OpenLearn

The Open University

Algorithmic Design

Algorithmic Design

OpenLearn | Free learning from The Open University

About this free course

This free course is an adapted extract from the Open University course T190 *Design practices*..

This version of the content may include video, images and interactive content that may not be optimised for your device.

You can experience this free course as it was originally designed on OpenLearn, the home of free learning from The Open University – https://www.open.edu/openlearn/science-maths-technology/design-innovation/algorithmic-design/content-section-0

There you'll also be able to track your progress via your activity record, which you can use to demonstrate your learning.

First published 2025.

Intellectual property

Contents

Introduction

Unleash your creativity and discover the fun of designing with code in this free OpenLearn course, *Algorithmic Design*.

In this hands-on course, you'll learn to think algorithmically and explore the aesthetics of code through playful experimentation with JavaScript and the p5.js library. You'll engage with design ideas such as repetition, modularity, and serendipity to generate unique, personalised wallpaper patterns.

By the end of the course, you'll have created beautiful, algorithmically generated wallpaper that showcases your new design skills – and adds a personal touch to your home.

It will be useful if you have some graph paper for drawing your tiles and access to an A4 printer (black and white or colour) to print out your wallpaper design.

This OpenLearn course is an adapted extract from the Open University module T190 *Design practices*. If you enjoy this introduction, consider enrolling in the full BDes qualification. It's designed to nurture your creativity, develop your problem-solving abilities, and help you turn ideas into action. You'll graduate with a respected degree, a professional portfolio, and the technical and creative confidence to launch your design career.

Learning outcomes

After studying this course, you should be able to:

- learn core design skills that allow you to experiment and play with algorithms
- learn how algorithms can be used to create visually compelling outcomes
- develop your creative judgement regarding the use of repetition, serendipity and modularity as part of your design process.

1 Introduction to algorithmic design

Figure 1

> In the emerging, highly programmed landscape ahead, you will either create the software or you will be the software. It's really that simple: Program, or be programmed. Choose the former, and you gain access to the control panel of civilization. Choose the latter, and it could be the last real choice you get to make.
>
> (Rushkoff, 2010, pp. 7–8)

This course explores how you can use computers and algorithms to design in a creative way.

Let's be clear, we are not just talking about using computers for familiar tasks like editing photos. In this course, you will be going deeper into the logic of computation and getting your hands dirty by tweaking and experimenting with algorithms to create designs.

But what is an **algorithm**? An algorithm is just a set of instructions on how to carry out a task. For example, a recipe is an algorithm with a sequence of steps:

1. Mix butter with sugar.
2. Add vanilla extract.
3. Add eggs (one at a time).
4. Add flour with the baking powder and milk.
5. Preheat oven to 175 °C.
6. Add the batter into the cupcake liners.
7. Bake the cupcakes for 15-20 minutes.
8. Let them cool, add icing, and serve.

When a computer encounters an algorithm, it progresses through a sequence of instructions to generate an output, such as the cupcake recipe above.

In the present day, algorithms have become ubiquitous, seamlessly integrated into every aspect of our recreational and professional lives, whether or not we directly engage with them. Algorithms run cars, traffic lights, and monitor air quality. Algorithms power the banking system and keep dialysis machines running. When we are online, the algorithms from companies such as Google, Amazon, and Facebook/Meta let us find information, recommend products and provide news based on what we like and who we are.

But why should designers care about algorithms? Well, today design can't be separated from computation. Algorithms can be used creatively to generate new ideas, shapes, and patterns. In today's design industry, computers are an essential tool for almost every designer, regardless of whether they are creating buildings, computer games, or working on tapestries and theatre lighting.

In this course, you will be using computation and algorithms not just as tools, but as a way to think about design. The logic of algorithms is different from human logic and processes. *Thinking with algorithms changes how designers work*. Algorithmic ways of thinking have spawned algorithmic art as well as computer games and provide new ways for people to interact. They have transformed traditional analogue forms of art and design, such as painting, sculpture, and drawing, to introduce new kinds of visual complexity and textures.

By thinking and designing with algorithms, this project will give you a glimpse of the underlying mechanism of the world, and propel you to the cutting edge of where the design discipline is heading.

Figure 2 Algorithmically generated wallpaper pattern

At the start of the course in Section 3: Playing with Code, you will learn some of the basic principles and skills of algorithmic design such as how to draw shapes and control colour by tweaking some code examples.

In Section 4: Making Tiles and Section 5: Making Patterns, the skills will take you deeper into programming by giving you opportunities to design your own algorithmic tiles and

then arrange them into a pattern. As you work through the skills, you will generate ideas that will contribute to your final wallpaper design.

In Section 6: Identify a place for your wallpaper design, you will concentrate on creating an algorithmic wallpaper design for your own home.

Across all sections, you should:

- keep a **codesnippet** text file where you will collect the code for your tiles

2 Algorithmic Design

Why learn algorithmic design?

What kind of design most interests you? It may be that you have never thought that algorithms could have any part to play in design. However, these ways of making and thinking about design are becoming increasingly important in the design world.

Watch the following interview with Aleksandra Jovanić to get a sense of how a contemporary artist and designer works with algorithms.

Video content is not available in this format.

Video 1 Design practioner interview: Aleksandra Jovanić talks about working with algorithms

Highlights from the interview

Here are a few points from Aleksandra's interview to consider.

- Aleksandra uses computer code to create systems that produce visual and audio outputs.
- Her work combines historical influences with modern programming techniques. She draws inspiration from old media devices, such as the phenakistoscope, to reinterpret them via code. In this course, you will adopt a similar approach, using the history of computing and patterns to inspire your creation of modern algorithmic wallpaper.
- Aleksandra's creative process involves using randomness within defined constraints. This allows her to generate many variations of a design, resulting in outcomes that are exciting and surprising even to her. This algorithmic way of designing with variability and randomness is something you will explore throughout this course.
- Aleksandra observes that algorithmic design is starting to include natural, tactile, and figurative designs as well as abstract geometric patterns.

This course involves learning to play with algorithms by tweaking code. But it will not attempt to teach you to program! That is too big a task for this short course, but if you are interested in further developing this skill, links will be provided to appropriate resources.

In this course, you will learn how to tweak code to make it do what you want. You won't necessarily understand all the code you will be working with, but, along the way, you will get a feeling for the potential of algorithmic design and learn some of the fundamental principles of algorithmic design.

This course should help you to look at all kinds of design in a new way, and might help you with many design problems and interventions. While people sometimes claim that computers take control away from us, computation actually opens up many new possibilities. Working with algorithms means that designers have to make more – not fewer – choices!

Figure 3 The complex curves of Zaha Hadid Architects' Heydar Aliyev Center in Azerbaijan were only possible using algorithmic tools

A major part of algorithmic design involves developing judgement and learning how to make aesthetic and pragmatic decisions as a designer. So, learning about algorithmic design will help you to develop your personal judgement about what are good and bad uses of algorithms – what works, and what doesn't.

Since 2022, the appearance of AI tools has radically transformed graphic design, illustration, and architecture through the use of automatically generated images and layouts. Figure 4 looks like it was drawn by hand, but it was actually generated by artificial intelligence (AI) via the software tool Adobe Firefly. It was created using the text prompt: 'What does algorithmic design look like?'. Acquiring an understanding of the logics and principles of algorithms will equip you to navigate the practical and ethical design challenges that AI tools will bring. Crucially, you will begin to understand some of what goes on behind the scenes when designers use ready-made design tools.

Figure 4 An AI-generated idea of what algorithmic design looks like

Even if, in the future, you don't intend to work with computers in such an in-depth way, having knowledge of these skills is still useful when working with other professionals, such as programmers and engineers, who you are likely to work with when collaborating on large design projects. Understanding how algorithms work gives you an appreciation for the technical aspects and roles in design, and the labour that takes place behind the scenes in complex design projects.

3 Playing with code

The aim of this course is to *Design algorithmic wallpaper for a place in your home*. You will be playing with code to make algorithmic tiles and then arrange those tiles into a larger wallpaper pattern (see Figure 5).

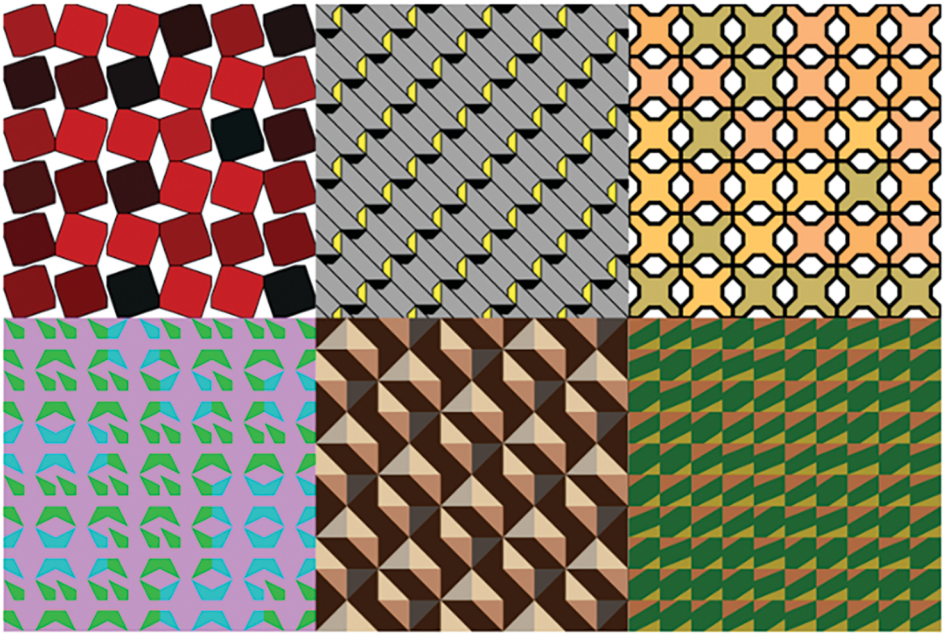


Figure 5 Six algorithmically generated wallpaper designs created using code

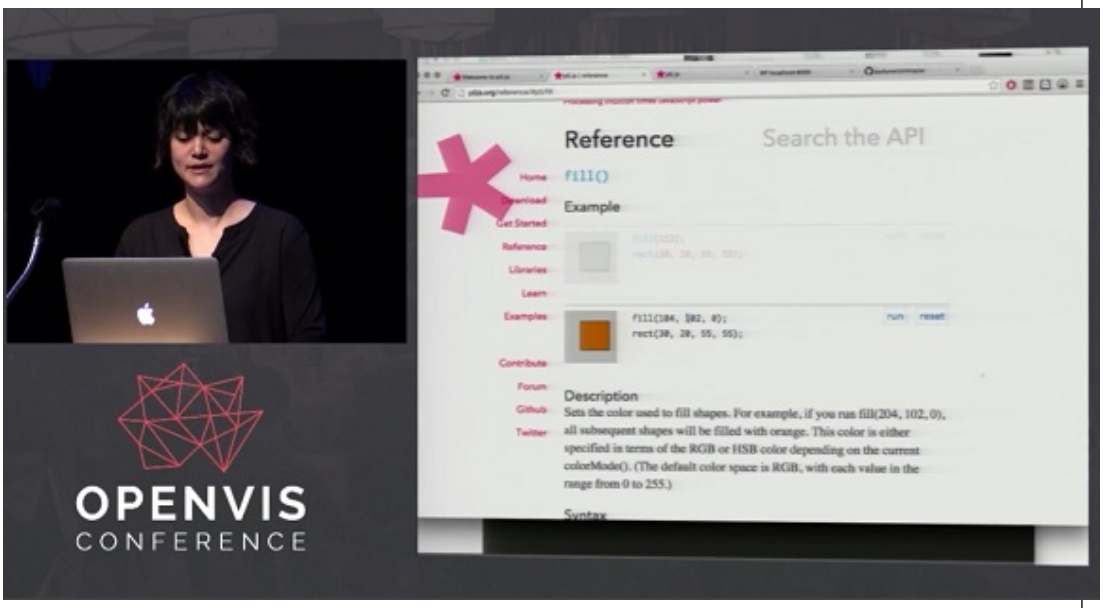
In this project, you will be using **p5.js**, which is a programming library that uses the JavaScript programming language and **syntax**. There are many different programming languages available; for example, you may have heard of Python, Java, or C++. Every language has different strengths and limitations and they have a slightly different syntax – meaning the grammar of commands that they use.

Figure 6 The logo of the p5.js programming language

We have chosen to use p5.js because it is based on Processing, an open-source software, which was developed for artists and designers for its ease of use and its ability to quickly get visual output onto the screen.

