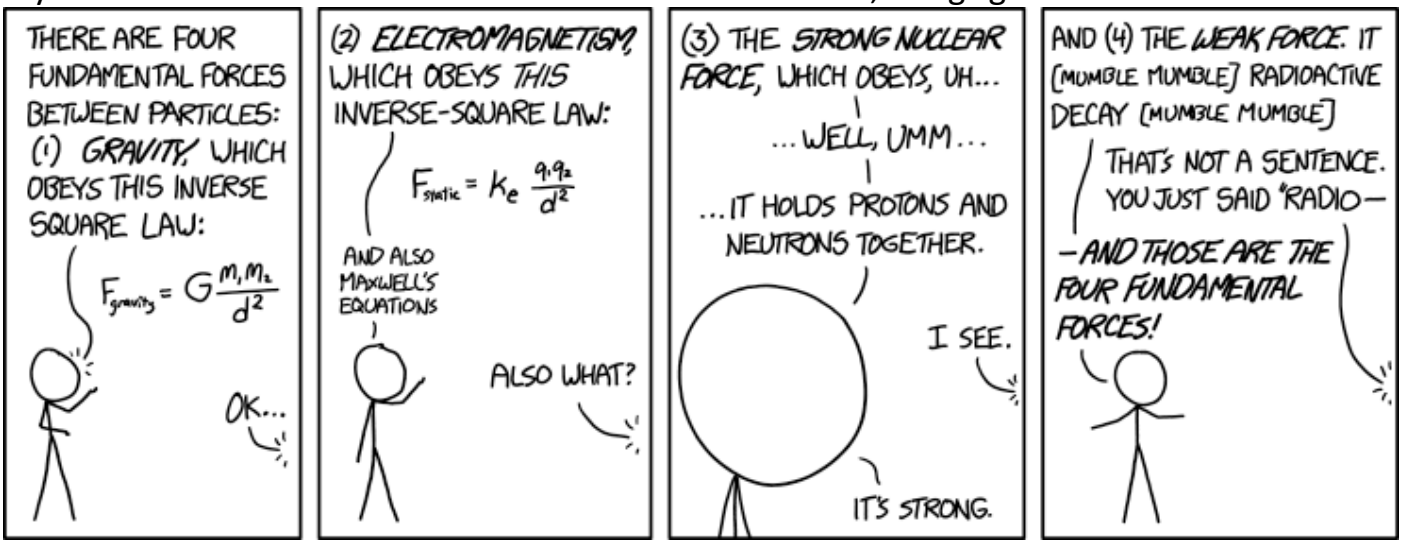
Charged Particles

Particles with mass are attracted by one of the four fundamental forces: gravity. Gravitational attraction is proportional to the mass of each particle and inversely proportional to the square of the distance between them.
Electrically charged particles experience an electromagnetic force which is similar to gravity in that it is proportional to the magnitude of the two charges and inversely proportional to the distance between. The one important difference is that charges can be negative, meaning we can get both attraction and repulsion.

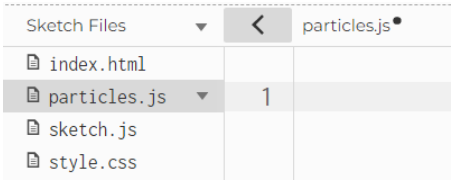
If you're curious about the other two fundamental forces, things get a little trickier…



xkcd.com

Setting up the environment

In order to explore and investigate the behaviour of charged particles, we will need a way of creating particles which move according to the law of electromagnetic attraction.



Start with a new file, *particles.js*, to house our class code, and don't forget to add a reference to the new file in *index.html*.

```
<script src="sketch.js"></script>
<script src="particles.js"></script>
```

The Particle class

```
class Particle{
  constructor(x, y, q){
    this.p = createVector(x, y);
    this.v = createVector(0, 0);
    this.a = createVector(0, 0);
    this.q = q;
  }
}
```

The *Particle* class will need a position and a charge as a minimum. In order to move, it'll also want velocity and acceleration vectors.

Displaying particles

```
display(){
  let r = map(abs(this.q)**0.5,1,10, 5, 30);
  noStroke();
  if (this.q > 0){fill("red")}
  else {fill("blue")}
  circle(this.p.x, this.p.y, 2*r);
}
```

We want a red circle for positive charges, and a blue for negative. So that the area of the circle is proportional to the charge, I'm using the square-root of *q* to determine the radius.

