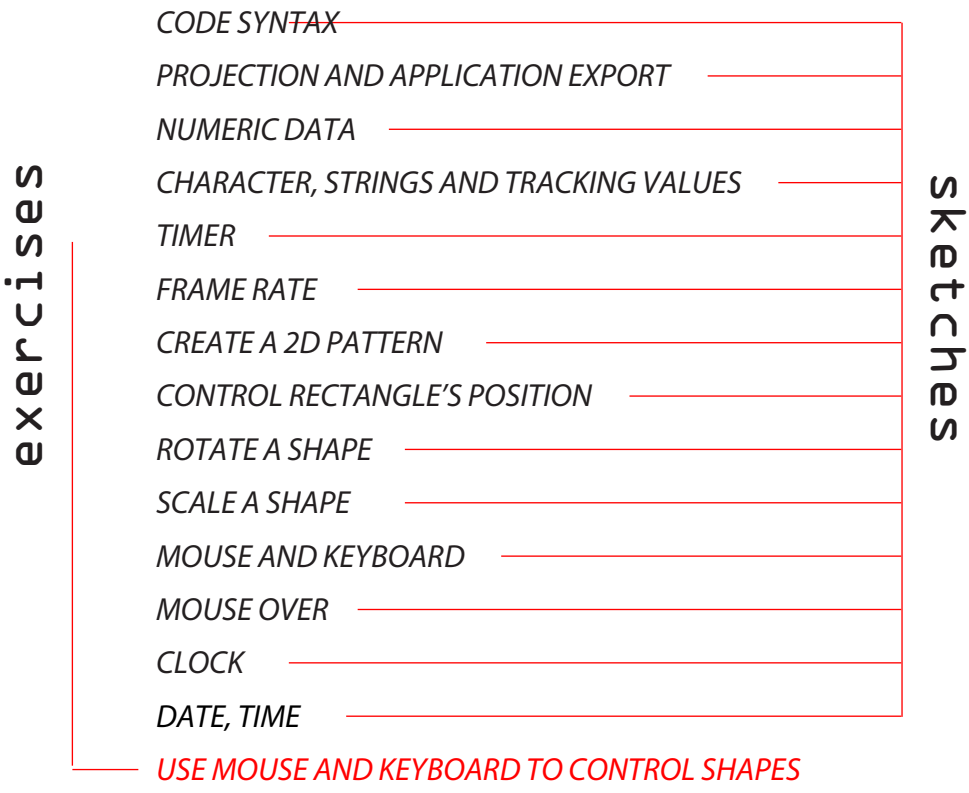
CONTROL AND CHOISE

THE ENGAGEMENT OF THE USER

creative
scripting
m e a l s

02


exercises
CODE SYNTAX
PROJECTION AND APPLICATION EXPORT
NUMERIC DATA
CHARACTER, STRINGS AND TRACKING VALUES
TIMER
FRAME RATE
CREATE A 2D PATTERN
CONTROL RECTANGLE'S POSITION
ROTATE A SHAPE
SCALE A SHAPE
MOUSE AND KEYBOARD
MOUSE OVER
CLOCK
DATE, TIME
USE MOUSE AND KEYBOARD TO CONTROL SHAPES
sketches

Methods and Oparators

SYNTAX AND GR4MMAR

Environment
cursor()
displayHeight
displayWidth
focused
frameCount
frameRate()
frameRate
height
noCursor()
size()
width

Structure
() (parentheses)
, (comma)
. (dot)
/* */ (multiline comment)
/** */ (doc comment)
// (comment)
; (semicolon)
= (assign)
[] (array access)
{} (curly braces)

catch
class
draw()
exit()
extends
false
final
implements
import
loop()
new
noLoop()
null

popStyle()
private
public
pushStyle()
redraw()
return
setup()
static
super
this
true
try
void

CONTROL

Logical Operators
! (logical NOT)
&& (logical AND)
|| (logical OR)