Here is a talk by Lauren McCarthy, one of the founders of p5.js, about the origins and spirit of the p5.js language. It is a lovely video, but you only need to watch the first 14 minutes or so.

Video content is not available in this format.
Video 2 Lauren McCarthy discusses the making of p5.js



P5.js allows us to embed a code editor into the module website, so you don't need to install any software on your computer to put your learning into practice. Any p5.js code you create can be embedded in any website, so anything you create in this topic can be shown on your personal website for the whole world to see.

There are many useful p5.js resources online, and the best place to start is the p5.js website. The website has many examples and tutorials, as well as a detailed reference section that lists all the syntax of the language.

There is also a very good p5.js web editor, which gives you lots of space to see and edit your code. You might find it useful to use this editor to work on the code that has been provided for you as part of the learning material, rather than using the embedded code editor.

Throughout this topic, there are some conventions used in the example code which aren't strictly necessary, but are good coding practice and will help with translation into other coding languages. For example, you will notice that code is indented and that semicolons are frequently used at the end of a line of code, but there are only a few occasions where this is strictly necessary in JavaScript.

3.1 Principle 1: Tweaking code

Code is just text. It is written in a strange language and grammar – its syntax – but it is still just text. The language and structure are strange because they need to be interpreted by a computer and, despite what you hear in the media, computers are very stupid and can only do what they are told.

As well as the code syntax, there are certain technical terms that you will come across, such as **parameters** and **variables**, that have specific meanings when talking about code. Unfortunately, this is unavoidable because there are technical concepts associated with programming that are very distinctive and don't have common language alternatives. We will try to avoid jargon as much as possible and explain terms as you go through these principles, but you might find that there are some terms that are not explained in enough detail. A good place to start is by searching online, where you'll find a wide range of tutorials. If you enrol as a student at the OU, you'll also have access to tutors and subject specialists who can provide additional support.

Initially, you may find it daunting to read a piece of code, but just like any language, the more you look at it and the more you use it, the more familiar it will become. And, just like any other text, code can be edited. The kind of edits you will have to make are tweaks – small changes to the code that have big impacts on what is created.

As you work through the coding activities, you will be encouraged to tweak the text of code inside the p5.js editor frames. Try not to be nervous about doing this; you are in a safe environment, which has been designed to allow you to experiment with what is possible with code. You cannot break your computer, or your network, or the internet by editing the code in these exercises! The worst you can do is stop the program from running, and that can be easily fixed by undoing the changes you made or reverting to the provided code.

The following program is a p5.js editor frame. It doesn't do anything right now. You will see many of these as you go through this course, but they will all be filled with example code that gets you started. The `Play` and `Stop` buttons make the code restart or stop. The canvas is the space to the right of the code that appears when you press `Play`.

You can undo any changes you make by using the `Undo` and `Revert` buttons that appear after a change has been made.

Interactive content is not available in this format.

Program 1 An empty p5.js editor frame

The words 'programming' and 'coding', and 'program' and 'code' are used interchangeably throughout this project. Yet an important distinction should be made between a **program**, which is fully functional and standalone, and a **codesnippet**, which is a chunk of code that does a small task as part of a larger program.

One of the main tasks you will be doing when tweaking code is moving little snippets of code around, sometimes to repeat a shape, or to take a codesnippet that you made previously and insert it into another program. Using codesnippets is a modular way of working with code, and it is a very important practical and conceptual skill to learn.

To help you with this, you need to create a codesnippet text file on your computer, where you will save any codesnippets that you are working with.

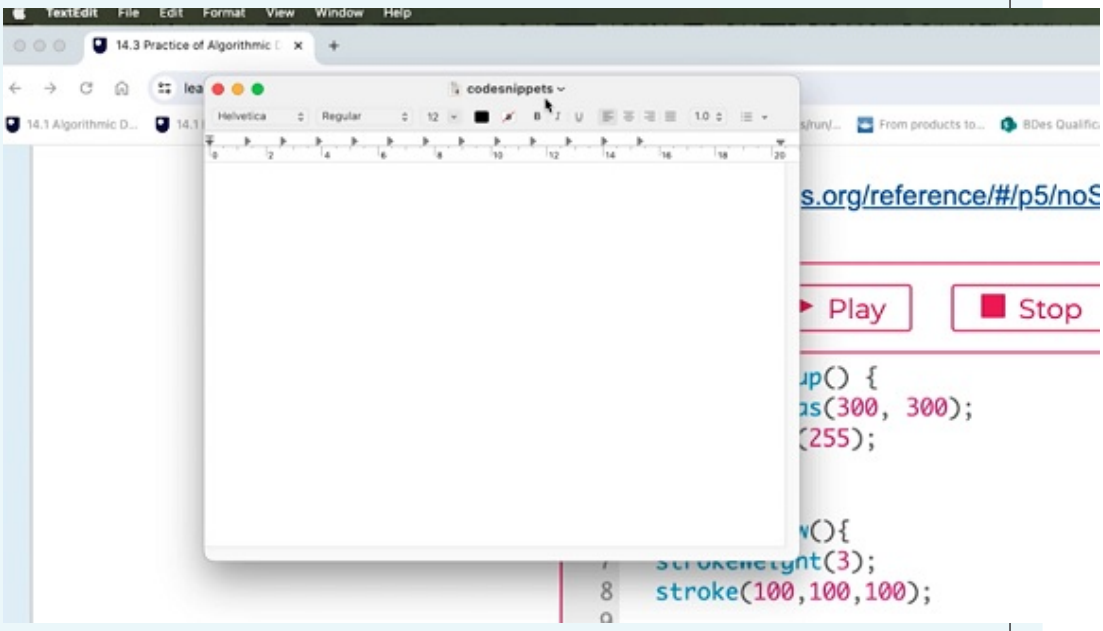
Don't be precious about this file; it's just a working place to store ideas, a bit like a sketchbook full of chunks of code. You will use this file for storing, copying, and pasting snippets into different programs.

Activity 1 Make a codesnippet text file

Allow 5 minutes to complete this activity

Follow the instructions in the following video to create a codesnippet text file on your computer where you will store little chunks of code.

Video content is not available in this format.



Video 3 How to make a code snippets file

3.2 Principle 2: Hello World!

In the following activity, you will run and tweak your first p5.js program.

Since the 1970s, ‘Hello, World!’ has been the name of the first program that every student learns when they encounter a new programming language.

The editor window below is running a program that draws circles as you move the mouse cursor around the canvas.

Try interacting with the program by moving your mouse cursor around within the grey canvas box area on the right side of the editor window.

Interactive content is not available in this format.

Program 2 Hello World!

You can stop the program by clicking the `Stop` button in the editor window and restart the program by clicking `Play`.

Fun, right? You just ran and played with your first p5.js program!

Let's take a look at the code a bit more closely. The Hello World! program is made up of three different functions.

A **function** is a block of code that carries out a specific purpose. All of the code that makes up the function is contained inside curly brackets: { }. Notice how the code inside the curly brackets is indented to indicate that it belongs to the function. Functions are very useful programming structures because they are discrete chunks of code that you can reuse in a modular way.

setup() function

The first function is called `setup` and runs once when the program first starts playing and defines some parameters that won't change.

```
function setup() {
createCanvas(300, 300);
background(100);
}
```

In this case, the `setup` function creates a canvas that is 300 × 300 pixels wide with this code:

```
createCanvas(300, 300)
```

Note, the comma between the two numbers is important or the program won't run; however, the space after the comma is not important and will not affect the running of the program.

The function also sets the colour of the canvas to grey using this code:

```
background(100)
```

The value `100` is a mid grey that is roughly halfway between 0 (black) and 255 (white). Colour will be discussed in more detail in Principle 4.

draw() function

The second function is called `draw`, and it keeps looping over and over, repeating the code within it. In this case, it keeps drawing a new circle at the positions of the mouse cursor.

```
function draw() {
circle(mouseX, mouseY, 20);
}
```

The position of this circle is defined by the horizontal position of the mouse cursor, `mouseX`, and the vertical position of the mouse cursor, `mouseY`. The number `20` shows that the diameter of the circle is set to 20 pixels.

If you are ever confused about a piece of code, there is a reference for each command on the p5.js website, for example:

- p5.js reference: circle

You will find links to other relevant p5.js references as you work through the following principles.

keyPressed() function

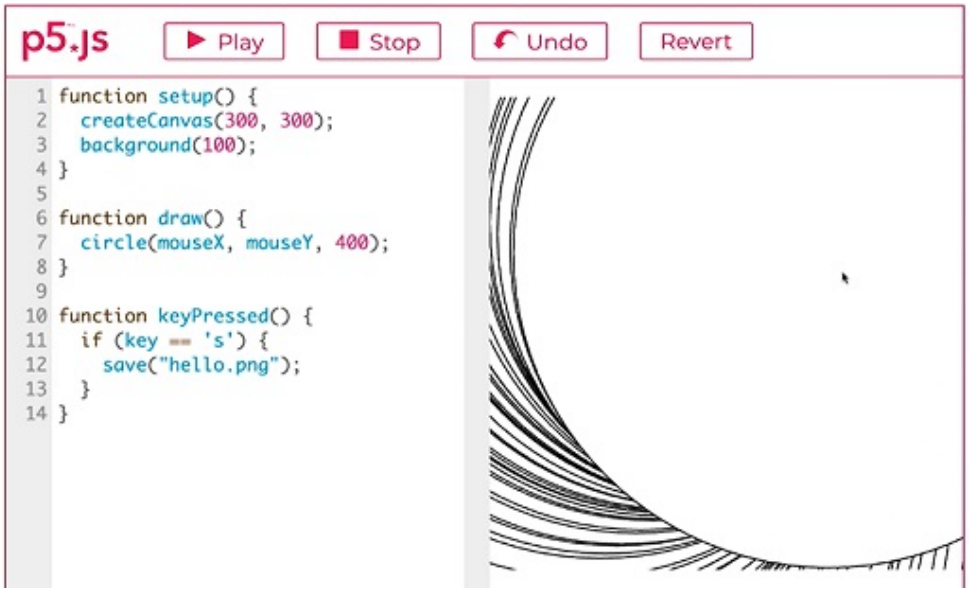
The third function is called `keyPressed`, which is waiting for any keyboard input. If it detects that the `s` key has been pressed, then it saves an image of the current canvas called `hello.png` to the computer.

In summary, while the program is running, it draws new circles at the cursor position every time the program loops. If you press the `s` key, it saves an image of the canvas.

In the next activity, you are going to make your first coding tweak! If you get stuck, the following video should help you out.

Video content is not available in this format.

Video 4 Playing with the Hello World! code



Activity 2 Change the size of the circle and save a small artwork

Allow 10 minutes to complete this activity

Scroll back up to Program 2 Hello World! and go to line 7 to find the circle diameter. Change the pink-coloured number 20 to a number between 1 and 200.

Once you have made the change, press the `Play` button and move the cursor over the canvas to see what effect this had. If you get lost and want to get back to the initial example code, press the `Revert` button.

Keep playing with this example until you have drawn something that you like. Use the right mouse button to save an image of the canvas onto your computer. You can also press the `s` key to save the image.

Well done! You made your own algorithmic artwork.

3.3 Principle 3: Shapes and Coordinates

In this principle, you will learn how to draw some shapes and how to position them on the canvas using the coordinate system in p5.js. In all these examples, you will be using a square canvas size of 300 × 300 pixels, so all our shapes always need to fit into that space.

X and *y*-coordinates in p5.js

p5.js uses an *x*- and *y*-coordinate system to identify locations on the canvas and to position shapes.

The x/y-coordinate system

The origin point for the coordinates starts in the top left-hand corner of the canvas with 0,0.

The *x*-coordinate is the horizontal distance in pixels from the left edge, while the *y*-coordinate is the vertical distance from the top edge.

The *x*-axis is left to right. The *y*-axis is top to bottom.

In Figure 7, I have used some graph paper to draw a dot at the position *x*2 and *y*3 which is written as `point(2, 3)` in p5.js code. The *x*-coordinate always comes first, followed by a comma and then the *y*-coordinate.

If you forget the comma, you will get an error in your program. Note that the coordinate system starts in the top left-hand corner.

