cs6630 | September 16 2014

INTRO TO PROCESSING

Hitesh Raju

University of Utah

slide acknowledgements:
http://processing.org/

administrivia . . .

- data exploration assignment due today!
- time series assignment out today
- open lab in class on Thurs.

SCI Visualization Journal Club

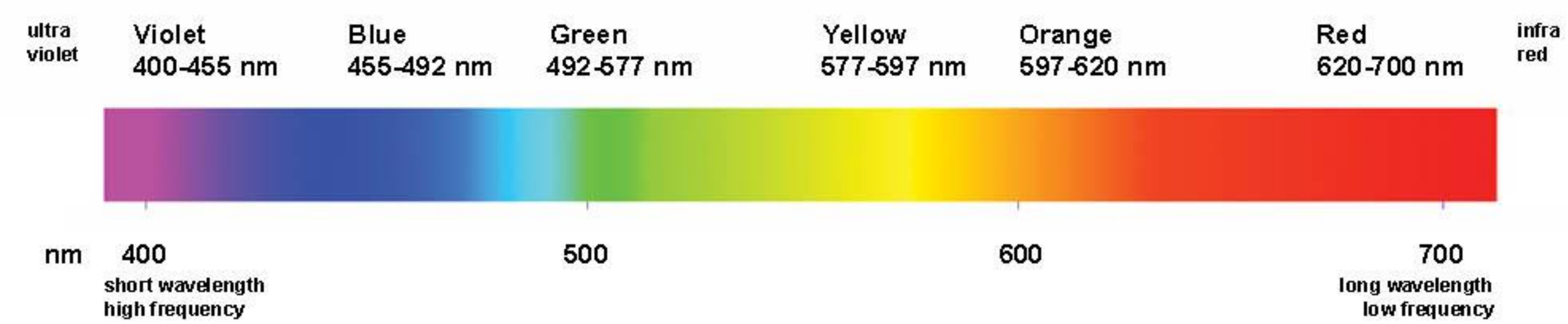
Meeting every Tuesday at 2:30pm is the Halvorsen Conference Room (WEB 4640)

This week's paper: Illustrative Visualization of Molecular Reactions using Omniscient Intelligence and Passive Agents" from EuroVis 2014

For more information, e-mail Paul (prosen@sci.utah.edu) or visit http://www.sci.utah.edu/the-institute/events/vjc-fall2014.html

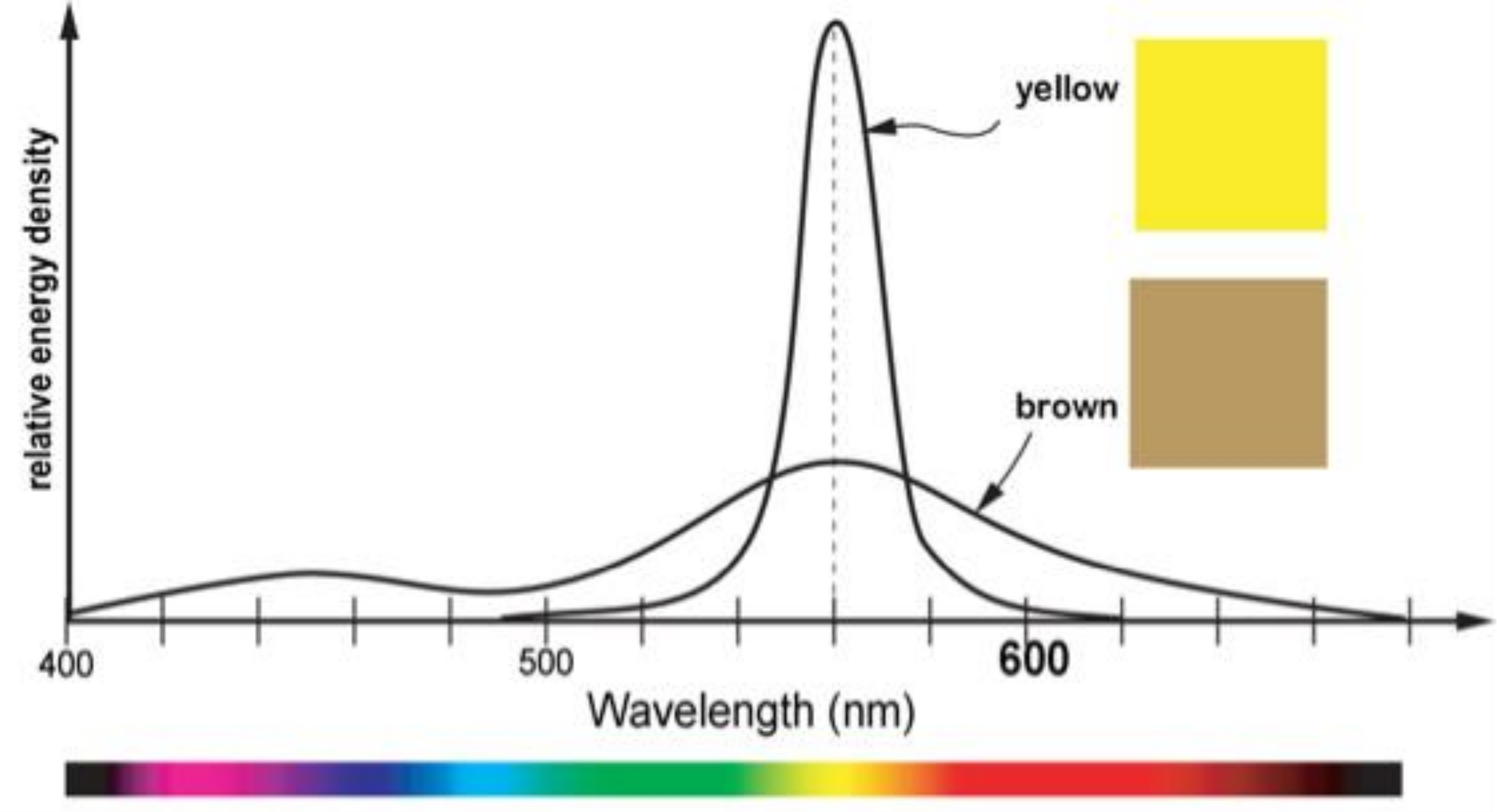
last time . . .

(human) visible light

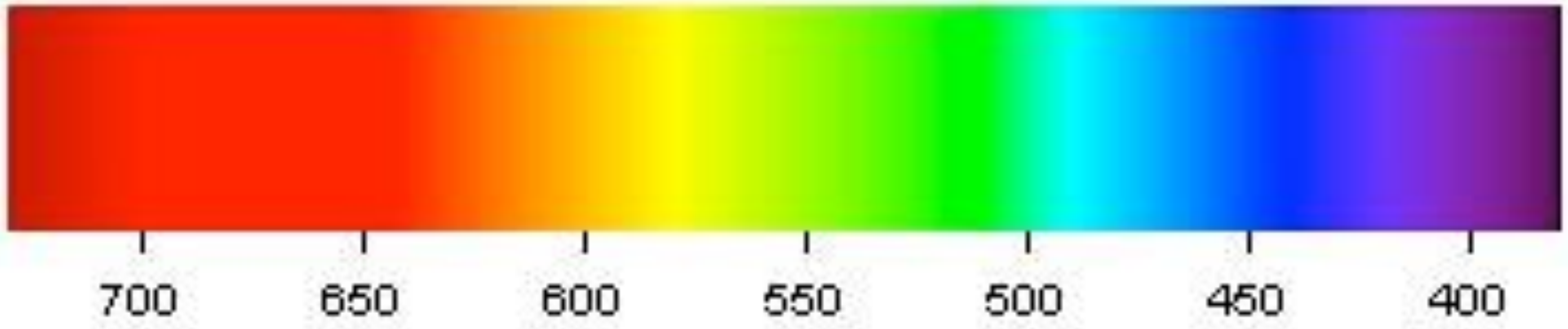


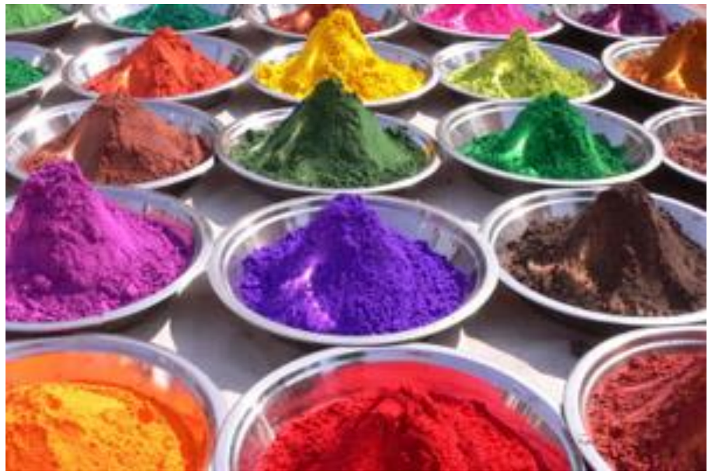
Color != Wavelength

but rather, a combination of wavelengths and energy

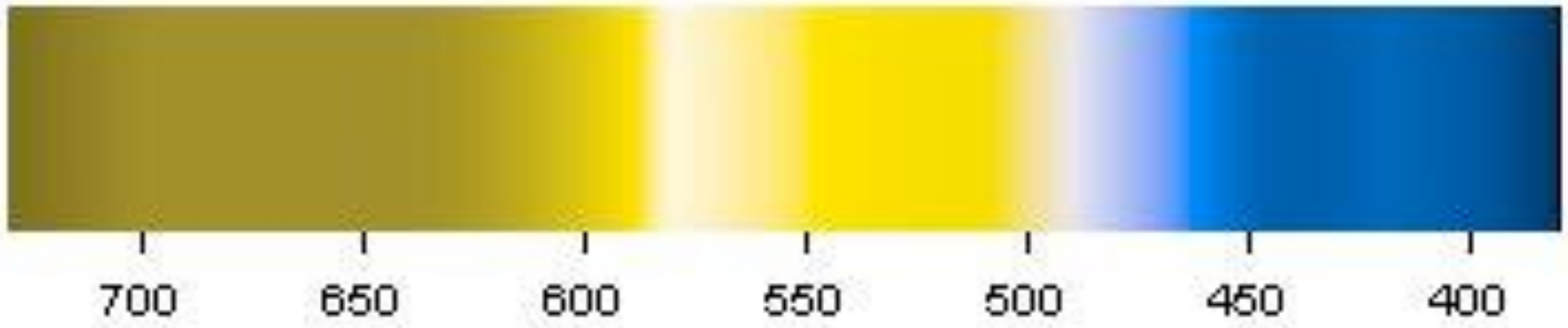


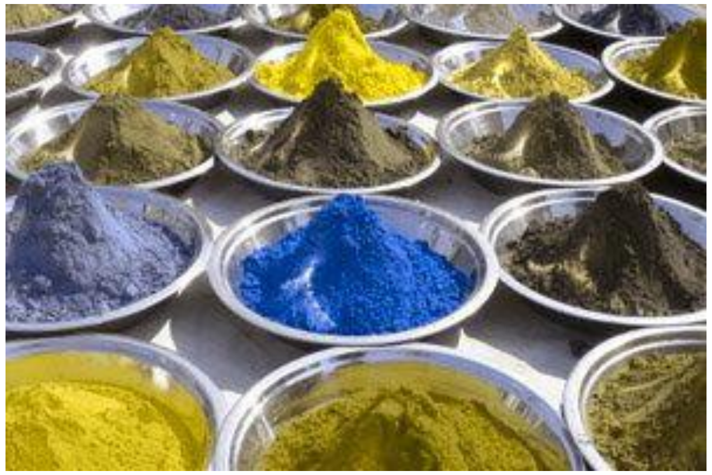
Normal



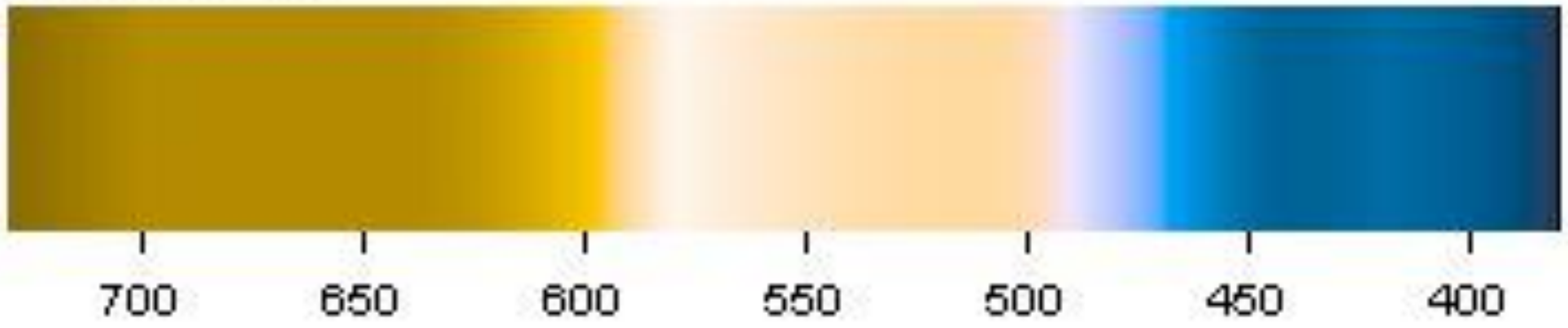


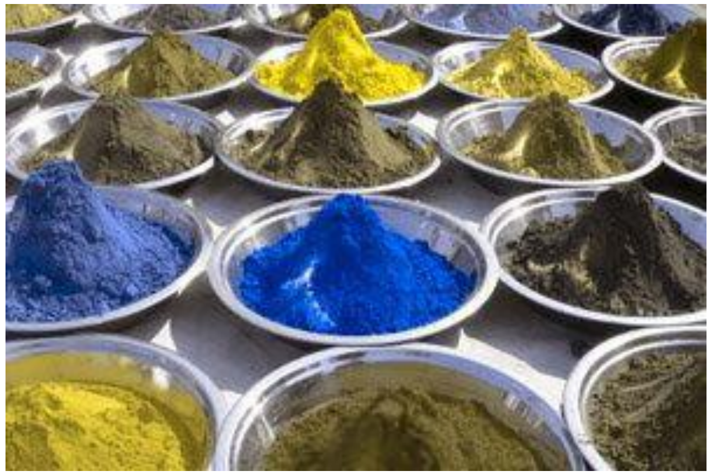
Protanopia



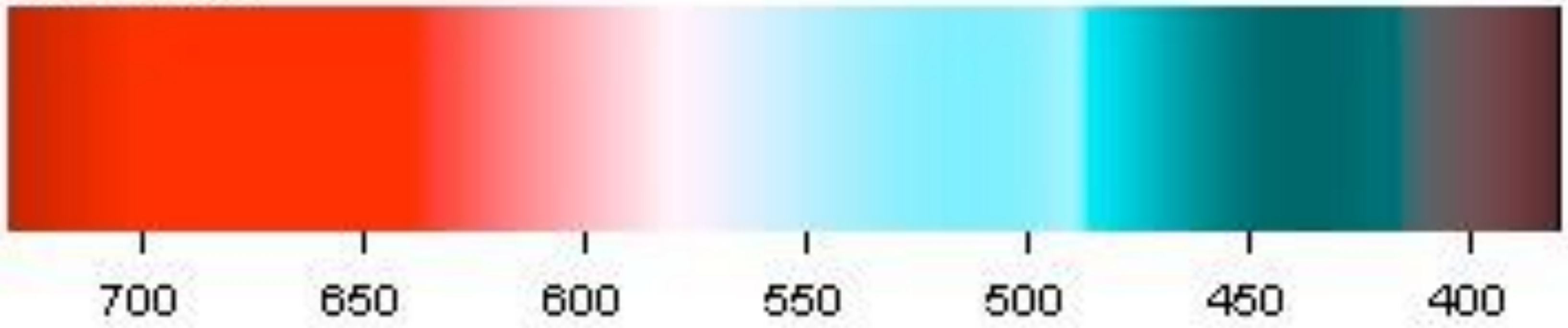


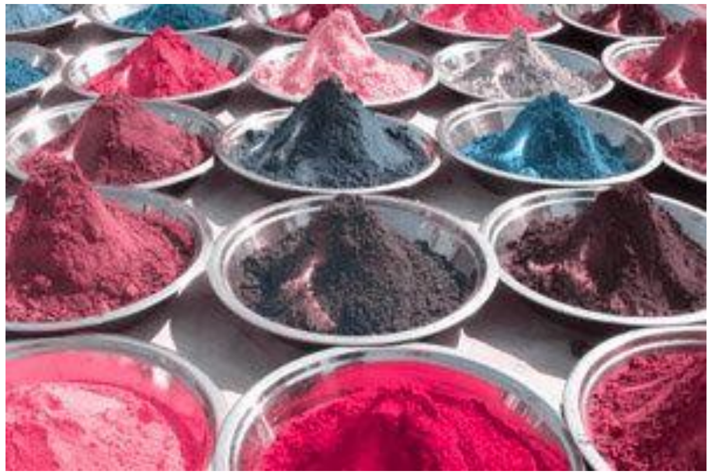
Deuteranopia





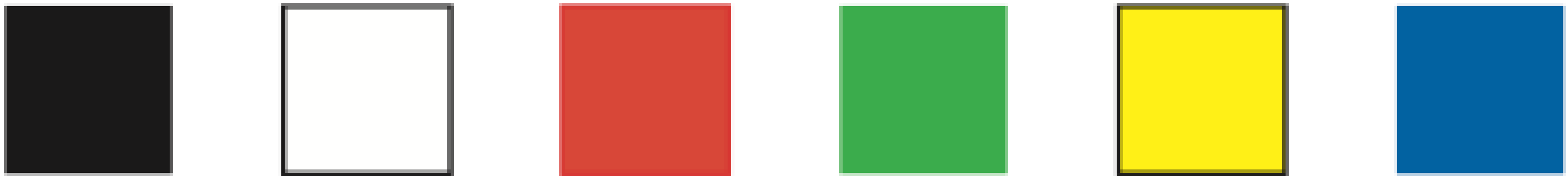
Tritanopia



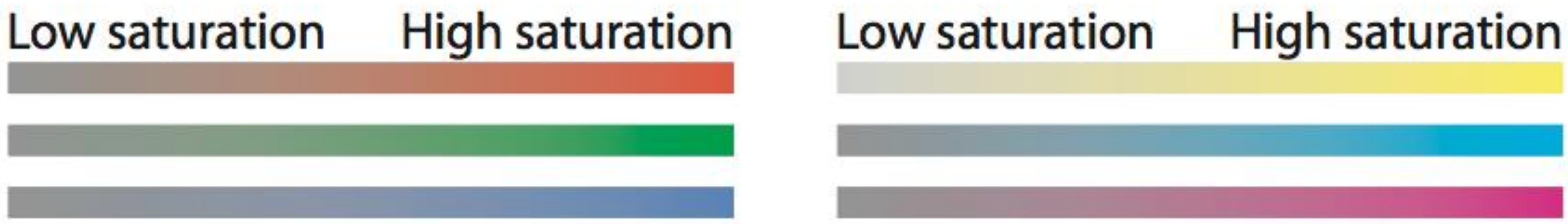


terms

- hue (chroma)



