tres elipses que tienen el mismo valor `Y` y el mismo tamaño, solo cambia su ubicación en x. Pero si cambiamos las variables `Y, D` todas las elipses cambian su ubicación y tamaño al mismo tiempo.

Según el tipo de datos que guardan las variables, se utilizan los siguientes tipos:

```
int (números enteros)
float (número con decimales)
boolean (verdadero o falso)
String (cadena de caracteres)

int a=23;
float a=15.50;
boolean a=true;
String p=”potato”;
```

1.3. CONDICIONALES

Sirven al programador para establecer el flujo del programa, permiten tomar decisiones y realizar una acción. Su sintaxis es la siguiente:

```
if (condición) {
realiza algo();
}
```

```
if (condición) {
realiza algo();
} else {
realiza otra cosa();
}
```

1.4. OPERADORES

Permiten comparar valores numéricos contenidos por variables y en base a ello realizar acciones predeterminadas.

```
== (igualdad)
if (a == b){
realiza algo();
}
```

```
> (mayor a)
if (a > b) {
realiza algo();
}
```

```
>= (mayor o igual a)
if (a >= b) {
realiza algo();
}
```

```
!= (no igualdad)
if (a != b) {
realiza algo();
}
```

```
< (menor a)
if (a < b){
realiza algo();
}
```

```
<= (menor o igual a)
if (a <= b) {
realiza algo();
}
```

1.5 CICLOS

Los ciclos son estructuras de repetición nos permiten ejecutar varias veces algo, permiten realizar acciones repetitivas de una mancera sencilla y rápida. Para realizar ciclos se pueden utilizar las instrucciones `While` y `for`.

```
while (condición) {
realiza algo();
}
```

```
for (inicio; test; actualización) {
realiza algo();
}
```

Por ejemplo para dibujar un patrón de líneas horizontales con una separación de 5 píxeles, con las instrucción `while` se dibujaría de la siguiente manera:

```
int i=0;                    // declara la variable i
while (i < height) {        // mientras i es menor h
line (0, i, width, i);      // dibuja una linea
i = i + 5;                  // incrementa i 5 píxeles
}
```

La instrucción `for` es una forma mas sintética de realizar lo mismo ya que en una linea se inicia, comprueba y actualiza la variable:

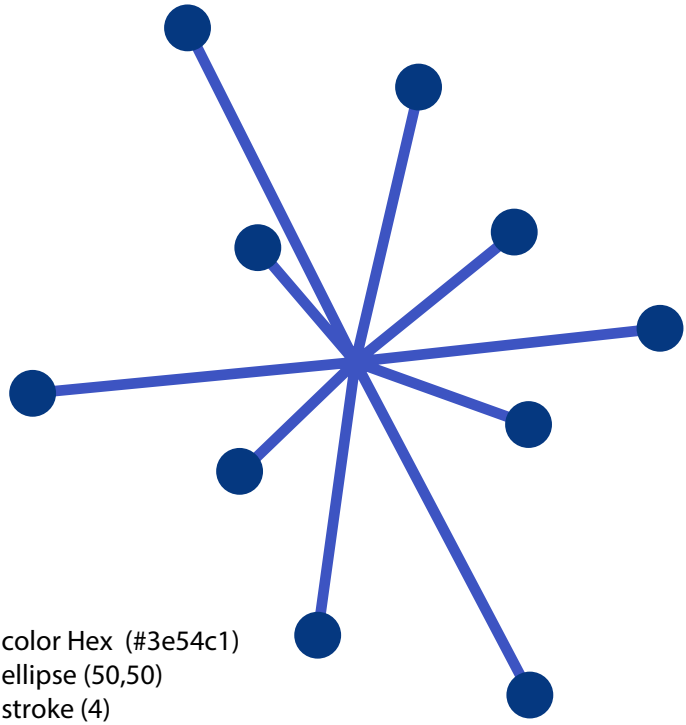
```
for (int i = 0;  i< height; i=i+5) {
line (0, i, width, i);
}
```

RANDOM

Para este curso, utilizaremos la instrucción `random` la cuál arroja valores de manera aleatoria y se utiliza de la siguiente manera:

```
random(valor);
random(valor1, valor2);
```

Para dibujar la siguiente figura es necesario crear 10 elipses y 10 líneas de color #3e54c1 con un grosor de 4 píxeles. Las elipses y un extremo de las líneas tienen las misma ubicación, mientras que el otro extremo va al centro.



