Processing
CHEAT SHEET

MARTÍN JEREZ

Processing CHEAT SHEET

MARTÍN JEREZ

Estructura básica
Esta es la estructura básica de un sketch.

```
void setup(){
//Se ejecuta una vez, al principio.
}
void draw(){
//Se ejecuta todo el tiempo a una
//determinada cantidad de FPS.
}
```

Tipos de variables

`int`
Variables enteras negativas y positivas.

`float`
Variables decimales o flotantes, negativas y positivas.

`boolean`
Variables de valor: TRUE o FALSE.

`color`
Variables para guardar color.

`char`
Variables para guardar caracteres.

`string`
Variables para guardar cadenas de texto.

Funciones básicas

`size(ancho,alto);`
Configura el tamaño de la ventana de trabajo en pixeles.

`background(color);`
Indica el color con el que se dibuja el fondo.

`smooth();`
Aplica un suavizado al sketch (antialiasing).

`frameRate(fps);`
Configura los frames por segundo (FPS) de la aplicación.

`println(string);`
Imprime una cadena de texto en la consola inferior.

Random y Noise

`random(low,high);`
Nos devuelve un valor random entre dos umbrales.

`randomSeed(seed);`
Cambia el seed del random.

`noise(valor);`
Nos devuelve un valor de una secuencia con Perlin Noise.

`noiseDetail(octaves);`
Ajusta el nivel de detalle producido por la función noise.

`noiseSeed(seed);`
Cambia el seed del noise.

Variables globales
Son variables que podemos llamar todo el tiempo.

`width`
Devuelve el valor en pixeles del ancho del sketch.

`height`
Devuelve el valor en pixeles del alto del sketch.

`mouseX`
Devuelve la posición del puntero del mouse en el eje X.

`mouseY`
Devuelve la posición del puntero del mouse en el eje Y.

`pmouseX`
Devuelve la posición anterior del puntero del mouse en el eje X.

`pmouseY`
Devuelve la posición anterior del puntero del mouse en el eje Y.

`frameCount`
Devuelve el frame actual del sketch.

`frameRate`
Devuelve la velocidad en FPS de nuestro sketch.

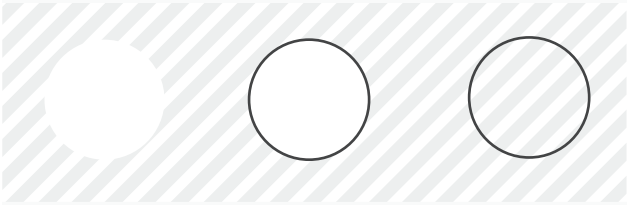
fill(),noFill(),stroke(),noStroke()
Son instrucciones para setear rellenos y bordes.

`fill(color);`
Configura el color de relleno de lo que se dibuje posteriormente.

`noFill();`
Indica falta de relleno de lo que se dibuje posteriormente.

`stroke(color);`
Configura el color de borde de lo que se dibuje posteriormente.

`noStroke();`
Indica falta de borde en lo que se dibuje posteriormente.



```
fill(255);
noStroke();
```

```
fill(255);
stroke(0);
```

```
noFill();
stroke(0);
```

Funciones de color

`colorMode(mode);`
Nos permite cambiar el modo de color por ejemplo **HSB** o **RGB**.

`red(color);`
Nos devuelve el valor de rojo de un color.

`green(color);`
Nos devuelve el valor de verde de un color.

`blue(color);`
Nos devuelve el valor de azul de un color.

`hue(color);`
Nos devuelve el tono de un color.

`saturation(color);`
Nos devuelve la saturación de un color.

`brightness(color);`
Nos devuelve el brillo de un color.

`alpha(color);`
Nos devuelve el valor de transparencia de un color.

`lerpColor(color1,color2,cantidad);`
Nos devuelve un momento de la mezcla de dos colores.

-Hay dos formas de escribir programas sin errores; sólo la tercera funciona. Alan J. Perlis

«No te preocupes si no funciona bien. Si todo lo hiciera, no tendrías trabajo.» -La ley de Mosher sobre la ingeniería de Software

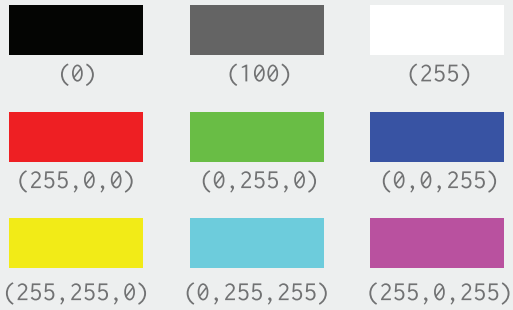
Manejo del color

Estas son las formas de pasar colores en Processing.

