

# Processing 01

Environment

Drawing

Syntax

File handling

Variables

Loops

Conditionals

UIC/ART150 / Raaf

based on [processing.org](http://processing.org) and Casey Reas, Ben Fry: Processing: a programming handbook for visual designers and artists. MIT Press.

# What is Processing?

From <http://www.processing.org>:

Processing is an open source programming language and environment for people who want to program images, animation, and interactions.

It is used by students, artists, designers, researchers, and hobbyists for learning, prototyping, and production. It is created to teach fundamentals of computer programming within a visual context and to serve as a software sketchbook and professional production tool. Processing is developed by artists and designers as an alternative to proprietary software tools in the same domain.

The original authors of Processing are Casey Reas and Ben Fry, who both studied under John Maeda at MIT.

# References

<http://processing.org/reference>

There are many examples in the learning section of the processing website:

<http://processing.org/learning>

Take a look at the “libraries” section and see how the functionality of Processing can be expanded for specific interests and subjects.

<http://processing.org/reference/libraries/>

## **Other Galleries:**

studio.Sketchpad -<http://studio.sketchpad.cc/>

<http://studio.sketchpad.cc/sp/padlist/all-portfolio-sketches>

OpenProcessing.org

## **Using Processing online with Processing.js:**

<http://processingjs.org/tools/processing-helper.html>

<http://processingjs.org/reference/>

## **Using Processing online with P5.js:**

<https://p5js.org/examples/simulate-multiple-particle-systems.html>

<https://code.tutsplus.com/tutorials/introduction-to-p5js--cms-25665>

<https://p5js.org/reference/>

# Basic Tips and Tricks

**Highlight a word** in the Processing code and ctrl click on it to display information about it in the **Processing Reference**.

**Where can find my files?** Each sketch resides in its own folder. You can browse to this folder by choosing Sketch -> Show Sketch -> Folder from the Processing menu.

Read “Processing & Programming Handbook for Visual Designers and Artists” by Casey Reas and Ben Fry if you are interested in a thorough introduction to the software and its functions.

## **Extending Processing's functionality —> libraries**

Processing has become an extremely powerful scripting environment for the inclusion of almost any type of media – mainly through the concept of libraries. These libraries consist of subroutines and code that extend the functionality of Processing, often to include a particular kind of media (such as video, sound or 3D graphics) or functionality (communications, interface design, fullscreen playback, etc.).

To install a library, unzip the file into the “libraries” folder in your sketchbook (you can find the sketchbook in your home directory > Documents > Processing). If you do not already have a libraries folder in your sketchbook, create this folder manually and unzip the files in there. In general, most libraries have instructions on the website where you can download them from, following these directions is the easiest way to install a library

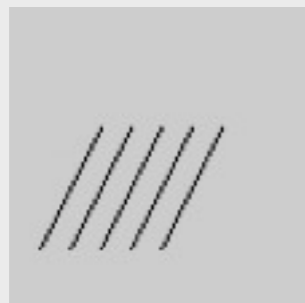
# Sketches

## All Processing projects are called sketches

Each sketch has its own folder (you can use the option in the menu Sketch/Show Sketch Folder to show the content of this folder)

Example sketch:

```
line(10, 80, 30, 40); // Left line
line(20, 80, 40, 40);
line(30, 80, 50, 40); // Middle line
line(40, 80, 60, 40);
line(50, 80, 70, 40); // Right line
```



# File handling and exporting

Remember: Each sketch has its own directory/folder where the main program file is located with the ending .pde

You can browse to this folder by choosing Sketch/Show Sketch folder from the Processing menu.

Example:

Sketch name: "My\_First\_Sketch\_01",

The directory for the sketch will be called "My\_First\_Sketch\_01"

The main file will be called "My\_First\_Sketch\_01.pde".

**EXPORTING TO WEB:** If you click on 'export' in the Processing menu bar, a subfolder called 'applet' will be created containing all files necessary to publish your work, including an index.html you can double-click to view in the Web browser.

You can reveal the content and location of your actual sketch folder by choosing 'Sketch/Show Sketch folder' in the Processing menu bar.

