Processing CHEAT SHEET

OPENLAB

Basic structure
This is the basic of any Processing sketch.

```
void setup(){
//Runs only once.
}
void draw(){
//Runs repeatedly during the execution.
}
```

Variables Types

`int`
Positive and negative integer variables.

`float`
Floating point negative and positive variables.

`boolean`
Variables of which values can be: TRUE or FALSE.

`color`
Stores color type values in different formats.

`char`
Stores a single character.

`string`
Stores a single string.

Basic functions

`size(width,height);`
Sets main window size in pixels.

`background(color);`
Sets window background color.

`smooth();`
Sets antialiasing on.

`frameRate(fps);`
Sets the application's FPS.

`println(string);`
Writes a string to the console.

Random & Noise

`random(low,high);`
Returns a random value within the limits.

`randomSeed(seed);`
Changes random seed.

`noise(value);`
Returns a value in Perlin Noise sequence.

`noiseDetail(octaves);`
Sets detail threshold for noise function results.

`noiseSeed(seed);`
Changes noise seed.

Global variables
These variables can be called anytime, anywhere.

`width`
Returns sketch's width in pixels.

`height`
Returns sketch's height in pixels.

`mouseX`
Return mouse pointer's position (X axis).

`mouseY`
Return mouse pointer's position (Y axis).

`pmouseX`
Returns previous mouse pointer's X axis.

`pmouseY`
Returns previous mouse pointer's Y axis.

`frameCount`
Returns sketch's current frame.

`frameRate`
Returns sketch's current FPS.

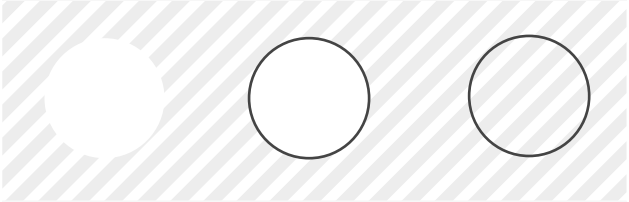
fill(),noFill(),stroke(),noStroke()
Shapes border, stroke & fill setting functions.

`fill(color);`
Sets the color used to fill shapes.

`noFill();`
Disables fill color.

`stroke(color);`
Sets the color used to draw strokes/borders around shapes.

`noStroke();`
Disables border for shapes.



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

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

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

Color functions

`colorMode(mode);`
Set color mode. Usually RGB or HSB.

`red(color);`
Return the red value of the color.

`green(color);`
Return the green value of the color.

`blue(color);`
Return the blue value of the color.

`hue(color);`
Return the hue value of the color.

`saturation(color);`
Return the saturation value of the color.

`brightness(color);`
Return the brightness value of the color.

`alpha(color);`
Return the transparency value of the color.

`lerpColor(color1,color2,moment);`
Returns a color value between two colors.

"There are two ways to write error-free programs; only the third one works." Alan J. Perlis

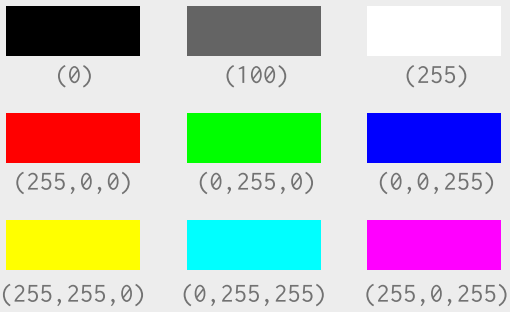
"Don't worry if it doesn't work right. If everything did, you'd be out of a job." Mosher's Law of Software Engineering

Color handling
These are the ways to pass color arguments.

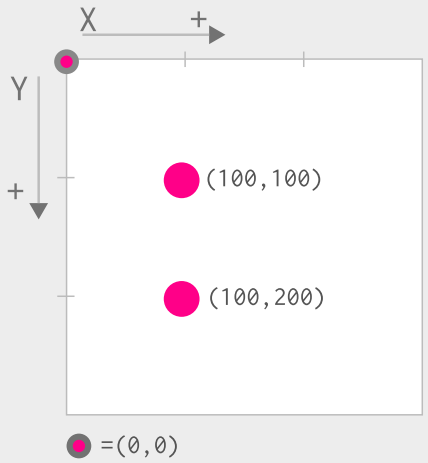
```
// Value scale goes from 0 to 255
color(grayscale);
color(grayscale,alpha);
color(red,green,blue);
color(red,green,blue,alpha);
```

* Components change depending on the number of arguments.

