Introduction

This document contains most of the Stanford CS106A course (Code in Place: 2020). The exceptions are assignment submition and validation, weekly zoom meetings, diagnostics and the online IDE (only necessary if you encounter trouble installing PyCharm).

The course schedule might not be as applicable now as it was when the course was live, but I believe there's plenty of research to back up the claim that most people learn better if they give themselves a day off in between learning a new skill. Don't try to rush it. That's what I used to do when I tried learning programming a couple years back, and it didn't stick. It wasn't untill I took this course that I finally really learned how to code, and its partly because of sticking to the schedule. However, if you do decide to go at it full-time, then instead of rushing it in order to get to the final project, try to exceed expectations on every assignment. That way you make sure to not miss anything important.

Every lecture is broken down into seperate youtube videos covering a sub-topic of the lecture's topic. You'll find links to all these youtube videos under each lecture. You will also find a link to the slides that was used in the videos, for each lecture.

Every assignment has a short description and an assignment handout. They also come with some starte code with which you can build your solution on.
Some assignments might have additional material. I suggest you familiarize yourself with this whenever its mentioned.

One material which is not mentioned in the assignments is the «Graphics reference». I've added this as a seperate PDF file. This will come in handy whenever you're doing any kind of graphical programming in python.

There might be some dates and other kinds of references to the course at the time it was live, which is no longer relevant. I hope it won't cause any confusion. Just keep in mind that this is the raw information from spring 2020, and I have not gone through and edited it to fit the format I'm now sharing this in.

Information on how to set up your coding environment on your computer is in assignment 1 (which in turn refers to two of the three additional PDF's).

Oh, and one more thing; **don't skip Karel**.

And most importantly; **have fun!**

Stanford CS106A course (Code in Place: 2020)

Contents

Course schedule

Week	**Monday**	**Wednesday**	**Friday**
1	Lecture 1	Lecture 2	Lecture 3
2	Lecture 4	Lecture 5 Assignment 1 due	Lecture 6
3	Lecture 7	Lecture 8	Lecture 9 Assignment 2 due
4		Lecture 10	Lecture 11
5	Lecture 12	Lecture 13 Assignment 3 due	Lecture 14
6	Final project	Final project	Final project due

Lectures

Lecture 1 - Welcome to Code in Place

Lecture1 – Welcome: https://www.youtube.com/watch?v=dxZFXJhZPvU&feature=emb_logo

Lecture1 - General Info:
https://www.youtube.com/watch?v=ukpUVAhdo94&feature=emb_logo

Lecture1 – Karel: https://www.youtube.com/watch?v=LpxjnuQwTg4&feature=emb_logo

Slides: https://codeinplace2020.github.io/faqs/1-Welcome.pdf

Lecture 2 - Control Flow in Karel

Lecture2 – Recap: https://www.youtube.com/watch?v=xAQlbo82EuU&feature=emb_logo

Lecture2 - For Loops: https://www.youtube.com/watch?v=yVmGFatf-Y8&feature=emb_logo

Lecture2 - While Loops:
https://www.youtube.com/watch?time_continue=1&v=S5y2u7VITMo&feature=emb_logo

Lecture2 - If Else: https://www.youtube.com/watch?v=ACkcPIB5SZs&feature=emb_logo

Lecture2 - Steeple Chase:
https://www.youtube.com/watch?v=nxu8NBAv2pM&feature=emb_logo

Slides: https://codeinplace2020.github.io/faqs/2-ControlFlow.pdf

Lecture code: https://codeinplace2020.github.io/faqs/Lecture2.zip

Lecture 3 – Decomposition

Lecture3 – Recap: https://www.youtube.com/watch?v=YFWUzglTrBQ&feature=emb_logo

Lecture3 – Morning: https://www.youtube.com/watch?v=Cz-wnRvlAMI&feature=emb_logo

Lecture3 – Mountain:
https://www.youtube.com/watch?v=ecqDCBm8tkY&feature=emb_logo

Lecture3 – Rhoomba: https://www.youtube.com/watch?v=JIQr_gtAWrc&feature=emb_logo

Lecture3 – WordSearch:
https://www.youtube.com/watch?v=62RtoSXfitU&feature=emb_logo

