PPEM 2020

Imágenes

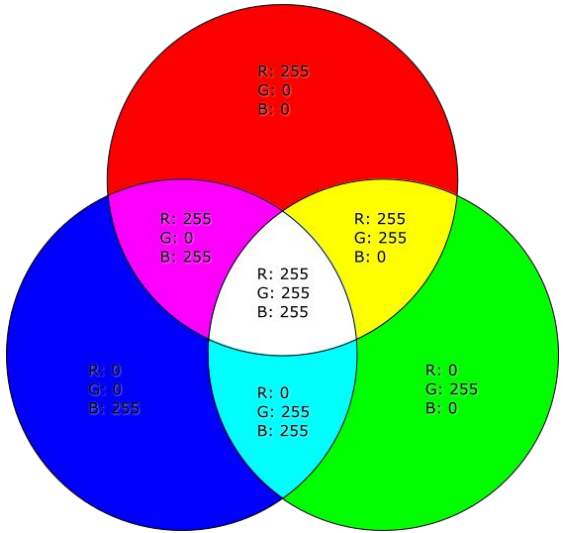
Repaso

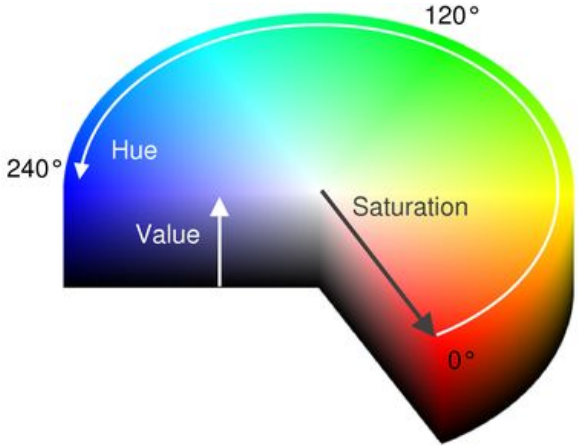
Figuras (quad, ellipse, arc, triangle, vertex)

Espacios de colores

Mouse y teclado

Librerías externas





Imágenes

PGraphics

PShape para visualizar formato svg

Imágen como array de píxeles

Funciones para acceder y manipular imágenes

Imágenes en movimiento (video)

PGraphics

createGraphics(w, h)

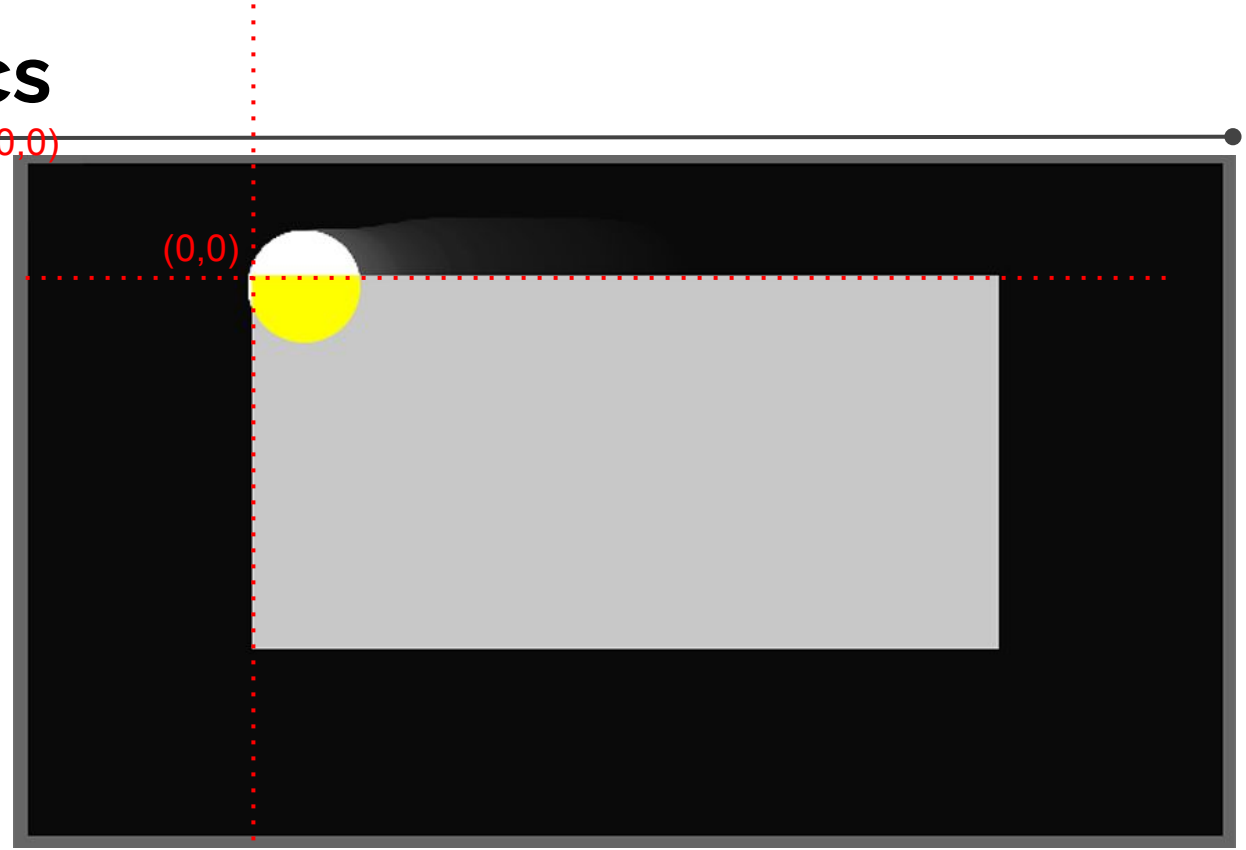
createGraphics(w, h, renderer)

createGraphics(w, h, renderer, path)

Renderer -> P2D, P3D (para sketches) y SVG, PDF (para dibujos offline, requieren “path” al archivo donde se guarda el dibujo)

