PROCESSING
COOKBOOK
by Angelos Floros

OBJECTS
TEXT, IMAGE, VIDEO

creative
scripting
m e a l s
04

exercises

sketches

- *ALPHABET*
- *LETTERS*
- *TEXT ALIGN`*
- *CHARACTER, STRINGS*
- *TEXT GUI*
- *LOAD DISPLAY IMAGE*
- *CREATE IMAGE*
- *ALPHA, RGB COLORS*
- *COLOR MANIPULATION*
- *LINEAR GRADIENT*
- *RADIAL GRADIENT*
- *WAVE GRADIENT*
- *BLUR*
- *BRIGHTNESS*
- *EDGE DETECTION*
- *HISTOGRAM*
- *IMAGE DISTORTION*
- *VIDEO PLAY, PAUSE, LOOP*
- *VIDEO FORWARD, REWIND*

Objects

TYPOGRAPHY

Loading & Displaying
PFont
createFont()
loadFont()
text()
textFont()

Attributes
textAlign()
textLeading()
textMode()
textSize()
textWidth()

Metrics
textAscent()
textDescent()

IMAGE

Loading & Displaying
image()
imageMode()
loadImage()
noTint()
requestImage()
tint()

Textures
texture()
textureMode()
textureWrap()

Pixels
blend()
copy()
filter()
get()
loadPixels()
pixels[]

Rendering
blendMode()
createGraphics()
PGraphics
loadShader()
PShader
resetShader()
shader()

Alphabet createFont(), loadFont(), text(), textFont()

http://processing.org/reference/createFont_.html
http://processing.org/reference/loadFont_.html
http://processing.org/reference/text_.html
http://processing.org/reference/textFont_.html

SYNTAX

createFont(name, size)
createFont(name, size, smooth)
createFont(name, size, smooth, charset)
loadFont(filename)
textFont(which, size)
text(c, x, y)
text(c, x, y, z)
text(str, x, y)
text(chars, start, stop, x, y)
text(str, x, y, z)
text(chars, start, stop, x, y, z)
text(str, x1, y1, x2, y2)
text(num, x, y)
text(num, x, y, z)

DESCRIPTION

PFont is the font class for Processing.

createFont()
Dynamically converts a font to the format used by Processing from a .ttf or .otf file inside the sketch's «data» folder or a font that's installed elsewhere on the computer.

loadFont()
Loads a .vlw formatted font into a PFont object.

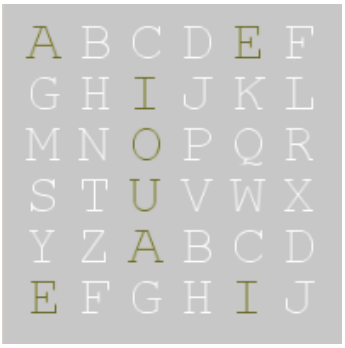
text()
Draws text to the screen. Displays the information specified in the first parameter on the screen in the position specified by the additional parameters.

textFont()
Sets the current font that will be drawn with the text()

PARAMETRES

name	String: name of the font to load
size	float: point size of the font
filename	String: name of the font to load
which	PFont: any variable of the type PFont
c	char: the alphanumeric character to be displayed
x	float: x-coordinate of text
y	float: y-coordinate of text
z	float: z-coordinate of text
chars	char[]: the alphanumberic symbols to be displayed
start	int: array index at which to start writing characters
stop	int: array index at which to stop writing characters
x1	float: by default, the x-coordinate of text, see rectMode() for more info
y1	float: by default, the x-coordinate of text, see rectMode() for more info
x2	float: by default, the width of the text box, see rectMode() for more info
y2	float: by default, the height of the text box, see rectMode() for more info
num	int, or float: the numeric value to be displayed

CODE




```
PFont fontA; // Δηλώνει τη χρήση κειμένου

void setup()
{
  size(200, 200);
  background(200);
  smooth();

     // Πρέπει να δημιουργηθεί η γραμματοσειρά 'Courier-36' (Tools>CreateFont)
  // και να αποθηκευτεί στον φάκελο data μέσα στον φάκελο εργασίας
  fontA = loadFont(«CourierNew-36.vlw»); // Χρήση γραμματοσειράς
  textAlign(CENTER);
  textFont(fontA, 32);   // Ορίζει τη γραμματοσειρά και το μέγεθος σε pixels
  noLoop();   // Τη σχεδιάζει μόνο μία φορά
}

void draw()
{
  fill(255);   // Χρωματίζει με λευκό χρώμα
  int margin = 6;  // Ορίζει το αριστερό όριο
  int gap = 30;  // Ορίζει το πάνω όριο
  translate(margin*1.5, margin*2);

  // Δημιουργεί έναν πίνακα με χαρακτήρες
  int counter = 0;

  for(int i=0; i<margin; i++) {
    for(int j=0; j<margin; j++) {
      char letter;
      int count = 65+(i*margin)+j;

 if(count <= 190) {
       letter = char(65+counter);
       if(letter == 'A' || letter == 'E' || letter == 'I' ||
         letter == 'O' || letter == 'U') {
         fill(104, 104, 0);
       }
       else {
         fill(255);
       }
      }
      else {
       fill(153);
       letter = char(48+counter);
      }
      // Σχεδιάζει το γράμμα
      text(letter, 15+j*gap, 20+i*gap);



// Λειτουργία μετρητή
      counter++;
      if(counter >= 26) {
        counter = 0;
      }
    }
  }
}
```