Slides: https://codeinplace2020.github.io/faqs/3-Decomposition.pdf

Lecture code: https://codeinplace2020.github.io/faqs/Lecture3.zip

Lecture 4 - Variables in Python

Lecture4 – Recap: https://www.youtube.com/watch?v=pkh2gDQ8tjM&feature=emb_logo

Lecture4 – HelloWorld:
https://www.youtube.com/watch?v=wEbmXvfl8TM&feature=emb_logo

Lecture4 - Add2Numbers:
https://www.youtube.com/watch?v=oUuIMt5KmyQ&feature=emb_logo

Slides: https://codeinplace2020.github.io/faqs/4-IntroPython.pdf

Lecture code: https://codeinplace2020.github.io/faqs/Lecture4.zip

Lecture 5 – Expressions

Lecture5 – Recap: https://www.youtube.com/watch?v=YwePpeJn828&feature=emb_logo

Lecture5 – Expressions:
https://www.youtube.com/watch?v=iTBsRFnaoJ0&feature=emb_logo

Lecture5 – Constants: https://www.youtube.com/watch?v=sAo9IdC223s&feature=emb_logo

Lecture5 - Math Library:
https://www.youtube.com/watch?v=H90Ud28sedo&feature=emb_logo

Lecture5 - Random Numbers:
https://www.youtube.com/watch?v=SQ2_cDLgrHI&feature=emb_logo

Lecture5 - Dice Simulator:
https://www.youtube.com/watch?v=_rMzEF0v6UI&feature=emb_logo

Slides: https://codeinplace2020.github.io/faqs/5-Expressions.pdf

Lecture code: https://codeinplace2020.github.io/faqs/Lecture5.zip

Lecture 6 - Control Flow Python

Lecture6 – Recap: https://www.youtube.com/watch?v=60AMFkbGZGY&feature=emb_logo

Lecture6 – Conditions:
https://www.youtube.com/watch?v=c6CZIQ3UFZE&feature=emb_logo

Lecture6 - Guess Num and Sentinal Sum:
https://www.youtube.com/watch?v=Y_IWN4OxhlM&feature=emb_logo

Lecture6 – Booleans: https://www.youtube.com/watch?v=Y7evkU5j7TY&feature=emb_logo

Lecture6 - For Loops:
https://www.youtube.com/watch?v=5BTJ4gVXaFQ&feature=emb_logo

Lecture6 - GameShow Teaser:
https://www.youtube.com/watch?v=mVoerPV6YLY&feature=emb_logo

Slides: https://codeinplace2020.github.io/faqs/6-ControlFlowRevisited.pdf

Lecture code: https://codeinplace2020.github.io/faqs/Lecture6.zip

Lecture 7 - Functions Revisited

Lecture7 - Recap with GameShow:
https://www.youtube.com/watch?v=wY68LUvnJ04&feature=emb_logo

Lecture7 - Functions are like Toasters:
https://www.youtube.com/watch?v=hmcuptr9WBE&feature=emb_logo

Lecture7 - Anatomy of a Function:
https://www.youtube.com/watch?v=lZ8DGnIRsng&feature=emb_logo

Lecture7 - Many Examples: https://www.youtube.com/watch?v=CS-BMynY5ko&feature=emb_logo

Lecture7 – IO: https://www.youtube.com/watch?v=8vXvRwj8fos&feature=emb_logo

Slides: https://codeinplace2020.github.io/faqs/7-Functions.pdf

Lecture code: https://codeinplace2020.github.io/faqs/Lecture7.zip

Lecture 8 - Functions More Practice

Lecture8 – Recap: https://www.youtube.com/watch?v=vMy48Q6aPk0&feature=emb_logo

Lecture8 – Factorial: https://www.youtube.com/watch?v=kZpiuJ1r3rg&feature=emb_logo

Lecture8 – DocTests: https://www.youtube.com/watch?v=rXtLAPxeSgI&feature=emb_logo

Lecture8 - Passing Primitives:
https://www.youtube.com/watch?v=vmzFKkyjo4o&feature=emb_logo

Lecture8 – Calendar: https://www.youtube.com/watch?v=8PCQndHgkPE&feature=emb_logo

Slides: https://codeinplace2020.github.io/faqs/8-Parameters.pdf

Lecture code: https://codeinplace2020.github.io/faqs/Lecture8.zip

Lecture 9 – Images

Lecture9 – Recap: https://www.youtube.com/watch?v=gjT_okH7HD8&feature=emb_logo

Lecture9 - Images in Python:
https://www.youtube.com/watch?v=iC82OUseeeY&feature=emb_logo

