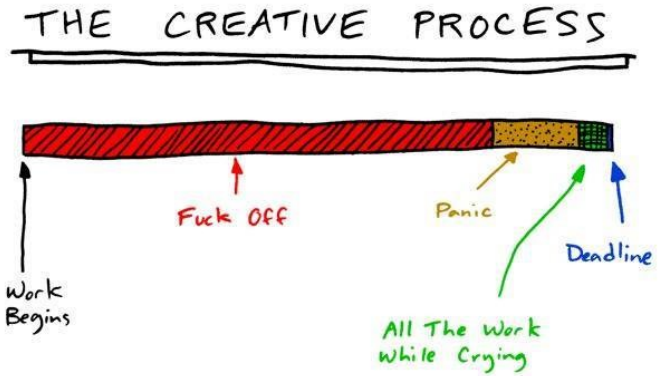
UNIVERSITÉ TOULOUSE III PAUL SABATIER

processing.org



http://www.irit.fr/~Philippe.Truillet

v. 2.2 – december 2011- november 2018

Brief history

Design by Numbers
http://dbn.media.mit.edu
Date: 1999-2001
Location: MIT Media Lab
John Maeda

Processing
http://www.processing.org
Date: Spring 2001
Location: MIT Media Lab
Ben Fry / Casey Reas

Processing 3

p5.js

Wiring
http://wiring.org.co
Date: 2003
Location: IDII
Hernando Barragán

Arduino
http://www.arduino.cc
Date: 2005
Location: IDII
Massimo Banzi

Genuino

ARDUINO

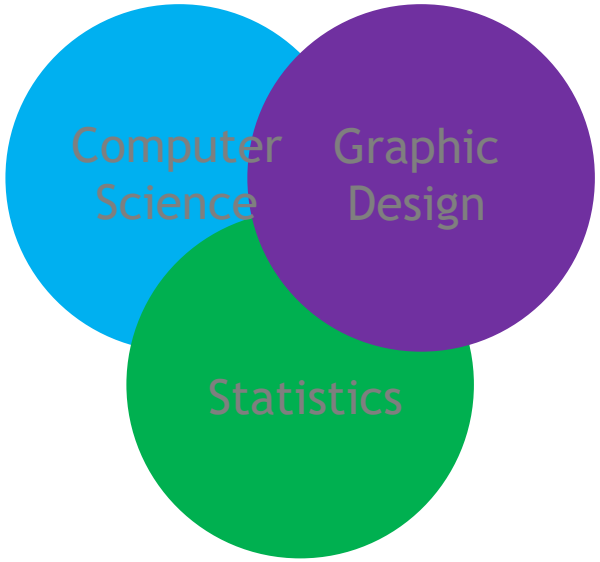
Visible Language Workshop
http://museum.mit.edu/150/115
Date: **1975**
Location: MIT
Muriel Cooper

What is Processing?

- Created in 2001 by B. Fry & C. Reas

- « *an electronic sketchbook for developing ideas* »,
  « *language that was created to develop visually oriented applications with an emphasis on animation and providing users with instant feedback through interaction* »

→ http://www.processing.org
→ http://processingjs.org
→ https://p5js.org



Currently: version 2.2.1 (19 May 2014)
3.4 (26 July 2018)

What is Processing?

Processing **IS** Java

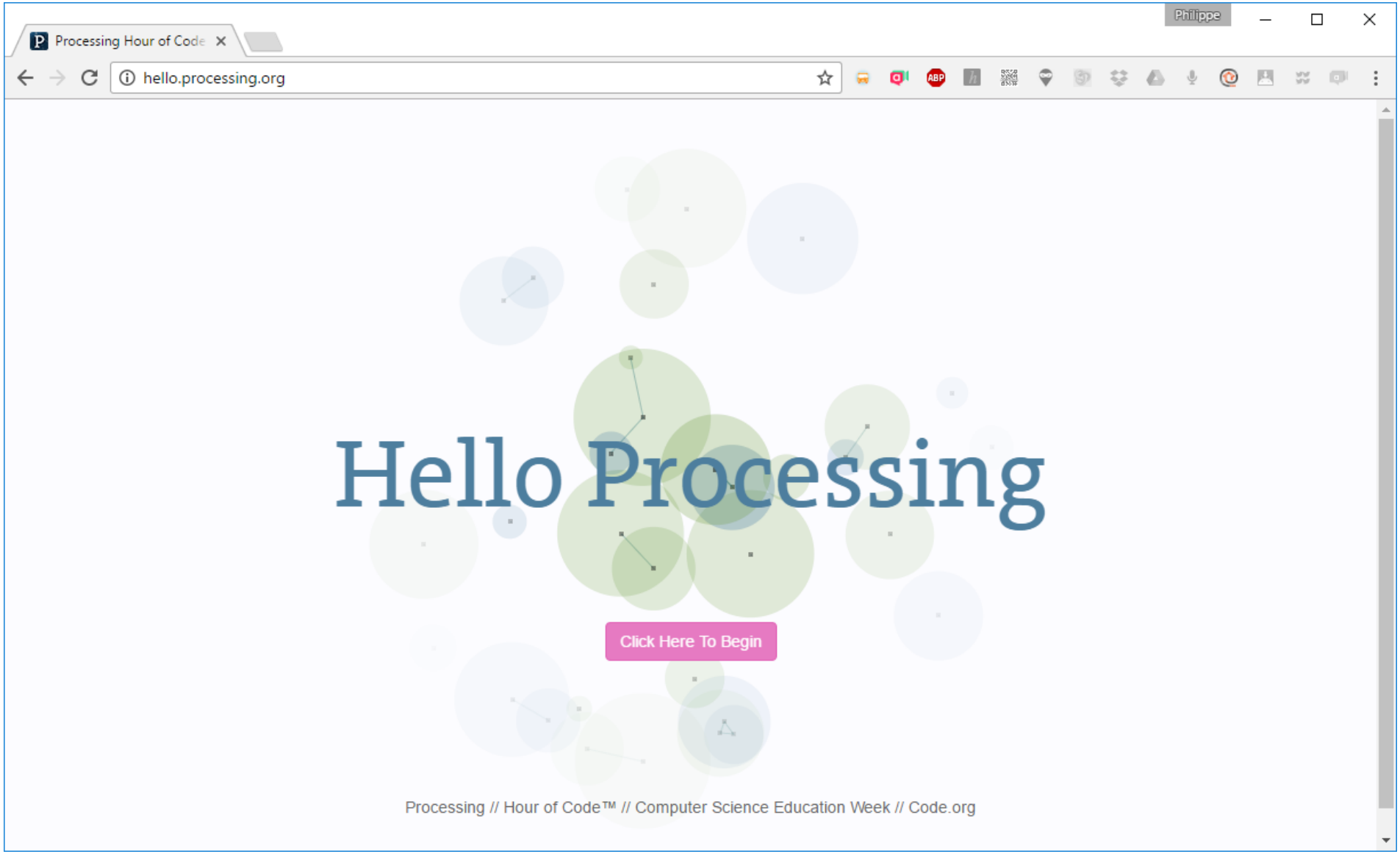
- With an other layout manager than AWT or swing
- Hence, we can use classes, threads, *external* jar files, …

Roots

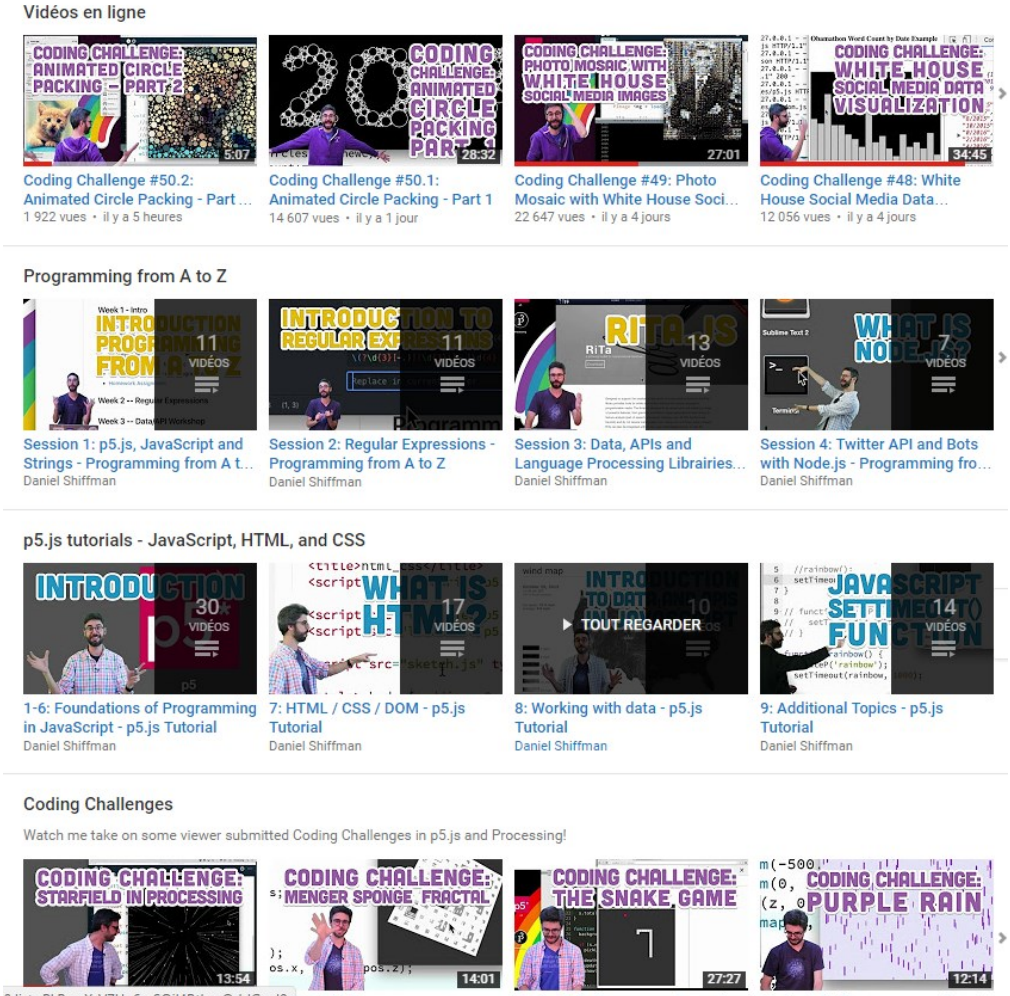
Originally built as a domain-specific targeted towards artists and designers, Processing has evolved into a full design and prototyping tool used for work, motion graphics and complex data visualization.

Processing is based on Java but elements are fairly simple and so, you can learn to use it even if you don't know Java.

Fundamentals of Processing carry over nicely of other languages


Philippe
Processing Hour of Code
hello.processing.org
Hello Processing
Click Here To Begin
Processing // Hour of Code™ // Computer Science Education Week // Code.org

https://www.youtube.com/shiffmanvideos



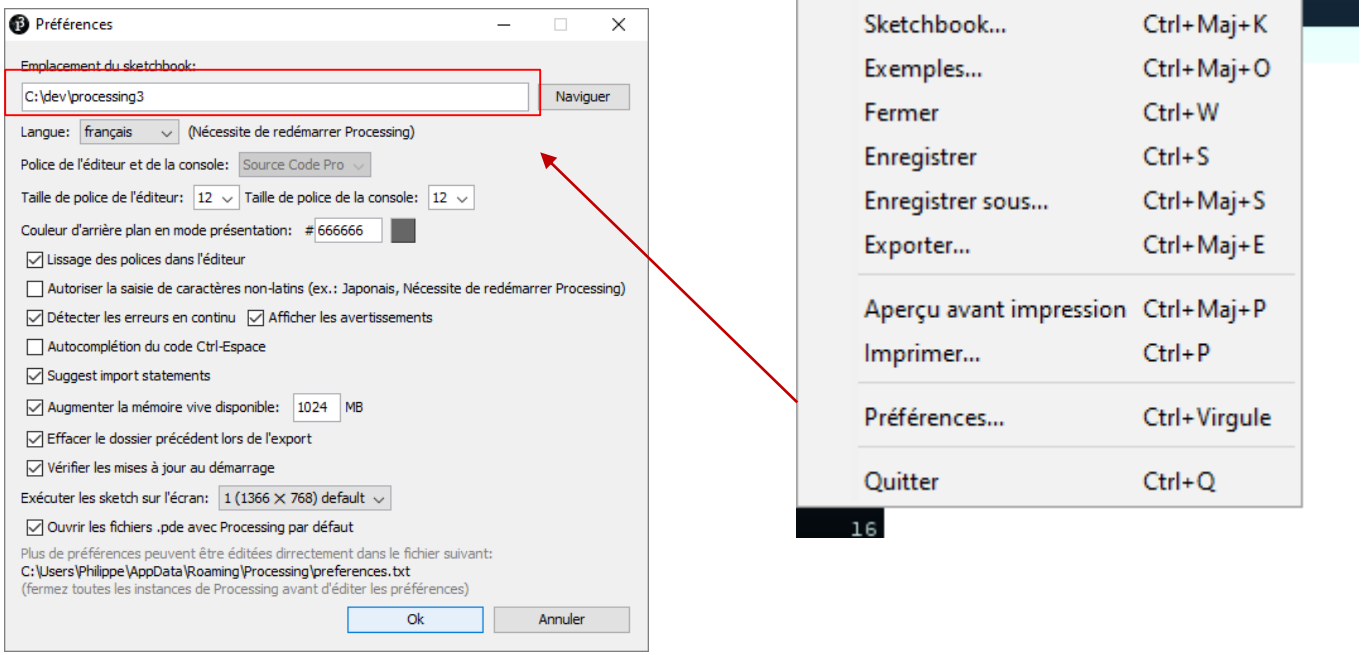
The language

- Open source environment
- Data visualization programming language
- Built on top of Java language