Letters

CODE


```
PFont f;
import java.util.Calendar;

void setup() {
  size(640, 360);
  background(0);

   f = createFont(«Monospaced», 24);  // Create the font
  textFont(f);
  textAlign(CENTER, CENTER);
}

void draw() {
  background(0);

  // Set the left and top margin
  int margin = 10;
  translate(margin*4, margin*4);

  int gap = 46;
  int counter = 35;

  for (int y = 0; y < height-gap; y += gap) {
    for (int x = 0; x < width-gap; x += gap) {

      char letter = char(counter);

       if (letter == ‘A’ || letter == ‘E’ || letter == ‘I’ || letter == ‘O’ ||
letter == ‘U’) {
        fill(255, 204, 0);
      } else {
        fill(255);
      }

      // Draw the letter to the screen
      text(letter, x, y);

      // Increment the counter
      counter++;
    }
  }
}

void keyPressed() {
  if (key==’s’ || key==’S’) {
    saveFrame(timestamp()+».png»);
  }
}

String timestamp() {
   return  String.format(«%1$ty%1$tm%1$td_%1$tH%1$tM%1$tS»,
Calendar.getInstance());
}
```

```
/**
 * Letters.
 *
  * Draws letters to the screen. This
requires loading a font,
 * setting the font, and then drawing
the letters.
 */

// The next line is needed if running in
JavaScript Mode with Processing.js
/* @pjs font=»Courier.ttf»; */
```

Text Align textAlign(), textSize(), textWidth()

http://processing.org/reference/textAlign_.html
http://processing.org/reference/textSize_.html
http://processing.org/reference/textWidth_.html

SYNTAX

```
textAlign(alignX)
textAlign(alignX, alignY)
textSize(size)
TextWidth(c)
textWidth(str)
```

DESCRIPTION

textAlign()
Sets the current alignment for drawing text. The parameters LEFT, CENTER, and RIGHT set the display characteristics of the letters in relation to the values for the x and y parameters of the text() function.

textSize()
Sets the current font size. This size will be used in all subsequent calls to the text() function. Font size is measured in units of pixels.

textWidth()
Calculates and returns the width of any character or text string.

PARAMETRES

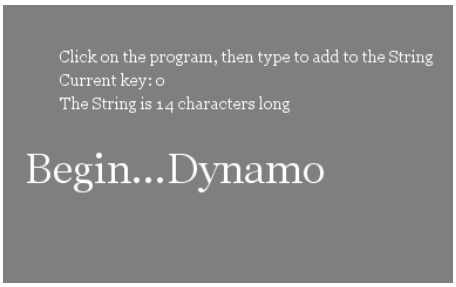
alignX	int: horizontal alignment, either LEFT, CENTER, or RIGHT
alignY	int: vertical alignment, either TOP, BOTTOM, CENTER, or BASELINE
c	char: the character to measure
str	String: the String of characters to measure

CODE


```
/**
 * Words.
 *
 * The text() function is used for writing words to the screen.
 * The letters can be aligned left, center, or right with the
 * textAlign() function.
 */

// The next line is needed if running in JavaScript Mode with Processing.js
/* @pjs font="Georgia.ttf"; */

PFont f;

void setup() {
  size(640, 360);
  println(PFont.list()); // prints the FontList
  f = createFont("Serif", 24); // Create the font
  textFont(f);
}

void draw() {
  background(102);
  textAlign(RIGHT);
  drawType(width * 0.25);
  textAlign(CENTER);
  drawType(width * 0.5);
  textAlign(LEFT);
  drawType(width * 0.75);
}

void drawType(float x) {
  line(x, 0, x, 65);
  line(x, 220, x, height);
  fill(0);
  text("ichi", x, 95);
  fill(51);
  text("ni", x, 130);
  fill(204);
  text("san", x, 165);
  fill(255);
  text("shi", x, 210);
}
```

Character Strings

CODE




```
/**
 * Characters Strings.
 *
 * The character datatype, abbreviated as char, stores letters and
 * symbols in the Unicode format, a coding system developed to support
 * a variety of world languages. Characters are distinguished from other
 * symbols by putting them between single quotes ('P').<br />
 * <br />
 * A string is a sequence of characters. A string is noted by surrounding
 * a group of letters with double quotes («Processing»).
 * Chars and strings are most often used with the keyboard methods,
 * to display text to the screen, and to load images or files.<br />
 * <br />
 * The String datatype must be capitalized because it is a complex datatype.
 * A String is actually a class with its own methods, some of which are
 * featured below.
 */

// The next line is needed if running in JavaScript Mode with Processing.js
/* @pjs font=»Georgia.ttf»; */

char letter;
String words = «Begin...»;

void setup() {
  size(400, 250);
  // Create the font
  textFont(createFont(«Georgia», 36));
}

void draw() {
  background(126); // Set background to black

  // Draw the letter to the center of the screen
  textSize(14);
  text(«Click on the program, then type to add to the String», 50, 50);
  text(«Current key: « + letter, 50, 70);
  text(«The String is « + words.length() +  « characters long», 50, 90);

  textSize(36);
  text(words, 20, 120, 360, 180);
}

void keyPressed() {
  // The variable «key» always contains the value
  // of the most recent key pressed.
  char k;
  k = (char)key;

  if ((key >= 'A' && key <= 'z') || key == ' ') {
    letter = key;
    words = words + key;
    // Write the letter to the console
    println(key);
  } else if (k==8) {
    words = words.substring(0,words.length()-1) ;
    println(«del»);
  }
}
```