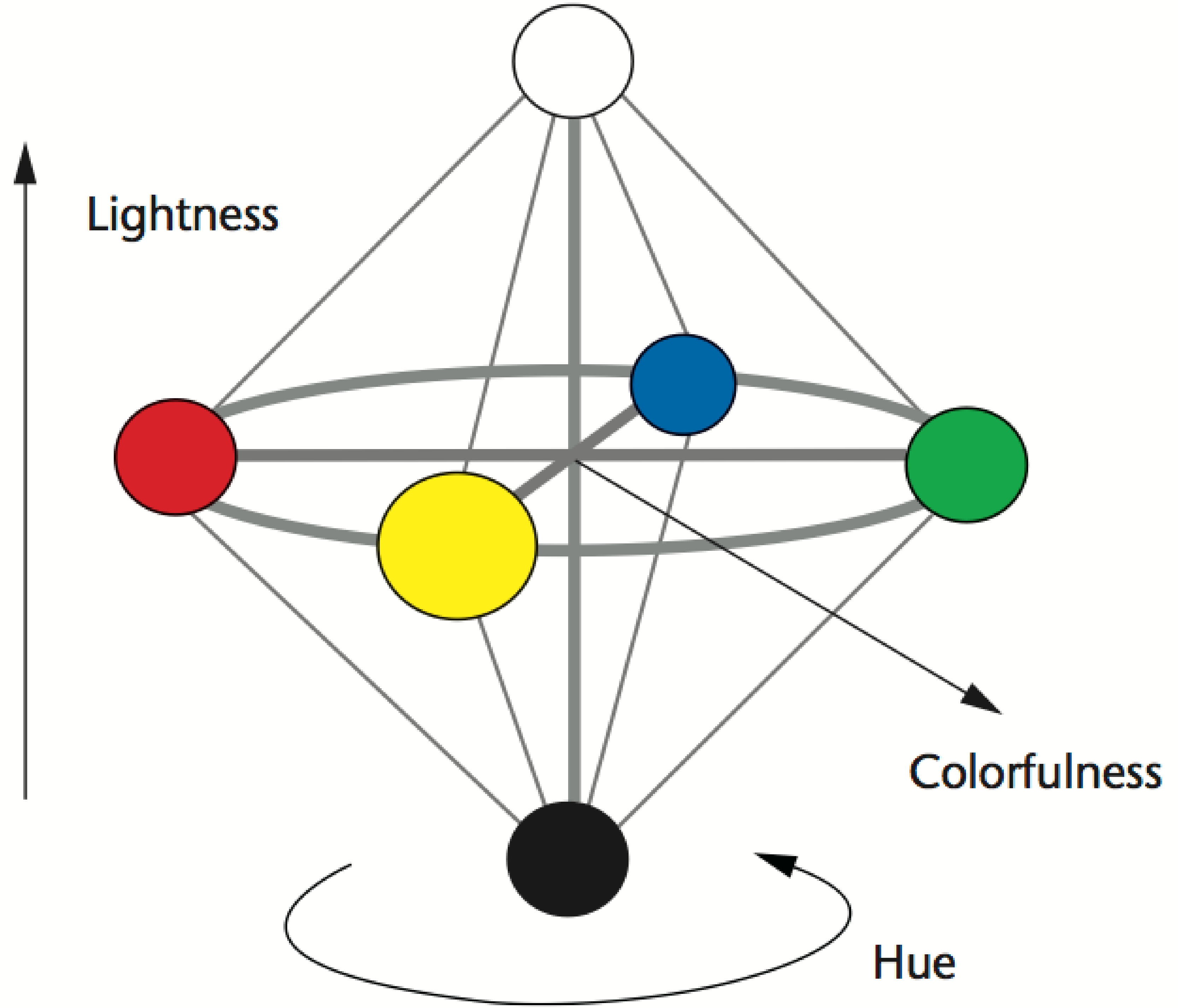
- saturation (chromaticity)



- luminance (lightness / brightness / value)

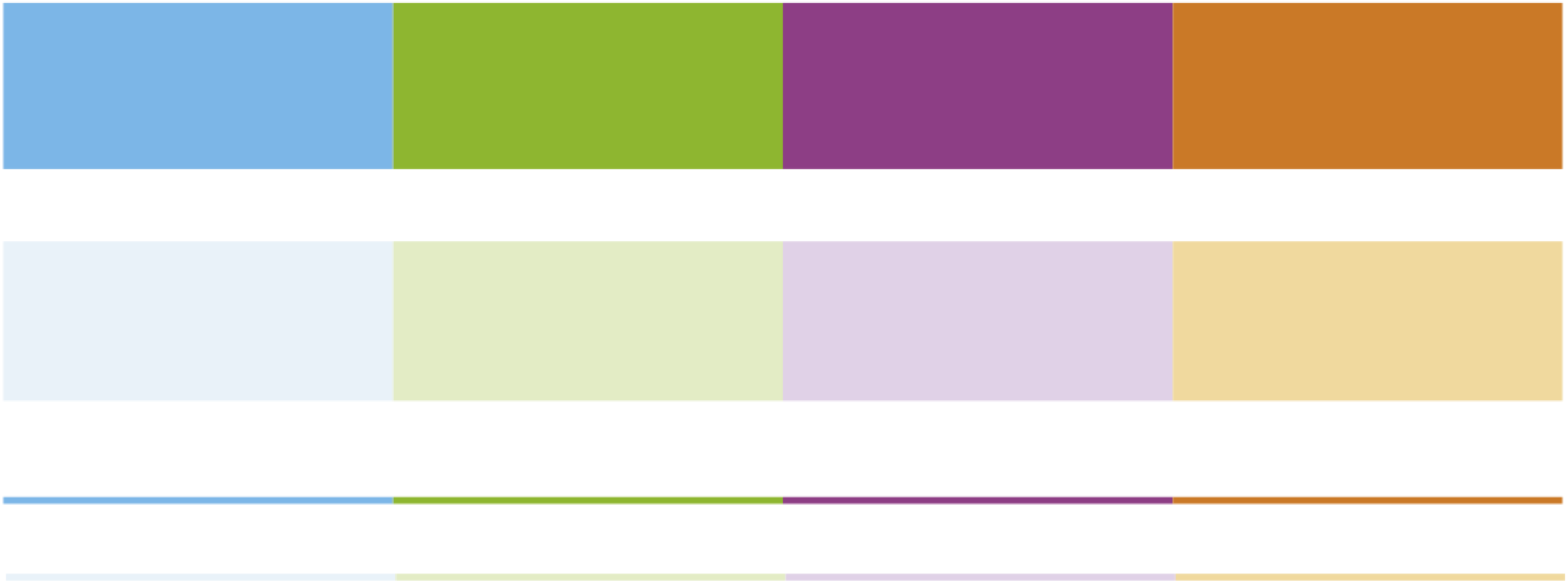
perceptual color spaces

change i
change



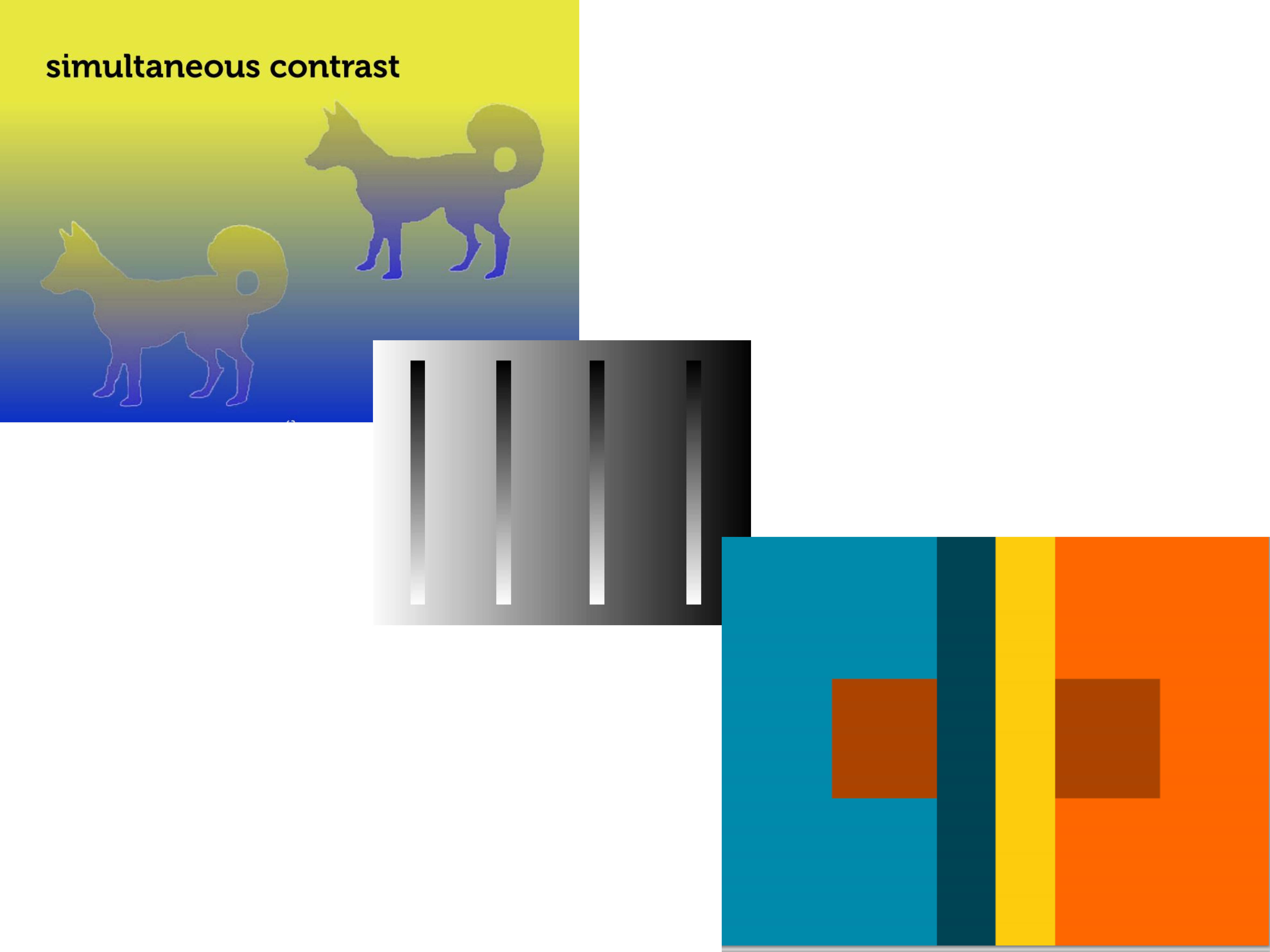
isual

size & color



“the smaller the mark, the less distinguishable are the colors”

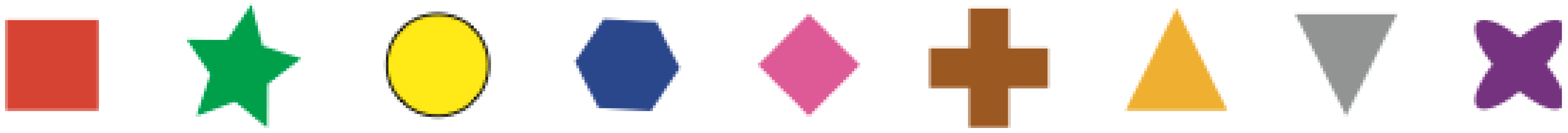
-Jacques Bertin



simultaneous contrast

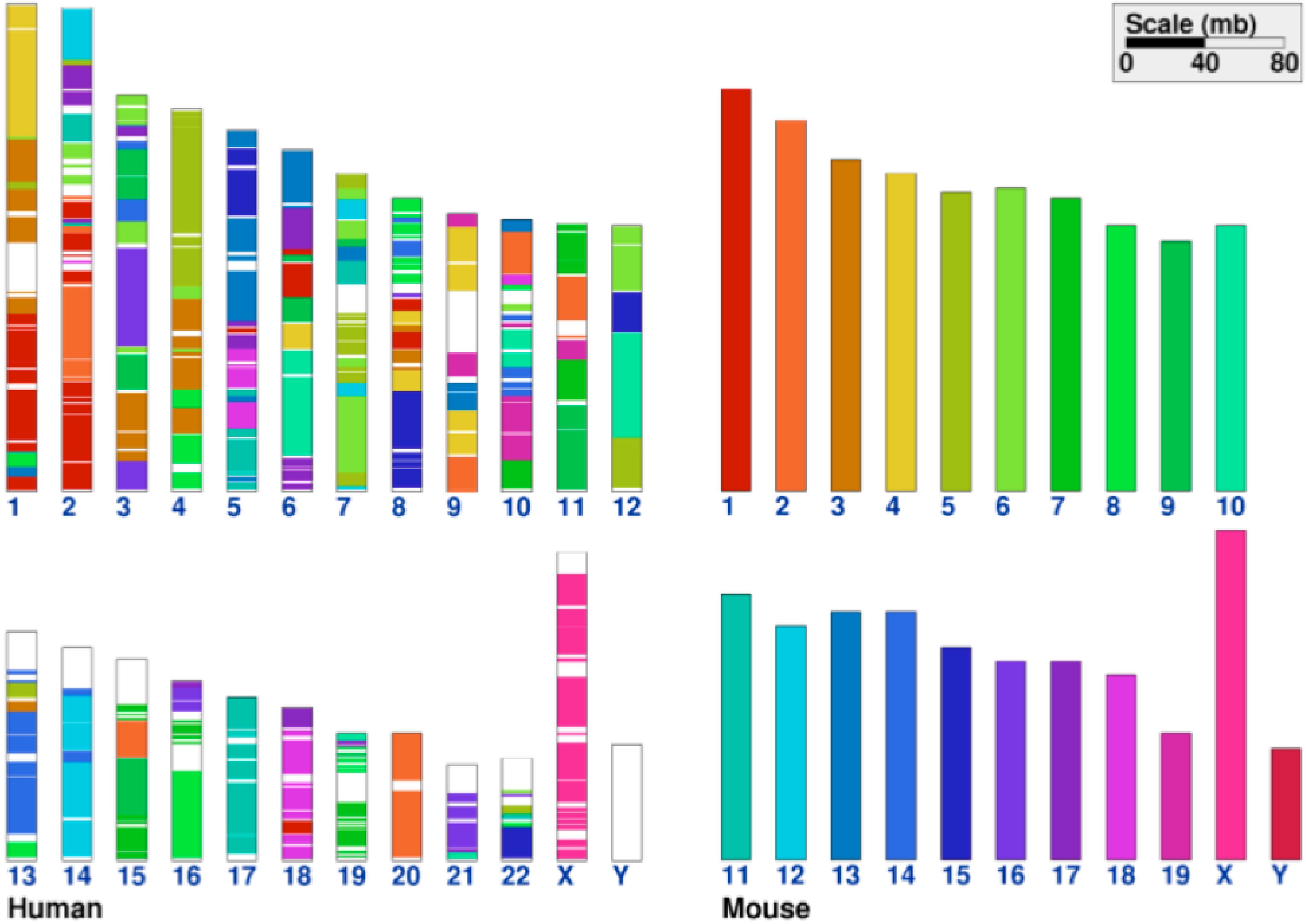
categorical

- color is great for categorical quantities!



distinguishability

- only good at 6 - 12 simultaneous colors

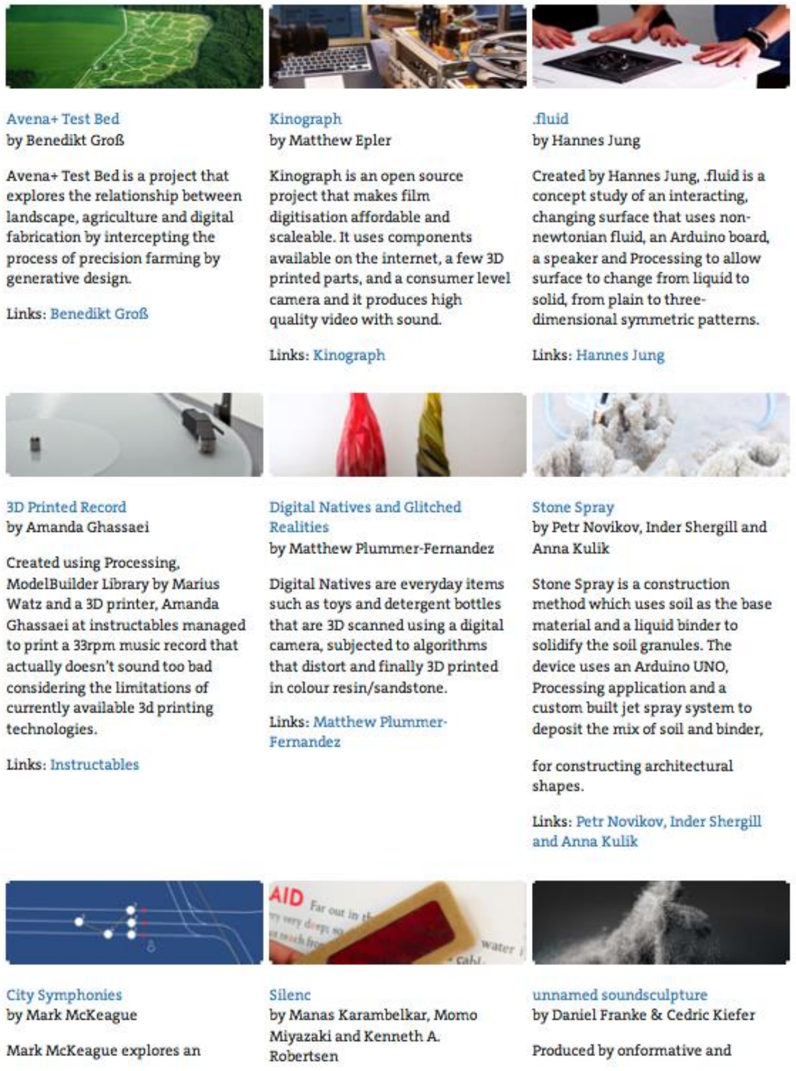


today . . .