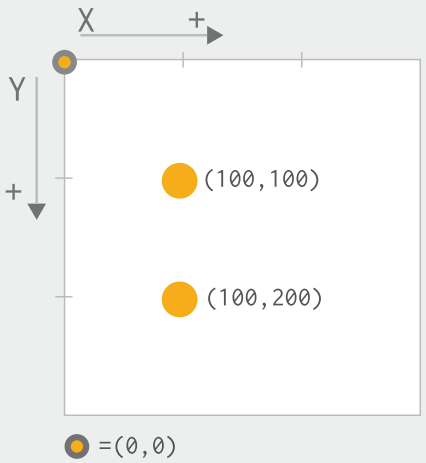
```
//La escala de valores va de 0 a 255.
color(grayscale);
color(grayscale,alpha);
color(red,green,blue);
color(red,green,blue,alpha);
```

Dependiendo de la cantidad de parámetros pasados, las componentes cambian.

Ejemplo de colores



Sistema de coordenadas



El (0,0) en un sketch de processing es la esquina superior izquierda, este eje es el que se cambia cuando hacemos un translate() o un rotate().

La unidad mínima dentro del sistema de coordenadas de una pantalla de computadora, es el pixel.

El sistema de coordenadas tiene por tamaño en pixeles los valores que le pasamos con el size(ancho,alto).

Operaciones con matrices

`pushMatrix();`
Guarda la matriz actual. Es decir los valores de translate , rotate y scale. A cada pushMatrix( ) le corresponde un popMatrix( ) final.

`popMatrix();`
Permite volver a la ultima matriz guardada. Se necesita un pushMatrix( ) previo para volver a una matriz anterior.

`printMatrix();`
Imprime la matriz actual en la consola.

`translate(posx,posy);`
Mueve el punto de anclaje a un determinado punto. Después de esta función el (0,0) es la posición pasada.

`rotate(radians);`
Cambia la rotación del plano en base al eje.

`scale(x,y);`
Escala el plano, afecta a los tamaños de todo lo dibujado en el plano. También pueden pasarse : **scale(x,y,z) o scale(multiplo).**

`shearX(radians);`
Aplica un shear en el eje X.

`shearY(radians);`
Aplica un shear en el eje Y.

`rotateX(radians);`
Aplica una rotación en el eje X . Es válido sólo en ambientes 3D.

`rotateY(radians);`
Aplica una rotación en el eje Y . Es válido sólo en ambientes 3D.

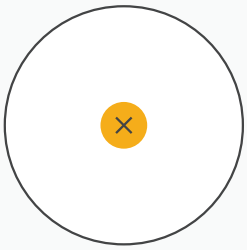
`rotateZ(radians);`
Aplica una rotación en el eje Z . Es válido sólo en ambientes 3D.

`pushStyle();`
Guarda el estilo actual de fill(), stroke(), tint(), strokeWeight(), strokeCap(), strokeJoin(), imageMode(), rectMode(), ellipseMode(), shapeMode(), colorMode(), textAlign(), textFont(), textMode(), textSize(), textLeading(), emissive(), specular(), shininess(), ambient().

`popStyle();`
Vuelve al estado de estilo anterior.

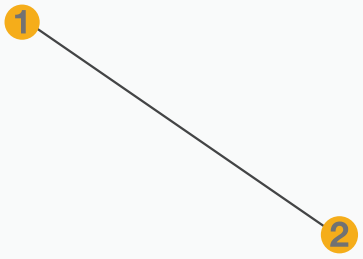
Geometria básica

Punto de anclaje



`ellipse(posx,posy,ancho,alto);`
Dibuja una elipse en la posición pasada, con el tamaño indicado.

`rect(posx,posy,ancho,alto);`
Dibuja un rectángulo en la posición pasada, con el tamaño indicado.



`line(posx1,posy1,posx2,posy2);`
Dibuja una línea en base a dos puntos indicados.

