Here are several ideas you can work on to practice and develop your python skills. By the end of each you will also have a fully functional game you can play! I've taken these games from a book called "a Gamut of Games" written by Sid Sackson and a book called "Dice Games Properly Explained" by Reiner Knizia, two relatively famous board game designers. So hopefully they are actually fun. For all the games below I leave it up to you to decide how you want to handle user I/O. If you want to use pygame to make nice visuals that's fine, it's also fine if you just want to do everything with text in the command line, or somewhere in between. You must use python.

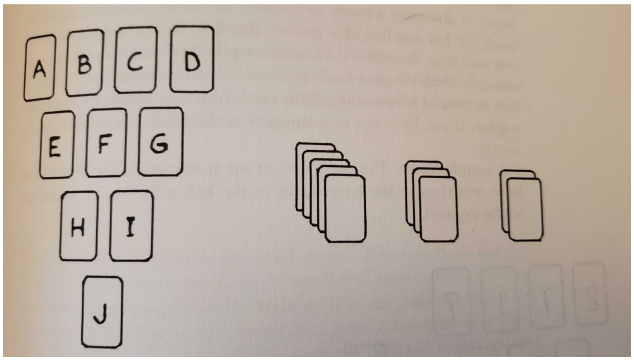
For an extra challenge, in addition to programming a playable version of a game, write a program that will automatically play the game. You can even work with a partner to code the game, then each write a program to play it and have them compete.

Finally, you can use your program to play the game many, many times and do some statistics on the results. Test the effect of the starting setup of the game on the probability of winning

Card Games

Because we have several card game options, consider how you implement them carefully. If you implement the first one correctly then you should be able to implement the others easily by reusing your code. This is a fundamental part of how to code smartly, and will come up frequently as you start coding for your research (e.g. you wrote an analysis for dataset A and now want to apply it to dataset B, or build it into a new analysis).

1. **Bowling solitaire**
    a. This is a game for one player that uses two copies of the numbers 10,9,8,7,6,5,4,3,2,1 (ace) from a deck of playing cards. The game is set up by shuffling the 20 cards, drawing 10 and arranging them in the shape of the 10 pins in bowling, an inverted pyramid. These are the "pin cards" The other 10 cards are then randomly grouped into a pile of 5, 3, and 2 cards. The top card of those piles are visible to the player. They are the "ball cards" See the diagram below for a visualization.



    b. The game is played over a series of rounds. Each round you may choose use one of the face-up ball cards to remove 1-3 pin cards.
        i. The pin cards you remove must be adjacent to each other (e.g. H-I, or E-F-H, or H-F-G).
        ii. They must also add up to a number with the same last digit as the ball card you selected (e.g. a 6 can get rid of a 6, or a 7-9, etc…).
        iii. The very first ball card you choose cannot remove A, B, C, or D, and it can also not remove F by itself.

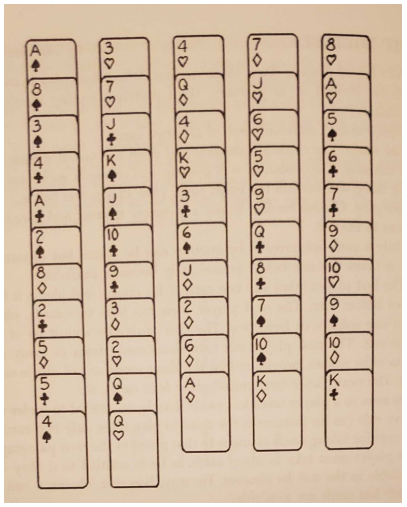
iv. After the first ball card, you may only remove cards if at least one of them was adjacent to an empty slot (e.g. if you removed H-E on turn 1, on turn 2 cannot remove C-G but you could remove F-C-G).

c. After you use a ball card, the next one in that pile becomes visible. You then repeat step b by choosing a new ball card. Continue until you cannot remove any pins with any of the available ball cards, or until you choose to stop. This ends the first half of the first frame. Record how many pins the user has removed.

d. Remove the top card from each stack of ball cards and continue attempting to remove all the pins with the remaining ball cards. When you cannot remove any more, or you run out of cards, this ends the second half of the first frame. Record how many pin cards the user removed. If you removed every pin in the first half you get a strike, if you removed every pin in the second half you get a spare.

e. Play 10 frames and as you go track the players score as follows. Each frame you score the number of pins you removed. A spare is worth 10 + the number of pins in your next half-frame. A strike is worth 10 + the number of pins in your next frame.

2. Suit Yourself



a. This is a game for 2-4 players. The game is set up by shuffling a standard deck of 52 playing cards and dealing them out face-up into 5 columns (2 columns of 11 cards and 3 columns of 10). These columns should be staggered, so that the bottom card is on top of all the others, as in the image to the right.

b. Round 1

i. Players take turns in clockwise order taking cards. You can only take cards that are not covered by other cards. Player 1 takes one card. The next player may take up to two cards of the same suit. Each player may take up to one more card than the previous player, as long as they are the same suit. Taken cards may be from any number of columns.

ii. As soon as someone chooses to take less than the maximum allowed, round 2 begins.

c. Round 2

i. Continuing turn order, each player chooses a suit and takes all cards of that suit available. Continue in this manner until every card has been taken

d. After round 2, everyone gets 1 point per card. Whoever has the most cards of each suit scores an additional 5 points. In the case of a tie both players get 5 points. Play 5 games and then the player with the most points wins.

