

Processing, p5.js & Computer Vision



Communication & Multimedia Design
& User Experience Design

Danica Mast
d.mast@hhs.nl

*(Based on earlier Processing readers and tutorials by a.o. A. Reuneker, A. J. Quanjer,
C.A. Detweiler, C. G. Heydra)*

Table of Contents

TABLE OF CONTENTS	2
0. ABOUT THIS READER	4
Structure	4
Exercises & Assignments	4
1. GETTING STARTED	5
What is Processing?	5
Installing Processing	5
The interface	6
Structure	7
Exercise 1A	7
Exercise 1B	8
Commenting	9
Exercise 1C	9
2. YOUR FIRST (REAL) PROGRAM	10
Drawing ellipses	10
Exercise 2A	10
Exercise 2B	11
Exercise 2C	12
Changing colors	13
Exercise 2D	13
Exercise 2E	13
Portfolio -exercise 2F: Your first program	14
Other Shapes	15
Exercise 2G	15
Exercise 2H	15
Portfolio-exercise 2I – Your second program	16
Variables	17
Exercise 2J	17
Exercise 2K	18
Animation: a growing circle	18
Exercise 2L	18
Exercise 2M	19
Interactivity: mouse following	20
Exercise 2N	20
Portfolio-exercise 2-O	21

Reference, Examples & Libraries	22
3. PROCESSING AND ARDUINO	24
Exercise 3A	24
Portfolio-exercise 3B: Processing to Arduino	25
4. COMPUTER VISION	26
Introduction	26
Processing & Computer Vision	27
Exercise 3A	27
Assignment 3B	27
Assignment 3D	27
Exercise 3C	27
Assignment 3E	27
5. P5.JS	28
p5.js	28
p5.js & Processing	28
p5.js Exploration	29
Exercise 5A	29
Exercise 5B	29
Portfolio- Exercise 5C	29
p5.js & input & output devices	29
Exercise 5D - Mobile	29
Exercise 5E – Camera filters	29
Exercise 5E – More camera stuff	30
Libraries	30
Exercise 5D – Computer Vision	30
Portfolio-exercise 5E	30

0. About this reader

Structure

This reader covers two main subjects: 'An Introduction to Processing' and 'Processing & Computer Vision'. The exercises will provide you with the basic knowledge and experience to enable you to experiment with Processing and computer vision during the HCI lab weeks.

Exercises & Assignments

This reader contains 'Exercises' and 'Portfolio-exercises'. 'Exercises' will help you understand the working of your code and are optional, but strongly recommended. 'Portfolio-exercises' are compulsory and will help you check and prove that you have understood the subject you've been working on. All exercises must be finished and included in your portfolio.

1. Getting Started

What is Processing?

Processing is an open source programming language and environment for people who want to program interactive images and animations. Processing is used by students, artists, designers, scientists and hobbyists to learn and develop concepts. Processing is used at universities of applied sciences and universities to teach the fundamentals of programming in a visual context, and functions as a digital sketchbook and professional production tool.

Processing is free to download and available for Linux, macOS and Windows. Processing is an open source project, initiated by Ben Fry and Casey Reas. The project has evolved from ideas from the 'Aesthetics and Computation Group' of the MIT Media Lab. *(Text based on introduction text Processing: <http://www.processing.org>)*

Processing is a programming environment that runs on the Java environment. The language is very similar to Java, but is easier and faster to use. Learning Processing therefore is easier for artists who want to make their concepts visual and interactive, and who want to share these creations with their audience.

Because Processing is Java-based, the written applications can be exported to applets, which can be used on the internet. However, stand-alone applications can also be created with Processing.

Installing Processing¹

To learn programming, you need to do more than just read this reader or skip through the examples and assignments. You need to experiment and practice.

To get started, download Processing and make your first sketch. Start by visiting <http://processing.org/download> and selecting the Mac or Windows version, depending on your operation system.

Installation is straightforward:

