

chapter one

Your First Sketch

This chapter gets one shape onto the screen. Along the way, it introduces the two functions that organize a basic p5.js sketch and the order in which shapes are painted.

A browser and a text editor

The browser provides an HTML `<canvas>` element and a two-dimensional drawing API. p5.js sits above that API and supplies concise functions such as `rect`, `circle`, `fill`, and `line`. It also manages the animation loop and some input events. Most examples in this book use p5.js; occasional raw Canvas examples show the browser operations underneath.

The quickest place to start is the p5.js web editor, where code and output share one window. To work locally, put two files in the same folder. The first is `index.html`:

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <script
src="https://cdn.jsdelivr.net/npm/p5@1.9.0"
></script>
  </head>
  <body>
    <script src="sketch.js"></script>
  </body>
</html>
```

The version number in that URL keeps the examples reproducible. The page needs an internet connection to fetch that copy of `p5.js`; you can instead download the library and refer to the local file.

Create an empty `sketch.js` beside the HTML file. When this book says to run a sketch, save that file and reload `index.html`, or press Play in the web editor.

note

HTML Canvas is a standard browser feature. `p5.js` provides a convenient interface to it and adds facilities of its own. The two are related, but they are not interchangeable names for the same API.

The two main functions

A basic sketch in `p5.js` global mode usually defines two functions:

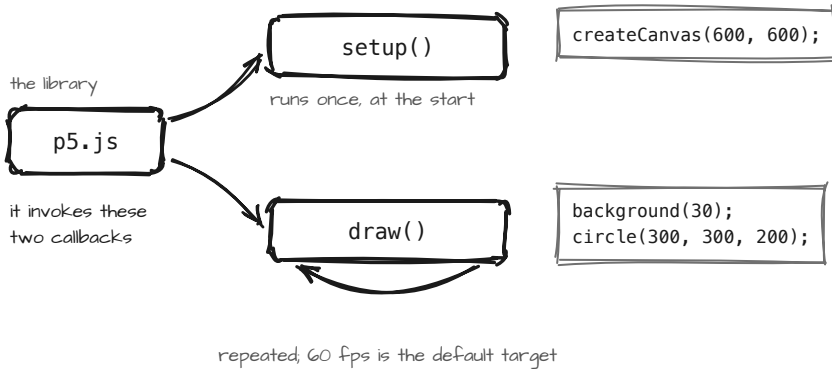
```
function setup() {  
  // p5 calls this once at startup  
}  
  
function draw() {  
  // p5 calls this for each animation frame  
}
```

`setup()` prepares the sketch. **`draw()`** updates a frame.

`p5` calls `setup()` once after it has initialized. It then calls `draw()` for successive animation frames. You define these functions but do not invoke them yourself. This arrangement is a callback: the library decides when your code runs.

The default target is 60 frames per second, but the actual rate depends on the display, browser, device load, and the work done in each frame. A static sketch does not need to redraw forever; it can call `noLoop()`, or do all its

drawing in `setup()`. We will keep `draw()` for now because the same examples will soon begin to move.



In `p5.js` global mode, the library calls `setup()` once and then calls `draw()` for successive frames. Sixty frames per second is a target, not a guarantee.

Sizing the canvas

Start by creating a drawing surface in `setup()`:

```
function setup() {
  createCanvas(600, 600);
}
```

The arguments are width and height. This call creates a canvas with a logical drawing area 600 units wide and 600 units high. In the default layout, those units correspond to CSS pixels. On a high-density display, p5 may use more physical device pixels behind the scenes so the result remains sharp.

The drawing grid is
600 × 600 logical
units.

Until something paints the canvas, it is transparent and the page behind it usually looks white. The `p5.js` editor's starter sketch may include `background(220);` replace the starter code with the examples here instead of combining two copies of `setup()` or `draw()`.

Drawing a circle

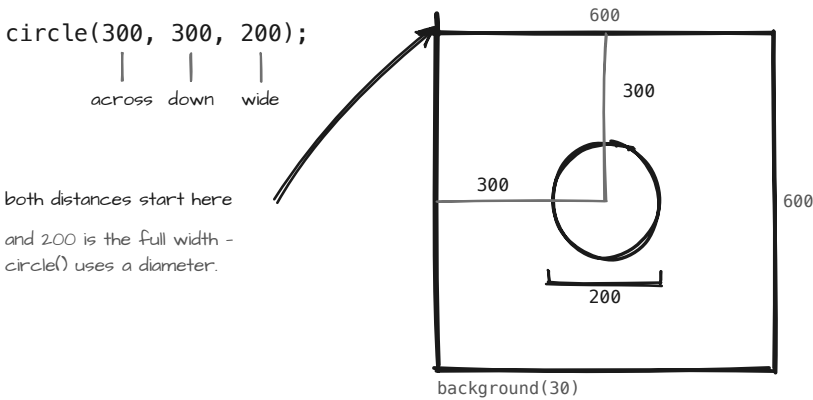
Add `draw()` and two drawing calls:

```
function setup() {  
  createCanvas(600, 600);  
}  
  
function draw() {  
  background(30);  
  circle(300, 300, 200);  
}
```

In p5's default RGB color mode, one argument to `background()` is a grayscale value from 0 (black) to 255 (white). The value 30 produces dark gray. Because `background()` paints the whole canvas on every frame, it also clears the previous frame.

30 is a grayscale value: **0** is black and **255** is white.

The three arguments to `circle()` are the center's x-coordinate, its y-coordinate, and the diameter. The point (300, 300) is the center of this 600-by-600 canvas, and a diameter of 200 extends 100 units in each direction. p5's default style gives the circle a white fill and a thin black outline.



The center is 300 units from the left edge and 300 from the top. The final argument is the 200-unit diameter, not the radius. These coordinates mark the center only because the canvas is 600 by 600.

Try a diameter of **500**, then an x-coordinate of **150**.

Change **200** to **500** and run the sketch again. Then restore it and change the first **300** to **150**. Altering one argument at a time makes each argument's effect easy to identify.

Why increasing y moves down

Canvas coordinates begin at the top-left corner. x increases to the right, and y increases downward. Thus (300, 300) lies halfway across and halfway down this canvas, while (300, 450) lies below it.

On the canvas, a larger y-coordinate is lower.

This differs from the Cartesian graphs used in much of school mathematics, where positive y points upward. Chapter 2 maps the canvas coordinate system in detail. For now, remember that a larger y-coordinate moves a point down.

What p5 is doing for you

The browser's Canvas API can draw the same filled shape directly. First, an HTML page needs its own canvas

element:

```
<canvas id="c" width="600" height="600">
</canvas>
```

JavaScript then obtains the two-dimensional rendering context:

```
const canvas = document.getElementById('c');
const ctx = canvas.getContext('2d');
```

The context stores drawing state and exposes the drawing methods. Here is a small animation loop using it without p5.js:

```
function draw() {
  ctx.fillStyle = 'rgb(30 30 30)';
  ctx.fillRect(0, 0, 600, 600);

  ctx.beginPath();
  ctx.arc(300, 300, 100, 0, Math.PI * 2);
  ctx.fillStyle = 'white';
  ctx.fill();

  requestAnimationFrame(draw);
}
requestAnimationFrame(draw);
```

arc() takes a radius of 100;
circle() took a diameter of 200.

p5.js - one line