Color examples



Coordinates system



In any Processing sketch, top left corner is the (0, 0) point. That axis changes when we make use of translate() or rotate().

Minimum unit of measurement in a computer screen is the Pixel.

Matrix operations

`pushMatrix();`
Saves the current matrix. Meaning the “translate, rotate and scale” values. For every pushMatrix( ) belongs a final popMatrix( ).

`popMatrix();`
Allows you to go back to the last saved matrix. You need a previous pushMatrix( ) to go back to an older matrix.

`printMatrix();`
Prints current matrix to the console.

`translate(posx,posy);`
Moves the anchor point to a certain position. After this the (0,0) is the certain position.

`rotate(radians);`
Changes plane rotation according to axis.

`scale(x,y);`
Scales the plane, affects all sizes on the sketch can also be: scale(x,y,z) or scale(multiple).

`shearX(radians);`
Applies a shear on X axis.

`shearY(radians);`
Applies a shear on Y axis.

`rotateX(radians);`
Applies rotation to X axis. Works only on 3D enviroments.

`rotateY(radians);`
Applies rotation to Y axis. Works only on 3D enviroments.

`rotateZ(radians);`
Applies rotation to Z axis. Works only on 3D enviroments.

`pushStyle();`
Saves the current style of fill(), stroke(), tint(), strokeWeight(), strokeCap(), strokeJoin(), imageMode(), rectMode(), ellipseMode(), shapeMode(), colorMode(), textAlign(), textFont(), textMode(), textSize(), textLeading(), emissive(), specular(), shininess(), ambient().

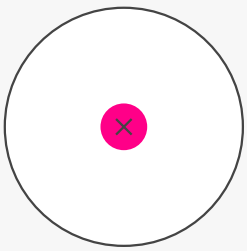
`popStyle();`
Goes back to last style used. You need a previous pushStyle() to go back to an older style.

"Everything should be made as simple as possible, but not simpler." Albert Einstein

"In theory there is no difference between theory and practice. In practice there is." Yogi Berra

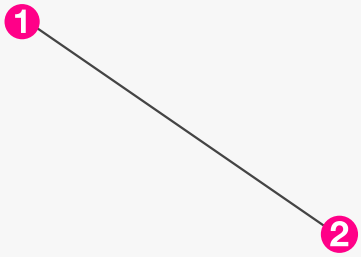
Basic geometry

Anchor point



`ellipse(posx,posy,width,height);`
Draws an ellipse centered in position (posx, posy) and with size "width" and "height".

`rect(posx,posy,width,height);`
Draws a rect anchored at top left corner, in position (posx, posy) and with size "width" and "height".



`line(posx1,posy1,posx2,posy2);`
Draws a line from point (posx1, posy1) to (posx2, posy2)

Other primitive shapes

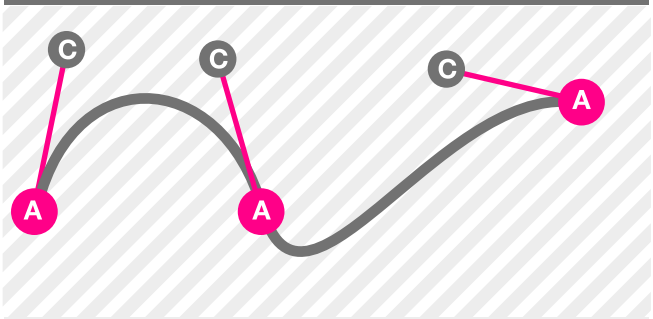
`point(posx,posy);`
Draws a point to the screen.

`quad(x1,y1,x2,y2,x3,y3,x4,y4);`
Draws a quadrilateral based upon the four vertex positions we pass.

`arc(posx,posy,width,height,startangle,endangle);`
Draws an arc in position (posx, posy), with size "width" and "height", and "startangle" and "endangle" passed as radians.

`triangle(x1,y1,x2,y2,x3,y3);`
Draws a triangle based upon three positions passed as arguments.