Otras formas primitivas

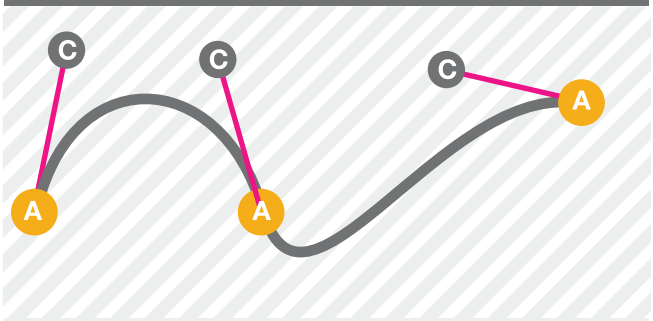
`point(posx,posy);`
Dibuja un punto en pantalla.

`quad(x1,y1,x2,y2,x3,y3,x4,y4);`
Dibuja un cuadrilatero en base a las cuatro posiciones que le pasemos.

`arc(posx,posy,width,height,startangle,endangle);`
Dibuja un arco en base a, la posición , el tamaño y dos variables angulares en radianes.

`triangle(x1,y1,x2,y2,x3,y3);`
Dibuja un triángulo en base a las tres posiciones pasadas.

Bezier y curvas



`bezier(x1, y1, x2, y2, x3, y3, x4, y4)`
Dibuja una curva Bezier. 1 y 4 son los anchor point , 2 y 3 son los control point.

`bezierDetail(nivel);`
Configura el nivel de detalle de las curvas Bezier.

`bezierTangent(a, b, c, d, momento);`
Devuelve la tangente del momento de una Bezier.

`bezierPoint(a, b, c, d, momento);`
Devuelve la posición en un eje del momento de una Bezier.

`curve(x1, y1, x2, y2, x3, y3, x4, y4);`
Dibuja una curva. 1 y 4 son los control point, 2 y 3 los point.

`curveTightness(tightness);`
Establece la tensión de las curvas posteriores.

`curvePoint(a, b, c, d, t);`
Devuelve la posición en un eje del momento de una curva.

`curveTangent(a, b, c, d, t);`
Devuelve la tangente del momento de una curva.

`curveDetail(detail);`
Setea el nivel de detalle de las curvas.

beginShape( ) y endShape( )

`beginShape();`
Comienza a escuchar vértices para armar una forma. Su uso es finalizado al llamar un endShape( ). **Puede pasarse un modo.**

`endShape();`
Termina de escuchar los vértices pasados.

`vertex(posx,posy);`
Dibuja un vértice en la posición indicada.

`bezierVertex(x2, y2, x3, y3, x4, y4);`
Define un vértice en base a una curva Bezier.

`curveVertex(x, y);`
Define un vértice en base al punto de una curva.

`texture(PImage);`
Determina la textura de loque se dibuja.

`beginContour();`
Comienza a escuchar vértices para restar una forma a otra.

`endContour();`
Deja de escuchar los vértices de resta.

* Modos disponibles en beginShape POINTS, LINES, TRIANGLES, TRIANGLE_FAN, TRIANGLE_STRIP, QUADS , y QUAD_STRIP.

"La máquina analítica teje patrones algebraicos, al igual que el telar de Jacquard teje flores y hojas." Ada Lovelace

Un lenguaje de programación es de bajo nivel cuando requiere que prestes atencion a lo irrelevante. - Alan J. Perlis.

Estructura de una función

```
//Crear la funcion
void insultar(){
println("FUCK U");
}

//Llamar la funcion
insultar();
```

Estructura de una clase

```
class NombreClase{
NombreClase(/*Variables*/){
//Constructor
}

void nombreMetodo(/*Variables*/){

}

}

//Declarar un objeto
NombreClase miClase;

void setup(){
//Inicializar el objeto
miClase = new NombreClase(/*Variables*/);
}
void draw(){
//Llamar un metodo del objeto.
miClase.nombreMetodo();
}
```

* Las clases pueden tener o no variables para ser inicializadas de la misma manera que los métodos.

Ciclos For

Son estructuras que se repiten con una condición.

```
//Ciclo for simple
for(int i = 0;i<condicion;i++){
//Esto se repite i veces
}

//Ciclo for anidado
for(int i = 0;i<condicion;i++){
    for(int j = 0;j<condicion;j++){
    //Esto se repite i*j veces
    }
}
```

* Dentro de la estructura podemos utilizar las variables de los índices para aprovechar las repeticiones.

