

Graphics

15-110 – Monday 09/23

Learning Goals

Use **programming** to specify algorithms to computers

- Use a **graphics library** to make programs that have visual output
- Use graphics to visualize **algorithmic patterns**

Tkinter is a Graphics Library

In order to start producing graphical results (instead of only text results), we need to import a new library that lets us draw shapes on the screen.

We'll import it a little differently than usual, in order to import everything into the library directly into the same **namespace**.

```
from tkinter import *
```

How Tkinter Works

In order to create graphics, we need to take a few preliminary steps. These will be provided to you as starter code, inside `makeCanvas()`.

First, **create a new window**- that's the thing that pops up on the screen.

Second, **create a new canvas**- that's the thing we can draw graphics on.

Next, **pack the canvas into the window**- that tells the canvas to fill the whole window.

We'll do all our drawing in `draw()`.

Finally, the last line will tell the window to stay open until we press the X button.

```
from tkinter import *
```

```
def draw(canvas):  
    pass
```

```
def makeCanvas(w, h):
```

```
    root = Tk()
```

```
    canvas = Canvas(root, width=w, height=h)
```

```
    canvas.configure(bd=0, highlightthickness=0)
```

```
    canvas.pack()
```

```
    draw(canvas)
```

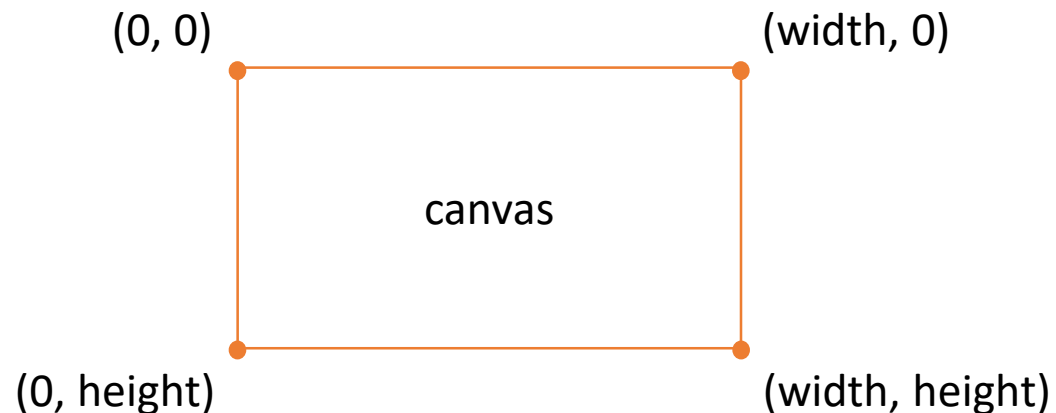
```
    root.mainloop()
```

```
makeCanvas(400, 400)
```

Coordinates on the Canvas

The canvas is a two-dimensional grid of pixels, where each pixel can be filled with a dot of color. This grid has a pre-set **width** and **height**; the number of pixels from left to right and the number of pixels from top to bottom.

We can refer to pixels on the canvas by their (x, y) coordinates. However, these coordinates are different from coordinates on normal graphs- they start at the **top left corner** of the canvas.

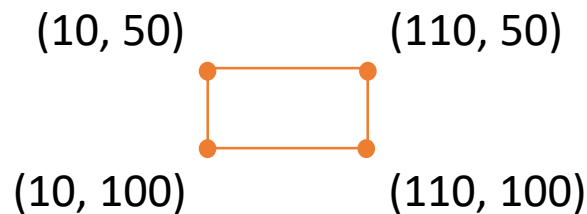


Drawing Basic Shapes

Drawing a rectangle

To draw a rectangle, we use the function **create_rectangle**. This function takes four required parameters: the x and y coordinates of the **left-top** corner, and the x and y coordinates of the **right-bottom** corner. The rectangle will then be drawn between those two points.