Lecture9 - First Examples:
https://www.youtube.com/watch?v=aeGbb8wC56g&feature=emb_logo

Lecture9 – GreenScreen:
https://www.youtube.com/watch?v=pAG9rAqA4N4&feature=emb_logo

Lecture9 – Mirrored: https://www.youtube.com/watch?v=x0PpSbK4k_s&feature=emb_logo

Lecture9 - Nested For vs For Each Pixel:
https://www.youtube.com/watch?v=DhohL7AOzsw&feature=emb_logo

Slides: https://codeinplace2020.github.io/faqs/9-Images.pdf

Lecture code: https://codeinplace2020.github.io/faqs/Lecture9.zip

Lecture 10 – Graphics

Lecture10 – Recap: https://www.youtube.com/watch?v=h9nnz_QSzZA&feature=emb_logo

Lecture10 - Blue Rect:
https://www.youtube.com/watch?v=3RMrC1wWyFE&feature=emb_logo

Lecture10 - Programming is Awesome:
https://www.youtube.com/watch?v=SfiEWn9RCXM&feature=emb_logo

Lecture10 – Checkers: https://www.youtube.com/watch?v=Y9Qi-6TWwpM&feature=emb_logo

Slides: https://codeinplace2020.github.io/faqs/10-Graphics.pdf

Lecture code: https://codeinplace2020.github.io/faqs/Lecture10.zip

Lecture 11 – Animations

Lecture11 – Recap: https://www.youtube.com/watch?v=B8-lPPUU7eY&feature=emb_logo

Lecture11 - Animation Loop:
https://www.youtube.com/watch?v=jz02xtVaBo8&feature=emb_logo

Lecture11 - Move to Center:
https://www.youtube.com/watch?v=frTXMIWSuq0&feature=emb_logo

Lecture11 - Bouncing Ball:
https://www.youtube.com/watch?v=qjsxi3UzoA0&feature=emb_logo

Lecture11 – References:
https://www.youtube.com/watch?v=g0G4S_woMRA&feature=emb_logo

Lecture11 – Pong: https://www.youtube.com/watch?v=XcvbczJF6CU&feature=emb_logo

Slides: https://codeinplace2020.github.io/faqs/11-Animations.pdf

Lecture code: https://codeinplace2020.github.io/faqs/Lecture11.zip

Lecture 12 – Lists

Lecture12 - Recap with Console:
https://www.youtube.com/watch?v=QioUAmUAIgE&feature=emb_logo

Lecture12 – None: https://www.youtube.com/watch?v=A-NrRd9GyYg&feature=emb_logo

Lecture12 – Lists: https://www.youtube.com/watch?v=vhknJZ-2Bzg&feature=emb_logo

Lecture12 - Lists as Parameters:
https://www.youtube.com/watch?v=w4beNu04CMs&feature=emb_logo

Lecture12 – AverageScores: https://www.youtube.com/watch?v=L_TyVmOQq-I&feature=emb_logo

Slides: https://codeinplace2020.github.io/faqs/12-Lists.pdf

Lecture code: https://codeinplace2020.github.io/faqs/Lecture12.zip

Lecture 13 - Text Procession

Lecture13 - Hook and Recap:

https://www.youtube.com/watch?v=BQQVnsE2DZI&feature=emb_logo

Lecture13 Working with Strings:

https://www.youtube.com/watch?v=xRhjkyJHFbE&feature=emb_logo

Lecture13 Helpful String Functions:

https://www.youtube.com/watch?v=MOhsuyHr6fU&feature=emb_logo

Lecture13 - Just Number and DNA to mRNA:

https://www.youtube.com/watch?v=fNChmzR6rVs&feature=emb_logo

Lecture13 Characters:

https://www.youtube.com/watch?v=SnJYJHmNW7s&feature=emb_logo

Lecture13 – Immutable: https://www.youtube.com/watch?v=-cIzBBzTnK8&feature=emb_logo

