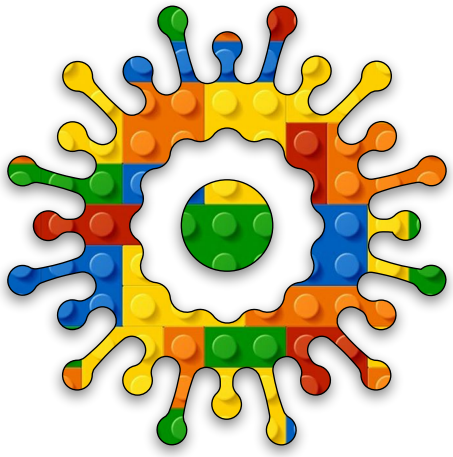
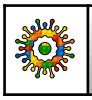
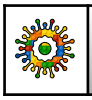
Physics Module B Unit #3 simple spring





Content

Module A Unit #3 Springs



Introduction to springs

Similar to the pendulum, this is looking at a spring-like simulation. Taken inspiration from Coding Challenge #160 spring forces (Coding Train YouTube channel).

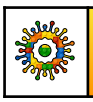
The spring simulation is split into two parts. The first section (`unit #3`) is a very simple spring. The second section (`unit #4`) creates a springy string. Both of them are simulation spring-like motion that you might see in the real world.

The equation for a spring is:

```
force = -1 * k * x
```

Where:
`k` is the spring constant and
`x` is the displacement

It is `-1` because the force is acting in the upwards direction and the mass of the bob is `1`.

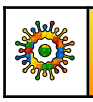


Sketch A3.1 starting point

❗ sketch.js

Start a new sketch in `sketch.js`.

```
function setup()
{
  createCanvas(400, 400)
}

function draw()
{
  background(220)
}
```



Sketch A3.2 adding a bob

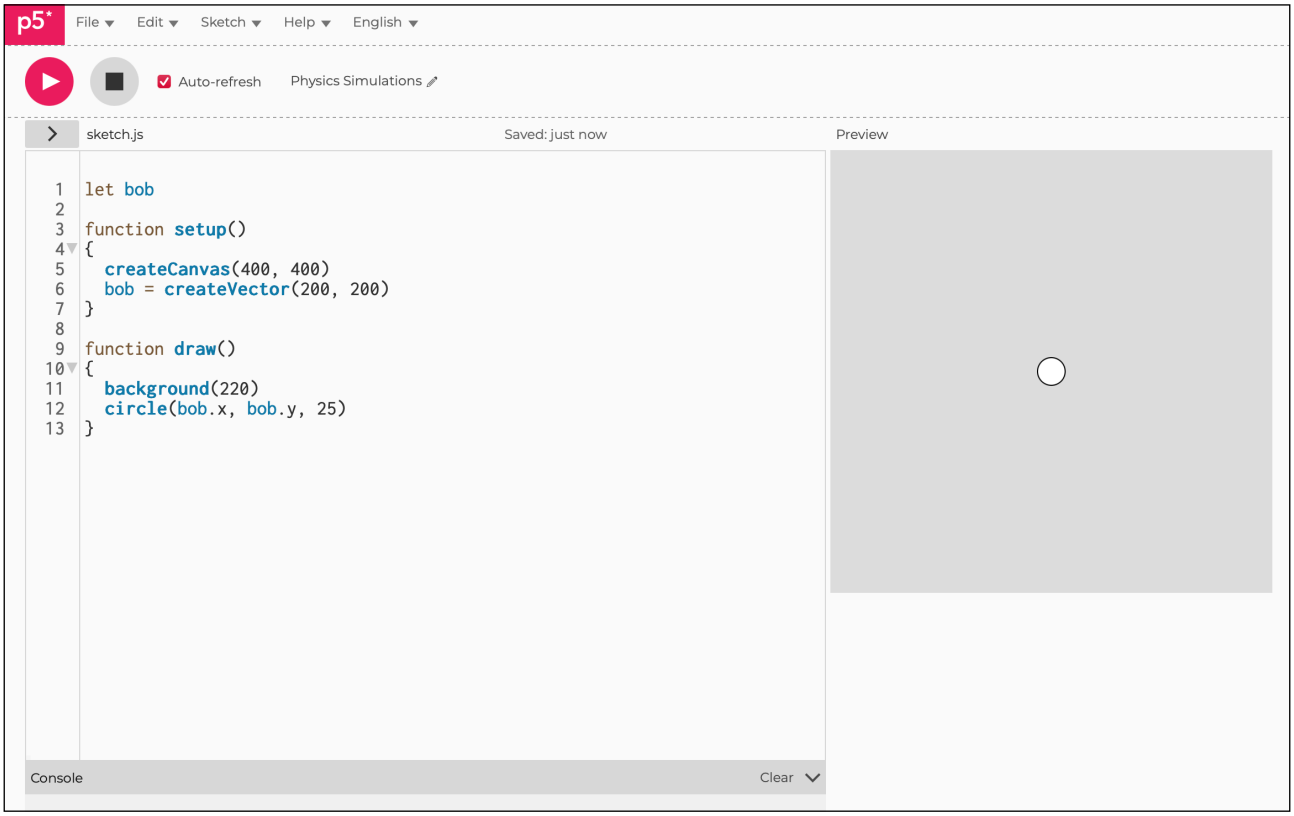
A bob, in case you didn't know, is the weight on the end of the string or spring. The other end is called the anchor for obvious reasons. We use a vector to place the bob on the canvas.