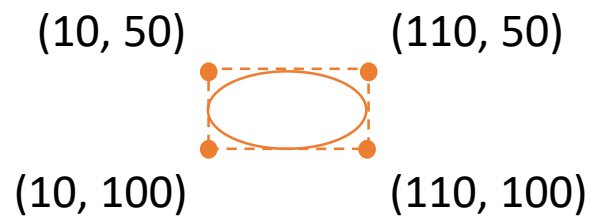
```
canvas.create_rectangle(10, 50, 110, 100)
```



Drawing an oval

We can draw more shapes than just rectangles. To draw an oval, use **create_oval**. This function uses the same parameters as `create_rectangle`, where the coordinates mark the oval's **bounding box**.

```
canvas.create_oval(10, 50, 110, 100)
```



Drawing Squares/Circles

If you want to draw a square or a circle, you need to ensure that the width of the shape equals the height.

How can you do that? Make sure that (right - left) is equal to (bottom - top)!

```
canvas.create_rectangle(50, 100, 150, 200)
```

Keyword Arguments Add Variety

With the basic parameters, we can only draw basic outlines of shapes. By adding **keyword arguments**, we can change the properties of these shapes!

Each keyword argument is associated with a **default value**, which is why we don't need to include them in every graphics call. To change that default value, include the keyword, followed by =, followed by the new value in the function call.

```
canvas.create_rectangle(50, 100, 150, 200, fill="green")
```

Keyword Argument - fill

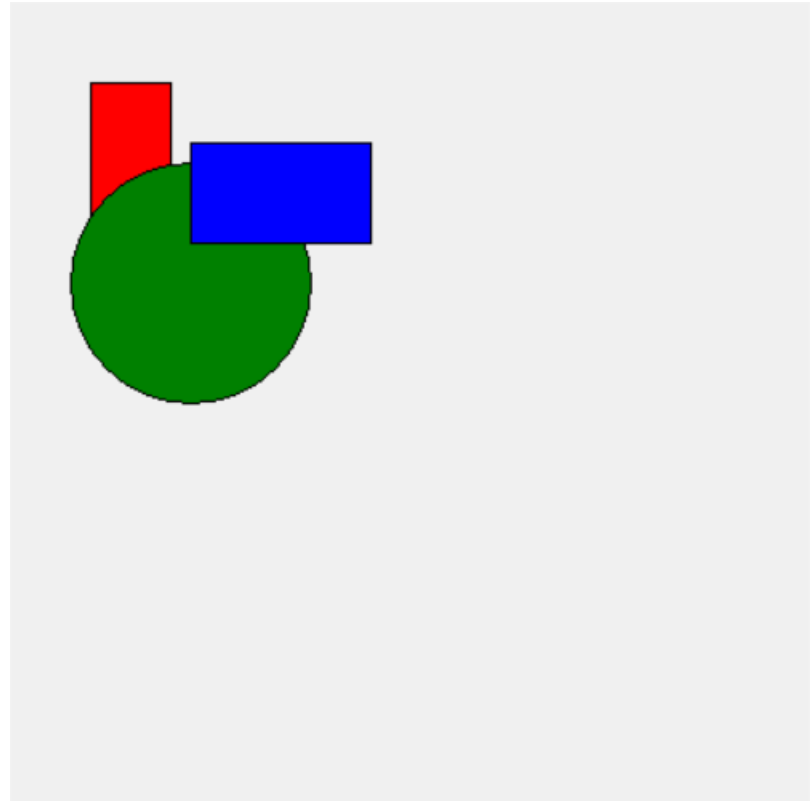
The fill argument can be used on any shape. It uses a string (the name of the color) to change the color of the shape.

```
canvas.create_rectangle(40, 40, 80, 140, fill="red")
```

```
canvas.create_oval(30, 80, 150, 200, fill="green")
```

```
canvas.create_rectangle(90, 70, 180, 120, fill="blue")
```

Note that when we draw shapes on top of each other, the one on top is the **last one called**. Order matters!



Making New Colors

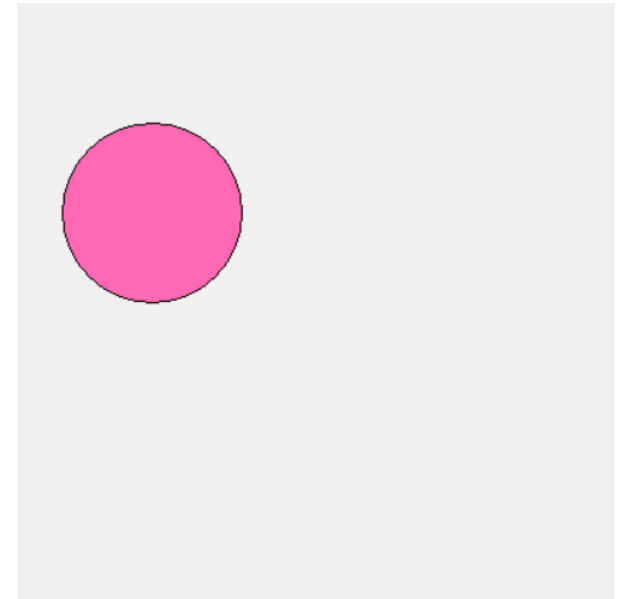
Interested in finding more Tkinter color names? There's a whole databank!