Relational Operators
!= (inequality)
< (less than)
<= (less than or equal to)
== (equality)
> (greater than)
>= (greater than or equal to)

Iteration
for
while

Conditionals
?: (conditional)
break
case
continue
default
else
if
switch

MATH

Operators
% (modulo)
* (multiply)
*= (multiply assign)
+ (addition)
++ (increment)
+= (add assign)
- (minus)
-- (decrement)
-= (subtract assign)
/ (divide)
/= (divide assign)

Bitwise Operators
& (bitwise AND)
<< (left shift)
>> (right shift)
| (bitwise OR)

Calculation
abs()
ceil()
constrain()
dist()
exp()
floor()
lerp()
log()
mag()
map()
max()
min()
norm()
pow()
round()
sq()
sqrt()

Trigonometry
acos()
asin()
atan()
atan2()
cos()
degrees()
radians()
sin()
tan()

Random
noise()
noiseDetail()
noiseSeed()
random()
randomGaussian()
randomSeed()

The Nature of Data

DATA

Primitive
boolean
byte
char
color
double
float
int
long

Composite
Array
ArrayList
String

Array Functions
append()
arrayCopy()
concat()
expand()
reverse()
shorten()
sort()
splice()
subset()

Conversion
binary()
boolean()
byte()
char()
float()
hex()
int()
str()
unbinary()
unhex()

String Functions
join()
match()
matchAll()
nf()
nfc()
nfp()
nfs()
split()
splitTokens()
trim()

INPUT

Mouse
mouseButton
mouseClicked()
mouseDragged()
mouseMoved()
mousePressed()
mousePressed
mouseReleased()
mouseX
mouseY
pmouseX
pmouseY

Keyboard
key
keyCode
keyPressed()
keyPressed
keyReleased()
keyTyped()

Files
BufferedReader
createInput()
createReader()
loadBytes()
loadStrings()
loadTable()
loadXML()
open()
parseXML()
selectFolder()
selectInput()

Time & Date
day()
hour()
millis()
minute()
month()
second()
year()

OUTPUT

Text Area
print()
println()

Image
save()
saveFrame()

Files
beginRaw()
beginRecord()
createOutput()
createWriter()
endRaw()
endRecord()
PrintWriter
saveBytes()
saveStream()
saveStrings()
saveXML()
selectOutput()

Code syntax setup(), draw(), size(), framerate

http://processing.org/reference/setup_.html
http://processing.org/reference/draw_.html
http://processing.org/reference/noLoop_.html
http://processing.org/reference/size_.html
http://processing.org/reference/frameRate.html

SYNTAX

```
size(w, h)
size(w, h, renderer)
frameRate(fps)
```

DESCRIPTION

setup(): The setup() function is called once when the program starts. It's used to define initial enviroment properties such as screen size and background color and to load media such as images and fonts as the program starts.

draw(): Called directly after setup(), the draw() function continuously executes the lines of code contained inside its block until the program is stopped or noLoop() is called. draw() is called automatically and should never be called explicitly. It should always be controlled with noLoop(), redraw() and loop(). After noLoop() stops the code in draw() from executing, redraw() causes the code inside draw() to execute once, and loop() will cause the code inside draw() to resume executing continuously.

size: Defines the dimension of the display window in units of pixels. The size() function must be the first line of code, or the first code inside setup(). Any code that appears before the size() command may run more than once, which can lead to confusing results.

frameRate: The system variable frameRate contains the approximate frame rate of a running sketch. The initial value is 10 fps and is updated with each frame. The value is averaged over several frames, and so will only be accurate after the draw function has run 5-10 times.

