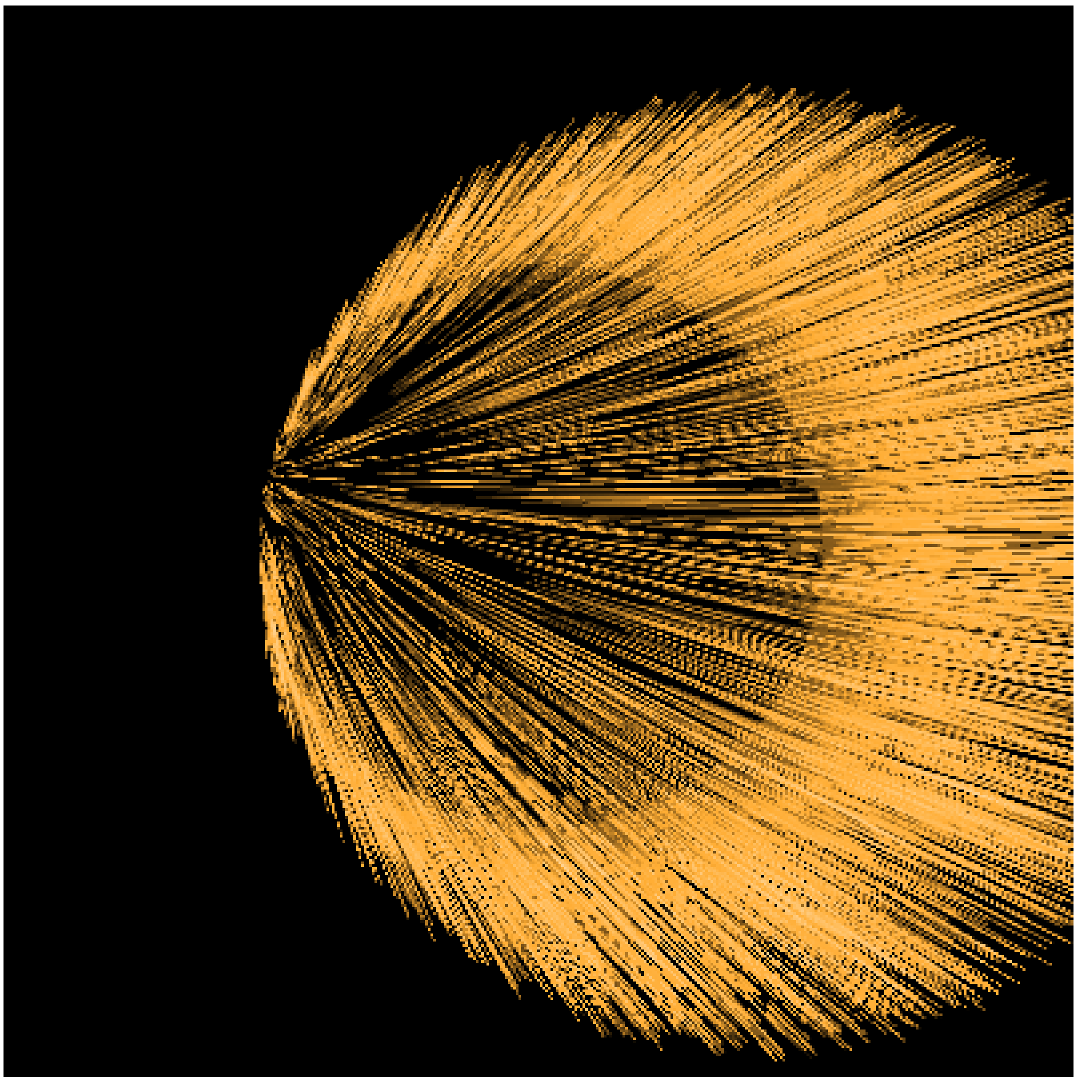
Slave to the Algorithm

Processing workshop RMIT School of Design
Karen ann Donnachie // karenann.donnachie@rmit.edu.au



In this workshop you will be introduced to generative and reactive computer programming, and you will learn how to design and code your first algorithmic sketches using Processing[1] the artist-friendly coding language created by Casey Reas and Benjamin Fry.