Crear PGraphics

```
PGraphics pg;
void setup() {
 size(640, 360);
 pg = createGraphics(400, 200);
}
void draw() {
 fill(0, 12);
 rect(0, 0, width, height);
 fill(255);
 noStroke();
 ellipse(mouseX, mouseY, 60, 60);
 pg.beginDraw();
 pg.background(200);
 pg.fill(255,255,0);
 pg.noStroke();
 pg.translate(-120,-60);
 pg.ellipse(mouseX, mouseY, 60, 60);
 pg.endDraw();
 image(pg, 120, 60);
}
```



```
shape(shape)
shape(shape, x, y)
shape(shape, x, y, w, h)
```

Cargar imagen svg como PShape

```
PShape s;

void setup() {
  size(500, 500);
  s = loadShape("hs.svg");
}

void draw() {
  shape(s, 10, 10, width-10, height-10);
}
```

Carga svg (creados en Inkscape and Adobe Illustrator) o obj (modelos 3D)

PShape

isVisible()

setVisible()

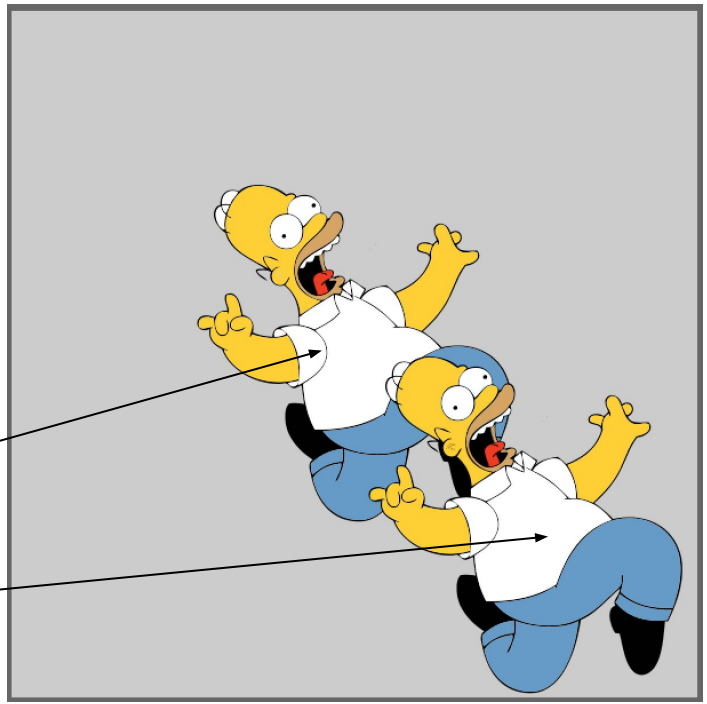
disableStyle()

enableStyle()

scale()

+

shapeMode: CENTER o CORNER



PShape para crear imágenes

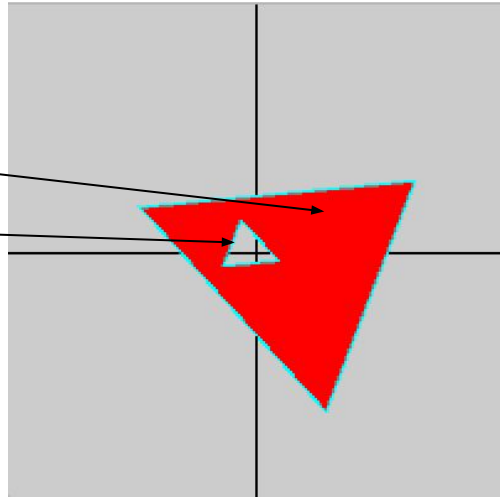
https://processing.org/reference/PShape.html

```
beginShape()    // para crear un forma customizada. Puede definirse vértice por vértice o como conjunto de formas
endShape()      // para finalizar la creación de la forma customizada
beginContour() // con esta función se puede "recortar" una forma definida usando beginShape()
endContour()   // para finalizar la creación del contorno
setFill()       // definir color de relleno de la forma. Se usa después de endShape, entre beginShape y endShape se usa fill()
setStroke()     // definir color del contorno
translate()     // desplazar
rotateX()       // rotar sobre el eje x
rotateY()       // rotar sobre el eje y
rotateZ()       // rotar sobre el eje z
rotate()        // rotar definiendo rotación en x, y y z
scale()         // escalar el tamaño
resetMatrix()   // deshacer transformaciones hechas con rotate, scale o translate
```

BeginShape y beginContour

```
PShape s;
void setup() {
 size(200, 200, P2D);
 s = createShape();
 s.beginShape();
 s.stroke(color(0,255,255));
 s.vertex(-50,50); // triángulo grande
 s.vertex(0,-50);
 s.vertex(50,50);
 s.beginContour(); // se define la figura que recorta
 s.vertex(-10,-10);
 s.vertex(0,10);
 s.vertex(10,-10);
 s.endContour();
 s.endShape(CLOSE);
 s.setFill(color(255,0,0));
}
```

```
void draw() {
 background(204);
 line(0,height/2,width,height/2); // lineas para marcar el (0,0) de la figura definida en PShape
 line(width/2,0,width/2,height);
 translate(width/2, height/2);
 s.rotate(0.01); // en cada draw rota la figura
 shape(s);
}
```



PShape getVertex y setVertex

```
PShape s;
void setup() {
  size(200, 200);
  s = createShape();
  s.beginShape();
  s.vertex(0, 0);
  s.vertex(60, 0);
  s.vertex(60, 60);
  s.vertex(0, 60);
  s.endShape(CLOSE);
}
```

```
void draw() {
  translate(70, 70);
  for (int i = 0; i < s.getVertexCount(); i++) {
    PVector v = s.getVertex(i);
    v.x += random(-2, 2);
    v.y += random(-2, 2);
    s.setVertex(i, v);
  }
  shape(s);
}
```