- Strong typing
- Polymorphism
- Classical inheritance

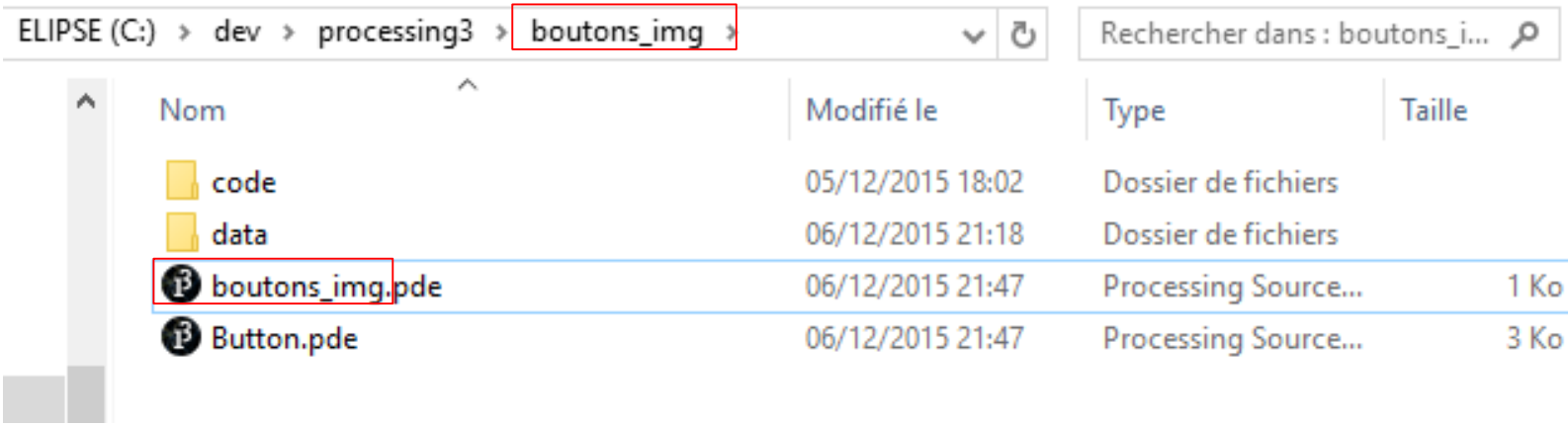
Structure

- « Sketches » (programs) are located in the « preferences » folder



Structure

- One sketch is composed of :
  - At least one « **.pde** » file (should be more – one per class). The main file must have the same name than the sketch folder
  - Possibly, other folders
    - « **data** » folder in which you can put images, sounds, etc. (**resources**)
    - « **code** » folder in which you can put jar files (**external lib**)

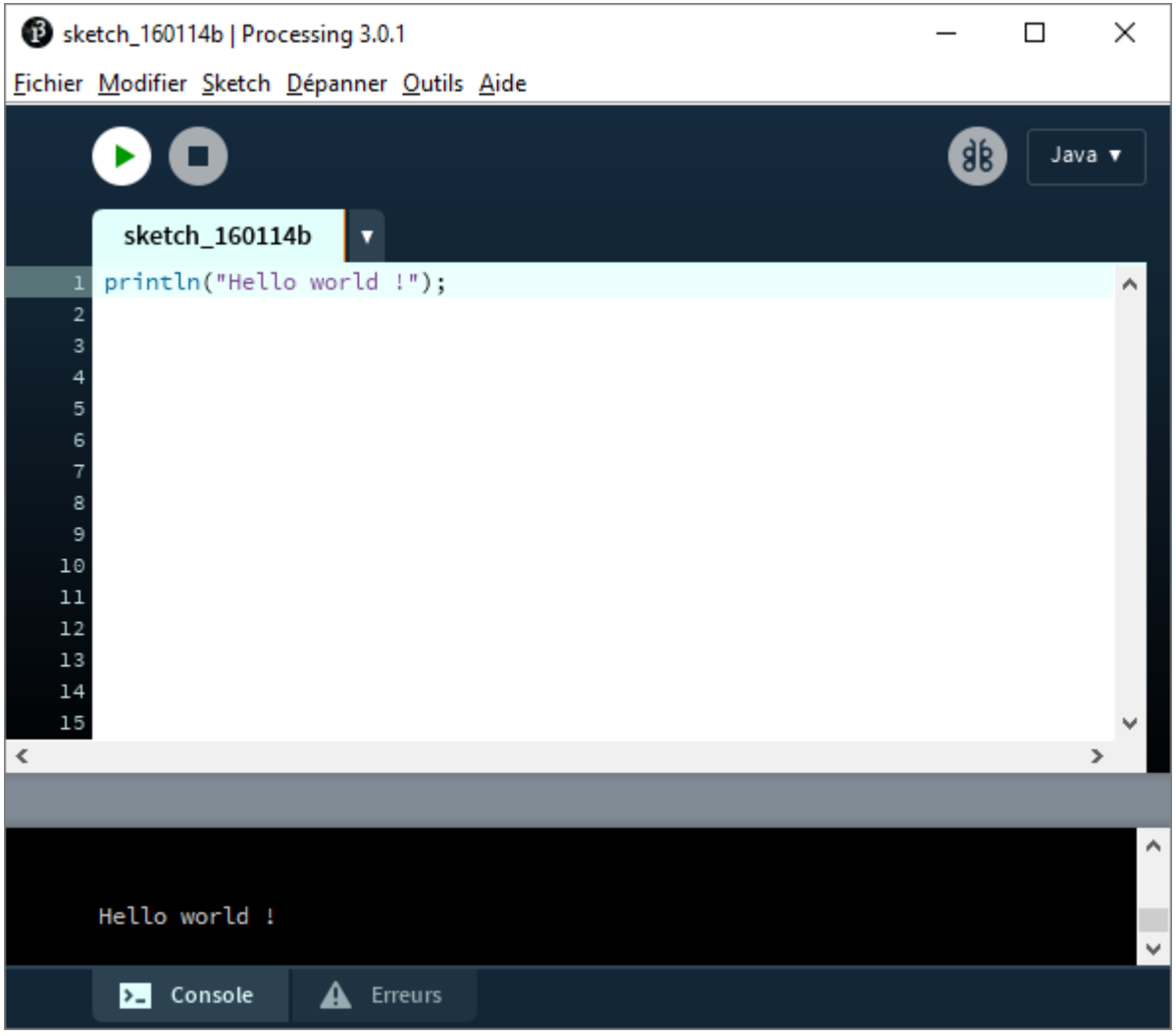


First example (Java)

```java
public class Hello
{
    public static void main (String args[])
    {
        System.out.println("Hello, world!");
    }
}
```

```
> javac Hello.java
> java Hello
> Hello, world!
```

First example (Processing)



Second example (bubbles)

bubbles | Processing 2.2.1

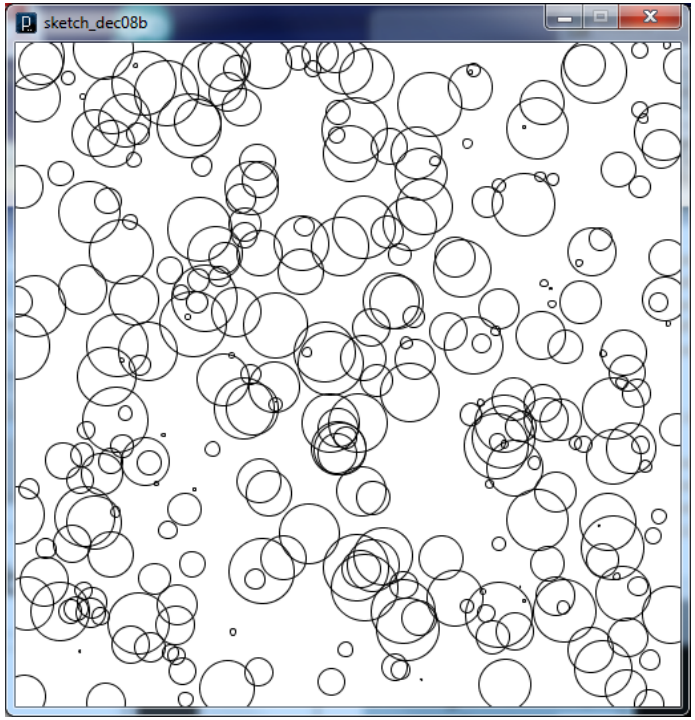
File Edit Sketch Tools Help

Java

bubbles

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

void draw()
{
  float x = random(width);
  float y = random(height);
  float diameter = random(50);
  ellipse(x,y, diameter, diameter);
}
```

14

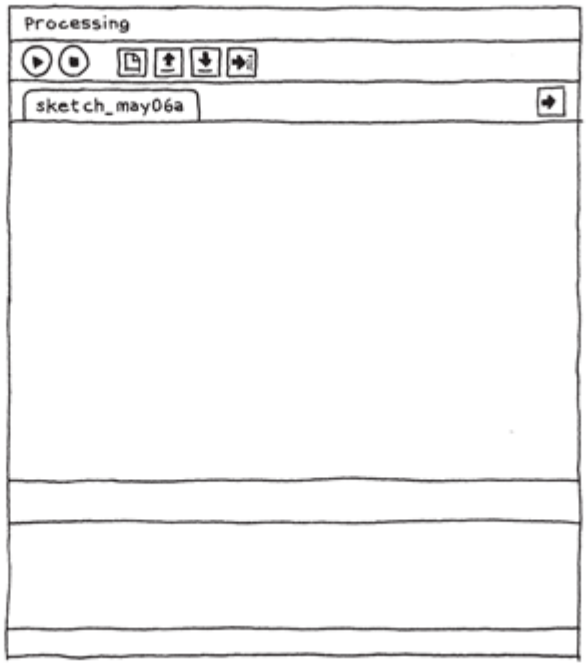


Download & install

- Download the IDE from *http://processing.org/download*

(stable version: 1.5.1, 2.2.1, 3.4)

Display window



Toolbar
Tabs
Text editor
Message area
Console

Stable Releases

3.0.1 (23 October 2015) Win 32 / Win 64 / Linux 32 / Linux 64 / Linux ARMv6hf / Mac OS X

3.0 (30 September 2015, 3p ET) Win 32 / Win 64 / Linux 32 / Linux 64 / Mac OS X

2.2.1 (19 May 2014) Win 32 / Win 64 / Linux 32 / Linux 64 / Mac OS X

1.5.1 (15 May 2011) Win (standard) / Win (no Java) / Linux x86 / Mac OS X

uncompress the archive
just start the executable!

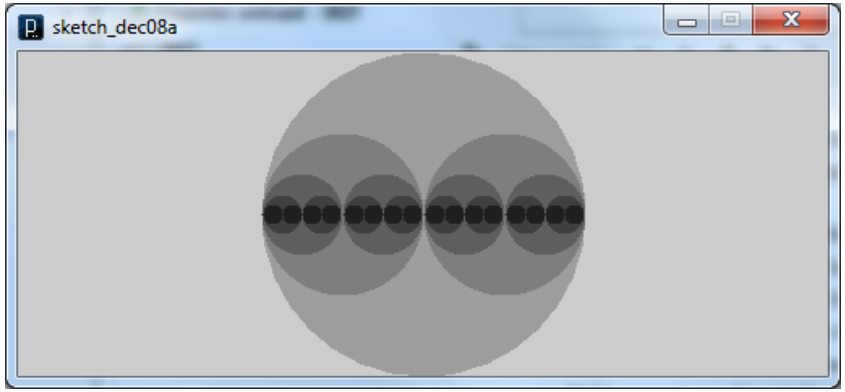
3.0.1 (23 October 2015)

Windows 64-bit
Windows 32-bit
Linux 64-bit
Linux 32-bit
Linux ARMv6hf
Mac OS X

http://processing.org/learning/gettingstarted

API

- Graphics
  - Procedural
  - Global scope
  - Graphics generation
  - Image processing




```
void setup()
{
  size(500,200);
  noStroke();
}

 void draw()
 {
    drawCircle(250, 100, 5);
 }

 void drawCircle(int x, int radius, int level)
 {
   float tt = 126 * level/4.0;
   fill(tt);
   ellipse(x, 100, radius*2, radius*2);
   if (level>1)
   {
     level = level -1;
     drawCircle(x-radius/2, radius/2, level);
     drawCircle(x+radius/2, radius/2, level);
   }
 }
```

Naming convention

- Legal names can only contain letters, underscores, numbers and $
- The initial character must be a letter, underscore or $ sign (not a number)
- These names are called **identifiers.**

Variables

- Container for storing data
- Allows data element to be reused
- Every variable has a name, value and data type
- A variable must be defined before it is used

Data types

- **byte**: small integer number
- **int**: integer number
- **long**: huge integer number
- **float**: floating-point numbers
- **double**: huge floating-point numbers

- **char**: one symbol in single quotes
- **String**: words
- **boolean**: true or false

- every structure you can define!

Operators

- **Mathematical operators**
  - +, -, *, /,%
- **Relational operators**

  >, >=, <, <=, ==, !=
- **Logical operators**

  && (and), || (or)

Comments

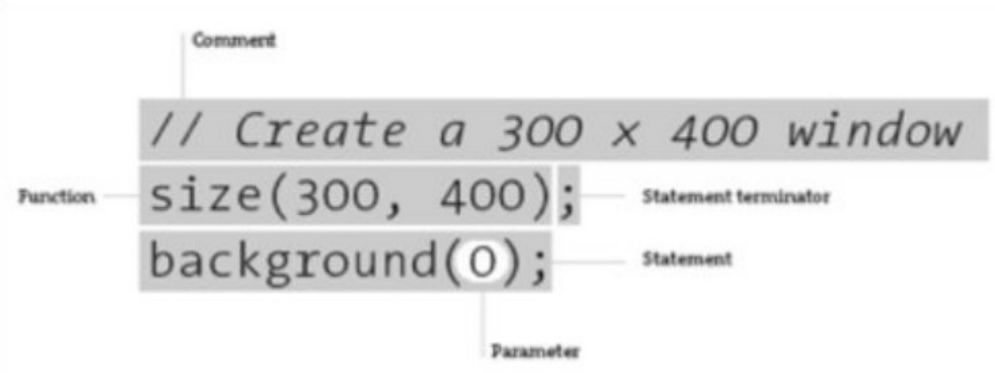
- Ignored by the computer
- Same as java or C++ comments

