We will be modeling waves on a string under tension, as in a guitar. This system is accurately described by the non-dispersive one-dimensional wave equation.

$$\frac{\partial^2 y(x,t)}{\partial t^2} = v^2 \frac{\partial^2 y(x,t)}{\partial x^2} \tag{1}$$

where the wave speed v is given by

$$v = \sqrt{\frac{T}{\mu}} \tag{2}$$

where T is the tension in the string, and μ is the mass per unit length of the string. Since a string could have a non-uniform linear mass density, we could write $\mu(x)$ which gives us a position-dependent speed $v(x)$.

WARNING WARNING WARNING Please do not name your file `string.py`, because there is a bug in python3+numpy which triggers an error on `import numpy` if such a file exists.

We will solve this partial differential equation using a finite-difference approach, in which we treat both time and space in terms of finite differences Δt and Δx. When we do this, our partial differential equation becomes a finite difference equation:

$$\frac{-2y(x,t) + y(x,t-\Delta t) + y(x,t+\Delta t)}{\Delta t^2} = v^2 \frac{-2y(x,t) + y(x-\Delta x,t) + y(x+\Delta x,t)}{\Delta x^2} \tag{3}$$

which we can simplify a bit to look like:

$$2y(x,t) - y(x,t-\Delta t) - y(x,t+\Delta t) = v^2 \tfrac{\Delta t^2}{\Delta x^2} \left(2y(x,t) - y(x-\Delta x,t) - y(x+\Delta x,t)\right) \tag{4}$$

We can now solve for the future value of y, and we will have the Verlet method:

$$y(x,t+\Delta t) = 2y(x,t) - y(x,t-\Delta t) - v^2 \tfrac{\Delta t^2}{\Delta x^2} \left(2y(x,t) - y(x-\Delta x,t) - y(x+\Delta x,t)\right) \tag{5}$$

This is a formula which you can implement. Note that in order to solve for the *future* value of y, you need to know both its current value *and* its previous value.

0.1 Creating a wave packet

So far (with balls and springs) you have mostly been working with normal modes. They are nice, but sometimes are less than helpful, if we want to understand how pulses propogate. Even looking at reflection and transmission, while possible using plane waves, can be inconvenient.

Travelling waves can also lead to interesting insights. Travelling waves are solutions of the wave equation that have the form:

$$y = f(x - vt)$$

where f is any smooth function.

For this task you will implement initial conditions for a "wave packet", which is a travelling wave that is localized in space, i.e. it has $y = 0$ for all positions the left or the right of the wave packet.

1. Write a fresh code to solve the linear wave equation. You may look a the code from the old balls and springs code, but I'd like you to start an entirely new file, and ensure that your variable names match the wave equation (i.e. use v in your code, and there are no "balls"). (As before, you will want to use 2D arrays to store your displacement $y(x, t)$.) Animations will be useful.

2. Implement the initial conditions for a wave packet, and verify that it moves in the expected direction using your code that solves the wave equation.

   Clarification Your wave packet should start somewhere in the center of your string, **not** right next to the wall. There should initially be a clearly visible region of $y \approx 0$ on each side of your wave packet.

   a) Choose a smooth function that approaches zero in both directions. Any function should work, but keep in mind that you'll probably end up wanting to tweak the width of the function as well as its location.

   b) Create a python function of position $f(x)$ that returns your smooth function.

   c) Set your initial conditions (i.e. $y(x, t)$ for the first two time steps) to $f(x - vt)$.

3. As before, create a static visualization of your wave packet. *Note that to be effective this time your static visualization should use time was one of its two axes, so that you can clearly see the packet moving over time.*

0.2 Reflection and transmission

Now imagine that you wrap wire around a portion of the string (think guitar string), increasing its mass density μ for just a portion of its length. This will change the wave speed on that portion of the string. You could alternatively thing of this as tying two different thicknesses of string together.

1. Adjust your code so $\Delta x \neq 1$ and $\Delta t \neq 1$. Further refine your code so that if you have variables named x or t, those variables correspond to the actual position and time in distance and time units (i.e. they are not restricted to integer values).

2. Ensure that you have defined a python function that gives the initial shape of your wave, i.e. $y(x, t = 0)$. Use this function to initialize your string for the first two time steps.

3. Make a portion of the string have half the speed v as the rest of the string. This is most easily accomplished by making v an array, and setting some of its elements to half the value of the rest of the string. Ensure that your wave packet is entirely on a portion of string that has the original speed. Observe the reflected and transmitted pulses with your animation.

4. Create a static visualization or two. Identify the two speeds of your string as slopes on your visualization. *Note: these slopes probably won't be correct unless you do the following task.*

5. Ensure all your figures have proper dimensions for each axis, e.g. x should be in distance units, and should correspond to the length of your string, which is not the same as the number of Δx in your string.

Extra fun Play with different configurations of heavy and light string or different wave packets to make pretty and/or interesting things happen.

Dispersion relation fun

1. Find a normal mode of the system (which has a repeating pattern). What is the period (and frequency) of this normal mode?
2. Plot the normal mode frequency versus $k = \frac{2\pi}{\lambda}$. This is called a **dispersion relation**.

0.3 Numerical stability

An algorithm is numerically unstable if small errors grow exponentially. In the particular case of our finite difference integration of the wave equation, our numerical stability is determined by the relationship between the resolution in space and time, Δx and Δt. To understand numerical stability physically, it is often helpful to consider the dimensions and behavior in the relevant dimensions.

In this case, it is useful to ask how quickly the y coordinate might change, and then to ensure that our Δt is considerably smaller than that value. There are various ways to estimate this value, but for the non-dispersional wave equation, there is a very simple way to think about this. (When we get to Schroedinger's equation, we will use one of the other approaches.)

Consider the case where $y = 0$ for $x > x_0$. We know that a signal will travel at speed v. If $\Delta x < v\Delta t$, then the signal ought to travel by more than one grid point in one time step. In our algorithm (if you look carefully) that cannot happen. Therefore, since we *know* we will get nonsense otherwise, we can conclude that

$$\Delta t < \frac{\Delta x}{v} \tag{6}$$

One can derive a more precise stability limit, but we can also just experiment to find what Δt values give good results. Note that this means that if you increase v, you must also decrease Δt.