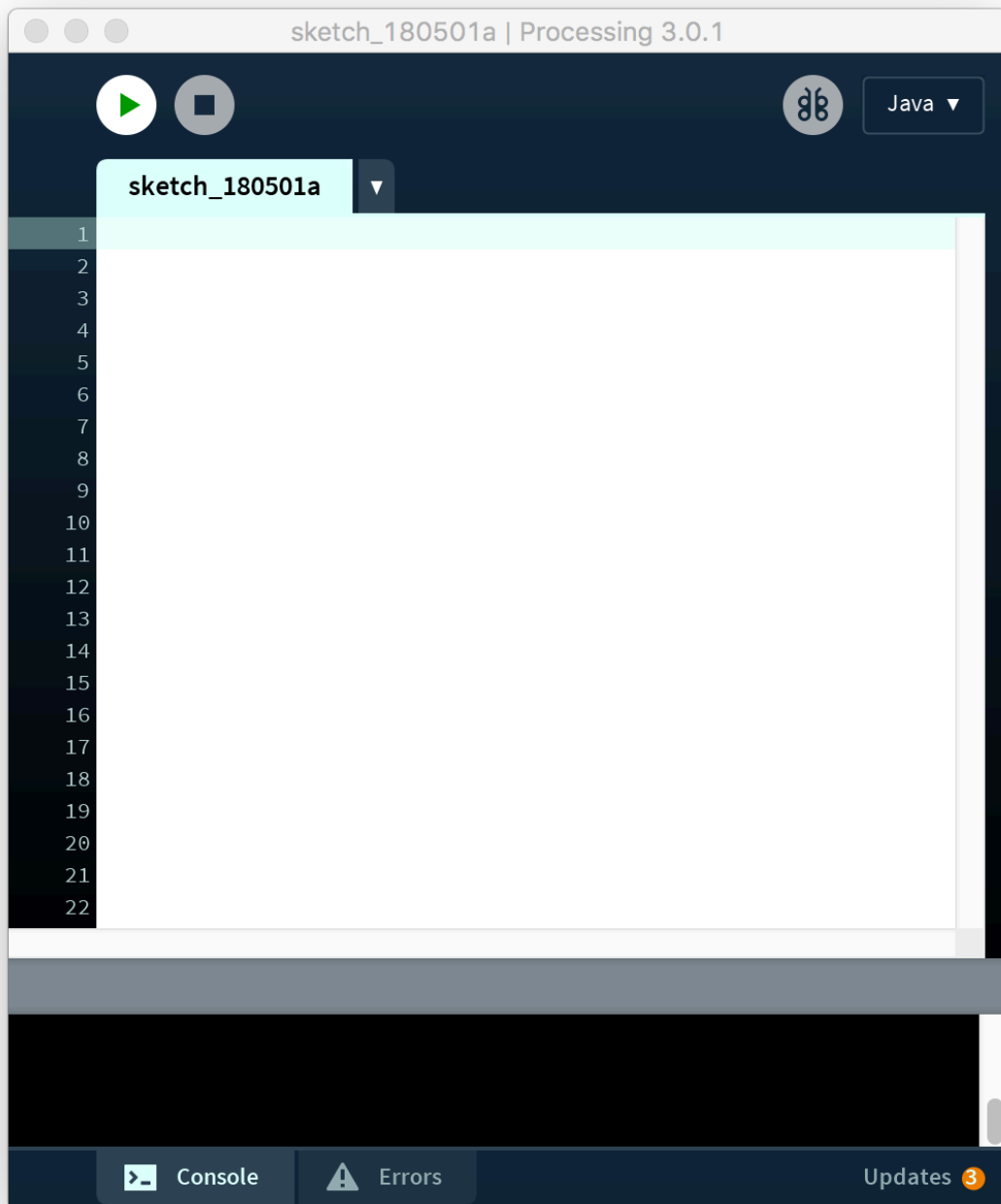
- On Windows, you'll have a .zip file. Double-click it and drag the folder inside to a location on your hard disk. It could be Program Files or simply the desktop, but the important thing is for the processing folder to be pulled out of that .zip file. Then double-click processing.exe to start.
- The macOS version is a disk image (.dmg) file. Drag the Processing icon to the Applications folder. If you're using someone else's machine and can't modify the Applications folder, just drag the application to the desktop. Then double-click the Processing icon to start.

¹ Based on: Reas, C., & Fry, B. (2015). *Getting Started with Processing: A Hands-On Introduction to Making Interactive Graphics*. Maker Media, Inc.



If your installation was successful, you'll be able to run the software and will probably see the main Processing window. (If not: click File>New in the Processing menu to create a new sketch)

The interface

Processing has a very basic interface. It mainly consists of a text editor, the large white area, in which you can type your code.



Button	Icon	Function
--------	------	----------

Run		Compiles the code (converts the input source code to machine language) and executes the program.
Stop		Stops the program that is being executed at the time of clicking.

The black bar at the bottom of the screen is called the console window. Here you can print text as output of your code. You use this regularly when debugging code.

Structure

Within a Processing program, the functions `setup()` and `draw()` are always called. Otherwise your program will not work. Below we will briefly explain what `setup` and `draw` mean.

```
void setup() {  
  size(300,200);  
  println("Hello World!");  
}  
  
void draw() {  
}
```

`Setup` is a function, just like `println` in our example. You may already notice that we can recognize a function from the (and) signs (parentheses) behind it. Often, we give input to a function by putting something between these parentheses.

In the above example, `println` is a function that has "Hello World!" as its input. This function prints something in the console window. We can use this for debugging. The function 'size' specifies the size of the display window (the separate grey window).

Exercise 1A

Type the above example in the text editor and click the Run button.

If you've typed everything correctly, you'll see "Hello World!" written once in the console window. And the display window will have a size of 300 by 200 pixels.

If you didn't type it correctly, the Message Area will turn red and complain about an error. If this happens, make sure that you've copied the example code exactly: the numbers should be contained within parentheses and have commas between each of them, and the line should end with a semicolon.

One of the most difficult things about getting started with programming is that you have to be very specific about the syntax. The Processing software isn't always smart enough to know what you mean and can be quite fussy about the placement of punctuation. You'll get used to it with a little practice.

The syntax (grammar of a programming language) of the setup function in Processing is always as in the above example. Setup is preceded by the word `void` (meaning setup itself has no output) and followed by two parentheses and an opening brace `{`, the contents of the setup and a closing brace `}`.

For now, the only thing you need to know about the setup function is that all statements that are given within setup (which are therefore between the braces that follow the setup line), are only executed once, at the start of the execution of the program, when the play button is pressed.

In the above example this is logical, considering we only want to tell processing to make a window of 300 by 200 pixels at the start of the program. This is done by typing `size(300,200)`; We chose to only execute the `println` function once.

The draw function is executed every time the screen refreshes. (60 times every second, unless indicated otherwise). This will continue endlessly, until the program stops.