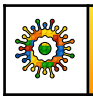
```
let bob

function setup()
{
  createCanvas(400, 400)
  bob = createVector(200, 200)
}

function draw()
{
  background(220)
  circle(bob.x, bob.y, 25)
}
```

Notes

Next, we need an anchor and a connecting spring.

Figure A3.2

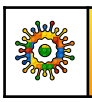

p5*
File
Edit
Sketch
Help
English
Auto-refresh
Physics Simulations
sketch.js
Saved: just now
Preview
let bob
function setup()
{
createCanvas(400, 400)
bob = createVector(200, 200)
}
function draw()
{
background(220)
circle(bob.x, bob.y, 25)
}
Console
Clear



Sketch A3.3 anchor and spring

As with the bob, we use a vector for the anchor and connect the two together with a spring (line).

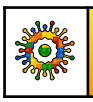
```
let bob
let anchor

function setup()
{
  createCanvas(400, 400)
  bob = createVector(200, 200)
  anchor = createVector(200, 0)
}

function draw()
{
  background(220)
  line(anchor.x, anchor.y, bob.x, bob.y)
  circle(anchor.x, anchor.y, 25)
  circle(bob.x, bob.y, 25)
}
```



Sketch A3.4 gravity

We need to add some gravity. For this simulation, we will take the mass as one and gravity as some arbitrary number that fits in with the simulation. Nothing will happen because we have not introduced gravity (or any other force) to the bob.

```
let bob
let anchor
let gravity

function setup()
{
  createCanvas(400, 400)
  bob = createVector(200, 200)
  anchor = createVector(200, 0)
  gravity = createVector(0, 0.1)
}

function draw()
{
  background(220)
  line(anchor.x, anchor.y, bob.x, bob.y)
  circle(anchor.x, anchor.y, 25)
  circle(bob.x, bob.y, 25)
}
```



Sketch A3.5 adding a force

The force acting on the spring is the difference between the bob and the anchor. We also need a rest length.

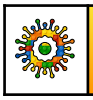
```
let bob
let anchor
let gravity
let restLength = 200

function setup()
{
  createCanvas(400, 400)
  bob = createVector(200, 200)
  anchor = createVector(200, 0)
  gravity = createVector(0, 0.1)
}

function draw()
{
  background(220)
  line(anchor.x, anchor.y, bob.x, bob.y)
  circle(anchor.x, anchor.y, 25)
  circle(bob.x, bob.y, 25)
  let force = p5.Vector.sub(bob, anchor)
}
```

Notes

Nothing is yet happening to the bob.