You will begin by coding a simple program that generates line drawings in order to become familiar with the code structure, syntax and development environment. Then, through some playful experimentation, you will learn how to incorporate simple data into your very own program.

[1] Processing is a free, open source programming language especially developed for working with visuals, sound and interactivity. Originally created by Casey Reas and Benjamin Fry, both formerly of the Aesthetics and Computation Group at the MIT Media Lab, Processing is designed with non-programmers in mind. See www.processing.org

Slave to the Algorithm : Overview

Workshop Structure: **INTRODUCTION TO ALGORITHMS, CREATIVE CODING & PROCESSING**

1. Introduction to Processing environment, structure and syntax.
2. Single line static generative work.
3. Extension of the static work through colour, stroke and background control.
4. Introduction to the 'setup' and 'draw' structure, creation of looping artwork.
5. Addition of interactivity through mouse position and keystroke.
6. Saving, exporting to video and stand-alone applications of the work.
7. [Optional extension] Introduction to data visualisation in Processing.
8. [Optional extension] Project: "Time to code."

REFERENCES
http://www.processing.org
http://www.processing.org/reference
https://www.processing.org/exhibition/

INTRODUCTION TO PROCESSING

Processing is an open source programming language especially developed for working with visuals, sound and interactivity. It has been developed with artists in mind and requires little to no programming experience.

Processing is installed on computers in the lab, if you would like to install on your home computer or laptop (windows/linux/mac), it is a free download, just go to ;

http://www.processing.org/download

PROCESSING: The application

Start: Launch Processing (double click), the sketch window will open:

1. The large area is simply a Text Editor
2. Along the top there is a small toolbar with play, stop, save and share etc.
3. Underneath there is a message window
4. At the bottom there is a console for technical issues

Syntax: Processing, like any programming language is a bit fussy about syntax (spelling and punctuation, capitalization, order of commands etc), as a guide, most lines will end with a **;** [semi-colon], most words will start with a lowercase letter, where two words are written as one word like 'mouse**P**ressed', the second word will have an uppercase letter. You will get used to these as you go along, for the moment it's enough to pay attention as you copy from this sheet.

OK, LET'S GO! STATIC DRAWING WITH CODE:

1. LINE

In the editor, type (exactly) the following:

```
line (20, 20, 100, 100);
```

Now press the Run button (looks like "play").

Your code will execute and you will see a new window appear. A black diagonal line should appear on a (default) grey background. The numbers you typed are the coordinates of the line (start x, start y, end x, end y).

Yay! You have just made a drawing from code. Now try **square**…

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

2. CIRCLE

Clear your text window and type the following;

```
ellipse(50, 50, 60, 60);
```

When you run this code, you will have drawn a circle with the centre 50 pixels down and 50 pixels from the left, with a width of 60 pixels and height of 60 pixels.

Let's make one of these shapes appear again but this time our shape and background will be coloured and we'll size the window to be bigger than the line:

```
size (240, 240);
background (0, 255, 0);
stroke (0, 0, 255);
line (20, 20, 220, 220);
```

*NOTE: Colour values of ‘stroke’ must be defined BEFORE you ask the program to draw your line.

The colours are made from red, green and blue values 0-255 inside the brackets in the format (R, G, B)—zero is off and 255 is the brightest — for example stroke (0, 0, 255) made our line blue and background (0, 255, 0) made our background bright green.
Alternatively you can give a single value which would be a shade of grey from 0 (black) to 255 (white) by just putting one number inside the bracket (for background, or stroke) eg.:

```
size (240, 240);
background (0, 255, 0);
stroke (125);
line (20, 20, 220, 220);
```

Now play around with all of these values like the line coordinates, colour values and size of the bounding box, just changing the numbers.

You can also add a fourth value to the colour which works as transparency, set from 0-255, so the values become (r, g, b, a).

