Lesson 12: Animating with Tk Interface

In this lesson, we'll start learning how to use the Tk Interface module (or `tkinter`).
The `tkinter` module is built into Python, and lets us draw images and even animations!

We'll use this to create an animation of lines being drawn to the screen randomly!

PART 1: Setting up Tkinter

In order to use any of the functions in the `Tkinter` module, we'll need to import it first.
We'll also import the `random` module, so that we can get random values later.

Try typing the code below into a new program:

```
from tkinter import *

import random

myTk = Tk()
```

Make sure to add the asterisk after the `from tkinter import` line!

Unlike before, this time we're using a *from-import* statement to import the module.
A from-import statement is a slightly different way to import functions from a module.

Instead of the usual format: `import module`

we'll use: `from module import objects`

A `*` is a wildcard, which means it stands for all objects in the module.
When we use `from module import *` , we import all its functions.

A from-import statement lets us import the contents of a module in a way that lets us not have to type the module name before calling any of its functions.

We use this to create a new Tk object by using the module's function `Tk()`.
This new Tk object gets stored in the variable myTk.

If we used a normal import statement instead, we'd have to use `myTk = tkinter.tk()`
By using a from-import statement, we can avoid typing `tkinter` extra times.

PART 2: Creating a Tk Canvas

In order to draw images and animations to the screen, we have to use a canvas object.

A Tk canvas works a lot like a real-life canvas used for painting.
Just like an artist draws lines and shapes onto a physical canvas,
we can draw lines and shapes onto our Tk canvas after we create it!

Add the following code to your program:

```
myTk = Tk()

canvas = Canvas(myTk, width=400, height=400)

canvas.pack()
```

Here, we're creating a canvas object, which we've placed in a variable called `canvas`.

In order to make the canvas object, we've called the `Tkinter` function `Canvas()`.
For this, we passed the Tk object it should use, as well as the width and height.

Sometimes, we need to tell python exactly which parameters we
want to set values for when we're calling a function.
We can do this by using the form: `function(var=value)`

After creating a canvas, we used its `pack()` function to place it in the Tk window.

Whenever we make a `tkinter` object, we'll always need to pack it into to the window.

PART 3: Displaying the Tk Window

By now, we've created a canvas object and placed it into our Tk window.

If you were to run your program now though, you wouldn't see the window appear.
This is because we still have to create a *main loop* for the Tk object.

A main loop in a program is a loop that runs continuously until the program exits, and controls input and output for the program, like waiting for mouse clicks or drawing lines.

Try adding the following code to your program:

```
canvas.pack()
while True:                 # This loop should run until
    myTk.update()           # the program is closed.
myTk.destroy()
```

Here, we've placed our Tk object's `update()` function inside a continuous loop. The `update()` function handles important Tk tasks, like displaying the Tk window.

In case we ever exit the `while` loop, we added `myTk.destroy()` to close the window.

If you close your Tkinter window, you'll probably get an error. This is because your program is trying to use myTk after it's destroyed.

This is one of the few times when you should ignore an error message.

PART 4: Drawing to the Canvas

Next, we'll learn about how `Tkinter` lets us draw shapes much easier than in `Turtle`.

Any time we want to create a shape in Tkinter, we'll need to create it on a canvas object. In order to do this, we'll call a function that follows this format:

```
canvas.create_shape(150, 200, 250, 300, fill="yellow",  . . .)
                    x1   y1   x2   y2        color     others
```

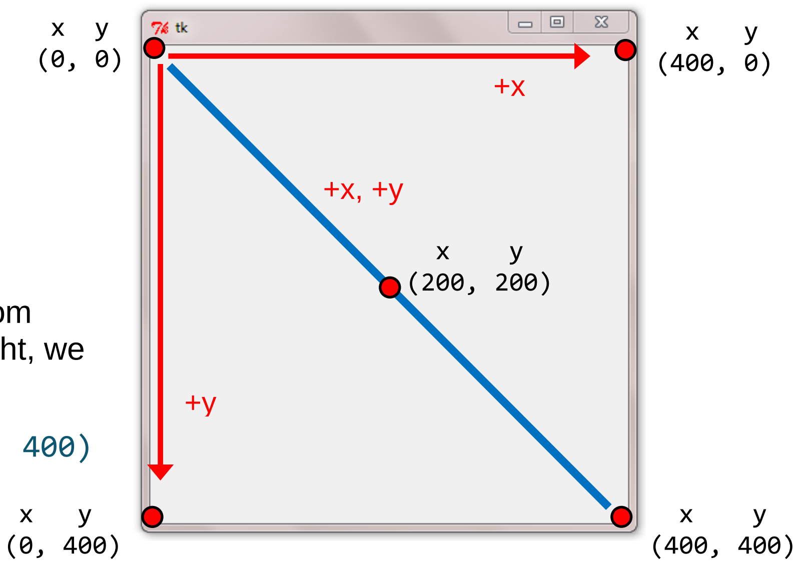
Where `(x1, y1)` are the coordinates for one endpoint, and `(x2, y2)` are the other's.