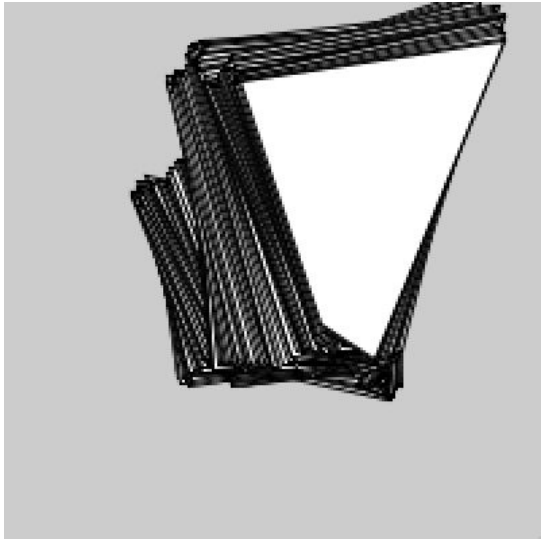
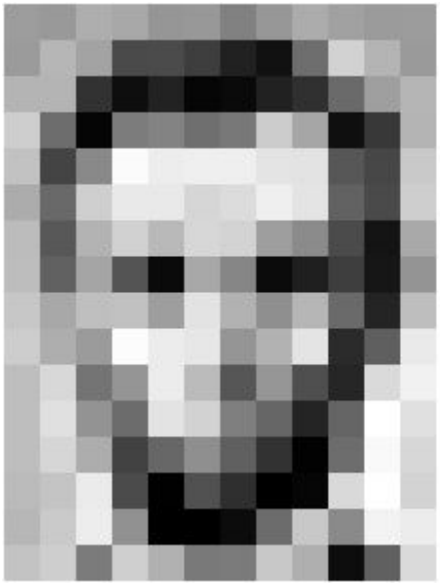


Imagen como array de píxeles

Array de pixeles



157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

How the pixels look:

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

How the pixels are stored:

0	1	2	3	4	5	6	7	8	9	·	·	·		

Para primeros ejercicios con las imágenes

1. Guardamos un sketch “fotoTest”
2. En la carpeta “fotoTest” generada en el espacio de trabajo de Processing creamos una carpeta “data”
3. Bajamos una imagen cualquiera de internet (formato .gif, .jpg, .tga, .png) y guardamos en la carpeta “data”
4. Escribimos siguiente código:

```
loadImage(filename)
loadImage(filename, extension)
```

Cargar una imagen de la computadora

```
PImage localImg;

void settings() {
 localImg = loadImage("pic.jpg"); // path local si está en el folder “data” adentro del folder del sketch
 // localImg = loadImage("/Users/ewe/dev/p3/sketch_170807a/data/pic.jpg"); path absoluto si no está en “data”
 size(localImg.width, localImg.height);
}

void draw() {
 background(0);
 image(localImg, 0, 0); // función para “mostrar” la imagen
}
```

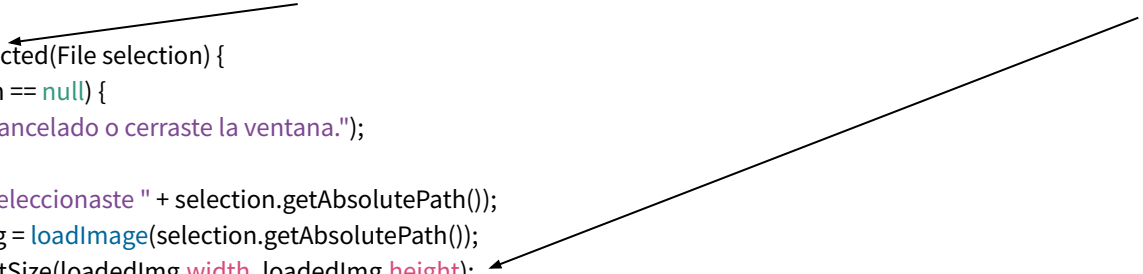
```
loadImage(filename)
loadImage(filename, extension)
```

Cargar una imagen de la web

```
PImage webImg;

void settings() {
  String url = "https://processing.org/img/processing-web.png";
  webImg = loadImage(url, "png");
  size(webImg.width, webImg.height);
}

void setup() {
  background(0);
  image(webImg, 0, 0);
}
```

Cargar una imagen en el momento

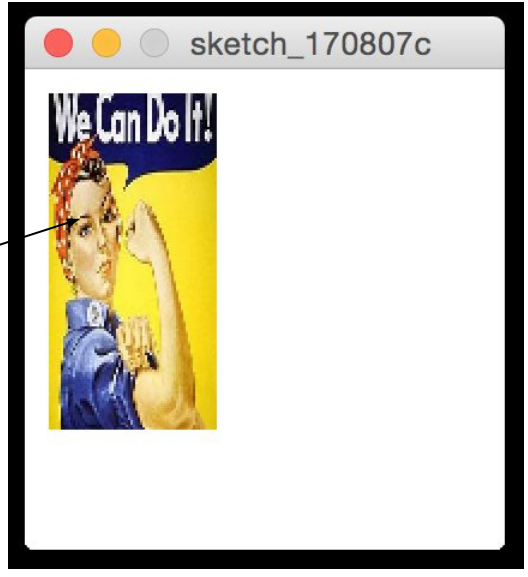
```
PImage loadedImg;
void settings() {
 loadedImg = loadImage("we.jpg");
 size(loadedImg.width, loadedImg.height);
}
void draw(){
 image(loadedImg,0,0);
}
void mouseClicked(){
 selectInput("Selecciona imagen:", "fileSelected");
}
void fileSelected(File selection) {
 if (selection == null) {
  println("Cancelado o cerraste la ventana.");
 } else {
  println("Seleccionaste " + selection.getAbsolutePath());
  loadedImg = loadImage(selection.getAbsolutePath());
  surface.setSize(loadedImg.width, loadedImg.height);
 }
}
```



