Instructions

These exercises are designed to help cement the concepts from lecture and give you an opportunity to apply them yourself. You are welcome to discuss and share ideas with each other, but should write the answers and code yourself.
Some questions ask you to complete a checkoff, which means you need to come to office hours at this link and talk to a staff member. Staff members may complete checkoffs for multiple students at one time, meaning you may not answer every question yourself but instead should listen to what your peers have to say and ask questions! Collaboration is an important part of computer science.

1 Tasks

1.1 Blackjack - blackjack.py

Blackjack is a popular game (feel free to read up more on the game on Wikipedia: https://en.wikipedia.org/wiki/Blackjack). The goal of the game is to get as close as possible to a score of 21 without going over. Below is a fragment of code. Given the inputs in the table below, what does it output?

Open **blackjack.py** and use comments to fill in the table. Do this without running the code. (It is an important skill to be able to understand what a piece of code does without running it.)

```python
total = card1 + card2
if total == 21:
    print("blackjack!")
elif total >= 17:
    print("stand")
elif total >= 12:
    if dealer_card > 6:
        print("hit")
    else:
        print("stand")
elif total > 9 or (total == 9 and dealer_card <= 6):
    print("double, then hit")
else:
    print("hit")
```

Complete the table below, which is provided in **blackjack.py**:

card1	card2	dealer_card	output
3	8	4	
9	8	10	
11	10	10	
3	5	4	
5	8	8	

Checkoff #1—Reason through your answers in a checkoff. What would change if we changed $\geq$ and $\leq$ to just $>$ and $<$, respectively?

1.2 Translation - translator.py

Sometimes people have to talk to automated systems over the phone. They might ask you to read out some sort of number for identification – for example, your confirmation number, bank account number, or social security number. If the number is 1234, you might read it out as "one-two-three-four."

Now imagine you have to speak to a Spanish company, but you don't know Spanish! We are going to write a really simple translator that will help you translate English numbers to Spanish numbers.

Open **translator.py**. Your code should ask for one or more digits, spelled out in English. Then, it should print out the Spanish numbers. Please use a dictionary for this! Your program should work as follows:

```
Enter one (or more) digits spelled out in English: one five seven
The spanish translation is: uno cinco siete
```

Note: You can assume your user correctly inputs one (or more) English digits separated by a space. You don't have to handle other cases. (But of course, you can if you like!)

Checkoff #2 — Show off your working translator.py file to a course staff! Could we have done this problem using a system of lists? What are the advantages of using a dictionary data structure over a list? How can it make your code simpler or more complex?

1.3 Acronyms - acronyms.py

This problem will allow you to practice with the methods for strings and lists. An acronym is a word formed by taking the first letters of the words in a phrase and making a word from them. For example, RAM is an acronym for "random access memory." In this exercise, you are going to write a program called **acronyms.py**. This program should allow the user to type in a phrase, and the program should then output the acronym for that phrase. An example of how your program should work is as follows:

```
Enter a phrase: Laugh out loud
Your acronym is : LOL
```

Note: The acronym should be all uppercase, even if the words in the phrase are not capitalized!

1.4 Simple graphics - simple_graphics.py

Let's make a simple graphics program. Open **simple_graphics.py** in IDLE, but don't run it yet! You can find all the documentation for the graphics module in the graphics reference sheet on Canvas. We explain the code in **simple_graphics.py** below.

from graphics import *	This tells us that we're going to use graphics. Importing graphics gives us access to all sorts of graphics commands that someone else has made for us.
win = GraphWin('My Circle', 100, 100)	We're creating a new graphics window object, called **win** of type **GraphWin** (part of the graphics module). We're calling it **My Circle** and telling it to have dimensions of 100 x 100 pixels.
c = Circle(Point(50, 50), 10)	We're creating a circle object called **c** with a center point at (50,50) and with a radius of 10. Note that the center of the circle is a (unnamed) **Point** with coordinates (x, y) = (50,50).
c.setFill('red')	We're coloring the circle red. This can take most normal colors. See the documentation under Generating Colors for more information.
c.draw(win)	Now that we've decided on attributes for the circle (position, size, and color), we're actually drawing it on the screen.
win.getMouse() # pause for click in window **win.close()**	The window keeps open until you use the mouse to click inside the window or click the close window button.

Now run the code. A window with your image will pop up!

Feel free to try playing around with some of the values – see where things break, and see where you can change things. Then change the code to do the following things:

(a) Make your circle bigger - specifically, double the radius.

(b) Your circle is currently centered in your window. Change your code so that the circle is now located in the upper right hand corner of the window. Specifically, the center of the circle should be located in the middle of the upper-right quadrant of the window. (Hint: coordinates are counted from the top-left corner. You can also use a trial-and-error approach by changing the numbers for the circle until you get what you want!)

(c) Make the circle yellow.

(d) Make the outline of the circle purple. (Hint: you will need to look up how to do this in the online graphics documentation. Remember that all the functions for "Graphics Objects" can be used for any shapes!)

Note: If your program gets stuck, you can press "ctrl" and "C" together in the console to force it to interrupt (terminate).

2 Optional Challenge[1]:

(a) win.getMouse() pauses the graphic until you click. Make a new, different circle every time you click for 3 clicks. For example, you can click three times, and each time you click, a smaller circle will appear inside of the previous circle. You'll need to use a new variable for each circle.

(b) Now try making the new circles REALLY new. Randomly pick the new circle's coordinates! You can pick random numbers like so. Make sure to include the **import random** line of code at the top of your **simple_graphics.py** file.

```python
import random
from graphics import *

# Prints a random number between 0 and 100 (inclusive)
big_guess = random.randint(0, 100)
print(big_guess)
```

(c) Once you get the hang of making randomly positioned circles, try making their sizes and colors randomized too.

```python
# Prints one of the options, randomly chosen.
# options can be tuple, set, dictionary, or list -- whatever you want!
options = [25, 50, 75]
chosen = random.choice(options)
print(chosen)
```

Submitting your PSET

After you've finished your PSET and checkoff: Log into your Canvas account, find the post for Problem Set 3 in Assignments, and submit all of the files that you created or edited. After you turn in your assignment, you're all done!

[1]optional problems are provided for those who have further interests and want to explore more. You are not responsible for those questions, however.