🛠️ Code Explanation

`let force = p5.Vector.sub(bob, anchor)`	When you subtract two vectors you receive a third vector, in this case the force on the spring.



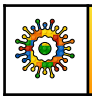
Sketch A3.6 the displacement

The force is a function of the distance between the bob and the anchor. If we stretch the spring, the force increases but only in relation to the rest length. The x value relates to the extension of the spring (displacement). We want the magnitude of that force but only relative to the rest length.

```
let bob
let anchor
let gravity
let restLength = 200

function setup()
{
  createCanvas(400, 400)
  bob = createVector(200, 200)
  anchor = createVector(200, 0)
  gravity = createVector(0, 0.1)
}

function draw()
{
  background(220)
  line(anchor.x, anchor.y, bob.x, bob.y)
  circle(anchor.x, anchor.y, 25)
  circle(bob.x, bob.y, 25)
  let force = p5.Vector.sub(bob, anchor)
  let x = force.mag() - restLength
}
```

🛠️ Code Explanation

`let x = force.mag() - restLength`	The mag() function returns the magnitude of the vector



Sketch A3.7 the constant

We introduce the spring constant `k` and the equation to calculate the `force` based on the `displacement`.

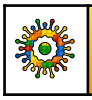
```
let bob
let anchor
let gravity
let restLength = 200
let k = 0.01

function setup()
{
  createCanvas(400, 400)
  bob = createVector(200, 200)
  anchor = createVector(200, 0)
  gravity = createVector(0, 0.1)
}

function draw()
{
  background(220)
  line(anchor.x, anchor.y, bob.x, bob.y)
  circle(anchor.x, anchor.y, 25)
  circle(bob.x, bob.y, 25)
  let force = p5.Vector.sub(bob, anchor)
  let x = force.mag() - restLength
  force.normalize()
  force.mult(-1 * k * x)
}
```

🗒️ Notes

We normalise the force because we don't want the force to be a function of the rest length before we attribute the equation for a spring. The variable is the x value, which is the displacement; the greater the displacement (x), the greater the force. When it is at rest, the force is equal to the gravitational force, and the displacement is very small due to the mass.

🛠️ Code Explanation

`force.normalize()`	This makes the value of the force equal to 1
`force.mult(-1 * k * x)`	We use the equation for a spring



Sketch A3.8 adding the forces

We need to allow the bob to move as the forces act on it. The velocity is a vector which is initialised to zero in the x and y directions.

```
let bob
let anchor
let gravity
let restLength = 200
let k = 0.01
let velocity

function setup()
{
  createCanvas(400, 400)
  bob = createVector(200, 200)
  anchor = createVector(200, 0)
  gravity = createVector(0, 0.1)
  velocity = createVector(0, 0)
}

function draw()
{
  background(220)
  line(anchor.x, anchor.y, bob.x, bob.y)
  circle(anchor.x, anchor.y, 25)
  circle(bob.x, bob.y, 25)
  let force = p5.Vector.sub(bob, anchor)
  let x = force.mag() - restLength
  force.normalize()
  force.mult(-1 * k * x)
  velocity.add(force)
```

```
    velocity.add(gravity)
    bob.add(velocity)
}
```