Tkinter uses a different coordinate system than you use in math class!

In Tkinter, the upper-left corner is the *origin* point `(0, 0)`, and lower coordinates have a larger value in the y-direction.

Here, if we wanted to draw a line from the upper left corner to the lower right, we would call the function:

`canvas.create_line(0, 0, 400, 400)`



Using this knowledge, we'll try to create an interesting pattern out of random lines!

At this point, our program should create an empty window that's 400 pixels wide and tall.
Now, we can define a function that will draw lines randomly onto the canvas we made.

Try adding the following code to your program:

```
canvas.pack()

def drawline():

    x = random.randint(0, 400)    # Create two random integers to use

    y = random.randint(0, 400)    # as the endpoint's coordinates

    canvas.create_line(200, 200, x, y) # Use a start point of (200, 200)

while True:

    drawline()      # Make sure you add this to the main loop!

    myTk.update()
```

Here, we've made a function `drawline()` that we'll call continuously in the main loop.

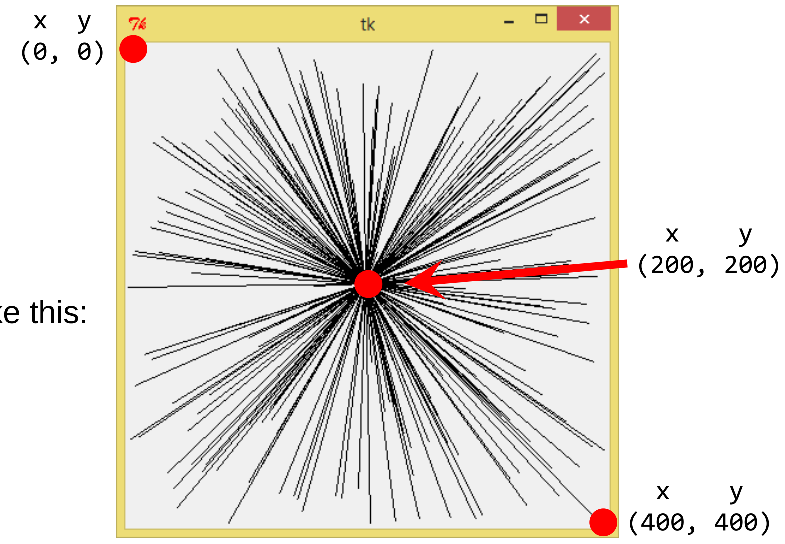
In it, we tell our canvas to make a line by calling `canvas.create_line()`, and passing it coordinates for the start point and the endpoint.

In this case, our program is making lines that all start at the coordinates `(200, 200)`, which is at the center of the window.
These lines extend out towards the point `(x, y)`, whose coordinates we got randomly.

Repeating this in the main loop will draw a new line every update!

Now, if you test your program, it should draw random lines in the window!

The result should look something like this:



PART 5: BONUS: Adding Color to your Lines!

Once your program is drawing lines correctly, we can start drawing them in color!

In order to do this, we'll create a list of possible colors, and choose from it randomly.

Try adding the following code to your program:

```
canvas.pack()
colorlist = ["red", "green", "blue"]
def drawline():
    x = random.randint(0, 400)
    y = random.randint(0, 400)
    color = random.choice(colorlist)
    canvas.create_line(200, 200, x, y, fill=color)  # Add this!
while True:
    . . .
```

Make sure to change the `create_line()` function so `fill=color`!

Here, we created a list of colors for our lines to choose from at random.

Then, in the `drawline()` function, we made a variable called color, and set it to the return value of the `random.choice()` function.

`random.choice()` chooses a random item from the list we give it, so `color` will be one of the colors from `colorlist`.