PARAMETRES

fps	frames per second
w	int: width of the display window in units of pixels
h	int: height of the display window in units of pixels
renderer	String: Either P2D, P3D, or PDF

CODE

```
// Title of the sketch
/*
 comment by others
 leave as it is
 */
// add your own comment

int a = 0; // define the variables

void setup() { // initialization process
  size(200, 200); // size of the window
  background(126); // set the background color
  frameRate(25); // set the framerate
  stroke(255, 0, 0);
}

void draw() { // main program in loop
  a++; // increase the "a" value
  line(a, 0, a, height); // prints the line
  println(a); // check the "a" value
}
```

Projections and Application Export resible, fullscreen

http://wiki.processing.org/w/Export_Info_and_Tips

DESCRIPTION

Processing can export Java Applications for the Linux, Macintosh, and Windows platforms. When the «Export Application» button is pressed or «Export Application» is selected from the «File» menu, a dialog box opens and you can select which platforms you want to export to. You may also select if you want the application to run full screen (in present mode). A folder will be created for each of the operating systems selected; each folder contains the application, the source code for the sketch, and all required libraries for a specific platform. The «application.xxxx» folders will be removed completely on export.
It is important that you don't have a method named main() in your sketch, unless you know what you're doing (writing your own main). Otherwise this will fool the preprocessor into thinking you have a clue, when in fact you don't.
If running in «Java» mode, where your code starts «public class blah extends PApplet», you'll need to write your own main() method in order for Export to Application to work. It should look something like this:

```
static public void main(String args[]) {
    PApplet.main(new String[] { «YourClassName» });
}
```

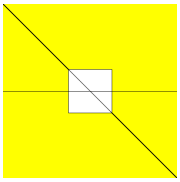
If you want to replace (or add) titlebar text, just do this in setup():

```
frame.setTitle(«This is in the titlebar!»);
```

The ESC key will quit a sketch, even in Present mode. To prevent this from happening, intercept the ESC on keyPressed() so that it isn't passed through to PApplet. Use the following code to prevent ESC from quitting the application:

```
void keyPressed() {
  if (key == ESC) {
    key = 0;  // Fools! don't let them escape!
  }
}
```

CODE



```
// Resizable Window
void setup() {
  size(400, 400);  // size always goes first!
  background(255,255,0);
  if (frame != null) { frame.setResizable(true); }
}

void draw() {
  rectMode(CENTER);
  rect(width/2, height/2, 100,100);
  line(0,0, width, height);
  line (0,height/2, 400, height/2);
}

// Fullscreen Mode
boolean sketchFullScreen() { return true; }

void setup() {
  size(displayWidth, displayHeight);  // uses the monitor's resolution
  background(255,255,0);
}

void draw() {
  rectMode(CENTER);
  rect(width/2, height/2, 100,100);
  line(0,0, width, height);
  line (0,height/2, 400, height/2);
}
```

Numeric Data int, boolean, float

http://processing.org/reference/boolean.html
http://processing.org/reference/byte.html
http://processing.org/reference/int.html
http://processing.org/reference/float.html

SYNTAX

```
boolean var
boolean var = booleanvalue
byte var
byte var = bytevalue
int var
int var = integervalue
float var
float var = floatvalue
```

DESCRIPTION

boolean
Datatype for the Boolean values true and false. It is common to use boolean values with control statements to determine the flow of a program.

byte
Datatype for bytes, 8 bits of information storing numerical values from 127 to -128. Bytes are a convenient datatype for sending information to and from the serial port and for representing letters in a simpler format than the char datatype.

int
Datatype for integers, numbers without a decimal point. Integers can be as large as 2,147,483,647 and as low as -2,147,483,648. They are stored as 32 bits of information.

float
Data type for floating-point numbers, a number that has a decimal point. Floats are not precise, so avoid adding small values (such as 0.0001) may not always increment because of rounding error. If you want to increment a value in small intervals, use an int, and divide by a float value before using it.