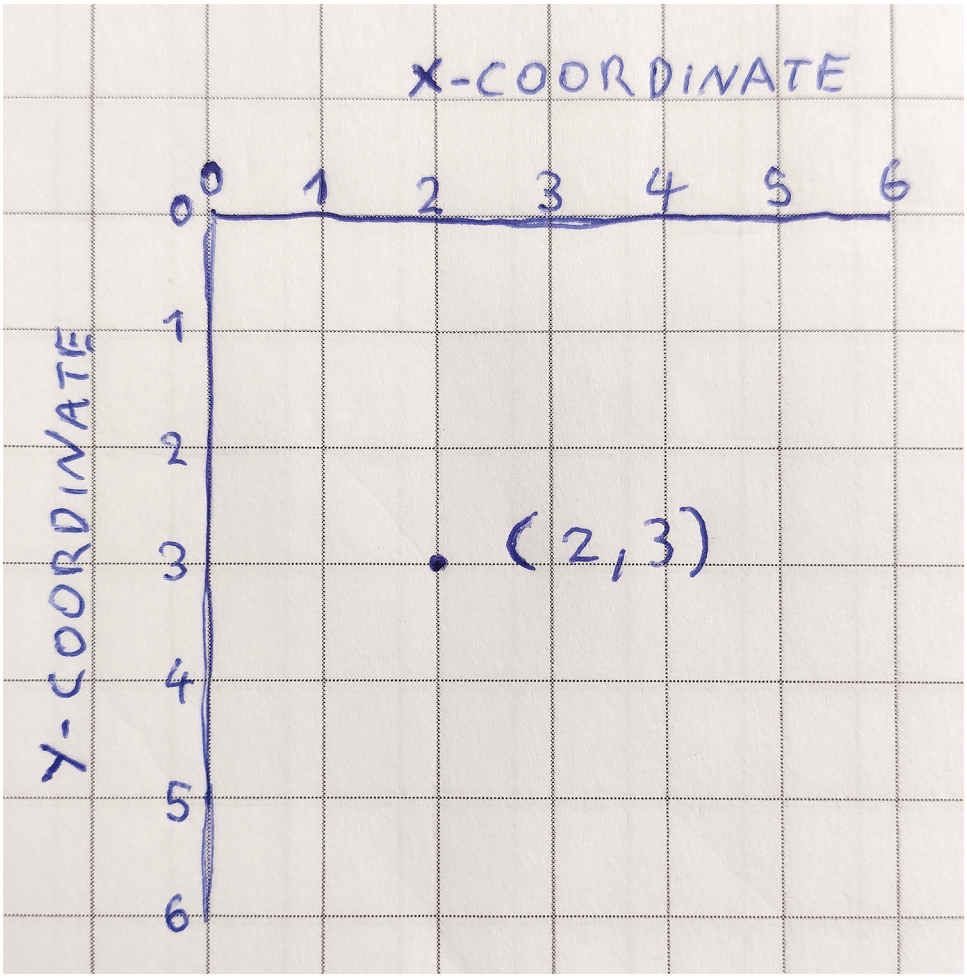


Figure 7 The *x*/*y*-coordinate system drawn on graph paper

Point

Drawing a point is easy in p5.js. It has two parameters for the *x*- and *y*-coordinates, `point(x, y)`:

- x: the number that specifies the *x*-coordinate
- y: the number that specifies the *y*-coordinate.

(p5.js reference: point)

Square

The square shape has three parameters, `square(x, y, s)`:

- x: the number that specifies the *x*-coordinate of the top left corner of the square
- y: the number that specifies the *y*-coordinate of the top left corner of the square
- s: the number that specifies the width and height of the square.

(p5.js reference: square)

Interactive content is not available in this format.

Program 3 Draw squares

Change the outline

Notice that the outline of the square in Program 3 is very thin. The way to control this is by using the `strokeWeight()` command – you can put any number you like inside the brackets. I recommend a number between 1 and 10.

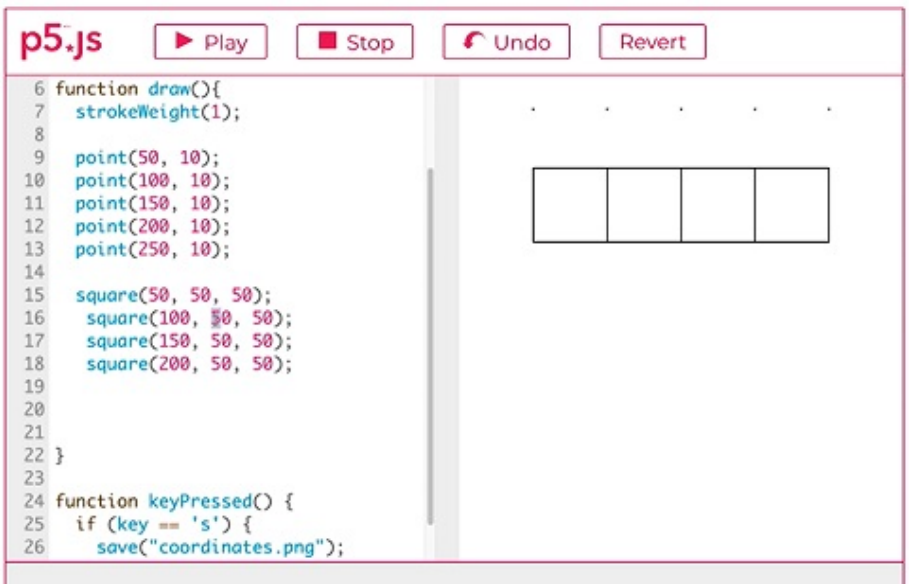
(p5.js reference: strokeWeight)

Note that it matters where in the code you put the commands. The program runs the code sequentially, line by line from top to bottom. If you add `strokeWeight()`, then all the subsequent shapes will have that property applied. This property persists for all shapes on the next loop. If you want the shapes to have a different outline, then you need to add another `strokeWeight()` command so that any subsequent shapes receive that property.

Watch Video 5, which shows you how to use the coordinate system.

Video content is not available in this format.

Video 5 Drawing squares using the coordinate system



Activity 3 Draw your own squares and tweak the outline

Allow 15-20 minutes to complete this activity

Use Program 3 Draw squares to draw multiple squares at different locations. You need to copy and paste the square command to create some additional squares and change the *x*- and *y*-coordinates. Remember that the origin point is in the top left-hand corner.

Change the `strokeWeight()` to give the points and squares a thick outline.

Keep playing with this activity until you understand the *x*- and *y*-coordinate system. This concept is important to understand for section 4 ‘Making Tiles’, where you will be creating tiles and wallpaper.

3.4 Principle 4: Colour

Knowing how to set colours is an important part of creative coding and will be crucial for your wallpaper project. On this page, you will learn how to use the fill and **stroke** commands to use the full colour spectrum.

By default, p5.js defines colour values using **RGB** (Red, Green, and Blue) as a value ranging from 0 to 255, where 0 signifies the absence of the colour and 255 represents the highest intensity of that colour. These three colours are the primary colours, from which additional colours can be created by blending these colours in varying ratios. For example, a yellow colour in RGB is created using a mix of red and green using the RGB values: 255,255,0.

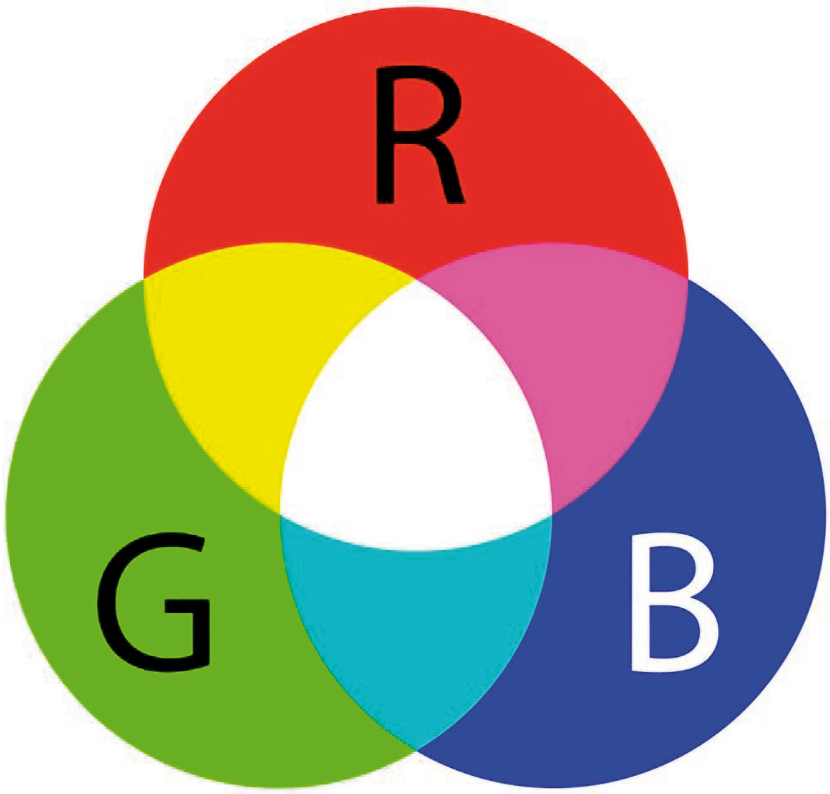


Figure 8 How RGB colours mix; the maximum value of all three primary RGB colours creates a white colour: 255,255,255

In Program 4 below, click on the small rectangular box on the canvas and you should see a colour picker appear. It allows you to pick different colours and see their RGB values at the same time.

Use the colour picker to change the colour of the square a few times. Notice how the numerical RGB values change as you select different colours.

Interactive content is not available in this format.

Program 4 RGB colour picker

Filled colour

The `fill()` command allows you to set the fill colour inside shapes. You are defining three values because you are mixing the three RGB colours together to create a single fill colour `fill(R, G, B)`:

- R = red value between 0 and 255
- G = green value between 0 and 255
- B = blue value between 0 and 255.

(p5.js reference: fill)

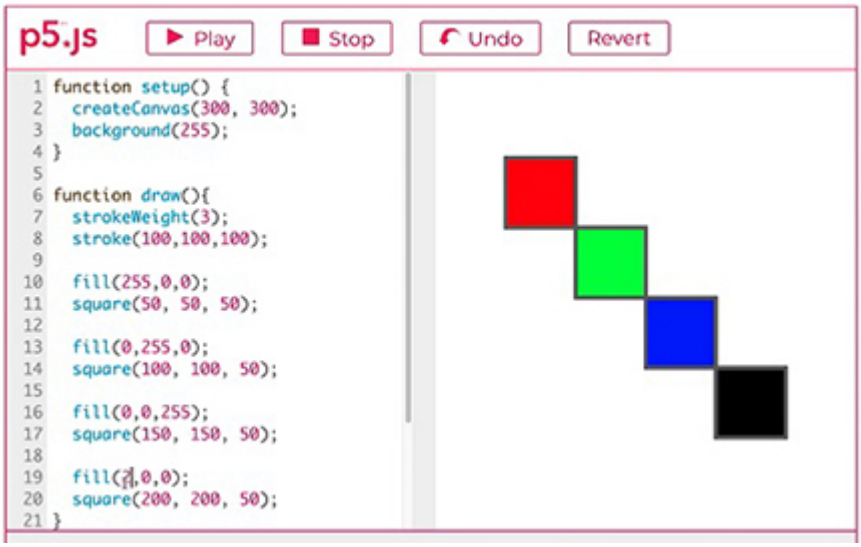
Outline colour

The `stroke()` command controls the outline colour of any shape, for example `stroke(R, G, B)` (p5.js reference: stroke).

Note that you can also use the `noStroke()` command to disable any stroke being drawn (p5.js reference: noStroke).

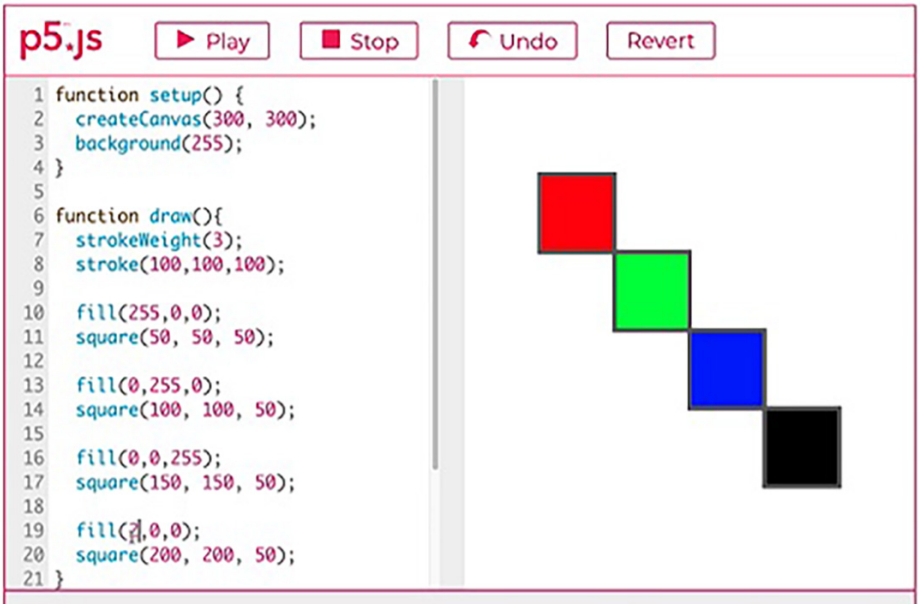
Interactive content is not available in this format.