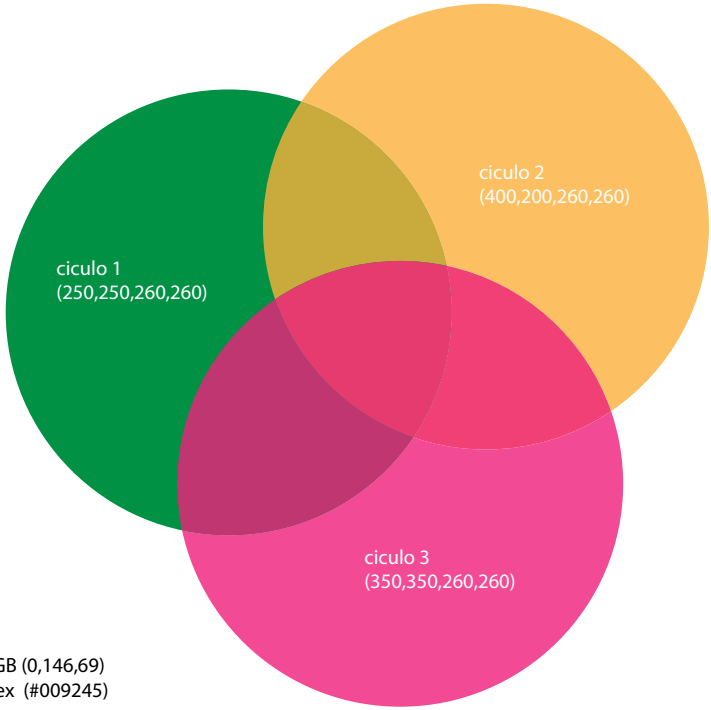
color Hex (#3e54c1)
ellipse (50,50)
stroke (4)

```
// inicia la variable i
int i=0;

// define tamaño de la ventana y color de fondo
void setup() {
  size(600,600);
  background(255);
}

void draw() {
  if (i<10) {
  float ranx = random(-250,250)+300;
  float rany = random(-250,250)+300;
  fill(#3e54c1);
  noStroke();
  ellipse(ranx,rany,50,50);
  stroke(#3e54c1);
  strokeWeight(4);
  line(300,300,ranx,rany);
  i++;
  }
}
```

Con esto ya estarás listo para dibujar el siguiente ejercicio:



color RGB (0,146,69)
color Hex (#009245)

color RGB (251,176,59) transparencia 120
color Hex (#fbb03b)

LIBRERÍAS

Geomerative. Es una librería desarrollada para Processing, ésta extiende las operaciones geométricas 2D. Incluye interpretes de TrueType y SVG lo que facilita su uso en proyectos de tipografía generativa.

http://www.ricardmarxer.com/geomerative/

Básicamente Geomerative se compone de cuatro partes:

1. Se importa la librería.
2. Se declaran los objetos a utilizar.
3. Se inicia la librería y se especifican las propiedades.
4. Se dibuja el objeto.

1. Importar la librería.
Esto se realiza en las primeras líneas de código con la siguiente instrucción:

```
import geomerative.*;
```

2. Declarar los objetos.
Enseguida se declaran los objetos a utilizar, para que estén accesibles en las funciones `setup()` y `draw()`

```
RShape grp;
```

3. Iniciar la librería.
Para iniciar la librería y declarar las propiedades del objeto, es necesario realizarlo en la función `setup()` con la siguiente instrucción:

```
RG.init(this);
grp = RG.getText("Hi", "FreeSans.ttf", 72, CENTER);
```

4. Dibuja el objeto.

Para dibujar el objeto, en la función `draw()` hay que utilizar la siguiente instrucción:

```
grp.draw();
```

Esto lo podemos ver en el siguiente ejemplo:

```
import geomerative.*;
RShape grp;

void setup(){
  size(600,400);
  RG.init(this);

  background(255);
  fill(255, 102, 0);
  stroke(0);
  grp = RG.getText("Hi", "FreeSans.ttf", 72, CENTER);
}

void draw(){
  background(255);
  translate(width/2, height/2);
  grp.draw();
}
```

PLANTILLA DEL CURSO.