Processing

what is it?

- programming environment
- visually oriented applications
- targets artists, designers, etc.



Avena+ Test Bed
by Benedikt Groß

Avena+ Test Bed is a project that explores the relationship between landscape, agriculture and digital fabrication by intercepting the process of precision farming by generative design.

Links: Benedikt Groß

Kinograph
by Matthew Epler

Kinograph is an open source project that makes film digitisation affordable and scaleable. It uses components available on the internet, a few 3D printed parts, and a consumer level camera and it produces high quality video with sound.

Links: Kinograph

.fluid
by Hannes Jung

Created by Hannes Jung, .fluid is a concept study of an interacting, changing surface that uses non-newtonian fluid, an Arduino board, a speaker and Processing to allow surface to change from liquid to solid, from plain to three-dimensional symmetric patterns.

Links: Hannes Jung

3D Printed Record
by Amanda Ghassaei

Created using Processing, ModelBuilder Library by Marius Watz and a 3D printer, Amanda Ghassaei at instructables managed to print a 33rpm music record that actually doesn't sound too bad considering the limitations of currently available 3d printing technologies.

Links: Instructables

Digital Natives and Glitched Realities
by Matthew Plummer-Fernandez

Digital Natives are everyday items such as toys and detergent bottles that are 3D scanned using a digital camera, subjected to algorithms that distort and finally 3D printed in colour resin/sandstone.

Links: Matthew Plummer-Fernandez

Stone Spray
by Petr Novikov, Inder Shergill and Anna Kulik

Stone Spray is a construction method which uses soil as the base material and a liquid binder to solidify the soil granules. The device uses an Arduino UNO, Processing application and a custom built jet spray system to deposit the mix of soil and binder,

for constructing architectural shapes.

Links: Petr Novikov, Inder Shergill and Anna Kulik

City Symphonies
by Mark McKeague

Mark McKeague explores an

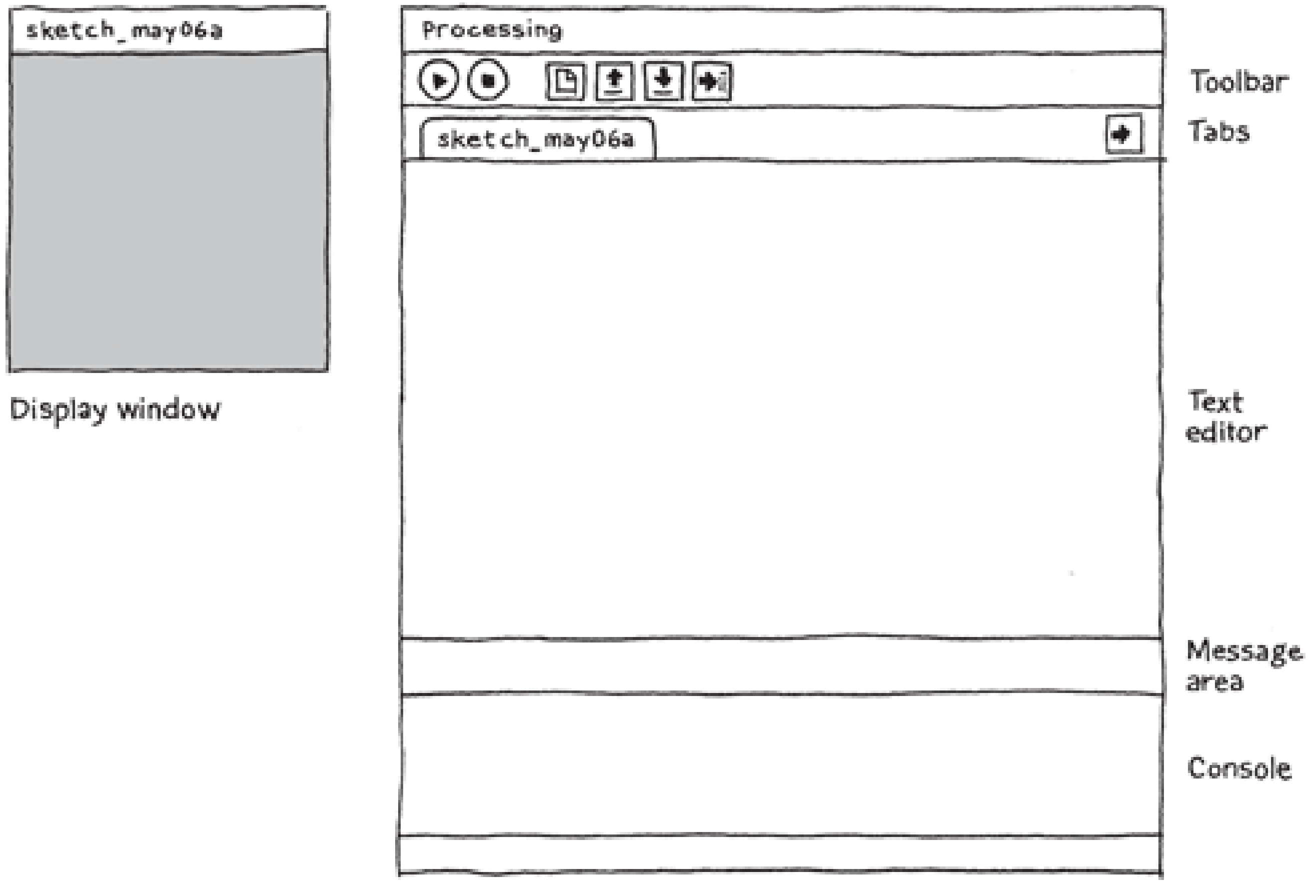
Silenc
by Manas Karambelkar, Momo Miyazaki and Kenneth A. Robertsen

unnamed soundsculpture
by Daniel Franke & Cedric Kiefer

Produced by onformative and

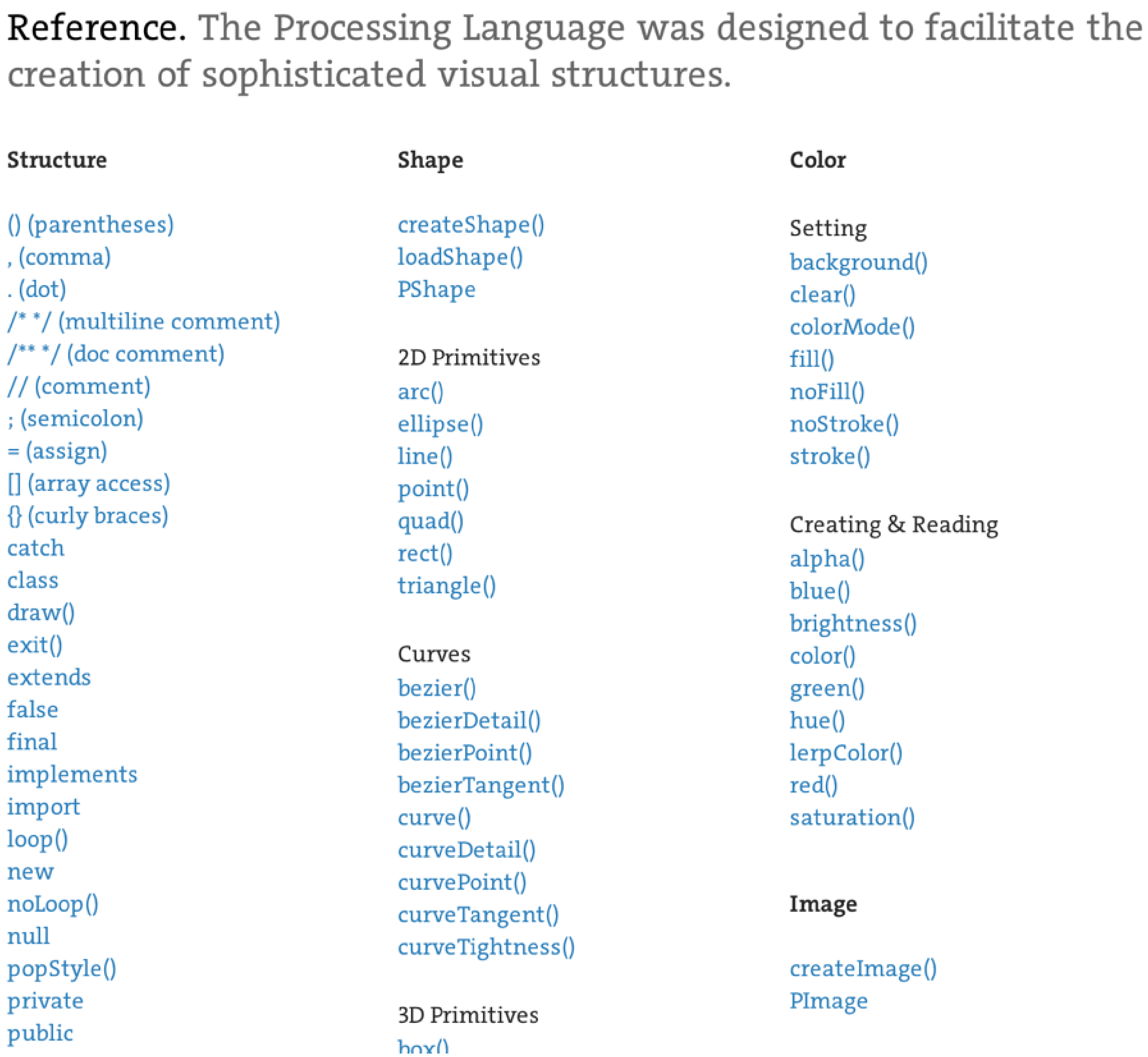
what is it?

- Processing Development Environment (PDE)



what is it?

- Processing API



Reference. The Processing Language was designed to facilitate the creation of sophisticated visual structures.

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

Shape

createShape()
loadShape()
PShape

2D Primitives
arc()
ellipse()
line()
point()
quad()
rect()
triangle()

Curves
bezier()
bezierDetail()
bezierPoint()
bezierTangent()
curve()
curveDetail()
curvePoint()
curveTangent()
curveTightness()

3D Primitives
box()

Color

Setting
background()
clear()
colorMode()
fill()
noFill()
noStroke()
stroke()

Creating & Reading
alpha()
blue()
brightness()
color()
green()
hue()
lerpColor()
red()
saturation()

Image

createImage()
PImage

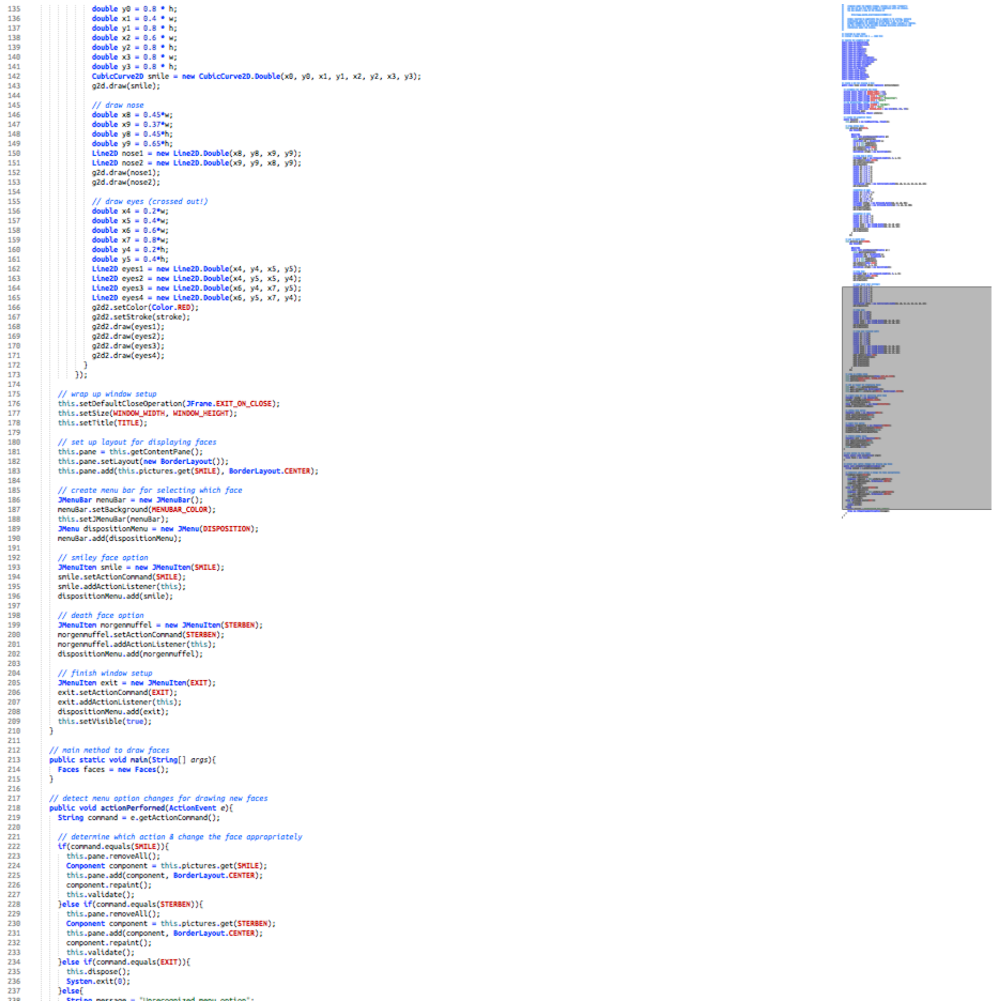
what is it?

- open-source, online community
  - http://forum.processing.org/
  - https://github.com/processing

why Processing?

why Processing?