```
/*
    Something there

*/

// just one line
```

Expressions, statements

- An **expression** is like a phrase and often a combination of operators

- A **statement** is a set of expressions, like a sentence and ends with a statement terminator



Control structures

- **Iteration**: while

The *while* structure executes a series of statements while the expression is true

- **Iteration**: for

A *for* structure has three parts: init, test and update separated by a semi-colon « ; ». The loop continues until the test evaluates to false

Control structures

- **Conditional**: if

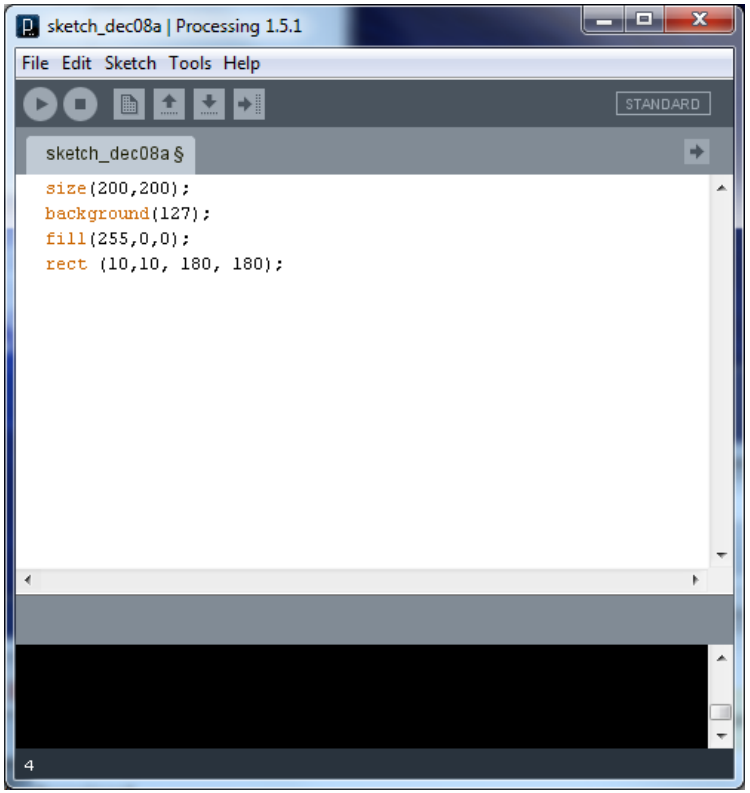
If the test evaluates to true, statements enclosed within the block are executed.

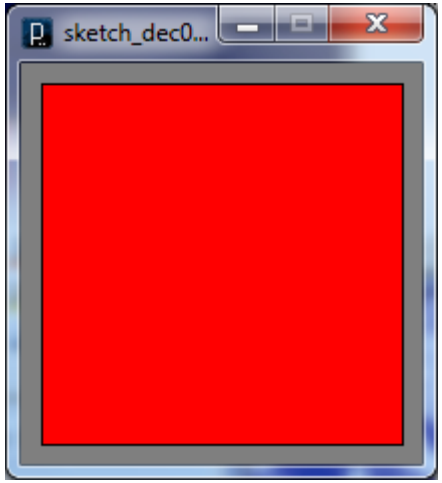
- **Conditional**: switch

switch is more convenient when you need to select three or more alternatives. Program jumps to the case with the same value as the expression. All remaining statements in the *switch* are executed unless redirected by a break.

Programming modes

- **Basic mode** (scripting mode): this mode is used for drawing static images and learning fundamentals. Lines of code have a direct representation on the screen





Programming modes

- **Continuous mode**: this mode provides a *setup()* function that is run once when the program begins and the *draw()* function which continually loops through the code inside.

  This allows writing custom functions and classes, using keyboard and mouse events.

Initialization : **setup()**

- size(w,h): sets the width/height of the drawing area

- Can call any other number of methods such as
  - background(c): draws and fills a background color

```
void setup()
{
  size(200,200);
  background(102);
}
```

Mainloop: **draw()**

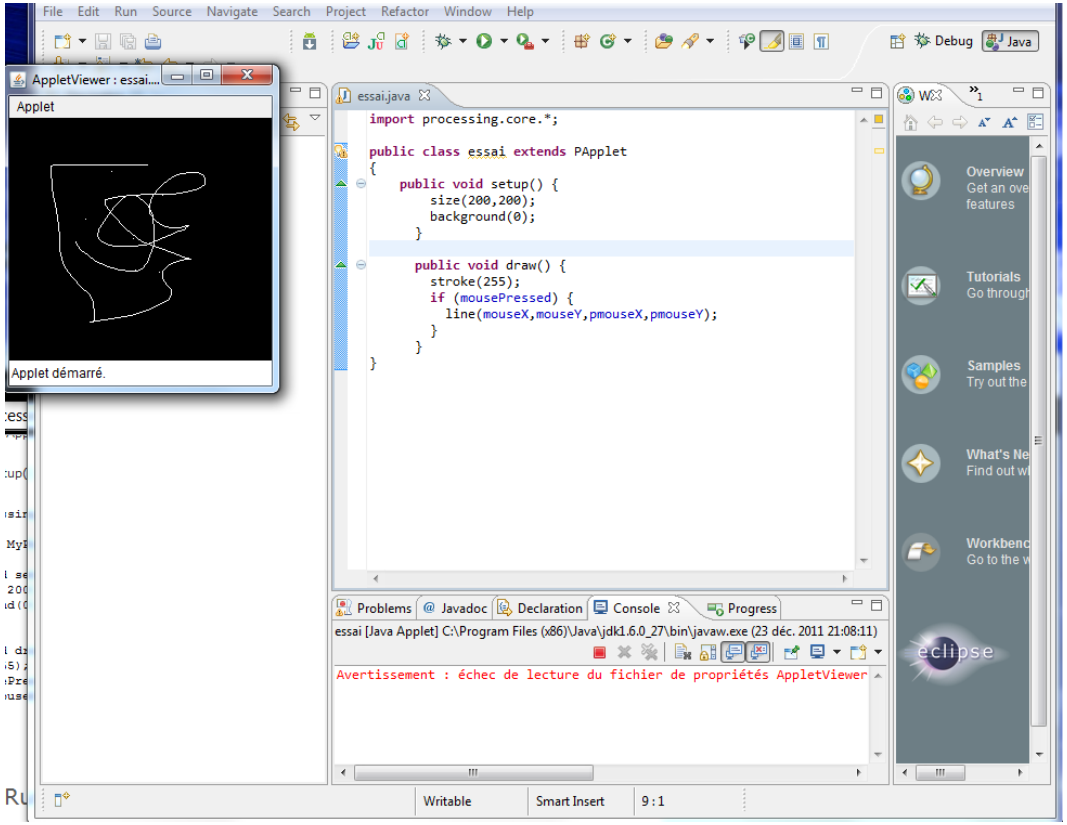
draw() gets called as fast as possible unless a framerate is specified (with FrameRate function)

- Can call any other number of methods such as
  - stroke(…) sets color of the outline drawing
  - fill(…) sets color of *inside* drawing
  - Events:
    - mousePressed is true if mouse is down
    - Mouse position: mouseX, mouseY
  - Different drawing methods : line(), rect(), arc(), ellipse(), point(), triangle(), bezier(), …

Programming modes

- « **java** » **mode**: can be used as a library in Eclipse – « **just** » (hum …) add `core.jar` and extend `Papplet`

→ http://processing.org/learning/eclipse/



#1 - Shapes

sketch_dec19b | Processing 2.0a4

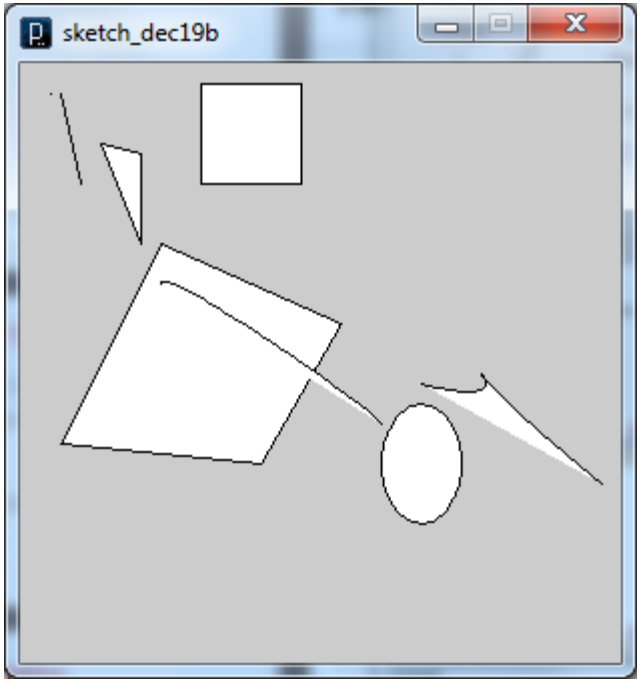
File Edit Sketch Tools Help

STANDARD

sketch_dec19b §

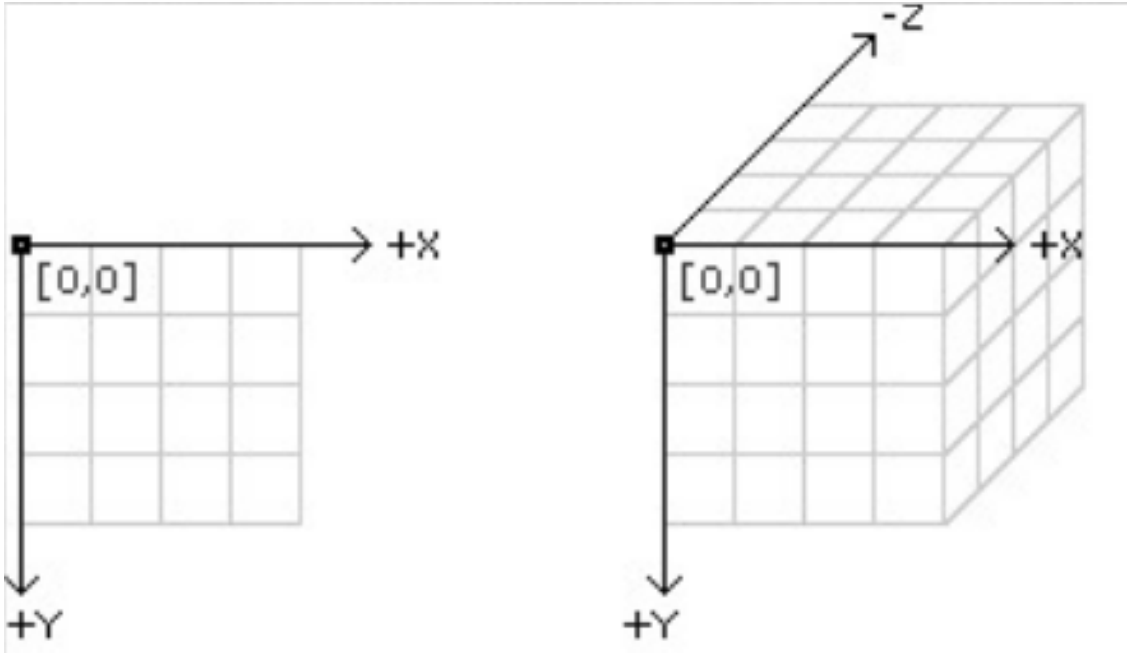
```
size(300,300);
point(15,15);
line(20,15, 30, 60);
triangle(40,40, 60, 90, 60,45);
quad(70,90,160,130, 120,200, 20, 190);
rect(90, 10, 50, 50);
ellipse(200,200, 40,60);
curve(100, 160, 180, 180, 70, 110, 190, 250);
bezier(200, 160, 280, 180, 170, 110, 290, 210);
```

9



#1: canvas

Processing uses a Cartesian coordinate system with this origin on the upper-left corner



The size of the display is ontrolled with the size() function

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

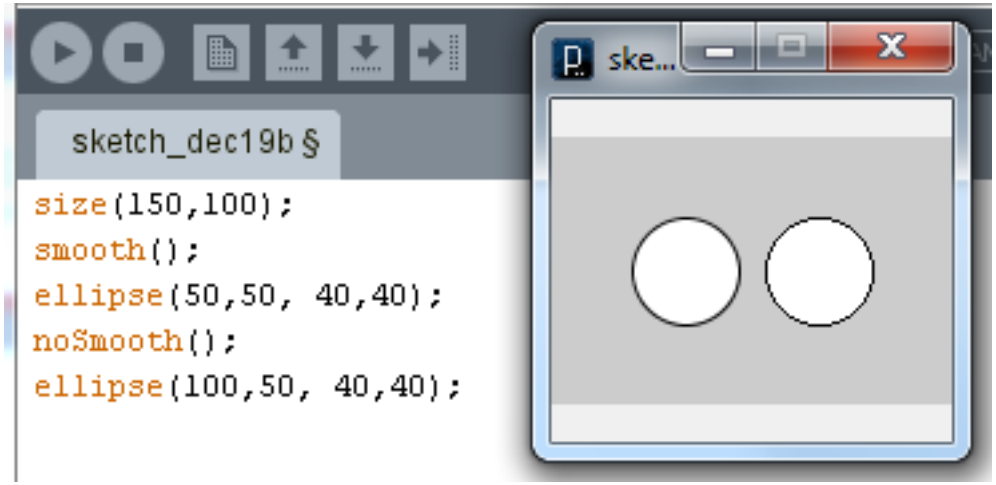
#1: (main) primitives shapes

- **Point**: point(x,y)
- **Line**: line(x1, y1, x2, y2)
- **Triangle**: triangle(x1, y1, x2, y2, x3, y3)
- **Quadrilateral**: quad(x1, y1, x2, y2, x3, y3, x4, y4)
- **Rectangle**: rect(x, y, width, height)
- **Circle & Ellipse**: ellipse(x, y, width, height)
- **Curve**: curve(x1, y1, x2, y2, x3, y3, x4, y4)
- **Bezier curve**: bezier(x1,y1, cx1,cy1, cx1, cy2,x2,y2)

The order in which shapes are drawn in the code defines which shape appears on top of others.

#1: drawing attributes

- smooth() and noSmooth(): enable and disable smoothing (aka anti-alisasing)

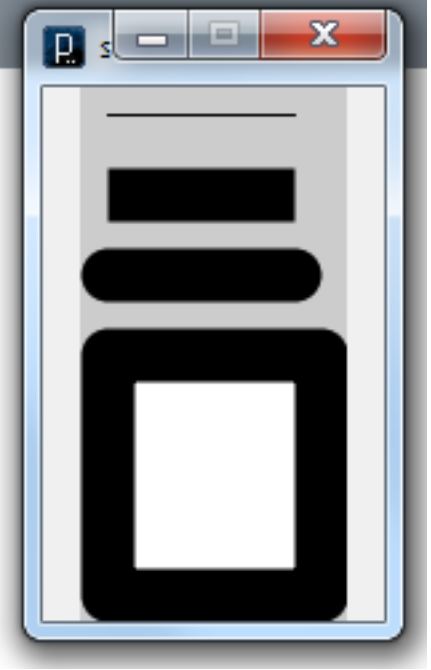


#1: drawing attributes

- strokeWeight(numeric_attr), strokeCap(ROUND | SQUARE | PROJECT), strokeJoin(BEVEL | MITER | ROUND): line attributes

sketch_dec19b §

