A particle system

So far, we've managed to create a single particle that we re-spawn whenever it dies. Now, we want to create a continuous stream of particles, adding a new one with each cycle through `draw()`. We could just create an array and push a new particle onto it each time:

```
var particles = [];

draw = function() {
  background(133, 173, 242);
  particles.push(new Particle(new PVector(width/2, 50)));


  for (var i = 0; i < particles.length; i++) {
    var p = particles[i];
    p.run();
  }
};
```

If you try that out and run that code for a few minutes, you'll probably start to see the frame rate slow down further and further until the program grinds to a halt. That's because we're creating more and more particles that we have to process and display, without ever removing any. Once the particles are dead, they're useless, so we may as well save our program from unnecessary work and remove those particles.

To remove items from an array in JavaScript, we can use the `splice()` method, specifying the desired index to delete and number to delete (just one). We'd do that after querying whether the particle is in fact dead:

```
var particles = [];

draw = function() {
```

```
  background(133, 173, 242);
  particles.push(new Particle(new PVector(width/2, 50)));

  for (var i = 0; i < particles.length; i++) {
    var p = particles[i];
    p.run();
    if (p.isDead()) {
      particles.splice(i, 1);
    }
  }
};
```

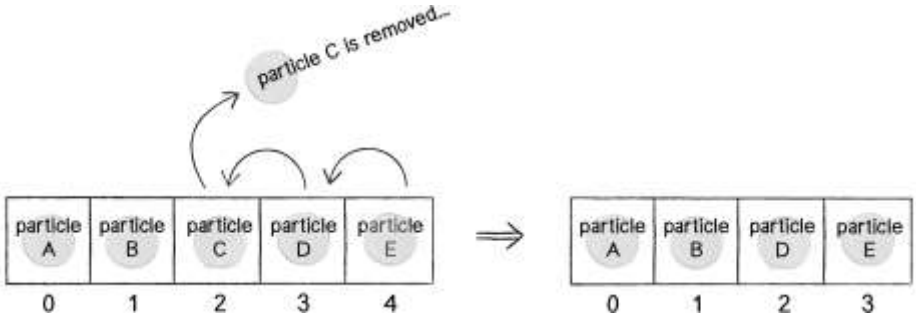
Although the above code will run just fine (and the program will never grind to a halt), we have opened up a medium-sized can of worms. Whenever we manipulate the contents of an array while iterating through that very array, we can get ourselves into trouble. Take, for example, the following code:

```
for (var i = 0; i < particles.length; i++) {
  var p = particles[i];
  p.run();
  particles.push(new Particle(new PVector(width/2, 50)));
}
```

This is a somewhat extreme example (with flawed logic), but it proves the point. In the above case, for each particle in the array, we add a new particle to the array (thus changing the `length` of the array). This will result in an infinite loop, as `i` can never increment past `particles.length`.

While removing items from the particles array during a loop doesn't cause the program to crash (as it does with adding), the problem is almost more insidious in that it leaves no evidence. To discover the problem we must

first establish an important fact. When an item is removed from an array, all items are shifted one spot to the left. Note the diagram below where particle C (index 2) is removed. Particles A and B keep the same index, while particles D and E shift from 3 and 4 to 2 and 3, respectively.



Let's pretend we are `i` looping through the array.

- when i = 0 → Check particle A → Do not delete
- when i = 1 → Check particle B → Do not delete
- when i = 2 → Check particle C → Delete!
- (Slide particles D and E back from slots 3 and 4 to 2 and 3)
- when i = 3 → Check particle E → Do not delete

Notice the problem? We never checked particle D! When C was deleted from slot #2, D moved into slot #2, but `i` has already moved on to slot #3. This is not a disaster, since particle D will get checked the next time around. Still, the expectation is that we are writing code to iterate through every single item of the array. Skipping an item is unacceptable.

There's a simple solution to this problem: just iterate through the array backwards. If you are sliding items from right to left as items are removed, it's impossible to skip an item by accident. All we have to do is modify the three bits in the for loop:

```
for (var i = particles.length-1; i >= 0; i--) {
  var p = particles[i];
```

```
    p.run();
    if (p.isDead()) {
      particles.splice(i, 1);
    }
  }
```