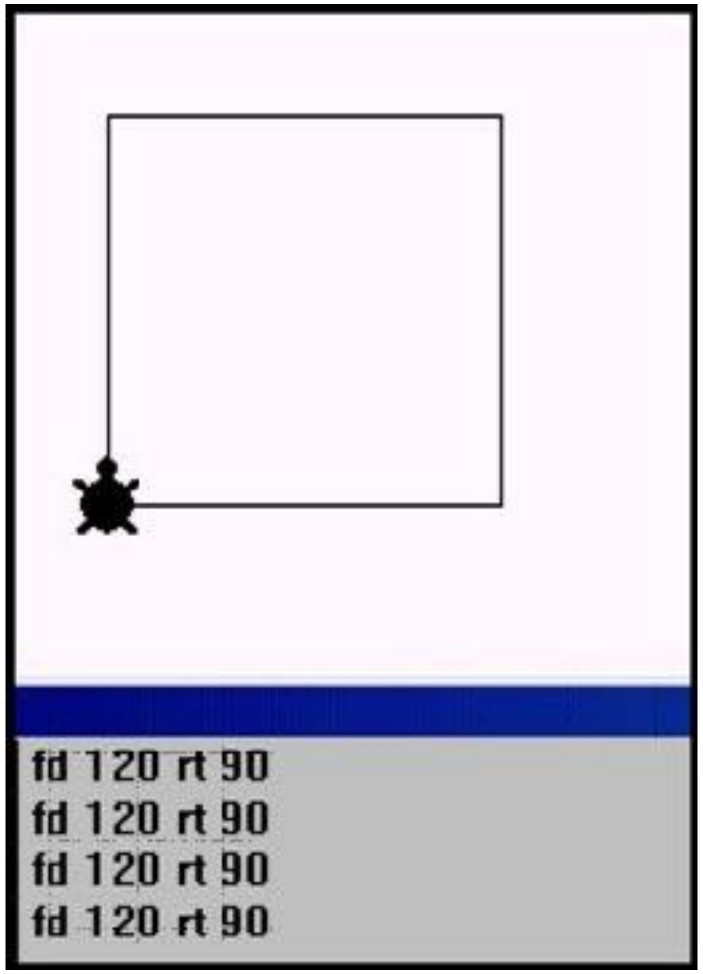
- difficulty to sketch with other languages
  - complicated setup
  - not easy to learn
  - repetitive code



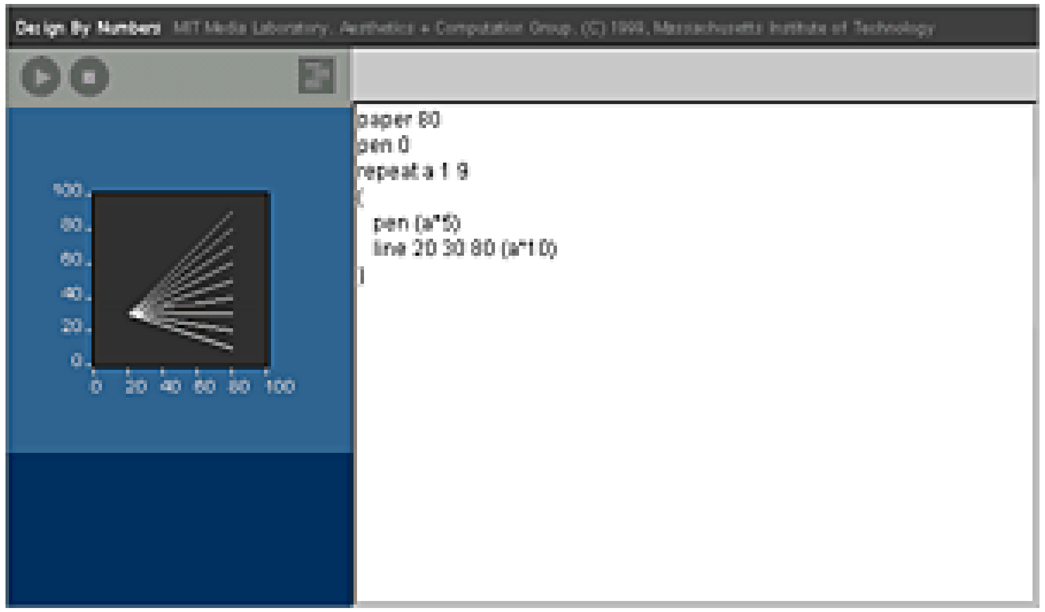
```
          double y0 = 0.8 * h;
          double x1 = 0.4 * w;
          double y1 = 0.8 * h;
          double x2 = 0.6 * w;
          double y2 = 0.8 * h;
          double x3 = 0.8 * w;
          double y3 = 0.8 * h;
          CubicCurve2D smile = new CubicCurve2D.Double(x0, y0, x1, y1, x2, y2, x3, y3);
          g2d.draw(smile);

          // draw nose
          double x8 = 0.45*w;
          double x9 = 0.37*w;
          double y8 = 0.45*h;
          double y9 = 0.65*h;
          Line2D nose1 = new Line2D.Double(x8, y8, x9, y9);
          Line2D nose2 = new Line2D.Double(x9, y9, x8, y9);
          g2d.draw(nose1);
          g2d.draw(nose2);

          // draw eyes (crossed out!)
          double x4 = 0.2*w;
          double x5 = 0.4*w;
          double x6 = 0.6*w;
          double x7 = 0.8*w;
          double y4 = 0.2*h;
          double y5 = 0.4*h;
          Line2D eyes1 = new Line2D.Double(x4, y4, x5, y5);
          Line2D eyes2 = new Line2D.Double(x4, y5, x5, y4);
          Line2D eyes3 = new Line2D.Double(x6, y4, x7, y5);
          Line2D eyes4 = new Line2D.Double(x6, y5, x7, y4);
          g2d2.setColor(Color.RED);
          g2d2.setStroke(stroke);
          g2d2.draw(eyes1);
          g2d2.draw(eyes2);
          g2d2.draw(eyes3);
          g2d2.draw(eyes4);
        }
      });

    // wrap up window setup
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    this.setSize(WINDOW_WIDTH, WINDOW_HEIGHT);
    this.setTitle(TITLE);

    // set up layout for displaying faces
    this.pane = this.getContentPane();
    this.pane.setLayout(new BorderLayout());
    this.pane.add(this.pictures.get(SMILE), BorderLayout.CENTER);

    // create menu bar for selecting which face
    JMenuBar menuBar = new JMenuBar();
    menuBar.setBackground(MENUBAR_COLOR);
    this.setJMenuBar(menuBar);
    JMenu dispositionMenu = new JMenu(DISPOSITION);
    menuBar.add(dispositionMenu);

    // smiley face option
    JMenuItem smile = new JMenuItem(SMILE);
    smile.setActionCommand(SMILE);
    smile.addActionListener(this);
    dispositionMenu.add(smile);

    // death face option
    JMenuItem morgenmuffel = new JMenuItem(STERBEN);
    morgenmuffel.setActionCommand(STERBEN);
    morgenmuffel.addActionListener(this);
    dispositionMenu.add(morgenmuffel);

    // finish window setup
    JMenuItem exit = new JMenuItem(EXIT);
    exit.setActionCommand(EXIT);
    exit.addActionListener(this);
    dispositionMenu.add(exit);
    this.setVisible(true);
  }

  // main method to draw faces
  public static void main(String[] args){
    Faces faces = new Faces();
  }

  // detect menu option changes for drawing new faces
  public void actionPerformed(ActionEvent e){
    String command = e.getActionCommand();

    // determine which action & change the face appropriately
    if(command.equals(SMILE)){
      this.pane.removeAll();
      Component component = this.pictures.get(SMILE);
      this.pane.add(component, BorderLayout.CENTER);
      component.repaint();
      this.validate();
    }else if(command.equals(STERBEN)){
      this.pane.removeAll();
      Component component = this.pictures.get(STERBEN);
      this.pane.add(component, BorderLayout.CENTER);
      component.repaint();
      this.validate();
    }else if(command.equals(EXIT)){
      this.dispose();
      System.exit(0);
    }else{
      String message = "Unrecognized menu option";
```

why Processing?

- based on:
  - Logo
  - Design by Numbers



```
fd 120 rt 90
fd 120 rt 90
fd 120 rt 90
fd 120 rt 90
```

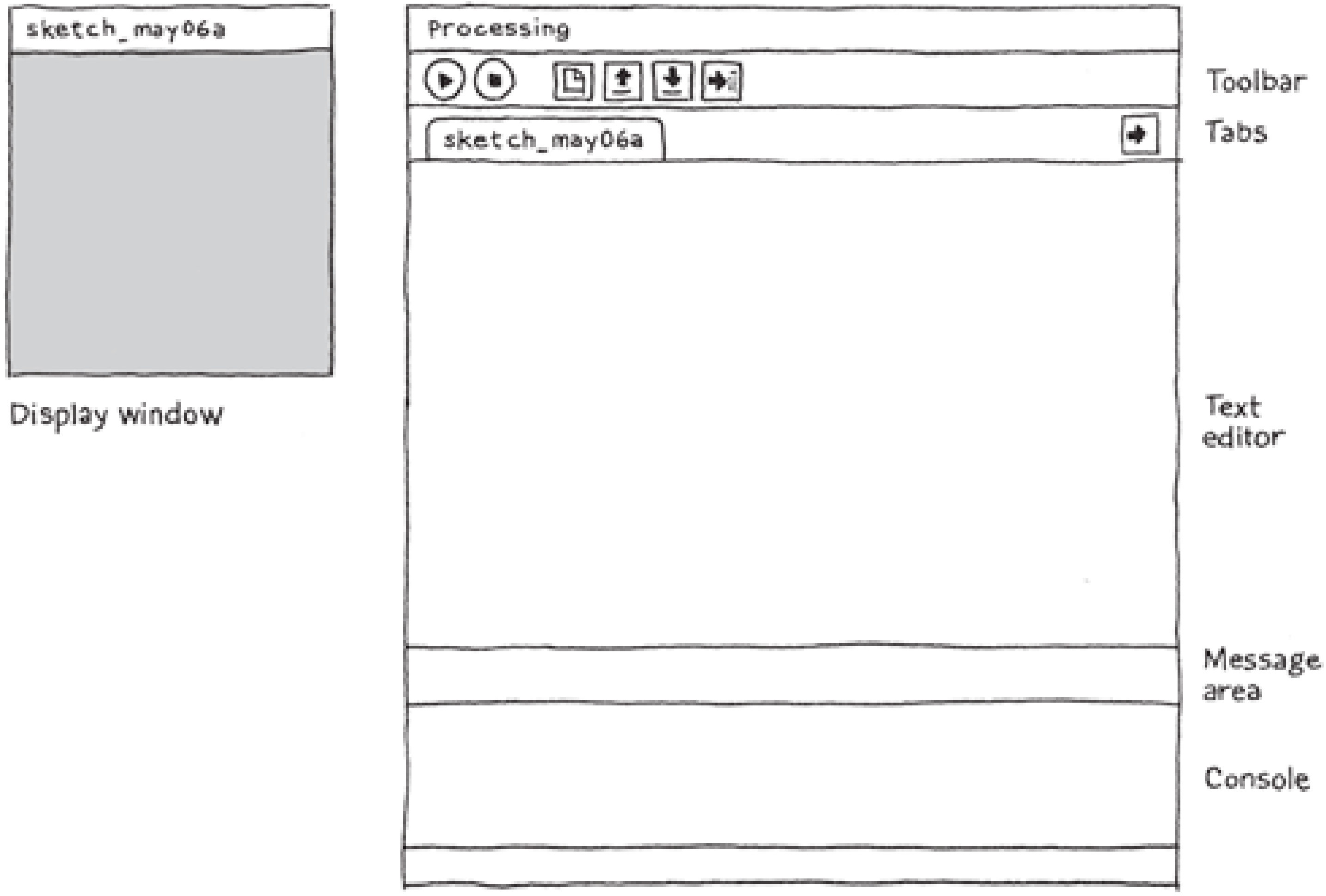


Design By Numbers MIT Media Laboratory, Aesthetics + Computation Group, (C) 1999, Massachusetts Institute of Technology

```
paper 80
pen 0
repeat a 1 9
{
  pen (a*5)
  line 20 30 80 (a*10)
}
```

why Processing?

- program = sketch



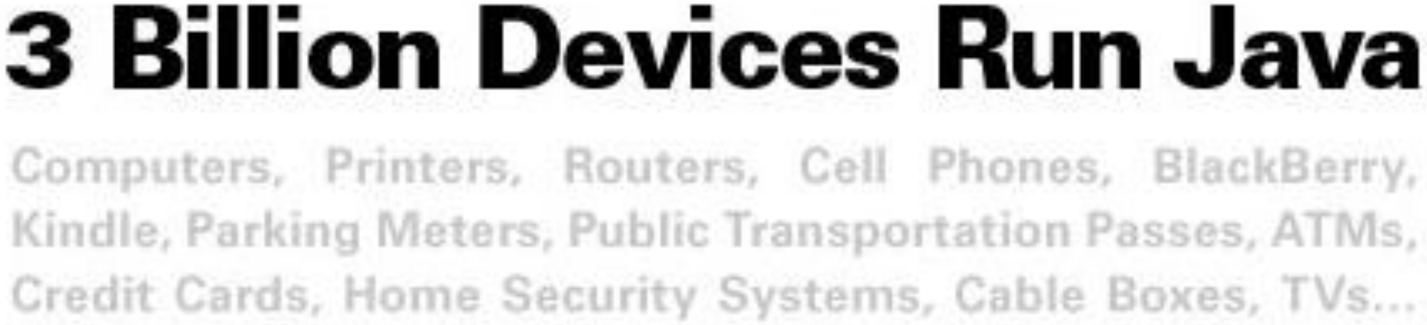
why Processing?

- programming syntax

```
void setup() {
  size(480, 120);
}

void draw() {
  if (mousePressed) {
    fill(0);
  } else {
    fill(255);
  }
  ellipse(mouseX, mouseY, 80, 80);
}
```

why Processing?

- Java-based
  - - complexity
  - + Big standard library
  - + lots of user-contributed libraries
- similar syntax & portability



3 Billion Devices Run Java

Computers, Printers, Routers, Cell Phones, BlackBerry, Kindle, Parking Meters, Public Transportation Passes, ATMs, Credit Cards, Home Security Systems, Cable Boxes, TVs...

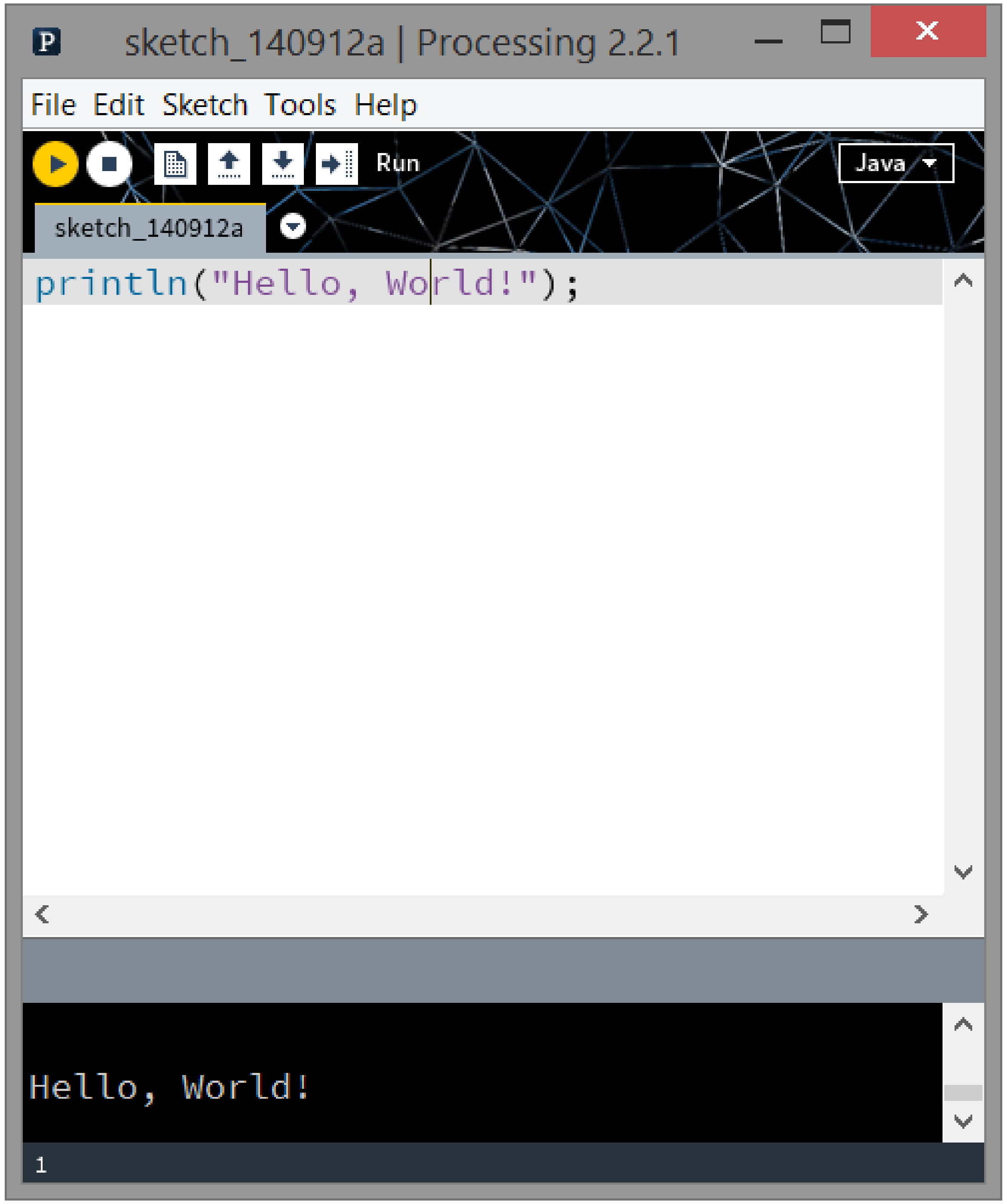
why Processing?

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

```
javac Hello.java
java Hello
```

why Processing?

- println(“Hello, World!”);

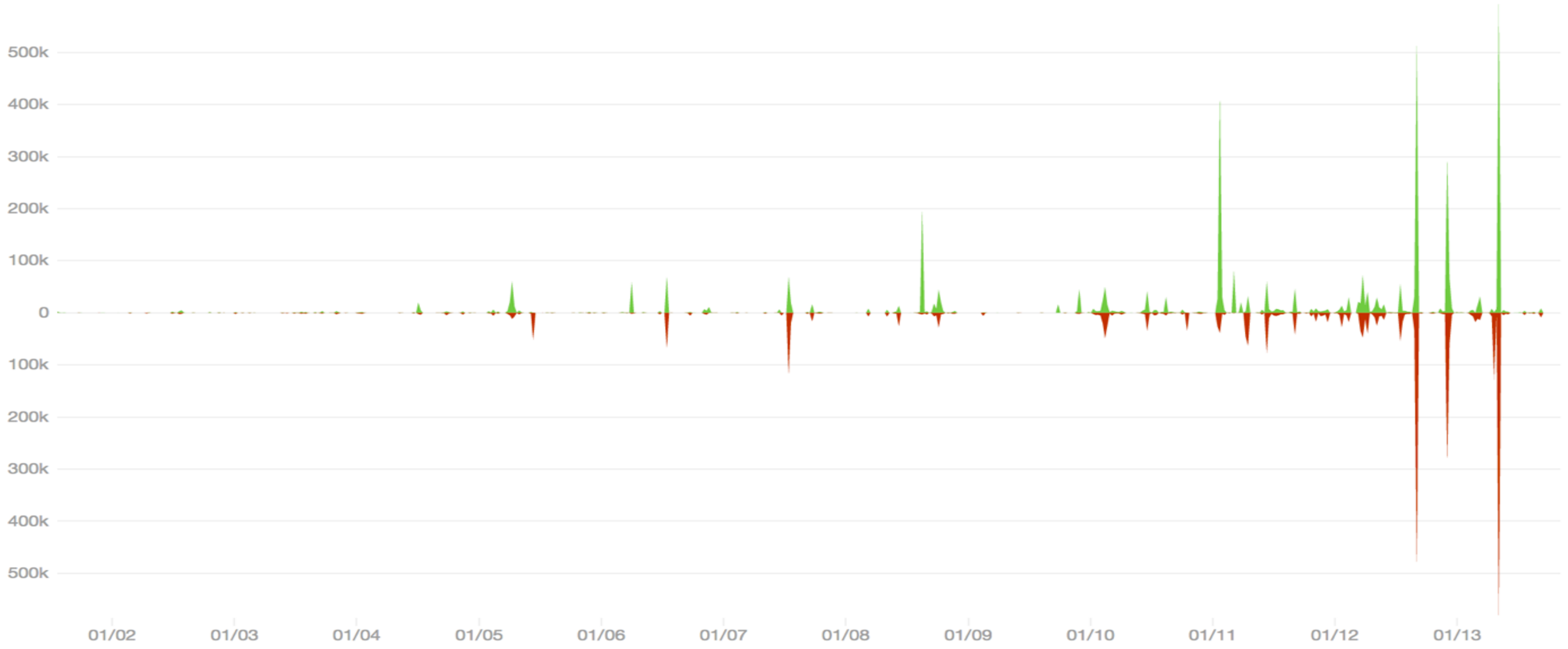

sketch_140912a | Processing 2.2.1
File Edit Sketch Tools Help
Run
Java
sketch_140912a
1

```
println("Hello, World!");
```

```
Hello, World!
```

why Processing?