The draw function is empty in the previous. In the following example we have put the `println` in the draw function.

```
void setup() {  
  size(300,200);  
  ellipse(50, 50, 80, 80);  
}  
  
void draw() {  
  println("Hello World!");  
}
```

Exercise 1B

Change your code to the above example in your text editor and click the Run button.

“Hello World!” will not just be printed in your console window once, but every time the screen refreshes until the program stops.

Commenting²

In the previous example, you've written code. Code is meant for a computer to understand and execute. You, as a human, might want to be able to add some notes. This can help you remember what is happening where. We call this commenting and it is done as follows:

```
//// this is a comment. The computer will ignore this line

/* A forward slash followed by an asterisk allows the comment to
continue until the opposite */
```

In the example below we have added comments to the example from previous chapter:

```
void setup() {
  size(300,200); //make a window that is 300 by 200 pixels
  ellipse(50, 50, 80, 80);
}

void draw() {
  println("Hello World!"); //print the text Hello World in the console window
}
```

Exercise 1C

Add more comments to the example code

² Based on Programming Bootcamp for UXD by Chris Heydra & Tim v.d. Bosch

2. Your first (real) program³

More interesting, and probably also more fun, is programming with Processing when we cannot only print bits of text in the console window, but when we can produce graphical output. In this section, you will learn all kinds of new functions, with which you can draw shapes in the work window in a very simple way. Later in this reader we will explore how we can add interaction to these shapes.

Drawing ellipses

Exercise 2A

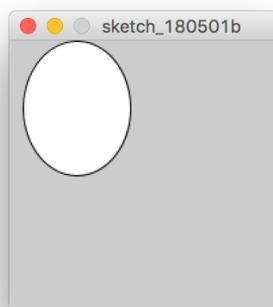
In the editor, type the following:

```
void setup(){
  size(200,200);
  ellipse(50, 50, 80, 100);
}

void draw(){
}
```

This code means “draw an ellipse, with the centre 50 pixels from the left and 50 pixels down from the top, with a width of 80 and height of 100 pixels.”

This example will show the following output in your display window when executed:



This is not a perfect circle.

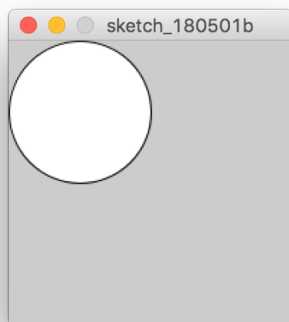
³ Based on: Reas, C., & Fry, B. (2015). *Getting Started with Processing: A Hands-On Introduction to Making Interactive Graphics*. Maker Media, Inc..

Exercise 2B

Adjust your code to make it a perfect circle, by changing the ellipse function in the previous example to the following:

```
ellipse(50, 50, 100, 100);
```

In this line, we have adjusted the third parameter of the ellipse function. Resulting in the following output:



But how does Processing actually count from the left and from the top? In other words, does the center of the circle in the example start at 100 pixels from the top of the display window, or does the angle begin at one hundred pixels from the top of the display window?

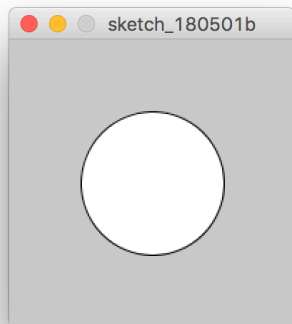
When we don't specify how processing should do this, the top left corner of (an imaginary square around the ellipse) will be used as a starting point.

If we want to use the center of the ellipse, we will have to add a line of code.

```
void setup() {  
  size(200,200);  
  ellipseMode(CORNER);  
  ellipse(50, 50, 100, 100);  
}  
  
void draw() {  
}
```

Exercise 2C

Copy and execute the previous code, this will lead to the following result:



Changing colors

In the previous examples, we have drawn a white ellipse with a black border. This is the standard output of Processing. If we want to change the way our circle looks we have to add some code.

Exercise 2D

To change the color of the border to red, add the following line to your code before you draw the shape:

```
stroke(255,0,0);
```

Colors are indicated in RGB values. So, red is 255 (maximum value) red, 0 blue and 0 green.

Exercise 2E

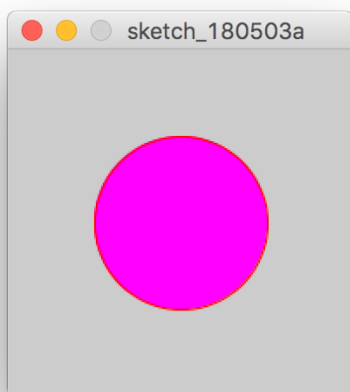
If we want our shape to have no border we add the following:

```
noStroke();
```

To change the color of the shape itself:

```
fill(255,0,255);
```

The shape is now purple (255 red, 0 green, 255 blue). This results in a purple circle with a red border.



