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 Understanding Loops - loops.py

For the following fragments of code, create a new file called **loops.py** and write what the output would be using comments for each of the examples below. **Do this *without* running the code**.
Example 1:

```
# Example 1:
num1 = 10
while num1 > 3:
    print(num1)
    num1 = num1 - 1
```

Example 2:

```
# Example 2:
num2 = 10
while True:
    if num2 < 7:
        # break makes us quit the loop
        break
    print(num2)
    num2 = num2 - 1
```

Example 3:

```
# Example 3:
divisor = 2
for i in range(0, 10, 2):
    print(i//divisor)
```

You can use for loops to iterate through strings.
Example 4:

```
# Example 4:
my_string = "stressed"
for letter in my_string:
    print(letter)
```

Example 5:

```
# Example 5:
my_string = "stressed"
res = ''
for letter in my_string:
    res = letter + res
print(res)
```

Once you are done, you can run the code to check your answers.

> *Checkoff #1—Discuss how your predictions compared with the actual results. Consider the following questions:*
>
> 1. *What's the difference between a for loop and while loop?*
> 2. *What's the function of "break"?*

1.2 Exponentials - exponential.py

Imagine that the ** operator for numbers has now been removed from the Python programming language. Open **exponential.py** and write a program that prompts the user for a base and an exponent and uses a `for` loop to calculate `base**exp`.
We will assume the user types in a positive integer. (You don't need to handle other cases.)

> *Checkoff #2—Discuss your solutions with a member of the course staff. Consider the following questions:*
>
> 1. *Can a `for` loop always be replaced with a `while` loop?*
> 2. *Can a `while` loop always be replaced with a `for` loop?*

1.3 Card Counting in Blackjack (continued)

A review from pset 3: As you may know from the movie *21*, there is a technique to count cards while playing Blackjack that can help the player determine when they are likely to win or lose! Specifically, when counting cards, you keep track of a running total that goes up and down based on the cards you have seen. In this problem, we will finish a program that will count cards for the user.

(a) We need to fill in the missing code in **hotandcold.py**. This program will ask for cards from the user and keep track of the running total card counting score. First, write the code that is able to update the count. The count will increase by 1 if the current card is 2 through 6, decrease by 1 if it is 10 through Ace (10, Jack, Queen, King, or Ace), and stay the same if the card is 7 through 9. (Example: if my count was 3 and my current card was a Jack, the count would update to 2)

(b) **Knowing when to bet.** Now, using the code you just wrote, finish the program so that it uses the total card counting score as follows:

- It should tell the user when to bet big (if the running total is 5 or greater).
- It should tell the user to find a new table for blackjack (if the running total is -5 or less).
- It should keep asking the user for the next card if the running total is not too high or too low (if the running total is between -5 and 5). Test out your program to make sure it works correctly, and if it does, you are done!

1.4 Animations - animate.py

Loops allow us to make animations. By drawing the same object over and over again at different locations on the screen, we can give the illusion of movement. The `graphics` module's `move` command makes this really easy. `move(dx,dy)` takes two arguments `dx`, the number of pixels to shift in the x-direction, and `dy`, the number of pixels to shift in the y-direction. **animate.py** gives an example of the `move` command on a circle.

The `sleep(ns)` command in the `time` module pauses the program for `ns` seconds before continuing execution. Placing a `sleep` command before moving the circle slows down the execution so that you can actually see the circle move.

(a) Open **animate.py**. Let's make the circle slide across the screen. Add a loop around the sleep and move commands. (To avoid using infinite loops, you can specify a finite number of time steps for the circle to move, e.g., 3.)

(b) Cool! If we let the animation keep going, then the circle will move off the right edge of the screen. Add a variable `x` to keep track of the circle's position and a variable `dx` for its velocity. The circle initially starts at (100, 150), so by updating `x` each time it moves, we can track where the circle is going. When the circle reaches the edge of the screen, make it change direction. Do this for both directions so that the circle bounces back and forth inside the screen.

2 Optional Challenge[1]:

Expand your animate.py program by having the ball move up and down as well:

1. Add some movement in the y-direction so that the circle bounces around the entire window. Experiment with starting the circle at different locations or changing its velocity.
2. Count the number of times the ball hits the edge of the screen. Stop the animation once the ball has bounced 10 times.

Submitting your PSET

After you've finished your PSET and checkoff: Log into your Canvas account, find the post for Problem Set 4 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.