```
size(100,200);
smooth();
line (10,10,80,10);

strokeCap(SQUARE);
strokeWeight(20);
line(10,40, 80,40);

strokeCap(ROUND);
line(10,70, 80,70);

strokeJoin(ROUND);
rect(10, 100, 80, 90);
```



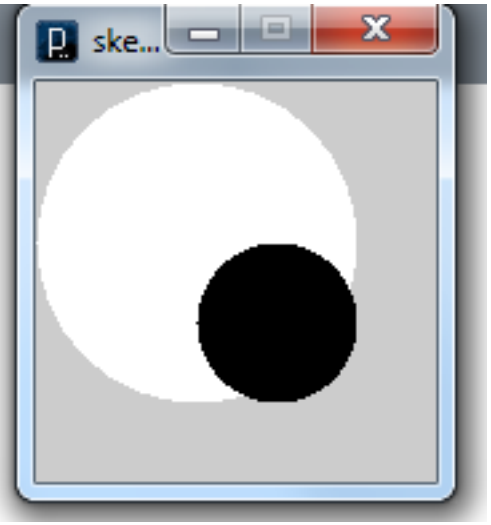
#1: drawing modes

- ellipseMode(**CENTER** |RADIUS | CORNER | CORNERS): change the way to draw ellipses

- rectMode(**CORNER** | CORNERS | CENTER): change the way to draw rectangles

sketch_dec19b §

```
size(150,150);
noStroke();
ellipseMode(RADIUS);
ellipse(60,60, 60, 60);
fill(0); // black
ellipseMode(CORNER);
ellipse(60,60, 60, 60);
```

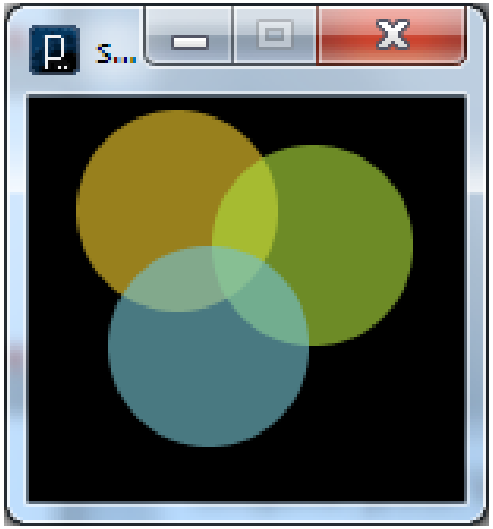


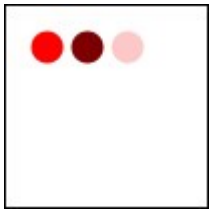
#1: assignments

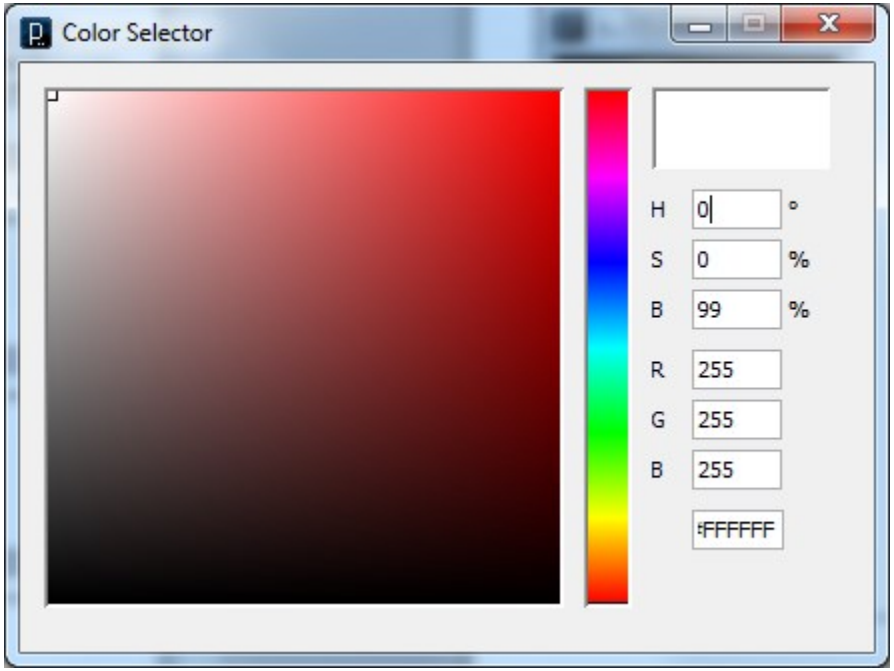
- Create a composition using at least one ellipse, one line and one rectangle

- Create a visual knot using beziers curve

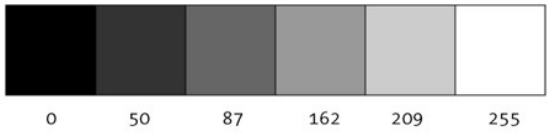
#2 - Colors







#2: use colors



- Setting colors
  - **background**(value) or background(R,G,B): sets the colors for the background
  - **colorMode**(mode): changes the way Processing interprets color data

  - **fill**(value) or **fill**(R,G,B, *alpha*): sets the color to fill shapes
  - **noFill**(): disables filling geometry

  - **stroke**(value) or **stroke**(R, G, B, *alpha*): sets the color used to draw lines and borders around shapes
  - **noStroke**(): disables drawing the stroke (outline)

#2: use colors

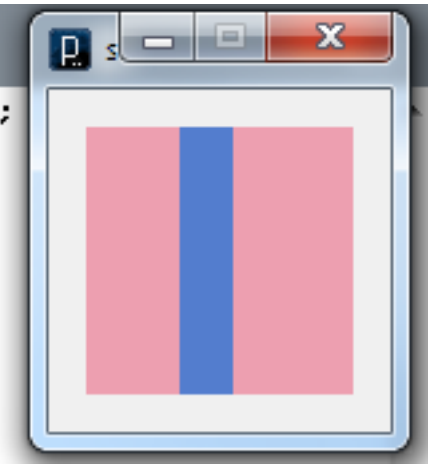
- Color Data

  The color data type is used to store colors in a program.

  - color(gray, *alpha*)
  - color(R,G,B, *alpha*)

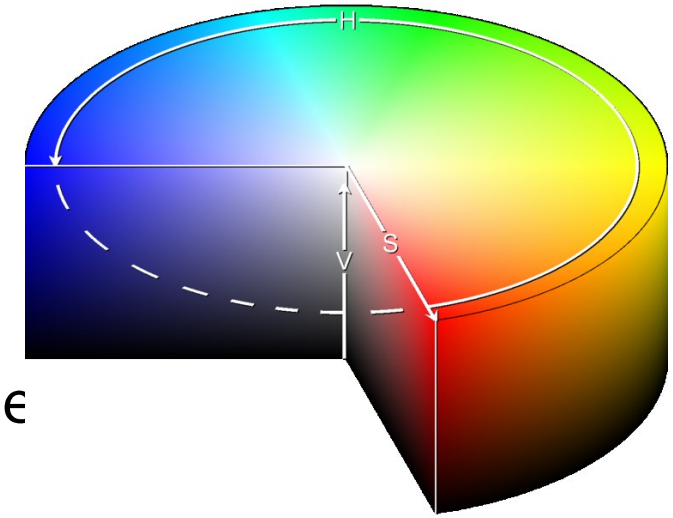
sketch_dec19b §

```
color pink = color(237,159,176,160);
color myblue = color(83,125,206);

background(pink);
noStroke();
fill(myblue);
rect(35,0,20,100);
```



#2: RGB vs HSB

You can use HSB (Hue, Saturation, Brightness) specification instead of RGB colors



- colorMode(mode)
- colorMode(mode, range);
- colorMode(mode, range1, range2, range

#2: HEX

Alternative notation for defining colors. Must begin with # sign

background(255,255,255) = background(#FFFFFF)

#2: assignments

- Use HSB color and a for structure to design a gradient between two colors
- Redraw your composition from #1 assignments using colors.
- Draw a simplified picture like Mondrian (father of neoplasticisim)

#3 - Mathematics

#3: arithmetic

Like any language, addition (+), substraction (-), multiplication (*), division(/) and modulus (%) [integer remainer after division] can be used to control position of elements or to change attribute values.

Order of operations:

- Multiplicative: *, /, %
- Additive: +, -
- Assignment: =

#3: shortcuts

Code shortcuts are used to make program more concise

- ++: adds the value 1 to a variable
- --: subtracts the value of 1
- The value can be incremented or decremented before or after the expression is evaluated (ex: x++ or ++x)

You can combine operations and assignments:

- +=, -=, *=, /= (ex: x+=2 is equivalent to x = x+2)

The negation operator changes the sign of value to its right

#3: constraining numbers

These functions are used to perform calculations on numbers that the standard arithmetic operators can't

- ceil(*value*): calculates the closest int value greater or equal to the value of its parameter
- floor(*value*): calculates the closest int value less or equal to the value of its parameter
- round(*value*): calculates the closest int value the value of its parameter
- min(*a, b, …*) and max(*a, b, …*): determines the smallest/largest value in a sequence of numbers.

#3: normalizing, mapping

Numbers are often converted to the range 0.0 to 1.0 for malking calculations. To normalize a number, you can divide it by the maximum value that it represents.

You can also use the *norm*() function

- norm(*value, low, high*): the *value* parameter is converted to a value between *low* and *high* values

After normalization, a number can be converted to another range of values. You can use the *lerp*() - linear interpolation- function

- lerp(*value1, value2, amt*): *value1* and *value2* define the min and max value an *amt* the value to interpolate.

#3: normalizing, mapping

Finally, the map() function is useful to convert directly from one range to another.

- map(*value, low1, high1, low2, high2*): *value* is the number to re-map from [*low1,high1*] range to [*low2,high2*]

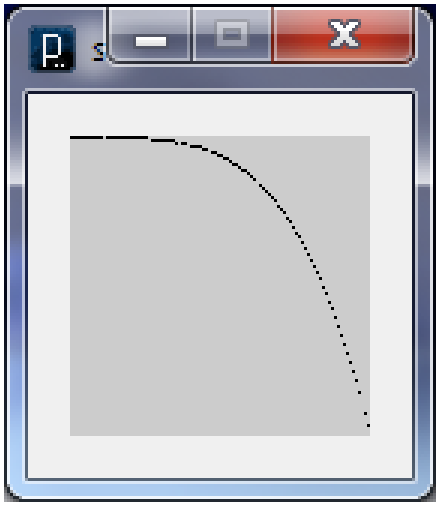
#3: exponents, roots

- sq(*value*): squares a number and return the result (positive number)
- sqrt(*value*): calculates the square root (opposite of sq function, always positive). *value* must be a positive number
- pow(*num, exponent*): calculates a number raised to an exponent.

#3: curves

Exponential functions are useful for creating simple curves. These equations have the form $y = x^n$ (where x is between 0.0 and 1.0)

```
for (int x=0;x<100;x++)
{
  float n = norm(x, 0.0, 100.0);
  float y = pow(n, 4);
  y*=100;
  point(x,y);
}
```



#3: assignments

- Draw the curve $y=1-x^5$ to the display window
- Use the data from curve $y=x^3+2x^2-5$ to draw something unique

#4 – Images

#4: display images

Processing can load images, display them or change their size, position, opacity and tint. Processing uses a special data type called *PImage*.

The image must be inside the data folder.

#4: images

Processing can load from a file by using the function *loadImage()* or create automatically an image from scratch with createImage()

Once the image is loaded, it is displayed with the *image()* function

- loadImage("*filename.ext*")
- createImage(width, height, RGB | ARGB | ALPHA);
- image(img, origin_x, origin_y, [width, height]);

There are many other useful functions to deal with PImage!

#4: tint

Images are colored with the tint() function. It affects only images.

- tint(gray, *alpha*)
- tint(value1,value2,value3, *alpha*)
- tint(color)

```
img = loadImage("rose.jpg");
tint(255,204,0,153);
image(img,0,0);
```

#4: transparency

gif and *png* images retain their transparency (respectively 1-bit and 8-bit transparency) when loaded and displayed in Processing. This allows anything drawn before the image to be visible through the transparent section of the image.