PARAMETRES

var	variable name referencing the float
booleanvalue	true or false
byte value	a number between 127 to -128
integervalue	any integer value
floatvalue	any floating-point value

CODE

```
0
10
20
21
1
0
```

```
// data calculations

int x = 0;
println(x);
x = x + 10;
println(x);
x += 10;
println(x);
x++;
println(x);
x -=20;
println(x);
x--;
println(x);
```

Character, Strings and Tracking Values char, print(), println()

http://processing.org/reference/char.html
http://processing.org/reference/String.html
http://processing.org/reference/print_.html
http://processing.org/reference/println_.html

SYNTAX

```
char var
char var = value
print(what)
println()
println(what)
```

DESCRIPTION

data
Datatype for characters, typographic symbols such as A, d, and $. A char stores letters and symbols in the Unicode format, a coding system developed to support a variety of world languages. Each char is two bytes (16 bits) in length and is distinguished by surrounding it with single quotes. Character escapes may also stored as a char. For example, the representation for the «delete» key is 127.

String
A string is a sequence of characters. The class String includes methods for examining individual characters, comparing strings, searching strings, extracting parts of strings, and for converting an entire string uppercase and lowercase. Strings are always defined inside double quotes («Abc»), and characters are always defined inside single quotes ('A'). To compare the contents of two Strings, use the equals() method, as in «if (a.equals(b))», instead of «if (a == b)».

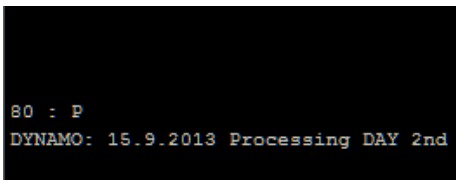
print()
Writes to the console area of the Processing environment. This is often helpful for looking at the data a program is producing. The companion function println() works like print(), but creates a new line of text for each call to the function. Individual elements can be separated with quotes («») and joined with the addition operator (+).

println()
This is often helpful for looking at the data a program is producing. Each call to this function creates a new line of output.

PARAMETRES

var	variable name referencing the value
value	any character
what	byte: boolean, byte, char, color, int, float, String, Object

CODE




```
// Character Data and Tracking Values
int i = 80;
char c = char(i);
int d = day();    // Values from 1 - 31
int m = month();  // Values from 1 - 12
int y = year();   // 2003, 2004, 2005, etc.
String value = «DYNAMO project-space»;

println(i + « : « + c);  // Prints «65 : A»
print(value);
print(«: « + d + «.» + m + «.» + y + «.»);
print(« Processing DAY 2nd»);
```

Timer millis, if, else

http://processing.org/reference/boolean.html
http://processing.org/reference/byte.html
http://processing.org/reference/int.html
http://processing.org/reference/float.html

SYNTAX

```
if (expression) {
  statements
} else if (expression) {
  statements
} else {
  statements
}
```

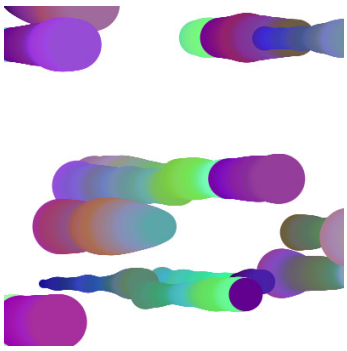
DESCRIPTION

millis()
Returns the number of milliseconds (thousandths of a second) since starting the program. This information is often used for timing events and animation sequences.

if
Allows the program to make a decision about which code to execute. If the test evaluates to true, the statements enclosed within the block are executed and if the test evaluates to false the statements are not executed.

else
Extends the if structure allowing the program to choose between two or more block of code. It specifies a block of code to execute when the expression in if is false.

