This exam has 8 problems (1 bonus) on 18 pages. There are no calculators, phones, or other electronic devices allowed during this exam. The last page has been left intentionally blank and can be used as additional scrap paper if desired. Recall that you may only use in-built functions that we have covered in the course (modules, class, assignments). You may find a list of helpful constructs and functions that you are also allowed to use in the second last page of this exam. If you are unsure, do not hesitate to ask!

Name: _____ **Score:**

ID/Email: _____

Person on your left: _____

Person on your right: _____

Problems	Score	Maximum
Problem 1		15
Problem 2		20
Problem 3		15
Problem 4		15
Problem 5		15
Problem 6		15
Problem 7		15
Problem 8		2
Total		110

Problem 1. Stringify

For this problem you will be writing two functions. You may assume you can use a function from a previous part of this problem in future parts as if they work correctly. Remember to include appropriate documentation at the top of each function (12 points).

(a) Write a function, stringify_digit that takes a single digit (integer) and returns the corresponding digit as a string. For example, stringify_digit(9) returns "9" and stringify_digit(0) returns "0". You may assume that only the digits 0 to 9 will be passed in as arguments to your function. You may not use the str() function for this function.

(b) Write a function reverse, that given a string, returns the reverse of the string. For example, reverse("test") returns "tset".

(c) Write a function named stringify that takes an integer and returns the string representation of that integer. For example, stringify(304) returns "304" and stringify(-234) returns "-234". Hint, one way to approach this problem is to consider building the output string one digit at a time.

Problem 2. Tic-Tac-Toe Recall the Tic-Tac-Toe program we wrote in class. For this section, we would like to build a robust Tic-Tac-Toe simulator between one human and one computer player.

- Your program should keep looping until the game ends (either one player wins or the game ends in a draw).
- Ask the human player for enter a valid coordinate to place an "X". Reprompt them until a valid unused coordinate is supplied. Update the board to reflect the user input.
- Check if the game has ended. If exit, print out the final state of the board, and print a statement indicating the result (who won or was it a draw).
- If the game has not ended, print the new state of the board.
- Continue to generate a random valid move for the computer player. Update the board to reflect the computer's choice.
- Check if the game has ended. If exit, print out the final state of the board, and print a statement indicating the result (who won or was it a draw).
- Repeat until the game ends.

To print the board, you should write a function that outputs a visually appealing description of the board. Consider the following code snippet below.

```
>>> board = [["x", "o", "x"], [".", "x", "."], ["x", "o", "o"]].
>>> print(print_board(board))
x,o,x
-----
.,x,.
-----
x,o,o
```

It will be helpful to implement several helper functions to check if a user has won given the board.

- print_board(board): return a string representation of the board
- check_rows(board, row, ch): given a row index, check if the player with character ch has won
- check_col(board, col, ch): given a col index, check if the player with character ch has won
- check_diagonals(board, ch): check if the player with character ch has won due to character placement on either diagonal

- check_win(board, ch): checks if the player using character ch has won the game using previously defined functions
- check_board_full(board): checks if all the locations on the board have been filled
- check_game_over(board): check if the game has ended (either player has won or the board is filled)

Problem 3. Nutrition

For this question you will be writing a nutrition analysis program that computes the calories in a series of commonly available food items based on their macronutrient content (fat, carbohydrates & protein). The foods you will be analyzing are stored in an external text file. Your program should begin by asking the user for the name of a file – if the file exists you should process the data stored in the file. If not, you should present an error to the user and not continue with the program (ending here, no need to re-run) Food items in the file are stored one product per line in the following format. Note that the item separator here is the "pipe" character ("|"):

Food Name|Fat Grams|Carbohydrate Grams|Protein Grams

Although the names of the column headers will not change (e.g. "Food Name"), the order of the column header could vary. Your program should check the ordering of the column readers by reading and parsing the first line of the file before proceeding.