Flashlight

CODE

/* OpenProcessing Tweak of *@*http://www.openprocessing.org/sketch/477*@* */
/* !do not delete the line above, required for linking your tweak if you re-upload */
/**
 * Text revealing flashlight
 * Click to reverse the effect
 * Use the UP and DOWN arrows to change the light size
 *
 * Elie Zananiri
 * ACAD Processing workshop
 * April 2008
 */

```
/*------- global variables ----------*/
int TRACKING = 2;

PFont f;
byte[] textFile;  // contents of external file split by glyphs
int lineHeight;

float flashlightSize = 50;
boolean whiteOnBlack = true;  // light mode toggle

/*------- initialization ------------*/
void setup() {
  size(400, 580);
  smooth();
  colorMode(RGB, 1.0);
  noCursor();

  // load and set the font
  f = loadFont("Mandingo-24.vlw");
  textFont(f, 12);
  lineHeight = int(textAscent()+textDescent());

  // load the contents of the external file into the byte array
  textFile = loadBytes("hamsters.txt");
}

/*------ function definitions -------*/
// writes a grid of characters with the ones closest to the mouse being brighter
void drawText() {
  // draw all characters in the grid one at a time
  int currX = TRACKING;
  int currY = lineHeight;
  for (int i=0; i < textFile.length; i++) {
    // get the next character
    char c = char(textFile[i]);
    if (Character.isWhitespace(c)) {
      // if it's any type of whitespace (space, tab, line feed),
      // convert it to a single space
      c = ' ';
    }

    // calculate the distance between the current position and the mouse
    float distance = dist(currX, currY, mouseX, mouseY);
    // set a fill color relative to that distance
    float col = map(distance, 0, flashlightSize, 0, 1);
    if (whiteOnBlack) {
      col = 1-col;
    }

    // draw the character
    fill(col);
    text(c, currX, currY);

    // set the next position
    currX += int(textWidth(c)+TRACKING);
```

CODE

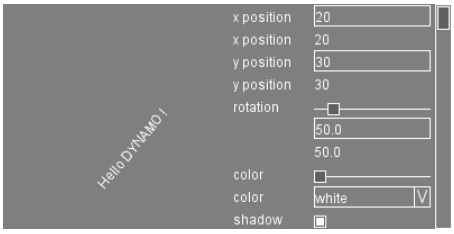
```
    if (currX > (width-TRACKING)) {
      // go to the next line
      currX = TRACKING;
      currY += lineHeight+TRACKING;
    }

    // if we've reached the end of the window
    if (currY > height) {
      // skip the remaining characters
      break;
    }
  }
}

/*------ event handlers -------------*/
void keyReleased() {
  if (key == CODED) {
    if (keyCode == DOWN) {
      flashlightSize -= 5;
    } else if (keyCode == UP) {
      flashlightSize += 5;
    }
  }
}

void mouseReleased() {
  whiteOnBlack = !whiteOnBlack;
}

/*------ main loop ------------------*/
void draw() {
  // erase the background
  if (whiteOnBlack) {
    background(0);
  } else {
    background(1);
  }

  drawText();
}
```

Text GUI

DESCRIPTION

Create a graphical user interface to manipulate text.

CODE




```
MinyGUI gui;

MinyInteger x, y;
MinyFloat rot;
MinyInteger colorChoice;
MinyBoolean shadow;
MinyString caption;

void mousePressed() { gui.onMousePressed();  }
void keyPressed()   { gui.onKeyPressed();    }

void setup()
{
  size(400, 200);
  x = new MinyInteger(20);
  y = new MinyInteger(30);
  rot = new MinyFloat(50.0);
  colorChoice = new MinyInteger(0);
  shadow = new MinyBoolean(true);
  caption = new MinyString("Hello DYNAMO !");

  gui = new MinyGUI(width/2, 0, width/2, height);
  gui.addEditBox("x position", x);
  gui.addDisplay("x position", x);
  gui.addEditBox("y position", y);
  gui.addDisplay("y position", y);
  gui.addSlider("rotation", rot, 0, 360);
  gui.addEditBox("", rot);
  gui.addDisplay("", rot);
  gui.addSlider("color", colorChoice, 0, 3);
  gui.addList("color", colorChoice, "white;red;green;blue");
  gui.addCheckBox("shadow", shadow);
  gui.addEditBox("text", caption);
  gui.addDisplay("", caption);
  gui.addButton("randomize", new ButtonRandomize());

  gui.fg = color(255);
  gui.bg = color(96);
  gui.selectColor = color(128);
}

class ButtonRandomize implements ButtonCallback
{
  void onButtonPressed()
  {
    rot.setValue(random(0.0, 360.0));
```

CODE

```
    x.setValue((int)random(-50, 50));
    y.setValue((int)random(-50, 50));
  }
}

void draw()
{
  background(126);

  fill(152); noStroke();

  gui.display();

  textAlign(CENTER);
  translate(width/4, height/2);
  translate(x.getValue(), y.getValue());
  rotate(radians(-rot.getValue()));
  if(shadow.getValue())
  { fill(128); text(caption.getValue(), 1, 1); }

  switch(colorChoice.getValue())
  {
    case 0: fill(255); break;
    case 1: fill(255, 0, 0); break;
```

Load Display Image PImage, loadImage(), imageMode()

http://processing.org/reference/PImage.html
http://processing.org/reference/loadImage_.html
http://processing.org/reference/imageMode_.html

SYNTAX

```
PImage img;
loadImage(filename)
loadImage(filename, extension)
imageMode(mode)
```

DESCRIPTION

PImage
Datatype for storing images. Processing can display .gif, .jpg, .tga, and .png images. Images may be displayed in 2D and 3D space. Before an image is used, it must be loaded with the loadImage() function. The PImage class contains fields for the width and height of the image, as well as an array called pixels[] that contains the values for every pixel in the image. The methods described below allow easy access to the image's pixels and alpha channel and simplify the process of compositing. Before using the pixels[] array, be sure to use the loadPixels() method on the image to make sure that the pixel data is properly loaded.