Checking progress

```
let particles = [];

function setup() {
  createCanvas(400, 400);
  particles.push(new Particle(width/2, height/2, -1));
  particles.push(new Particle(width/2, height/4, 2));
}

function draw() {
  background(220);
  for (let particle of particles){
    particle.display();
  }
}
```

If you're anything like me, you'll make a small but non-zero number of minor errors even (possibly especially) while writing routine code. In general, if you can find a way to 'check your working' at regular intervals it makes it much easier to find and fix the small errors as you go along.

In the main *sketch.js* file, make an array for the particles, add a few and display them.

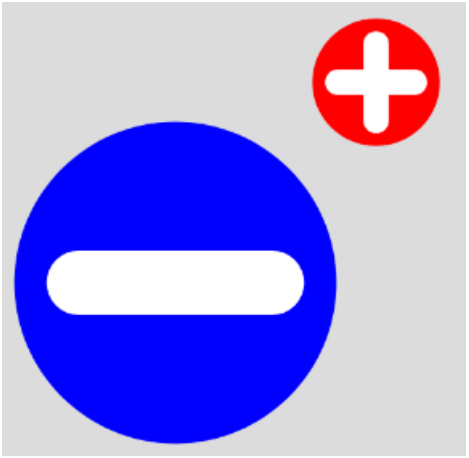
Adding the add

```
display(){
  let r = map(abs(this.q)**0.5,1,10, 5, 30);
  noStroke();
  if (this.q > 0){fill("red")}
  else {fill("blue")}
  circle(this.p.x, this.p.y, 2*r);
  fill("white");
  let w = r/5;
  rect(this.p.x - r + w, this.p.y - w,
       r * 2 - 2*w, 2*w, w);
  if (this.q > 0){
    rect(this.p.x - w, this.p.y - r + w,
         2*w, r * 2 - 2*w, w);
  }
}
```

Positively charged particles should have a + sign, and negatives a – sign. We could use text, but drawing the shapes directly will make it easier to scale them up and down as needed.

I am defining a dummy variable *w* to set the width of the lines, and using rectangles for each line. Note that the *rect* function takes the arguments *x, y, w, h, r* where *x* and *y* represent the coordinates of the top-left corner, *w* gives the width, *h* the height, and *r* the radius of curvature (if required) for the corners.

Since all the parameters depend upon the size of the circle, making some extra-large particles will help you get the proportions right.



Charges may apply

It's time to implement motion according to the electromagnetic force. We'll need an *update* method within our *Particle* class, along with a couple of helper methods to deal with the details.
It's often helpful to write the functions in the order in which they'll be called (top-down), and worry about the details when we get there. This can be thought of as a 'low-resolution' plan, much like early plans for HS2 probably involved laying a ruler across a map of England, leaving smaller details till later in the process. All too often, as work is done on the overall plan, the precise requirements at the high-resolution level change substantially.

```
update(particles){
  for (let particle of particles){
    if (particle != this){
      this.applyCharge(particle);
    }
  }
  this.v.add(this.a);
  this.p.add(this.v);
  this.a.mult(0);
}
```

We start with an *update* method which takes *particles* as its argument. Since this is anticipated to be an array containing all particles, including this one, we only apply the force for the other particles. Note that I'm leaving the details of the *applyCharge* method for later.

Electromagnetic force

```
applyCharge(particle, k=1){
  let f = p5.Vector.sub(this.p, particle.p);
  let dist = f.mag();
  f.normalize();
  f.mult(k * this.q * particle.q / dist**2);
  this.a.add(f);
}
```

We begin by constructing a vector from the particle to *this*. I need both the direction (for the force) and the size (for the distance).
If the charges are the same, their product will be positive, and the force will be away from the particle. If different, the force will be towards the particle.

```
function draw() {
  background(220);
  for (let particle of particles){
    particle.update(particles);
    particle.display();
  }
}
```

Back in *sketch.js*, we need to include a call to the new *update* method. Remember to pass it the *particles* array.

Consider what happens if/when two particles are very close, or even share the same location. How can we fix this?

Know your limits

In *constructor*:

```
this.q = q;
this.maxV = 10;
```

In *update*:

```
this.v.add(this.a);
this.v.limit(this.maxV);
this.p.add(this.v);
```

One of the simplest ways to make sure particles don't fly off at daft speeds is to set a limit.