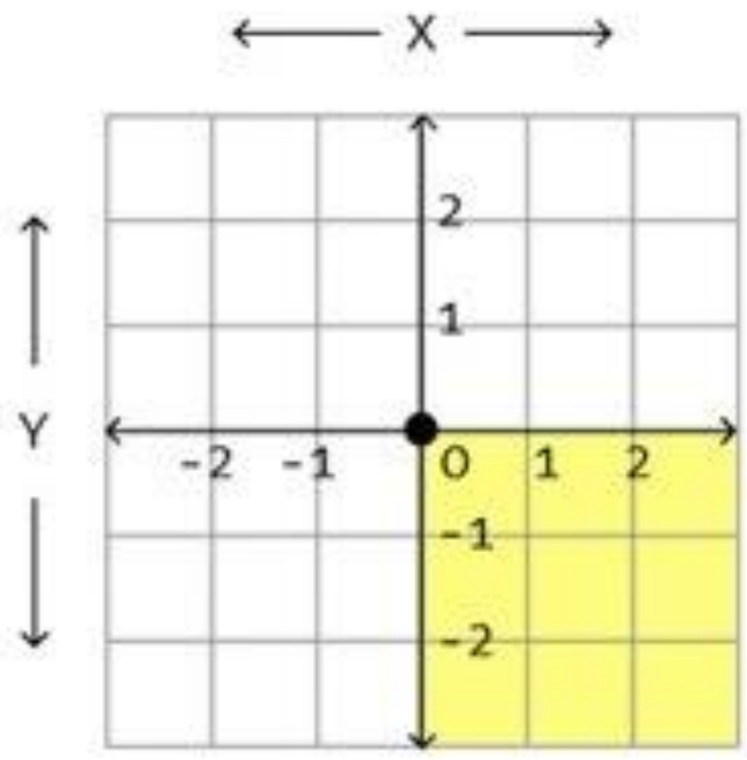
- active development



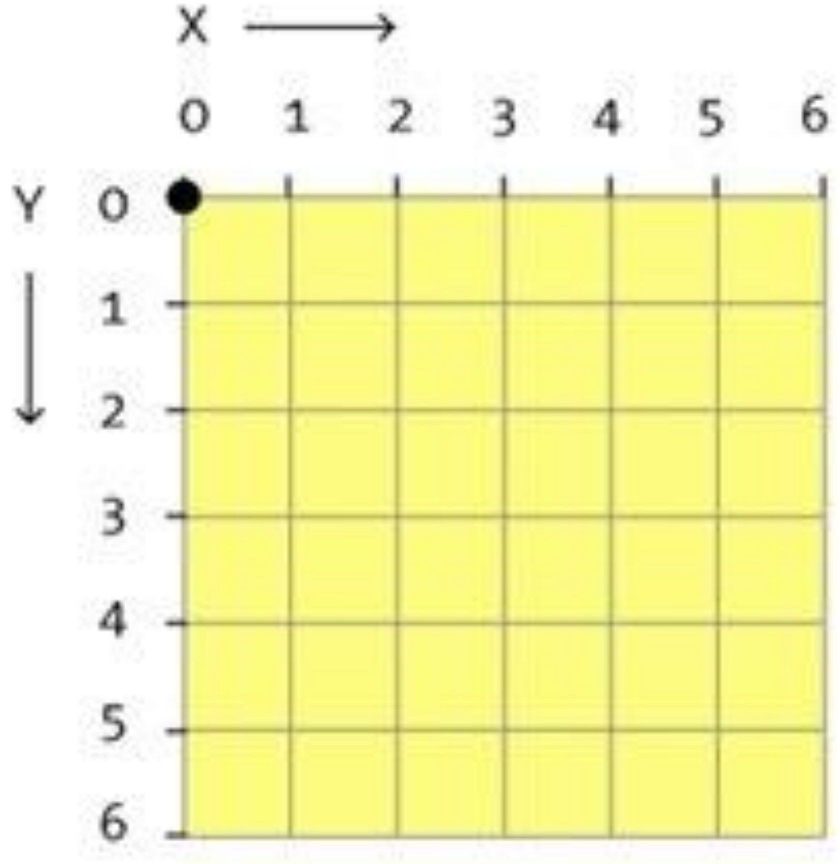
graphics

monitors

- grid of pixels



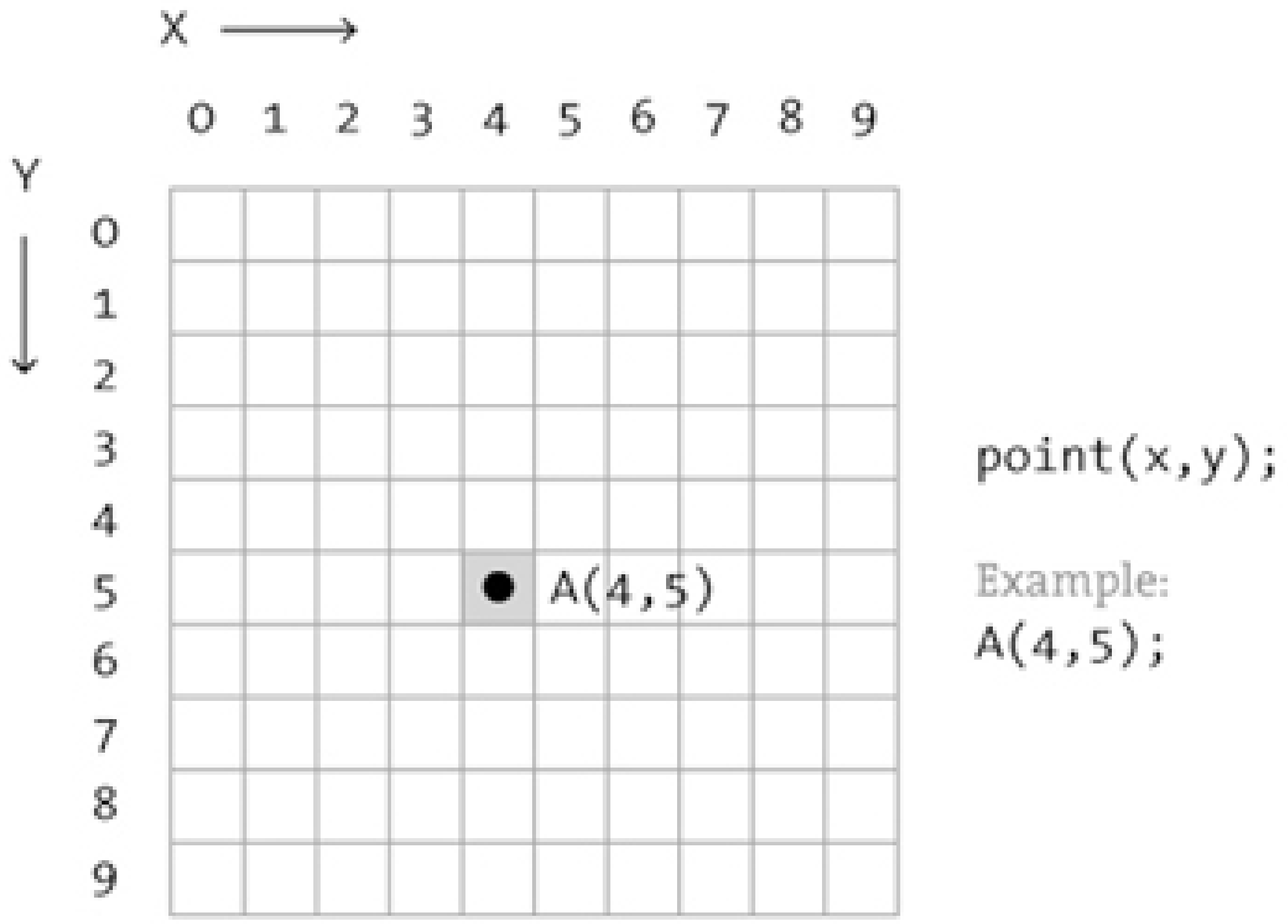
Eighth Grade



Computer

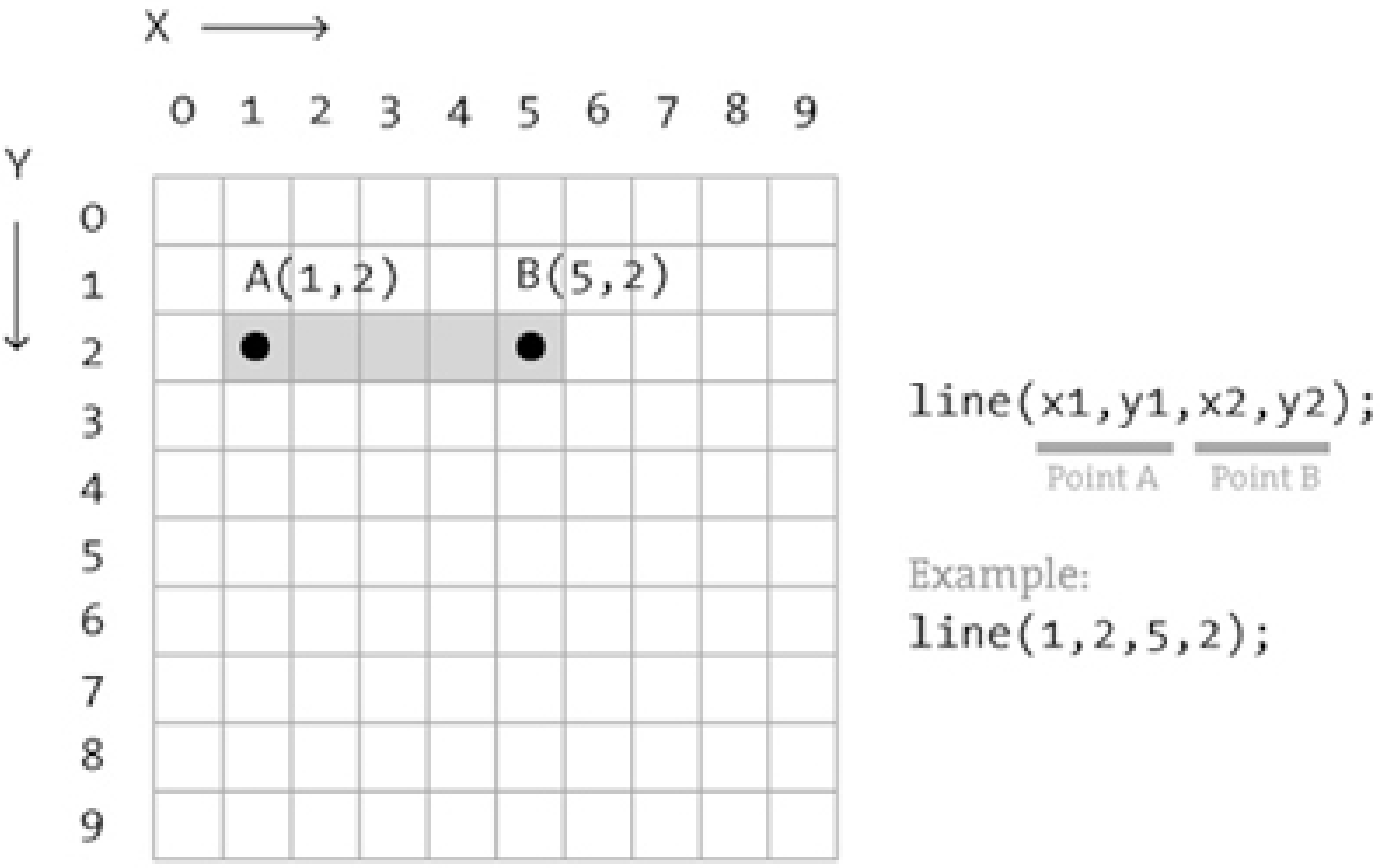
shape

```
point(x, y);
```



shape

```
line(x1, y1, x2, y2);
```



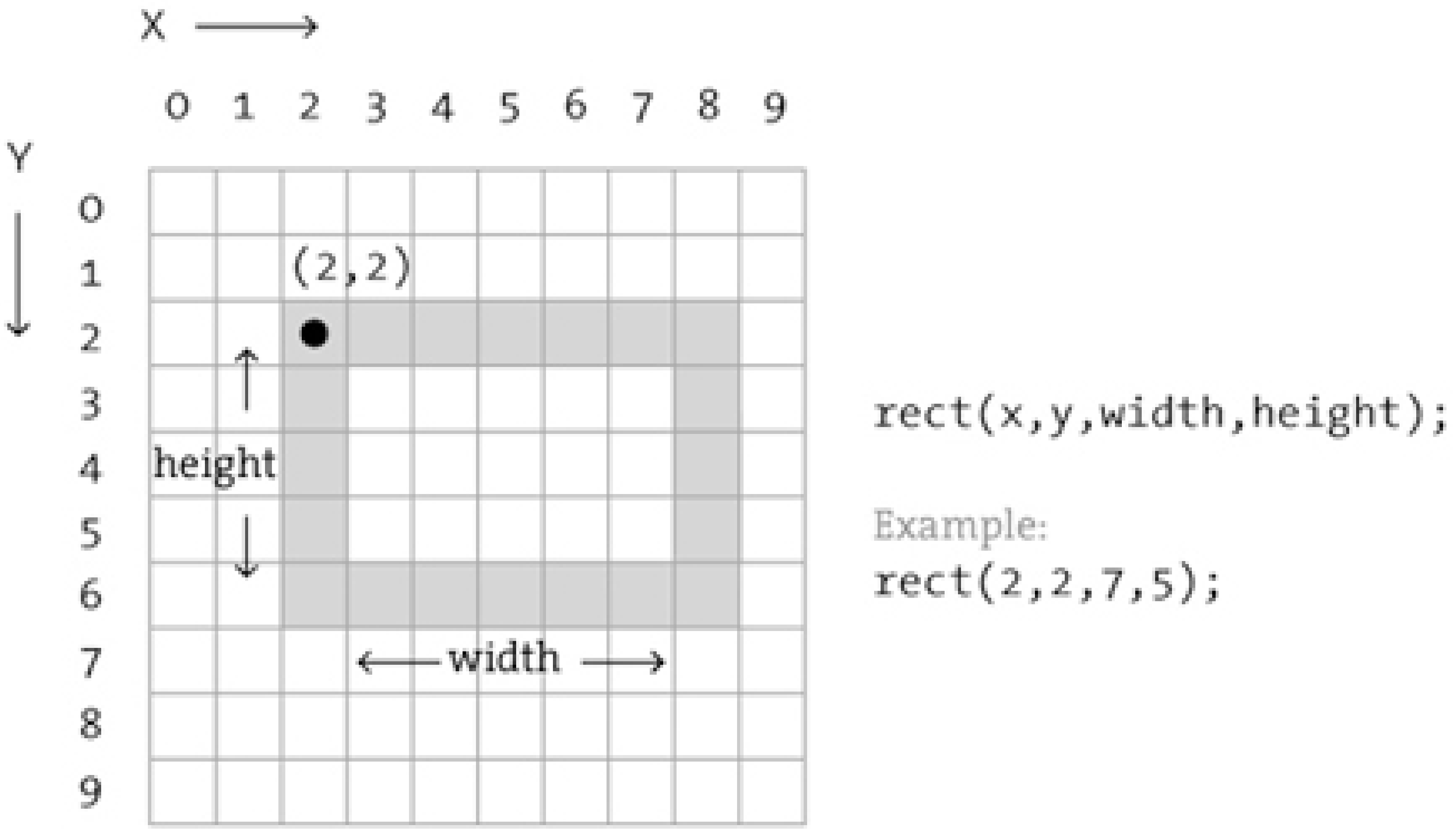
```
line(x1,y1,x2,y2);
```

Example:

```
line(1,2,5,2);
```

shape

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



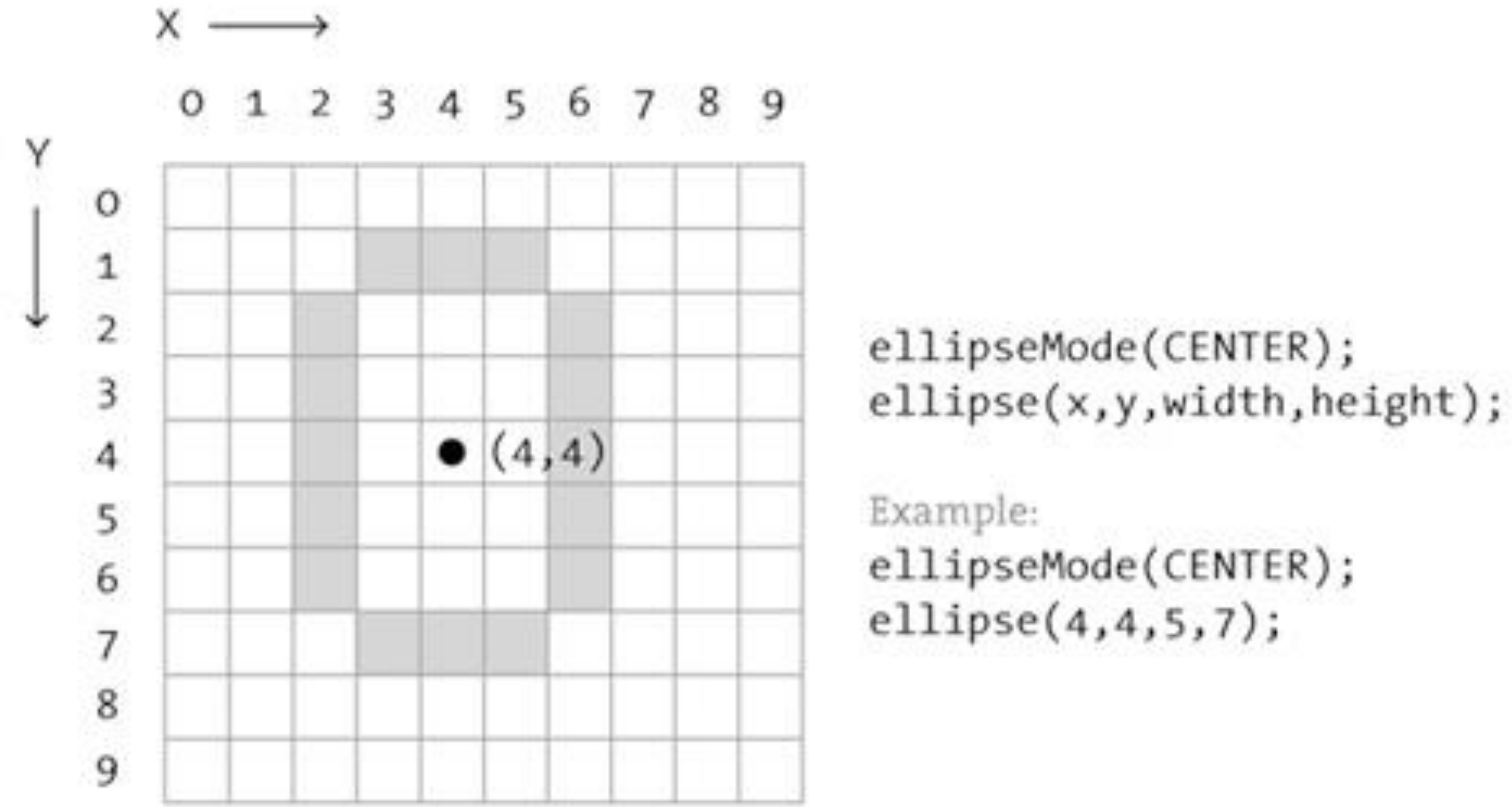
```
rect(x,y,width,height);
```

Example:

```
rect(2,2,7,5);
```

shape

```
ellipseMode(CENTER);
ellipse(x, y, width, height);
```



```
ellipseMode(CENTER);
ellipse(x,y,width,height);
```

Example:

```
ellipseMode(CENTER);
ellipse(4,4,5,7);
```