# Environment + Toolbar

The GUI consists of a text editor for writing code, message area, console, menu, toolbar, tabs for managing files

**Run:** Compiles the code, opens a display window, and runs the program inside.

**Stop:** Terminates a running program, but does not close the display window.

# Canvas size, Coordinates & Drawing in Processing

The size of the display window is controlled with the `size()` function:

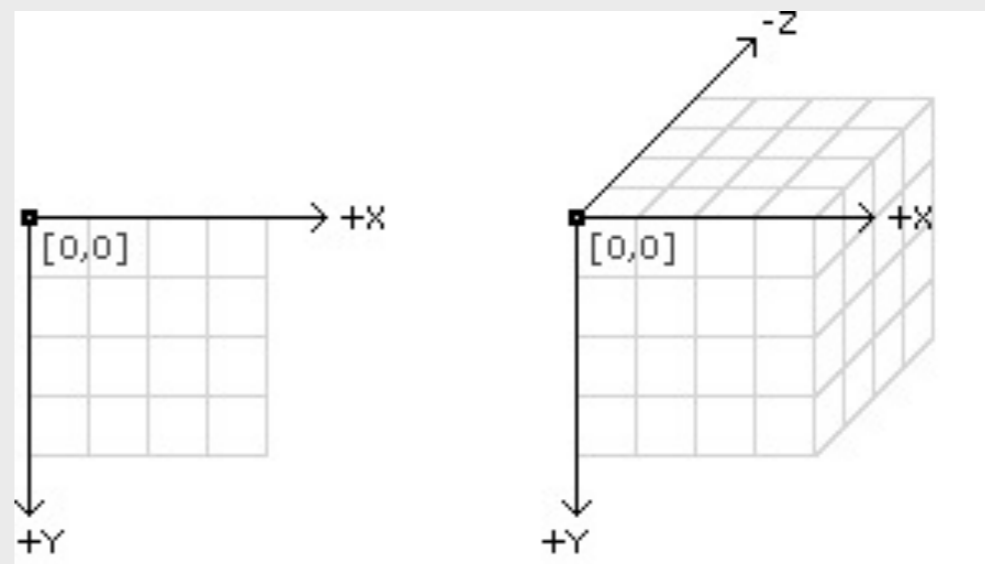
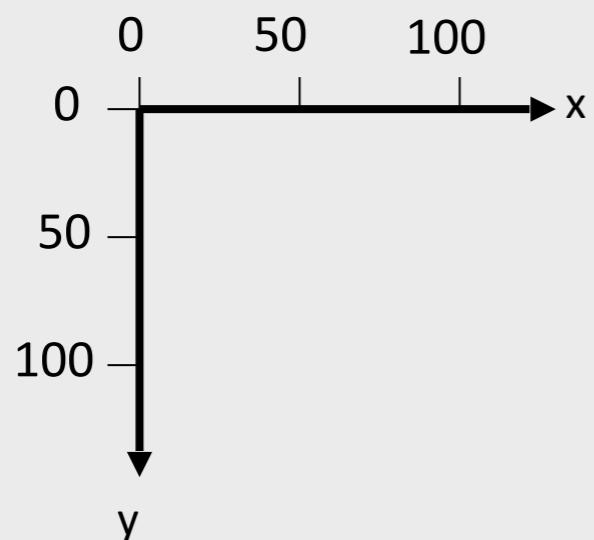
```
size(width, height)
```

The size function has two parameters: the first sets the width of the **CANVAS** (ie, **the window**) and the second sets its height.

A position on the screen is comprised of an x-coordinate and a y-coordinate. The x-coordinate is the horizontal distance from the origin and the y-coordinate is the vertical distance.

In a 400 pixel wide by 400 high window, `[0, 0]` is the upper-left pixel, `[320, 240]` is in the lower-right. The last visible pixel in the lower-right corner of the screen is at `[319, 239]`

## Coordinate system





# Basic Programming Mode - **STATIC**

```
size(200, 200);  
background(255);  
noStroke();  
fill(255, 204, 0);  
rect(30, 20, 50, 50);
```

# Color

<https://processing.org/tutorials/color/>

Colors represented as Red Green Blue (RGB) values

- Each one ranges from 0 to 255



# Drawing

Defining the canvas size: `size (300, 300); // 300 pixels wide, 300 pixels high`

Setting the background color:

```
background(0); //black
```

```
background(255); //white
```

```
background(255, 0, 0); //makes canvas red in rgb syntax
```

# Syntax

Common elements of the programming language and how they are used:

- Statement
- Statement Terminator -
- Function - somefunction()
- Parameter - values added to the function
- Comment - //

```
//example statements  
line(10, 80, 30, 40);  
i = i + 1;  
String txt = "Boots and Pants";  
print(txt);
```

note: Processing is case-sensitive: `String`, not `string`!