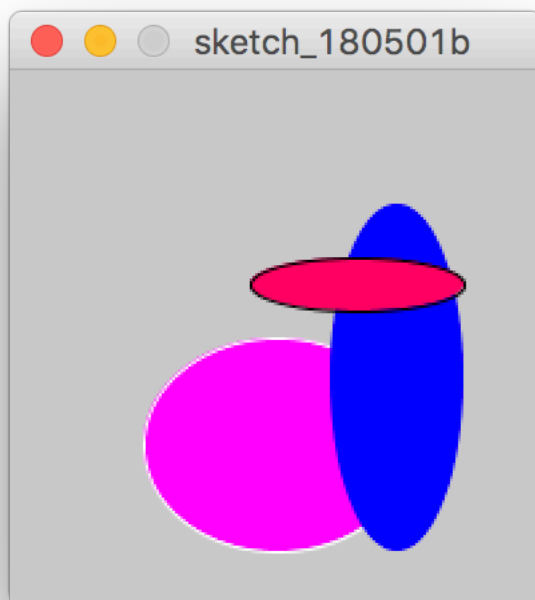
Portfolio -exercise 2F: Your first program

Write a program that draws three (or more) ellipses:

- at various locations,
- with various colors
- various widths and heights
- with and without borders (of various colors)

Copy or printscreen your (aligned) code and give a screenshot of the result in the display window.

The result could look similar to the following output (but of course with your own colors, sizes and locations).



Other Shapes

Besides ellipses we can draw other shapes.

Lines

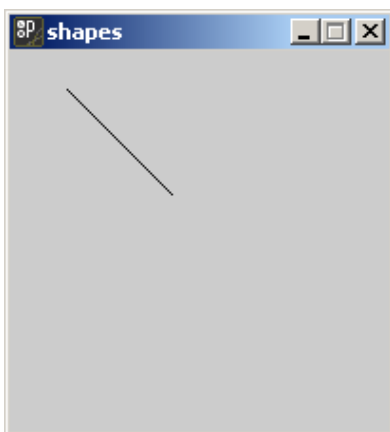
The function line has four parameters. We use the function in the following way:

```
line(30, 20, 85, 75);
```

Exercise 2G

Change your code to draw a line.

The first two parameters represent the x and y coordinates of the starting point, the last two parameters represent the x and y coordinates of the end point. In the above example, this means that a line will be drawn that starts at 30 pixels from the left side and 20 pixels from the top of the output window and is extended to 85 pixels from the left side of the output window and 75 pixels from the left. top. This looks like this:



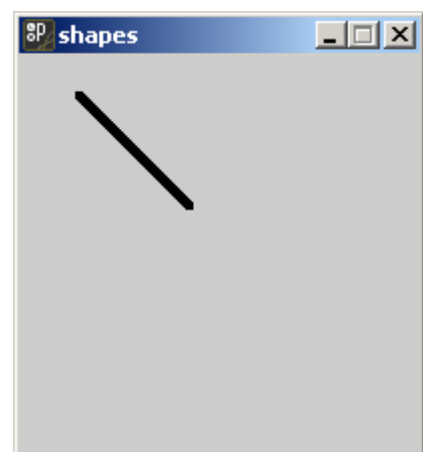
The line runs neatly downwards. If we want a thicker line, we can set this using the `strokeWeight` function. This works as follows:

```
strokeWeight(5);
```

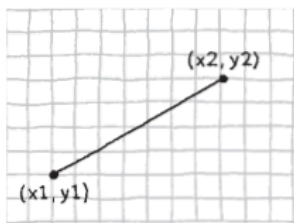
Exercise 2H

change your code to draw a thicker line.

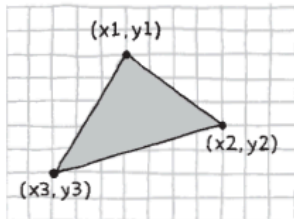
With this line, the line will be thicker than one pixel, or five pixels thick! The result looks like this:



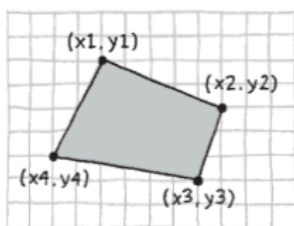
Below is an illustration of other shapes with the according code.



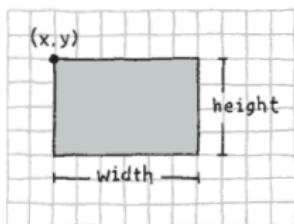
```
line(x1, y1, x2, y2)
```



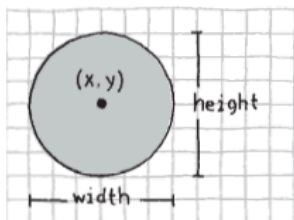
```
triangle(x1, y1, x2, y2, x3, y3)
```



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



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



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

Source: http://cmuems.com/resources/getting_started_with_processing.pdf

Portfolio-exercise 2I – Your second program

Write a program that draws:

- at least four different shapes (not circles),
- in various colors,
- at various locations.

Variables

Until now, we have worked with static numbers. To make your code more dynamic it can be useful to also use variables.

A variable can be seen as a virtual box with a name that you can temporarily store a value on. We call the storage of values in a variable, the declaration of variables.

In Processing you declare variables before you create the void setup. If you create your variables on this spot, you can use them anywhere in your code.

```
int diameter;
```

We have now made a variable of the type int (integer, a whole number) and given it the name diameter. You can choose a name for your variable yourself, but no spaces may appear in this name and they may not start with a number.

A variable can only contain one type of data (text, integers, comma numbers, etc.) but there are different types of variables for different types of data. Via the link below you will find an example of different types of variables and how they are declared in processing.

<http://www.learningprocessing.com/examples/chapter-4/example-4-1/>

The variables don't contain a value yet. We assign a value to the variable as follows:

```
int diameter = 10;
```

We have now set the value 10 in the variable with the name diameter.