shape

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

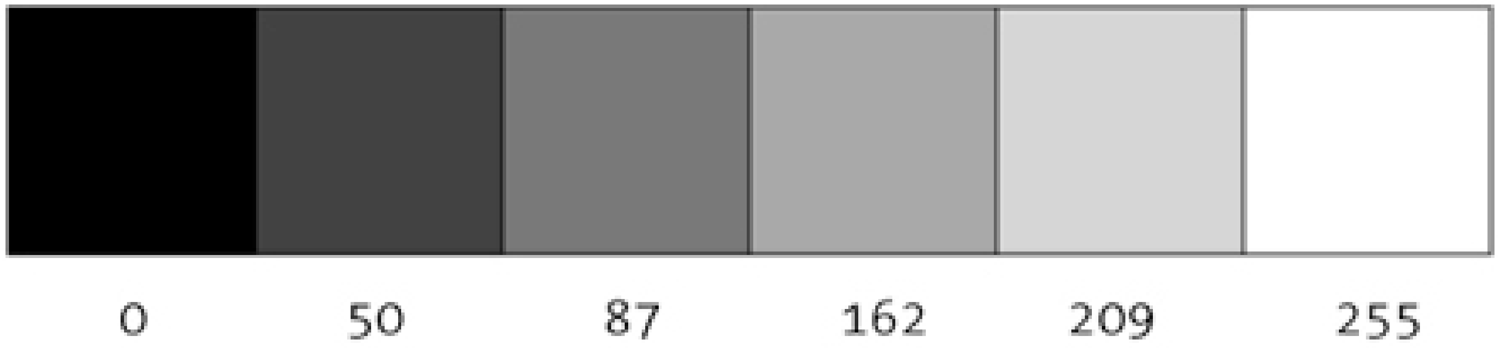
```
quad(x1, y1, x2, y2, x3, y3, x4, y4);
```

```
arc(x, y, width, height, start, stop);
```

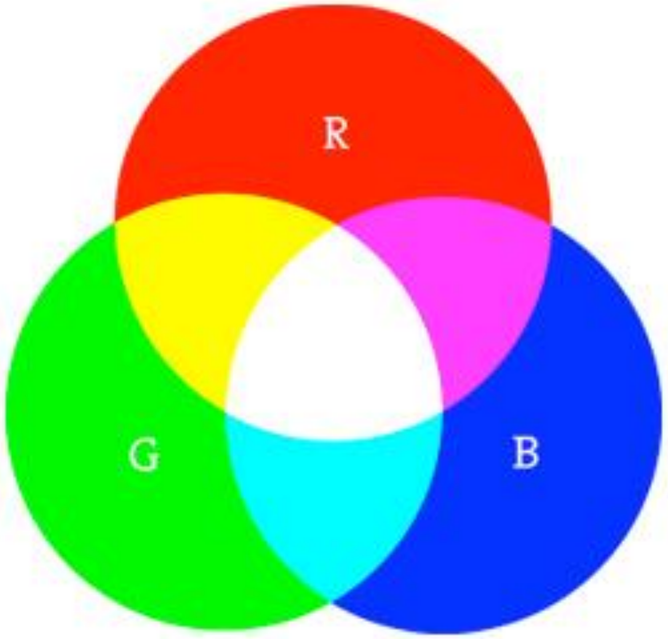
color

- luminance

```
background(255);
```



color



- RGB (default)

```
color c1 = color(r, g, b);


color c2 = #RRGGBB;
```

color

- RGBA:
  - a = alpha / transparency / opacity
  - 0 = transparent; 255 =opaque (solid)

```
color c1 = color(r, g, b, a);
```

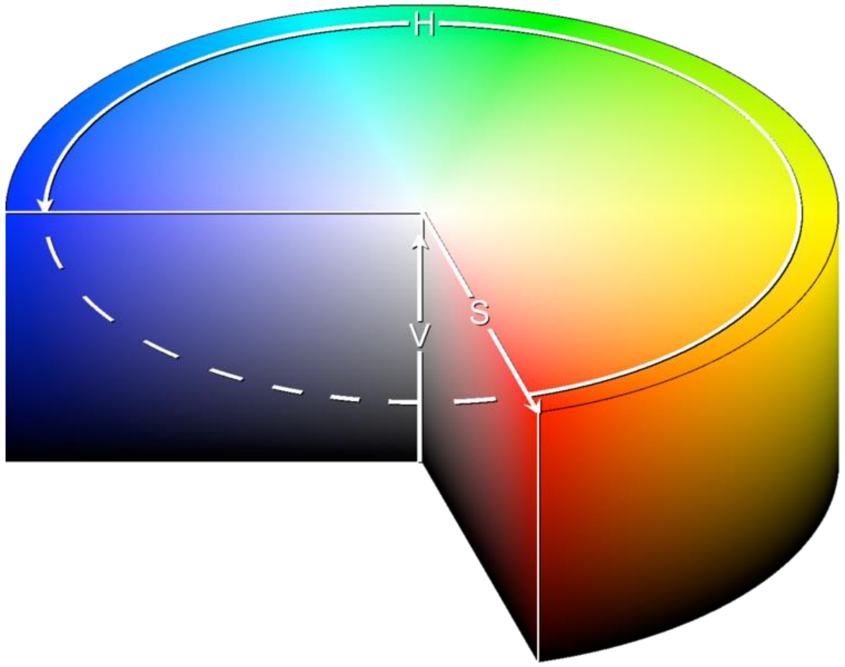
color

- color modes
  - custom range

```
colorMode(RGB, 100);
```

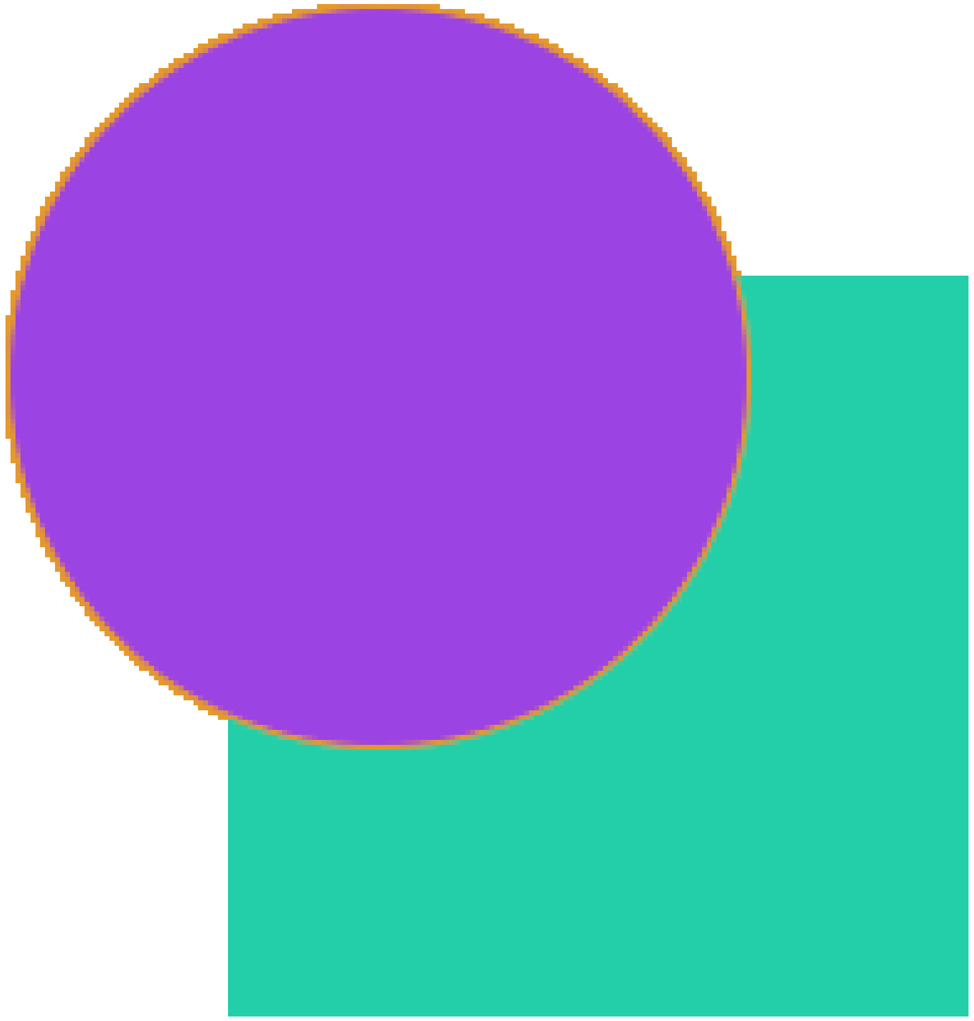
  - HSB

```
colorMode(HSB);
```



properties

```
noStroke();
fill(c1);
rect(...);

fill(c2);
stroke(c3);
ellipse(...);
```