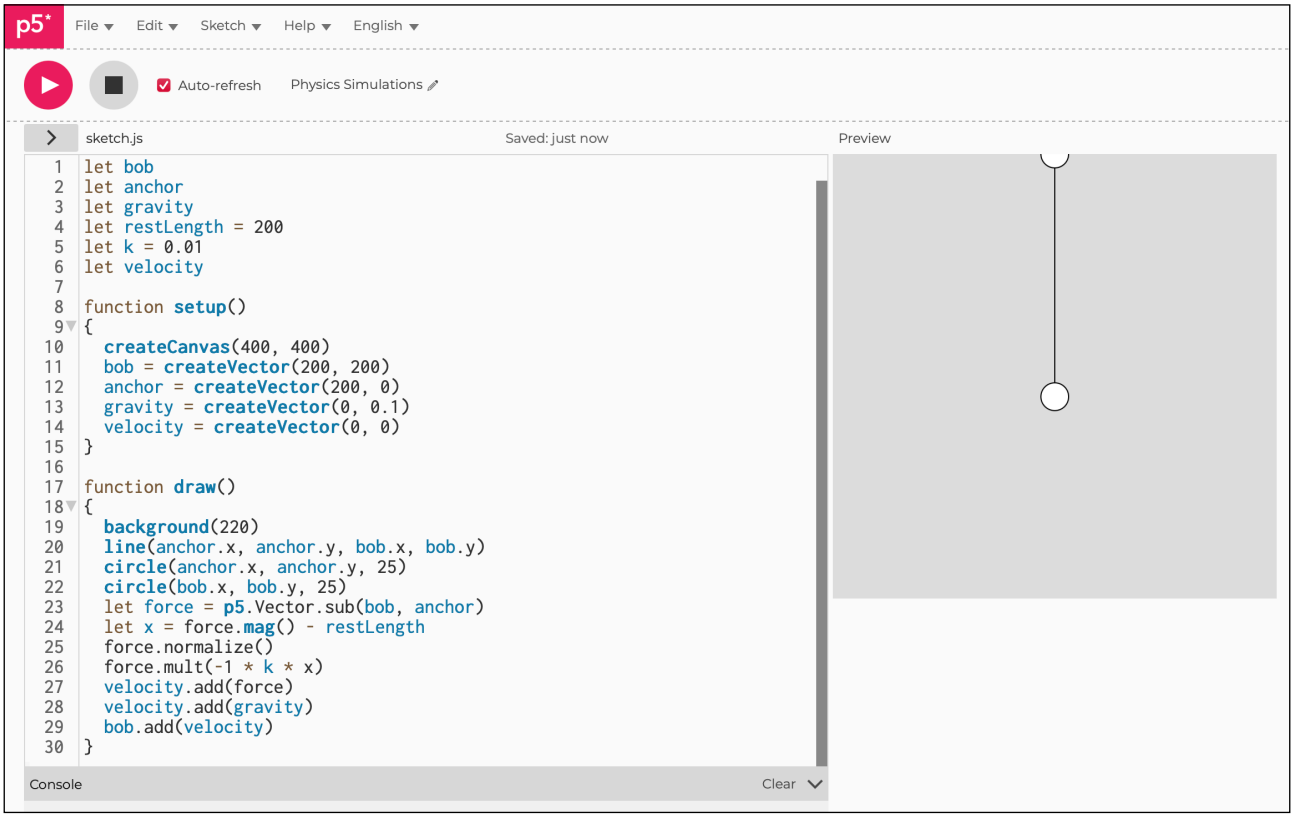
🗒️ Notes

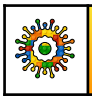
You should see the bob drop very slightly and start to bob up and down as the spring equation takes effect.

🛠️ Code Explanation

`velocity.add(force)`	First we add the force to the velocity
`velocity.add(gravity)`	Next we add the gravity vector
`bob.add(velocity)`	Finally we add the velocity component to to bob itself

Figure A3.8


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Physics Simulations
sketch.js
Saved: just now
Preview
let bob
let anchor
let gravity
let restLength = 200
let k = 0.01
let velocity
function setup()
{
createCanvas(400, 400)
bob = createVector(200, 200)
anchor = createVector(200, 0)
gravity = createVector(0, 0.1)
velocity = createVector(0, 0)
}
function draw()
{
background(220)
line(anchor.x, anchor.y, bob.x, bob.y)
circle(anchor.x, anchor.y, 25)
circle(bob.x, bob.y, 25)
let force = p5.Vector.sub(bob, anchor)
let x = force.mag() - restLength
force.normalize()
force.mult(-1 * k * x)
velocity.add(force)
velocity.add(gravity)
bob.add(velocity)
}
Console
Clear