Para este curso he preparado una plantilla donde ya está programada la librería Geomerative, la cuál explico enseguida:

Comenzamos importando la librería Geomerative y PDF

```
import geomerative.*;
import processing.pdf.*;
```

Declaramos una variable que utilizaremos para exportar a PDF.

```
boolean record;
```

Declaramos los objetos font; (tipo de letra), SampleText; (texto) y pnts; (segmentos en el contorno de cada letra)

```
RFont font;
String SampleText = "A B C";
RPoint[] pnts;
```

En la función `setup()` iniciamos la librería y declaramos las propiedades del objeto `font;`

```
  RG.init(this);
  font = new RFont("FreeSans.ttf", 80, RFont. LEFT) ;
```

Después definimos la separación de los segmentos.

```
  RCommand.setSegmentLength(16);
  RCommand.setSegmentator(RCommand.UNIFORMLENGTH);
```

Comprobamos si la longitud de la cadena SampleText es mayor a `0` entonces se genera un el grupo `grp;` se asigna el grupo al texto `SampleText` y se colocan los puntos en su contorno.

```
if ( SampleText. length () > 0) {
  RGroup grp;
  grp = font.toGroup(SampleText);
  // coloca los puntos a lo largo del outline
  pnts = grp.getPoints();
```

En la función `draw()` comenzamos comprobando si la variable `record` es verdadera, entonces genera un pdf y si es falsa, no hace nada y sigue corriendo el programa.

```
void draw() {
  if (record) {
    beginRecord(PDF, "pdf/typo-####.pdf");
  }
  background(#3e5497); // color de fondo
  translate (76, 450);  // posición del texto
```

En esta parte de hace un ciclo para que lo que realicemos en la función `dibuja1();` se realice en cada uno de los segmentos del contorno del texto `SampleText.`

```
  for (int i=0; i<pnts.length; i++) {
    pushMatrix();
    translate(pnts[i].x, pnts[i].y);
    dibuja1();
    popMatrix();
  }
```

Ahora comprobamos si la variable `record` es verdadera, dejamos de grabar y cambiamos la variable a false.

```
  if (record) {
    endRecord();
    record = false;
  }
```

Si quieres exportar frame por frame, solo quita el comentario // y mientras el programa esté corriendo generará archivos .tif Los signos #### sirven para que nombre los archivos de manera secuencial, para que no se reescriban unos sobre otros.

```
  //saveFrame("img/typo-######.tif");
```

Ahora cuando presionas la tecla "e" por medio de la función `keyPressed()` la variable record cambia a verdadero. De esta manera cada que presiones "e" se exporta un archivo PDF.

```
void keyPressed() {
  if (key=='e') {
    record = true;
  }
```

La plantilla queda de la siguiente manera:

```
import geomerative.*;
import processing.pdf.*;
boolean record;
```

```
RFont font;
String SampleText = "A B C";
RPoint[] pnts;

void setup() {
  size (1400, 600);
  RG.init(this);
  font = new RFont("FreeSans.ttf", 80, RFont. LEFT) ;
  RCommand.setSegmentLength(16);
  RCommand.setSegmentator(RCommand.UNIFORMLENGTH);

  if ( SampleText. length () > 0) {
    RGroup grp;
    grp = font.toGroup(SampleText);
    pnts = grp.getPoints();
  }
}

void draw() {
  if (record) {
    beginRecord(PDF, "pdf/typo-####.pdf");
  }
  background(#3e5497); // color de fondo
  translate (76, 450);  // posición del texto
  for (int i=0; i<pnts.length; i++) {
    pushMatrix();
    translate(pnts[i].x, pnts[i].y);
    dibuja1();
    popMatrix();
  }
  if (record) {
    endRecord();
    record = false;
  }
  //saveFrame("img/typo-######.tif");
}
```

```
void keyPressed() {
  if (key=='e') {
    record = true;
  }
}
```