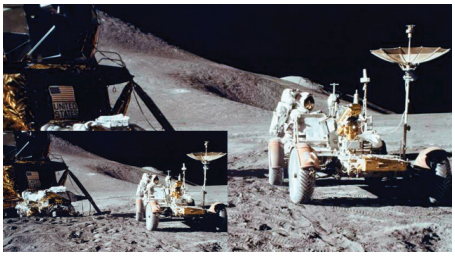
loadImage()
Loads an image into a variable of type PImage. Four types of images (.gif, .jpg, .tga, .png) images may be loaded. To load correctly, images must be located in the data directory of the current sketch.

imageMode()
Modifies the location from which images are drawn by changing the way in which parameters given to image() are intepreted.

PARAMETRES

img	Image: assumes a MediaTracker has been used to fully download the data and the img is valid
filename	String: name of file to load, can be .gif, .jpg, .tga, or a handful of other image types depending on your platform
extension	String: type of image to load, for example «png», «gif», «jpg»
mode	int: either CORNER, CORNERS, or CENTER

CODE



```
/**
 * Load and Display
 *
 * Images can be loaded and displayed to the screen at their actual size
 * or any other size.
 */
```

```
// The next line is needed if running in JavaScript Mode with Processing.js
/* @pjs preload="moonwalk.jpg"; */

PImage img;  // Declare variable "a" of type PImage

void setup() {
  size(640, 360);
  // The image file must be in the data folder of the current sketch
  // to load successfully
  img = loadImage("moonwalk.jpg");  // Load the image into the program
}

void draw() {
  // Displays the image at its actual size at point (0,0)
  image(img, 0, 0);
  // Displays the image at point (0, height/2) at half of its size
  image(img, 0, height/2, img.width/2, img.height/2);
}
```

Create Image createImage()

http://processing.org/reference/createImage_.html
http://processing.org/reference/pixels.html

SYNTAX

createImage(w, h, format)

DESCRIPTION

Creates a new PImage (the datatype for storing images). This provides a fresh buffer of pixels to play with. Set the size of the buffer with the width and height parameters. The format parameter defines how the pixels are stored. See the PImage reference for more information.

pixels[]
Array containing the values for all the pixels in the display window. These values are of the color datatype. This array is the size of the display window. For example, if the image is 100x100 pixels, there will be 10000 values and if the window is 200x300 pixels, there will be 60000 values. The index value defines the position of a value within the array. For example, the statement color b = pixels[230] will set the variable b to be equal to the value at that location in the array.

PARAMETRES

w	int: width in pixels
h	int: height in pixels
format	int: Either RGB, ARGB, ALPHA (grayscale alpha channel)

CODE

```
/**
 * Background Image.
 *
 * This example presents the fastest way to load a background image
 * into Processing. To load an image as the background, it must be
 * the same width and height as the program.
 */

/**
 * Create Image.
 *
 * The createImage() function provides a fresh buffer of pixels to play with.
 * This example creates an image gradient.
 */

PImage bg, img;
int y;

void setup() {
  size(640, 360);
  // The background image must be the same size as the parameters
  // into the size() method. In this program, the size of the image
  // is 640 x 360 pixels.
  bg = loadImage("moonwalk.jpg");

  // build gradient
  img = createImage(width, 230, ARGB);
  for (int i = 0; i < img.pixels.length; i++) {
    float a = map(i, 0, img.pixels.length, 255, 0);
    img.pixels[i] = color(0, 153, 204, a);
  }
}

void draw() {
  background(bg);
  strokeWeight(5);
  stroke(226, 204, 0);
  line(0, y, width, y);

  y++;
  if (y > height) {
    y = 0;
  }
  image(img, 0, y);
  //image(img, mouseX-img.width/2, mouseY-img.height/2);
}
```

Alpha, RGB Colors alpha(), red(), green(), blue()

http://processing.org/reference/alpha_.html
http://processing.org/reference/red_.html
http://processing.org/reference/imageMode_.html
http://processing.org/reference/blue_.html

SYNTAX

alpha(rgb)
red(rgb), green(rgb)), blue(rgb)

DESCRIPTION

alpha()
Extracts the alpha value from a color.

red(), green(),
Extracts the red value from a color, scaled to match current colorMode(). The value is always returned as a float, so be careful not to assign it to an int value.

The red() function is easy to use and understand, but it is slower than a technique called bit shifting. When working in colorMode(RGB, 255), you can acheive the same results as red() but with greater speed by using the right shift operator (>>) with a bit mask.

The green() function is easy to use and understand, but it is slower than a technique called bit shifting. When working in colorMode(RGB, 255), you can acheive the same results as green() but with greater speed by using the right shift operator (>>) with a bit mask.