```
PImage img;
size(160,200);
img=loadImage("fitg.png");
background(0,0,255); //blue
image(img,0,0);
```

#4: filters

Digital images can be easily reconfigured and combined with other images.

Processing provides function to filter images. The PImage class has also a filter method that can isolate filtering to a specific image

- filter (*mode [, level]*) where mode = THRESHOLD, GRAY, INVERT, POSTERIZE, BLUE, OPAQUE, ERODE, DILATE

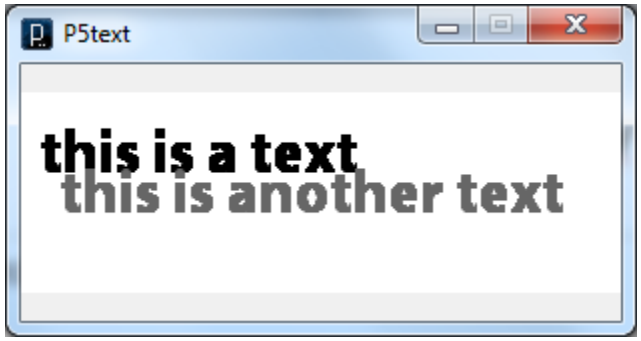
Filter function affects only what has already been drawn.

It is also possible to blend or mask some images and copy pixels with blend(), mask() or copy() functions.

#4: assignments

- Draw two images in the display window with a different tint
- Load a png image with transparency and create a collage by layering the image.
- Apply BLUR filter to the png image

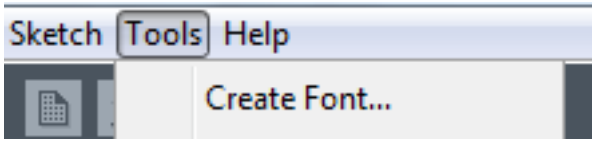
#5 - Typography



#5 – loading fonts, drawing text

Processing can load fonts and display. Processing uses a special data type called *PFont*.

Processing uses a special format e.g. VLW enabling Processing to render quickly text.



You will need to use the « *Create Font…* » in the « *Tools* » menu

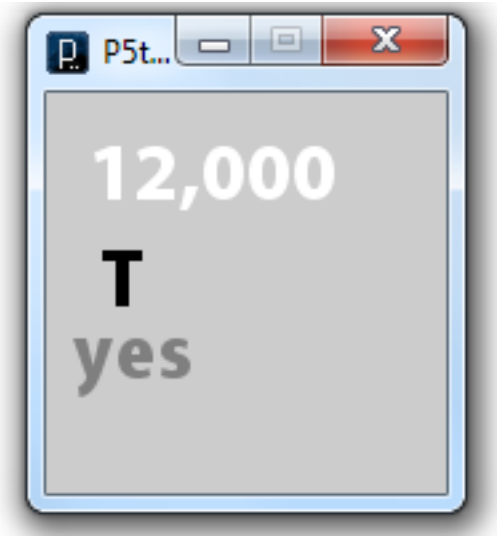
The font is saved in the « *data* » subdirectory of your project.

```
PFont myfont;
myfont = loadFont("MyriadWebPro-Bold-30.vlw");
textFont(myfont);
fill(0);
text("a text",10,40);
```

#5 – loading fonts, drawing text

1) Use loadFont() function to create a PFont variable
2) Use text() function to draw characters to te screen
   - text(data, x, y) `// data may be String, char, int, float`
   - text(stringdata, x, y, width, height)

```
size(150,150);
PFont myfont;
myfont = loadFont("MyriadWebPro-Bold-30.vlw");
textFont(myfont);
text(12.0,10,40);
fill(0);
text('T', 20,80);
fill(127);
text("yes", 10, 80, 100, 200);
```

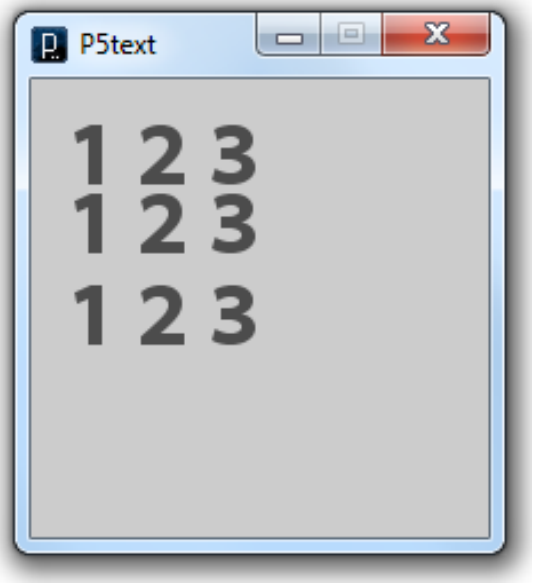


#5: text attributes

- *textSize(size)*: set the current font size (in pixels). This function resets the text leading.

- *textLeading(dist)*: set the spacing between lines of text

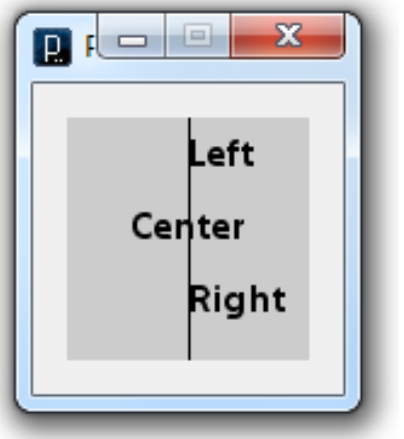
```
size(200,200);
PFont myfont;
myfont = loadFont("MyriadWebPro-Bold-30.vlw");
textAlign(RIGHT);
textFont(myfont);

textSize(40);
fill(0,160);
text("1 2 3", 10, 10, 90, 90);
textLeading(30);
text("1 2 3", 10, 40, 90, 90);
textLeading(90);
text("1 2 3", 10, 80, 90, 90);
```



#5: text attributes

- *textAlign(LEFT | CENTER | RIGHT)*: set the alignment for drawing text

```
PFont myfont;
myfont = loadFont("MyriadWebPro-Bold-30.vlw");
textFont(myfont);
textSize(16);

line(50,0,50,100);
fill(0);
textAlign(LEFT);
text("Left", 50, 20);
textAlign(CENTER);
text("Center", 50, 50);
textAlign(LEFT);
text("Right", 50, 80);
```



- *textWidth()*: calculates and returns the pixel width of any character or text string.

#5: assignments

- Draw your favorite word to the display window in your favorite typeface

- Use two different typefaces to display a dialogue between two characters.

#6 - Data

```
165956862 MTM State=alive Cursors=3 t=952313484
165956862 CIM TestPoint idFinger=1 x=826 y=214 precision=1 t=952313484
165956862 CIM ResPoint idFinger=0 idObject=delimitation t=952344684
165956862 CIM ResPoint idFinger=1 idObject=delimitation t=952344684
165956862 CIM ResPoint idFinger=1 idObject=layer1 t=952344684
165956862 CIM ResPoint idFinger=1 idObject=layer3 t=952344684
165956862 CIM ResPoint idFinger=3 idObject=delimitation t=952344684
165956862 MTM State=alive Cursors=3 t=952360284
165956862 CIM TestPoint idFinger=1 x=826 y=214 precision=1 t=952360284
165956878 CIM ResPoint idFinger=0 idObject=building006 t=952360284
165956878 CIM ResPoint idFinger=0 idObject=layer1 t=952360284
165956894 CIM TestPoint idFinger=0 x=908 y=122 precision=1 t=952313484
165956894 CIM ResPoint idFinger=1 idObject=delimitation t=952313484
165956940 CIM ResPoint idFinger=3 idObject=layer1 t=952391484
165956940 CIM TestPoint idFinger=3 x=977 y=107 precision=1 t=952313484
165956940 CIM ResPoint idFinger=3 idObject=building006 t=952391484
165956940 CIM ResPoint idFinger=1 idObject=POI007 t=952391484
165956940 CIM ResPoint idFinger=1 idObject=rond007 t=952391484
165956940 CIM ResPoint idFinger=0 idObject=layer1 t=952391484
165956940 CIM ResPoint idFinger=0 idObject=building006 t=952391484
165956940 CIM TestPoint idFinger=1 x=826 y=214 precision=1 t=952391484
165956940 MTM State=alive Cursors=3 t=952391484
165956940 CIM TestPoint idFinger=1 x=826 y=214 precision=1 t=952375884
165956940 MTM State=alive Cursors=3 t=952375884
165956940 CIM TestPoint idFinger=1 x=826 y=214 precision=1 t=952375884
165956940 MTM State=alive Cursors=3 t=952375884
165956940 CIM ResPoint idFinger=0 idObject=delimitation t=952391484
165956940 CIM TestPoint idFinger=3 x=977 y=107 precision=1 t=952391484
```

#6: file import

Import text files: you can import data by using the *loadStrings*() function

- String[] lines = *loadStrings*("myfile.ext");

Once loaded, you can manage the strings with *split()* or *splitTokens()* functions

- *split*(str, delim): str is a string and delim a character or a string
- *splitTokens*(str) or *splitTokens*(str, tokens): if tokens parameter is not used, all whitespace characters (space, tab, new line, etc.) are used as delimiters

#7- Inputs

#7: mouse (coordinates)

Mouse input is a way to control the position and attributes of shapes on screen. The Processing variables *mouseX* and *mouseY* store the coordinates relatively to the origin of the display window (there are also pmouseX and pmouseY for previous coordinates)

```
void setup()
{
  size(100,100);
  smooth();
  noStroke();
}

void draw()
{
  background(127);
  ellipse(mouseX, mouseY, 33, 33);
  ellipse(mouseX+20, mouseY, 22, 22);
  ellipse(mouseX-20, mouseY, 11, 11);
}
```

#7: mouse (buttons)

We can also use mouse states to interact with Processing.

- *mouseButton*: LEFT, CENTER, RIGHT independently or in combination

- *mousePressed*: **true** if pressed, **false** if not.

#7: mouse (cursor)

The cursor can be hidden with the *noCursor*() function or be set with the *cursor*() function.

- *noCursor*(): hide the cursor
- *cursor*(ARROW | CROSS | HAND | MOVE |TEXT | WAIT)

```
void setup()
{
  size(100,100);
  smooth();
  noStroke();
}

void draw()
{
  if (mousePressed == true)
    cursor(CROSS);
  else cursor(HAND);
}
```

#7: mouse (events)

An event is a polite interruption of the normal flow of the program. The code inside an event function is run once each time the corresponding event occurs. There are 4 event functions for mouse

- *mousePressed*(): when a button is pressed
- *mouseReleased*(): when a button is released
- *mouseMoved*(): when the mouse is moved
- *mouseDragged*(): when the mouse is moved while a button is pressed

#7: keyboard (states)

Keyboards are typically used to input characters for composing documents. Processing can register the most recently pressed key and whether a key is currently pressed.

- *keyPressed*: **true** if pressed, **false** if not.

The *key* variable stores the most recently pressed key (a a char).

BACKSPACE, TAB, ENTER, RETURN, ESC and DELETE are also recognized directly.

#7: keyboard (states)

In addition, Processing is able to decode such as arrow keys, ALT, CRTL, SHIFT, … keys (key value = CODED).

The variable *keyCode* stores ALT, CONTROL, SHIFT, UP, DOWN, LEFT, RIGHT key as constants

#7: keyboard (events)

There are 2 event functions for keyboard

- *keyPressed*(): when a key is pressed
- *keyReleased*(): when a key is released

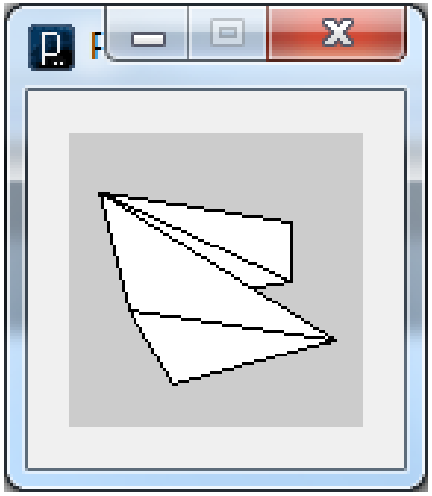
#7- assignments

- Propose two unique shapes that behave differently in relation to the mouse.

- Use the number keys on the keyboard to modify the movement of a line

- Use the arrow keys to change the position of a shape within the display window

#8 - Transformations



#8: shapes

- PShape is a special datatype to store shapes (see the online doc for more)

The shape() function is used to draw the shape to the display window. Processing can load and display SVG (Scalable Vector Graphics) and OBJ shapes.

