Newton's Equations

*As of **Week 3**, it is becoming clear that a lot of modeling, simulation, and rendering is going to boil down to computing new positions for particles every time `draw()` is called.*

Computing the new positions realistically requires Newton's Laws of Motion.

The Basics

We begin with the definition of velocity:

```
vx = (xnew - xold) / deltat;
vy = (ynew - yold) / deltat;
```

We usually know `xold` and `yold` and want to know `xnew` and `ynew`. So we solve for those:

```
xnew = xold + vx * deltat;
ynew = yold + vy * deltat;
```

Unless `vx` and `vy` are constants, we are actually going to need to update them too! That comes from the equations for acceleration:

```
ax = (vxnew - vxold) / deltat;
ay = (vynew - vyold) / deltat;
```

Those equations are essentially the definition of acceleration. Just as we did above, we solve for the new values in terms of the old:

```
vxnew = vxold + ax * deltat;
vynew = vyold + ay * deltat;
```

The Plot Thickens

Well, we have a new value of `vx` and an old value of `vx`. Ditto for `vy`. Which of those belongs in the equations for `xnew` and `ynew`?

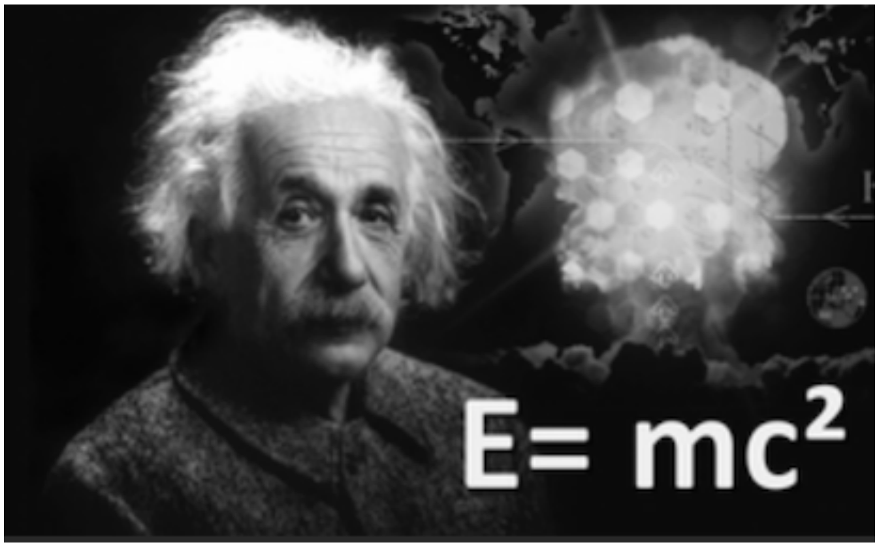
You actually have choices there. The simplest thing is to use `vxold` and `vyold` in the equations for `xnew` and `ynew`. We can be fancier later and try things like `(vxnew + vxold)/2` instead of just using `vxold`, but at present simpler is better.

So with the simplest choice for updating `xnew` and `ynew`, we now have the following four equations:

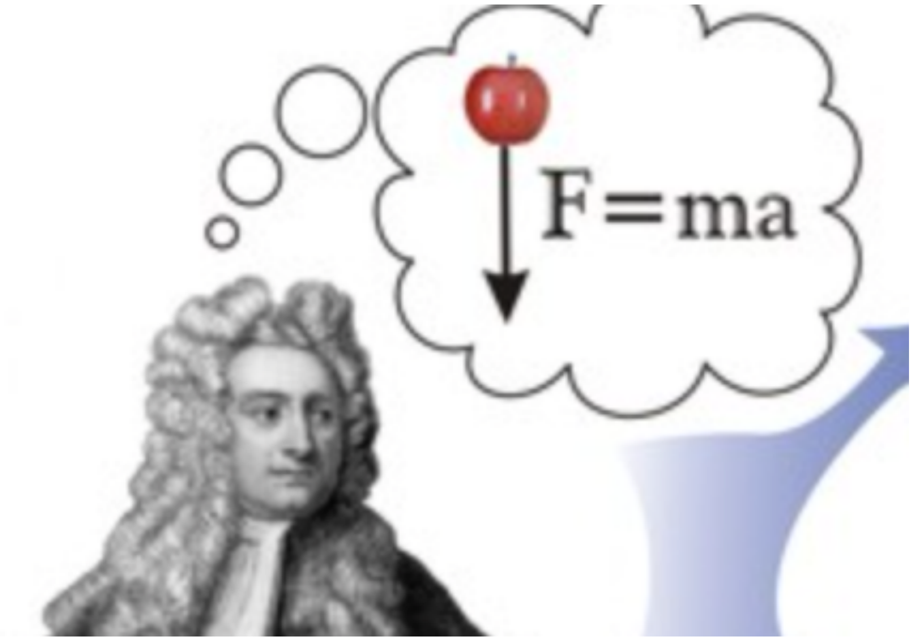
```
xnew = xold + vxold * deltat; // 1
ynew = yold + vyold * deltat; // 2

vxnew = vxold + ax * deltat;  // 3
vynew = vyold + ay * deltat;  // 4
```

The Most Famous Equation in Physics

You might be wondering where we get `ax` and `ay` from. The answer is the most famous equation in physics:



OOPS! The other most famous in equation in physics:



In that equation, `m` is the mass of the particle whose acceleration we are trying to find. We just divide through by `m` to get the acceleration:

```
ax = fx / m;
ay = fy / m;
```

So where do we get `fx` and `fy` from?

All we have done is trade one thing we didn't know for another. We didn't know `ax`, but now we need to know `fx`, and we didn't know `ay`, but now we need to know `fy`. So it may seem that Newton's formulae for `ax` and `ay` are of next-to-no help. However, in very many systems that we wish to model, `fx` and `fy` are known from physics theory (such as the Law of Universal Gravitation, another of Newton's triumphs).

Putting it all together

```
xnew = xold + vxold * deltat; // 1
ynew = yold + vyold * deltat; // 2

vxnew = vxold + fx * deltat / m; // alternate version of 3
vynew = vyold + fy * deltat / m; // alternate version of 4
```

These are Newton's equations of motion, written the way a software developer would write them. I haven't done Newton full justice, but this is the seed from which all of our physics modeling will grow.

Whether we use equation 3 vs. the alternate version of 3 (and equation 4 vs. the alternate version of 4) will depend on the problem. Sometimes we model `ax` and `ay` directly and use equations 3 and 4. Sometimes we get `fx` and `fy` from theory and use the alternate versions 3 and 4.

NB

If the program has no reason to keep the old values around, it is ok to just stomp the old values with the new values. So in the computation of the new values in the `draw()` function, you might just see this:

```
x = x + vx * deltat;
y = y + vy * deltat;

vx = vx + fx * deltat / m;
vy = vy + fy * deltat / m;
```