Lecture13 - ReverseString and Palindrome:

https://www.youtube.com/watch?v=PB4tJZHdcAk&feature=emb_logo

Lecture13 – FakeMedicine:

https://www.youtube.com/watch?v=BbE4dnoAmXs&feature=emb_logo

Slides: https://codeinplace2020.github.io/faqs/13-TextProcessing.pdf

Lecture code: https://codeinplace2020.github.io/faqs/Lecture13.zip

Lecture 14 – Dictionaries

Lecture14 - Recap with Files:

https://www.youtube.com/watch?v=GyexyR1qwZE&feature=emb_logo

Lecture14 - What are Dictionaries:

https://www.youtube.com/watch?v=iW6PlKk5XZk&feature=emb_logo

Lecture14 - Mutability and Dictionaries:

https://www.youtube.com/watch?v=vN9qV2hHbGk&feature=emb_logo

Lecture14 – Dictionapalooza:

https://www.youtube.com/watch?v=IUTaANNVS_w&feature=emb_logo

Lecture14 – CountWords:

https://www.youtube.com/watch?v=Pvcvy0W38T8&feature=emb_logo

Lecture14 PhoneBook:

https://www.youtube.com/watch?v=jx8u6dFUxpY&feature=emb_logo

Slides: https://codeinplace2020.github.io/faqs/14-Dictionaries.pdf

Lecture code: https://codeinplace2020.github.io/faqs/Lecture14.zip

Assignments

Assignment 1: Karel

Getting Started

There is a starter project including all of these problems that you can access using a link below. Once you have the starter code set up, edit the program files so that the assignment actually does what it's supposed to do (see the assignment handout), which will involve a cycle of coding, testing, and debugging until everything works. The final step is to submit your assignment. You should write the code for your solution on your own.

Note: You may only use concepts covered in the Karel coursereader to solve these problems. In particular, you may not use Python concepts you may have learned previously such as variables, parameters,return, break, etc. If you have any questions about what is ok/not ok to use, please feel free to ask on Ed.

Note: Karel can be programmed either on your computer using an application like PyCharm, or it can be programmed in browser like we did for assignment 0. We would like everyone to try and install PyCharm, it will help down the road. We will make the browser-based option available on Wednesday, April 15th, 11:50pm, Anywhere on earth. Happy coding!

Note: If you wrote a Karel program as an extension in ExtensionKarel.py and made new Karel Worlds to run the program in, please submit the .w files alongside your assignment code.

Assignment handout: https://codeinplace2020.github.io/faqs/Assignment1.pdf

Starter code: https://compedu.stanford.edu/codeinplace/starterCode/Assignment1.zip

Get PyCharm: See attached PDF

Using Karel in PyCharm: See attached PDF

Karel reader: https://compedu.stanford.edu/karel-reader/docs/python/en/intro.html

Assignment 2: KhansoleAcademy

Getting Started

There is a starter project including all of these problems that you can access using a link below. Once you have the starter code set up, edit the program files so that the assignment actually does what it's supposed to do (see the assignment handout), which will involve a cycle of coding, testing, and debugging until everything works. The final step is to submit your assignment. You should write the code for your solution on your own.

Assignment handout: https://codeinplace2020.github.io/faqs/Assignment2.pdf

Starter code: https://codeinplace2020.github.io/faqs/Assignment2.zip

Python reader: https://codeinplace2020.github.io/pythonreader/en/intro/

Assignment 3: Images

Getting Started

There is a starter project including all of these problems that you can access using a link below. Once you have the starter code set up, edit the program files so that the assignment actually does what it's supposed to do (see the assignment handout), which will involve a cycle of coding, testing, and debugging until everything works. The final step is to submit your assignment. You should write the code for your solution on your own.