```
PShape s1, s2;
s1 = loadShape("myshape.svg");
s2 = createShape(RECT, 0, 0, 80, 80);
…
shape(s1, 10, 10, 80, 80);
```

#8: transform function

Actually, the coordinates can be translated, rotated and scaled so shapes can be drawn with different positions, orientations and sizes.

The translation function moves the origin from the upper-left corner of the screen to another location

- translate (*x,y*) The value of parameters are added to any shapes drawn after the function is run.

The rotation function rotates the coordinate system

- rotate(*angle*)

The scale function magnifies the coordinate system

- scale(*size*) or scale(*xsize, ysize*)

#8: pop and push

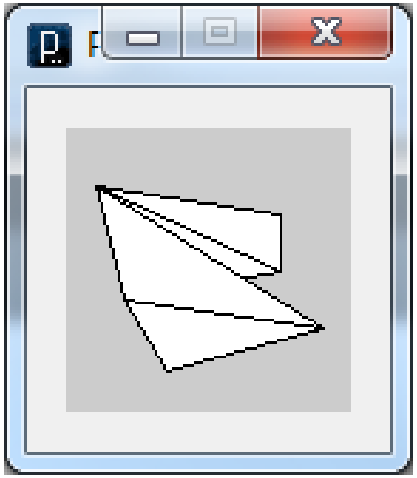
The transformation matrix is a set of numbers that defines how geometry is drawn on the screen. The *pushMatrix()* function records the current state of all transformations. To return to a previous state, we have to use *popMatrix()*

Each matrix can be seen as a sheet of paper with a current list of many transformations (translation, rotation, scale)

#8- assignments

- Create a unique shape using PShape
- Use translate() to reposition the shape
- Use translate() and rotate() to rotate the shape around its own center

#1 - Vertices

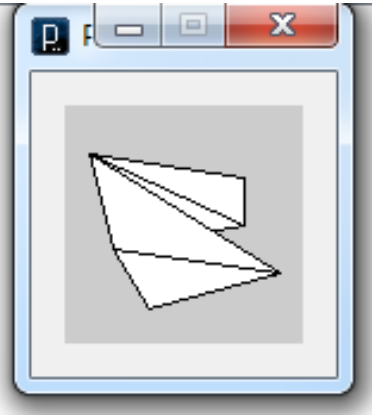


#1: vertex

To create a shape from vertex points,

- first use *beginShape()* function
  - beginShape(POINTS | LINES | TRIANGLES | TRIANGLES_STRIP | TRIANGLE_FAN | QUADS | QUAD_STRIP)
- specify a series of points with the *vertex()* function
  - vertex(x,y): all shapes are filled white and have a black outline connecting points
- then complete the shape with *endShape()* function

```
beginShape(TRIANGLE_STRIP);
vertex(75, 30);
vertex(75, 50);
vertex(10, 20);
vertex(20, 60);
vertex(90, 70);
vertex(35, 85);
endShape();
```



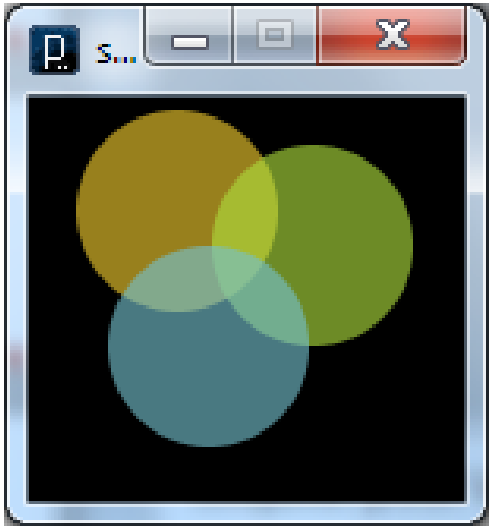
#1: curve and bezier vertex

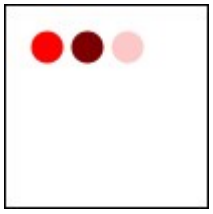
Two functions to create shapes made of curves.

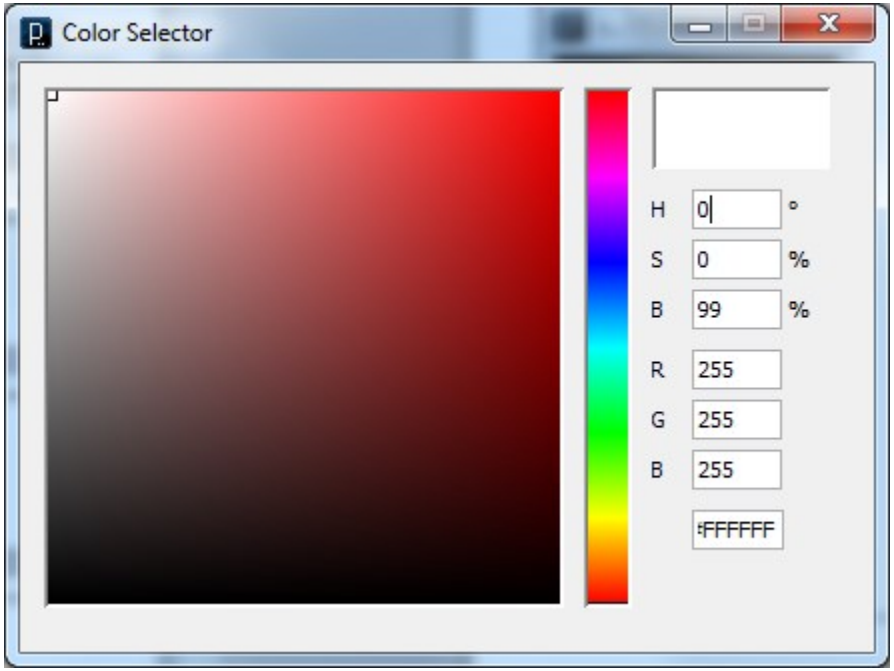
- *curveVertex(x, y)*: there must be at least 4 functions to draw a segment
- *bezierVertex(cx1,cy1,cx2,cy2,x,y)*

A good technique for creating complex shapes is to draw them first on a vector drawing program such as Adobe Illustrator or Inkscape (http://inkscape.org) 

#2 - Colors







#2: components

In Processing, *color* data type is a single number that stores the individual components of a color e.g. red, green, blue and alpha components.

It is possible to extract components from a color by using *red()*, *green()*, *blue()* and *alpha()* functions

The *hue()*, *saturation()* and *brigthness()* functions work like the previous ones but make really sense in the HSB color model.

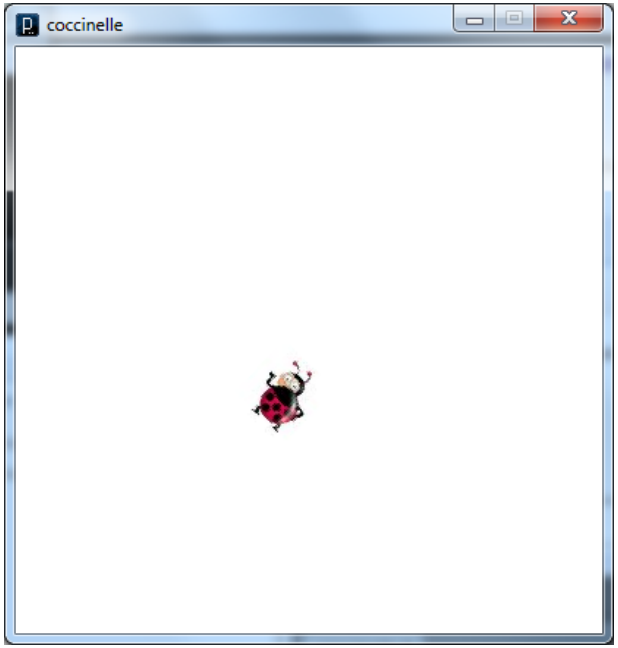
You can switch from RGB to HSB color model by using *colorMode()* function

#4 – video

#4: animation and video

To animate images, you just have to modify some Images properties

```
PImage coccinelle;
float x, y ;
float rot; // image rotation

void setup()
{
  size(400,400);
  coccinelle = loadImage("coccinelle.jpg");
  x = 0.0;
  y = width / 2.0;
  rot = 0.0;
}

void draw()
{
  background(255);
  translate(x,y);
  rotate(rot);
  image(coccinelle, 0,0);
  x+=1;
  rot +=0.01;
  if (x>width)
  {
    x=0;
  }
}
```

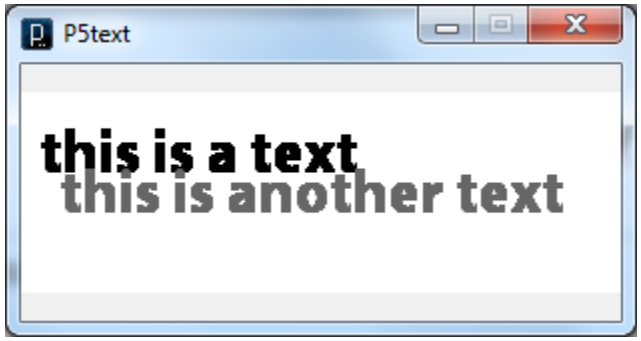


#4: animation and video

- 5 basic steps to display video

```
// Step 1: import the video library
import processing.video.*;

// step 2: declare the Capture object
Capture video;

void setup()
{
  size(320,240);
  // Step 3: initialize capture object with size and frame per second
  video = new Capture(this, 320,240,30);
  video.start();
}

void draw()
{
  if (video.available())
  {
    // Step 4: read the image from the camera
    video.read();
    // Step 5: display the image
    image(video,0,0);
  }
}
```

#5 - Typography



#5: text in action

For typography to move, the program requires a *draw()* function.

```
PFont font;

void setup()
{
   size(100,100);
   font = loadFont("MyriadWebPro-Bold-30.vlw");
   textFont(font);
   noStroke();
   frameRate(15);
}

void draw()
{
  fill(204,24);
  textSize(random(10,30));
  rect(0, 0, width, height);
  fill(0);
  text("Random", random(-100,100), random(-20, 120));
}
```

#5: text in action

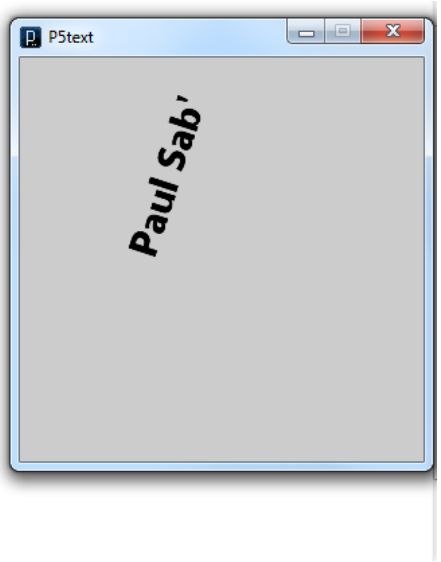
To put text into motion, simply draw it at a different position each frame.

You can also change transparency or color of the text by using the *fade()* function

You can also apply the transformations *translate()*, *scale()* and *rotate()* to create motion

Finally, you can use rapid serial visual presentation (RSVP) to display words Store the words within an array.