Operadores condicionales

<	<=	==	>	>=	!=
Menor que	Menor o igual	Igual a	Mayor que	Mayor o igual	Distinto a

Operadores lógicos

Son conectores entre condiciones.

&&	||	!
AND	OR	NOT

Estructura condicional

```
if(condicion1){
//Si se cumple la condicion 1.
}else if(condicion2){
//Si se cumple la condicion 2.
}else{
//De lo contrario.
}
```

* Las condiciones se arman utilizando operadores condicionales y/o lógicos.

Estructura while

```
while(condicion1){
//Si la condicion1 es verdadera.
}
```

PLEASE SHARE THIS

Cualquier tecnología lo suficientemente avanzada es indistingible de la magia. - Arthur C. Clarke

El lenguaje moldea la forma en que pensamos y determina en qué podemos pensar - B.L. Whorf

Mostrar una imágen
Las imágenes van dentro de la carpeta data del sketch.

```
PImage img;

void setup() {
  img = loadImage("nombredearchivo.jpg");
}

void draw() {
  image(img, 0, 0);
}
```

Formatos soportados: **JPG** , **GIF** , **TGA** y **PNG**.

Mostrar un texto
La tipografía va dentro de la carpeta data del sketch.

```
PFont font;

void setup() {
  font = loadFont("Helvetica-32.vlw");
  textFont(font, 32);
}

void draw() {
  text("Hello", 0, 0);
}
```

Para crear la tipografía en formato **VLW** se puede utilizar la herramienta integrada en: Tools / Create Font...

Mostrar un shape
Los vectores van dentro de la carpeta data del sketch.

```
PShape mishape;

void setup() {
  mishape = loadShape("miShape.svg");
}

void draw() {
  shape(mishape, 0, 0);
}
```

Formato soportado: **SVG**.

Funciones de imágen

`image(img,posx,posy,width,height);`
Nos permite dibujar una imágen en pantalla.

`loadImage(fileName);`
Inicializa un PImage pasándole la ubicación de la imágen.

`requestImage(fileName);`
Inicializa un PImage en un thread separado.

`tint(color);`
Configura el tinte de la imágen a dibujar.

`noTint();`
Quita el tinte configurado.

`saveFrame(filename);`
Nos guarda un screenshot de lo que vemos en el sketch.

Funciones de texto

`text(string,posx,posy);`
Nos permite mostrar un texto en pantalla.

`loadFont(fileName);`
Inicializa un PFont pasándole la ubicación de la tipografía.

`textFont(font,size);`
Configura la tipografía y el tamaño de lo que se dibuje.

`textAlign(mode);`
Configura el tipo de alineación: LEFT , CENTER o RIGHT

`textLeading(size);`
Configura la distancia entre líneas de texto.

Easing target
Nos sirve para suavizar el paso de valores.

```
float x;
//Valor de suavizado
float easing = 0.05;

void setup() {
  size(220, 120);
}

void draw() {
  background(0);
  float targetX = mouseX;
  x += (targetX - x) * easing;
  ellipse(x, 40, 10, 10);
}
```

Ejemplo tomado del libro **"Getting started with Processing"** de Reas & Fry. O'Reilly / Make 2010

"Los programas deben ser escritos para que la gente los lea y despues, talvez, para que las máquinas los ejecuten." - Abelson / Sussman

"Ni hay, ni hubo jamás, ni habrá algún lenguaje de programación que te haga un poquitito más difícil siquiera escribir código incorrecto." - Ley de Flon

Captura de eventos

`void mousePressed()`
Se ejecuta cuando se presiona el mouse.

`void mouseClicked()`
Se ejecuta cuando el mouse fue presionado y soltado.

`void mouseMoved()`
Se ejecuta cada vez que el mouse se mueve sin estar presionado.

`void mouseDragged()`
Se ejecuta cada vez que el mouse se mueve estando presionado.

`void mouseReleased()`
Se ejecuta cuando el mouse deja de estar presionado.

`void keyPressed()`
Se ejecuta cuando se presiona una tecla.

`void keyTyped()`
Se ejecuta cuando se presiona una tecla , exceptúando por ejemplo SHIFT , CTRL , o ALT.