For example, here is some sample data stored within one of these files:

```
Food Name|Fat Grams|Carbohydrate Grams|Protein Grams
Bread (white, 1 slice)|0.82|12.65|1.91
Bread (whole wheat, 1 slice)|1.07|12.26|2.37
Apple (1 medium)|0.23|19.06|0.36
Hamburger (McDonalds)|8|32|12
Grilled Chicken Salad (1, medium, with dressing)|12|13|19
Orange Juice (1 glass, small)
Orange Juice (1 glass, large)|0.5,25.79,1.74,112
Grape Juice (1 glass, small)|XYZ|123|FOOBAR
```

Note that most lines of data are intact, but some have too few or too many items (orange juice), and some contain non-numeric characters (grape juice). These items can be considered corrupted and should not be analyzed. Once you have successfully opened a file you should compute the calories value for each item using the following formulas: 1g fat = 9 calories, 1g carbohydrate = 4 calories, and 1g protein = 4 calories. Your program should then write out a new file called "results.txt" with the calorie information appended at the end of the line. Calories should be formatted to 2 decimal places. For example, processing the text file described above should produce the following output in "results.txt" (columns headers should be in the order given with calories appened to the end of each line). Note the omission of the three "corrupted" lines of data:

```
Food Name|Fat Grams|Carbohydrate Grams|Protein Grams|Calories
Bread (white, 1 slice)|0.82|12.65|1.91|65.62
Bread (whole wheat, 1 slice)|1.07|12.26|2.37|68.15
Apple (1 medium)|0.23|19.06|0.36|79.75
```

```
Hamburger (McDonalds)|8|32|12|248.00
Grilled Chicken Salad (1, medium, with dressing)|12|13|19|236.00
```

Note that you cannot assume that your text file will contain only these products – you may potentially have to deal with thousands (or millions!) of products formatted using the encoding scheme described here. Also note that there is no user input required for this question. In addition, at the end of the program you should output the item name and calorie value of the items with the highest and lowest calorie values found in the file.

Problem 4. Cards
In a standard deck of 52 cards, each card has a rank and a suit.

The ranks are: 2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, King, Ace.

The suits are: Clubs, Diamonds, Hearts, Spades.

For this question, we represent each rank by a string of length one:

'2' to '9' for 2 to 9, 'T' for 10, 'J' for Jack, 'Q' for Queen, 'K' for King, 'A' for Ace.

We also represent each suit by a string of length one:

'C' for Clubs, 'D' for Diamonds, 'H' for Hearts, 'S' for Spades.

Finally, we represent a card by a string of length 2, the first character being the rank and the second being the suit. For example, Queen of Hearts is represented by 'QH', and 3 of Clubs is represented by '3C'.

In the game of bridge, a hand consists of 13 cards, which we represent by a list of cards. When given a hand, here is the common way to count points.