properties

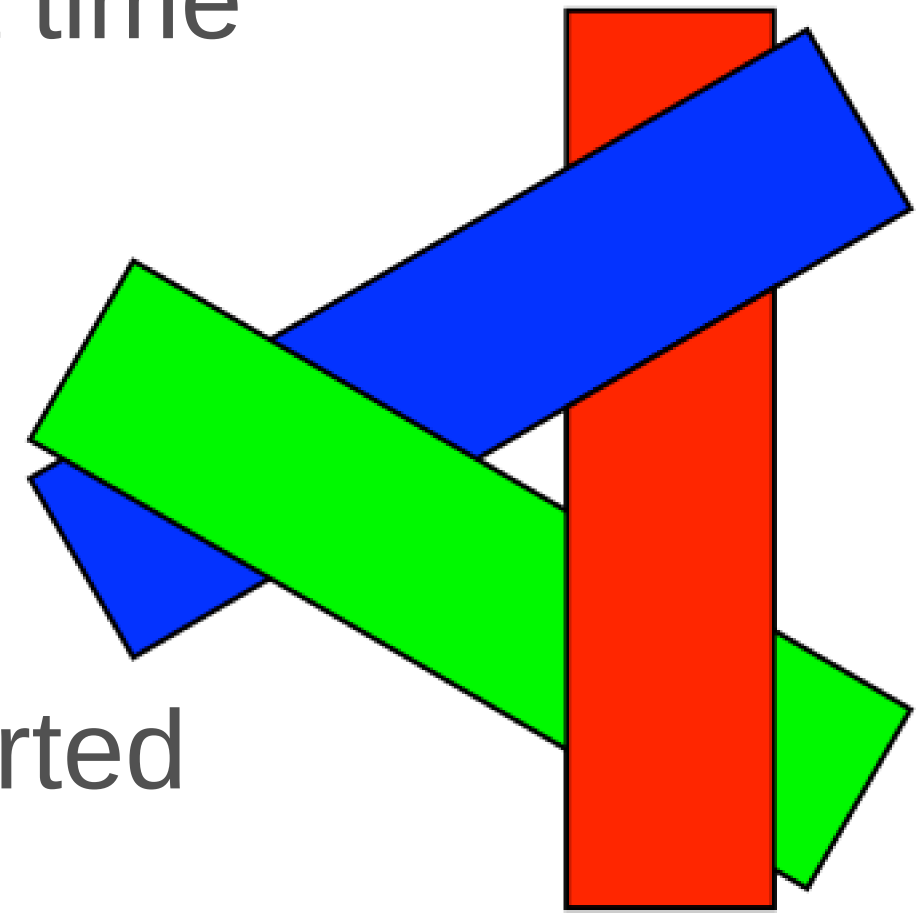
```
no
no
el



re
```

order

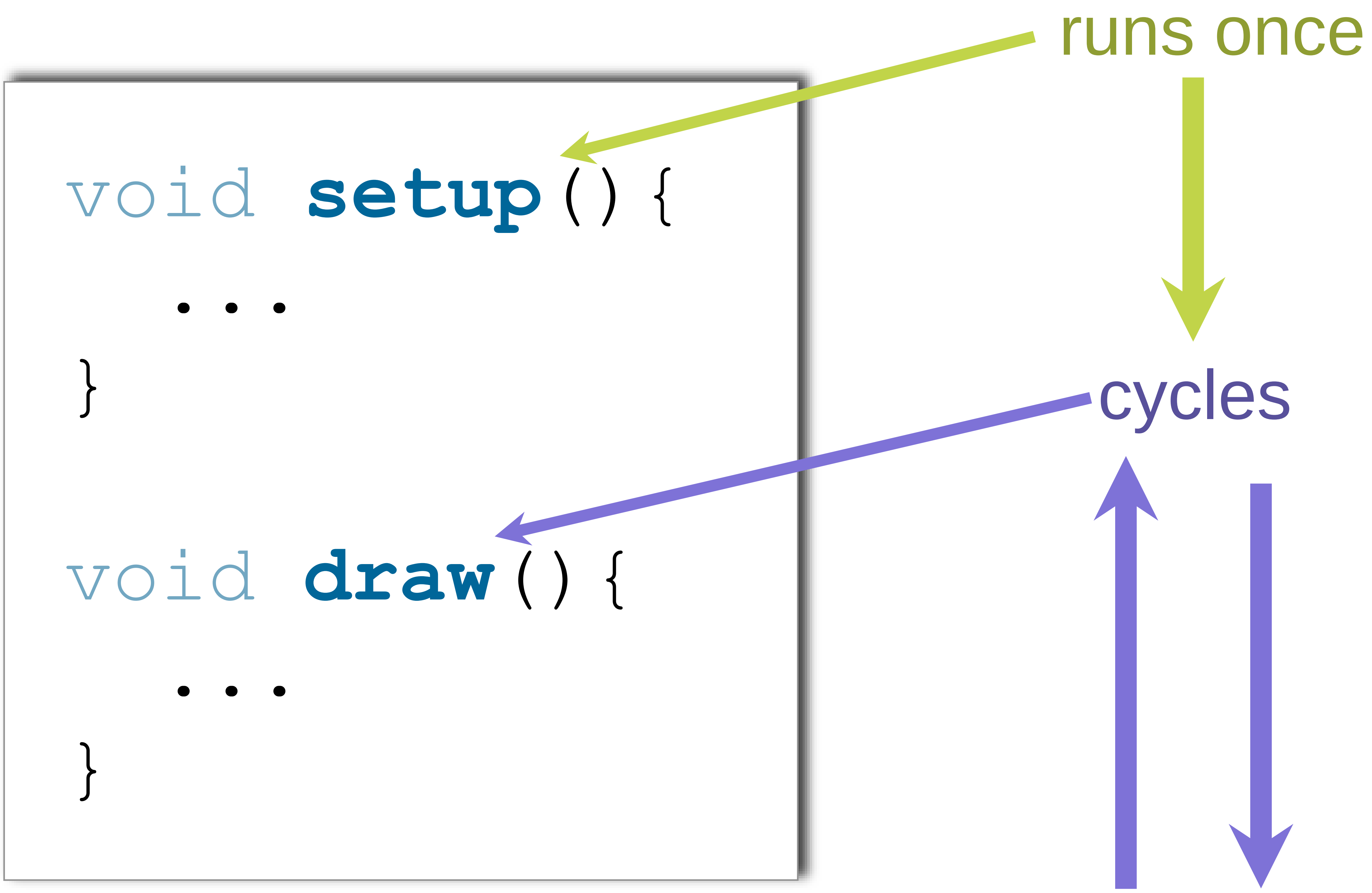
- shapes are painted one at a time

- overlap can occur

- some shapes are not supported



animation



```
void setup() {
   ...
}

void draw() {
   ...
}
```

runs once

cycles

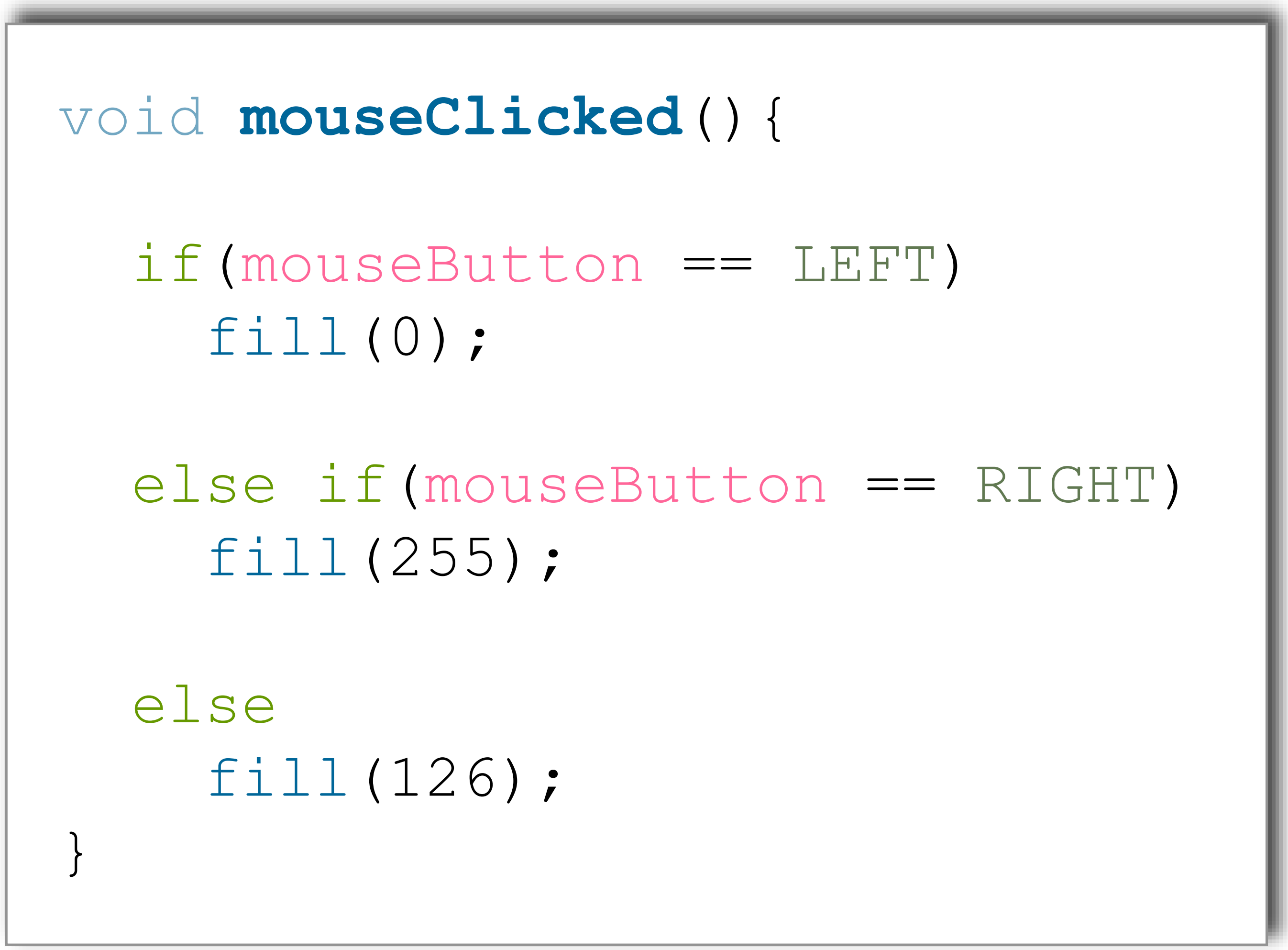
text

```
// in setup()
PFont myFont;
myFont = createFont("Georgia", 32);


// in draw()
textFont(myFont);
textAlign(CENTER, CENTER);
text("Hello, World!", width/2, height/2);
```

programming

interaction

- mouse



```
void mouseClicked(){

  if(mouseButton == LEFT)
    fill(0);

  else if(mouseButton == RIGHT)
    fill(255);

  else
    fill(126);
}
```

```
void mousePressed()

void mouseReleased()

void mouseClicked()

void mouseDragged()

void mouseMoved()

void mouseWheel()
```

```
mouseX
mouseY
pmouseX
pmouseY
```

interaction

- keyboard

```
void keyTyped(){

  if(key == 'b')
    fill(0);

  else if(key == 'w')
    fill(255);

  else
    fill(126);
}
```

```
void keyPressed()

void keyReleased()

void keyTyped()
```

```
keyPressed
key
keyCode
```

structure

- comments, variables, arrays, loops

- ArrayList (also FloatList, IntList, StringList)

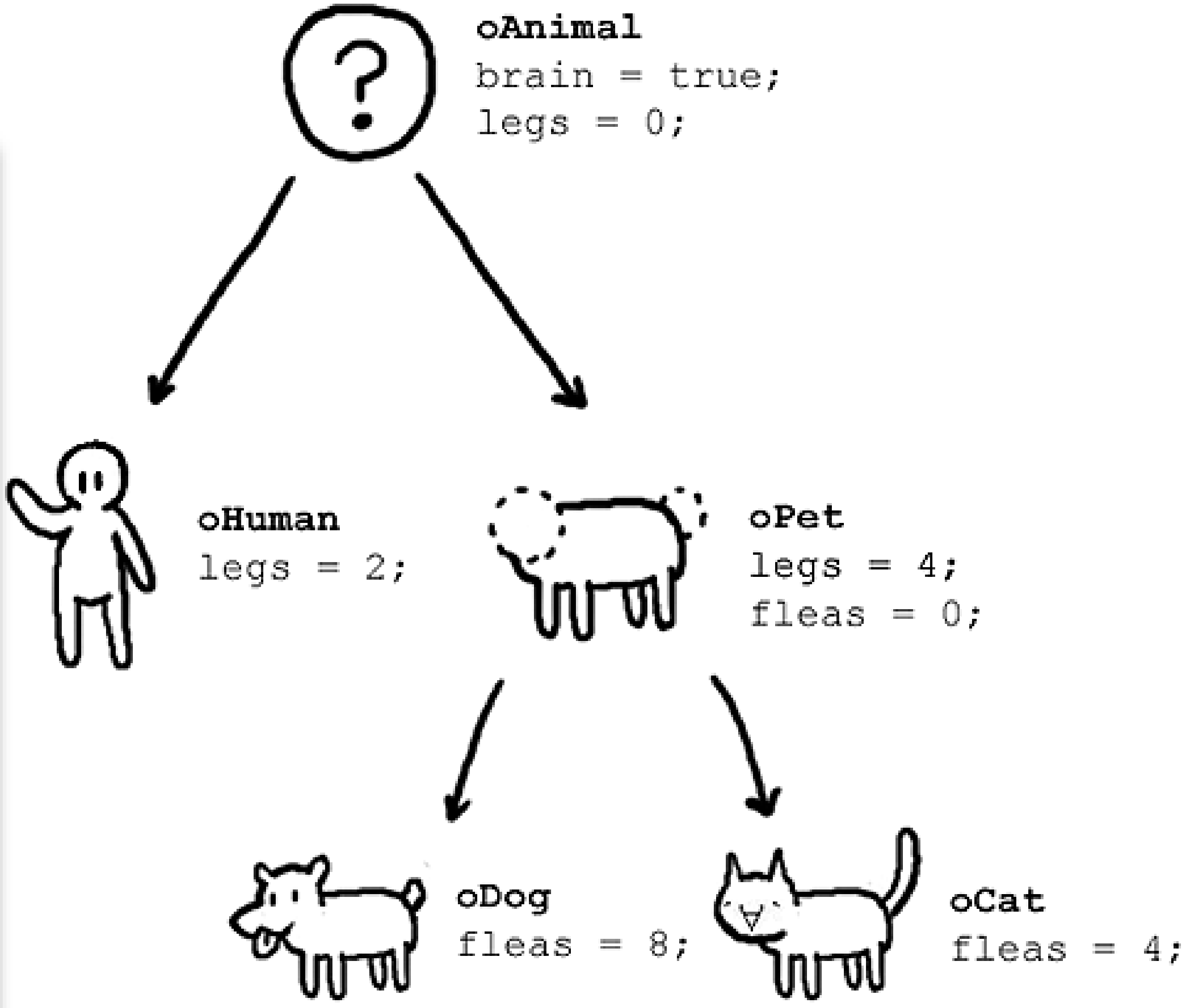
- HashMap (dict: also FloatDict, IntDict, StringDict)

- Table, XML, JSON

object-oriented

- with classes

```
class oAnimal{
  boolean brain;
  int legs;

  oAnimal(){
    brain = true;
    legs = 0;
  }
}
```



```
oAnimal
brain = true;
legs = 0;
```

```
oHuman
legs = 2;
```

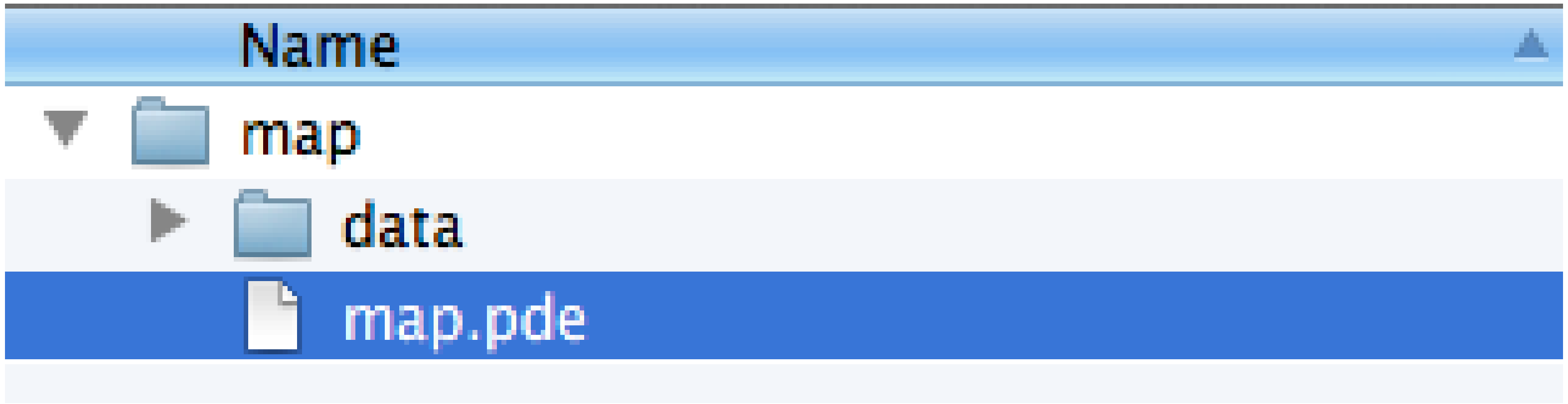
```
oPet
legs = 4;
fleas = 0;
```

```
oDog
fleas = 8;
```

```
oCat
fleas = 4;
```

folder structure

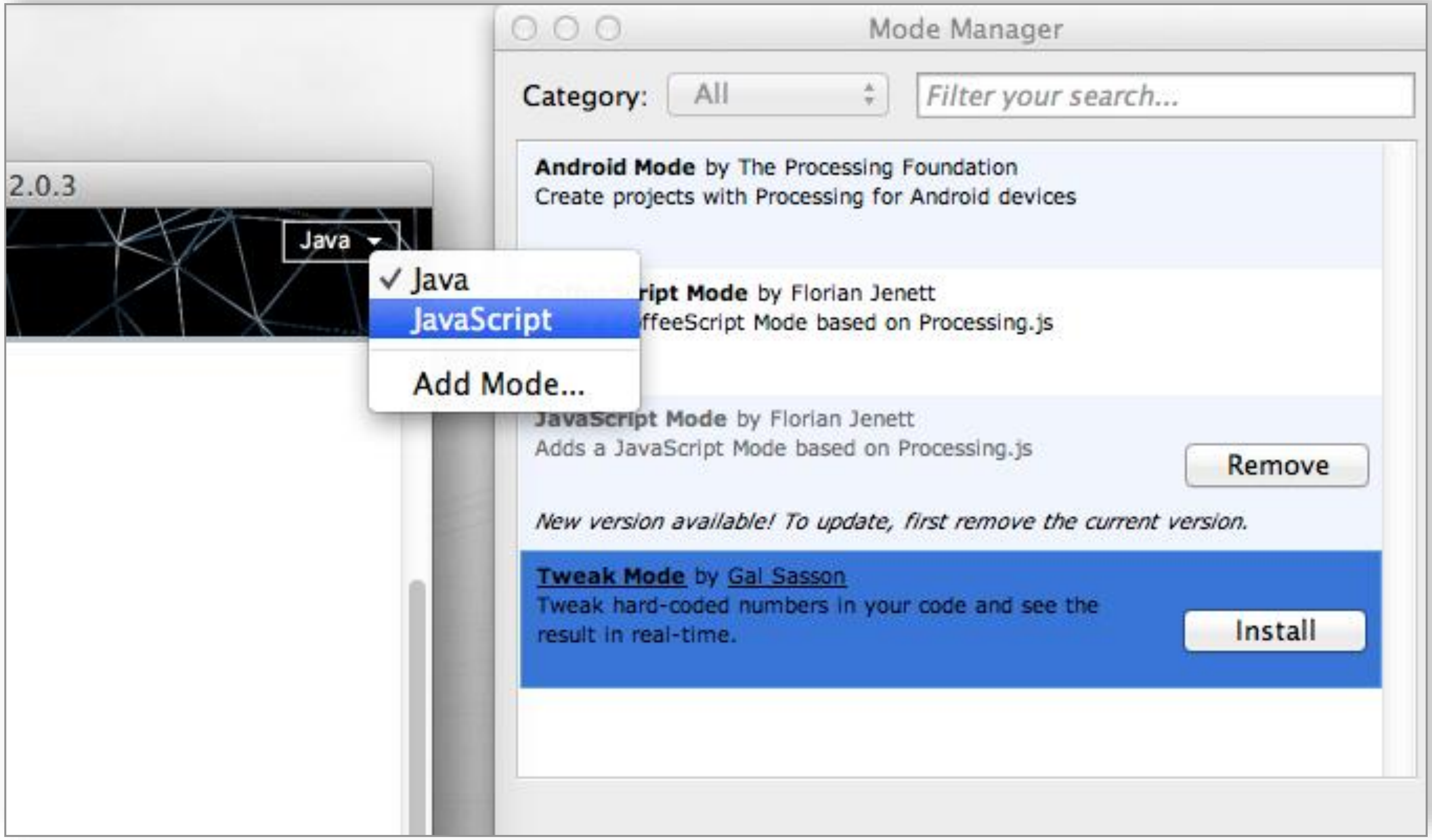
- folder *[NAME]* & *[NAME]*.pde must match



- optional data folder (for images, input)

modes

- Java (default), JavaScript, Android, etc.