<https://wiki.tcl-lang.org/page/Color+Names%2C+running%2C+all+screens>

If you want to use a color that doesn't have a name, you can make your own, using the RGB system we discussed in Data Representation.

Instead of a name, make a string "#RRGGBB", where you replace RR with the red value in hex, GG with green, and BB with blue. "#FF69B4" is hot pink!

```
canvas.create_oval(30, 80, 150, 200, fill="#FF69B4")
```



Keyword Argument - width

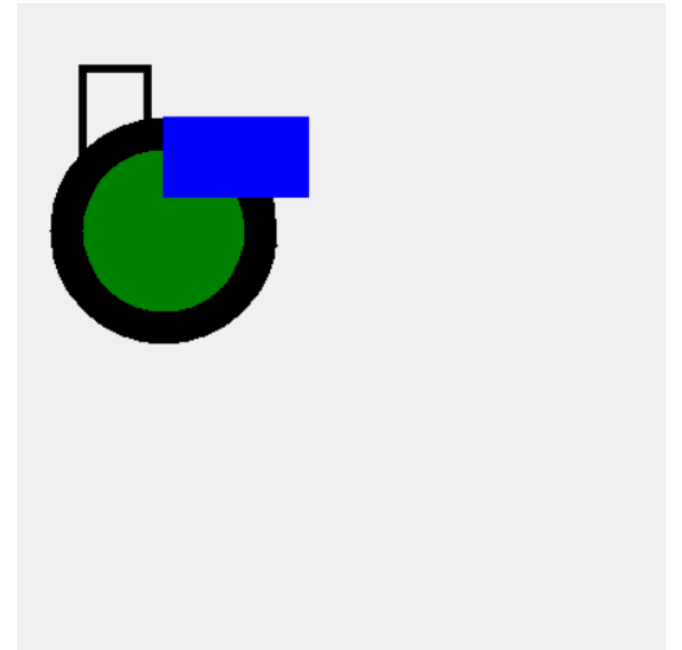
Another keyword argument is **width**, which specifies how many pixels wide the border of the shape should be.

```
canvas.create_rectangle(40, 40, 80, 140, width=5)
```

```
canvas.create_oval(30, 80, 150, 200, width=20, fill="green")
```

```
canvas.create_rectangle(90, 70, 180, 120, fill="blue", width=0)
```

Note that setting width to 0 removes the border completely.



Drawing Text

Drawing text on the canvas works a bit differently from drawing rectangles and ovals. We only specify one coordinate- the pixel where the text will be drawn.

```
canvas.create_text(200, 200, text="Hello World")
```

text is technically a keyword argument, but is necessary in order to draw text at all.

Keyword Argument - font

When drawing text, we can use the keyword argument **font** to change the appearance of the text.

font takes a string with one to three pieces of information- the font name, the font size, and the font type.