Bezier and curves



`bezier(x1, y1, x2, y2, x3, y3, x4, y4)`
Draws a Bezier. positions 1 and 4 are the main anchor points, 2 and 3 work as control points.

`bezierDetail(level);`
Sets the Bezier detail level.

`bezierTangent(a, b, c, d, moment);`
Returns the Bezier's tangent at "time".

`bezierPoint(a, b, c, d, moment);`
Returns the Bezier's axis position at "time".

`curve(x1, y1, x2, y2, x3, y3, x4, y4);`
Draws a curve. positions 1 and 4 are the main anchor points, 2 and 3 work as control points.

`curveTightness(tightness);`
Sets tightness for the next curves.

`curvePoint(a, b, c, d, t);`
Returns the curve's axis position at "time".

`curveTangent(a, b, c, d, t);`
Returns the curve's tangent at "time".

`curveDetail(detail);`
Sets the curve detail level.

beginShape( ) and endShape( )

`beginShape();`
Starts listening for vertices to build a shape. It stops listening when endShape() is called. Modes can be passed as arguments.

`endShape();`
Stops listening for vertices.

`vertex(posx,posy);`
Draws a vertex in position (posx, posy)

`bezierVertex(x2, y2, x3, y3, x4, y4);`
Defines a vertex based on a Bezier curve.

`curveVertex(x, y);`
Defines a vertex based on a curve.

`texture(PImage);`
Sets the texture for a drawn shape.

`beginContour();`
Starts listening for vertices to cut a previous form.

`endContour();`
Stops listening for "beginContour" vertices.

* Available modes for beginShape are POINTS, LINES, TRIANGLES, TRIANGLE_FAN, TRIANGLE_STRIP, QUADS and QUAD_STRIP.

"A programming language is low level when its programs require attention to the irrelevant." Alan J. Perlis

"The Analytical Engine weaves algebraic patterns, just as the Jacquard loom weaves flowers and leaves." – Ada Lovelace

Functions structure

```
// Create the function
void hello(){
println(”Hello!”);
}

// Call the function
insult();
```

Class structure

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

void methodName(/*Variables*/){

}

}

//Declare an object
ClassName myClass;

void setup(){
  //Initialize an object
  myClass = new ClassName(/*Variables*/);
}
void draw(){
  //Call an object method
  myClass.methodName();
}
```

Classes may or may not have variables or variables to be initialized.

For loops

These are codeblock that cycle through a condition.

```
//Simple usage of For loop
for(int i = 0;i<condition;i++){
//Code in here will repeat i times
}

//Nested For loop
for(int i = 0;i<condition;i++){
    for(int j = 0;j<condition;j++){
    //Code here will repeat i*j times
    }
}
```

Inside the block we can take advantage of index variables.

Conditional operators

<	<=	==	>	>=	!=
Less than	Less than or equal to	Equal	Greater than	Greater than or equal to	Not equal

Logical operators

Work as connectors between conditions

&&	||	!
AND	OR	NOT

Conditional structure

```
if(condition1){
// Code to run if condition1 is True
}else if(condition2){
// Code to run if condition2 is True
}else{
// Runs if no previous condition was True
}
```

Conditions result from values comparison using logical or conditional operators.

While structure

```
while(condition1){
// Code to run until condition1
// becomes False
}
```

PLEASE SHARE THIS

“Language shapes the way we think, and determines what we can think about.” B. L. Whorf

"Any sufficiently advanced technology is indistinguishable from magic." Arthur C. Clarke

Display an image
Images must be stored in your sketch's "data" directory.

```
PImage img;

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

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

Supported formats: **JPG**, **GIF**, **TGA** and **PNG**.

Display text
Fonts must be stored in your sketch's "data" directory.

```
PFont font;

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

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

We can create fonts in **VLW** format using the menu function: **Tools / Create Font..**

Display a shape
Vectors must be stored in your sketch "data" directory.

```
PShape myshape;

void setup() {
  myshape = loadShape("myShape.svg");
}

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

Supported formats: **SVG**.

Image functions

`image(img,posx,posy,width,height);`
Draws an image in the main screen.

`loadImage(fileName);`
Initializes a PImage passing an image file name or path as an argument

`requestImage(fileName);`
Initializes a PImage on a separate thread.