Program 5 Recolour the squares



Video content is not available in this format.

Video 6 Changing and playing with colour



Activity 4 Recolour the squares

Allow 20 minutes to complete this activity

Use Program 5 Recolour the squares to tweak the numerical RGB colour values for the fill and outline of the squares.

Experiment with inserting `mouseX` or `mouseY` in place of the numerical RGB values as well as parameters of the squares, as shown in Video 6.

4 Making Tiles

In this section you will apply the algorithmic principles you have learned to create tiles and patterns using code.

In the first set of skills, you will learn how to create a single tile and tweak the design to create four individual tiles. In the second set, you will learn to arrange these tiles to create patterns and test them out.

Figure 9 shows the design process that you will be following.

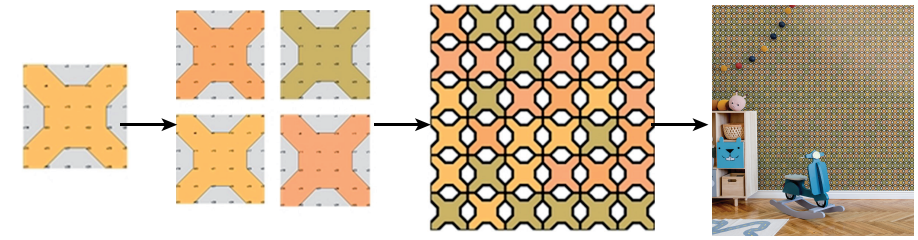


Figure 9 Design process from single tile to pattern

A tile is a shape that can be repeated to form a larger pattern. Think of the tiles in your bathroom. Maybe they are just plain white tiles, but perhaps they have some decorative shapes that create an overall visual pattern. Designing tiles requires thinking in a modular way. You need to focus on an individual tile while also thinking about how multiple tiles will look next to each other (Figure 10).

In this introduction, you will learn about the historical and contemporary context of tile and pattern design. Don't worry, in this course you won't have to make anything as complex as these examples.

(a) (b)

Figure 10 (a) This is a commercial ceramic tile design that uses geometric shapes. (b) When the tiles are arranged together, they form a repeating pattern that looks much more complex than the single tile.

Tiles and mosaics are some of the oldest pieces of design, originating from ancient Babylon and Rome, and were used in many contexts, such as public outdoor spaces as well as private indoor dining areas. Mosaics are made by arranging coloured pieces of

stone or glass to form patterns. Tiles and mosaics take many different shapes and forms but often include geometric patterns that repeat like an algorithm. It is fun to think about how you might recreate the black-and-white triangle pattern in Figure 11 using code.

Figure 11 A Roman floor mosaic with intricate geometric patterns and a figurative head of Medusa in the centre *c.*115–150 BCE

Contemporary tile design

Figure 12

People are still creating new tile designs today. Huguet is a family-owned Spanish tile manufacturer which has been making **encaustic cement tiles** since 1933. Their tiles integrate traditional tile-making techniques with creative designs by contemporary architects and designers from around the world.

The typeface tiles in the following image were crafted through a collaboration with the designer, Sascha Lobe, drawing inspiration from the modular typeface ‘Le Corbusier’. This design facilitates the rearrangement of tiles to form new words by aligning the tiles to create various letter shapes.

Figure 13

4.1 Skill 1: Create a dot-to-dot tile

In this skill, you will be creating your own tile design by drawing it on graph paper and then transferring it into p5.js code. If you don't have any graph paper, you can just use a blank piece of paper and draw a small grid for this exercise.

Many of you may be familiar with dot-to-dot puzzles (Figure 14). The goal is to draw a line connecting the dots to reveal an image. You will be using a similar method to create your own decorative shapes, which will fit on a square tile.

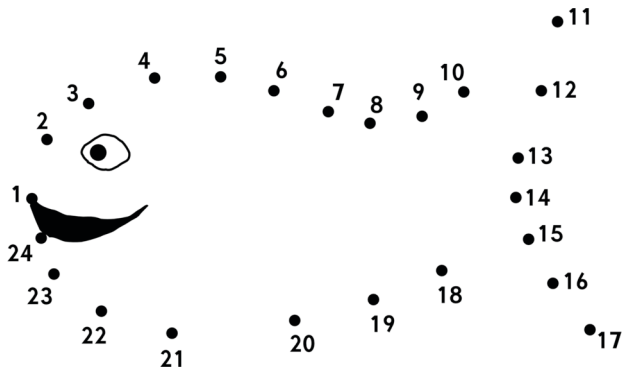
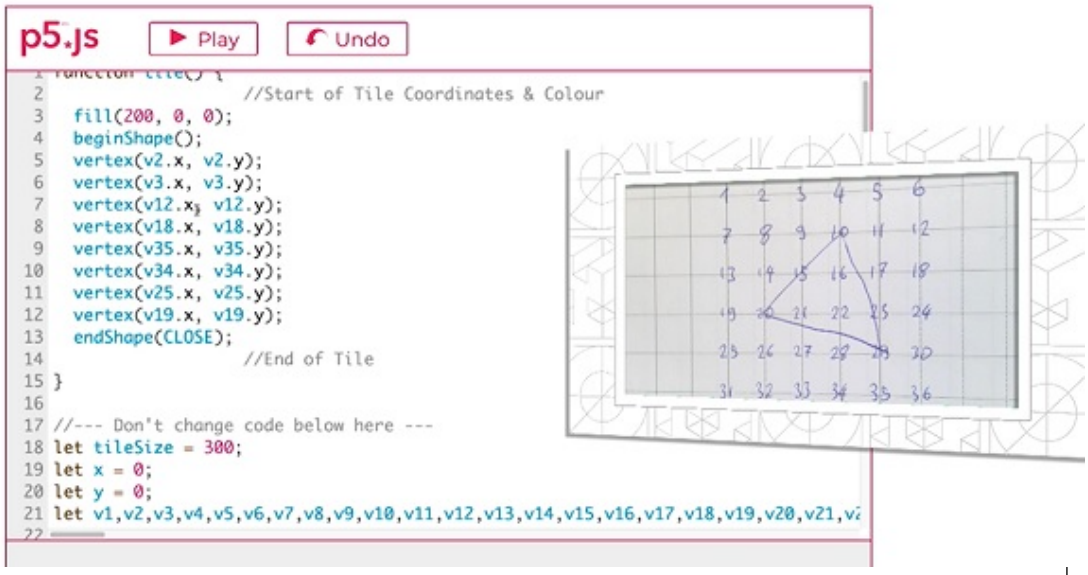


Figure 14 A dot-to-dot drawing

You are going to use the same principles as a dot-to-dot drawing to help you create unusual digital shapes.

It would be helpful if you start by drawing your shape on paper, and then you are going to transfer it into a dot-to-dot tile-maker program. Video 7 shows you how this works.

Video content is not available in this format.
Video 7 Transferring a shape to the dot-to-dot tile maker



Start by getting a piece of graph paper and creating a 5 × 5 grid. Next, number every intersection, where the horizontal and vertical lines cross, from 1 to 36, with the number sequences going from left to right. On a 5 × 5 grid, there will be six intersection points on each line.

It is easiest to explain this with an image. Figure 15 is a photograph of some graph paper on which I wrote 36 coordinate numbers on the intersections of the lines to represent the surface of a single square tile.

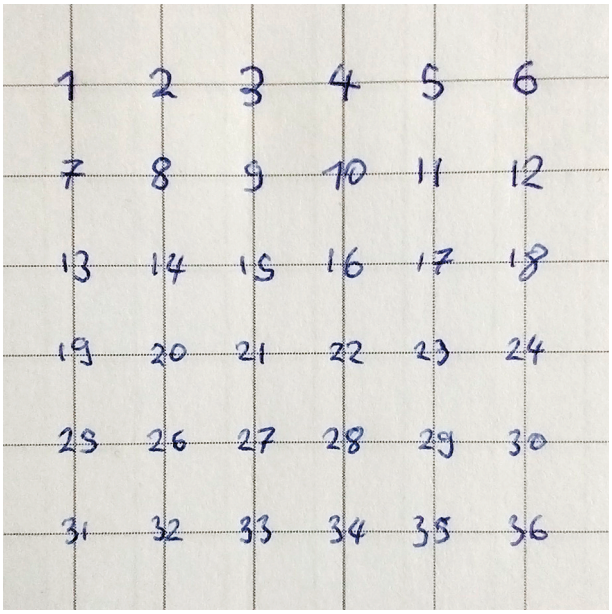


Figure 15 36 numbered intersections on graph paper

Vertices in p5.js

Just like the dot-to-dot puzzle, you will define your shape using numbers – in this case, the numbers at each of the intersections that you have labelled on your graph paper. The formal term for one of the dots in a shape is a **vertex** – the plural is vertices.

The more vertices you have, the more complex the shapes you can create; they are much more versatile than the circles and squares you have used so far.

You will tell the p5.js program that you want to make a shape by giving the command `beginShape()` and then give the series of vertex points that need to be joined. The command to finish making the shape is `endShape(CLOSE)`. You can use as many vertices as you want to make your shape.

Note, each vertex point has an *x*- and a *y*-coordinate. In this exercise, these two coordinates are always the same.

```
beginShape();
vertex(x-coordinate, y-coordinate); //one vertex point
vertex(x-coordinate, y-coordinate); //another vertex point
vertex(x-coordinate, y-coordinate); //another vertex point
endShape(CLOSE);
```

To draw a triangle you need three vertices, while a square has four vertices and a pentagon five. When using vertices, your shapes don't have to be ordinary geometric shapes; you can create more complex and funky designs.

(p5.js reference: vertex)

Creating a tile design

To create a tile design, start with the graph paper and connect some of the numbered intersections. You don't have to start in the corner with dot 1.

For Figure 16, I drew a tilted square with smoothed off corners on the graph paper. I used some yellow stickers to highlight the eight vertex coordinates that I need to write down to transcribe this shape into p5.js.

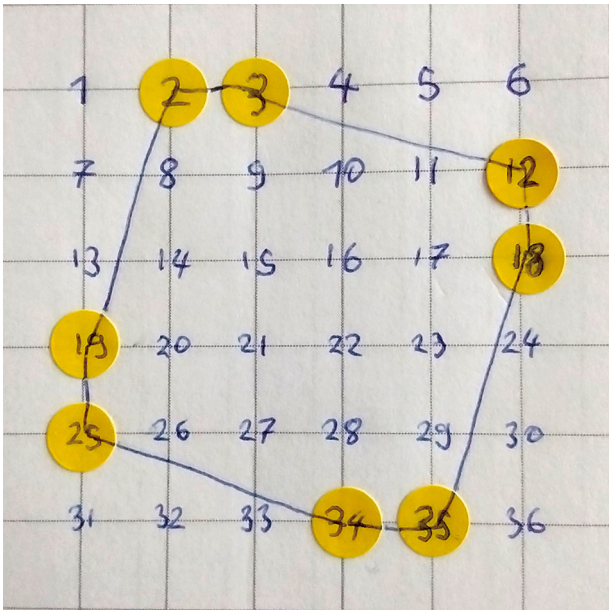


Figure 16 A tile design on graph paper

In Figure 16, I drew a shape that connects the dots 2, 3, 12, 18, 35, 34, 25, and 19. To make this shape appear in p5.js, you have to use these coordinate numbers as vertex coordinates. I started in the top left-hand corner, going clockwise, but it doesn't matter where you start, as long as you go through all the coordinates in a continuous sequence. This is the code for recreating the shape in p5.js:

```
fill(200, 0, 0); //This is the fill colour - a red
beginShape(); //This starts drawing the shape
vertex(v2.x, v2.y); //The first dot with x and y-coordinates
vertex(v3.x, v3.y);
vertex(v12.x, v12.y);
vertex(v18.x, v18.y);
vertex(v35.x, v35.y);
vertex(v34.x, v34.y);
vertex(v25.x, v25.y);
vertex(v19.x, v19.y); //The final dot in the shape
endShape(CLOSE); //This finishes drawing the shape
```

Each line of the vertex code represents a single coordinate point that is connected together to form the shape. You can add as many vertex points as you like by inserting new lines of code. You don't have to return to your starting point because the shape will automatically be closed by the command `endShape(Close)`.