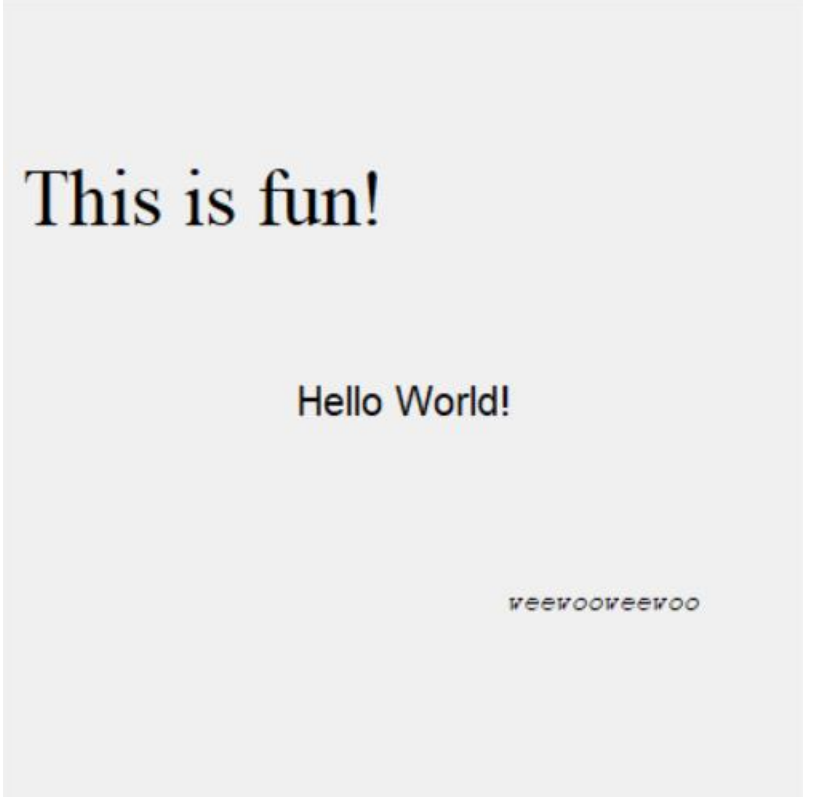
```
canvas.create_text(200, 200, text="Hello World!",  
                  font="Arial")
```

```
canvas.create_text(100, 100, text="This is fun!",  
                  font="Times 30")
```

```
canvas.create_text(300, 300, text="weewooweewoo",  
                  font="Courier 10 italic")
```

You can find a full list of fonts and types here:

<https://effbot.org/tkinterbook/tkinter-widget-styling.htm#fonts>



This is fun!

Hello World!

weewooweewoo

Keyword Argument - anchor

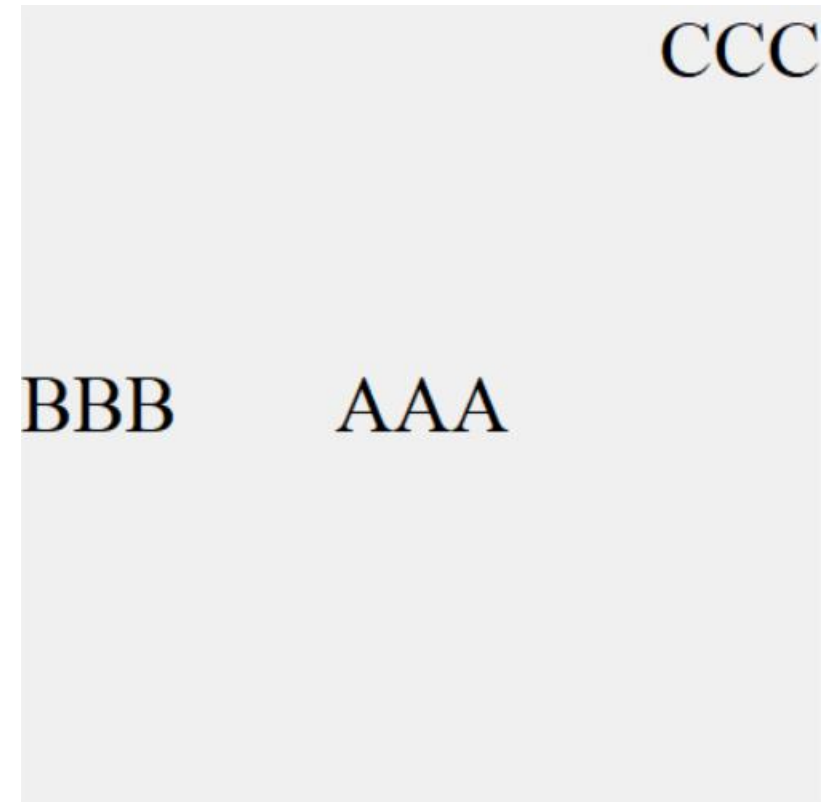
The point used in the `canvas.create_text` call is actually an **anchor** for the text, to describe how it is drawn. That anchor defaults to the center of the text box, but we can change it to be any compass point instead.

```
canvas.create_text(200, 200, text="AAA", font="Times 30",  
                  anchor="center")
```

```
canvas.create_text(0, 200, text="BBB", font="Times 30",  
                  anchor="w")
```

```
canvas.create_text(400, 0, text="CCC", font="Times 30",  
                  anchor="ne")
```

Note that the anchor describes the **point on the text box** that will correspond to the (x, y) coordinate. Since CCC's anchor is "ne" (north-east), the upper-right corner of the text box is set to $(400, 0)$.