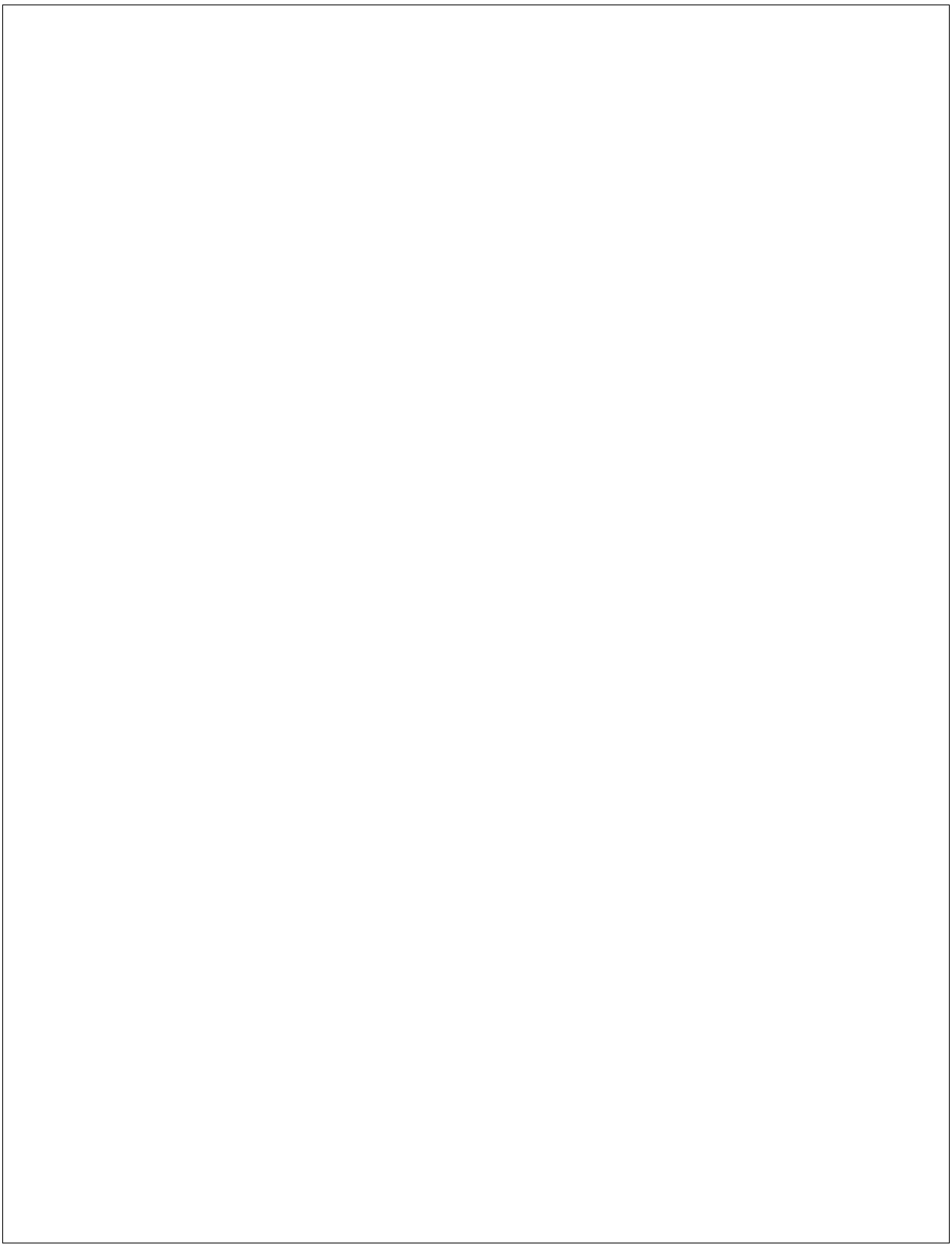
High rank points:

- For each Ace, we count 4 points.
- For each King, we count 3 points.
- For each Queen, we count 2 points.
- For each Jack, we count 1 point.

Short suit points:

- For each suit, we count 1 point if the hand contains exactly two cards of that suit.
- For each suit, we count 2 points if the hand contains exactly one card of that suit.
- For each suit, we count 3 points if the hand contains no cards of that suit.

Write a function which takes a hand as input and returns the number of points in that hand.



Problem 5. Transpose

A function was in the process of being written for a module. Your job is to complete it. Fortunately for you, they left comments to help you complete it (comments are life savers aren't they!). Complete the code in the space provided.