The code uses variables starting with the letter `v`, followed by the number of the dot coordinate. So, dot 12 from the graph paper becomes `v12` in the program. Each vertex needs both an *x*- and a *y*-coordinate. For this program, the numbers for the *x*- and *y*-coordinates are the same. The vertex for dot 12 would be written as `vertex(v12.x, v12.y);` The command 'vertex' tells the program you are giving information about a specific point and the information in brackets gives the detailed information. Note the full stops, commas, and the semicolon after the bracket:

```
vertex(v12.x, v12.y);
```

Look at Program 6 Dot-to-dot tile maker, below. There is a single function called `tile()` that starts in line 1 that you need to make changes to. The `fill()` command in line 3 controls the colour of the shape. The shape itself starts in line 4 with `beginShape()` and ends with `endShape(Close)` in line 13. In between are the vertex points. This is where you will substitute your own coordinates to create your own tile decorations.

There is a comment in line 17:

```
//— Don't change code below here —
```

The code below this point makes the coordinate system easier to work with, but you don't have to understand how this code works or make any changes here.

Interactive content is not available in this format.

Program 6 Dot-to-dot tile maker

In the next activity, you can use Program 6 to create your own tile design.

Activity 5 Create your tile design

Allow 15-20 minutes to complete this activity

Create your own tile design on graph paper and then transcribe it into code.

- Write the numbers 1 to 36 on the intersections of a 5 × 5 grid on some graph paper.
- Connect some of the intersections to create a shape for your tile design.
- Transcribe these dot coordinates from the graph paper into Program 6 Dot-to-dot tile maker. Click on `Play` to rerun the program.
- Choose the colour of your tile using the `fill()` command. You might find it useful to use the RGB colour selector which you used in Principle 4: Colour.

If the visual output is not what you expect, check that you are inserting both the *x*- and *y*-coordinates for each vertex, for example `vertex(v12.x, v12.y)`. Reread the text above about vertices.

If the program doesn't run, make sure that you haven't written your coordinates in the wrong place. Look for the comments to help guide you to the place to change the code.

4.2 Skill 2: Multiple shapes on a tile

It is possible to draw multiple shapes on a single tile. This is not a necessary requirement for this wallpaper project, but it offers creative possibilities in terms of tile design and might be fun for you to experiment with.

To create multiple shapes, each shape requires its own `beginShape()` and `endShape(Close)` commands. Each shape can be assigned a different colour using the `fill()` command. Shapes can overlap, with the later ones in the code covering earlier ones, as shown in the example below.

Interactive content is not available in this format.

Program 7 Creating multiple shapes on a tile

4.3 Skill 3: Design 4 tiles

Great job on mastering the creation of a single tile! Now, let's make a few additional tiles so that you can create a larger pattern.

In this skill, you are going to create four tiles that you will use for your pattern design. Work iteratively and tweak the colours and/or coordinates for the tiles.

At the end, you will need to save your four tiles in your codesnippet text file. If you have not made that file yet, go back to Principle 1: Tweaking code and watch the video on how to do that.

I created some numerical tile designs using the dot-to-dot system. Take a look at the code for each tile to make sure you understand how the digits are drawn, and then use Programs 8-11 to create four tiles.

Interactive content is not available in this format.

Program 8 Tile number 1

Interactive content is not available in this format.

Program 9 Tile number 2

Interactive content is not available in this format.

Program 10 Tile number 3

Interactive content is not available in this format.

Program 11 Tile number 4

Now work through the following activity to create your own tile designs.

Activity 6 Design and save four different tiles

Allow 40 minutes to complete this activity

Your task is to create and save four separate tile designs.

- Use the four tile editor windows (Programs 8–11) to make your tiles. Create your first design and then copy and paste your tile design into the next window to tweak the colours and coordinates for the next tile. Adding or removing a single vertex can create a very different design.
- Scroll up and down this page to look at your four tiles together. Note that when your tiles are arranged into a pattern later, you won't see the grey background and numbered coordinates.
- When you are happy with your tile designs, copy and paste the codesnippet for each tile into your codesnippet text file and save the file. At the end of this exercise, you need to have the code for all four tiles within your text file. **You**

should only copy the code that defines your tile design, not all the other miscellaneous code that makes the tile-maker program run.

- Check that your codesnippet text file looks a bit like the following figure. You will need your four different tile designs later in the next skills session.

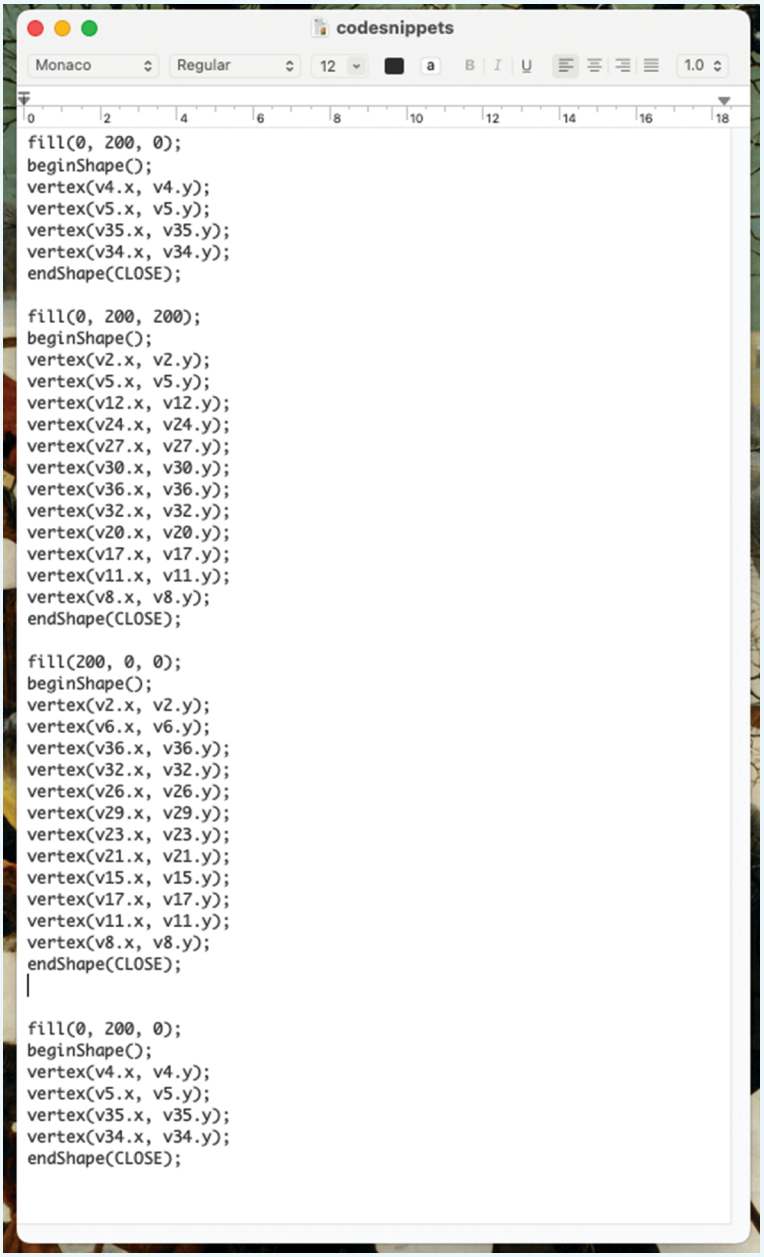


Figure 17 Screenshot of code snippets

5 Making Patterns

In these skills, you will be using the tiles you designed in Skill 3: Design 4 tiles to create, redesign, and test your own algorithmic tile patterns.

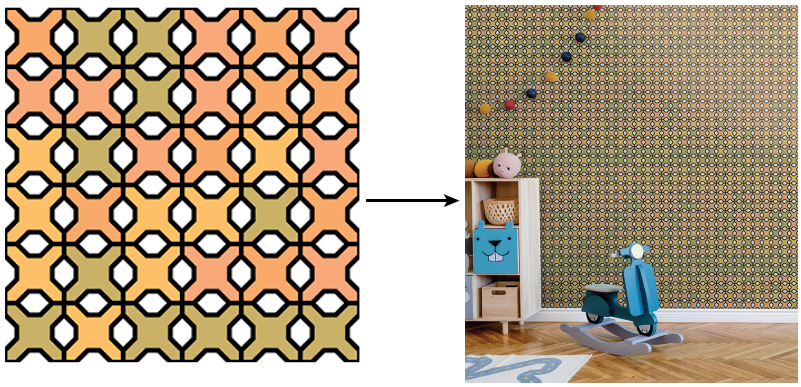


Figure 18 Creating and testing patterns

This introduction will provide you with some historical context about pattern design before you get started on making your own patterns using p5.js.

Technically, patterns are known as ‘tessellations’ – flat surfaces composed of geometric shapes, referred to as tiles, without any overlaps or gaps.

Tessellations can be regular or irregular. Regular means that the surface is made from a single shape that is repeated so that all the edges align. There are three shapes that can form regular tessellations: squares, equilateral triangles, and hexagons. Any one of these can be repeated infinitely to create a surface with no gaps (see diagrams (a), (b), and (c) in Figure 19). Irregular tessellations can be made from almost any kind of geometric shape combined with a different shape or shapes (see diagrams (d), (e), and (f)).

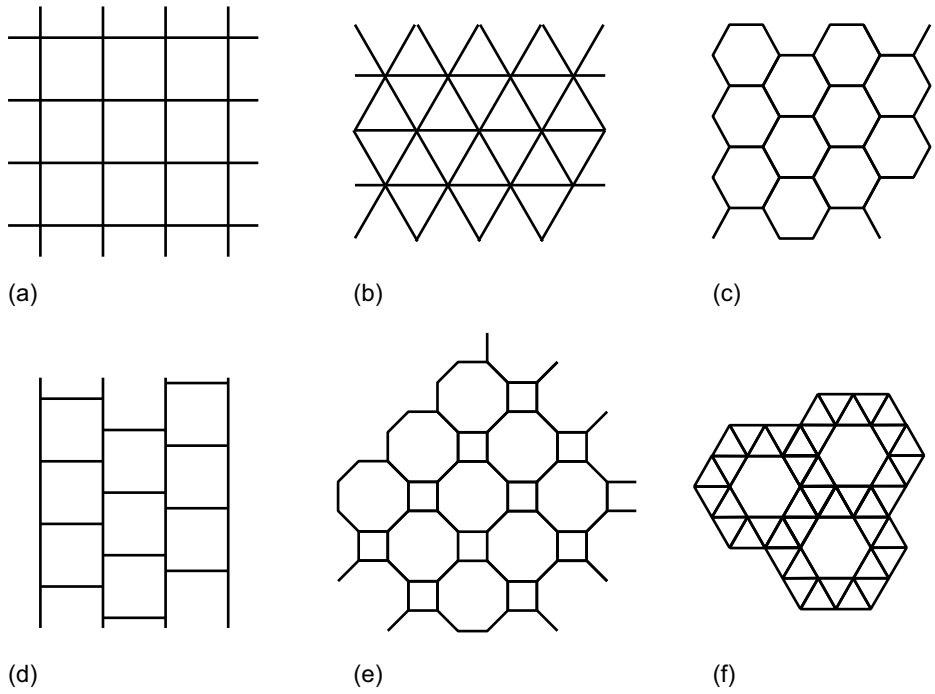


Figure 19 (a)–(c) Regular tessellations; (d)–(f) irregular tessellations

The artist, Maurits Cornelis Escher, is famous for making tessellations with irregular tiles shaped like animals and other natural objects – notice how the horsemen are repeated in interlocking rows in Figure 20. Escher was a Dutch graphic artist (1898–1972) famous for using mathematics for visual effect. He created intricate drawings, lithographs, and woodcuts. Today, Escher's prints are widely used in all forms of decorative design.