`tint(color);`
Sets the tint value of an image.

`noTint();`
Disables image tint.

`saveFrame(filename);`
Saves a screenshot of the current frame.

Text functions

`text(string,posx,posy);`
Displays a text on the screen.

`loadFont(fileName);`
Initializes a PFont passing a font file name or path as an argument.

`textFont(font,size);`
Sets font type and size.

`textAlign(mode);`
Sets align mode to: LEFT, RIGHT or CENTER.

`textLeading(size);`
Sets the spacing between lines of text in units of pixels.

Easing target
Easing allows us to smooth the passing of values.

```
float x;
//Easing value
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);
}
```

Example taken from "Getting started with Processing" by Reas & Fry. O'Reilly / Make 2010

"There is not now, and never will be, a language in which it is the least bit difficult to write bad programs." Flon's Law

"Programs must be written for people to read, and only incidentally for machines to execute." Abelson / Sussman

Events capture

`void mousePressed()`
Runs when any mouse button is pressed.

`void mouseClicked()`
Runs when any mouse button is pressed and released.

`void mouseMoved()`
Runs everytime mouse is moved and NOT pressed.

`void mouseDragged()`
Runs everytime mouse is moved while a button is pressed.

`void mouseReleased()`
Runs when any mouse button is released.

`void keyPressed()`
Runs on a key press event.

`void keyTyped()`
Runs when a key is pressed except for SHIFT, CTRL or ALT.

`void keyReleased()`
Runs on a key release event.

keyPressed (Boolean)
Returns True or False if any key is pressed.

```
void draw() {
  if(keyPressed == true) {
    fill(0); //If any key is pressed
  } else {
    fill(255); //Otherwise...
  }
  rect(25, 25, 50, 50);
}
```

It's a special variable we can use for cheking a keypress status.

mousePressed (Boolean)
Returns True or False if any mouse button.

```
void draw() {
  if(mousePressed == true) {
    fill(0); //If mouse is pressed
  } else {
    fill(255); //Otherwise...
  }
  rect(25, 25, 50, 50);
}
```

It's a special variable we can use for cheking mouseclick status

key
It's a special variables which returns the last key pressed.

```
void draw() {
  if (keyPressed) {
    if (key == 'b' || key == 'B') {
      // If B key is pressed
    }
  } else {
    // Otherwise...
  }
}
```

key variable is case sensitive.

keyCode
Special variable for detecting special keys.

```
void keyPressed() {
  if (key == CODED) {
    if (keyCode == UP) {
    // If UP arrow key is pressed.
    } else if (keyCode == DOWN) {
    // If down arrow key is pressed.
    }
  } else {
    // Otherwise...
  }
}
```

Other keys: **BACKSPACE** , **TAB** , **ENTER** , **RETURN** , **ESC**, **DELETE** , **RIGHT** , **LEFT**.

"When someone says, "I want a programming language in which I need only say what I want done," give him a lollipop." Alan J. Perlis

"Computers are good at following instructions, but not at reading your mind." Donald Knuth

One dimension array

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

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

* Arrays index starts from 0

Arrays & For loops

```
//Declaration of an array.
int [] arrayInt;

void setup(){
//Set the array's size.
arrayInt = new int[50];

//Initialize each index value.
 for(int i = 0; i<arrayInt.length; i++){
 arrayInt[i] = i;
 }
}
void draw(){
//Print each array's index value.
 for(int i = 0; i<arrayInt.length; i++){
 println(arrayInt[i]);
 }
}
```

* For loops allow to quickly initialize each array's index.

Array functions

`append(array,value);`
Add a value to an array.

`arrayCopy(src, srcPos, dst, dstPos, length);`
Copy an array or part of it into another.

`concat(a,b);`
Concatenates two arrays.

`expand(array,newSize);`
Expands and array's size value.

`reverse(array);`
Reverses an array order.

`shorten(array);`
Reduces array's size by one index.

`sort(array);`
Sorts an array in increasing order.

`splice(array,value/array,index);`
Inserts a value or an array inside any given index.

`subset(array, start, count)`
Extracts a set from a given array from "start" to "count".