Image

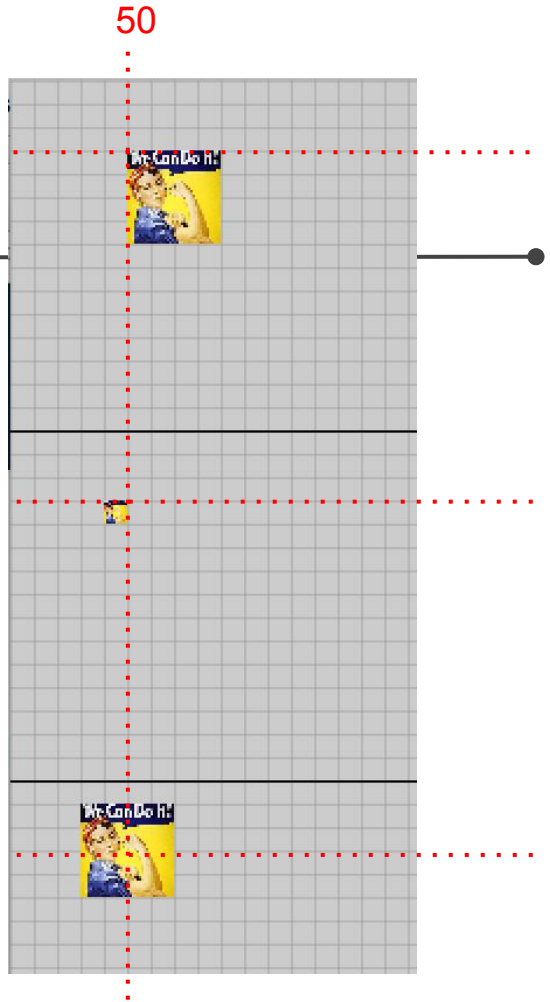
image(img, x, y) // (x,y) de la esquina superior izquierda, ancho y alto de la imagen
image(img, x, y, w, h) // forzamos el w y h

```
PImage img;

void setup() {
  size(200, 200);
  img = loadImage("we.jpg");
}

void draw() {
  background(255);
  image(img,10,10,70,140);
}
```



imageMode

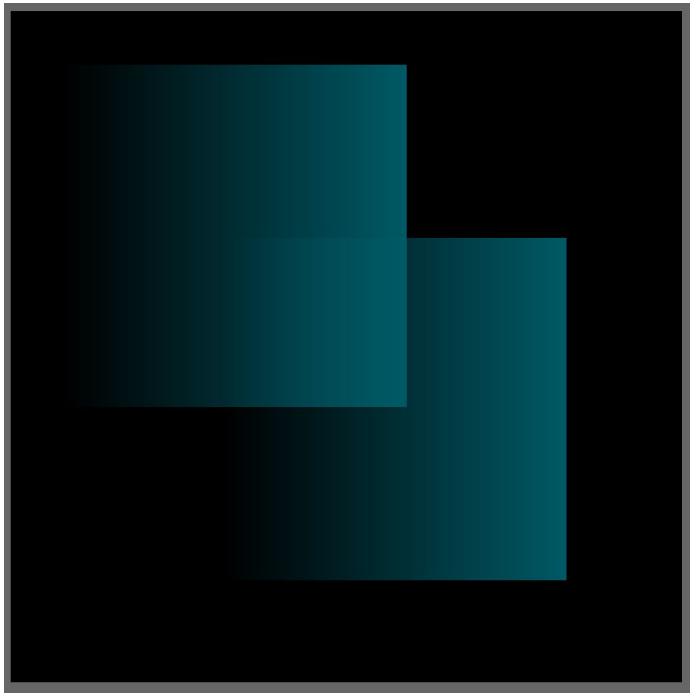
```
PImage img = loadImage("we.jpg");
size(400,800);
imageMode(CORNER);
image(img, 50, 30, 40, 40); // image(img,x,y,w,h), (x,y) de la esquina superior izquierda

translate(0,150);
line(0,0,width,0);
imageMode(CORNERS);
image(img, 50, 30, 40, 40);  // image(img,x,y,x2,y2)

translate(0,150);
line(0,0,width,0);
imageMode(CENTER);
image(img, 50, 30, 40, 40); // image(img,x,y, w, h), (x,y) del centro
```



Crear una imagen // array pixels[]

```
PImage img ;
void setup(){
  size(500,500);
  img = createImage(255, 255, ARGB);
  for (int i = 0; i < img.pixels.length; i++) {
    img.pixels[i] = color(0, 90, 102, i % img.width);
  }
}

void draw(){
  background(0);
  image(img, 40, 40);
  image(img, mouseX-img.width/2, mouseY-img.height/2);
}
```

Seteo el color pixel por pixel



```
pimg.set(x, y, c)
pimg.set(x, y, img)
```

Crear una imagen // img.set(x,y,c)

```
PImage img ;
void setup(){
 size(500,500);
 img = createImage(255, 255, ARGB);
 for (int i = 0; i < img.width; i++) {
  for (int j = 0; j < img.height; j++) {
   img.set(i,j,color(0, 90, 102, i));
  }
 }
}

void draw(){
 background(0);
 image(img, 40, 40);
 image(img, mouseX-img.width/2, mouseY-img.height/2);
}
```

Seteo el color pixel por pixel

```
pimg.set(x, y, c)
pimg.set(x, y, img)
```

Combinación de imágenes con set()

```
PImage img, img2 ;

size(350,350);
background(0);
img = createImage(255, 255, ARGB);
img2 = loadImage("we.jpg");
for (int i = 0; i < img.width; i++) {
  for (int j = 0; j < img.height; j++) {
    img.set(i,j,color(0, 90, 102, i));
  }
}
img.set(140,40,img2);
image(img, 40, 40);
```