Drawing Lines

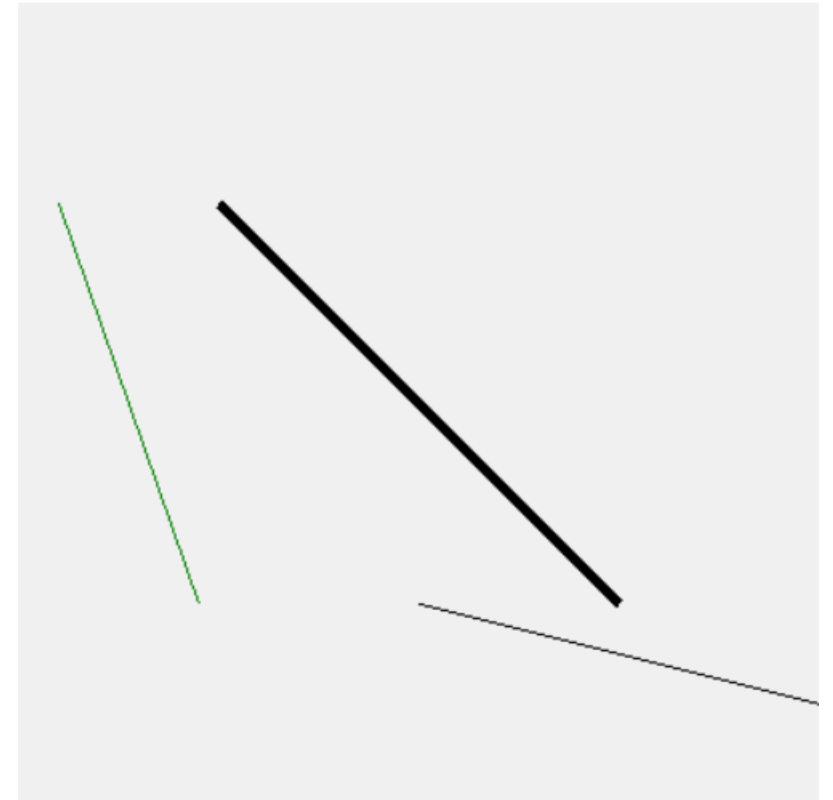
Finally, to draw a line on the screen, we go back to using two coordinates. This time, they are the two endpoints of the line.

```
canvas.create_line(200, 300, 400, 350)
```

```
canvas.create_line(20, 100, 90, 300, fill="green")
```

```
canvas.create_line(100, 100, 300, 300, width=5)
```

Note that we can again use fill and width here to modify the lines.



Problem Solving with Graphics

Thinking Graphically

Now that we have all the basic shapes, we can start putting them together in different ways to make interesting images.