CODE



```
float x, y, r, g, b, radius;
int timer;

void setup() {
  size(500, 500);
  background(255);
  noStroke();
  smooth();
}

void draw() {
  x = frameCount % width; // move x, use modulo to keep it within bounds
   if (millis() - timer >= 2000) {  // change the y every 2 seconds
    y = random(height);
    timer = millis();
  }

  // use frameCount and noise to change the red color component
  r = noise(frameCount * 0.01) * 255;
  // use frameCount and modulo to change the green color component
  g = frameCount % 255;
  // use frameCount and noise to change the blue color component
  b = 255 - noise(1 + frameCount * 0.025) * 255;
  // use frameCount and noise to change the radius
  radius = noise(frameCount * 0.01) * 100;
  color c = color(r, g, b);
  fill(c);
  ellipse(x, y, radius, radius);
}
```

frameRate frameRate(), frameRate, frameCount

http://processing.org/reference/char.html
http://processing.org/reference/String.html
http://processing.org/reference/print_.html
http://processing.org/reference/println_.html

SYNTAX

frameRate(fps)

DESCRIPTION

frameRate()
Specifies the number of frames to be displayed every second. For example, the function call frameRate(30) will attempt to refresh 30 times a second. If the processor is not fast enough to maintain the specified rate, the frame rate will not be achieved. Setting the frame rate within setup() is recommended. The default rate is 60 frames per second.

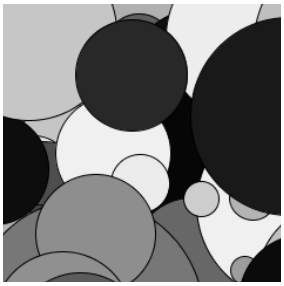
frameRate
The system variable frameRate contains the approximate frame rate of a running sketch. The initial value is 10 fps and is updated with each frame. The value is averaged over several frames, and so will only be accurate after the draw function has run 5-10 times.

frameCount
The system variable frameCount contains the number of frames that have been displayed since the program started. Inside setup() the value is 0, after the first iteration of draw it is 1, etc.

PARAMETRES

fps float: number of desired frames per second

CODE



```
float x, y, radius;

void setup() {
  size(500, 500);
  background(255);
  smooth();
  frameRate(200);
}

void draw() {
  x = random(width);
  y = random(height);
  radius = random(200);
  fill(random(255));
  ellipse(x, y, radius, radius);
  frame.setTitle(int(frameRate) + « fps»);
}
```

Create a 2D pattern for

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

SYNTAX

```
for (init; test; update) {
  statements
}

for (datatype element : array) {
  statements
}
```

DESCRIPTION

Controls a sequence of repetitions. A basic for structure has three parts: init, test, and update. Each part must be separated by a semicolon (;). The loop continues until the test evaluates to false. When a for structure is executed, the following sequence of events occurs:

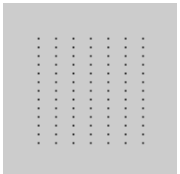
1. The init statement is run.
2. The test is evaluated to be true or false.
3. If the test is true, jump to step 4. If the test is false, jump to step 6.
4. Run the statements within the block.
5. Run the update statement and jump to step 2.
6. Exit the loop.

In the first example above, the for structure is executed 40 times. In the init statement, the value i is created and set to zero. i is less than 40, so the test evaluates as true. At the end of each loop, i is incremented by one. On the 41st execution, the test is evaluated as false, because i is then equal to 40, so i < 40 is no longer true. Thus, the loop exits.

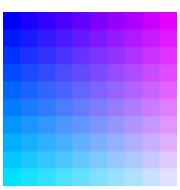
PARAMETRES

init	statement executed once when beginning loop
test	if the test evaluates to true, the statements execute
update	executes at the end of each iteration
statements	collection of statements executed each time through the loop
datatype	datatype of elements in the array
element	temporary name to use for each element of the array
array	name of the array to iterate through

CODE



```
// Draw a 10X10 point pattern
for (int i = 20; i <= 80; i = i+10) {
  for (int j = 20; j <= 80; j = j+5) {
    point(i, j);
  }
}
```



```
// Draw a 10X10 rect pattern
for (int i = 0; i < 10; i++) {
  for (int j = 0; j < 10; j++) {
    noStroke();
    fill(i*25,j*25,255);
    rect(i*10, j*10, 10, 10);
  }
}
```