```
PFont font;
String s ="Paul Sab'";
float angle = 0.0;

void setup()
{
   size(300,300);
   font = loadFont("MyriadWebPro-Bold-30.vlw");
   textFont(font, 10);
   fill(0);
}

void draw()
{
  background(204);
  angle += 0.05;

  pushMatrix();
  translate(100,150);
  scale((cos(angle/4.0) + 1.2) * 2.0);
  rotate(angle);
  text(s,0,0);
  popMatrix();
}
```



#5: assignments

- Select a noun and a adjective. Animate the noun to reavel the adjective

- Use the transformation functions to animate a short sentence word per word

#7- Inputs

#7: time, date

Processing values can read the value of the computer's clock with *second()*, *minute()* and *hour()*, *millis()* and *day()*, *month()*, *year()* functions.

Placing theses functions inside *draw()* allow the time to be read continously.

#7: assignments

- Make an abstract clock that communicates the passage of time through graphical quantity rather than numerical symbols.

UNIVERSITÉ TOULOUSE III PAUL SABATIER

#9: OOP

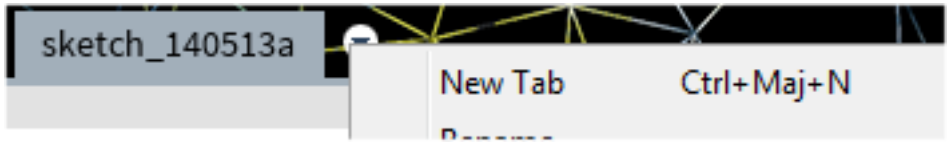
#9: objects and classes

As Processing is written in java, you can use objects (initially, this part is hidden to users).

Defining a class is creating your own data type over the primitive ones (that means *int*, *float*, *boolean*, *String*, *PImage* and *PFont*)

To define a new class, you have to use the *class* keyword inside a new tab

```
class classname
{
        //Attributes
        // Constructor(s)
        //Methods
}
```



#10: network

It’s very easy to make connections with any network such as Serial, Zigbee, bluetooth or TCP.

#10: Serial connection

- There is a special library to **read** and **write** data from external devices (for example Arduino)

```
import processing.serial.*;

Serial myPort; // The serial port
println(Serial.list());  // List all the available
serial ports
// Open the port you are using at the rate you want:
myPort = new Serial(this, COM9, 9600);
myPort.write(65); // Send an A out the serial port
…
```

#10: web client

```
import processing.net.*;
Client c;
String data;

void setup() {
  size(500, 700);
  background(50);
  fill(200);
  c = new Client(this, "www.processing.org", 80); // Connect to server on port 80
  c.write("GET / HTTP/1.0\r\n"); // Use the HTTP "GET" command to ask for a Web page
  c.write("\r\n");
}

void draw() {
  if (c.available() > 0) { // If there's incoming data from the client...
    data = c.readString(); // ...then grab it and print it
    text(data);
  }
}
```

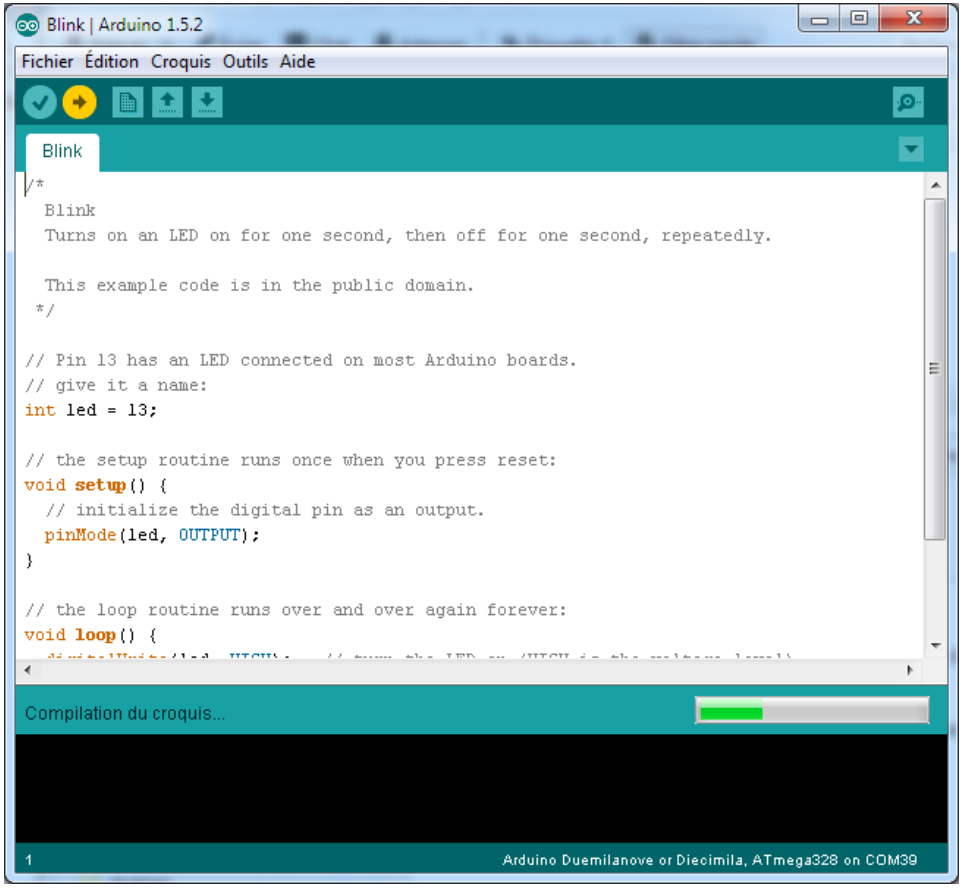
#10: RSS client

Download and install a rss client from *http://www.irit.fr/~Philippe.Truillet/ens/ens/ostfalia/data/RSS.zip*

Try it!

Modify the code to open the RSS feeds from your preferred newspaper, attribute number to each retrieved title and display **the corresponding article into a circle** when the user enters a number.

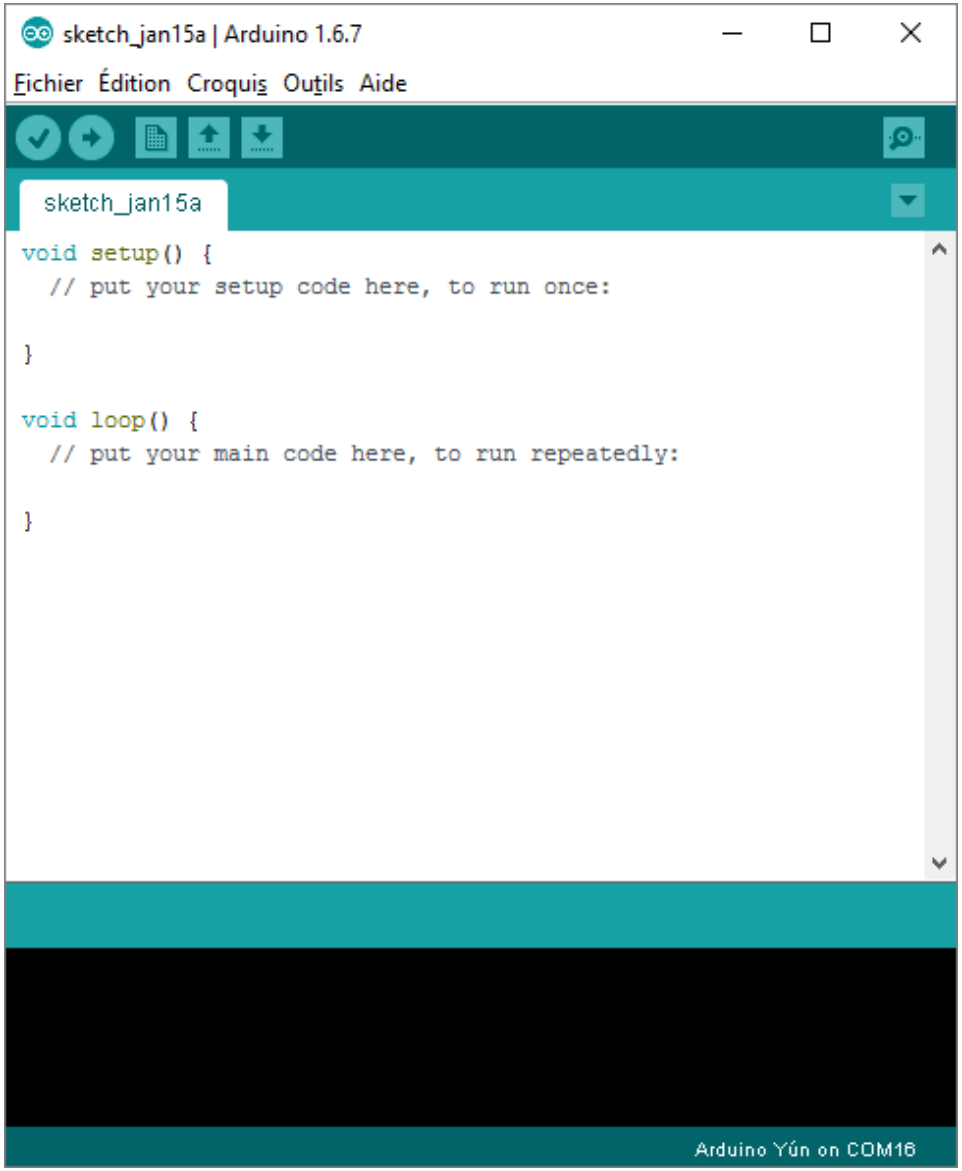
#10: Processing and arduino



Arduino: how-to?

- Installation
  - Go to *http://arduino.cc/en/Main/Software*
  - Install the 'IDE and drivers able to use serial communication

Arduino: how-to?



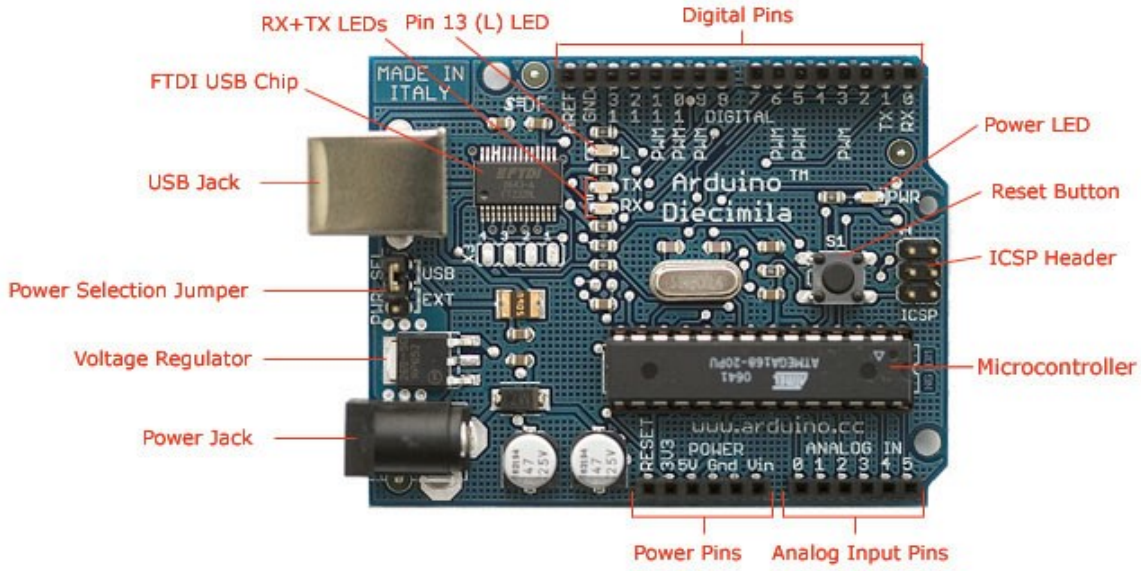
Hardware

- Several versions:
  - Diecimila / ATmega 168 / 16 Ko
  - Duemilanove / ATmega 328 / 32 Ko
  - Mega / ATmega1280 / 128 Ko
  - UNO, Due, Yun (Linux one), Leonardo, …

+ IoT platform (Onion Omega, Airboard, Particle Photon, …)

Power

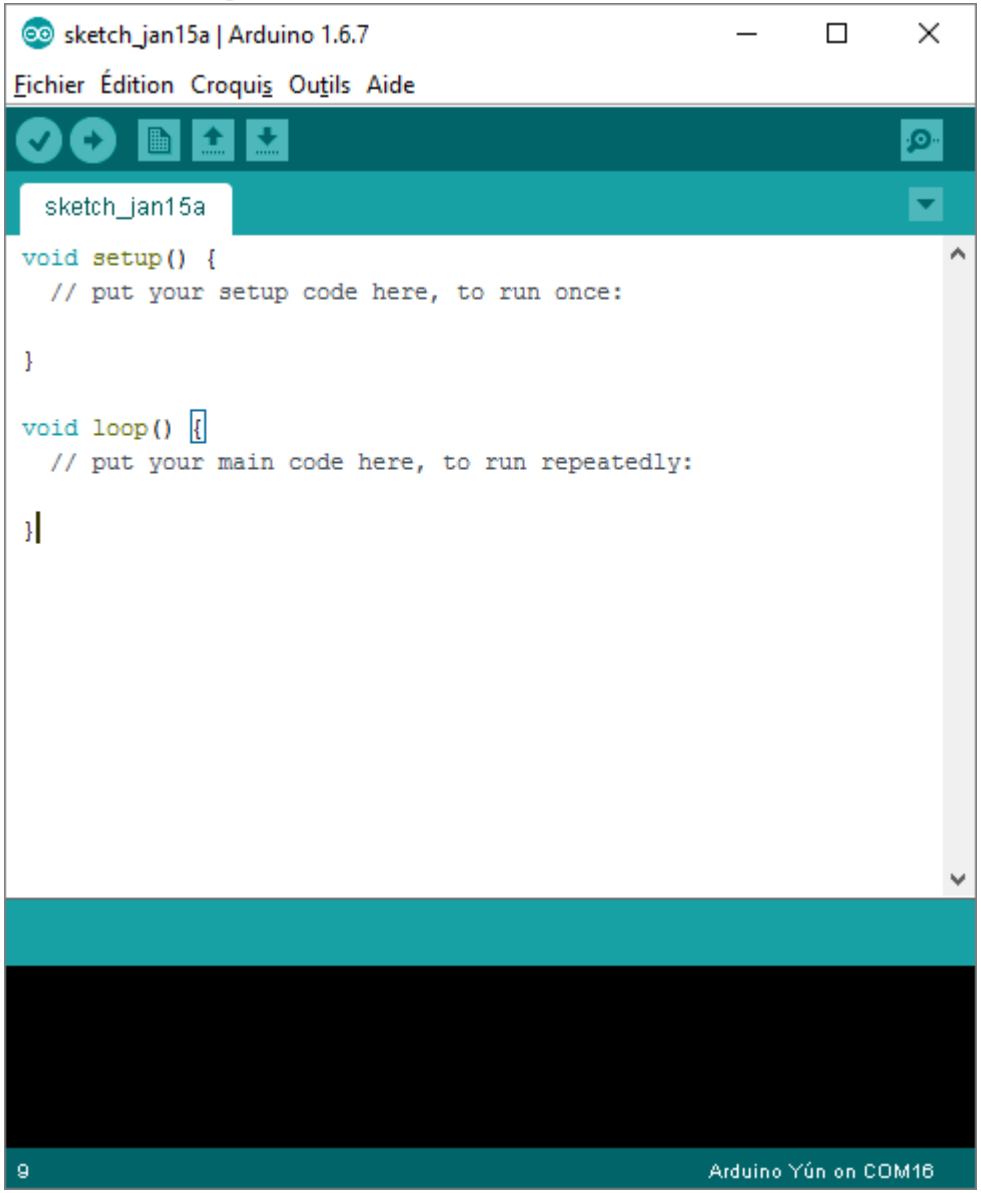
- By USB connection (5 V until to 500 mA)
- Or external power (batteries)



Photograph by SparkFun Electronics. Used under the Creative Commons Attribution Share-Alike 3.0 license.

Programming …

- Integrated environment multi OS based on java (v. **1.8.7** – 11/09/2018)



Programming

- 2 main functions for a « sketch »
  - **setup():** like in processing to init variables, I/O pins, …
  - **loop()**: the mainloop (like the *draw* function in Processing)

Programming

- C-like syntax
- Reference : http://arduino.cc/fr/Main/Reference

Structure

Fonctions de base

Ces deux fonctions sont obligatoires dans tout programme en langage Arduino :

- void setup()
- void loop()

Structures de contrôle

- if
- if...else
- for
- switch case
- while
- do... while
- break
- continue
- return

Syntaxe de base

- ; (point virgule)
- {} (accolades)

Variables et constantes

Les variables sont des expressions que vous pouvez utilisez dans les programmes pour stocker des valeurs, telles que la tension de sortie d'un capteur présente sur une broche analogique.

Constantes prédéfinies

Les constantes prédéfinies du langage Arduino sont des valeurs particulières ayant une signification spécifique.

- HIGH | LOW
- INPUT | OUTPUT
- true | false

A ajouter : constantes décimales prédéfinies

Expressions numériques

- Expressions numériques entières

Fonctions

Entrées/Sorties Numériques

- pinMode(broche, mode)
- digitalWrite(broche, valeur)
- int digitalRead(broche)

Entrées analogiques

- int analogRead(broche)

Sorties "analogiques" (génération d'impulsion)

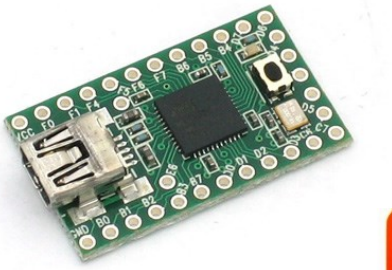
- analogWrite(broche, valeur) - PWM

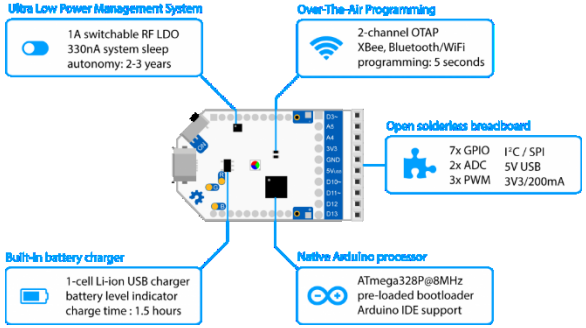
Entrées/Sorties Avancées

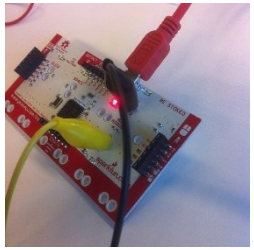
- tone()
- noTone()
- shiftOut(broche, BrocheHorloge, OrdreBit, valeur)
- unsigned long pulseIn(broche, valeur)

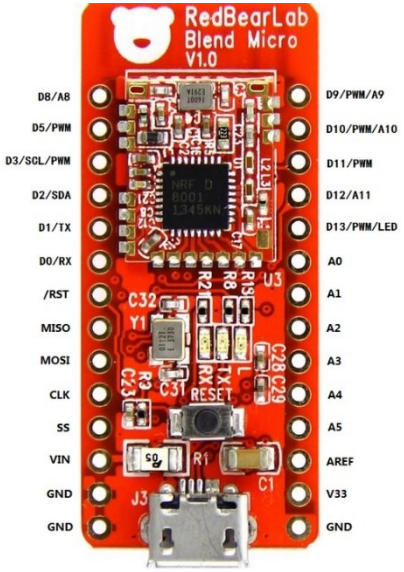
Arduino: the family…

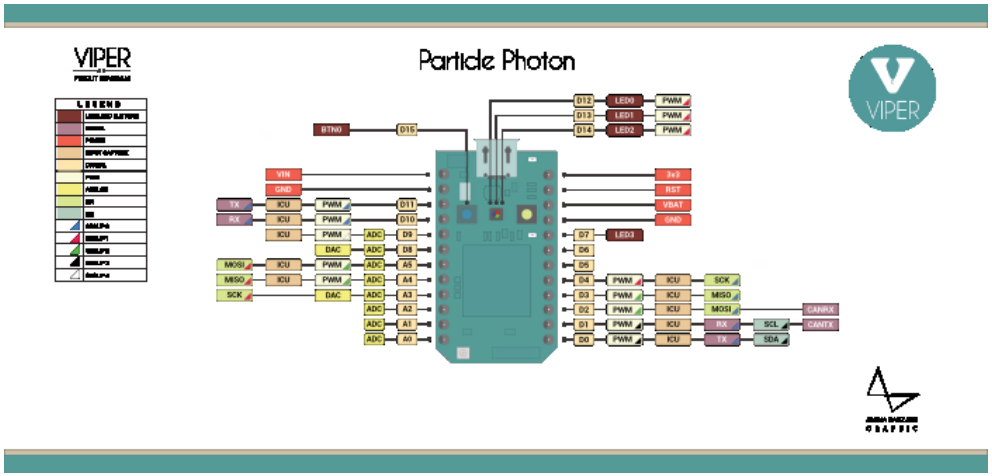
- Many other projects: Teensy, Makey-Makey, The Airboard, Particle Photon, clones, …











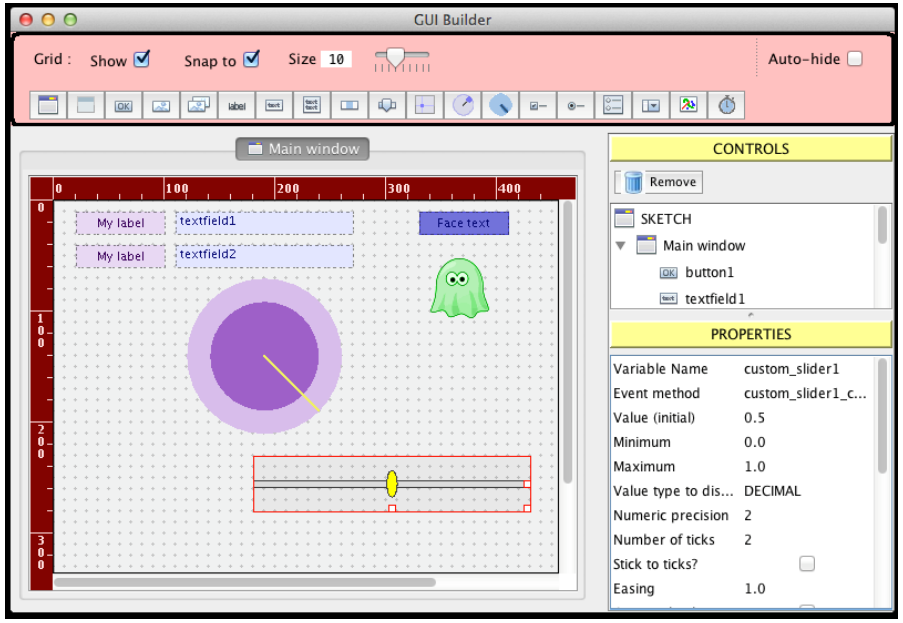
#11: third-party libraries

Finally, you can (re) use some libraries (classes, functions) written by others to manage for example sound, mixed reality, networks, etc.

#11: GUI Builder Tool

You can use G4P (GUI for Processing) and GUI Builder Tool (http://lagers.org.uk/g4ptool/index.html) to generate easily a “standard interface”.

Libraries are available for download or installation through the PDE



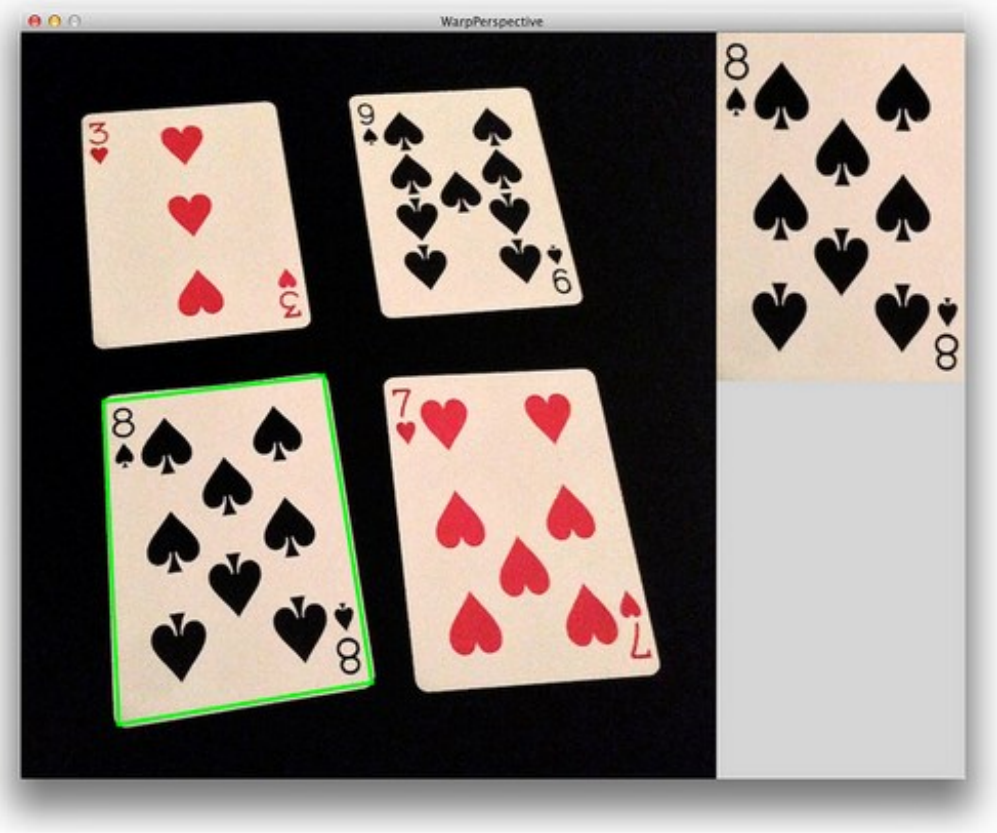
#11: speech recognition and TTS

- Try to use STT library (*http://stt.getflourish.com*) to be able to use the Google speech recognizer in Processing!

- Try to use at the same time TTS library (*http://www.local-guru.net/blog/pages/ttslib*) at the same time

#11: OpenCV

- Try to use OpenCV (*https://github.com/atduskgreg/opencv-processing/releases*) and enjoy image processing!



#11: … and more!

- You can find several libraries to manage kinect, Leap Motion, encode and decode QRCode (Zxing project), make augmented reality applications, use middleware, etc.

Be curious and enjoy!

#12: my own libraries

Moreover, you can also create some new libraries (**classes**, functions) for you and for the entire community!

How to deal with classes? Simply by putting them in different tabs and write the code or by importing them like in Java (you can put .class file in the “code” directory) ☺

#12: assignments

- Create a new interactive object that reacts to mouse pressed and mouse released events (could be for example a pie menu or a button)

- Insert this interactive object onto a sketch to illustrate its capabilities.

#13: tips

Full screen :

- *Processing 2 :*

