#2: Graphics Part 1

SAMS SENIOR NON-CS TRACK

Last time

Learned what programming is

Used data values and operations to compute values

Used debugging to identify errors in code

Ex1-1 Feedback

Go to Autolab and click on ex1-1 > View handin history

If you click on the blue scores, you can see your feedback!

Most of the problems went well, but the error identification problem was a bit rough. Let's go over that now.

Today's Learning Goals

Recognize how a computer makes images from pixels

Use code to draw rectangles and ovals in the window

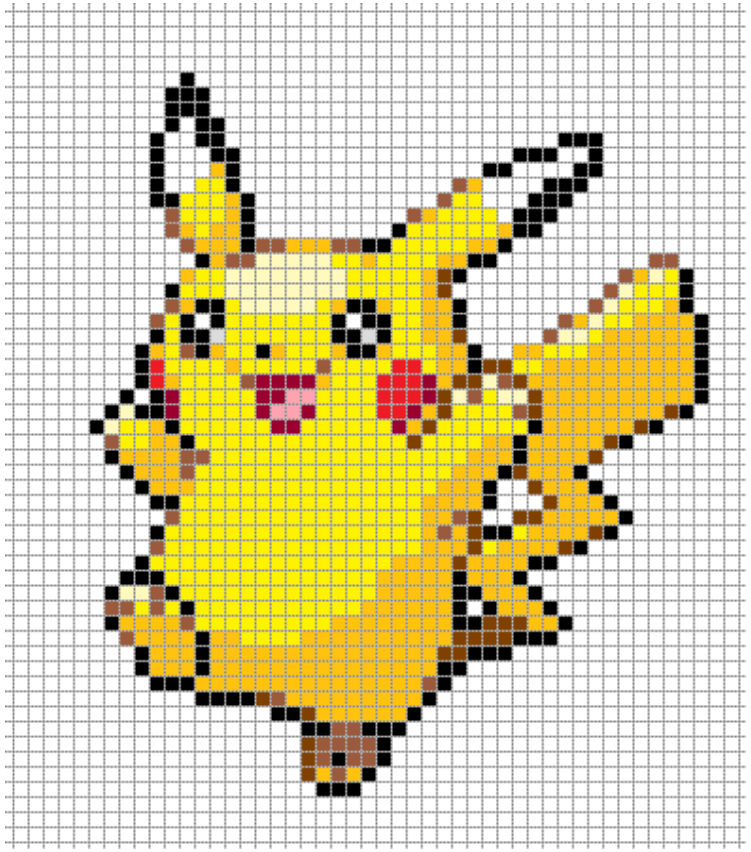
Graphics

Pixels

The image shown by your computer screen isn't continuous, like a real-world object. Instead, it is a two-dimensional grid of **pixels**.

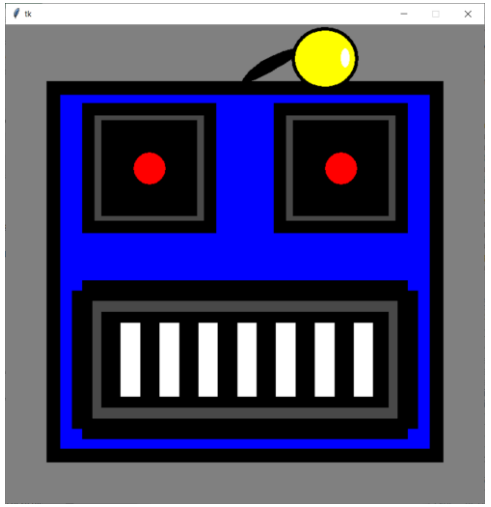
A pixel is a dot on the screen that has a specific color. Pixels tend to be very small, so we don't notice them most of the time.

Fun fact: you can change how many pixels are used to form the images on your screen! If you go to your computer's Display settings, you can tell it how many pixels tall and wide the screen should be.



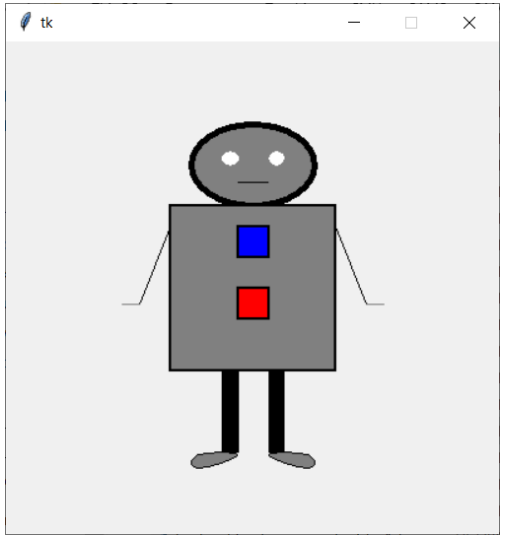
Drawing with code

There are many ways to approach making images to code. We're going to use an approach that draws **basic shapes** based on pixel coordinates. By combining many different shapes, we can make more advanced images!



We'll start today by drawing rectangles and ovals; on Thursday we'll add in text, lines, and polygons.