Objects array
Arrays can be made up from a given class.

```
//Declaration of an object array.
Particle[] particles;

void setup(){
//Set the size of our array.
particles = new Particle[50];

//Initializing every index.
for(int i = 0; particles.length; i++){
particles[i] = new Particle();
}

}
void draw(){
//Call a function of each object.
for(int i = 0; i<particles.length; i++){
particles[i].draw();
}
}
```

Two-dimensional arrays
These arrays can be called with two values.

```
//Declaration of our array.
int [][] array2D;

void setup(){
//Declaration of our array.
array2D = new int[width][height];

//Initializing every index.
for(int i = 0; i<width; i++){
  for(int j = 0; j<height; j++){
  array2D[i][j] = int(random(100));
  }
}
}

void draw(){
//Displaying each index value.
for(int i = 0; i<width; i++){
  for(int j = 0; i<height; j++){
  println(array2D[i][j]);
  }
}
}
```

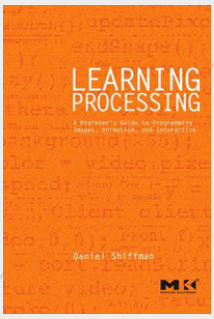
* To go over a multiple dimensions array we have to use nested For loops.

"First, solve the problem. Then, write the code." John Johnson

"Life would be much easier if I had the source code."

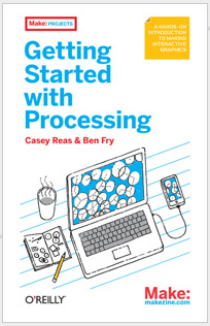
Some reference books.

Learning Processing: A Beginner's Guide



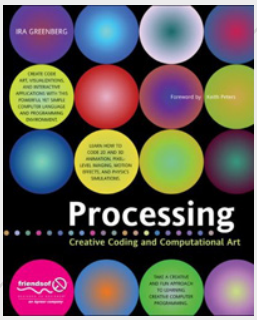
An excellent book for beginners. Covers a lot of topics. Perfectly explained.

Getting Started with Processing.



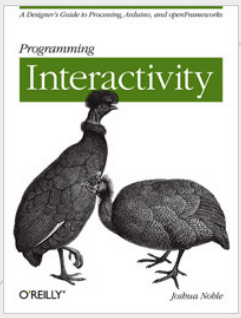
A good book as a complement to "Learning Processing", both of them make a good introduction.

Processing: Creative coding and Computational Art.



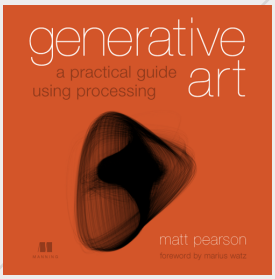
Another good alternative to start with Processing. Contains a diverse amount of examples.

Programming Interactivity.



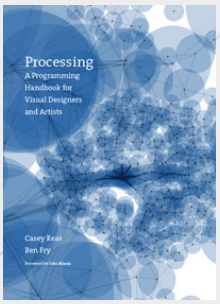
It's an introduction to Processing, Openframeworks and Arduino. Covers many aspects of of the three.

Generative Art: a practical guide using processing.



It's a Generative art oriented book. Covers some Processing based projects and comes with a lot of examples to download.

Processing: A programming handbook.



This book reviews some important aspects to go further in the task of learning Processing.

"Computers are useless. They can only give you answers." Pablo Picasso

"Beware of bugs in the above code; I have only proved it correct, not tried it." Donald Knuth

Useful links

`processing.org`
Official Processing's website. Documentation and download.

`openprocessing.org`
Open Processing community where you can upload and review related works.

`wiki.processing.org`
Processing's official Wiki.

`forum.processing.org`
Processing official forum.

`vimeo.com/channels/processing`
Processing channel on Vimeo.

`flickr.com/groups/processing/`
Processing account on Flickr.

`creativeapplications.net`
This forum gathers digital installations and works made with Processing and other creative coding tools.

`createdigitalmotion.com`
One of the most updated blogs with information about new communication media.

"The function of good software is to make the complex appear to be simple." - Grady Booch

"Controlling complexity is the essence of computer programming" Brian Kernigan

PLEASE SHARE THIS

Processing
CHEAT SHEET

openlab.surattack.com

OPENLAB

SURATTACK.COM

Questions and Suggestions: **info@surattack.com**