# Comments

- Comments are non-program text you put in the file to describe to others (and yourself) what you're doing
- Important for being able to look back at your code and understand it
- Single-line comments begin with `//`
- Multi-line comments begin with `/*` and end with `*/`

`Commenting and uncommenting lines useful for  
figuring out code`

# Drawing Graphic Primitives

```
size(200, 200);

//draws a point at x=10 and y=20
point(10, 10);

// draws a line from x1, y1 to x2, y2
line(10, 10, 100, 100);

// draws a rectangle over x1, y1, width, height
rect(10, 10, 180, 140);

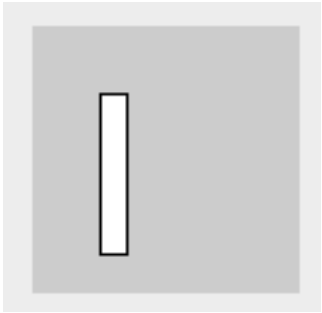
// draws an ellipse x, y, width, height
ellipse(120, 50, 40, 40);

// draws a four sided polygon x1, y1, x2, y2, x3, y3, x4, y4
quad(38, 31, 86, 20, 69, 63, 30, 76);

// draws a triangle x1, y1, x2, y2, x3, y3
triangle(120, 120, 80, 160, 160, 180);
```

# Drawing Shapes

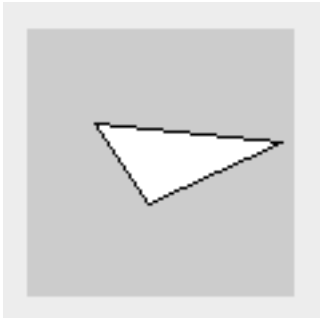
`rect(x, y, width, height)`



`rect(25, 25, 10, 60);`

# Drawing Shapes

```
triangle(x1, y1, x2, y2, x3, y3)
```

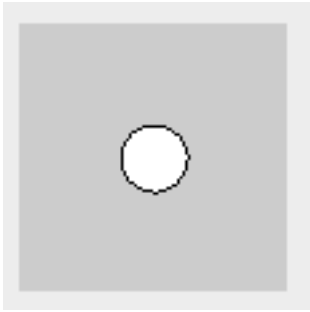


```
triangle(25, 35, 45, 65, 95, 42);
```



# Drawing Shapes

`ellipse(x, y, width, height)`



`ellipse(50, 50, 25, 25);`

# Defining Stroke and Fill Color

```
stroke(120); // gray stroke
stroke(0, 0, 255); // blue stroke
stroke(200, 80); // gray fill, transparent
noStroke(); // no Stroke
fill(100); // gray fill
fill(255, 0, 0); // red fill
fill(0, 255, 0, 127); // green fill semi transparent
noFill();
```

Values can be 00-2544- (which is 88 bits)

One number = grayscale value

Three numbers ==R,G,BB- or red, green, blue values

Four numbers = the fourth number is an alpha - ie,, just how transparent or opaque the fill is

# strokeWeight() and smooth()

```
background(0); // Sets the black background
stroke(255); // Sets line value to white
strokeWeight(5); // Sets line width to 5 pixels
smooth(); // Makes lines with smooth edges
line(10, 80, 30, 40); // Left line
line(20, 80, 40, 40);
line(30, 80, 50, 40); // Middle line
line(40, 80, 60, 40);
line(50, 80, 70, 40); // Right line
```

example:

```
rect(10, 10, 50, 50);
fill(204); // Light gray
rect(20, 20, 50, 50);
fill(153); // Middle gray
rect(30, 30, 50, 50);
fill(102); // Dark gray
rect(40, 40, 50, 50);
```

[https://processing.org/reference/rectMode\\_.html](https://processing.org/reference/rectMode_.html)