Control Position translate, pushMatrix, popMatrix

http://processing.org/reference/translate_.html
http://processing.org/reference/popMatrix_.html
http://processing.org/reference/pushMatrix_.html

SYNTAX

```
translate(30, 20);
rect(0, 0, 55, 55);
```

DESCRIPTION

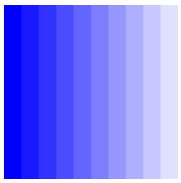
Specifies an amount to displace objects within the display window. The x parameter specifies left/right translation, the y parameter specifies up/down translation, and the z parameter specifies translations toward/away from the screen. Using this function with the z parameter requires using P3D as a parameter in combination with size as shown in the above example.

Transformations are cumulative and apply to everything that happens after and subsequent calls to the function accumulates the effect. For example, calling translate(50, 0) and then translate(20, 0) is the same as translate(70, 0). If translate() is called within draw(), the transformation is reset when the loop begins again. This function can be further controlled by using pushMatrix() and popMatrix().

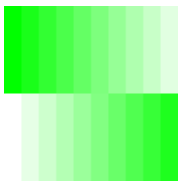
PARAMETRES

x	float: left/right translation
y	float: up/down translation
z	float: forward/backward translation

CODE



```
// Draw 10 height-size rects in series

for (int i = 0; i < 10; i++) {
  noStroke();
  fill(i*25, i*25, 255);
  rect(0, 0, 10, 100);
  translate(10, 0);
}
```



```
// Draw 2 series of 10 half height-size rects in series

pushMatrix(); // saves the current the coordinate system
for (int i = 0; i < 10; i++) {
  noStroke();
  fill(i*25, 255,i*25);
  rect(0, 0, 10, 50);
  translate(10, 0);
}
popMatrix(); // restores the prior coordinate system

pushMatrix();
for (int i = 0; i < 10; i++) {
  noStroke();
  fill(255-i*25, 255, 255-i*25);
  rect(0, 50, 10, 50);
  translate(10, 0);
}
popMatrix();
```

Rotate a shape rotate, for, translate

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

SYNTAX

rotate(angle)
rotate(angle, x, y, z)

DESCRIPTION

Rotates a shape the amount specified by the angle parameter. Angles should be specified in radians (values from 0 to TWO_PI) or converted to radians with the radians() function.

Objects are always rotated around their relative position to the origin and positive numbers rotate objects in a clockwise direction. Transformations apply to everything that happens after and subsequent calls to the function accumulates the effect. For example, calling rotate(HALF_PI) and then rotate(HALF_PI) is the same as rotate(PI). All tranformations are reset when draw() begins again.

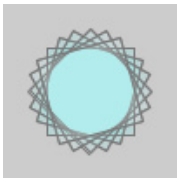
PARAMETRES

angle float: angle of rotation specified in radians using PI (180°)

RELATED

popMatrix()
pushMatrix()
rotateX()
rotateY()
rotateZ()

CODE



```
// Translate origin to center
translate(width/2,height/2);

for (int i=0; i<12; i++) {
  rectMode(CENTER);
  stroke(120);
  fill(i*25, 255, 255, 20);
  rect(0, 0, 50, 50);
  rotate(PI/12);
}
```

Scale a shape scale, for, translate

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

SYNTAX

scale(s)
scale(x, y)
scale(x, y, z)

DESCRIPTION

Increases or decreases the size of a shape by expanding and contracting vertices. Objects always scale from their relative origin to the coordinate system. Scale values are specified as decimal percentages. For example, the function call scale(2.0) increases the dimension of a shape by 200%.

Transformations apply to everything that happens after and subsequent calls to the function multiply the effect. For example, calling scale(2.0) and then scale(1.5) is the same as scale(3.0). If scale() is called within draw(), the transformation is reset when the loop begins again. Using this fuction with the z parameter requires using P3D as a parameter for size(), as shown in the third example above. This function can be further controlled with pushMatrix() and popMatrix().