Putting it all together, we have this:

```
// A single Particle object
var Particle = function(position) {
  this.acceleration = new PVector(0, 0.05);
  this.velocity = new PVector(random(-1, 1), random(-1, 0));
  this.position = position.get();
  this.timeToLive = 255.0;
};

Particle.prototype.run = function() {
  this.update();
  this.display();
};

Particle.prototype.update = function(){
  this.velocity.add(this.acceleration);
  this.position.add(this.velocity);
  this.timeToLive -= 2;
};

Particle.prototype.display = function() {
  stroke(255, 255, 255, this.timeToLive);
  strokeWeight(2);
```

```
  fill(210, 210, 255, this.timeToLive);
  ellipse(this.position.x, this.position.y, 12, 12);
};

Particle.prototype.isDead = function() {
  if (this.timeToLive < 0) {
     return true;
  } else {
     return false;
  }
};

var particles = [];

draw = function() {
  background(133, 173, 242);
  particles.push(new Particle(new PVector(width/2, 50)));

  for (var i = particles.length-1; i >= 0; i--) {
    var p = particles[i];
    p.run();
    if (p.isDead()) {
      particles.splice(i, 1);
    }
  }
};
```

OK. Now we've done two things. We've written an object to describe an individual `Particle`. We've figured out how to use arrays to manage many `Particle` objects (with the ability to add and delete at will).

We could stop here. However, one additional step we can and should take is to create an object to describe the collection of Particle objects itself—the `ParticleSystem` object. This will allow us to remove the bulky logic of looping through all particles from the main tab, as well as open up the possibility of having more than one particle system.

If you recall the goal we set at the beginning of this chapter, we wanted our program to look like this:

```
var ps = new ParticleSystem(new PVector(width/2, 50));


draw = function() {
  background(0, 0, 0);
  ps.run();
};
```

Let's take the program we wrote above and see how to fit it into the `ParticleSystem` object.

Here's what we had before - note the bolded lines:

```
var particles = [];


draw = function() {
  background(133, 173, 242);
  particles.push(new Particle(new PVector(width/2, 50)));


  for (var i = particles.length-1; i >= 0; i--) {
    var p = particles[i];
```

```
    p.run();
    if (p.isDead()) {
      particles.splice(i, 1);
    }
  }
};
```

Here's how we can rewrite that into an object - we'll make the `particles` array a property of the object, make a wrapper method `addParticle` for adding new particles, and put all the particle running logic in `run`:

```
var ParticleSystem = function() {
  this.particles = [];
};

ParticleSystem.prototype.addParticle = function() {
  this.particles.push(new Particle());
};

ParticleSystem.prototype.run = function() {
  for (var i = this.particles.length-1; i >= 0; i--) {
      var p = this.particles[i];
      p.run();
      if (p.isDead()) {
        this.particles.splice(i, 1);
      }
    }
};
```

We could also add some new features to the particle system itself. For example, it might be useful for the `ParticleSystem` object to keep track of an origin point where particles are made. This fits in with the idea of a

particle system being an “emitter,” a place where particles are born and sent out into the world. The origin point should be initialized in the constructor.

```
var ParticleSystem = function(position) {
  this.origin = position.get();
  this.particles = [];
};

ParticleSystem.prototype.addParticle = function() {
  this.particles.push(new Particle(this.origin));
};
```

Here it is, all together now:

```
// A single Particle object
var Particle = function(position) {
 this.acceleration = new PVector(0, 0.05);
 this.velocity = new PVector(random(-1, 1), random(-1, 0));
 this.position = position.get();
 this.timeToLive = 255.0;
};

Particle.prototype.run = function() {
 this.update();
 this.display();
};

Particle.prototype.update = function(){
 this.velocity.add(this.acceleration);
```

```
  this.position.add(this.velocity);
  this.timeToLive -= 2;
};

Particle.prototype.display = function() {
  stroke(255, 255, 255, this.timeToLive);
  strokeWeight(2);
  fill(210, 210, 255, this.timeToLive);
  ellipse(this.position.x, this.position.y, 12, 12);
};

Particle.prototype.isDead = function() {
  if (this.timeToLive < 0) {
     return true;
  } else {
     return false;
  }
};

var ParticleSystem = function(position) {
   this.origin = position.get();
   this.particles = [];
};

ParticleSystem.prototype.addParticle = function() {
   this.particles.push(new Particle(this.origin));
};
```

```
ParticleSystem.prototype.run = function() {
  for (var i = this.particles.length-1; i >= 0; i--) {
     var p = this.particles[i];
     p.run();
     if (p.isDead()) {
       this.particles.splice(i, 1);
     }
   }
};

var ps = new ParticleSystem(new PVector(width/2, 50));

var draw = function() {
  background(204, 90, 204);
  ps.run();
  ps.addParticle();
};
```