- `rectMode(CENTER)`
- `rectMode(CORNER) //default`
- `rectMode(CORNERS)`

Example code:

```
rectMode(CENTER);  
rect(35, 35, 50, 50);  
rectMode(CORNERS);  
fill(100);  
rect(35, 35, 50, 50);
```

# Continuous Mode

Adding more structure to a program opens further possibilities. The `setup()` and `draw()` functions make it possible for the program to run continuously – this is required to create animation and interactive programs.

```
void setup()
{
  size(200, 200);
  noStroke();
  background(255);
  fill(0, 102, 153, 20);
  smooth();
}

void draw()
{
  ellipse(mouseX, mouseY, 50, 50);
}
```

# Variables

- used to store values
- has a **name** (that you choose) & a **value** (choose a name that is descriptive)
- the name is Case-sensitive
- the name must not begin with special characters

```
String name = "ham"; // Declare and assign int
number = 32; // Declare and assign
int counter = 12; // Declare and assign
print(number);
print(name);
print(counter);
println(number); // println prints the line with a carriage return afterwards
println(name);
println(counter);
```

# Variable / Data Types

`int` //Integer: e.g. 1, 2, 3, ...

`float` //Floating point number: e.g. 0.1, 2.747, ...

`char` //Character: "\$", "A", stores one character.

`String` //String: e.g. "skinnyjeans", series of characters.

`boolean` //Boolean: true or false; 1 or 0.

```
int x;           // Declare the variable x of type int
float y;        // Declare the variable f of type float
boolean b;     // Declare the variable b of type boolean
x = 50;        // Assign the value 50 to x
y = 12.6;      // Assign the value 12.6 to f
b = true;      // Assign the value true to b
```

# Scope of Variables

## Local Variables

Declared inside a function.

Can only be used inside the function where it is declared.

```
void setup() {  
    int bgColor = 200;    // local variable  
    background(bgColor); // use local variable  
}
```

```
void draw() {  
    background(bgColor); //Error: cannot use bgColor here  
    line(0, 0, width, height);  
}
```



# Scope of Variables

## Global Variable

Declared outside the setup() and draw().  
Can be used anywhere in your sketch.

```
int bgColor = 200; // global variable

void setup() {
    background(bgColor); // use global variable
}

void draw() {
    background(bgColor); //use global variable
    line(0, 0, width, height);
}
```

# Scope of Variables

If a local variable is declared with the same name as a global variable, the program will use the local variable to make its calculations within the current scope.

```
int bgColor = 200;    // local variable

void setup() {
    int bgColor = 100;    // local variable
    background(bgColor); // use local variable
}

void draw() {
    background(bgColor); // use global variable
    line(0, 0, width, height);
}
```

# Mouse position

`mouseX`: X-Position the mouse within the applet

`mouseY`: Y-Position

```
line(mouseX, 20, mouseX, 80);
```

`mousePressed` returns true while mouse is pressed, false if not.

```
void draw() {  
    if (mousePressed == true) {  
        fill(0);  
    } else {  
        fill(255);  
    }  
    rect(25, 25, 50, 50);  
}
```

# Mouse Interaction

`mouseX, mouseY`

Stores the current position of the mouse inside the window

`pmouseX, pmouseY`

Stores the position of the mouse in the previous frame inside the window

`mousePressed, mousePressed()`

Used to detect if/when the mouse is being pressed (clicked)

`mouseButton`

Stores information about what button is being pressed

`mouseReleased()`

Called every time the mouse is released

`mouseDragged()`

Called every time when the mouse is dragged (pressed and moved)

`mouseMoved()`

Called every time when the mouse moves and not pressed

# Mouse Interaction

mouseX, mouseY

```
void setup()
{
  size(500, 200);
  strokeWeight(5);
  stroke(0, 100);
  smooth();
}

void draw()
{
  ellipse(mouseX, mouseY, 5, 5);
}
```

# Mouse Interaction I

mouseX, mouseY

pmouseX, pmouseY

pmouseX and pmouseY contains the previous horizontal and previous vertical coordinate of the mouse. It is the position of the mouse in the frame previous to the current frame.