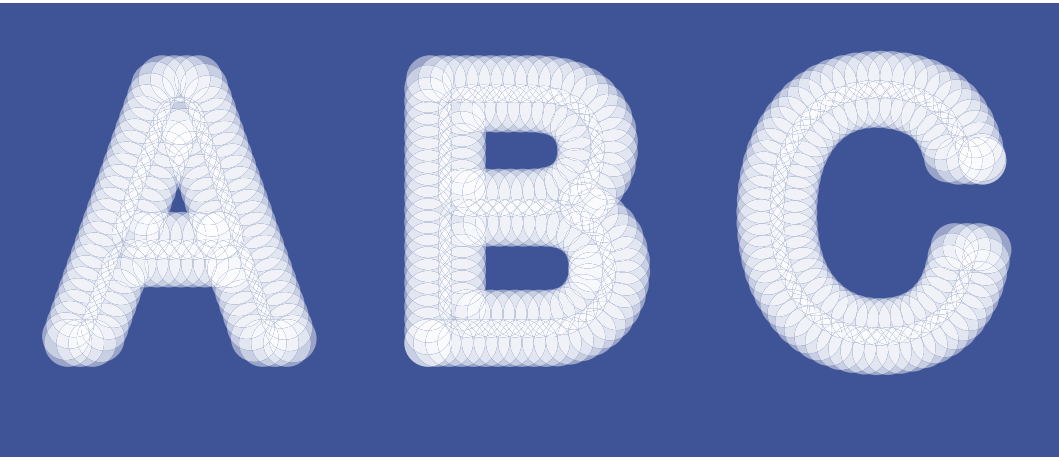
Finalmente la función `dibuja1();` dibuja líneas en cada punto de los segmentos del contorno y su distancia está determinada por la posición del mouse.

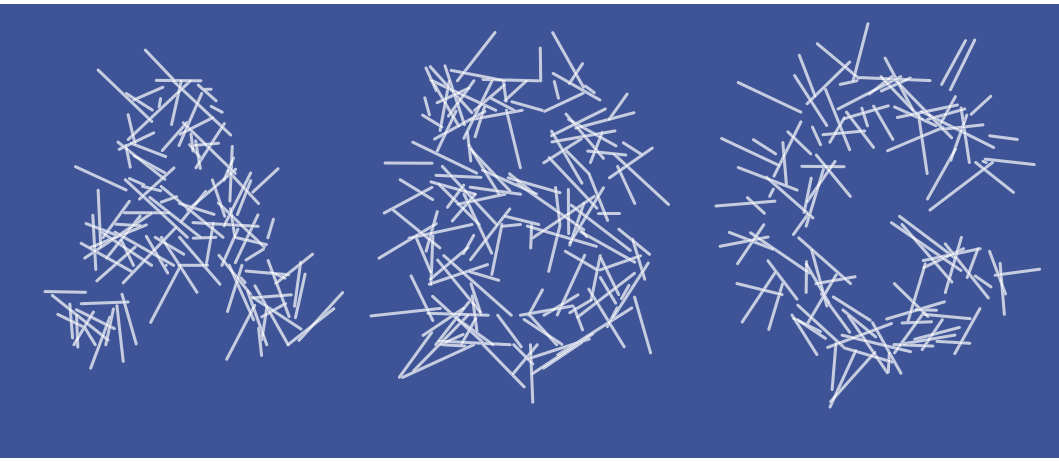
```
void dibuja1() {
  strokeWeight(6); //Ancho de linea
  stroke(255, 180); //Color de linea
  line(0,0,mouseX,mouseX);
}
```

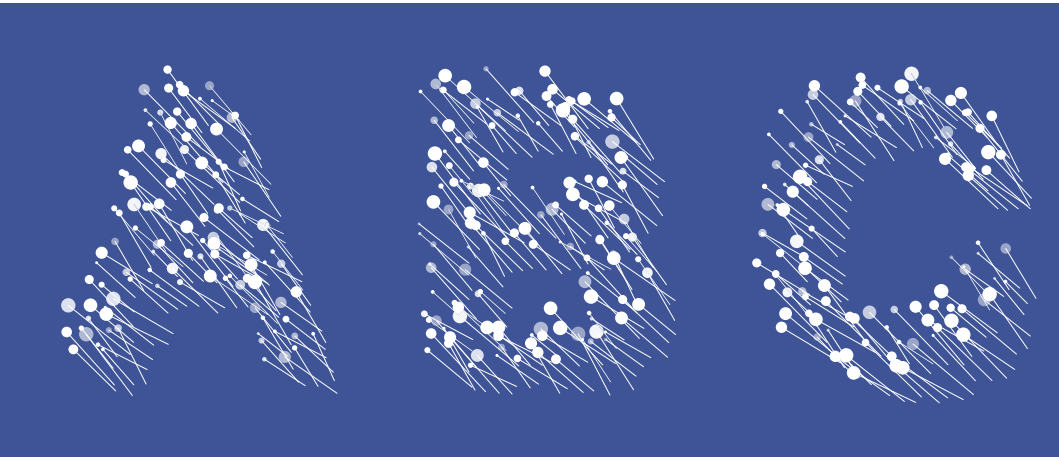
En esta plantilla se incluyen otros ejemplos, para el proyecto final de este curso puedes realizar modificaciones o crear nuevos programas que te permitan generar nuevas formas de tipografía, también puedes cambiar el tipo de letra, solo es necesario que sea un archivo ttf y si eres un experto hasta podrías hacer tu propio programa. Dudas y comentarios los vemos a través del foro.