Seteo el color pixel por pixel

Seteo los pixeles de una imagen

```
vertex(x, y, u, v)
```

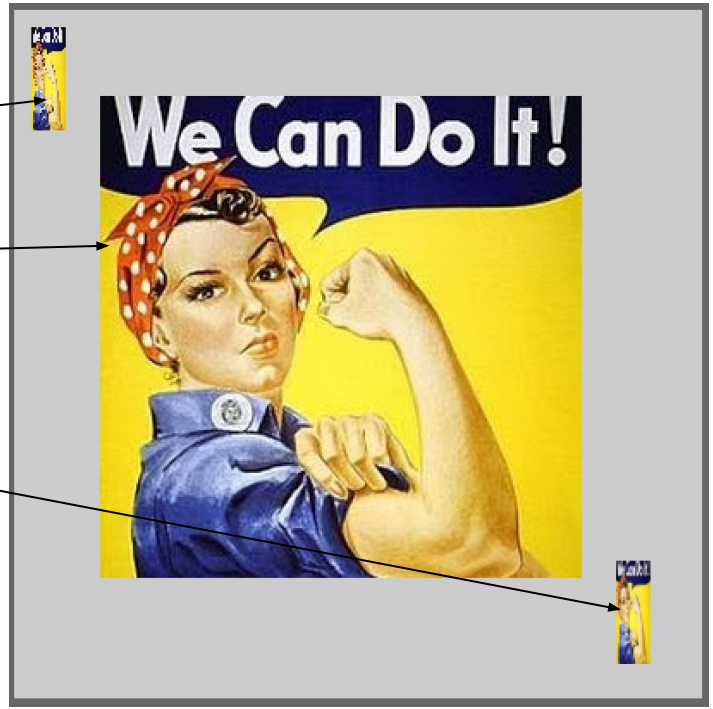
Imagen como textura

```
PImage img;

void setup() {
  size(400, 400, P3D);
  img = loadImage("we.jpg");
}

void draw() {
  background(255);
  beginShape();
  texture(img);
  vertex(20, 40, 0, 0);
  vertex(width, 0, img.width, 0);
  vertex(width-40, height-20, img.width, img.height);
  vertex(0, height-40, 0, img.height);
  endShape();
}
```

```
pimg.resize(w, h)
```

Resize

```
size(400,400);
PImage img =loadImage("we.jpg");
image(img,10,10,20,60);

image(img, 50, 50);

img.resize(20,60);
image(img, 350, 320);
```



! cambia las dimensiones de la imagen !

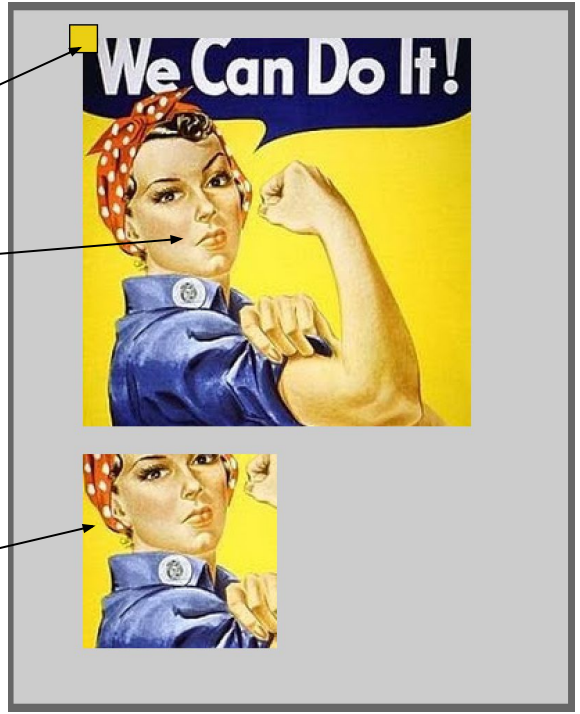
! si uno de los parámetros == 0, se escala en proporción con el otro !

```
pimg.get(x, y)
pimg.get(x, y, w, h)
pimg.get()
```

Get

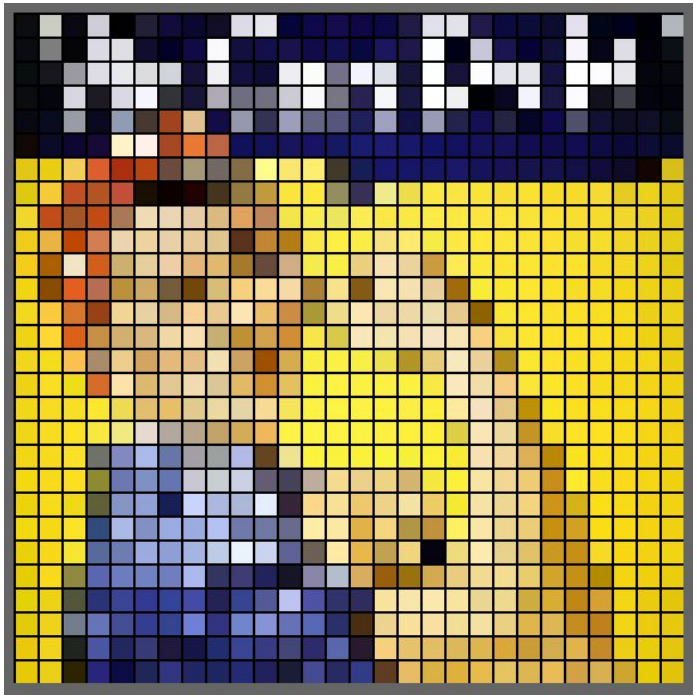
Para consultar colores de los pixeles, conseguir un “recorte” o copia de una imagen

```
size(400,500);
PImage img =loadImage("we.jpg");
image(img, 50, 20);

color c = img.get(img.width-1,img.height/2);
fill(c);
rect(40,10,20,20);

PImage part = img.get(10,100,img.width/2,img.height/2);
image(part, 50, 320);
```