This is very useful to **determine the velocity of a mouse movement or gesture**. By subtracting the previous from the current mouse position the current mouse velocity can be determined.

```
void draw()  
{  
  background(204);  
  line(mouseX, 20, pmouseX, 80);  
}
```

# Conditionals

Statements within the `if` section are only executed in case the condition (`i < 35`) is `true`; statements within the `else` section are executed only in case the condition is `false`.

```
void setup () {  
    size(300, 300);  
}
```

```
void draw() {  
    if(mouseX < 150) {  
        line( 0, mouseY, 150, mouseY );  
    }  
    else {  
        line( 150, mouseY, 300, mouseY );  
    }  
}
```

# Relational Operators

Used to compare values (conditionals):

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

```
5 > 4      // True
5 < 3      // False
5 > 5      // False
5 >= 5     // True
5 >= 6     // False
5 != 5     // False (not equal)
5 == 5     // True
5 == 4     // False
```



# Mouse Interaction II

## mousePressed

mousePressed is a system variable which is true if the button is pressed and false if the button is not pressed.

```
void setup() {
  size(200, 200);
  rectMode(CENTER);
  background(255);
  smooth();
  noStroke();
}

void draw() {
  if(mousePressed == true) {
    fill(random(255), 100);
  } else {
    fill(0);
  }
  rect(mouseX, mouseY, 30, 30);
}
```

# Mouse Interaction III

## mousePressed()

The mousePressed() system function is called every time the mouse button is pressed.

```
int fillColor = 0;
void draw() {
    fill(fillColor);
    rect(25, 25, 50, 50);
}
void mousePressed()
{
    if(fillColor == 0) {
        fillColor = 255;
    } else {
        fillColor = 0;
    }
}
```

# Loops: For and While

The `for()` loop uses defined conditions

```
for (int i=40; i<80; i=i+5) {  
    line(30, i, 80, i);  
}
```

The `while()` loop repeats as long as the condition is true

```
int i=0;  
while (i<80) {  
    line(30, i, 80, i);  
    i = i+5;  
}
```

note: if the test condition in the while loop cannot be false, the program freezes

# Using Libraries

Libraries allow you to extend Processing capabilities.

To use a library

1. Download it and extract it.
2. Find the folder that contains the library subfolder and put it inside the Processing libraries folder (put the parent folder, not the library folder directly).

Mac: Documents->Processing->libraries.

PC: libraries folder inside your Processing main folder.

Look for libraries at: <http://processing.org/reference/libraries/>

# Using Libraries

Interesting Libraries:

Sudden Motion Sensor:

Allows mac users to use the built-in accelerometer to control Processing sketches.

<http://www.shiffman.net/p5/sms/>

GUI libraries:

Control P5: <http://www.sojamo.de/libraries/controlP5/>

GUI Components: <http://www.lagers.org.uk/g4p/index.html>

Sprites for Processing:

Handles automatic motion of sprites and collision detection.

<http://www.lagers.org.uk/s4p/index.html>

# Using Libraries

Traer Physics

Particle Systems and simple physics for processing (no collision detection though)

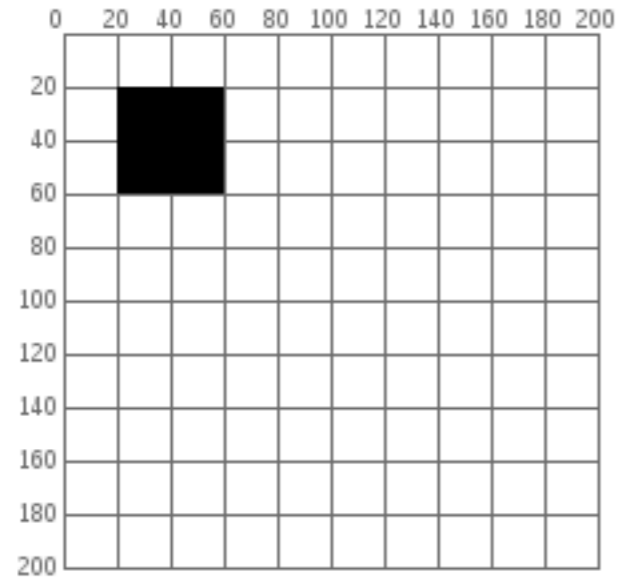
<http://www.cs.princeton.edu/~traer/physics/>