Assignment handout: https://codeinplace2020.github.io/faqs/Assignment3.pdf

Starter code: https://codeinplace2020.github.io/faqs/Assignment3.zip

Images reference: https://codeinplace2020.github.io/faqs/imageReference.pdf

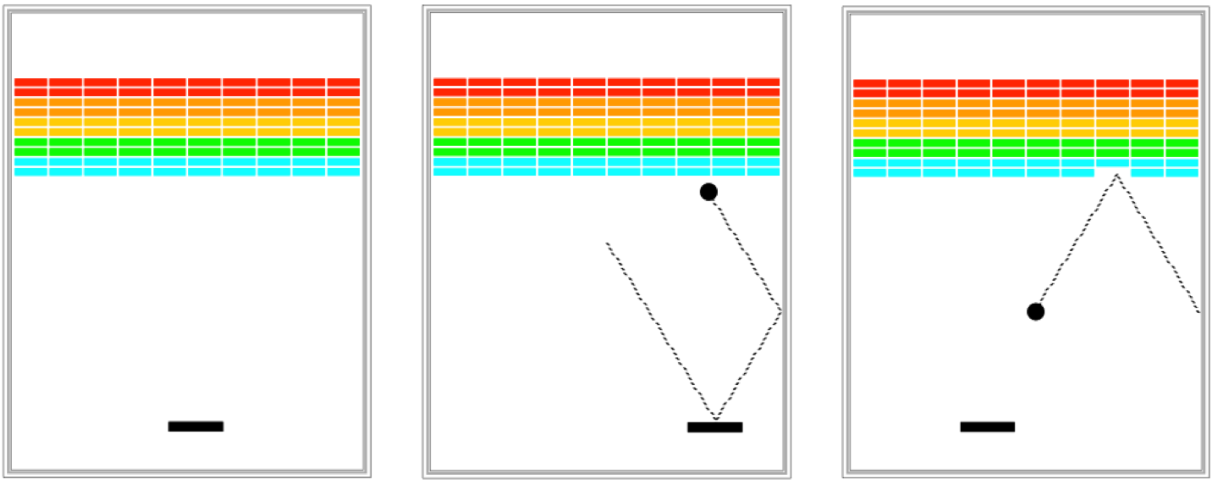
Final project

Final project starter code: https://codeinplace2020.github.io/faqs/BrickBreaker.zip

Your job in this assignment is to write the classic arcade game of BrickBreaker, which was invented by Steve Wozniak before he founded Apple with Steve Jobs (moment of silence). It is a large assignment, but entirely manageable as long as you break the problem up into pieces.

The BrickBreaker Game

In Breakout, the initial configuration of the world appears as shown on the right. The colored rectangles in the top part of the screen are bricks, and the slightly larger rectangle at the bottom is the paddle. The paddle is in a fixed position in the vertical dimension, but moves back and forth across the screen along with the mouse until it reaches the edge of its space.



A complete game consists of three turns. On each turn, a ball is launched from the center of the window toward the bottom of the screen at a random angle. That ball bounces off the paddle and the walls of the world, in accordance with the physical principle generally expressed as "the angle of incidence equals the angle of reflection" (which turns out to be very easy to implement as discussed later in this handout). Thus, after two bounces--one off the paddle and one off the right wall--the ball might have the trajectory shown in the second diagram. (Note that the dotted line is there to show the ball's path and won't appear on the screen.)

As you can see from the second diagram, the ball is about to collide with one of the bricks on the bottom row. When that happens, the ball bounces just as it does on any other collision, but

the brick disappears. The third diagram shows what the game looks like after that collision and after the player has moved the paddle to put it in line with the oncoming ball.

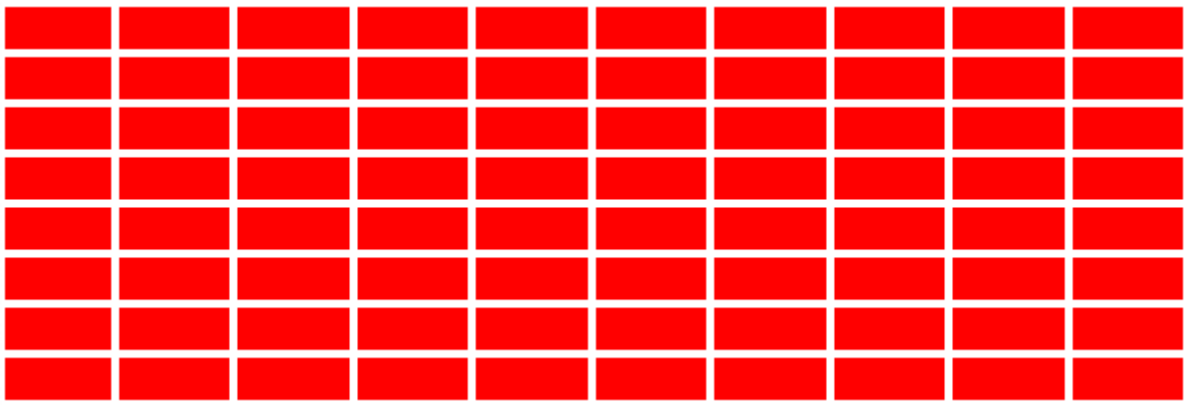
The play on a turn continues in this way until one of two conditions occurs:

1. The ball hits the lower wall, which means that the player must have missed it with the paddle. In this case, the turn ends and the next ball is served if the player has any turns left. If not, the game ends in a loss for the player.
2. The last brick is eliminated. In this case, the player wins, and the game ends immediately.