The blue() function is easy to use and understand, but it is slower than a technique called bit masking. When working in colorMode(RGB, 255), you can acheive the same results as blue() but with greater speed by using a bit mask to remove the other color components. For example, the following two lines of code are equivalent means of getting the blue value of the color value c:

```
float b1 = blue(c);  // Simpler, but slower to calculate
float b2 = c & 0xFF;  // Very fast to calculate
```

PARAMETRES

rgb int: any value of the color datatype

CODE

```
/**
 * Alpha Mask.
 *
 * Loads a "mask" for an image to specify the transparency
 * in different parts of the image. The two images are blended
 * together using the mask() method of PImage.
 */

// The next line is needed if running in JavaScript Mode with Processing.js
/* @pjs preload="moonwalk.jpg,mask.jpg"; */

PImage img;
PImage imgMask;

void setup() {
  size(640, 360);
  img = loadImage("moonwalk.jpg");
  imgMask = loadImage("mask.jpg");
  img.mask(imgMask);
  imageMode(CENTER);
}

void draw() {
  background(0, 102, 153);
  image(img, width/2, height/2);
//  image(img, mouseX, mouseY);
}
```

Color Manipulation brightness(), hue(), saturation()

http://processing.org/reference/brightness_.html
http://processing.org/reference/hue_.html
http://processing.org/reference/hue_.html

SYNTAX brightness(rgb), hue(rgb), saturation(rgb)

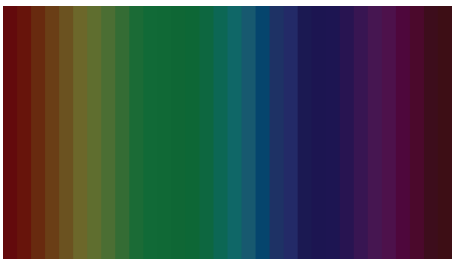
DESCRIPTION brightness(): Extracts the brightness value from a color.

hue(): Extracts the hue value from a color.

saturation(): Extracts the saturation value from a color.

PARAMETRES rgb int: any value of the color datatype

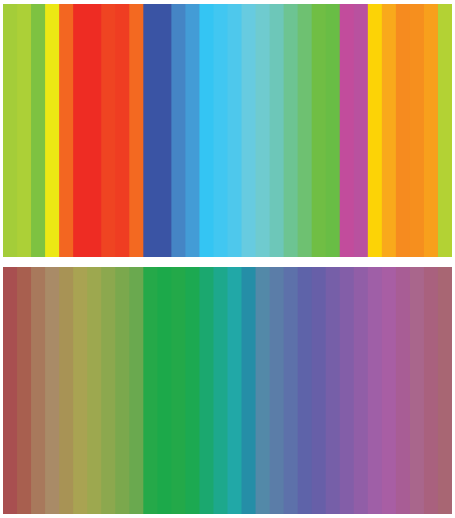
CODE



```
int barWidth = 20;
int lastBar = -1;

void setup() {
  size(640, 360);
  colorMode(HSB, width, 100, width);
  noStroke();
  background(255);
}

void draw() {
  int whichBar = mouseX / barWidth;
  if (whichBar != lastBar) {
    int barX = whichBar * barWidth;
    fill(barX, 100, mouseY);
    rect(barX, 0, barWidth, height);
    lastBar = whichBar;
  }
}
```

```
/**
 * Brightness
 * by Rusty Robison.
 *
 * Brightness is the relative lightness or
darkness of a color.
 * Move the cursor vertically over each
bar to alter its brightness.
 */
```

CODE



```
/**
 * Hue.
 *
 * Hue is the color reflected from or transmitted through an object
 * and is typically referred to as the name of the color (red, blue, yellow, etc.)
 * Move the cursor vertically over each bar to alter its hue.
 */


/**
 * Saturation.
 *
 * Saturation is the strength or purity of the color and represents the
 * amount of gray in proportion to the hue. A "saturated" color is pure
 * and an "unsaturated" color has a large percentage of gray.
 * Move the cursor vertically over each bar to alter its saturation.
```

```
-----------------------
int barWidth = 20;
int lastBar = -1;

void setup()
{
  size(640, 360);
  colorMode(HSB, height, height, height);
  noStroke();
  background(0);
}

void draw()
{
  int whichBar = mouseX / barWidth;
  if (whichBar != lastBar) {
    int barX = whichBar * barWidth;
    fill(mouseY, height, height);
    rect(barX, 0, barWidth, height);
    lastBar = whichBar;
  }
}

-----------------------


int barWidth = 20;
int lastBar = -1;


void setup() {
  size(640, 360);
  colorMode(HSB, width, height, 100);
  noStroke();
}


void draw() {
  int whichBar = mouseX / barWidth;
  if (whichBar != lastBar) {
    int barX = whichBar * barWidth;
    fill(barX, mouseY, 66);
    rect(barX, 0, barWidth, height);
    lastBar = whichBar;
  }
```

Linear Gradient lerpColor()

http://processing.org/reference/lerpColor_.html

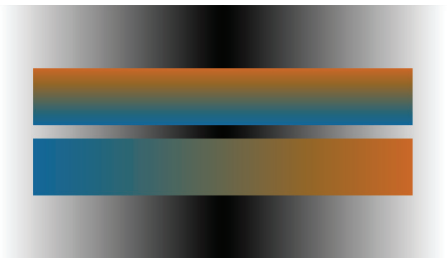
SYNTAX lerpColor(c1, c2, amt)

DESCRIPTION Calculates a color or colors between two color at a specific increment. The amt parameter is the amount to interpolate between the two values where 0.0 equal to the first point, 0.1 is very near the first point, 0.5 is halfway in between, etc.

