Creative Coding - Week 1

Processing, Part 1

Lecturer: Dr Michael Cook (mike.cook@kcl.ac.uk)

In this week's session we're going to learn how to work with p5.js, an online Javascript tool for rapid creative coding, mainly for 2D and 3D art and games. This will involve a short introduction to the tool, then lots of time for you to play with it yourself and try to make a piece of art.

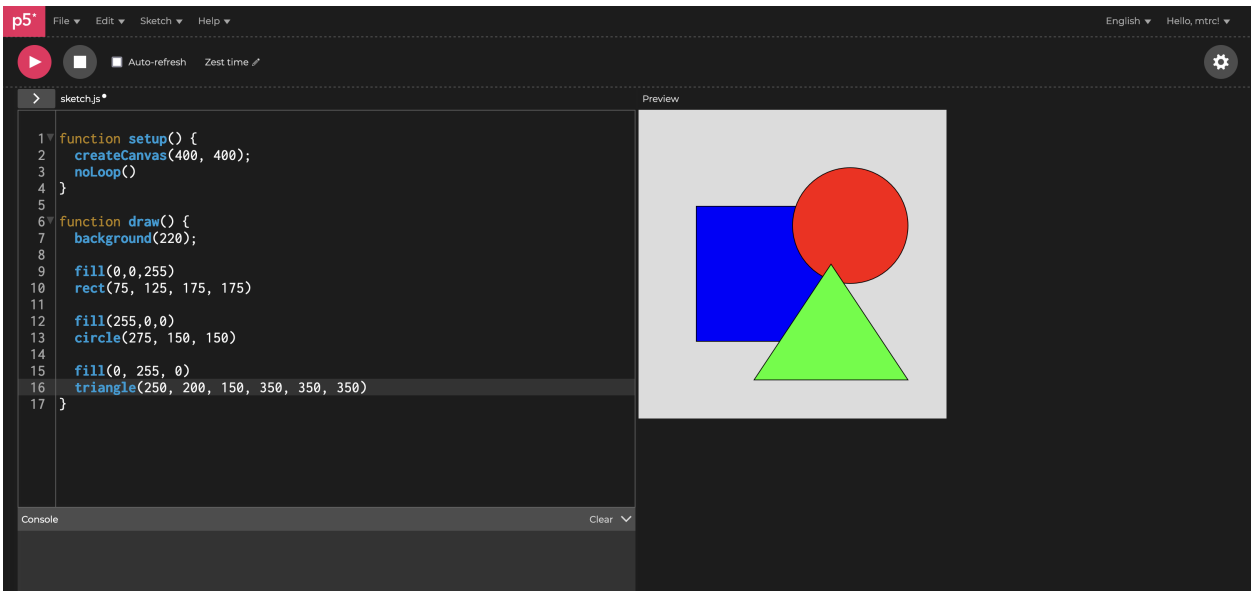
p5.js has a browser-based editor, which you can find here:

editor.p5js.org

I recommend registering an account on the p5.js website, which will let you save and load your scripts (called *sketches*) from any device. You can find a login/signup link in the top right of the editor screen.

1 Using p5.js

The browser editor has two panels. On the left, by default, is where you write your code. On the right is where the output is shown.



To run your program, click the play button in the top left, or press CTRL + ENTER (CMD + ENTER on Mac). p5.js is a simplified Javascript environment, but most Javascript you write will work. There are two important methods built into the environment, however:

- `setup()` is run once, at the beginning of the program's execution. The first line is usually a call to `createCanvas(w, h)` which creates the output that we're going to draw on with our sketch code.
- `draw()` is run thirty times per second (by default) and is where people usually put their code to draw things on the screen. You don't have to write all your drawing code in here, and you

can also call `draw()` yourself additional times if you want to. If your sketch has no input or motion, you may want to disable `draw()` being called thirty times per second. To do this, put `noLoop()` in your `setup()` function - you can see this has been done in the example above.