Remember you cannot break Processing, it’s a sandbox to test and experiment.

You can also see your sketch ‘presented’ at any time in the centre of a clear screen by pressing shift+Run. Just press ‘stop’ to close the window and go back to editing mode.

Try to colour your circle or put it on a coloured background as in the previous example, experiment with the coordinates or width and height values to see how it changes, try to make your display window / program area bigger or smaller to make your circle fit (with the ‘size’ command). Remember to colour your stroke before you draw your circle.

3. INTERACTIVITY

OK, so we can draw a circle, a box and a line, but what we really want to do is have our drawing machine be animated and responsive in some way.

MOUSE FOLLOW: Enter this code:

```
void setup() {
      size (480, 360);
            }
void draw() {
      ellipse(mouseX, mouseY, 60, 60);
            }
```

When you run this code your window should show a white circle on a grey background and if you move your mouse, more circles will be drawn all over the place. What you have just typed in are two ‘functions’ or sets of instructions.

The draw function is using the input coordinates from your mouse.

Woohoo!

MOUSE CLICK: Now let’s add click-ability. Insert the code **in bold** to your current code:

```
void setup () {
      size (480, 360);
      background(255);
            }
void draw() {
      if (mousePressed) {
            fill (255, 128, 0);
            }
      else {
            fill (255);
            }
      ellipse(mouseX, mouseY, 60, 60);
           }
```

Now when you run your sketch you will see that the circles are white **`(255)`** when the mouse is not pressed, but are orange **`(255, 128, 0)`** while the mouse button is pressed down.

CLEAR IT: We can add another interaction from any key pressed through the function, we will ‘erase’ the drawing by overlaying a white background.

ADD this code at the bottom of your sketch:

```
void keyPressed() {
       background(255); // THIS SHOULD BE THE COLOUR YOU SET
           }
```

MORE: See if you can work out what this sketch may look like before you run it:

```
void setup() {
size (480, 360);
background(255);
smooth(); //this antialiases the drawing
      }
void draw() {
if (mousePressed) {
      fill (255, 255, 0, 200);
      }
else {
      fill (255, 0);
      }
translate(mouseX, mouseY); // this moves the action to your pointer!
rotate (mouseX);
rect(-15, -15, 30, 30);
      }

void keyPressed() {
       background(255);
           }
```

PRACTISE!
Try getting the program to change the **size, stroke, fill colour & transparency** of the shape.
You can remove the stroke with **`nostroke();`**
You can modify the **background colour**, in our `setup()` function.
Change the size of the shape, make it get bigger or smaller depending on the placement of the mouse.
Experiment with the drawing by try getting it to draw **'line'** as our first exercise instead or circle remember to insert 4 coordinates inside the brackets separated by commas (and any of these coordinates *can* be mouseX and/or mouseY or a math involving these values).

What happens if you insert the mouseX or mouseY as part of your colour value…?

4. ADDING SOME MORE GROOVY

You can open new sketches to try out these functions and then add where you want to your drawing sketch.

Draw a continuous line:

```
void setup() {
  size (800, 600);
  background(255);
  smooth();
    }
void draw() {
   if (mousePressed) {
     line (mouseX, mouseY, pmouseX, pmouseY);
        }
    }
void keyPressed() {
  background(255);
    }
```

You can set **line thickness** with the strokeWeight() function. Eg. strokeWeight(4);

You can even set dynamic line thickness. INSERT these lines before “line”:
(can you work out what the function calculates?)

```
  float weight = dist(mouseX, mouseY, pmouseX, pmouseY);
  strokeWeight (weight);
```

RANDOMISE: add the declaration of the **variable to the top of the sketch,**
and the **function** at the end.

```
int r,g,b,w; //this gets added to the TOP of the sketch

void keyPressed () {
  r = int(random(256));
  g = int(random(256));
  b = int(random(256));
  w = int(random(256));
  background (r,g,b); //randomises the background
  stroke (w); //randomises the stroke colour
      }
```

4. SAVING, EXPORTING AS IMAGE, VIDEO OR APPLICATION

So once we have a sketch up and running, we should save the sketch using a name and version (so we can go back if we break it :).
Pressing 'save' (down arrow in toolbar) will show the processing location to save to, I suggest you choose 'new folder' and save all your work in there. This folder will then appear under 'file -> sketchbook.' Take this folder away with you, or email it to yourself if you want to have the sketches at home.
Export the program as a 'stand-alone', this is a simple option to distribute the work, just press the right arrow, embed java if you are not sure about the version of Java present on the destination machine, and Voila! You have just made a Mac OS 'app'.