`void keyReleased()`
Se ejecuta cada vez que se deja de presionar una tecla.

keyPressed (Boolean)
Nos devuelve si hay o no una tecla presionada.

```
void draw() {
  if(keyPressed == true) {
    fill(0); //Esta presionada
  } else {
    fill(255); //No esta presionada
  }
  rect(25, 25, 50, 50);
}
```

* Es una variable que podemos consultar para saber si se presionó una tecla.

mousePressed (Boolean)
Nos devuelve si esta o no el mouse presionado.

```
void draw() {
  if(mousePressed == true) {
    fill(0); //Esta presionado
  } else {
    fill(255); //No esta presionado
  }
  rect(25, 25, 50, 50);
}
```

* Es una variable que podemos consultar para saber si se presionó el mouse.

key
Nos devuelve la última tecla presionada.

```
void draw() {
  if (keyPressed) {
    if (key == 'b' || key == 'B') {
      // Si esta presionada la tecla.
    }
  } else {
    // De lo contrario.
  }
}
```

* El key distingue si una tecla es minúscula o mayúscula.

keyCode
Es usada para detectar teclas especiales.

```
void keyPressed() {
  if (key == CODED) {
    if (keyCode == UP) {
    // Si esta presionada la tecla arriba.
    } else if (keyCode == DOWN) {
    // Si esta presionada la tecla abajo.
    }
  } else {
    // De lo contrario.
  }
}
```

* Otras teclas: **BACKSPACE** , **TAB** , **ENTER** , **RETURN** , **ESC** , **DELETE** , **RIGHT** , **LEFT**.

"Las computadoras son buenas siguiendo instrucciones, pero no leyendo tu mente." - Donald Knuth

"Cuando alguien diga: "Quiero un lenguaje de programación en el que solo necesite decir lo que quiero.", dale un chupetín." - Alan Perlis *

Arrays en una sola línea.

```
int [] arrayInt = { 43, -2 , 8 , 1};

println(arrayInt[0]);// Imprime 43
println(arrayInt[1]);// Imprime -2
println(arrayInt[2]);// Imprime 8
```

El índice de los arrays arranca desde 0.

Arrays y Ciclos For.
Es la forma de iterar los espacios del array.

```
//Declaramos un array.
int [] arrayInt;

void setup(){
//Le damos un tamanio al array.
arrayInt = new int[50];

//Inicializamos todos los espacios.
 for(int i = 0; i<arrayInt.length; i++){
 arrayInt[i] = i;
 }
}
void draw(){
//Imprimir todos los espacios.
 for(int i = 0; i<arrayInt.length; i++){
 println(arrayInt[i]);
 }
}
```

Los ciclo for nos permiten inicializar rápidamente todos los espacios del array.

Funciones de Arrays

`append(array,value);`
Agrega un valor a un arreglo.

`arrayCopy(src, srcPos, dst, dstPos, length);`
Copia un array o parte de uno, hacia otro.

`concat(a,b);`
Concatena dos arrays.

`expand(array,newSize);`
Expande el valor de un array.

`reverse(array);`
Revierte el orden del array.

`shorten(array);`
Resta un espacio a un array.

`sort(array);`
Ordena un array de menor a mayor.

`splice(array,value/array,index);`
Inserta un valor o un array dentro de otro en un índice.

`subset(array, start, count)`
Extrae un array en base a los espacios de otro.

Arrays de objetos
Podemos armar arrays de una clase en particular.

```
//Declaramos un array de objetos.
Particula [] particulas;

void setup(){
//Le damos un tamanio al array.
particulas = new Particula [50];

//Inicializamos todos los espacios.
for(int i = 0; particulas.length; i++){
particulas[i] = new Particula();
}

}
void draw(){
//Llamamos una funcion de los objetos.
for(int i = 0; i<particulas.length; i++){
particulas[i].dibujar();
}
}
```

Arrays de dos dimensiones.
Son arrays para trabajar con dos valores.

```
//Declaramos un array.
int [][] array2D;

void setup(){
//Le damos un tamanio al array.
array2D = new int[width][height];

//Inicializamos todos los espacios.
for(int i = 0; i<width; i++){
  for(int j = 0; j<height; j++){
  array2D[i][j] = int(random(100));
  }
}
}

void draw(){
//Inicializamos todos los espacios.
for(int i = 0; i<width; i++){
  for(int j = 0; i<height; j++){
  println(array2D[i][j]);
  }
}
}
```

Para recorrer arrays de más de una dimensión necesitamos usar ciclos for anidados.

"Primero, soluciona el problema. Luego, escribe el codigo" - John Johnson

La vida sería mucho más sencilla si pudiéramos echar un vistazo al código fuente.