2 Drawing, Colouring and Moving

I've made two sketches to introduce you to the basic features in p5. You can find the first here:

https://editor.p5js.org/mtrc/sketches/19qcKcqk5

Try changing bits around and getting a feel for how the different modes work. When you're done, you can check out this sketch template to get some examples of how to quickly get things happening on-screen:

https://editor.p5js.org/mtrc/sketches/mAo$_x$*hnDQ*

Once you've had a look at both of these, or are ready to move on, go on to Part 4 and start working on something!

3 Finding Answers

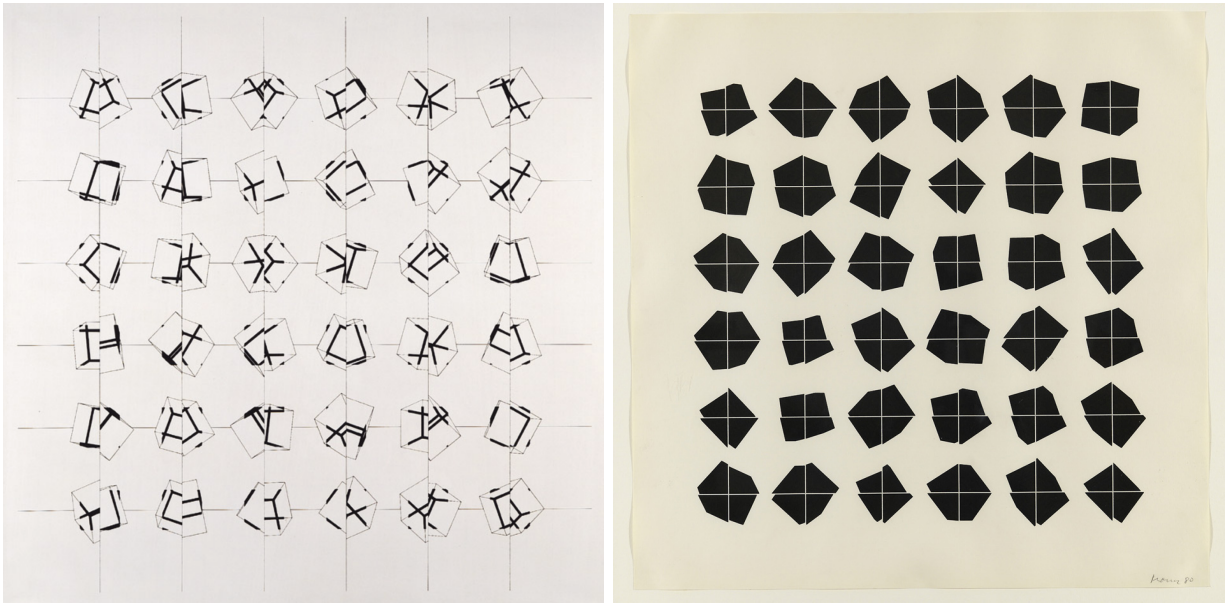
p5.js has an extensive and very useful reference page that lists all the different functions it contains, with sample code and explanations for each. If you want to find out how to do something, this is the first and best place to look:

https://p5js.org/reference/

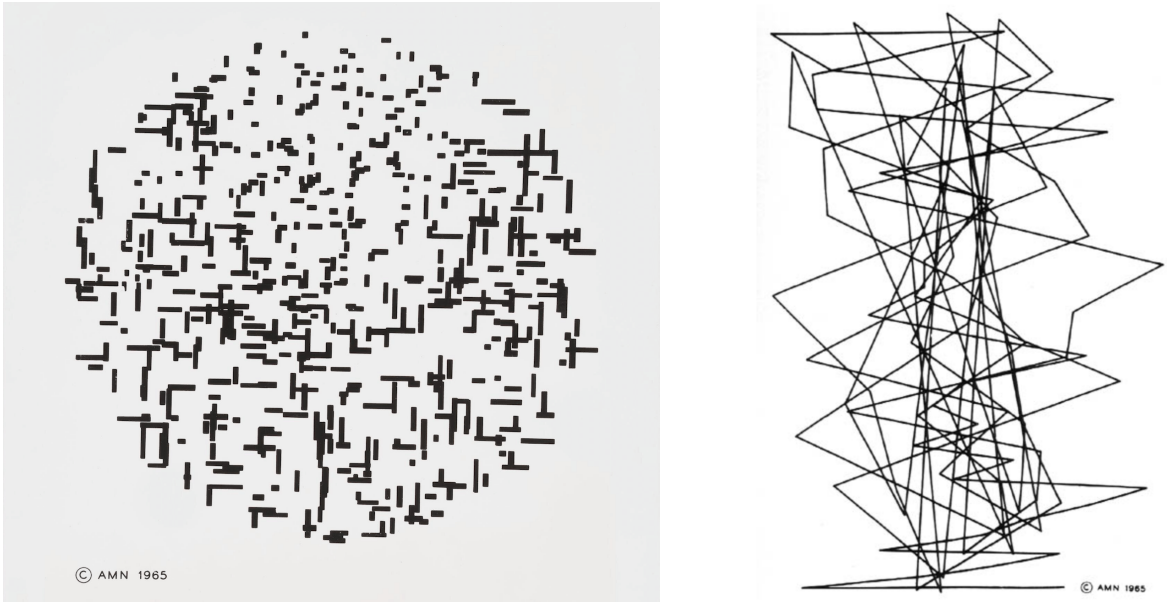
4 Challenge: Digital Forgeries

In the 1960s, 70s and 80s a number of pioneers experimented with using computers and code to create art. Some of them were scientists and engineers who had rare access to computers and were able to tinker. Later, some were early adopters who experimented with early personal computers. Some used plotter-style devices to physically draw their work onto paper (which many digital artists still do today).

I've selected some examples of their work and included them in figures later in this document, along with their names. Pick one of your favourites, or search online to find other 20th century digital art, and try to replicate it in p5.js. Ask for help if you need advice on how to achieve something. I've chosen examples that use simple geometric shapes in unusual but repeating patterns, which p5.js is great at working with.

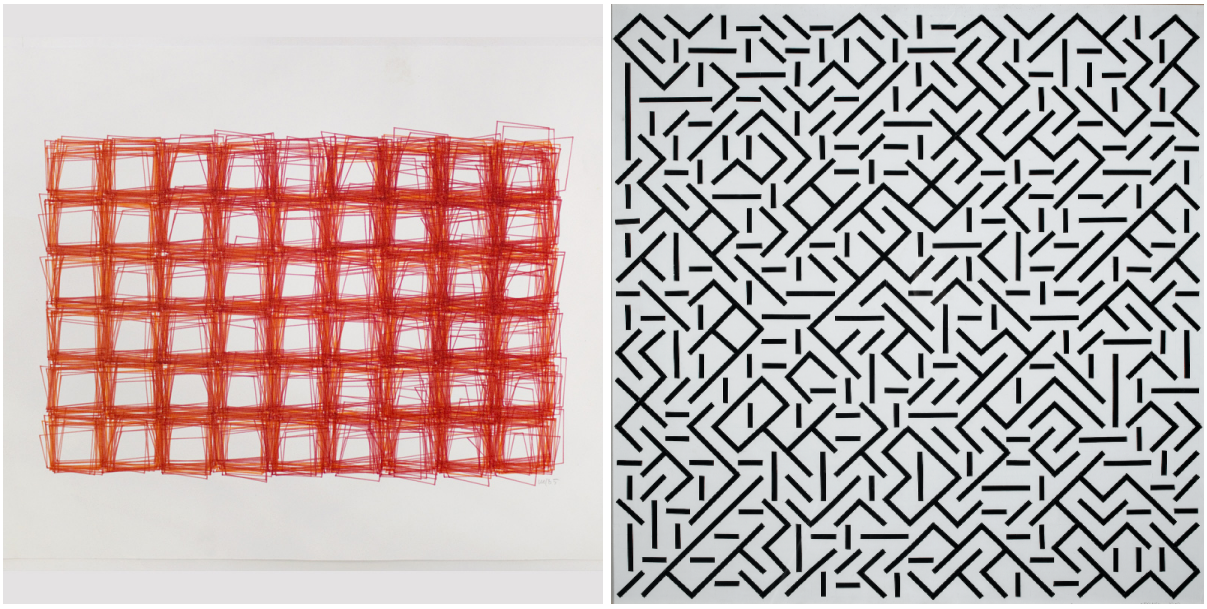


(a) "P-197" (left) and "P-300b" (right) by Manfred Mohr.