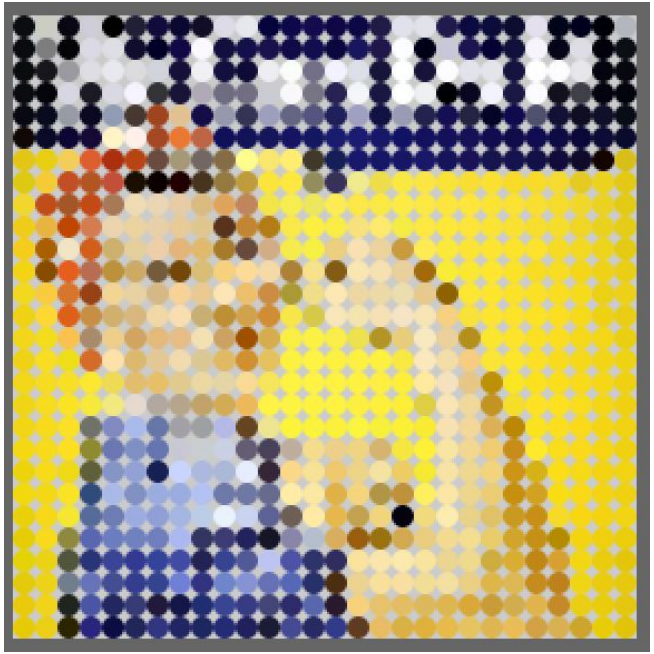
Get y imágenes pixeladas

```
PImage img ;
void settings() {
 img = loadImage("we.jpg");
 size(img.width, img.height);
}
void setup(){
 for (int i = 0; i < img.width; i+=10) {
  for(int j = 0; j < img.height; j+=10){
   fill(img.get(i,j));
   rect(i,j,10,10);
  }
 }
}
```



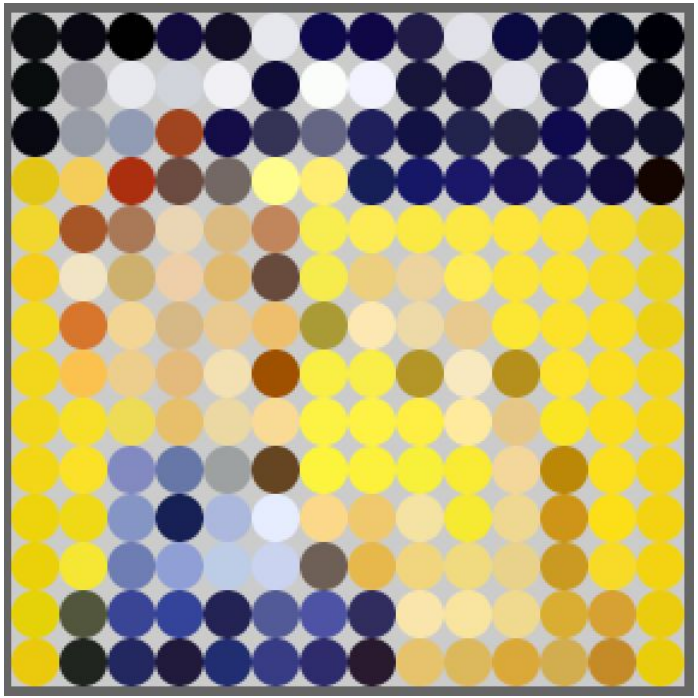
Get y pixeles circulares

```
PImage img ;
void settings() {
 img = loadImage("we.jpg");
 size(img.width, img.height);
}
void setup(){
 noStroke();
 ellipseMode(CORNER);
 for (int i = 0; i < img.width; i+=10) {
  for(int j = 0; j < img.height; j+=10){
   fill(img.get(i,j));
   ellipse(i,j,10,10);
  }
 }
}
```



Más abstracto?

```
PImage img ;
int figureWH = 20;
void settings() {
 img = loadImage("we.jpg");
 size(img.width, img.height);
}
void setup(){
 noStroke();
 ellipseMode(CORNER);
 for (int i = 0; i < img.width; i+=figureWH) {
  for(int j = 0; j < img.height; j+=figureWH){
   fill(img.get(i,j));
   ellipse(i,j,figureWH,figureWH);
  }
 }
}
```



Modificar píxel por píxel

Bajamos de internet dos imágenes cualquiera de la misma resolución y cambiamos cada segundo pixel de la imagen 1 a la imagen 2.

Intercambio de píxeles

```
size(400,500);
PImage img =loadImage("we.jpg");
PImage img2 =loadImage("hamster.jpg");

for (int i = 0; i < img.width; i+=2) {
  for(int j = 0; j < img.height; j++){
    img.set(i,j, img2.get(i,j));
  }
}

image(img, 0, 0);
```

Cambio de píxeles

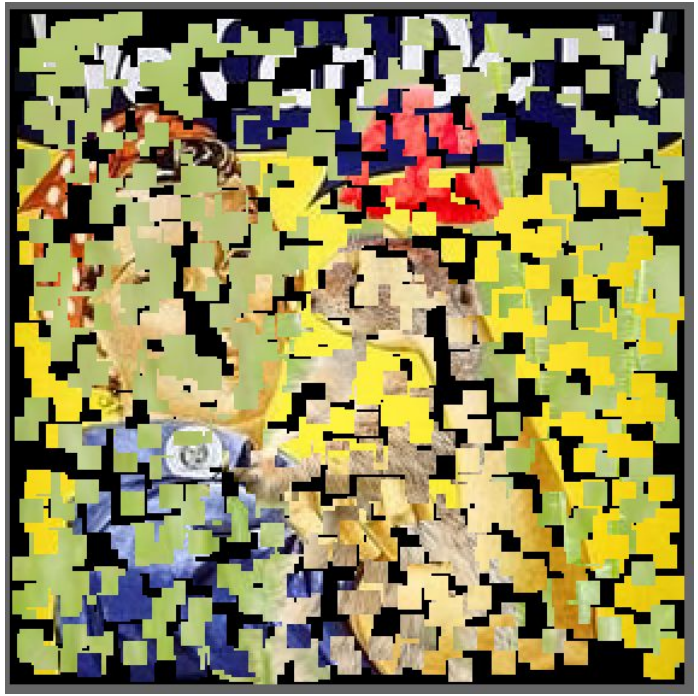
```
int col = 0;
PImage img,img2;
void setup(){
 size(280,280);
 img =loadImage("we.jpg");
 img2 =loadImage("hamster.jpg");
 frameRate(30);
}
void draw(){
 for(int j = 0; j < img.height; j++){
  img.set(col,j, color(img2.get(col,j),10));
 }
 col+=1;
 image(img, 0, 0);
}
```

transparencia

```
pimg.copy()
pimg.copy(sx, sy, sw, sh, dx, dy, dw, dh)
pimg.copy(src, sx, sy, sw, sh, dx, dy, dw, dh)
```