In the *constructor* method, set a *maxV* attribute, and in the *update* method, use the vector method *limit* to ensure the velocity never exceeds a given value.

```
applyCharge(particle, k=1){
  let f = p5.Vector.sub(this.p, particle.p);
  let dist = max(f.mag(), 2);
```

Another sensible precaution deals with the edge case of two particles sharing a position. In *applyCharge*, the distance is set to never go below 2 pixels, so we never have a divide-by-zero error.

Don't let them get away

Although repulsion is to be expected between opposing charges, it would be nice if the particles don't disappear off the screen. One way to ensure this is to set some form of artificial boundary control (eg bounce if you hit the edge), but quite an elegant solution would be to surround particles with a fixed ring of charged particles. We should build in an option for particles to be 'fixed', so they don't move.

```
class Particle{
  constructor(x, y, q, fixed=false){
    this.p = createVector(x, y);
    this.v = createVector(0, 0);
    this.a = createVector(0, 0);
    this.q = q;
    this.maxV = 10;
    this.fixed = fixed;
  }
```

An optional argument *fixed* is added to the *constructor* function, and we can check this in *update* before deciding whether to apply a charge:

```
update(particles){
  for (let particle of particles){
    if (particle != this && !this.fixed){
      this.applyCharge(particle);
    }
```

```
function setup() {
  createCanvas(400, 400);
  createBoundary();
```

In *sketch.js*, we make reference to a *createBoundary* function in *setup* which will form a ring of fixed charged particles.

```
function createBoundary(n=100){
  for (let i=0; i<n; i++){
    let x = width/2 + width*0.4 * cos(2*PI*i/n);
    let y = height/2 + height*0.4 * sin(2*PI*i/n);
    particles.push(new Particle(x, y, -2, fixed=true));
  }
}
```

This function will produce a circle of charged particles which are fixed in place, and yet will still exert force on others.

A few improvements and things to try

Drag

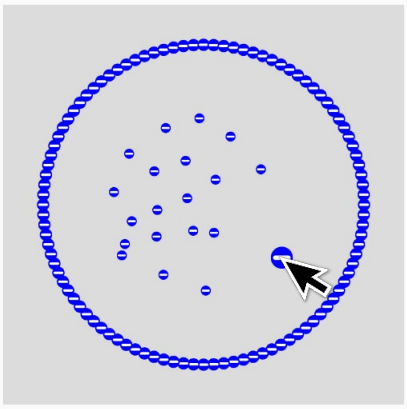
It is interesting to investigate the behaviour of, say, a pair of negatively charged particles within a fixed ring of negatively charged particles. Their motion will approach a periodic repetitive pattern, however. Try building in some drag by reducing the velocity by a small percentage in every *update* call:

```
this.v.add(this.a);
this.v.limit(this.maxV);
this.v.mult(0.99);
this.p.add(this.v);
```

Mouse

Add a feature where a particle, instead of having its position dictated by charge, tracks the mouse.

Maybe use the built-in *mousePressed* function to flip the charge of the particle whenever you click.



Electromagnetic strength

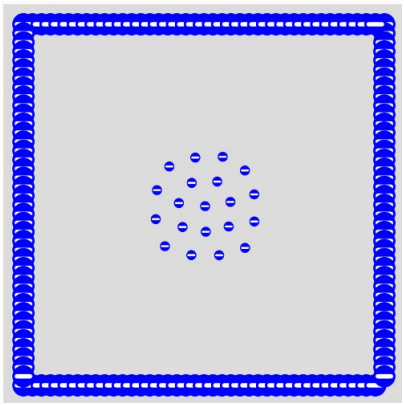
Modify the value of *k* in the *applyCharge* function to change how strong the force is. This should increase the repellent force between electrons.

```
applyCharge(particle, k=10){
```

Square boundary

Make a square boundary instead of a circular one. You could even make it lie just outside the canvas if you want it hidden.

```
function createSquareBoundary(q=-10, r=4, n=40){
  let b = 20;
  for (let i=0; i<n; i++){
    let d = map(i, 0, n, b, height - b);
    particles.push(new Particle(d, b, q, true));
    particles.push(new Particle(d, height - b, q, true));
    particles.push(new Particle(b, d, q, true));
    particles.push(new Particle(width - b, d, q, true));
  }
}
```



Fox among the chickens

Create a charged particle which moves, but not according to the charge forces. Make it bounce around the screen or something, and watch the other particles scatter out of the way…