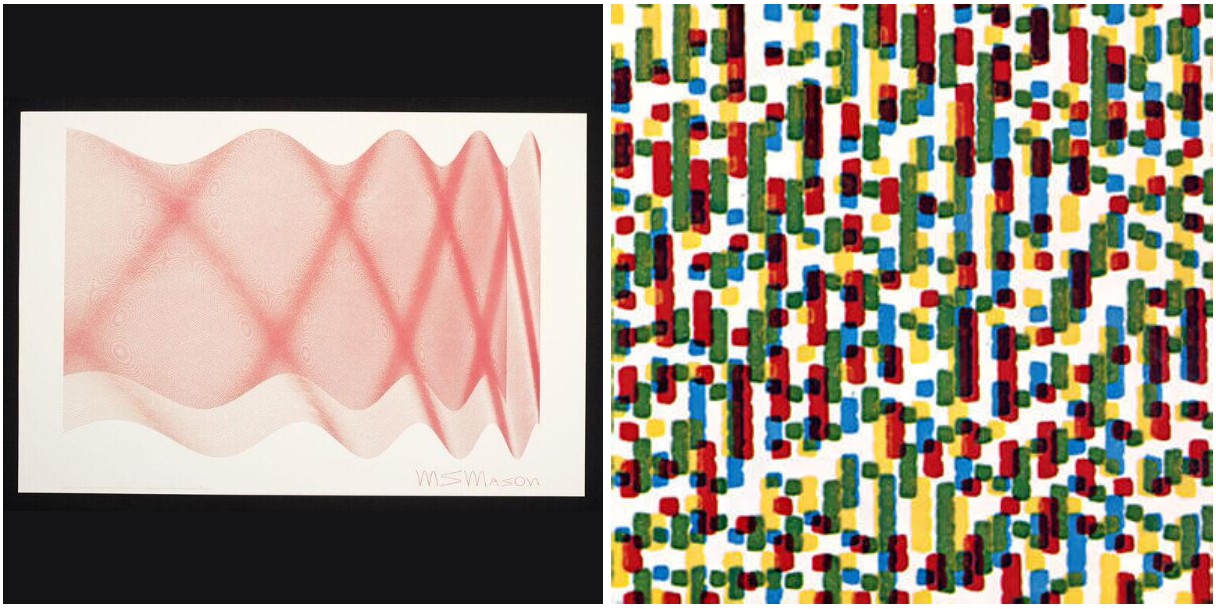


(b) "Computer Composition with Lines" (left) and "Gaussian Quadratic" (right) by A. Michael Noll.

Figure 1: Digital art by artists from the 1970s and 1980s.



(a) Unknown title (left) and "Four Randomly Distributed Elements" (right) by Vera Molnár.



(b) "Asymmetry (red)" (left) by Maughan Sterling Mason and unknown title (right) by Béla Julesz. For Asymmetry (red), I recommend looking up the original work on the V&A website - it uses Moiré patterns to create interesting visual effects.

Figure 2: Digital art by more artists from the 1970s and 1980s.

5 What Next

5.1 Share your work with me!

I'd love to see what you make, and I'd also love to be able to share people's work at future department events, or show to students in future years for inspiration! I would love it if you shared what you make - anonymously if you prefer, or with your name for full credit.

Ways you can share your work:

- Email the code, or a link to your p5.js sketch (mike.cook@kcl.ac.uk) - you can find sharing options in File → Share on the p5js editor.
- Post it to GitHub and share with me (username: gamesbyangelina or just email a link)
- If you want to submit anonymously, put your code somewhere like pastebin and send it anonymously in this form: https://forms.gle/qh8UWnqW1fNJyfxz8.