Exporting **images (or video)** is slightly trickier… you need to **insert** the action

```
saveFrame (“nameOfFile_####.jpg”);
```

Explanation [from the processing.org reference page]: "Saves a numbered sequence of images, one image each time the function is run. To save an image that is identical to the display window, run the function at the end of draw() or within mouse and key events such as mousePressed() and keyPressed(). Use the Movie Maker program in the Tools menu to combine these images to a movie."

You can replace 'nameOfFile' with any name you like and the hashtags are so that Processing will number the files consecutively up to 4 digits (so up to 9999 images). You can save .jpg, .png, .tif or .tga files. You can even save .gif files if you want to!

Three new shapes:

```
 point (x, y); // any point in 2d space
triangle (x1, y1, x2, y2, x3, y3); //set three corners
arc(a, b, c, d, start, stop, mode); //*semi-circle
*Where:
a  float: x-coordinate of the arc's ellipse
b  float: y-coordinate of the arc's ellipse
c  float: width of the arc's ellipse by default
d  float: height of the arc's ellipse by default
start  float: angle to start the arc, specified in radians
stop  float: angle to stop the arc, specified in radians
Example:   arc (300, 300, 150, 150, 0, PI, CHORD);
```

Arcs are drawn along the outer edge of an ellipse defined by the a, b, c, and d parameters. Use the start and stop parameters to specify the angles (in radians) at which to draw the arc. There are three ways to draw an arc; the rendering technique used is defined by the optional seventh parameter. The three options, depicted in the above examples, are PIE, OPEN, and CHORD. The default mode is the OPEN stroke with a PIE fill.
[from https://processing.org/reference/arc_.html]

SAVING IMAGE SEQUENCE EXAMPLE: So a drawing sketch may now look like this:

```
void setup () {
  size (480, 360);
        }
void draw() {
  if (mousePressed) {
    fill (255, 128, 0);
      }
  else {
    fill (255);
      }
  ellipse(mouseX, mouseY, 60, 60);
  saveFrame ("frames/circles_###.jpg");
  }
```

When we run the sketch, it only takes a few seconds for hundreds of jpgs to fill our new “frames” folder it will keep generating images automatically from run until we press stop.
We can then import these through **“Tools -> Moviemaker”** we choose the folder of images/frames (from inside our sketch folder), for the moment we will not add sound but you can choose the size and framerate.
Choose the name for your movie and you will then find it in the folder you choose to save it in. The moviemaker window doesn’t close by itself so just close it once you are done.
Alternatively you can film the screen or part of the screen with Quicktime or on of the many 3rd party screen-grab programs that will record your actions and the results as you run your Processing sketch.

Congratulations! You have made a movie from code!
[you can stop here if you like, otherwise keep playing]

5. MORE PROCESSING EXAMPLES

Processing comes bundled with a large selection of examples for all the basic functions. Your “examples” window may have opened on launch, if not, go under ‘file -> Examples..’
In this window you will find examples of code and sketches that work with text, images, graphics and so on.
Try opening ‘Demos -> Graphics -> Yellowtail’ or any of the others.
Read the introduction in the code (in grey) and run the program. To see what any particular part of the code does, you can highlight the operator or function in your code and control click to ‘Find in reference’ you will see a page about that function with simple examples on how to use it and links to related terms.
Try any of the other examples you find.
Try browsing the processing.org gallery to see others.