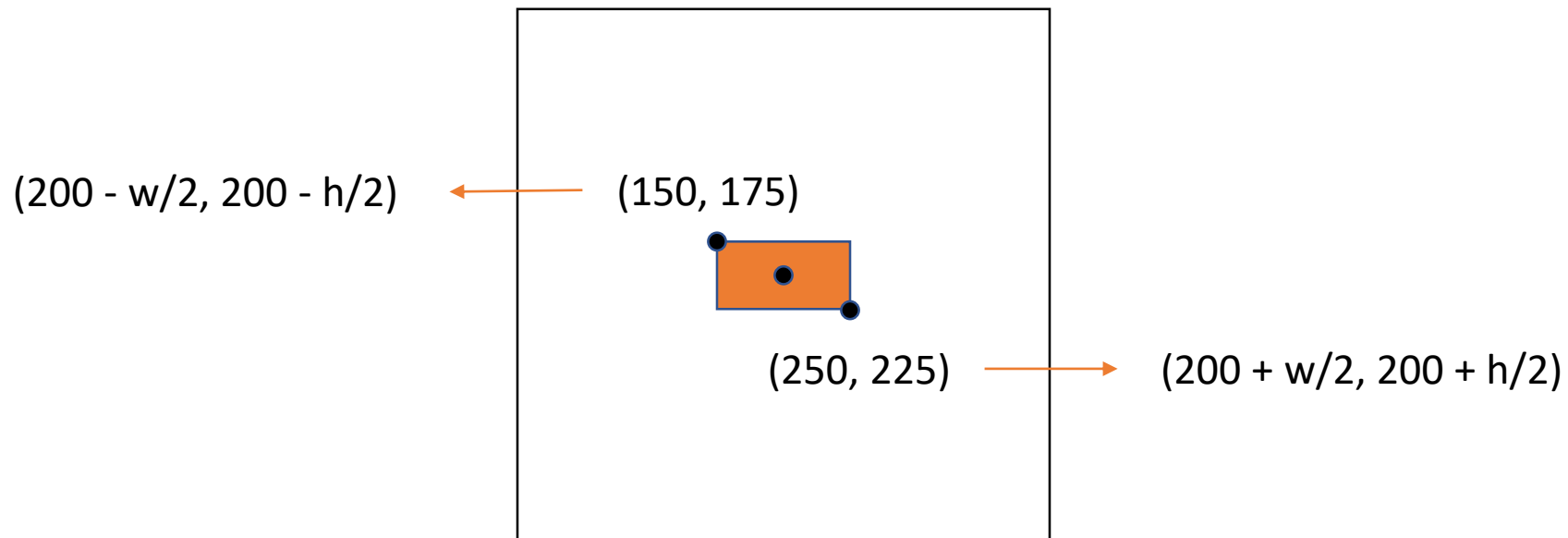
We'll generally do this by first determining where on the canvas a shape should be placed, then using logic and math to determine which coordinates correspond to that position.

We can also use guess-and-check to move shapes around by small amounts of pixels!

Centering a Shape

For example, say we want to place a 100px x 50px rectangle in the middle of a 400px x 400px canvas. How do we find the coordinates?

We can find the center point logically- it's (200, 200). To find the upper-left and lower-right points, use the width and the height!



Drawing a Grid

As another example, let's say we want to draw a 10x10 grid of circles (like bubble wrap). We could do this by calling `canvas.create_oval` 100 times, but that's a lot of work!

Instead, we can use logic to determine how to compute the position of each circle **based on the positions of the other circles**.

Drawing a Row of Circles

First, let's simplify and just draw 10 circles in a row. If we want them to fill the canvas, how wide should each circle be?

```
circleSize = canvasWidth / 10
```

The first circle starts at x coordinate 0; the next is one circle over, so it starts at circleSize. The third circle has two circles before it, so it starts at 2*circleSize.

If we number the circles from 0 to 9, each circle starts at $n * \text{circleSize}$.

Now we can write a graphics call for each circle **in terms of its number**. That means we can draw all ten circles using a loop!

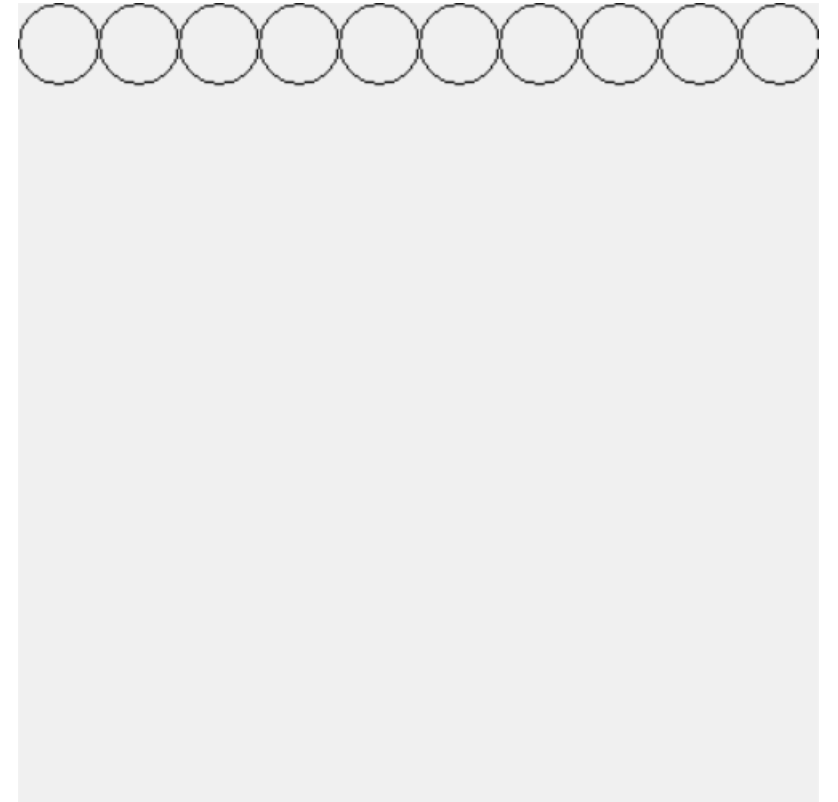
```
canvasWidth = 400
```

```
circleSize = canvasWidth / 10
```

```
for n in range(10):
```

```
    left = n * circleSize
```

```
    canvas.create_oval(left, 0, left + circleSize, circleSize)
```