PARAMETRES

c1	int: interpolate from this color
c2	int: interpolate to this color
amt	float: between 0.0 and 1.0

CODE

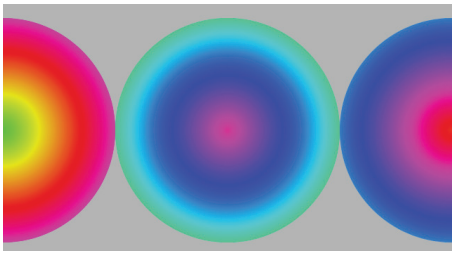


```
int Y_AXIS = 1;
int X_AXIS = 2;
color b1, b2, c1, c2;

void setup() {
  size(640, 360);

  // Define colors
  b1 = color(255);
  b2 = color(0);
  c1 = color(204, 102, 0);
  c2 = color(0, 102, 153);
  noLoop();
}

void draw() {
  // Background
  setGradient(0, 0, width/2, height, b1, b2, X_AXIS);
  setGradient(width/2, 0, width/2, height, b2, b1, X_AXIS);
  // Foreground
  setGradient(50, 90, 540, 80, c1, c2, Y_AXIS);
  setGradient(50, 190, 540, 80, c2, c1, X_AXIS);
}

void setGradient(int x, int y, float w, float h, color c1, color c2, int axis
) {

  noFill();

  if (axis == Y_AXIS) {  // Top to bottom gradient
    for (int i = y; i <= y+h; i++) {
      float inter = map(i, y, y+h, 0, 1);
      color c = lerpColor(c1, c2, inter);
      stroke(c);
      line(x, i, x+w, i);
    }
  }
  else if (axis == X_AXIS) {  // Left to right gradient
    for (int i = x; i <= x+w; i++) {
      float inter = map(i, x, x+w, 0, 1);
      color c = lerpColor(c1, c2, inter);
      stroke(c);
      line(i, y, i, y+h);
    }
  }
}
```

```
/**
 * Simple Linear Gradient
 *
 * The lerpColor() function is useful for
interpolating
 * between two colors.
 */
```

RadialGradient

CODE



```
int dim;

void setup() {
  size(640, 360);
  dim = width/2;
  background(0);
  colorMode(HSB, 360, 100, 100);
  noStroke();
  ellipseMode(RADIUS);
  frameRate(1);
}

void draw() {
  background(0);
  for (int x = 0; x <= width; x+=dim) {
    drawGradient(x, height/2);
  }
}

void drawGradient(float x, float y) {
  int radius = dim/2;
  float h = random(0, 360);
  for (int r = radius; r > 0; --r) {
    fill(h, 90, 90);
    ellipse(x, y, r, r);
    h = (h + 1) % 360;
  }
}
```

```
/**
 * Radial Gradient.
 *
 * Draws are series of concentric circles
to create a gradient
 * from one color to another.
 */
```

WaveGradient

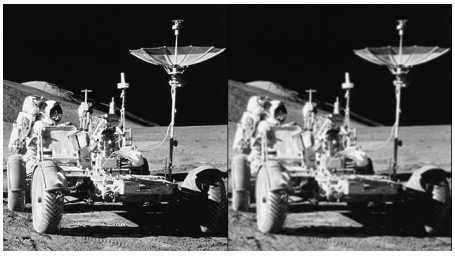
CODE

```
float angle = 0;
float px = 0, py = 0;
float amplitude = 30;
float frequency = 0;
float fillGap = 2.5;
color c;

void setup() {
  size(640, 360);
  background(200);
  noLoop();
}

void draw() {
  for (int i =- 75; i < height+75; i++){
    // Reset angle to 0, so waves stack properly
    angle = 0;
    // Increasing frequency causes more gaps
    frequency+=.002;
    for (float j = 0; j < width+75; j++){
      py = i + sin(radians(angle)) * amplitude;
      angle += frequency;
      c =  color(abs(py-i)*255/amplitude, 255-abs(py-i)*255/amplitude,
j*(255.0/(width+50)));
      // Hack to fill gaps. Raise value of fillGap if you increase frequency
      for (int filler = 0; filler < fillGap; filler++){
        set(int(j-filler), int(py)-filler, c);
        set(int(j), int(py), c);
        set(int(j+filler), int(py)+filler, c);
      }
    }
  }
}
```

```
/**
 * Wave Gradient
 * by Ira Greenberg.
 *
 * Generate a gradient along a sin()
wave.
 */
```

Blur

CODE

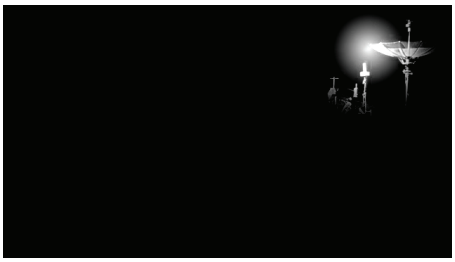


```
/**
 * Blur.
 *
 * A low-pass filter blurs an image. This program analyzes every
 * pixel in an image and blends it with the neighboring pixels
 * to blur the image.
 */

// The next line is needed if running in JavaScript Mode with Processing.js
/* @pjs preload="moon.jpg"; */
```

```
float v = 1.0 / 9.0;
float[][] kernel = {{ v, v, v },
                    { v, v, v },
                    { v, v, v }};

PImage img;

void setup() {
  size(640, 360);
  img = loadImage("moon.jpg"); // Load the original image
  noLoop();
}

void draw() {
  image(img, 0, 0); // Displays the image from point (0,0)
  img.loadPixels();

  // Create an opaque image of the same size as the original
  PImage edgeImg = createImage(img.width, img.height, RGB);

  // Loop through every pixel in the image
  for (int y = 1; y < img.height-1; y++) {   // Skip top and bottom edges
    for (int x = 1; x < img.width-1; x++) {  // Skip left and right edges
      float sum = 0; // Kernel sum for this pixel
      for (int ky = -1; ky <= 1; ky++) {
        for (int kx = -1; kx <= 1; kx++) {
          // Calculate the adjacent pixel for this kernel point
          int pos = (y + ky)*img.width + (x + kx);
          // Image is grayscale, red/green/blue are identical
          float val = red(img.pixels[pos]);
          // Multiply adjacent pixels based on the kernel values
          sum += kernel[ky+1][kx+1] * val;
        }
      }
      // For this pixel in the new image, set the gray value
      // based on the sum from the kernel
      edgeImg.pixels[y*img.width + x] = color(sum);
    }
  }
  // State that there are changes to edgeImg.pixels[]
  edgeImg.updatePixels();

  image(edgeImg, width/2, 0); // Draw the new image
```