NextText

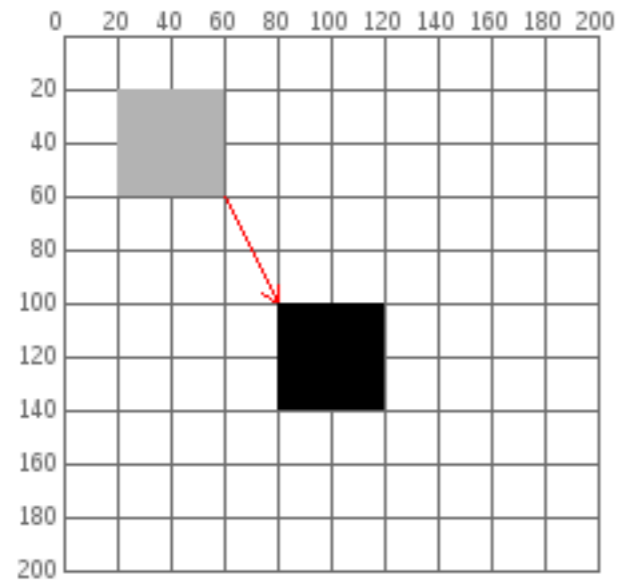
Auto text animations and control

<http://www.nexttext.net/>

# Transformations

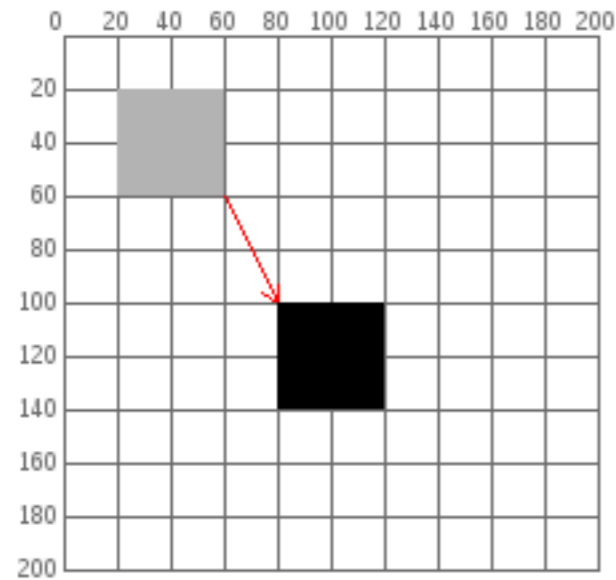


`rect(20, 20, 40, 40)`



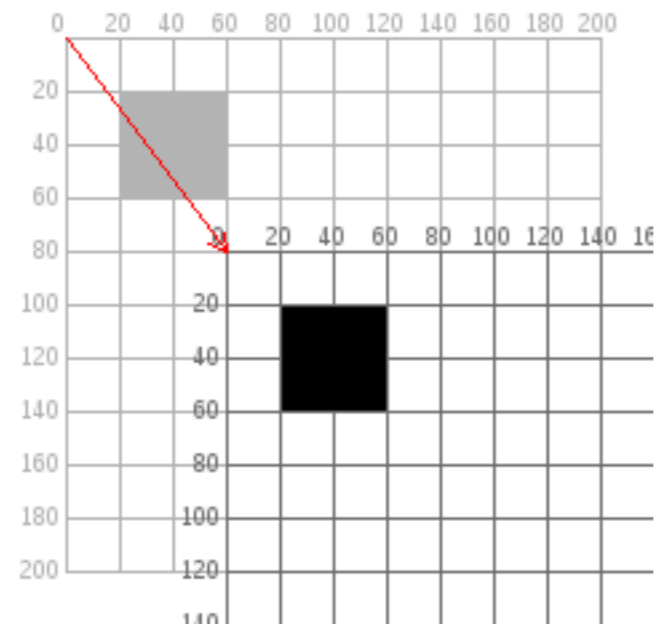
`rect(20 + 60, 20 + 80, 40, 40)`

# Translate



```
rect(20 + 60, 20 + 80, 40, 40)
```

Using **transformations**: we move the coordinate system instead of individual objects



```
translate(60, 80);  
rect(20, 20, 40, 40)
```



# Translate

Follow the mouse using translate.