Success in this assignment will depend on breaking up the problem into manageable pieces and getting each one working before you move on to the next. The next few sections describe a reasonable staged approach to the problem.

1. Create bricks

Before you start playing the game, you have to set up the various pieces. An important part of the setup consists of creating the rows of bricks at the top of the game, which look like this:



The number, dimensions, and spacing of the bricks are specified using constants in the starter file, as is the distance from the top of the window to the first line of bricks. Initially you can make all the bricks 'red'. If you feel like adding some color you can color every two rows in the following rainbow-like sequence: 'red', 'orange', 'yellow', 'green', 'cyan'.

2. Add a bouncing ball

At one level, creating the ball is easy, given that it's just a filled oval. The interesting part lies in getting it to move and bounce appropriately. To start, create a ball and put it in the center of the window. As you do so, keep in mind that the coordinates of the create_oval function do not specify the location of the center of the ball but rather its upper left corner and bottom right corner.

The program needs to keep track of the how much to move the ball each animation heart-beat, which consists of two separate components, which you will presumably declare as variables like this:

```
change_x = 10
change_y = 10
```

The "change" components represent the change in position that occurs on each animation time step. Initially, the ball should be heading downward, and you might try a starting velocity of +10.0 for chnage_x and change_y.

Recall that there are two ways to move an object in the python graphics library:

```
# Move the object to a specific new_x, new_y
# Note this doesn't work in some old versions of tkinter
canvas.moveto(object, new_x, new_y)


# Increase the x coordinate by change_x and the y coordinate by change_y
canvas.move(object, change_x, change_y)
```

Once you've created your ball and the change variables, your next challenge is to get the ball to bounce around the world, ignoring entirely the bricks. This will require that you program an "animation loop" where you move the ball and then pause. Then you can consider how to make the ball bounce. To do so, you need to check to see if the coordinates of the ball have gone beyond the boundary. Thus, to see if the ball has bounced off the right wall, you need to see whether the coordinate of the right edge of the ball has become greater than the width of the window; the other three directions are treated similarly. For now, have the ball bounce off the bottom wall so that you can watch it make its path around the world.

You might find these helper methods useful. They allow you to get the left and top coordinates of a graphical object:

```
def get_top_y(canvas, object):
    return canvas.coords(object)[1]

def get_left_x(canvas, object):
    return canvas.coords(object)[0]
```

Computing what happens after a bounce is simple. If a ball bounces off the top or bottom wall, all you need to do is reverse the sign of change_y. Symmetrically, bounces off the side walls simply reverse the sign of change_x.

3. Add the paddle

The next step is to create the paddle. There is only one paddle, which is a filled rectangle. You even know its position relative to the bottom of the window.

The challenge in creating the paddle is to make it track the mouse. Here, however, you only have to pay attention to the x coordinate of the mouse because the y position of the paddle is fixed.

Each time through the animation loop, ask for the location of the mouse and move the rectangle representing the paddle. To get the location of the mouse you can use the canvas function winfo_pointerx