Brightness

CODE

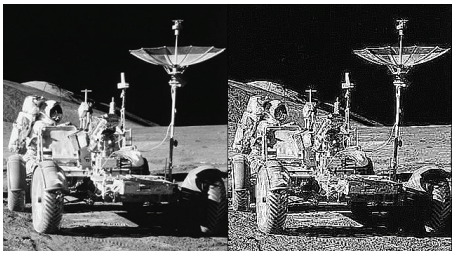


```
/**
 * Brightness
 * by Daniel Shiffman.
 *
 * This program adjusts the brightness of a part of the image by
 * calculating the distance of each pixel to the mouse.
 */

// The next line is needed if running in JavaScript Mode with Processing.js
/* @pjs preload="moon-wide.jpg"; */
```

```
PImage img;

void setup() {
  size(640, 360);
  frameRate(30);
  img = loadImage("moon-wide.jpg");
  img.loadPixels();
  // Only need to load the pixels[] array once, because we're only
  // manipulating pixels[] inside draw(), not drawing shapes.
  loadPixels();
}

void draw() {
  for (int x = 0; x < img.width; x++) {
    for (int y = 0; y < img.height; y++ ) {
      // Calculate the 1D location from a 2D grid
      int loc = x + y*img.width;
      // Get the R,G,B values from image
      float r,g,b;
      r = red (img.pixels[loc]);
      //g = green (img.pixels[loc]);
      //b = blue (img.pixels[loc]);
      // Calculate an amount to change brightness based on proximity to the mouse
      float maxdist = 50;//dist(0,0,width,height);
      float d = dist(x, y, mouseX, mouseY);
      float adjustbrightness = 255*(maxdist-d)/maxdist;
      r += adjustbrightness;
      //g += adjustbrightness;
      //b += adjustbrightness;
      // Constrain RGB to make sure they are within 0-255 color range
      r = constrain(r, 0, 255);
      //g = constrain(g, 0, 255);
      //b = constrain(b, 0, 255);
      // Make a new color and set pixel in the window
      //color c = color(r, g, b);
      color c = color(r);
      pixels[y*width + x] = c;
    }
  }
  updatePixels();
}
```

EdgeDetection

CODE



```
/**
 * Edge Detection.
 *
 * A high-pass filter sharpens an image. This program analyzes every
 * pixel in an image in relation to the neighboring pixels to sharpen
 * the image. This example is currently not accurate in JavaScript mode.
 */

// The next line is needed if running in JavaScript Mode with Processing.js
/* @pjs preload="moon.jpg"; */
```

```
float[][] kernel = {{ -1, -1, -1},
                    { -1,  9, -1},
                    { -1, -1, -1}};

PImage img;

void setup() {
  size(640, 360);
  img = loadImage("moon.jpg"); // Load the original image
  noLoop();
}

void draw() {
  image(img, 0, 0); // Displays the image from point (0,0)
  img.loadPixels();
  // Create an opaque image of the same size as the original
  PImage edgeImg = createImage(img.width, img.height, RGB);
  // Loop through every pixel in the image.
  for (int y = 1; y < img.height-1; y++) { // Skip top and bottom
edges
    for (int x = 1; x < img.width-1; x++) { // Skip left and right edges
      float sum = 0; // Kernel sum for this pixel
      for (int ky = -1; ky <= 1; ky++) {
        for (int kx = -1; kx <= 1; kx++) {
          // Calculate the adjacent pixel for this kernel point
          int pos = (y + ky)*img.width + (x + kx);
          // Image is grayscale, red/green/blue are identical
          float val = red(img.pixels[pos]);
          // Multiply adjacent pixels based on the kernel values
          sum += kernel[ky+1][kx+1] * val;
        }
      }
      // For this pixel in the new image, set the gray value
      // based on the sum from the kernel
      edgeImg.pixels[y*img.width + x] = color(sum, sum, sum);
    }
  }
  // State that there are changes to edgeImg.pixels[]
  edgeImg.updatePixels();
  image(edgeImg, width/2, 0); // Draw the new image
}
```