SLAVE TO THE ALGORITHM (Part 2)

1. REFRESH
A sketch will usually contain a ‘setup’ function and a ‘draw’ function.
Processing works by drawing layers one on top of another, so if you change the background it will effectively wipe out (cover over) what came before it.

2. Working with Images
Processing is a fantastic tool for working with images and text as well as generating graphics.
To load an image or a text, it is good practice to work with your source files copied over to your sketch’s ‘data folder,’ so when Processing compiles the final program it will find and embed the images or text as necessary for each output.
The easiest way to create your sketch’s data folder is by dragging an image or text file *onto* the main sketch window, if there isn’t a data folder already, Processing will automatically create one for you. You can always see your sketch’s folder by pressing COMMAND + K. Once the image/text is copied over, use the loadImage() or loadStrings():

```
//Example of loading an image from the data folder
 PImage = loadImage (“picture.jpg”);

//Example of loading text
 String[] lines = loadStrings(“yourtext.txt”);
```

To use PImage to load an image and display it:

```
PImage photo;

void setup() {
 photo = loadImage ("XXX.jpg");
 size(640, 480);
      }

void draw() {
  image(photo, 0, 0);
      }
```

We can then add filters eg. photo.filter(GRAY) added in the setup function after loadImage… will turn it to greyscale, or other filters such as BLUR, THRESHOLD, etc; We can rotate, scale, resize…
We can build functions to animate or adjust the image. To “Pointilize” your image for example… your sketch may look something like this:

```
PImage img;
int smallPoint, largePoint;

void setup() {
  size(640, 360);
  img = loadImage("XXX.jpg");
  smallPoint = 4;
  largePoint = 40;
  imageMode(CENTER);
  noStroke();
  background(255);
      }

void draw() {
  float pointillize = map(mouseX, 0, width, smallPoint,  largePoint);
  int x = int(random(img.width));
  int y = int(random(img.height));
  color pix = img.get(x, y);
  fill(pix, 128);
  ellipse(x, y, pointillize, pointillize);
       }
```

Try placing more than one image at a time in the window.
Try some of the image examples in the image processing examples folder (for example 'LinearImage' or 'Explode.')

2. WORKING WITH TEXT

First we need to create a font from one on our system. Under "Tools" choose "create a font".

```
PFont font;
void setup() {
  size(480, 120);
  smooth();
  font=loadFont("Consolas-48.vlw");
  textFont(font);
   }
void draw(){
textSize(48);
text(" big word", 10, 30);
textSize(32);
text("medium word", 10, 60);
textSize(8);
text("tiny word", 10, 90);
}
```

Text can also be set inside a box, to do that we can set two more parameters inside the text box. Try adding a box and writing a longer sentence to test how Processing fits the text in.

Combining more ideas… let's make a dot-to dot…
First we need to create a sketch that will only draw when we press the mouse, so we can place our dots where we need to. Next we need to add numbers as we place the dots, then finally we need to save the file (but only when we are happy with it), we'll trigger the file save with any keyPress.

```
int dotx=1; //this is our starting number
int timelapsed;
void setup (){
  smooth(); //this makes our graphics anti-aliased
  background (255);
  size (640, 480);
  noStroke();
  fill(0);
      }

void draw (){
            // nothing in here, we don't want automation
      }

void mousePressed (){ //only on mouse click
  ellipse ( mouseX, mouseY, 2,2); // draws small circles
  textSize(10);
  text(dotx, mouseX+10, mouseY+10); //offset from dot
  dotx++; //our dot number increases automatically
      }

void keyPressed (){ //any key will output an image
  timelapsed=int(millis()/1000);
  save ("dot_to_dot"+key+".png"); //save the final piece
      }
```

*each time you save, if you choose the same key, will write over the last file, so change your chosen key if you want to save more than once.
Again, you can go through the Reference Files of Processing to find other ways of working with your images, text, movies or graphics.

3. TIME TO CODE [in this part you will write your own code from scratch]
You will make your very own standalone clock application!
This program is an example of working with dynamic data, but we use a variable easily accessible from the application— the internal clock of your computer.
Processing already includes access to the current time which can be read with the second(), minute(), and hour() functions.