Here are two pictures your classmates in the CS track made last week!



Tkinter Canvas

In Python, we can draw graphics on the screen using many different modules. We'll use Tkinter in class because it's built-in, but there are other options for outside of class (turtle, pygame, Panda3D...)

Tkinter creates a new window on the screen and puts a **canvas** into that window. We'll call **built-in functions** on that canvas in order to draw on it.

NOTE: Tkinter documentation can be found at http://infohost.nmt.edu/tcc/help/pubs/tkinter/web/canvas.html

Tkinter starter code

We'll use the code on the right to set up the window and canvas when we want to draw graphics. It's okay to not understand what this code does for now.

You can write your own code where the comment says # Put your code here!

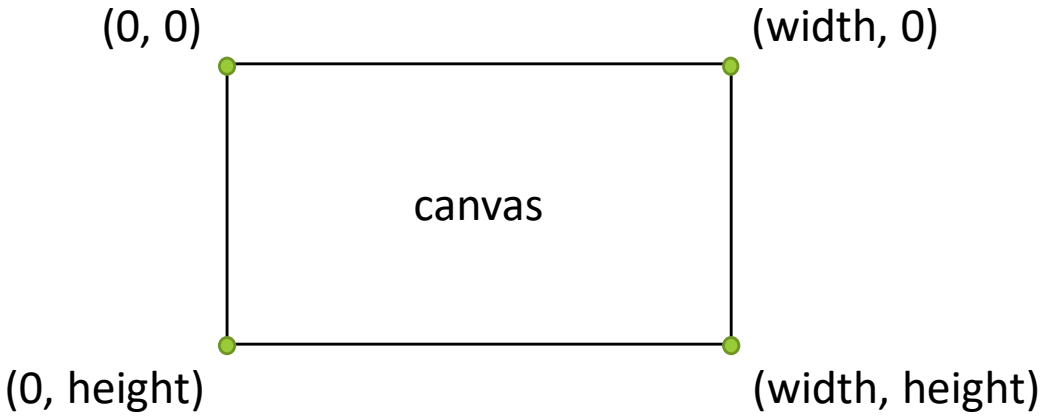
Every time we run this code, it will generate a new window. We can close the window by pressing the x in the top corner; that will let the code move on to whatever we've written next.

```
from tkinter import *


# Don't worry about what the next four lines of code do
root = Tk()
canvas = Canvas(root, width=400, height=400)
canvas.configure(bd=0, highlightthickness=0)
canvas.pack()



# Put your code here!



root.mainloop() # this tells the window to stay open until we close it
```

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.



Built-in Canvas Functions

In order to draw on the canvas, we'll need to call **functions** that will change how it looks.

A function is Python's way of remembering how to do an action, or a series of actions. The **print** command from last week is a built-in function.

We can call a function on a number of **parameters**, which then give the function information about what to do. These parameters are just values, like the numbers we used last time.

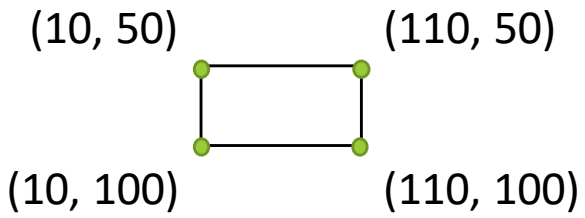
To call a function on the canvas, we'll use the following syntax:

```
canvas.<function_name>(<parameters>)
```

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 top-left corner, and the x and y coordinates of the bottom-right corner. The rectangle will then be drawn between those two points.

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



Exercise 1: Draw a rectangle

Go to the schedule page and download the starter file for today's lecture. You'll write exercise code under the comment with the exercise's number.

Exercise 1: write a line of code that draws a 100px x 200px rectangle from the coordinate (100, 125) to the coordinate (200, 325).

Note- every time you close the graphics window, it will move on to the next exercise. This is normal and okay.

Placing Shapes

It can be difficult sometimes to find the right coordinates for putting a shape in exactly the correct location. There are a few techniques that can make finding coordinates easier, though.

1) Try drawing images on paper before writing the code. Divide the paper up into segments to figure out approximately where the drawn shapes are located, then use those coordinates.

2) Work from reference points in the shapes. If you know where the corner of a shape should be, determine the opposing corner using the size. If you know the center and the size, subtract/add half the size to get the left/right sides.

3) Guess and check! Try adding a single shape to the image, then run the code to see where it shows up. You can then adjust the coordinates to move it or change the size as needed.

Rectangle Parameters

We can also add many **optional parameters** to the rectangle function to change the rectangle's appearance. An optional parameter is set up with the syntax:

```
canvas.create_rectangle(<x1>, <y1>, <x2>, <y2>, <parameter_name>=<value>)
```

For rectangles, we'll use two different kinds of optional parameters: **fill** and **outline**. On any given create_rectangle call, we can use one, both, or neither of this parameters. Here are all the color names we can use: http://www.science.smith.edu/dftwiki/index.php/Color_Charts_for_TKinter

```
canvas.create_rectangle(10, 50, 110, 100, fill="yellow") # makes rectangle yellow
```

```
canvas.create_rectangle(10, 50, 110, 100, outline="red") # makes border red
```

Exercise 2: Draw a colored square

Exercise 2: write a line of code that draws a square in the middle of the 400px x 400px screen that is **yellow** with a **red** outline. The square can be whatever size you want, but should be **centered** in the screen.

