# Particle types

Now we're going to use more advanced object-oriented programming techniques like inheritance, so you may want to review "Inheritance" in the Intro to JS course and come back. Don't worry, we'll wait!

Feeling good about how inheritance works? Good, because we're going to use inheritance to make different types of `Particle` sub-objects, which share much of the same functionality but also differ in key ways.

Let's review a simplified `Particle` implementation:

```
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();
};

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

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

Particle.prototype.display = function() {
  fill(127, 127, 127);
  ellipse(this.position.x, this.position.y, 12, 12);
};
```

Next, we create a new object type based on `Particle`, which we'll call `Confetti`. We'll start off with a constructor function that accepts the same number as arguments, and simply calls the `Particle` constructor, passing them along:

```
var Confetti = function(position) {

  Particle.call(this, position);

};
```

Now, in order to make sure that our `Confetti` objects share the same methods as `Particle` objects, we need to specify that their prototype should be based on the `Particle` prototype:

```
Confetti.prototype = Object.create(Particle.prototype);

Confetti.prototype.constructor = Confetti;
```

At this point, we have `Confetti` objects that act exactly the same way as `Particle` objects. The point of inheritance isn't to make duplicates, it's to make new objects that share a lot of functionality but also differ in some way. So, how is a `Confetti` object different? Well, just based on the name, it seems like it should look different. Our `Particle` objects are ellipses, but confetti is usually little bits of square paper, so at the very least, we should change the `display` method to show them as rectangles instead:

```
Confetti.prototype.display = function(){

  rectMode(CENTER);

  fill(0, 0, 255, this.timeToLive);

  stroke(0, 0, 0, this.timeToLive);

  strokeWeight(2);

  rect(0, 0, 12, 12);

};
```

Here's a program with one `Particle` object instance and one `Confetti` object instance. Notice they behave similarly but differ in their appearance:

```
/* This program contains 2 objects:

  - Particle

  -- Confetti (sub-object of Particle)


  At the bottom, it creates a Particle and Confetti and animates them.

*/


/* The 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();

};


Particle.prototype.run = function() {

  this.update();

  this.display();

};


Particle.prototype.update = function(){

  this.velocity.add(this.acceleration);

  this.position.add(this.velocity);

};

Particle.prototype.display = function() {

  stroke(0, 0, 0);
  strokeWeight(2);

  fill(255, 0, 0);

  ellipse(this.position.x, this.position.y, 12, 12);

};

/* The Confetti object */

var Confetti = function(position) {

  Particle.call(this, position);

};

Confetti.prototype = Object.create(Particle.prototype);

Confetti.prototype.constructor = Confetti;


Confetti.prototype.display = function(){

  rectMode(CENTER);

  fill(0, 0, 255);

  stroke(0, 0, 0);

  strokeWeight(2);

  rect(this.position.x, this.position.y, 12, 12);

};


var particle = new Particle(new PVector(width/2, 50));

var confetti = new Confetti(new PVector(width/2, 50));

draw = function() {

  background(168, 255, 156);

  particle.run();

  confetti.run();

};
```