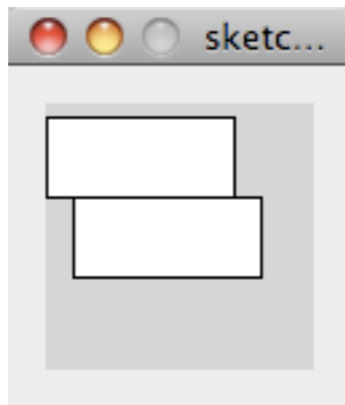
```
void setup()
{
  size(200, 200);
  noStroke();
  fill(255, 0, 0);
}

void draw()
{
  background(255);

  translate(mouseX, mouseY);
  ellipse(0, 0, 40, 40);
}
```

# Translate

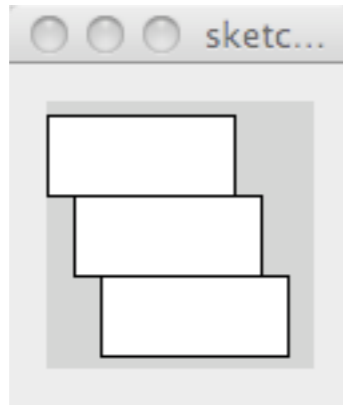
Only the second rect is affected by translate



```
rect(0, 5, 70, 30);  
translate(10, 30);  
rect(0, 5, 70, 30);
```

# Translate

Transformation accumulate (are additive)



```
rect(0, 5, 70, 30);  
translate(10, 30);  
rect(0, 5, 70, 30);  
translate(10, 30);  
rect(0, 5, 70, 30);
```

# Translate

```
void setup()
{
  size(200, 200);
  noStroke();
}

void draw()
{
  background(255);

  fill(255, 0, 0, 100);
  translate(mouseX, mouseY);
  ellipse(0, 0, 40, 40);

  // Note that this ellipse will move twice as fast as the previous,
  // because transformation accumulate
  fill(0, 255, 0, 100);
  translate(mouseX, mouseY);
  ellipse(0, 0, 40, 40);
}
```

# Translate

```
void setup()
{
  size(200, 200);
  background(255);
  noStroke();

  // draw the original position in gray
  fill(192);
  rect(20, 20, 40, 40);

  // draw a translucent red rectangle by changing the coordinates
  // passed to the rect function
  fill(255, 0, 0, 128);
  rect(20 + 60, 20 + 80, 40, 40);

  // draw a translucent blue rectangle by translating the grid
  fill(0, 0, 255, 128);
  translate(60, 80);
  rect(20, 20, 40, 40);
}
```

# Rotate

The rotate function rotates the coordinate system allowing you to draw shapes at an angle

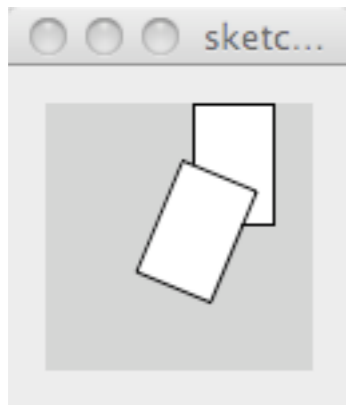
Rotations are specified in radians and in clockwise direction.

Rotations are also accumulated.

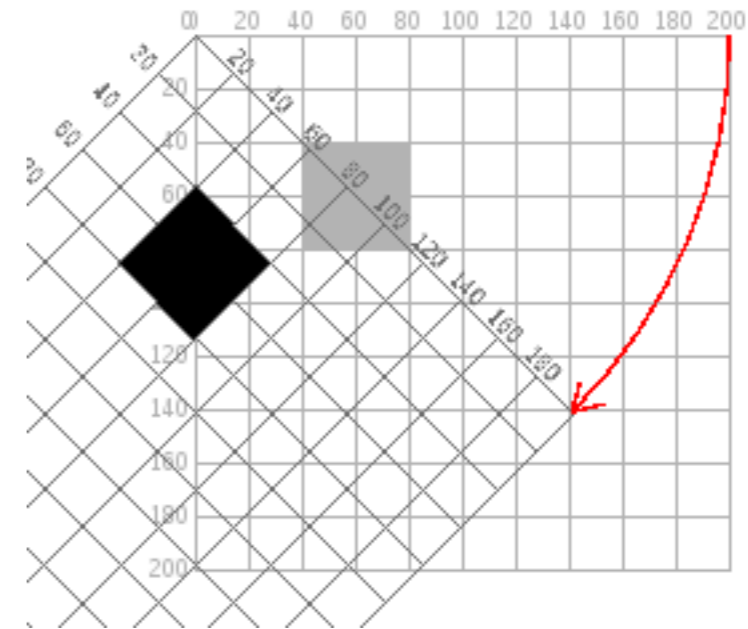
You can transform values to/from radians/degrees using the functions

```
radians(float value);
```

```
degrees(float value);
```



```
smooth();  
rect(55, 0, 30, 45);  
rotate(PI/8);  
rect(55, 0, 30, 45);
```