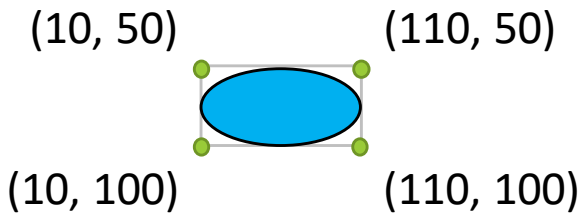
Hint: to center the rectangle, note that the middle point of the screen is (200, 200). You should set your coordinates so that left and right are equally spaced around 200, as are top and bottom.

Note: you'll need to close the window from the first exercise to check your work on the second.

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**. create_oval also has the same optional parameters as create_rectangle.

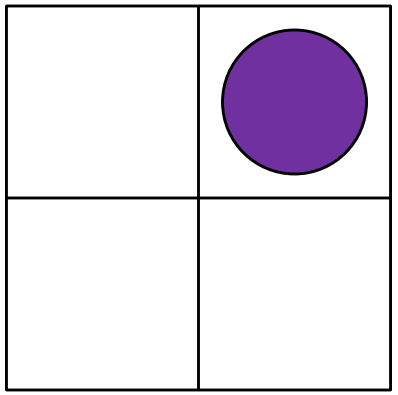
```
canvas.create_oval(10, 50, 110, 100, fill="blue")
```



Exercise 3: Draw a top-right circle

Exercise 3: write a line of code that draws a purple circle in the **top-right corner** of the 400px x 400px window. The circle can be whatever size you want, but should be **centered** in that top-right quadrant.

This is demonstrated below (you don't have to draw the square or lines):



Drawing multiple shapes

If we draw more than one shape, the shapes can overlap! Shapes which are drawn **later** are drawn on top.

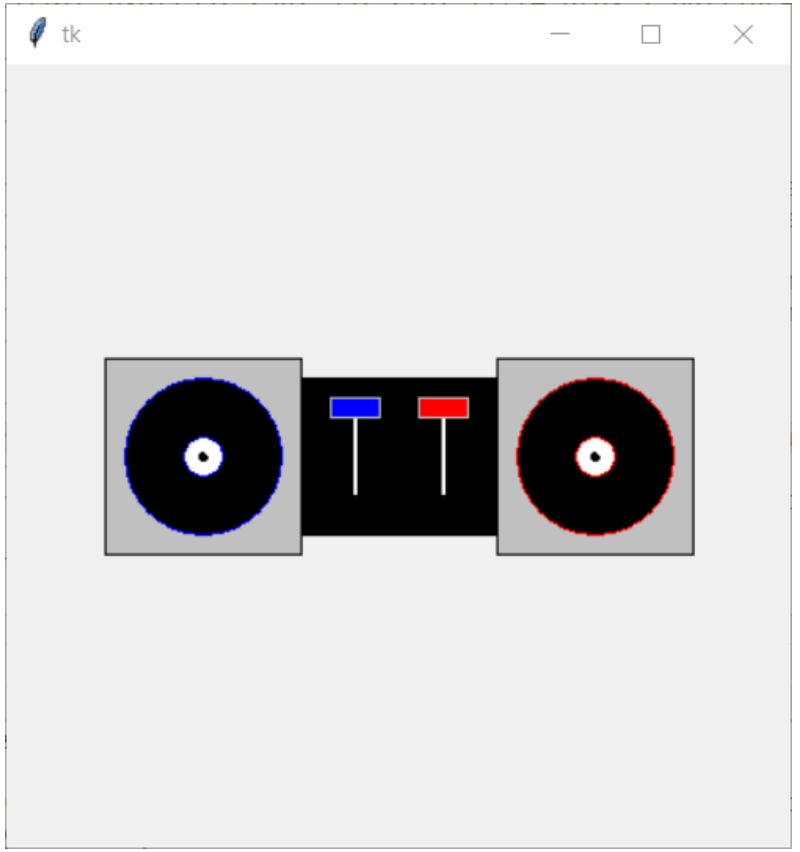
```
canvas.create_rectangle(  0,   0, 150, 150, fill="yellow")
canvas.create_rectangle(100,  50, 250, 100, fill="orange")
canvas.create_rectangle( 50, 100, 150, 200, fill="green", outline="red")
canvas.create_rectangle(125,  25, 175, 190, outline="purple")
```

Exercise 4: Draw a picture!

Exercise 4: using draw_rectangle and draw_oval as many times as needed, draw the picture to the right on the canvas.

Your image doesn't need to be pixel-perfect the same, but should have all the same components to get full credit.

Suggestion: this has a lot of parts! Start by drawing just the left side of the image. If you can get that working, you can mimic the code to create the right side.



Today's Learning Goals

Recognize how a computer makes images from pixels

Use code to draw rectangles and ovals in the window