Lastly, we changed `create_line()` so that the line drawn would be the color we chose.

You can put whatever colors you like into `colorlist`!

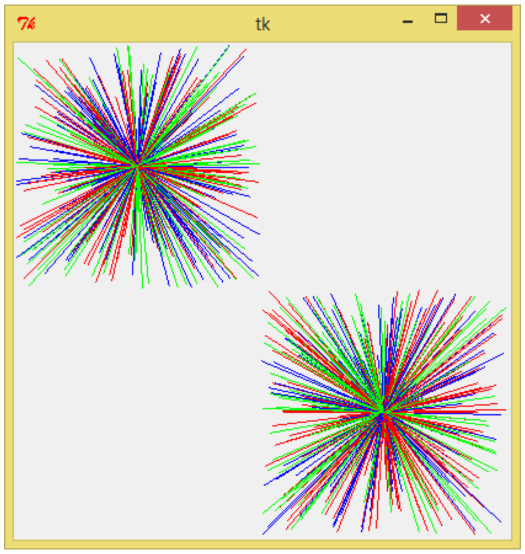
Now, your program should draw lines with random colors!

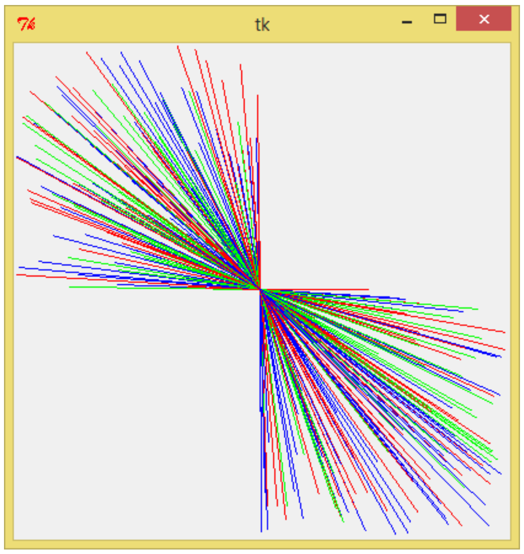
PART 6: BONUS: Making Different Patterns

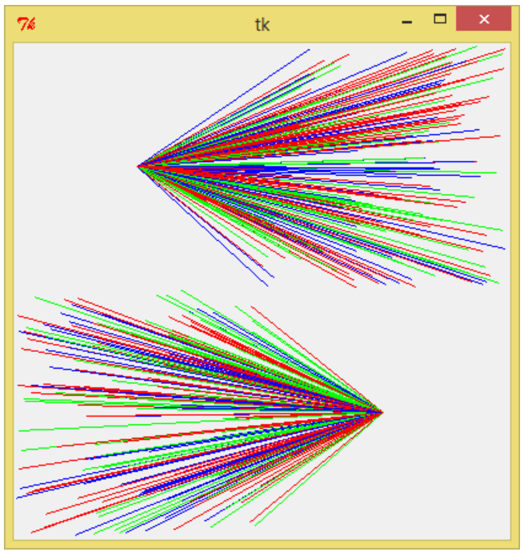
Using what we've learned about the coordinate system in `Tkinter`, we can modify our program to make many different line patterns!

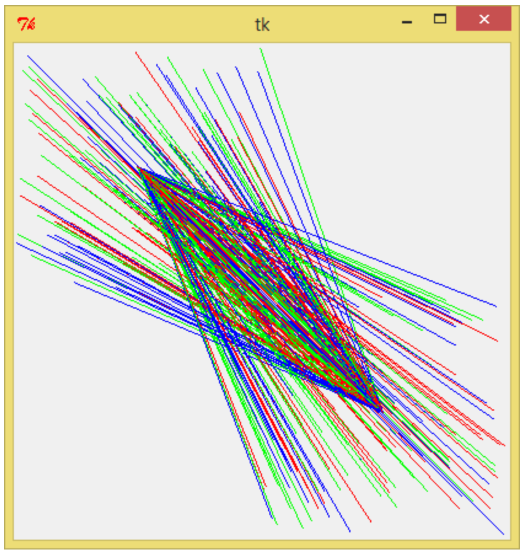
By making a second set of random coordinates (`x2`, `y2`) and creating a second line with `canvas.create_line()` inside of `drawline()`, you can make the patterns below!

Try modifying your `drawline()` function to make one of these patterns:









You can also try designing your own interesting patterns!