# Rotate

To rotate objects from their center, you need to combine translation + rotation.

Example with a square:

- A. Translate the coordinate system's origin (0, 0) to where you want the upper left of the square to be.
- B. Rotate the grid  $\text{PI}/4$  radians ( $45^\circ$ )
- C. Draw the square at the origin.

```
// Draw from the center
rectMode(CENTER);

// move the origin to the pivot point
translate(width/2, height/2);

// then pivot the grid
rotate(radians(45));

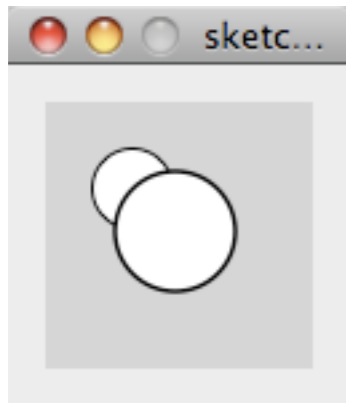
// and draw the square at the origin
fill(0);
rect(0, 0, 40, 40);
```

# Scale

The scale function scales the coordinate system allowing you to draw shapes at an different sizes

The scales are specified as percentages in decimal mode: 2.0 = 200%.

Be careful, as scaling also affects the position of objects that are not drawn at 0, 0.



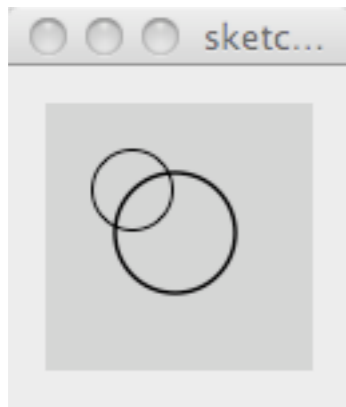
```
smooth();  
ellipse(32, 32, 30, 30);  
scale(1.5);  
ellipse(32, 32, 30, 30);
```



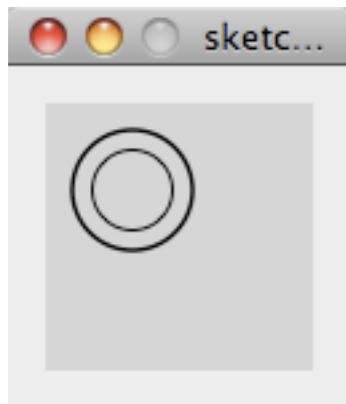
# Scale

Be careful, as scaling also affects the position of objects that are not drawn at 0, 0.

See in this example, how drawing the ellipses at (32, 32) vs drawing them at (0, 0) with a translation of (32, 32) makes a different outcome when scale is involved.



```
noFill();  
smooth();  
ellipse(32, 32, 30, 30);  
scale(1.5);  
ellipse(32, 32, 30, 30);
```



```
noFill();  
smooth();  
translate(32, 32);  
ellipse(0, 0, 30, 30);  
scale(1.5);  
ellipse(0, 0, 30, 30);
```

# Examples

Drawing a color wheel

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

void draw() {
  if (frameCount % 10 == 0) {
    fill(frameCount * 3 % 255, frameCount * 5 % 255, frameCount * 7 % 255);

    translate(100, 100);
    rotate(radians(frameCount * 2 % 360));

    rect(0, 0, 80, 20);
  }
}
```

# Examples

Using a for loop to accumulate transformations

```
size(200, 200);  
background(0);  
smooth();  
stroke(255, 100);  
  
translate(width/2, 80);  
for ( int i = 0; i < 18; i++ )  
{  
    strokeWeight(i);  
    rotate(PI/12);  
    line(0, 0, 55, 0);  
}
```