Sketch A3.9 dampening

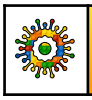
In real life, the bob wouldn't continue to move indefinitely; it would slowly come to a halt. So, to make it more realistic, we add some dampening to introduce a decay.

```
let bob
let anchor
let gravity
let restLength = 200
let k = 0.01
let velocity

function setup()
{
  createCanvas(400, 400)
  bob = createVector(200, 200)
  anchor = createVector(200, 0)
  gravity = createVector(0, 0.1)
  velocity = createVector(0, 0)
}

function draw()
{
  background(220)
  line(anchor.x, anchor.y, bob.x, bob.y)
  circle(anchor.x, anchor.y, 25)
  circle(bob.x, bob.y, 25)
  let force = p5.Vector.sub(bob, anchor)
  let x = force.mag() - restLength
  force.normalize()
  force.mult(-1 * k * x)
```

```
  velocity.add(force)
  velocity.add(gravity)
  bob.add(velocity)
  velocity.mult(0.99)
}
```

🛠️ Code Explanation

`velocity.mult(0.99)`	This multiples the velocity by 0.99 which gives it a slight decay on each iteration.



Sketch A3.10 mouse

This is all well and good, but we can interact with the bob and move it as if we are pulling the bob with our hands. We reset the velocity to zero using the set() function; otherwise, we have a residual velocity and unrealistic movement.

❗ click on the canvas to move the bob and release the mouse to release the bob.

```
let bob
let anchor
let gravity
let restLength = 200
let k = 0.01
let velocity

function setup()
{
  createCanvas(400, 400)
  bob = createVector(200, 200)
  anchor = createVector(200, 0)
  gravity = createVector(0, 0.1)
  velocity = createVector(0, 0)
}

function draw()
{
  background(220)
  line(anchor.x, anchor.y, bob.x, bob.y)
  circle(anchor.x, anchor.y, 25)
  circle(bob.x, bob.y, 25)
  if (mouseIsPressed)
```

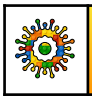
```
  {
    bob.x = mouseX
    bob.y = mouseY
    velocity.set(0, 0)
  }
  let force = p5.Vector.sub(bob, anchor)
  let x = force.mag() - restLength
  force.normalize()
  force.mult(-1 * k * x)
  velocity.add(force)
  velocity.add(gravity)
  bob.add(velocity)
  velocity.mult(0.99)
}
```

🗒️ Notes

Still a little bit unnatural.

🛠️ Code Explanation

`velocity.set(0, 0)`	The velocity is reset using set() function to zero as it clearly isn't moving when released.



Sketch A3.11 improvements

I have made the following tweaks to improve it slightly.

```
let bob
let anchor
let gravity
let restLength = 100
let k = 0.01
let velocity

function setup()
{
  createCanvas(400, 400)
  bob = createVector(200, 100)
  anchor = createVector(200, 0)
  gravity = createVector(0, 1)
  velocity = createVector(0, 0)
}

function draw()
{
  background(220)
  line(anchor.x, anchor.y, bob.x, bob.y)
  circle(anchor.x, anchor.y, 25)
  circle(bob.x, bob.y, 25)
  if (mouseIsPressed)
  {
    bob.x = mouseX
    bob.y = mouseY
    velocity.set(0, 0)
```

```
  }
  let force = p5.Vector.sub(bob, anchor)
  let x = force.mag() - restLength
  force.normalize()
  force.mult(-1 * k * x)
  velocity.add(force)
  velocity.add(gravity)
  bob.add(velocity)
  velocity.mult(0.99)
}
```

🌻 Challenges

1. You could alter the k value.
2. What other values would you need to change?