Histogram

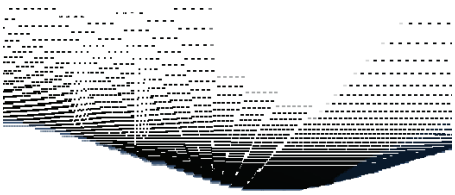
CODE

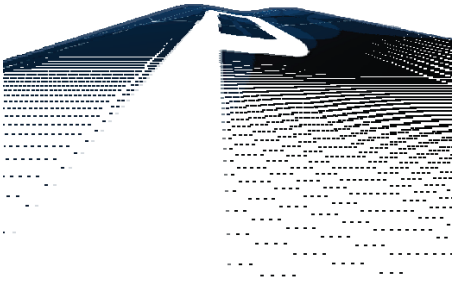
```
/**
 * Histogram.
 *
 * Calculates the histogram of an image.
 * A histogram is the frequency distribution
 * of the gray levels with the number of pure black values
 * displayed on the left and number of pure white values on the right.
 *
 * Note that this sketch will behave differently on Android,
 * since most images will no longer be full 24-bit color.
 */

// The next line is needed if running in JavaScript Mode with Processing.js
/* @pjs preload="frontier.jpg"; */
```

```
import java.util.Calendar;

String timestamp() {
  return String.format("%1$ty%1$tm%1$td_%1$tH%1$tM%1$tS",
Calendar.getInstance());
}

// Load an image from the data directory
// Load a different image by modifying the comments

int[] hist = new int[256];

void setup() {
  size(640, 360);
  PImage img = loadImage("frontier.jpg");
  image(img, 0, 0);

  // Calculate the histogram
  for (int i = 0; i < img.width; i++) {
    for (int j = 0; j < img.height; j++) {
      int bright = int(brightness(get(i, j)));
      hist[bright]++;
    }
  }

  // Find the largest value in the histogram
  int histMax = max(hist);

  stroke(255);
  // Draw half of the histogram (skip every second value)
  for (int i = 0; i < img.width; i += 2) {
    // Map i (from 0..img.width) to a location in the histogram (0..255)
    int which = int(map(i, 0, img.width, 0, 255));
    // Convert the histogram value to a location between
    // the bottom and the top of the picture
    int y = int(map(hist[which], 0, histMax, img.height, 0));
    line(i, img.height, i, y);
  }
}

void draw() {
}

void keyPressed() {
  if (key=='s' || key=='S') saveFrame(timestamp()+"_##.png");
}
```

Image Distortion, Save the Screen Window

CODE





```
import java.util.Calendar; // loads the current Calendar
String timestamp() { // apply the current Calentar
  return String.format("%1$ty%1$tm%1$td_%1$tH%1$tM%1$tS",
Calendar.getInstance());
}


float[][] pArray;
PImage img;
float f;
int w;
int h;
int s=2;
int mode = 1;
void setup() {
  size(640,360);
  textFont(createFont("Calibri-12.vlw",12));
  img = loadImage("processing.png");
  w = img.width;
  h = img.height;
  pArray = new float[w][h];
  noStroke();
}
void draw() {
  background(25);
  for (int y=h-1;y-->0;f+=0.1) {
    float mx=mouseX,my=mouseY,wi=width;
    for (int x=w-1;x-->0;pArray[x][y]=sin((x+f)/(0.1+mx/
wi*100.0))*10) {
      //Offsets the y position of the pixel by the sin.
      int x2 = x*s;
      int y2 = (y+(int)pArray[x][y])*s;

      if (mode == 1) {
        //Rendering mode 1, 2d.
        y2+=25;
        fill(img.get(x,y));
        rect(x2,y2,s,s);
        if (get(x2,y2-s) != img.get(x,y)) rect(x2,y2-s,s,s);
      }
      else {
        //Rendering mode 2, 3d.
        fill(img.get(x,y));
```

```
      float z = ((h-y2)/2+80);
      float e=100;
      int bx = width/2+(int)((x - e)*(e/z))-10;
      int by = height/2+(int)((y+y-h - e)*(e/z))/2+60;

      rect(bx,by,s,s);
      if (get(bx-s,by) != img.get(bx,by)) rect(bx-s,by,s,s);
    }
   }
  }
  fill(255);
  if (mode==1) text("Click to switch to 3d mode",10,10);
  else text("Click to switch to 2d mode",10,10);
}

void mousePressed() {
  if (mode == 1) mode = 0;
  else mode = 1;
}

void keyPressed() { // saves the sreen window
  if (key=='s' || key=='S') saveFrame(timestamp()+"_##.png");
}
```

```
/*
Made by SumWon

Instructions:

Move the mouse along the x axis
to increase and decrease the frequency
of the sign wave.

Click anywhere to switch from 2d mode
to 3d mode and vice versa.

*/
```

Video Play, Pause and Loop

CODE

```
/**
 * Loop.
 *
 * Move the cursor across the screen to
draw.
 * Shows how to load and play a
QuickTime movie file.
 */
```

```
import processing.video.*;

Movie myMovie;
Boolean VideoStatus=true;

void setup() {
  size(320, 320);
  background(123);
  // Load and play the video in a loop
  myMovie = new Movie(this, "station.mov");
  myMovie.loop();
  imageMode(CENTER);

}

void movieEvent(Movie myMovie) {
  myMovie.read();
}

void draw() {
  image(myMovie, width/2,height/2);
   println(VideoStatus + " " + int(key) + " " + keyCode);
}

void keyPressed() {
  if (key==32) {
    if (VideoStatus==true) {
    VideoStatus=false;
    myMovie.pause();
    } else {
    if (VideoStatus==false) {
    VideoStatus=true;
    myMovie.play();
     }
    }
  }
}
```

Video Forward/Rewind

CODE

```
/**
 * Forward Rewind the video
 *
 * Move the cursor across the screen to draw.
 * Shows how to load and play a QuickTime movie file.
 */
```

```
import processing.video.*;

Movie myMovie;

void setup() {
  size(400, 400);
  background(0);
  // Load and play the video in a loop
  myMovie = new Movie(this, "station.mov");
  myMovie.loop();
  //  frameRate(120);
  imageMode(CENTER);
}

void movieEvent(Movie myMovie) {
  myMovie.read();
}

void draw() {
  myMovie.speed(int (mouseX-width/2)/40);
  image(myMovie, width/2, height/2);
}
```