Let's walk through the process:

1. Setup the size of your 'clock', including background.
2. To show the time, we will need to set parameters in setup() for your text, such as textSize(), textAlign().
3. In the draw function, so that it constantly updates, make Processing write the values of hour(), minute() and second() as text. Insert a text box and test with your time as the text.
4. Debug!
5. List the issues that need addressing in your code. There are some to do with the formatting of our numbers that can be easily fixed. When you are happy with your code, we can export the clock as a standalone application, so you can keep it on your desk.

But, it's rather boring, and actually anyone can do that , right? :), So, lets change the visualisation a bit, rather than simply showing the actual time in numbers, we will translate the data into a colour clock, and perhaps remove the time altogether (we can add a toggle switch to temporarily show the numerical time).

6. You can choose if this should happen inside a shape or if you want the whole background to 'be' the clock. So add a shape or an additional background() call in the draw() function.
7. Set the colour of the background to read the hour(), minute(), second() values as R,G,B.
8. How can we approximately map these values to the full spectrum of colour (0-255)?
9. Code the toggle switch with our keyPressed or mousePressed test to display the time.

How could you add 'hands' to your clock? Try to think outside the standard visualisations…

```
String second;
void setup() {
  background(hour()*10, minute()*4, second()*4);
  size(800, 500);
  textSize(150);
  textAlign(CENTER);
}
void draw(){
  background(hour()*10, minute()*4, second()*4);
  minute=nf(minute(),2);
  second=nf(second(),2);
  //fill(hour()*10, minute()*4, second()*4);
  //ellipse(250, 250, 500, 500);
  if (keyPressed){
  text(hour()+":"+minute+":"+second, 0, 180, 800, 350);
  }
}
```

4. LOVING LIBRARIES

A library is a pre-packaged set of code to add functionality to your processing sketch, this could be to capture images from cameras, kinect, or other devices, to make pdf files, to draw bezier curves or many other things. Look under ‘Sketch -> Add Library’ to see the types of libraries available. You can search for and add more libraries. Once added you can experiment with these libraries through your Examples window as ‘contributed libraries.’

LINKS AND USEFUL REFERENCES

Raven Kwok: https://vimeo.com/74877028
Processing Favourites: http://www.openprocessing.org/browse/?viewBy=most&filter=favorited
Processing.org Exhibition: https://processing.org/exhibition/
Angelo Plessas: http://www.angeloplessas.com/index.php?/websites/-more-websites/
Rafael Rozendaal: http://www.newrafael.com/websites/
Ryoji Ikeda Test pattern #5: https://www.youtube.com/watch?v=nkC3lu6x6i0
Yoko Ono Instructional pieces: http://www.a-i-u.net/yokosays.html
Flong: http://www.flong.com/
ThreeJS: http://threejsplaygnd.brangerbriz.net/archive/
myShrine: www.myShrine.org
“Selfie cameras”: www.twitter.com/delayedrays www.twitter.com/us_others
Karen ann Donnachie http://www.karenanndonnachie.com

Shared Processing sketches, 2019

https://www.openprocessing.org/sketch/498723

https://www.openprocessing.org/sketch/648192

https://www.openprocessing.org/sketch/205584

https://www.openprocessing.org/sketch/567018

https://www.openprocessing.org/sketch/110105

https://www.openprocessing.org/sketch/566709

Github Repo List

https://github.com/karenanndonnachie/CreativeCoding2019Repo

https://github.com/KeyTrent/

https://github.com/oliverrst/olisrep

https://github.com/shangriffo

https://gist.github.com/wendywanomg

https://github.com/Eugenieyi

https://github.com/julienccch/creative-coding-

https://github.com/zaligill/Zali-Creative-Coding

https://github.com/kangelax

https://github.com/remoribarits

https://github.com/momeow/momeow

https://github.com/ORBasten/creative_coding

https://github.com/Rosettax

JIANFANG LIN https://github.com/yyqxlin