Exercise 2J

Add this to your previous example:

```
int diameter = 10; //addition

void setup() {
  size(200,200);
  stroke(255,0,0);
  ellipseMode(CORNER);
  ellipse(50, 50, 100, 100);
}

void draw() {
}
```

Nothing happens yet. The variable is not called anywhere yet. To refer to a variable and the value stored in it, use the name of the variable.

Exercise 2K

Adjust your code to the following:

```
int diameter = 10; //addition

void setup() {
  size(200,200);
  stroke(255,0,0);
  ellipseMode(CORNER);
  ellipse(50, 50, diameter, diameter);
}

void draw() {
}
```

We have now replaced the height and width of the ellipse by the variable.

Animation: a growing circle

In the following example, we see that you can easily make an animation with processing. Because void draw is called again and again, the graphical output is constantly refreshed.

Exercise 2L

By writing a simple piece of code, a value of 1 is added to the diameter of the circle for each new call.

```
int diameter = 10;

void setup() {
  size(200,200);
  smooth();
  background (200,200,200);
  frameRate(15);
}

void draw() {
  ellipse(width/2, height/2, diameter, diameter);

  diameter++;
}
```

This results in an animation of a growing circle.

Exercise 2M

Change the example so that the circle starts at a diameter of 200 pixels and then shrinks.

Interactivity: mouse following

Exercise 2N

We change the previous example to make it interactive

```
int diameter = 10;

void setup() {
  size(200,200);
  background (200,200,200);
  frameRate(15);
}
void draw() {
  ellipse(mouseX, mouseY, diameter, diameter);

  diameter++;
}
```

This program creates a window that is 200 pixels wide and 200 pixels high, and then starts drawing growing white circles at the position of the mouse.

You see that all previously drawn circles remain visible. If we just want to show the last drawn circle, we add the following line to the draw function, before we draw the ellipse:

```
background (200,200,200);
```

The circle will now follow your mouse, grow, but all previous circles are overwritten by a new background, making them invisible.

Portfolio-exercise 2-0

Write a program that draws at least two different shapes, with various colors and make them interactive.

Ideas:

- Draw a line where one of the ends follows the mouse.
- Draw a circle where the color is related to the location of the mouse.
- Draw a rectangle with the size related to the x location of the mouse.
- Etc.

Reference, Examples & Libraries

Reference

In the previous chapters you've learned to use some of the functionalities of Processing, but much more is possible. And sometimes it can be difficult to remember the exact syntax of what you already know. Processing has something helpful for this. It has a reference, a guide, like a dictionary where you can look up how to use commands of Processing.

There are two main ways of accessing Processing's reference:

1. When you select a command in the text editor and right click, you will get the option 'find in reference', clicking on it will send you to the according entry in the reference and give you the information on how to use that specific command.
2. To browse to the full reference, go to menu, click help and then reference. This will give you an overview of the full reference 'dictionary'

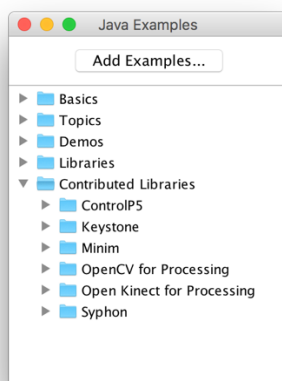
Examples

So far, we have written all the previous code ourselves. We have had a look at Processing's basic functionalities. But there is much more that Processing can do.

To get an idea of the power and possibilities of processing, have a look at Processing's examples.

Included in Processing are many examples of the functionalities that Processing and its libraries (we will get to them later) have to offer.

To access the examples, in the menu go to file and then click examples. A new window will open showing you all examples categorized.



Libraries

A Processing library is a collection of code that extends the software beyond its core functions and classes. Libraries have been important to the growth of the project, because they let developers add new features quickly. As smaller, self-contained projects, libraries are easier to manage than if these features were integrated into the main software.

In addition to the libraries included with Processing (these are called the core libraries), there are over 100 contributed libraries that are linked from the Processing website. All libraries are listed online at <http://processing.org/reference/libraries/>.

To add a library, select 'Import Library' from the Sketch menu and click 'add library'. Choose the library you want to add and click install.

To use a library, select Import Library from the Sketch menu and click on the library you want to use. Choosing a library will add a line of code that indicates that the library will be used with the current sketch. For instance, when the OpenGL Library is added, this line of code is added to the top of the sketch:

```
import processing.opengl.*;
```

(based on: http://cmuems.com/resources/getting_started_with_processing.pdf)

3. Processing and Arduino

In the Arduino workshop you experimented with sending Arduino output to Processing. You can also do this the other way around by sending data from Processing to an Arduino.

Processing can send data through a serial connection and there is a specific library needed in Processing, you can find more information here:

<https://processing.org/reference/libraries/serial/index.html>

We have previously learned that if we want to use an external library we have to install it. To do this, follow the next few steps:

Installing the serial library

- Open Processing and install the serial library by doing the following:
- Menu: Sketch>Import Library>Add Library
- the 'Library Manager' window opens
- Type serial in the search field (make sure you have a working internet connection)
- Click on install
- Wait for the download bar to fill up
- Hurray! The serial library is installed