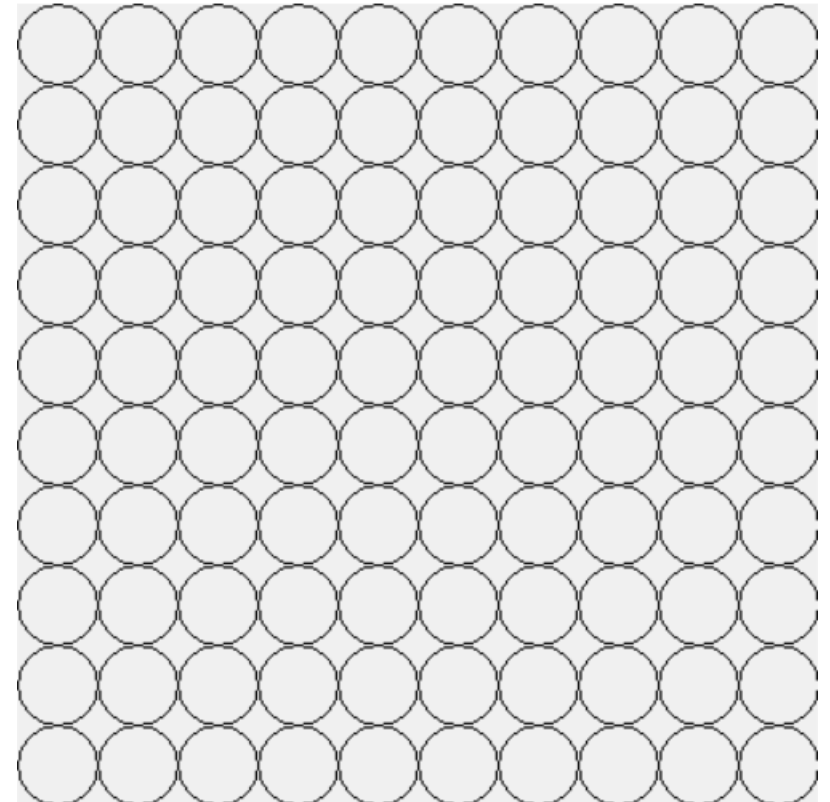


Turn a Row Into a Grid

To go from drawing a row of circles to a grid of circles, we just need to repeat the same logic, except with the y dimension as well. We can't just loop once- **we need to use a nested loop**.

We'll refer to a circle's position in the grid by its **row** (how many circles are above it) and its **column** (how many circles are to the left of it). Rows correspond to y; columns correspond to x. By looping over the rows and columns, we can draw all the circles!

```
canvasWidth = 400
size = canvasWidth / 10
for row in range(10):
    top = row * size
    for col in range(10):
        left = col * size
        canvas.create_oval(left, top, left + size, top + size)
```



Tkinter Can Do Even More!

There's plenty of things Tkinter can draw and plenty of additional keyword arguments that we haven't covered here.

If you're interested in learning more, check out the Tkinter documentation: <http://effbot.org/tkinterbook/canvas.htm>

Learning Goals

Use **programming** to specify algorithms to computers

- Use a **graphics library** to make programs that have visual output
- Use graphics to visualize **algorithmic patterns**