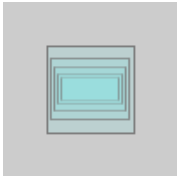
PARAMETRES

s	float: percentage to scale the object
x	float: percentage to scale the object in the x-axis
y	float: percentage to scale the object in the y-axis
z	float: percentage to scale the object in the z-axis

RELATED

translate()
popMatrix()
pushMatrix()

CODE



```
// Translate origin to center
translate(width/2,height/2);

for (int i=0; i<5; i++) {
  rectMode(CENTER);
  stroke(120);
  fill(i*25, 255, 255, 20);
  rect(0, 0, 50, 50);
  scale(0.9,0.7);
}
```

Mouse and Keyboard mousePressed, keyPressed

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

DESCRIPTION

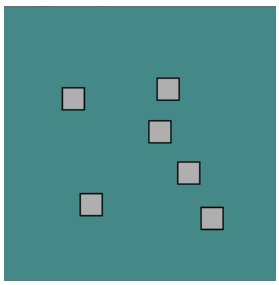
mousePressed()
The mousePressed() function is called once after every time a mouse button is pressed. The mouseButton variable (see the related reference entry) can be used to determine which button has been pressed.

keyPressed
The keyPressed() function is called once every time a key is pressed. The key that was pressed is stored in the key variable. For non-ASCII keys, use the keyCode variable. The keys included in the ASCII specification (BACKSPACE, TAB, ENTER, RETURN, ESC, and DELETE) do not require checking to see if they key is coded, and you should simply use the key variable instead of keyCode If you're making cross-platform projects, note that the ENTER key is commonly used on PCs and Unix and the RETURN key is used instead on Macintosh. Check for both ENTER and RETURN to make sure your program will work for all platforms.

mouseX
The system variable mouseX always contains the current horizontal coordinate of the mouse.

mouseY
The system variable mouseY always contains the current vertical coordinate of the mouse.

CODE



```
// Learning Processing
// Daniel Shiffman
// http://www.learningprocessing.com

// Example 3-5: mousePressed and keyPressed
// Modified by Angelos Floros

void setup() {
  size(200, 200);
  background(255);
}

void draw() {
  // Nothing happens in draw() in this example!
}

void mousePressed() { // tracks mouseClicks
  stroke(0);
  fill(175);
  rectMode(CENTER);
  rect(mouseX, mouseY, 16, 16);
}

void keyPressed() { // tracks any keyboard action
  if (keyPressed) {
    println(key + " " + keyCode);
    background(keyCode, keyCode*2, keyCode*2);
  }
}
```

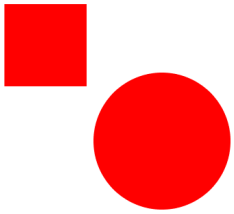
Mouse Over scale, for, translate

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

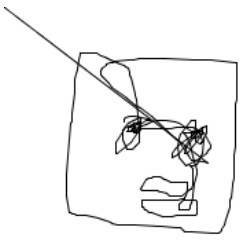
DESCRIPTION

pmouseX, pmouseY
The system variable pmouseX always contains the horizontal position of the mouse in the frame previous to the current frame. The system variable pmouseY always contains the vertical position of the mouse in the frame previous to the current frame.

CODE



```
void setup() {
  size(400, 400);
}

void draw() {
  background(255);

  int rX = 40;
  int rY = 60;
  int rW = 120;
  int rH = 120;
  if (mouseOverRect(rX, rY, rW, rH)) { fill(0, 0, 255); }
  else { fill(255, 0, 0); }
  rect(rX, rY, rW, rH);

  int cX = 270;
  int cY = 260;
  float cD = 200;
  if (mouseOverCircle(cX, cY, cD)) { fill(0, 255, 0); }
  else { fill(255, 0, 0); }
  ellipse(cX, cY, cD, cD);
}

boolean mouseOverCircle(int x, int y, float diameter) {
  return (dist(mouseX, mouseY, x, y) < diameter*0.5);
}

boolean mouseOverRect(int x, int y, int w, int h) {
  return (mouseX >= x && mouseX <= x+w && mouseY >= y &&
mouseY <= y+h);
}
```