Android Mode by The Processing Foundation
Create projects with Processing for Android devices

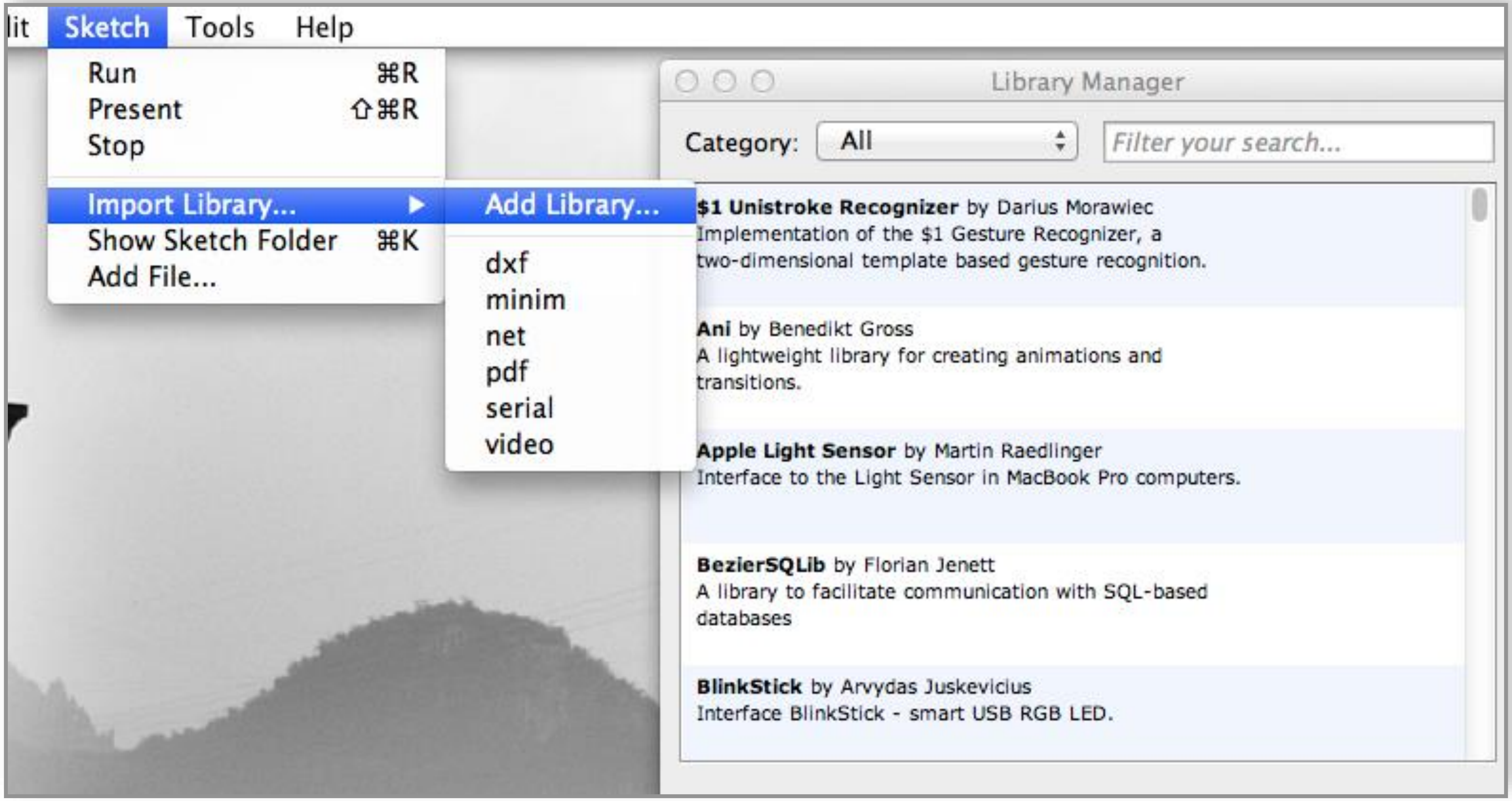
ript Mode by Florian Jenett
ffeeScript Mode based on Processing.js

JavaScript Mode by Florian Jenett
Adds a JavaScript Mode based on Processing.js

New version available! To update, first remove the current version.

Tweak Mode by Gal Sasson
Tweak hard-coded numbers in your code and see the result in real-time.

libraries



$1 Unistroke Recognizer by Darius Morawiec
Implementation of the $1 Gesture Recognizer, a two-dimensional template based gesture recognition.

Ani by Benedikt Gross
A lightweight library for creating animations and transitions.

Apple Light Sensor by Martin Raedlinger
Interface to the Light Sensor in MacBook Pro computers.

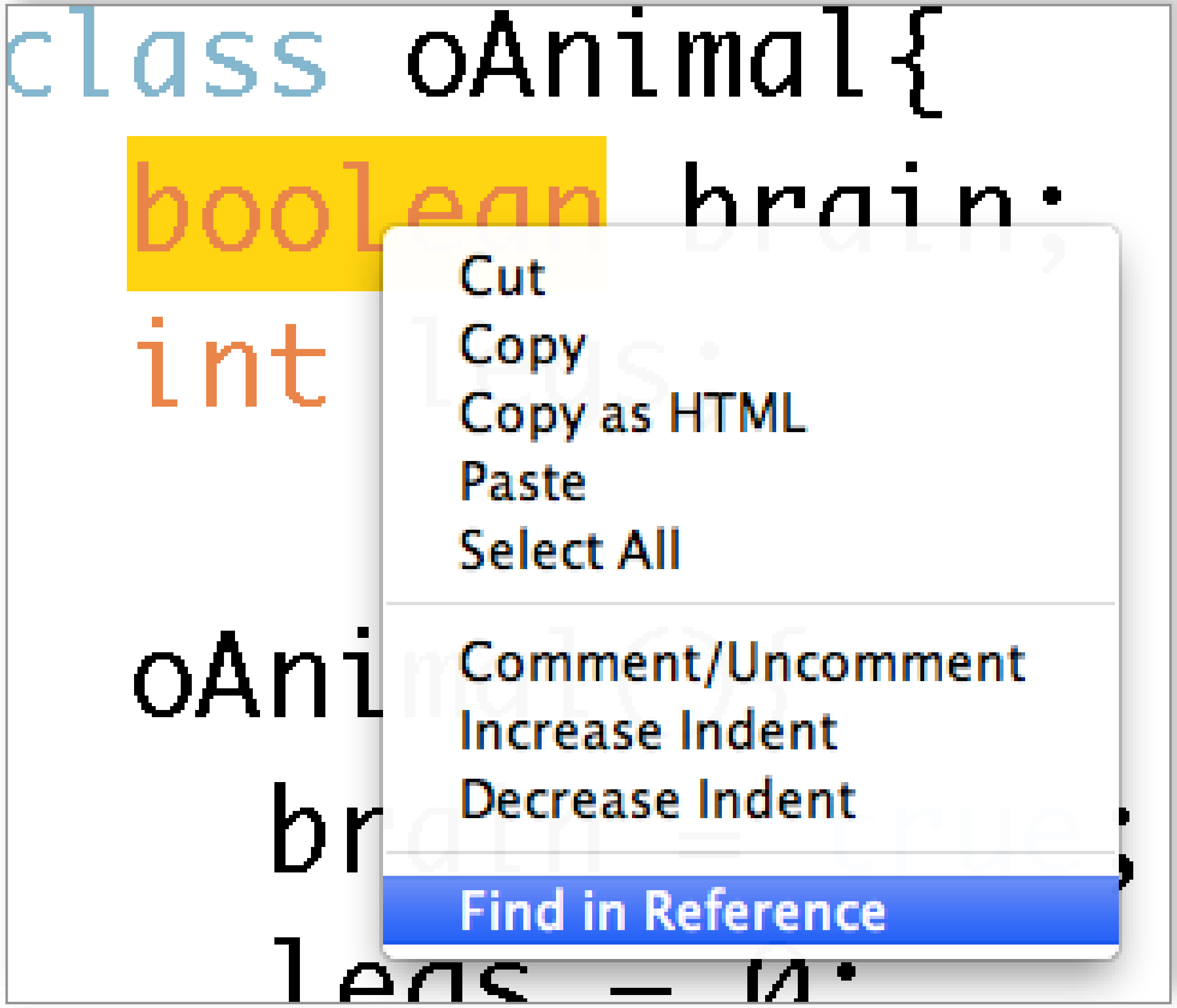
BezierSQLib by Florian Jenett
A library to facilitate communication with SQL-based databases

BlinkStick by Arvydas Juskevicius
Interface BlinkStick - smart USB RGB LED.

documentation

- available online

- also in the PDE

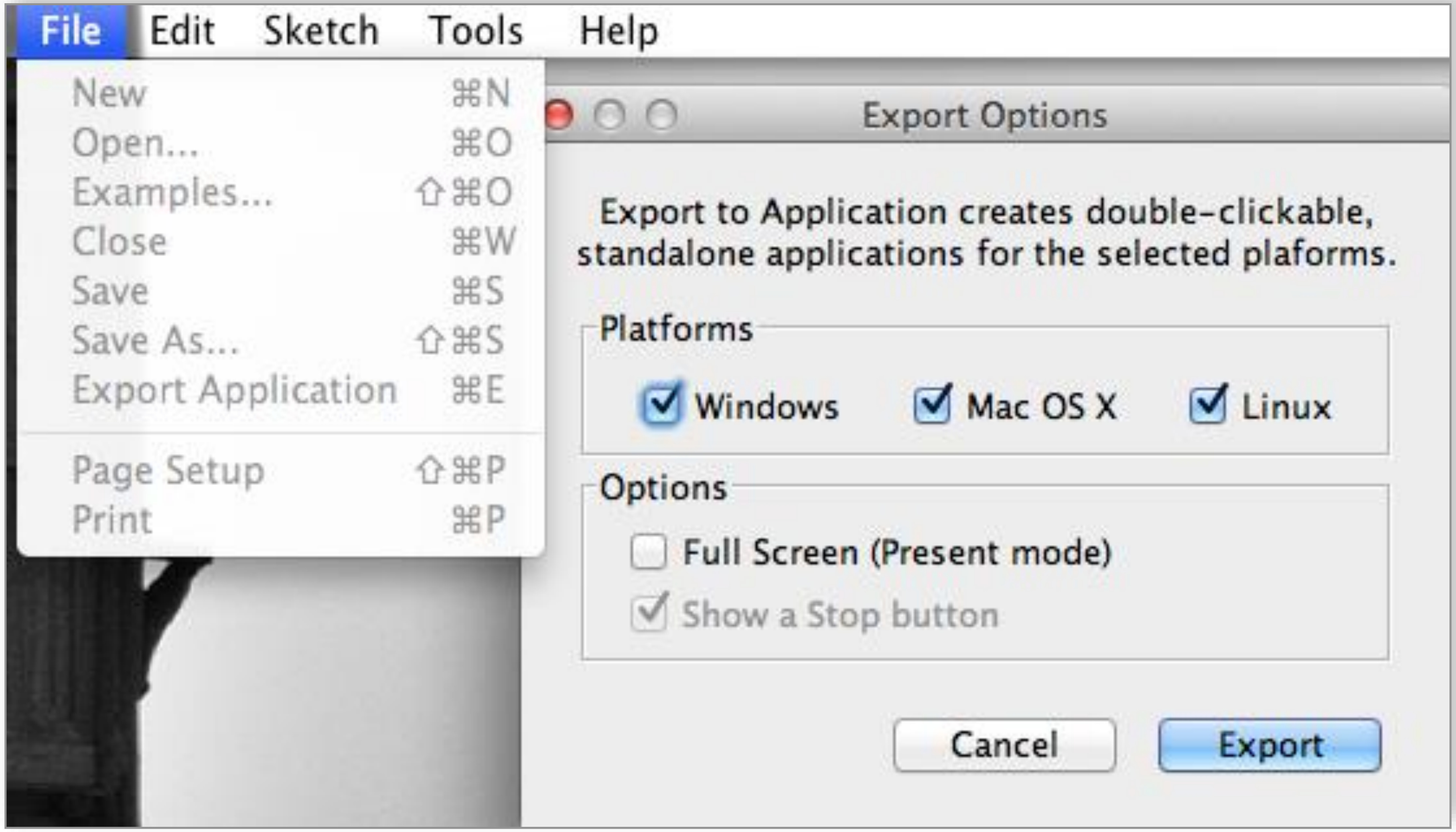


```
class oAnimal{
  boolean brain;
  int legs;

  oAnimal(){
    brain = true;
    legs = 0;
```

- http://processing.org/reference/

exporting

- creating applications is simple



Export to Application creates double-clickable, standalone applications for the selected plaforms.

Platforms

☑ Windows ☑ Mac OS X ☑ Linux

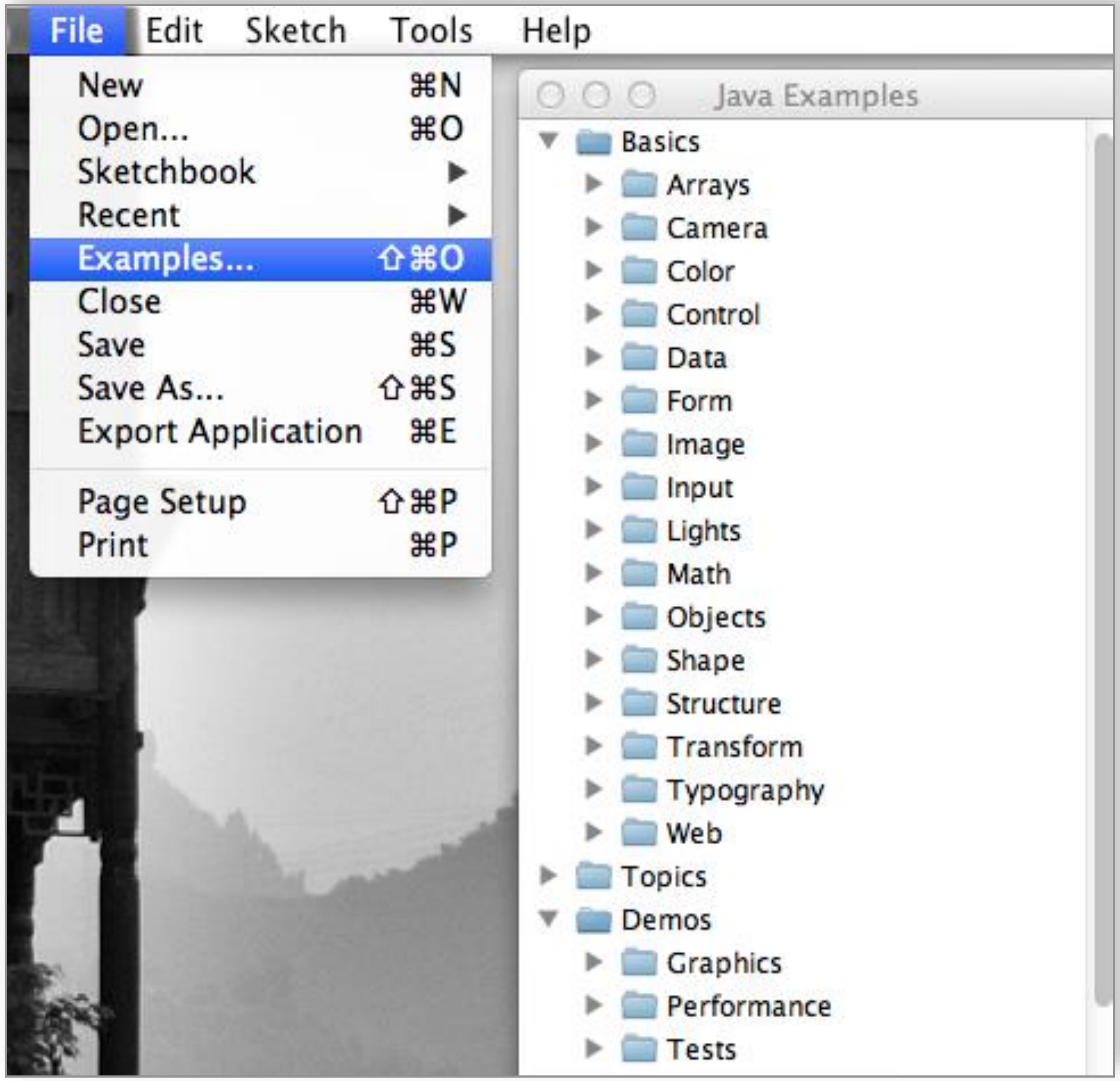
Options

□ Full Screen (Present mode)

☑ Show a Stop button

examples

- variety of samples



assignment

instructions

- download Processing
- download time series project
- follow along in chapter

lab

- no lecture on Thurs.

- bring in laptops or email if you will be in a lab

- work on time series assignment; ask questions

L8. Tasks

REQUIRED READING

Chapter 3

Why: Task Abstraction

3.1 The Big Picture

Figure 3.1 breaks down into actions and targets the reasons *why* a vis tool is being used. The highest-level actions are to use vis to consume or produce information. The cases for consuming are to present, to discover, and to enjoy; discovery may involve generating or verifying a hypothesis. At the middle level, search can be classified according to whether the identity and location of targets are known or not: both are known with lookup, the target is known but its location is not for locate, the location is known but the target is not for browse, and neither the target nor the location are known for explore. At the low level, queries can have three scopes: identify one target, compare some targets, and summarize all targets. Targets for all kinds of data are finding trends and outliers. For one attribute, the target can be one value, the extremes of minimum and maximum values, or the distribution of all values across the entire attribute. For multiple attributes, the target can be dependencies, correlations, or similarities between them. The target with network data can be finding paths, and with spatial data the target can be understanding shape.

3.2 Why Analyze Tasks Abstractly?

presence is worth the penalties of lower resolution and no workflow integration. It is very rare that immersion would be necessary for nonspatial, abstract data. Using 3D for visual encoding of abstract data is the uncommon case that needs careful justification. The use of an immersive display in this case would require even more careful justification.

► The need for justifying 3D for abstract data is covered in Section 6.3.

6.7 Overview First, Zoom and Filter, Details on Demand

Ben Shneiderman's influential mantra of **Overview First, Zoom and Filter, Details on Demand** [Shneiderman 96] is a heavily cited design guideline that emphasizes the interplay between the need for overview and the need to see details, and the role of data reduction in general and navigation in particular in supporting both.

A vis idiom that provides an **overview** is intended to give the user a broad awareness of the entire information space. Using the language of the what–why–how analysis framework, it's an idiom with the goal of *summarize*. A common goal in overview design is to show all items in the dataset simultaneously, without any need for navigation to pan or scroll. Overviews help the user find regions where further investigation in more detail might be productive. Overviews are often shown at the beginning of the exploration process, to guide users in choosing where to drill down to inspect in more detail. However, overview usage is not limited to initial reconnaissance; it's very common for users to interleave the use of overviews and detail views by switching back and forth between them many times.

When the dataset is sufficiently large, some form of *reduce* action must be used in order to show everything at once. Overview creation can be understood in terms of both filtering and aggregation. A simple way to create overviews is by zooming out geometrically, so that the entire dataset is visible within the frame. Each object is drawn smaller, with less room to show detail. In this sense, overviews are created by removing all filtering: an overview is created by changing from a zoomed-in view where some items are filtered out, to a zoomed-out view where all items are shown. When the number of items in a dataset is large enough, showing an overview of the entire dataset in a single screen using one mark