5.2 Sharing with other people

p5.js allows you to save and share your work with others, if you register an account and log in. You can also use the `save()` method to capture screenshots. There are lots of ways to share your work online, including the usual subreddits and Discord servers, as well as tags/keywords like #generative and #p5js. Every January there is also a generative art event called #genuary where people post digital art daily. See `https://genuary.art` for more information, and for optional daily challenges.

5.3 Further Reading

p5.js is a web adaptation of Processing, a Java-based offline tool that has almost identical syntax and structure. Because p5.js is so lightweight and can be run in a browser, it has become more popular over time, but you can find the original Processing and documentation online at https://processing.org/. Both Processing and p5.js are maintained by the Processing Foundation.

p5.js has a lot of libraries and add-ons written, including code for using machine-learned models for things like gesture recognition, video process, games and more. We'll be looking at some of these in future weeks.

The Coding Train is a series of programming tutorials by Dan Shiffman, an NYU Professor. Although some of his content is aimed at new programmers, there's hundreds of videos and projects that extend to very advanced topics and interesting creative projects, and most of them use p5.js. I highly recommend checking it out: https://thecodingtrain.com/.

6 Tips

6.1 Useful Tip - Shapes

p5js has lots of basic shape functions like `rect`, `circle` and `line`, but it can also draw shapes through a description of their points. Look up `beginShape` on the p5js docs or ask me for more information.

```
//Begin drawing the shape.
\beginShape()
//Add a point at (10, 10)
\vertex(10, 10)
\vertex(20, 10)
\vertex(20, 20)
\vertex(20, 10)
//Stop adding points and interpret them as a shape.
\endShape(CLOSE)

/*
    endShape can take different arguments to do different things with the points
    you give it. LINES interprets the points in pairs as defining the start and end
     points of lines.
\endShape(LINES)
*/
```

`vertex()` can only be called between `beginShape` and `endShape`. There are other similar methods you can look up, like `curveVertex` or `bezierVertex`. This is useful for drawing irregular shapes, connecting points together, and lots more. Try putting random co-ordinates into calls to `\vertex`!

6.2 Useful Tip - Pushing and Popping

A lot of drawing operations are much easier to do if we are drawing centered on the origin point (0,0) - like rotation. There are also methods we can call that change things about p5js that we don't want to persist, like scaling the drawing context. p5js has a feature to help with these situations! `push()` pushes a new set of systems settings onto a stack. This means that any changes you make to p5's draw settings are made to this new pushed system setting. It'll stay on the stack until you call `pop()`, which removes them and goes back to the previous stack of settings.

In the below example, I use `push()` in a for loop. I then call `translate(x,y)` which moves the origin to the co-ordinates I give it. Then I `rotate(x)` the canvas by 45 degrees. When I call `rect(x,y,w,h)`, even though I'm telling it to draw at (0,0), because I moved the origin with `translate()` it appears at the location we translated to. Afterwards, I call `pop()` which removes all of these changes, so if I do any more drawing afterwards it forgets all of the rotations and translations I did.

```
angleMode(DEGREES)

for(var i=0; i<4; i++){
    push()
    translate(10+i*20, 10)
    rotate(45)
    rect(0, 0, 10, 10)
    pop()
}
```

You can see the result in Figure 3 below.

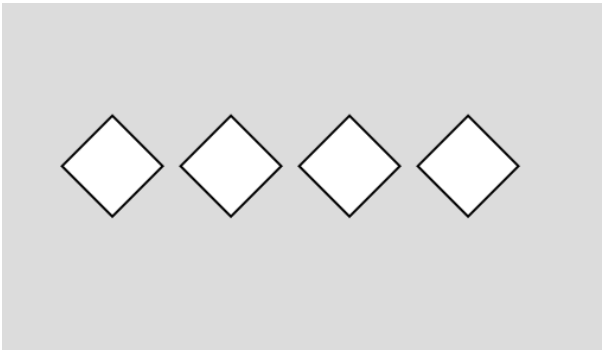


Figure 3: Four squares, rotated and positioned using pop() and push().