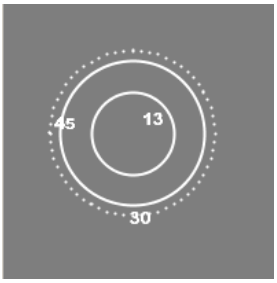
```
// Learning Processing
// Daniel Shiffman
// http://www.learningprocessing.com
// Example 3-4: Drawing a continuous line

void setup() {
  size(200, 200);
  background(255);
  smooth();
}

void draw() {
  stroke(0);
  // Draw a line from previous mouse location to current mouse location.
  line(pmouseX, pmouseY, mouseX, mouseY);
}
```

Clock

DESCRIPTION

Prints the time, using rotated text.

CODE




```
int cx, cy;
float secondsRadius, minutesRadius, hoursRadius, clockDiameter;

void setup() {
  size(200, 200);
  stroke(255);
  smooth();

  int radius = min(width, height) / 2;
  secondsRadius = radius * 0.60;
  minutesRadius = radius * 0.45;
  hoursRadius = radius * 0.15;
  clockDiameter = radius * 0.10;

  cx = width / 2-6;
  cy = height / 2-6;

  PFont metaBold;
  metaBold = loadFont("Arial-Black-48.vlw");
  textFont(metaBold,12);
}

void draw() {
  background(126);
  noFill();
  ellipse(cx, cy, minutesRadius*2+15, minutesRadius*2+15);
  ellipse(cx, cy, hoursRadius*2+30, hoursRadius*2+30);
  ellipse(cx, cy, clockDiameter - 10, clockDiameter - 10);

  // Angles for sin() and cos() start at 3 o'clock;
  // subtract HALF_PI to make them start at the top
  float s = map(second(), 0, 60, 0, TWO_PI) - HALF_PI;
  float m = map(minute() + norm(second(), 0, 60), 0, 60, 0, TWO_PI) - HALF_PI;
  float h = map(hour() + norm(minute(), 0, 60), 0, 24, 0, TWO_PI * 2) - HALF_PI;

  // Draw the hands of the clock
  stroke(255);
  text(hour(), cx + cos(h) * hoursRadius * .6,  cy + sin(h) * hoursRadius * .6);
  text(minute(),  cx + cos(m) * minutesRadius-12, cy + sin(m) * minutesRadius);
  text(second(),cx + cos(s) * secondsRadius-2,  cy + sin(s) * secondsRadius+6);

  // Draw the minute ticks
  strokeWeight(2);
   beginShape(POINTS);
  for (int a = 0; a < 360; a+=6) {
    float x = cx + cos(radians(a)) * secondsRadius;
    float y = cy + sin(radians(a)) * secondsRadius;
    vertex(x, y);
  }
  endShape();
}
```

Date, Time

DESCRIPTION

Prints the current Time and Date.

CODE


```
void setup() {
  size(200, 200);
  PFont metaBold;
  metaBold = loadFont("Arial-Black-48.vlw");
 textFont(metaBold,20);
}

void draw() {
  background(204);
  int d = day();       // Τιμές από 1 - 31
  int m = month();  // Τιμές από 1 - 12
  int y = year();      // 2003, 2004, 2005, κ.λ.π.
  int ho = hour();
  int mi = minute();
  int se = second();
  text(d + "/"+ m + "/"+ y,40,80);
  text(ho + ":" + mi + ":" + se,45,120);
}
```

Use mouse and keyboard to control shapes

DESCRIPTION

Use mouse and keyboard events to control shapes. Use the keyboard to create shapes and mouse to move , fill, transform, etc. Choose a desirable size window and export the application to fullscreen mode.

CODE