Exercise 3A

Copy the code below and paste it in a new Processing sketch:

```
import processing.serial.*;

Serial myPort; // Create object from Serial class
int val;       // Data received from the serial port

void setup()
{
  size(200, 200);

  String portName = Serial.list()[0]; //adjust 0 to the port your Arduino
  //is connected to
  myPort = new Serial(this, portName, 9600);
}

void draw() {
  background(255);
  if (mouseOverRect() == true) { // If mouse is over square,
    fill(204);                  // change color and
    myPort.write('H');          // send an H to indicate mouse is over square
  }
  else {                        // If mouse is not over square,
    fill(0);                    // change color and
    myPort.write('L');          // send an L otherwise
  }
}
```



```

    rect(50, 50, 100, 100);          // Draw a square
}

boolean mouseOverRect() { // Test if mouse is over square
    return ((mouseX >= 50) && (mouseX <= 150) && (mouseY >= 50) && (mouseY
<= 150));
}

```

- Build a new circuit on your breadboard, with an LED connected to pin 13 (this is also connector to the on board LED).
- Copy the code below and paste it in a new Arduino sketch

```

// Read data from the serial and turn ON or OFF a light depending on the
value

char val; // Data received from the serial port
int ledPin = 13; // Set the pin to digital I/O 4

void setup() {
    pinMode(ledPin, OUTPUT); // Set pin as OUTPUT
    Serial.begin(9600); // Start serial communication at 9600 bps
}

void loop() {
    while (Serial.available()) { // If data is available to read,
        val = Serial.read(); // read it and store it in val
    }

    if (val == 'H') { // If H was received
        digitalWrite(ledPin, HIGH); // turn the LED on
    }
    else {
        digitalWrite(ledPin, LOW); // Otherwise turn it OFF
    }
    delay(100); // Wait 100 milliseconds for next reading
}

```

Portfolio-exercise 3B: Processing to Arduino

Make a new combination between Processing and Arduino. Use a different actuator on the Arduino side and something different from exercise 2A on the Processing side.

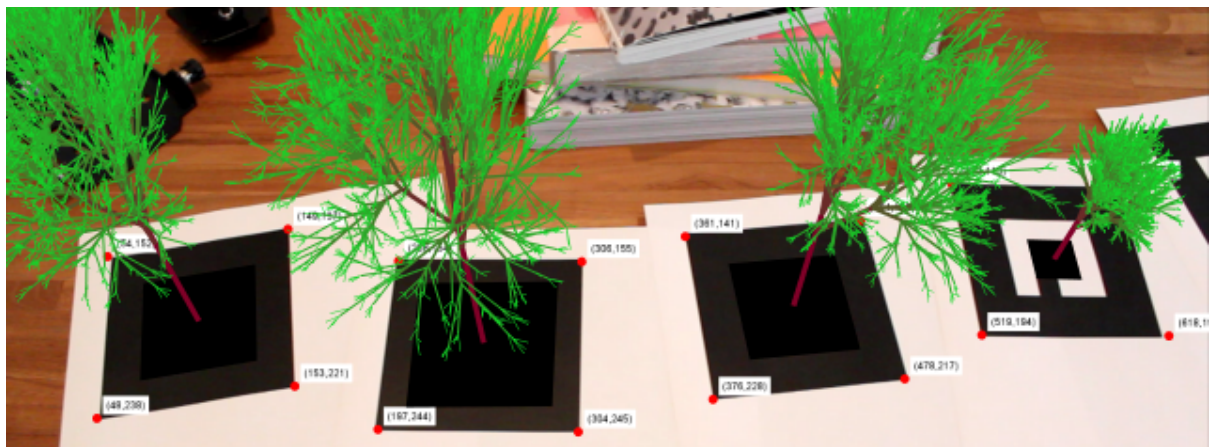
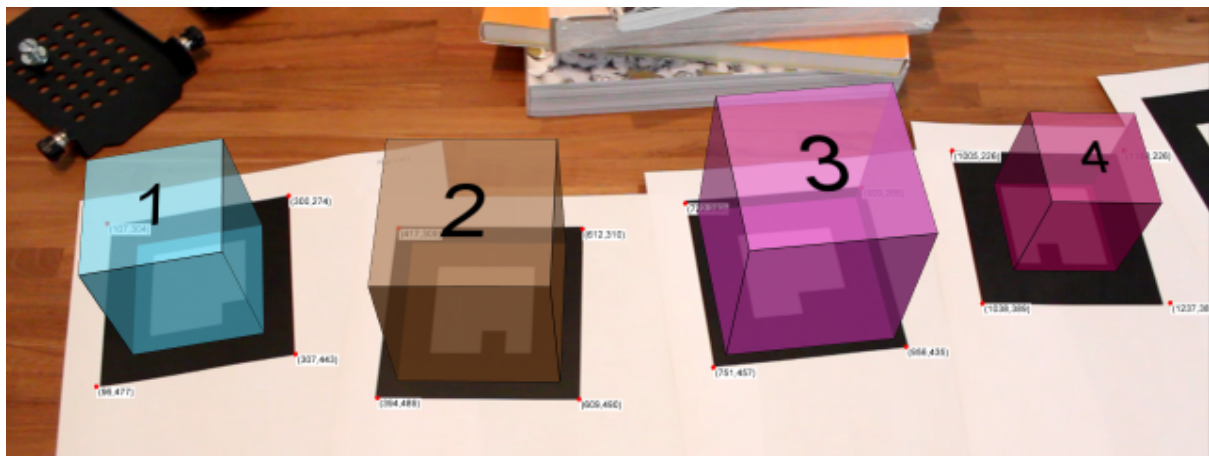
4. Computer Vision