Copy = get, set y resize en uno

```
PImage img,img2,img3;
void setup(){
 size(280,280);
 img =loadImage("we.jpg");
 img2 =loadImage("hamster.jpg");
 img3 = createImage(img.width,img.height,RGB);
}
void draw(){
 int randomX = ceil(random(img.width - 10));
 int randomY = ceil(random(img.height - 10));
 if(frameCount%2 == 0)
  img3.copy(img, randomX, randomY, 10, 10, randomX, randomY,10,10);
 else
  img3.copy(img2, randomX, randomY, 10, 10, randomX, randomY,10,10);
 image(img3,0,0);
}
```



```
tint(rgb)
tint(rgb, alpha)
tint(gray)
tint(gray, alpha)
tint(v1, v2, v3)
tint(v1, v2, v3, alpha)
```

Tint

```
colorMode(HSB);
size(500,500);
PImage img =loadImage("we.jpg");
background(255);
image(img,0,0);
for(int i = 10; i<width; i+=50){
  translate(50,50);
  tint(random(255),255,255);
  image(img,0,0);
}
```

```
tint(rgb)
tint(rgb, alpha)
tint(gray)
tint(gray, alpha)
tint(v1, v2, v3)
tint(v1, v2, v3, alpha)
```

Tint con transparencia

tint(rgb)

tint(rgb, alpha)

tint(gray)

tint(gray, alpha)

tint(v1, v2, v3)

tint(v1, v2, v3, alpha)

Máscaras


```
pimg.mask(maskArray)
pimg.mask(img)
```

Usa una imagen como fuente de valores de la opacidad.
Imagen y máscara tienen que ser del mismo tamaño!

```
PImage photo, roseImage, maskImage;
void setup() {
  size(280, 280);
  photo = loadImage("we.jpg");
  roseImage = loadImage("rose.png");
  maskImage = new PImage(photo.width,photo.height);
  maskImage.copy(roseImage, 0, 0, roseImage.width, roseImage.height, 0, 0,280,280); // hacemos coincidir las dimensiones de la imagen y de la máscara
  photo.mask(maskImage);
  background(255, 0, 0); // fondo rojo
  image(photo, 0, 0);
}
```

Filtros PImage

8 filtros por defecto:

THRESHOLD, GRAY, OPAQUE, INVERT, POSTERIZE, BLUR, ERODE, DILATE







Filtros PImage

THRESHOLD // pasa la imagen a blanco y negro

GRAY // escala de grises

OPAQUE // sustituye la transparencia por blanco

INVERT // invierte los colores

POSTERIZE // limita cada canal de la imagen a un número de colores especificado como el parámetro

BLUR // desenfoque gaussiano, el parámetro especifica el radio de la difuminación

ERODE // reduce áreas claras

DILATE // expande áreas claras

```
pimg.filter(kind)
pimg.filter(kind, param)
```

Filtros en acción

```
PImage img;
void setup() {
  size(280,280);
  img = loadImage("we.jpg");
  image(img, 0, 0);
}
void draw() {
}
void keyPressed() {
  switch(key){
    case '0':
      img.filter(THRESHOLD); // imagen en blanco y negro
      break;
    case '1':
      img.filter(GRAY); //escala de grises
      break;
    case '2':
      img.filter(INVERT); //invierte los colores
      break;
    case '3':
      img.filter(POSTERIZE, 4); // limita la cantidad de los colores
      break;
    case '4':
      img.filter(BLUR, 6); // desenfoque Gaussiano
      break;
    case '5':
      img.filter(ERODE); // reduce las zonas claras
      break;
    case '6':
      img.filter(DILATE); // expande las zonas claras
      break;
    default:
      img = loadImage("we.jpg");
      break;
  }
  image(img, 0, 0);
}
```

```
pimg.filter(kind)
pimg.filter(kind, param)
```

Filtro blur aplicado a una imagen

```
PImage img;

void setup() {
  size(280,280);
  img = loadImage("we.jpg");
  frameRate(5);
}

void draw() {
  img.filter(BLUR,2);
  image(img, 0, 0);
}
```

Se aplica a la imagen en cada draw

THRESHOLD, GRAY, OPAQUE, INVERT, POSTERIZE, BLUR, ERODE, DILATE

```
filter(shader)
filter(kind)
filter(kind, param)
```

Filtro blur de todo

```
PImage img;

void setup() {
  size(280,280);
  img = loadImage("we.jpg");
  image(img, 0, 0);
  frameRate(5);
}

void draw() {
  filter(BLUR, 2);
}
```

Se aplica a todo ya dibujado

THRESHOLD, GRAY, OPAQUE, INVERT, POSTERIZE, BLUR, ERODE, DILATE

```
filter(shader)
filter(kind)
filter(kind, param)
```