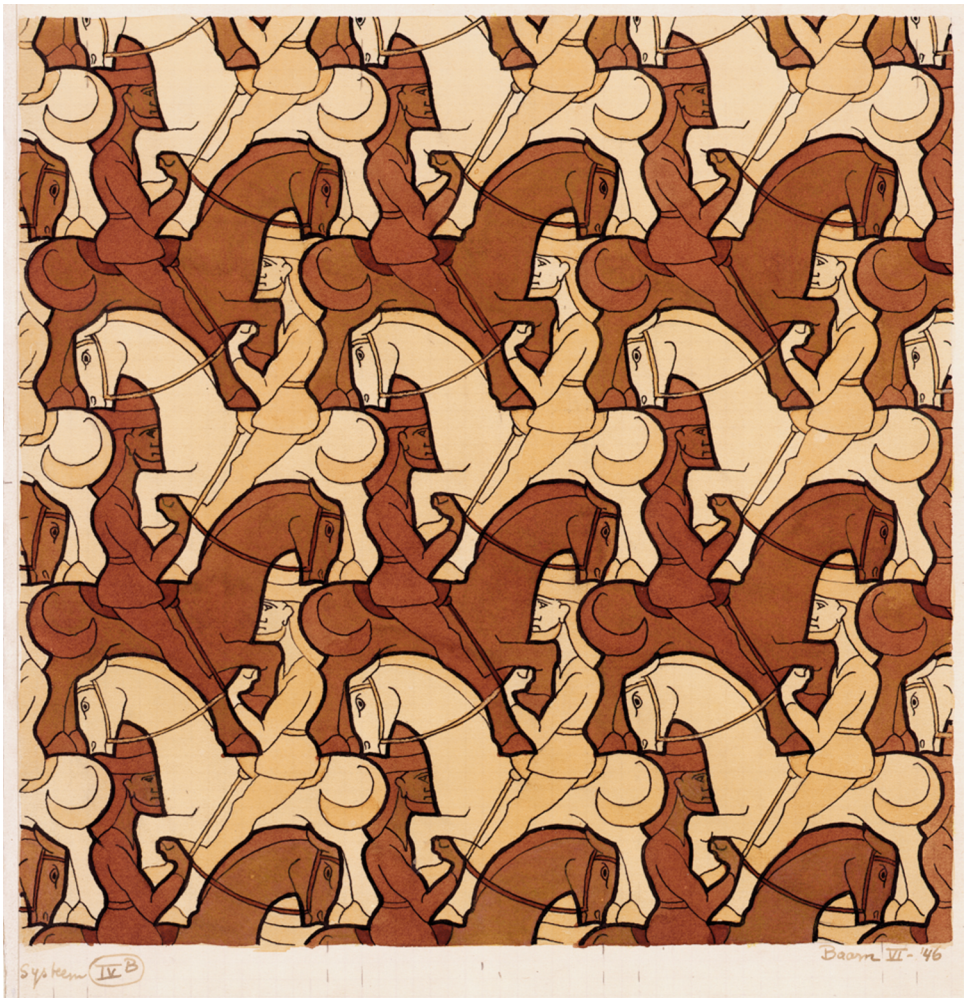


Figure 20 Horsemen drawing by Escher

In these skills, you are only going to focus on regular tessellations of square tiles. Squares are easy to think about and code, and you can make some amazing patterns with them. This introduction draws inspiration from the aesthetics and structures of Truchet tiles, described below, which you will bring into the twenty-first century using p5.js code.

Truchet tiles

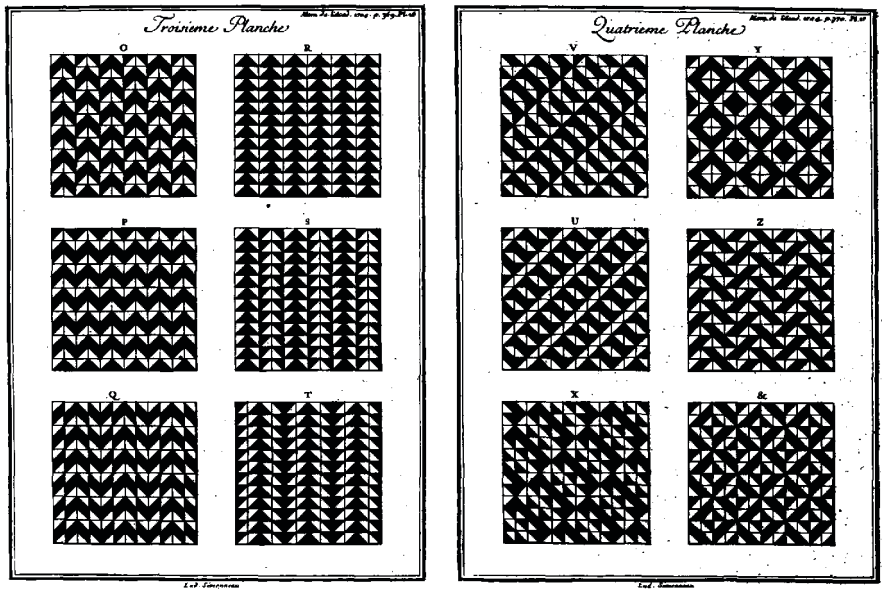


Figure 21

These square tile patterns are named after Sebastien Truchet (1657–1729), a priest from France who was a polymath across art/design and the sciences. He created important inventions in mathematics, engineering, graphic design, and typography. Truchet is known for a book of intricate tile patterns that he published in *Memoire sur les Combinaisons* (1704). These patterns were popularised in 1987 and have become very influential in art and design as well as topology.

Truchet first got the idea for these patterns by looking at ceramic tiles that were piled up and ready to be used:

> During the last trip that I took to the canal d'Orléans by order of His Royal Highness, in a château called Motte St. Lyt 4 leagues this side of Orléans, I found several ceramic tiles that were intended for tiling the floor of a chapel and several other apartments. They were of square shape, divided by a diagonal line into two colored parts. In order to be able to form pleasing designs and patterns by the arrangement of these tiles, I first examined the number of ways in which these tiles could be joined together in pairs, always in checkerboard array.
>
> (Sebastien Truchet, translated by Pauline Boucher, from Smith and Boucher, 1987, p. 373–374)

The four rotations of the Truchet tile can be seen in the figure below.

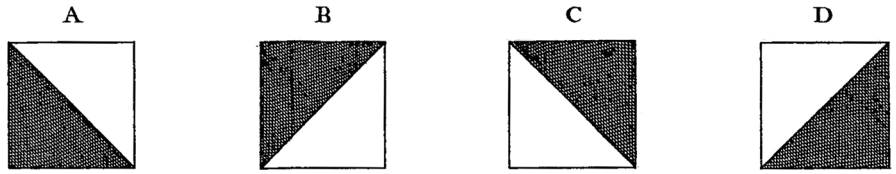


Figure 22

All the Truchet patterns are made from a single square tile with a black-and-white triangle decoration that is rotated in four different orientations. Truchet found that he could create a vast range of visual patterns by arranging these simple tiles in different ways. To record the variations, Truchet created one of the earliest ways of encoding complex patterns, long before the invention of the computer.

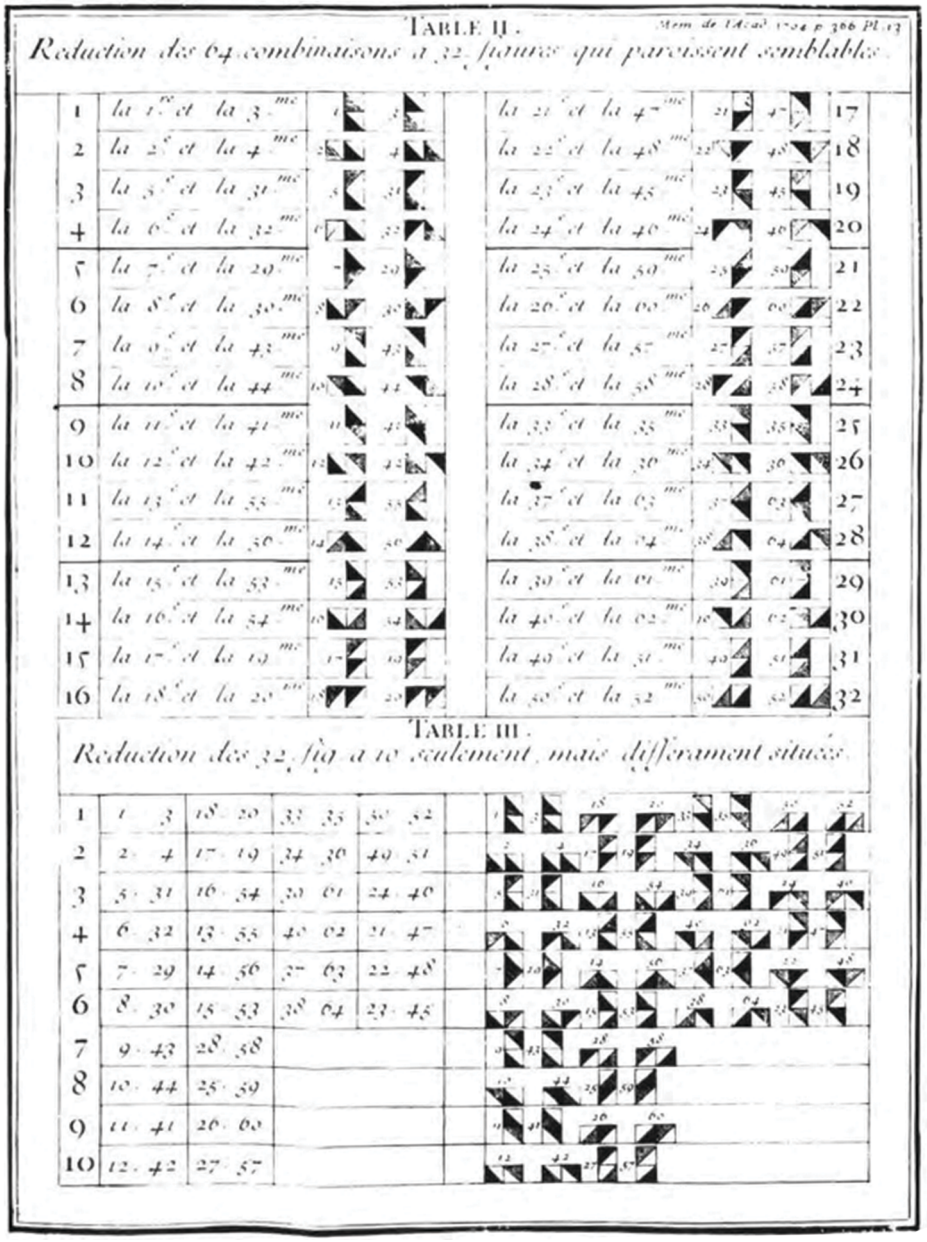


Figure 23

The figure below shows some of Truchet's more complex patterns, made with the same tiles. You will be able to recreate these using p5.js.

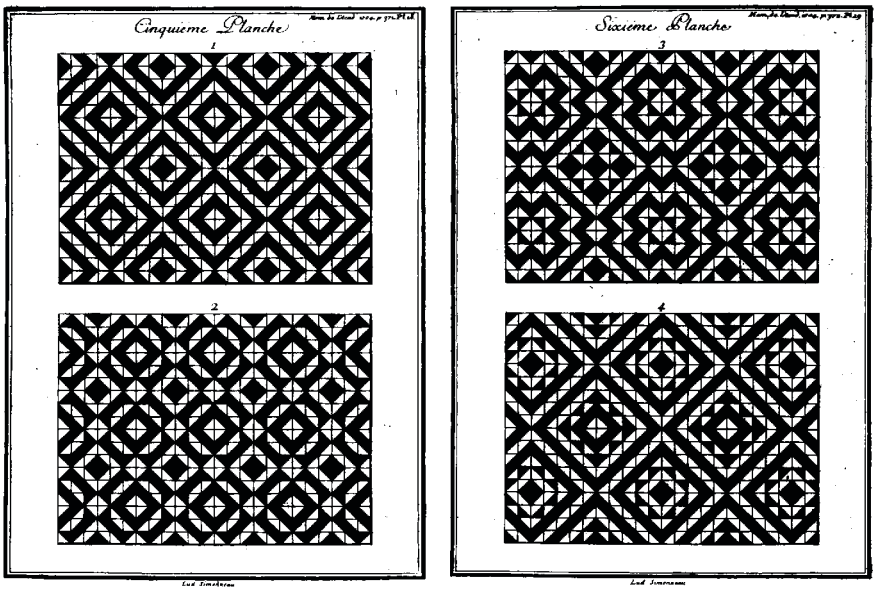


Figure 24

5.1 Skill 4: The pattern maker

In this skill, you will make patterns by controlling how tiles are positioned next to each other and playing with Truchet tiles. To do that, we have provided you with a p5.js pattern-maker program that makes it easier to control the arrangement of four individual tiles to make a repeating pattern.