Introduction

Source: <https://www.techopedia.com/definition/32309/computer-vision>

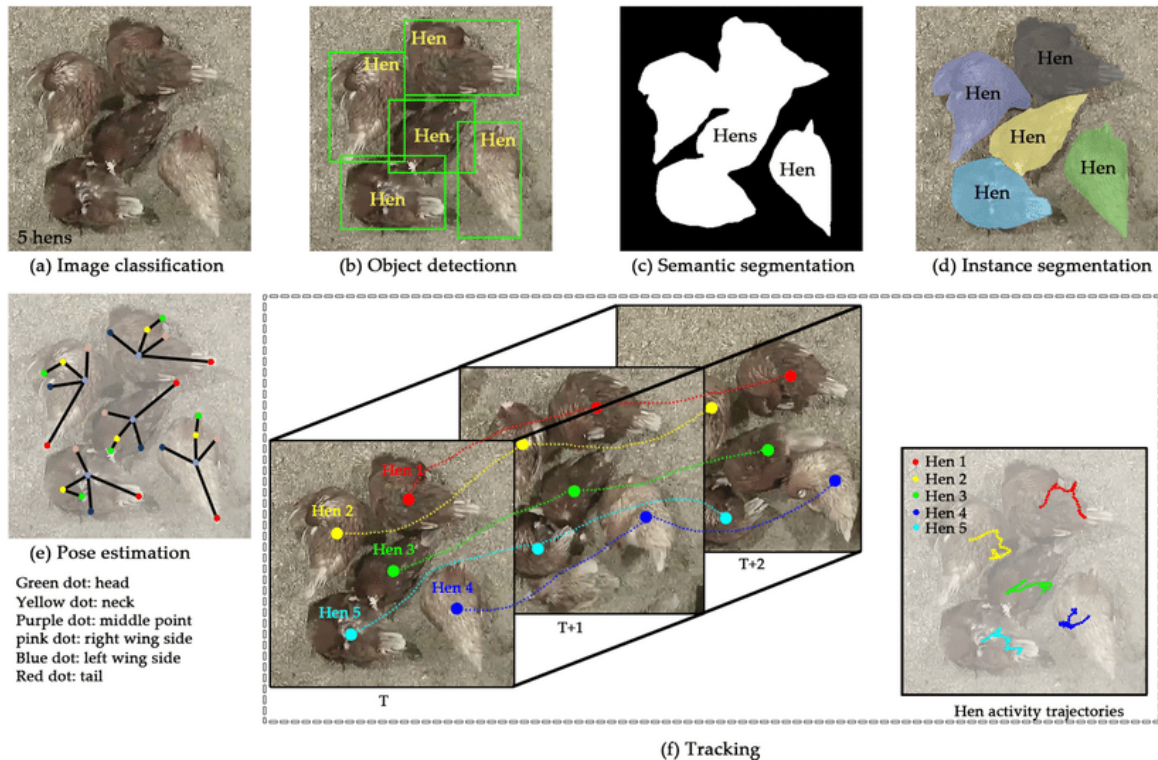
Computer vision is a field of computer science that works on enabling computers to see, identify and process images in the same way that human vision does, and then provide appropriate output. It is like imparting human intelligence and instincts to a computer. In reality though, it is a difficult task to enable computers to recognize images of different objects.

Computer vision is closely linked with artificial intelligence, as the computer must interpret what it sees, and then perform appropriate analysis or act accordingly.



[This article](#) nicely explains in more detail what Computer Vision is and how it works.

The image below gives a good overview of the tasks that can be done with computer vision:



4

Processing & Computer Vision

As we mentioned in the libraries chapter, you can extend the functionality of Processing beyond graphics and images into audio, video, and communication with other devices (such as an Arduino).

One of the devices you can easily use in a cool and powerful way is a webcam/camera.

In previous years, we used the OpenCV and video libraries in Processing to explore and experiment with Computer Vision. Processing however is an open-source project and currently the video library is not properly working on MacOS. We expect and hope that this will be solved in 2022.

Instead, we've added a chapter on p5.js, which is very closely related to Processing and does allow you to explore computer vision techniques (besides a lot of other nice input and output technologies).

⁴ Li, Guoming & Huang, Yanbo & Chen, Zhiqian & Chesser, Gary & Purswell, Joseph & Linhoss, John & Zhao, Yang. (2021). Practices and Applications of Convolutional Neural Network-Based Computer Vision Systems in Animal Farming: A Review. Sensors. 21. 1492. 10.3390/s21041492.

5. p5.js

p5.js

p5.js is a JavaScript library for creative coding, with a focus on making coding accessible and inclusive for artists, designers, educators, beginners, and anyone else! p5.js is free and open source because we believe software, and the tools to learn it, should be accessible to everyone.

Using the metaphor of a sketch, p5.js has a full set of drawing functionality. However, you're not limited to your drawing canvas. You can think of your whole browser page as your sketch, including HTML5 objects for text, input, video, webcam, and sound.

(source: <https://p5js.org/>)

p5.js & Processing

P5 is a direct JS port of the Processing language. Processing.js is a converter which interprets pure Processing code into JS on the fly.⁵

The p5.js language looks very similar to the Processing language with a few [differences](#).

The '[cheat sheet](#)' below⁶ gives a nice overview of p5.js syntax and basic functions.