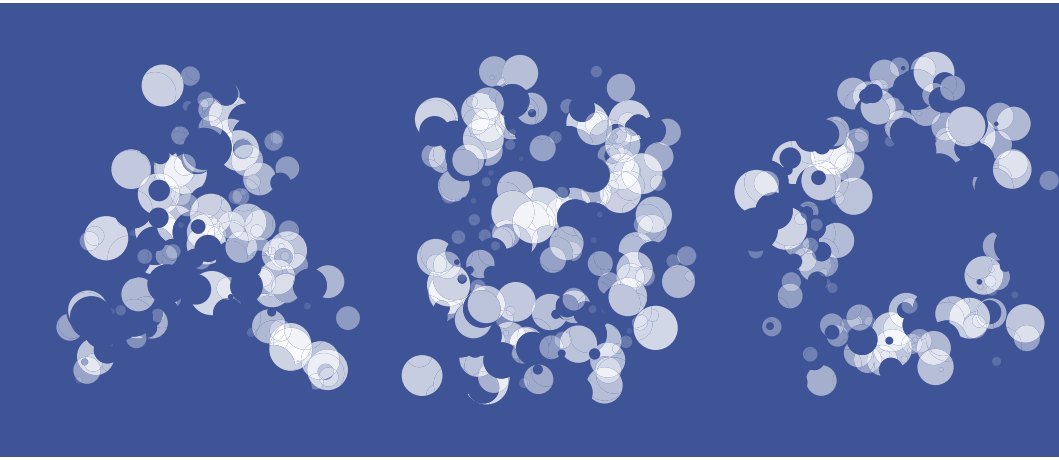
dibuja1();
dibuja2();
dibuja3();

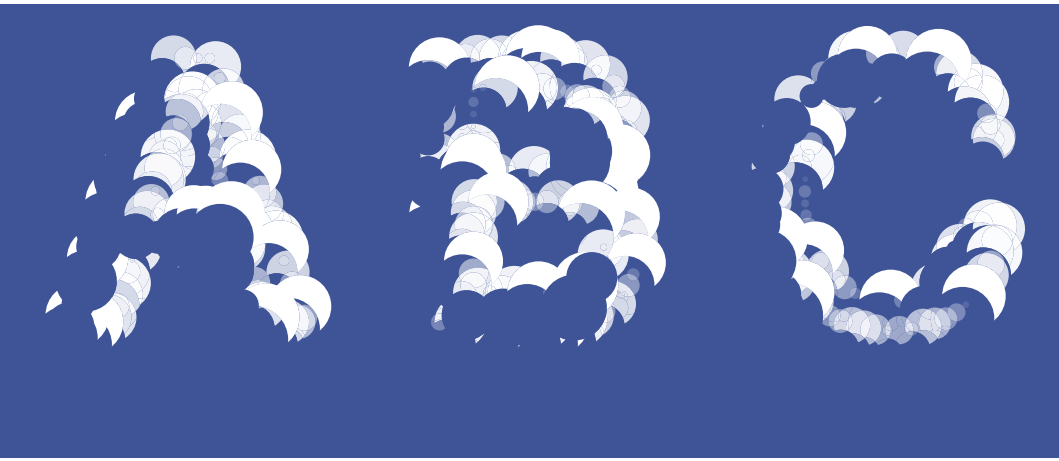

ABC


ABC









RECURSOS ADICIONALES

Processing:
Sitio oficial: https://processing.org/
Welcome to processing: http://hello.processing.org/
Referencia: https://processing.org/reference/
Librerías: https://processing.org/reference/libraries/

Tutoriales.
Anatomía, J David Eisenberg.
https://processing.org/tutorials/anatomy

Fun programming.
http://funprogramming.org/

Introduction to coding.
https://www.youtube.com/playlist?list=PLG8vJUg0ALmssb3E-Yekn-314MyeVCX2-7

Tutoriales de processing
https://processing.org/tutorials/

Video tutorial de processing
http://www.plethora-project.com/education/2011/09/12/processing-tutorials/

Joan Soler-Adillon. Tutorial de Processing
http://www.joan.cat/processing/

```
dibuja4();
dibuja5();
dibuja6();
```