```
circle(300, 300, 200);
```



becomes

the same fill, with no library

```
ctx.beginPath();
ctx.arc(300, 300, 100, 0, Math.PI * 2);
ctx.fillStyle = 'white';
ctx.fill();
```

four steps,
in this order

the 100 is a radius - half of the 200 p5 took.
0 to $\text{Math.PI} * 2$ is one whole turn of the arc.

For a filled circle, p5's `circle()` call expands into several Canvas operations. The raw API takes a radius and angles in radians.

`fillRect()` paints the background. `beginPath()` starts a new path, and `arc()` adds a full circle: its radius is 100 and its angles run from 0 through 2π radians. `fill()` paints the path. This compact version omits the thin black outline that p5 applies by default; matching it would also require `strokeStyle` and `stroke()`.

`requestAnimationFrame()` asks the browser to call `draw()` before a future repaint. The browser controls the timing and may pause or reduce callbacks for a background tab. p5 builds its frame loop on this browser facility and adds its own frame-rate handling.

The comparison explains why p5 is useful: it chooses common defaults and packages several Canvas operations into short drawing calls. The underlying Canvas API remains available through `drawingContext` when a sketch needs it.

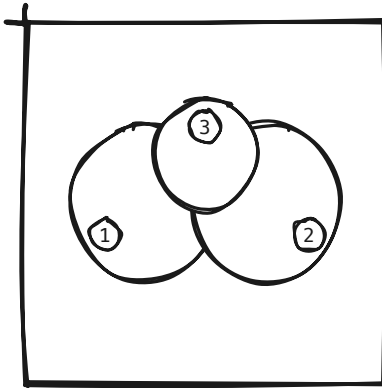
A picture made from three circles

Replace the first sketch with this one:

```
function setup() {
  createCanvas(600, 600);
}

function draw() {
  background(30);
  circle(200, 300, 260);
  circle(400, 300, 260);
  circle(300, 220, 180);
}
```

Nine arguments
make three
overlapping circles.



background(30)

- ① circle(200, 300, 260);
the left body
- ② circle(400, 300, 260);
the right body
- ③ circle(300, 220, 180);
the head - drawn last, so on top

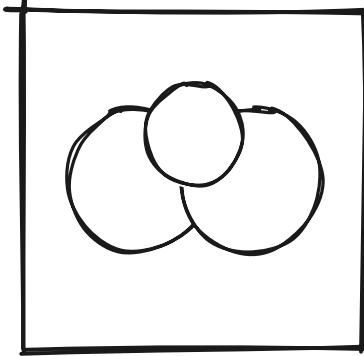
The two 260-unit circles share a y-coordinate. The smaller circle is higher because its y-coordinate is 220 rather than 300. It is drawn last, so it covers the other two where they overlap.

Canvas uses immediate-mode drawing: each call paints into the current bitmap. Later shapes cover earlier pixels when their filled areas overlap. In this example, the small circle appears in front because its call comes last. `background()` belongs first because it would cover every shape drawn before it.

Same shapes,
different order: the
overlap changes.

as written

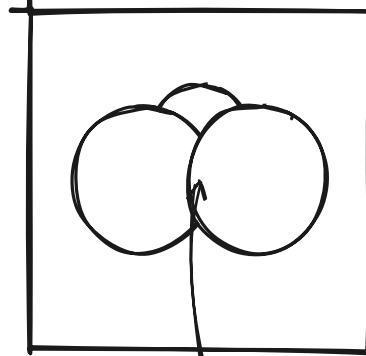
head drawn last - 1 2 3



background(30)

Exercise II

head drawn first - 3 1 2



buried where they overlap

Exercise 1.1 changes only the call order. When the small circle is drawn first, the two larger circles cover its lower portion.

Chapter 2 makes the coordinate grid explicit so that changes to position can be deliberate.

Exercises

Exercise 1.1. Move the small circle's call before the two large circles. Predict which portions will remain visible, then run the sketch and compare.

Exercise 1.2. Starting from the centered circle, move it into the bottom-right region and reduce its diameter to about 60. Remember that a larger second argument moves it downward.

Exercise 1.3. Draw three equal circles in a vertical column through the center of the canvas. Space them evenly by changing only their y-coordinates.

Exercise 1.4. Before running the code, predict the results of `background(255)` and `background(0)`. Try both and explain why the white fill has little contrast against one of them.