(There's also a [Dutch version](#) and versions in [other languages](#)!)

p5.js

a cheat sheet
for beginners!

program structure

```
//runs once when program starts
function setup(){
  createCanvas(800,600); //width,height in pixels
}

//run continuously after setup
function draw(){
  //rendering loop
}
```

system variables

```
windowWidth / windowHeight
//width / height of window

width / height
//width / height of canvas

mouseX / mouseY
//current horizontal / vertical mouse position
```

non-visual feedback

```
print();
//report data to the output console

//double slash to comment code (program skips it)
```

color

```
fill(120); //gray: 0-255
fill(100,125,255); //r, g, b: 0-255
fill(255, 0, 0, 50); //r, g, b, alpha
fill('red'); //color string
fill('#ccc'); //3-digit hex
fill('#222222'); //6-digit hex fill
color(0, 0, 255); //p5.Color object
```

math

```
+ - / * //basic math operators

random(low,high); //ranged random number

map(value, in1, in2, out1, out2);
//map a value from input range to output range
```

2d primitives

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

ellipse(x, y, width, height);

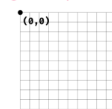
rect(x, y, width, height);

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

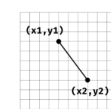
beginShape();
vertex(x1, y1);
vertex(x2, y2);
vertex(x3, y3);
//add more vertex
endShape(CLOSE);

text("string", x, y, boxwidth, boxheight);
```

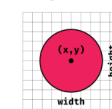
grid system



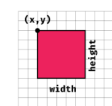
line()



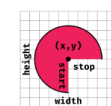
ellipse()



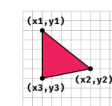
rect()



arc()



vertex()



attributes

```
background(color);
//set the background color

fill(color);
//set the fill color

noFill();
//disables fill

stroke(color);
//set the stroke color

strokeWeight(weight);
//set the stroke's width in pixels

noStroke();
//disables stroke

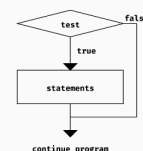
ellipseMode(MODE);
rectMode(MODE);
//CENTER,CORNER

textSize(pixels);
```

if/then logic

```
if(test){
  //statements
}

== //equal to (double is important)
!= //not equal
> //greater than
< //less than
>= //greater than or equal
<= //less than or equal
```



Compiled by Ben Moren <http://bmoren.com> CC-BY-SA-NC-4.0
Translation by Ben Moren

⁵ <https://www.sitepoint.com/processing-js-vs-p5-js-whats-difference/>

⁶ <https://github.com/bmoren/p5js-cheat-sheet/blob/master/p5cheatsheet.pdf>

p5.js Exploration

Exercise 5A

Watch the interactive! video introduction on : <http://hello.p5js.org/>

Exercise 5B

Follow the P5.js getting started exercise at <https://p5js.org/get-started/>.

(You can stop at 'What's next')

Portfolio- Exercise 5C

Add (random) colour to the sketch you made in the getting started exercise.

Make a screenshot or recording of the result of your sketch and add this to your portfolio.

p5.js & input & output devices

Just as with processing, p5.js allows you to combine all sorts of input and output devices.

Exercise 5D - Mobile

P5.js sketches can easily run on mobile devices and use their sensors.

To experience this, open the following [example](#) on a smart phone:



Press the 'display sketch' button on your device. See what happens to the bubbles when you shake your device.

Exercise 5E – Camera filters

P5.js can also use the input of your device's camera

Check the following example: <https://p5js.org/examples/dom-video-capture.html>

See what happens when you comment out (add //) the filter function.

The filter() function can do different things by adding different parameters (that's what you write between the () in a function):

<https://p5js.org/reference/#/p5/filter>

<https://editor.p5js.org/danicamast/sketches/ibWc3Qt94>

See what happens when you uncomment the filter function and add one of the following parameters:

- POSTERIZE, 2
- GRAY
- DILATE
- BLUR, 10

Exercise 5E – More camera stuff

Open <https://editor.p5js.org/danicamast/sketches/0l1NFhR4o>.

Move your mouse across the screen and see what happens.

Libraries

There's a whole range of libraries you can use in p5.js: <https://p5js.org/libraries/>

Some nice examples:

- ASCII art camera
 - https://www.tetoki.eu/asciiart/asciiart_camera.html
- Speech recognition
 - <https://idmnyu.github.io/p5.js-speech/examples/04simplerecognition.html>
- Speech recognition controlled drawing
 - <https://idmnyu.github.io/p5.js-speech/examples/05continuousrecognition.html>

Exercise 5D – Computer Vision

We already briefly looked into using your device's camera with p5.js. And we explained what computer vision is in the previous chapter.

P5.js has multiple libraries allowing you to use computer vision.

- Real time object detection and recognition using YOLO
 - <https://editor.p5js.org/danicamast/sketches/U-suR7Bkb>
- Skeleton tracking and face-tracking using PoseNet
 - https://editor.p5js.org/danicamast/sketches/s-8us8Y_P
- Face tracking
 - <https://editor.p5js.org/danicamast/sketches/5z-XT8yOJ>
 - https://editor.p5js.org/danicamast/sketches/8SHvtPV_B
- Hand tracking
 - <https://editor.p5js.org/danicamast/sketches/i9PfSc70w/add-to-collection>
- Hand tracking combined with a moving particle system
 - <https://editor.p5js.org/danicamast/sketches/zvnmKKHvo>
- Face painting
 - <https://handsfree-face-painting.glitch.me/>

More p5.js computer vision examples: <https://kylemcdonald.github.io/cv-examples/>

Portfolio-exercise 5E

Experiment/tinker with one of the examples mentioned in 5D and create your own variation of one of the sketches. (If you have found another example or want to use one of the

examples of 5E, you probably can, consult your workshop lecturer)
Put a video in your portfolio together with (a link to) the code.