3. Mate

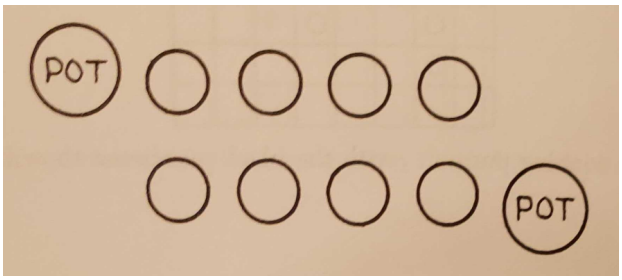
a. This is a game for 2 players, and it is a particularly good one to code an AI opponent for. To begin, shuffle a deck containing the 10s, 9s, 8s, 7s, and 6s of every suit together (20 cards). Deal each player 10 cards.

b. Player 1 chooses any card and plays it face-up in front of themself. Player 2 must then play a card of the same suit. If player 2 has none, then they must play a card of the same number. When you have multiple options, you may choose which card to play.
c. Then, identify the player who played the higher card. Numbers are ranked numerically, and suits are ranked from highest to lowest: spades, hearts, diamonds, clubs. Repeat step b with this player going first.
d. Play continues like this until a player cannot match the suit **or** number, and that player loses.
e. As an optional variant, after being dealt your 10 cards, have both players choose one card to discard without showing the opponent. You will play with 9 cards each.
    i. If you write a really good AI, you can handicap it by having it discard more cards before playing.

Board games

These two games require a particular "board" in front of you to interact with. Think about what the best way to represent it is, to enable you to most easily write code to interact with it.

1. **Cups**
    a. This game is related to mancala. Arrange 10 cups in between two players as shown in the image on the right. Players also receive 40 tokens each.



    b. Players take turns choosing from one of two options. On their turn a player may either
        i. Take 1-4 tokens from their supply and place them in the 4 cups in front of them, one at a time from left to right. If the last token lands in an empty cup, they take all of the tokens in their opponent's cup across from it and add them to their pot.
        ii. Choose one of their cups and move all its tokens one at a time into the cups to its right. They may only do this if the last token lands in their pot. If a cup has more tokens than it needs to reach the pot, they can never choose that cup with this action.
    c. The game ends when both players cannot make a move. Whoever has more tokens in their pot wins.
    d. As an optional variant, code the game so you may have any number of cups per side, and each player starts with 10 times that many tokens.
2. **Paper Boxing**
    a. This is a game for two players. Both players take a blank 4x4 grid and distribute the numbers 1-15 across each square secretly, leaving the top left square blank.
        i. For example:

	1	11	4
12	5	8	9
15	10	2	6
3	7	14	13

ii. Both players begin in the blank square. The first player chooses an adjacent square to travel to (including diagonal). The second player then does the same on their board. Whoever is on the higher number wins the round. That player then goes first in the next round, by once again moving to an adjacent square.

iii. You cannot return to a square you have already visited. If at any point you are fully surrounded by squares you have already visited, you immediately lose the game (not round).

iv. Once all 15 squares have been visited, whoever won more rounds wins the game.

3. **Chess**
    a. Rules: https://en.wikipedia.org/wiki/Rules_of_chess
    b. Package to access stockfish (best chess AI currently): https://pypi.org/project/stockfish/
        i. You will also need to download stockfish itself and point the python library to it: https://stockfishchess.org/
        ii. The stockfish package comes with a way to visualize the board state. You can take inspiration from it, but you should program that yourself since that's the whole point.

Dice games

It might be interesting to revisit these after the probability lecture next week. For these games, it is particularly easy to test hypotheses by simulating many games and performing some statistical analyses.

1. **Thousand**
    a. Any number of players can play this game. Each player starts with a blank 3x3 grid. Players take turns in clockwise order.
    b. On a player's turn, they throw 3 six sided dice. They may then distribute the results however they wish on their grid.
    c. Once everyone's grid is filled, read each row as a 3 digit number and add them up. Whoever's total is closest to 1000 wins.
2. **Criss and Cross**
    a. Up to 10 players can play this game. Each player starts with a blank 5x5 grid, except for the central square which contains a random number 0-9. Each grid has a different central number.
    b. Each round a player rolls two six sided dice. Sum the result to obtain a single number. Treat 10s as 0, 11s as 1, and 12s as 2. Every player must then place that number in one square on their grid.

i. Whenever a row or column is completely filled, that player scores it. A row/column scores as follows
    1. 5 of a kind = 10 points
    2. 4 of a kind = six points
    3. 3 of a kind = 3 points
    4. 2 of a kind = 1 point
    5. Sequence of 5 consecutive digits = 5 points
        a. For sequences 0s can count as high or low but you may not run over (you can't do 8-9-0-1-2).
    6. You may score multiples of the above in a single row/column.
ii. Keep track of row scores separately from column scores.

c. After 24 turns when the grids are all full, the players compare their scores from rows and from columns, keeping the lower one. The player with the highest score then wins.
d. For an optional two player variant, play on one grid, taking turns placing the rolled number somewhere. One player will only score rows and the other will only score columns. Highest score at the end wins.