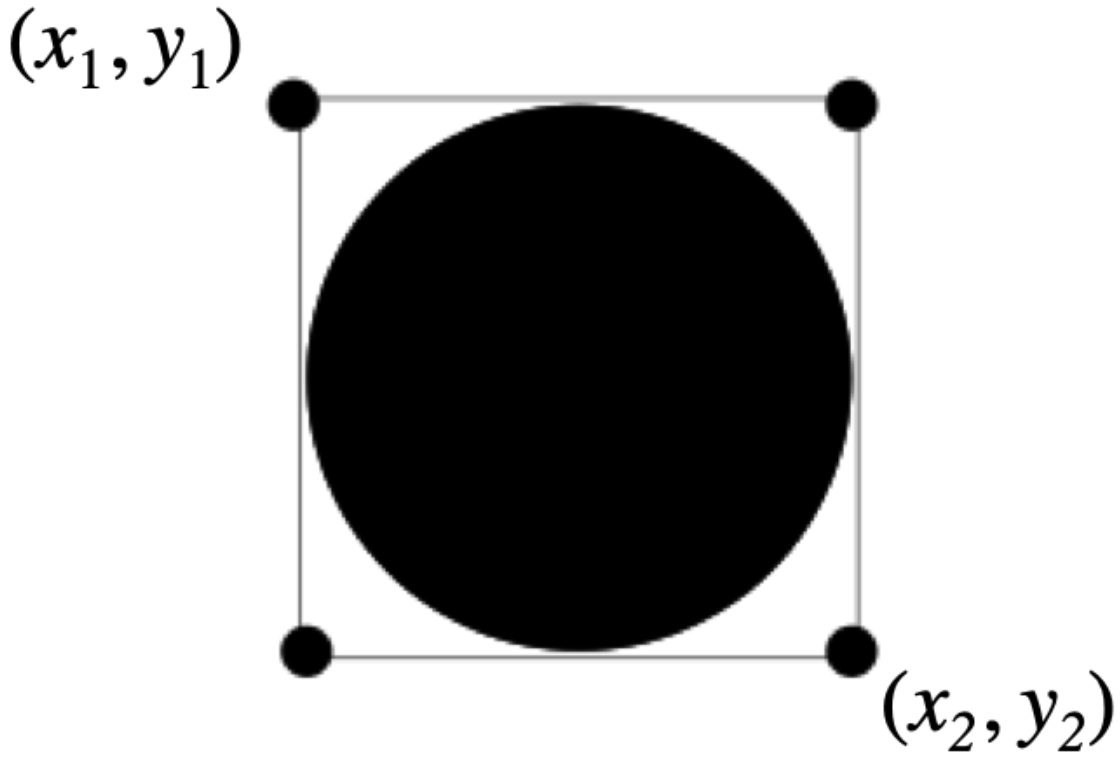
```
mouse_x = canvas.winfo_pointerx()
```

4. Check for Collisions

Now comes the interesting part. In order to make BrickBreaker into a real game, you have to be able to tell whether the ball is colliding with another object in the window. As scientists often do, it helps to begin by making a simplifying assumption and then relaxing that assumption later. Suppose the ball were a single point rather than a circle. In that case, how could you tell whether it had collided with another object?

There is a canvs function: canvas.find_overlapping(x_1, y_1, x_2, y_2) which returns a list of all objects that are overlapping the rectangle defined by the four coordinates.

The easiest thing to do, which is in fact typical of real computer games, is to check a larger bounding box for collision. Look for any objects in this rectangle:



To get a list of objects that are overlapping the ball you can use code as follows. Note that the ball itself will be in the colliding_list:

```
# this graphics method gets the location of the ball as a list
ball_coords = canvas.coords(ball)
# the list has four elements:
x_1 = ball_coords[0]
y_1 = ball_coords[1]
x_2 = ball_coords[2]
y_2 = ball_coords[3]
# we can then get a list of all objects in that area
colliding_list = canvas.find_overlapping(x_1, y_1, x_2, y_2)
```

If you get all overlapping objects in that rectangle, you will be returned a list. In that list will be the ball, as well as any object the ball is currently colliding with. To test if a colliding object is the ball, you can simply check if the element is == to your ball variable.

If the ball collides with the paddle you should make the ball bounce. If the ball collides with bricks you should remove all bricks and flip the direction of the change_y variable.

5. Extra Touches

If you've gotten to here, you've done all the hard parts. There are, however, a few more details you could take into account if you have time.

1. Can you make your bricks rainbow colored like in the first image?
2. Take care of the case when the ball hits the bottom wall. In the prototype you've been building, the ball just bounces off this wall like all the others, but that makes the game pretty hard to lose. Modify your loop structure so that it tests for hitting the bottom wall as one of its terminating conditions.
3. Make your game play three turns!
4. Make it so that your ball bounces differently, depending on where it hits the paddle. If it hits the left of the paddle it bounces more to the left. If it hits the right of the paddle it bounces more to the right.
5. Check for the other terminating condition, which is hitting the last brick. How do you know when you've done so? Although there are other ways to do it, one of the easiest is to have your program keep track of the number of bricks remaining. Every time you hit one, subtract one from that counter. When the count reaches zero, you must be done. It would be nice to give the player a little feedback that at least indicates whether the game was won or lost.
6. Test your program to see that it works. If you think everything is working, here is something to try: Just before the ball is going to pass the paddle level, move the paddle quickly so that the paddle collides with the ball rather than vice-versa. Does everything still work, or does your ball seem to get "glued" to the paddle? This error occurs because the ball collides with the paddle, changes direction, and then collides with the paddle again before escaping. How can you fix this bug?

Extensions

If you have the basic play working well, then this would be a great opportunity to go above and beyond. We encourage you to use your imagination to come up with fun ideas.