6.3 Useful Tip: HSB Space

By default, p5js uses RGB colour. This means that you make a colour by giving it values between 1 and 255 for each of the components of a colour - red, green and blue.

```
//Max red, no blue or green = bright red
color(255, 0, 0)
//Adding a fourth number sets the opacity (how transparent the colour is)
color(255, 0, 0, 127)
//Providing a single number is interpreted as repeating that number three times, i
    .e. you get a shade of grey
color(0) //black
color(255) //white
color(100) //dark grey
```

You can set p5js to use HSB (Hue, Saturation, Brightness) instead of RGB, by calling `colorMode(HSB, 100)`. The second argument, 100, sets the maximum value of any of the three components (instead of 255, the default for RGB). HSB works differently to RGB. The first value, Hue, is entirely in charge of the shade of colour something appears, like red or green. Saturation and Brightness affect how strongly the colour is expressed. It's a little hard to express in words, but play around with it and you'll see how it differs. Figure 4 shows a visual depiction of this.

There are lots of advantages to using HSB over RGB, but one major one is that it is a lot easier to change colour algorithmically and still retain control over its style or complementarity. For example, we might want to pick random colours that are all the same intensity. Doing this with RGB values is tricky because intense colours are not always in the same part of the RGB space. But in HSB, we can randomise hue and keep saturation and brightness the same. Figure 5 shows an example of two colours picked to be complementary (opposite one another on the colour wheel) using HSB.

```
colorMode(HSB, 100)
//Random hue, strongly expressed
color(random()*100, 80, 80)
//Random hue, lighter/more faded tone
color(random()*100, 20+random()*10, 80)
//Two colours that are opposite on the colour wheel
var h1 = random()*100
var h2 = (h1+50) % 100
```

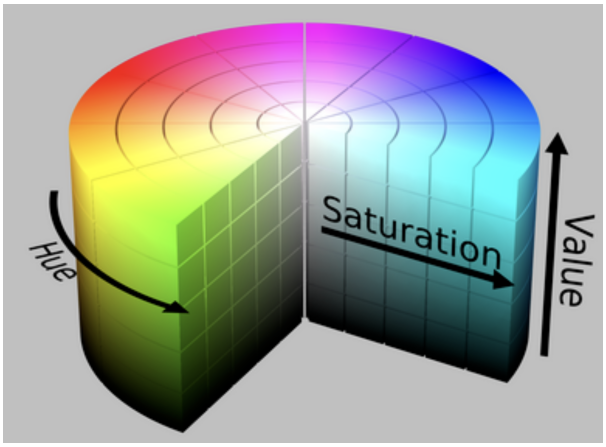


Figure 4: A visual representation of HSB (also called HSV) colour space.

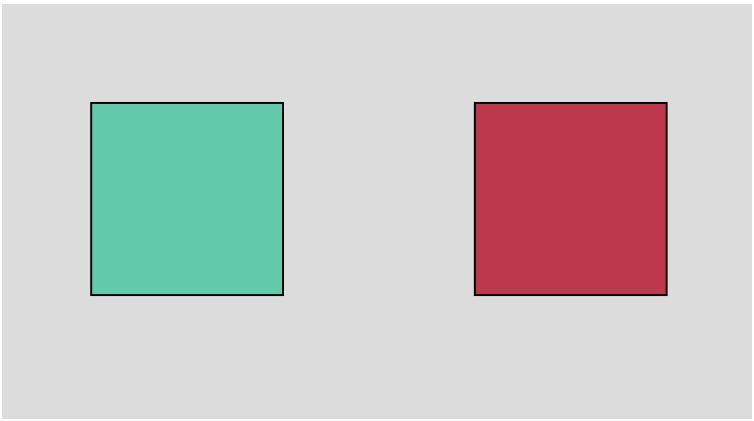


Figure 5: Two complementary colours, generated randomly by picking two colours on either end of the hue range.

```
9  var c1 = color(h1, 80, 80)
10 var c2 = color(h2, 80, 80)
```