This will be the longest program you have seen so far, but don't worry, you don't have to understand every single line of code. The goal of the course is to teach you to play with code, not learn programming.

All the parts of the program that you have to tweak are marked with comments. Specifically, there are a number of variables at the top of the code that control how the program works, as well as four separate tile functions. You shouldn't have to scroll much to find all the relevant bits. In each tile function, there are comments that show you where the code for the tile design starts and ends.

Near the bottom of the code is a comment:

```
//— Don't change code below here —
```

Please heed this warning. The setup and draw functions and the coordinate definitions are all located below that comment. There is no need for you to look at this code, understand it, or make any changes.

Since the programs in these skills are quite long, you might prefer to work in the external p5.js editor to be able see more of the code. To do that, copy and paste the example code there.

The pattern-maker program (Program 12) draws four tiles that we refer to as Tile1, Tile2, Tile3, and Tile4. These are the tiles I generated in Programs 8-11 (Skill 3: Design 4 tiles).

Interactive content is not available in this format.

Program 12 Pattern maker

The tiles are arranged in a rosette shape, which is repeated in a sequence until the width and height of the canvas is covered.

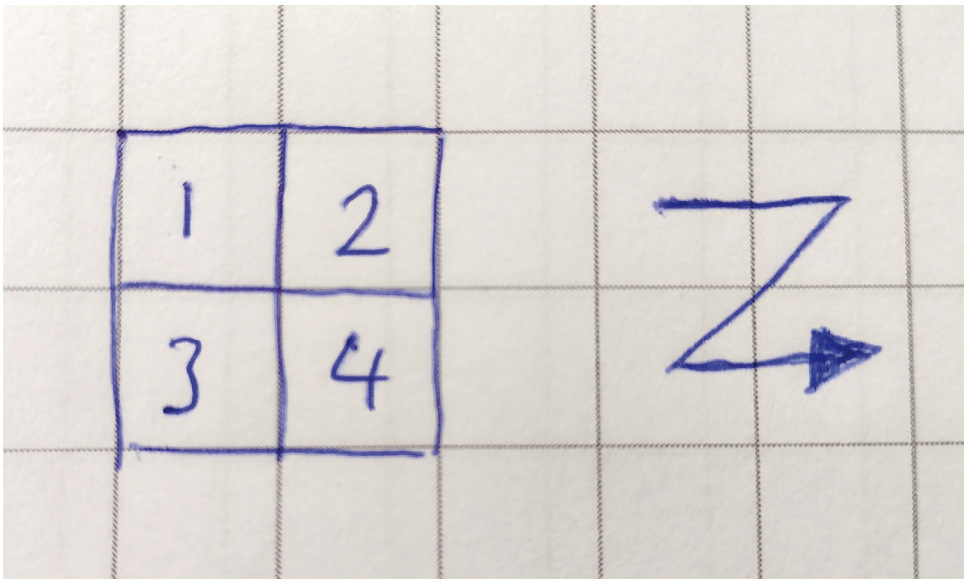


Figure 25 The rosette pattern which controls how the tiles are positioned on the canvas

The canvas is 300 pixels tall and wide, and each tile is 50 pixels square. So, the program keeps drawing these tile rosettes until it gets to the edge of the canvas and then starts again on the next line.

The result is that, with the default settings, the pattern maker will draw 3 × 3 rosettes on the screen to draw 36 individual tiles.

Try changing the value of the variable `tileSize` in Program 12 Pattern maker to see what happens.

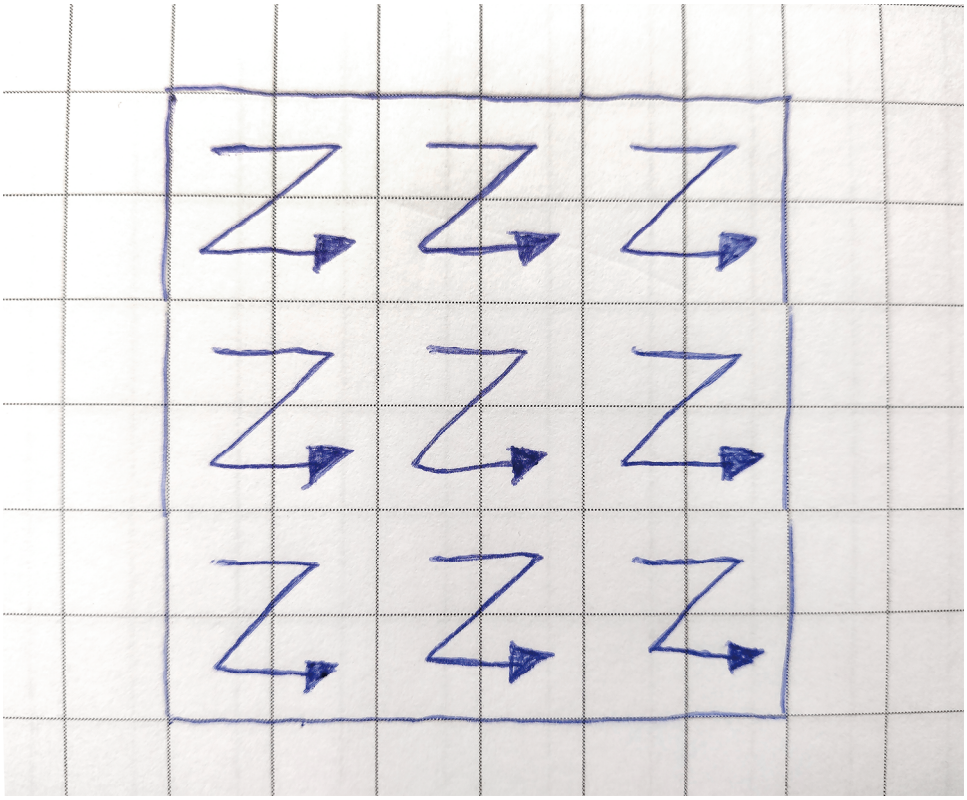


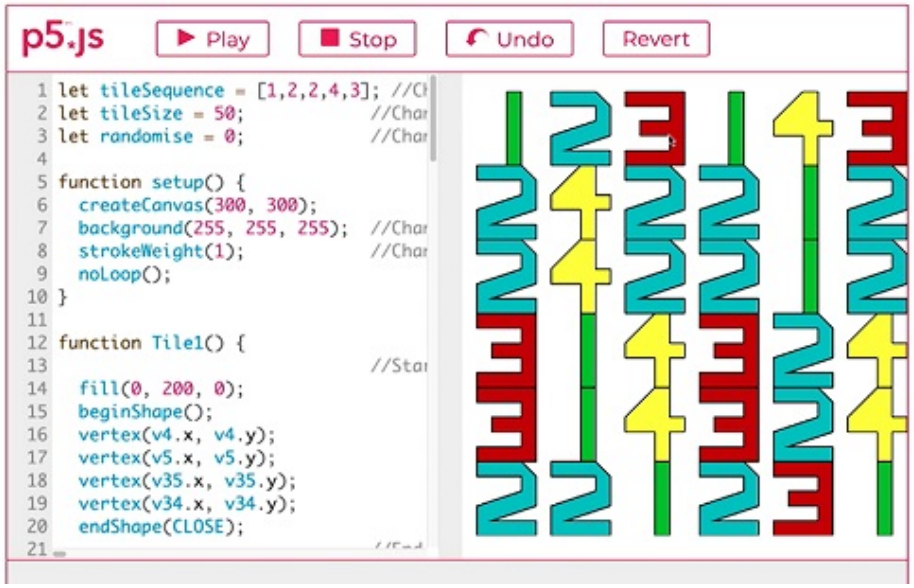
Figure 26 How the program fills the canvas by repeating the tile rosettes

The part of the code that controls the way the tiles are arranged is called an array, which is a special kind of variable that can hold multiple bits of data. It looks like this:

```
tileSequence = [1,2,3,4]
```

You can make your own creative pattern by changing the sequence of the tiles in this array. Video 7 explains how to do this.

Video content is not available in this format.
Video 7 Changing the sequence of tiles



Activity 7 Change the tile sequence

Allow 15-20 minutes to complete this activity

Use Program 12 Pattern maker to change the sequence in which the tiles repeat themselves.

- Try changing the order of the tiles in the `tileSequence` array, for example 1,3,2,4.
- Try repeating some of the tiles in the sequence, for example 1,2,2,4.
- Try adding more than four tiles in to the sequence, for example 1,2,2,3,4.

Notice how the tiles sometimes shift and the overall pattern changes in surprising ways as the tiles create diagonal repetitions.

What is happening here? By increasing or decreasing the number of tiles in the sequence, you are shifting the sequence of the tiles to stretch across multiple rosettes. The effect is that not every rosette is the same anymore.

By adding or removing a single number in the `tileSequence`, you are creating an entirely different design and your own aesthetic judgement about these patterns becomes important.

If you add eight tiles into the `tileSequence` array, then you are defining two rosettes and will end up with something more controlled and regular. If you want to create something more unpredictable, then use more or fewer tiles to create shifting patterns that extend across multiple rosettes.

In the next activity, you will recreate the eighteenth-century tiles and patterns of Sebastien Truchet. The tiles are drawn using the dot-to-dot notation system, with three vertices to draw a series of black triangles. The four versions of the Truchet tile are numbered Tile1–Tile4 in Program 13.

Interactive content is not available in this format.

Program 13 Truchet tiles

Activity 8 Recreate the Truchet patterns

Allow 15-20 minutes to complete this activity

Here are the four rotations of the Truchet tiles, numbered 1–4.

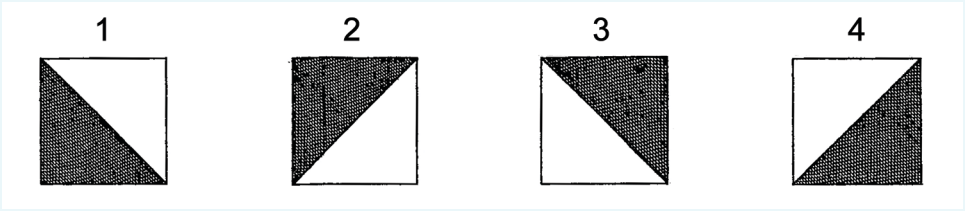


Figure 27

Use Program 13 Truchet tiles to recreate the three Truchet patterns below by changing the numbers in the `tileSequence` array. Note, that the first pattern requires just a single tile. The second pattern requires four tiles. The third pattern requires eight tiles to make two rosettes.

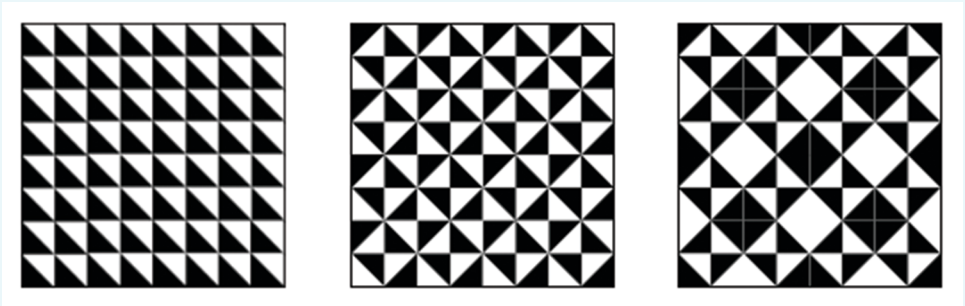


Figure 28

If you find this tricky, don't worry! Take a look at the discussion below and see if the solutions help you to understand how the patterns are made.

If you find this easy, recreate some of the other Truchet patterns illustrated in the Introduction, or create your own by playing with the code.

Discussion

Here are some solutions to recreate the Truchet tiles:

- First pattern: `tileSequence = [1]`
- Second pattern: `tileSequence = [2,1,3,4]`
- Third pattern: `tileSequence = [2,1,3,4,4,3,1,2]`

5.2 Skill 5: Create your pattern

In this skill, you will create your own unique pattern using the tiles that you previously designed and saved in your codesnippet text file.

If you skipped ahead and don't have those tiles, go back to Skill 3: Design 4 tiles to create the four tiles.

If you already tried to make the four tiles and found it difficult, don't worry. You can work through this skill by tweaking the tiles provided in Program 14.