```
size(displayWidth, displayHeight, P2D); // SETUP

// Full sreen mode
boolean sketchFullScreen() {
    return(true);
}
```

- *Processing 3 :*

```
fullScreen()  // SETUP
```

Links

- **Processing** : www.processing.org

- **Processing.js** : processingjs.org

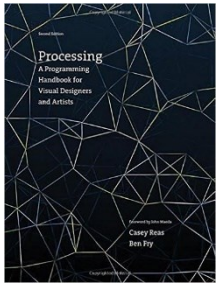
- **p5js** : p5js.org

p5.js

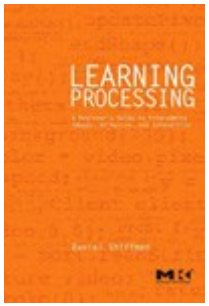
- **Reference**: http://processing.org/reference
- **Learning Processing**: http://www.learningprocessing.com
- **Hello Processing** : http://hello.processing.org

UNIVERSITÉ TOULOUSE III PAUL SABATIER

Links



Processing: A Programming Handbook for Visual Designers and Artists, Casey Reas and Ben Fry



Learning Processing: A Beginner's Guide to Programming Images, Animation, and Interaction, Daniel Shiffman
http://learningprocessing.com

The Nature of Code: Simulating Natural Systems with Processing, Daniel Shiffman
http://natureofcode.com