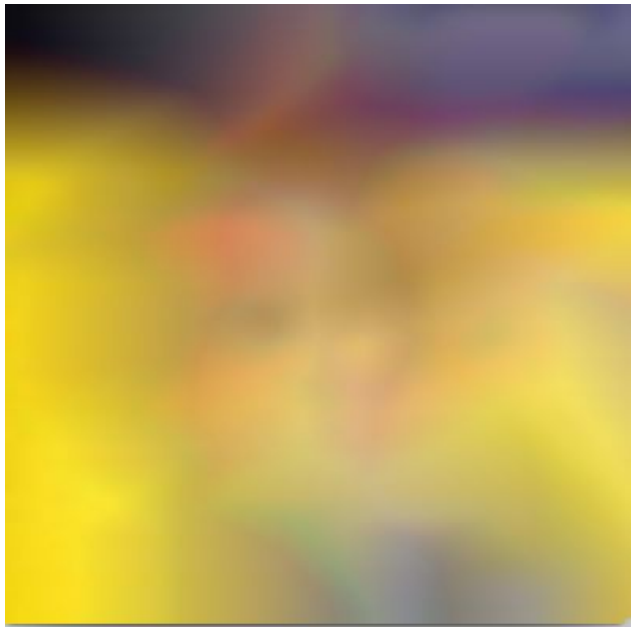
Filtros en acción

```
PImage img;
void setup() {
 size(280,280);
 img = loadImage("we.jpg");
 image(img, 0, 0);
}
void draw() {
}
void keyPressed() {
 switch(key){
  case '0':
   filter(THRESHOLD); // imagen en blanco y negro
   break;
  case '1':
   filter(GRAY); //escala de grises
   break;
  case '2':
   filter(INVERT); //invierte los colores
   break;
  case '3':
   filter(POSTERIZE, 4); // limita la cantidad de los colores
   break;
  case '4':
   filter(BLUR, 6); // desenfoque Gaussiano
   break;
  case '5':
   filter(ERODE);  // reduce las zonas claras
   break;
  case '6':
   filter(DILATE); // expande las zonas claras
   break;
  default:
   image(img, 0, 0);
   break;
 }
}
```

```
filter(shader)
filter(kind)
filter(kind, param)
```

Filtro con shader

```
PShader blur;
PImage img;

void setup() {
  size(200, 200, P2D);
  blur = loadShader("blur.glsl");
  img = loadImage("we.jpg");
  image(img, 0, 0);
}

void draw() {
  filter(blur);
}
```



```
blend(sx, sy, sw, sh, dx, dy, dw, dh, mode)
blend(src, sx, sy, sw, sh, dx, dy, dw, dh, mode)
```

Blend

https://processing.org/reference/PImage_blend_.html

BLEND
ADD
SUBTRACT
DARKEST
LIGHTEST
DIFFERENCE
EXCLUSION
MULTIPLY
SCREEN
OVERLAY
HARD_LIGHT
SOFT_LIGHT
DODGE
BURN

```
blend(sx, sy, sw, sh, dx, dy, dw, dh, mode)
blend(src, sx, sy, sw, sh, dx, dy, dw, dh, mode)
```

Todos los blend

```
PImage img,img2;
int mode = ADD;
void setup() {
 size(280,280);
 img = loadImage("we.jpg");
 img2 = loadImage("hamster.jpg");
 image(img,0,0);
}
void draw() {
}
void keyPressed() {
 image(img, 0, 0);
 switch(key){
  case 'q':
   mode = ADD;
   break;
  case 'w':
   mode = SUBTRACT;
   break;
case 'e':
 mode = DARKEST;
 break;
case 'r':
 mode = LIGHTEST;
 break;
case 't':
 mode = BURN;
 break;
case 'y':
 mode = BLEND;
 break;
case 'u':
 mode = DIFFERENCE;
 break;
case 'i':
 mode = EXCLUSION;
 break;
case 'o':
 mode = MULTIPLY;
 break;
 case 'p':
  mode = SCREEN;
  break;
 case 'a':
  mode = OVERLAY;
  break;
 case 's':
  mode = HARD_LIGHT;
  break;
 case 'd':
  mode = SOFT_LIGHT;
  break;
 case 'f':
  mode = DODGE;
  break;
 default:
  image(img, 0, 0);
  break;
 }
 blend(img2, 140, 0, 140,280, 140, 0, 140,280, mode);
}
```

```
blendMode(mode)
```

blendMode

Recomendado sobre blend pero no tan completo. No tiene : OVERLAY, HARD_LIGHT, SOFT_LIGHT, DODGE, BURN, pero tiene REPLACE.

BLEND
ADD
SUBTRACT
DARKEST
LIGHTEST
DIFFERENCE
EXCLUSION
MULTIPLY
SCREEN
REPLACE

```
blendMode(mode)
```

Las variantes de blend

```
int mode = BLEND;
PImage img,img2;
void setup() {
 size(280,280);
 img = loadImage("we.jpg");
 img2 = loadImage("hamster.jpg");
 image(img,0,0);
}
void draw() {
}
void keyPressed() {
 //background(255);
 switch(key){
  case 'q':
   mode = ADD;
   break;
  case 'w':
   mode = SUBTRACT;
   break;
```

```
  case 'e':
   mode = DARKEST;
   break;
  case 'r':
   mode = LIGHTEST;
   break;
  case 't':
   mode = DIFFERENCE;
   break;
  case 'y':
   mode = EXCLUSION;
   break;
  case 'u':
   mode = MULTIPLY;
   break;
  case 'i':
   mode =  SCREEN;
   break;
```

```
  case 'o':
   mode =  REPLACE;
   break;
  default:
   mode = BLEND;
   break;
 }
 blendMode(mode);
 image(img,0,0);
 image(img2,0,0);
}
```

Secuencia de imágenes

Trucos para hacer secuencias animadas:

https://www.processing.org/examples/sequential.html

https://www.processing.org/examples/animatedsprite.html

saveFrame

saveFrame()

saveFrame(filename)

TIFF (.tif), TARGA (.tga), JPEG (.jpg), o PNG (.png)

saveFrame(); // guarda como screen-0001.tif, screen-0002.tif, etc.

saveFrame("line-######.png"); // guarda como line-000001.png, line-000002.png, etc.

saveFrame("out/line-######.png"); // guarda como line-000001.png, etc. en la carpeta “out” que se crea automáticamente al ejecutar el sketch

saveFrame y exportar como animación

saveFrame("outFolder/line-######.png");

+

Tool -> Movie Maker