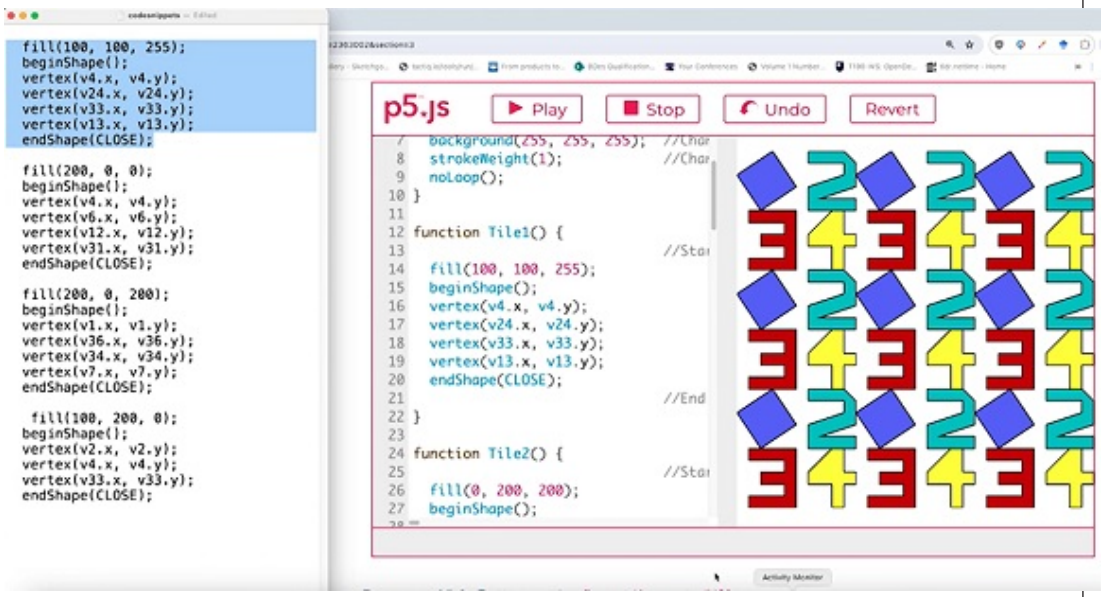
Interactive content is not available in this format.

Program 14 Pattern maker

Video 9 explains how to insert your codesnippets text into the pattern maker.

Video content is not available in this format.

Video 9 Adding codesnippets to the pattern maker



Now try making a pattern using your tiles.

Activity 9 Insert your tiles to make a pattern

Allow 40 minutes to complete this activity

You are going to use the four tiles that you generated in Skill 3: Design 4 tiles to make a pattern using Program 14 Pattern maker. You will replace the existing code with the code that you saved into your codesnippet text file.

To generate your own pattern:

1. Copy the code for your first tile from your codesnippet text file and paste it into the function `Tile1()`, replacing the code that is there.
2. Repeat this action for your remaining tiles, pasting the respective code into `Tile2()`, `Tile3()`, and `Tile4()`.

   Your code will look similar to what was there before, but it may not be as neatly aligned. Don't worry, that won't affect anything as long as you have pasted the code within the curly brackets { } of the tile functions.
3. Press `Play` and you should see your own tiles arranged in a pattern. Hurray!

   If you encounter a problem, simply press `Revert` and start again. The most likely cause is that you have accidentally pasted your code over some of the function code.
4. Play around with the arrangement of your tiles by tweaking the `tileSequence` array.

5.3 Skill 6: Redesign and test your pattern

Take a moment to reflect. Do you like the pattern you created in Skill 5? If you're anything like me, those tiles you put together didn't quite make sense as a pattern. When I first gave it a go, I didn't think about how the tiles would fit together or make an overall pattern.

In this skill, you will learn to design patterns in a modular way. So, you will redesign your pattern to improve it, try variations, and test it to think about your pattern in terms of its design qualities. Design judgement is a key skill to develop within algorithmic design.

Activity 10 Redesign your pattern

Allow 40 minutes to complete this activity

In the following figure, I used graph paper to draw four tiles, which I labelled Tile 1, 2, 3, and 4. These four tiles form a rosette shape. I also marked some of the coordinates to make it easier to transcribe all the vertices into my code.

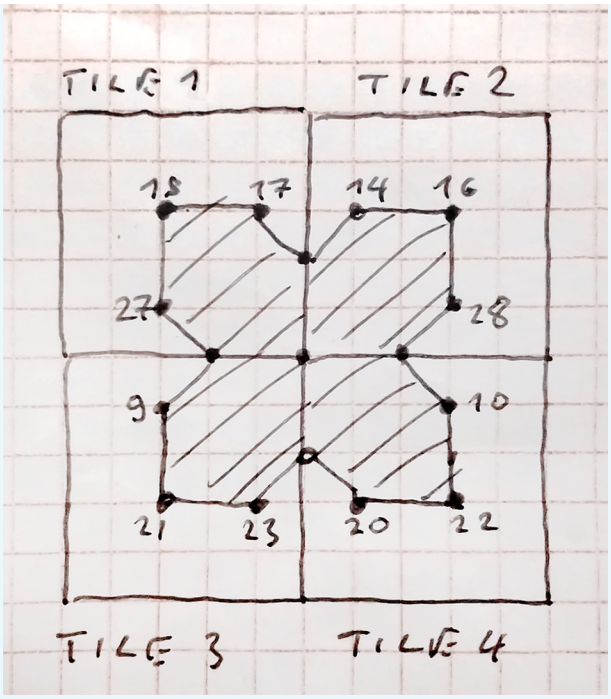


Figure 29

Your aim in this activity is to redesign and improve your existing pattern.

- Use a piece of graph paper and draw four blank tiles in a rosette shape, as in the figure above.
- Pick one of your existing tiles that you like best and draw it into one of the four spaces of the rosette. Now create three new tiles that complement it within the rosette pattern.
- Think about visually reflecting, rotating, or inverting your existing tile. Often creating some kind of symmetrical repetition of a single shape can work well as a pattern design.
- Consider how the tiles join up when placed next to each other. Think about the rosettes that will be positioned above and below the current one you are designing. What shapes will they form when they fit together?
- Use Program 15 Redesign your pattern below to insert your pattern design. When you are adding in and numbering your vertices' dots, remember that the four tiles in the rosette are separate and each have coordinates that run from 1–36.
- Consider changing the fill and stroke colours of your tiles. By selecting related complementary colours for the tiles, you can create more cohesive designs. Remember the visual effects that the Truchet patterns created using just black-and-white shapes.

Interactive content is not available in this format.

Program 15 Redesign your pattern

A useful way to rethink your pattern design and come up with some unexpected visual alternatives is to randomise it and use serendipity as a creative tool.

Activity 11 Try variations of your pattern

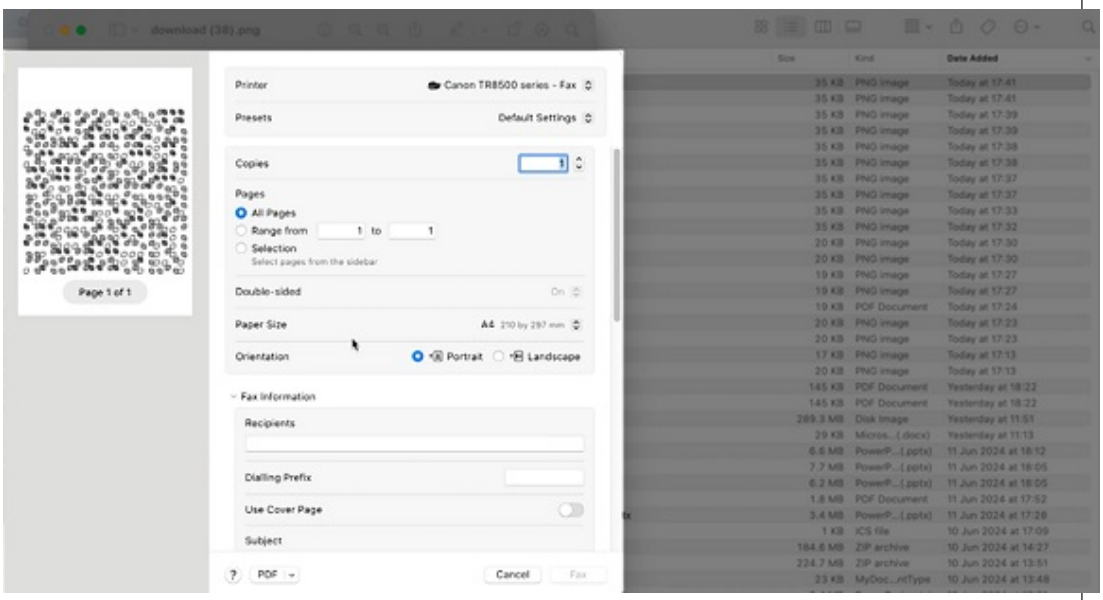
Allow 10-15 minutes to complete this activity

Use Program 15 Redesign your pattern to try variations of your pattern.

- Randomise the order of tiles by changing the variable `randomise` in line 3. You can change its value from 0 to 1. When you press `Play`, this will randomise the order of the tiles in your `tileSequence` array within each rosette.
- Keep pressing `Play` to generate new versions. Seeing some of these versions of your pattern might give you some ideas for how to improve your design.
- Save images of some of the random variations of your tile designs by right-clicking and saving them as PNG files to your disk or by pressing the `S` key.

An important part of the design process is testing your design in various ways and contexts. The next video demonstrates how to see the pattern at different scales and how to print it onto paper.

Video content is not available in this format.
Video 10 Saving and printing a pattern



The next activity suggests a variety of ways for you to test your pattern design.

Activity 12 Test your pattern design

Allow 40 minutes to complete this activity

You don’t have to do all these tests, but I recommend that you do at least two:

- **Test 1**: try out how your pattern looks at different scales by changing the `tileSize` variable. The default is `tileSize = 50`, but you can try changing it to 10, 25, or 100 to see how your pattern looks at different scales. If you choose odd numbers, then the number of tiles per row will change and your pattern will look different.
- **Test 2**: save your pattern design by right-clicking and saving it as a PNG file to your disk or by pressing the `S` key. Now you can upload it to this external wallpaper manufacturer’s website to see what your pattern will look like as wallpaper.
- **Test 4**: print the PNG image of your pattern onto A4 paper. Follow the instructions in Video 10 above, and make sure you select `Print Entire Image`, and keep the image proportions the same so that your design does not get distorted. Hang your design on the wall and admire it.

6 Identify a place for your wallpaper design

Now that you know how to technically create an algorithmic pattern, it's time to design a unique algorithmic wallpaper for your own home! I suggest you start by picking a specific place in your home where you will place your wallpaper. Identifying this site will help to inspire your design by giving it some context.

Go around your home and take a slow and deliberate look at all the spaces where a small piece of wallpaper might fit. You are not going to be able to fill a large area with your design, maybe just 50 cm × 50 cm. So, large walls are out of the question. But there might be some interesting nooks and crannies you can use.

Initially, when searching for a location for my wallpaper, I looked at a shelf which is right next to my work desk, where I reach over to grab things like batteries and cables (Figure 30). Once I clear it up a bit, it should be easier to see the back wall a bit more. I am the only person who sees this corner of my studio every day and it feels like a cosy little place to add my own wallpaper. Maybe the design could be something colourful and energetic with sharp, angular edges to help me work faster. This was one possible location I found for my wallpaper design, along with my family's kitchen cabinet, and my son's bedroom.

Figure 30 A potential place for this author's wallpaper

Consider unusual places, such as drawers and cupboards. Small spaces will be easier to cover with wallpaper and your design might have more impact there. If you choose a self-contained space, then you might be able to visually 'take it over' with just a small piece of wallpaper. Ask yourself these questions:

- How does this space make me and others feel?
- What kind of an atmosphere would wallpaper create in this space?
- What colours, shapes and patterns does this space make me think of?

To turn algorithmic pattern into wallpaper you will have to print it out onto at least two A4 sheets so that you can attach it temporarily to a surface in your home with masking tape or removable adhesive.

Activity 13 Pick a location

Spend no more than 20 minutes on this activity

If you found a few locations, make a choice and focus on just a single one. This will be the context for your wallpaper design.

Write down a couple of sentences about why you chose this place and the atmosphere you want to create with your wallpaper design.

7 Look for inspiration

Now that you know the kinds of tiles and patterns you can design, it’s time to look for some inspiration from other historical tile and pattern designs.

A good place to look is the Victoria and Albert Museum (V&A), which has an online catalogue. You can search their catalogue for tiles or for fabrics. The V&A also has a good guide on creating Islamic tile patterns.