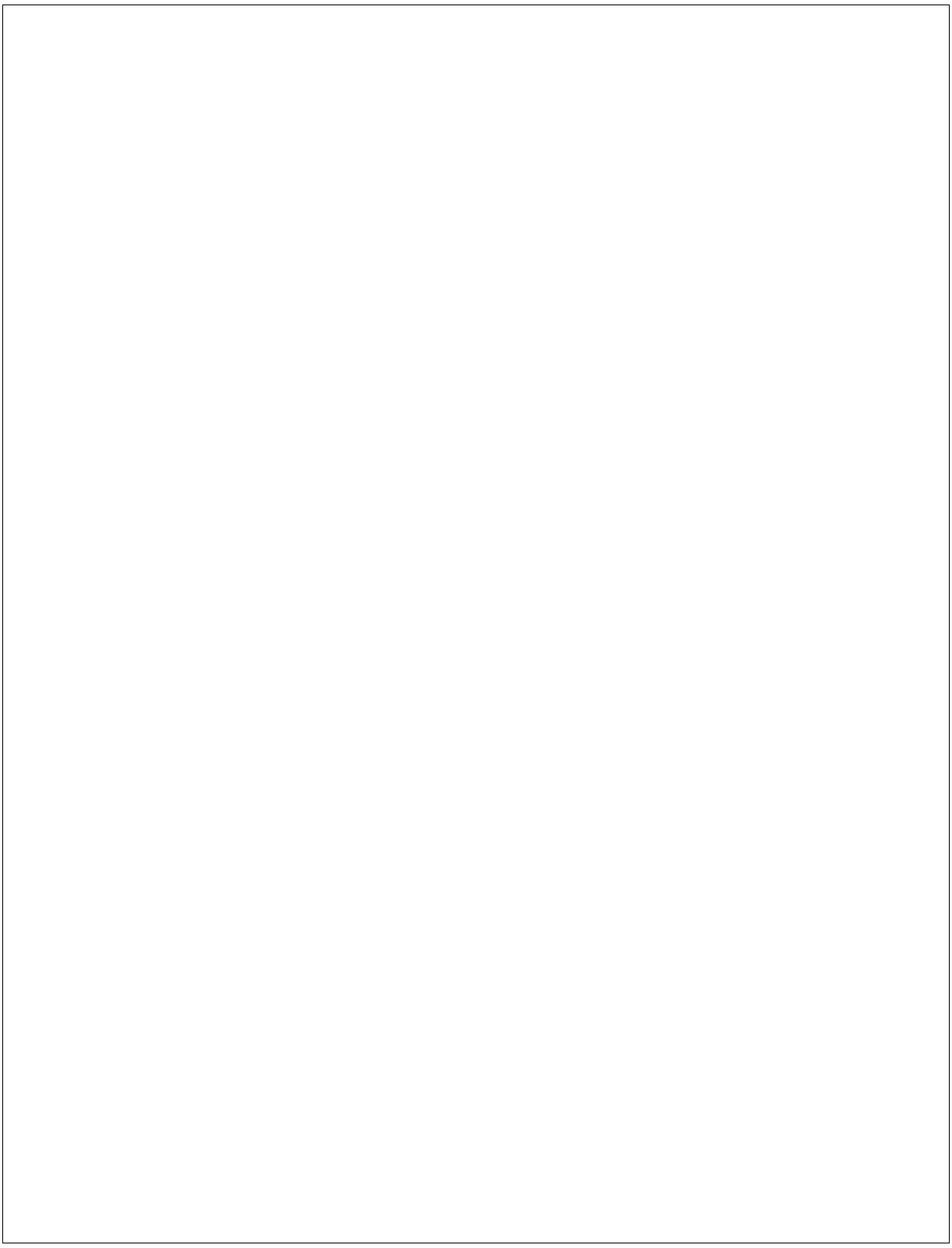
```
def transpose(strlist):
''' (list of str) -> list of str
Return a list of m strings, where m is the length of a longest string
in strlist, if strlist is not empty, and the i-th string returned
consists of the i-th symbol from each string in strlist, but only from
strings that have an i-th symbol, in the order corresponding to the
order of the strings in strlist.
Return [] if strlist contains no nonempty strings.
>>> transpose([])
>>> []
>>> transpose([''])
>>> []
>>> transpose(['transpose', '', 'list', 'of', 'strings'])
>>> ['tlos', 'rift', 'asr', 'nti', 'sn', 'pg', 'os', 's', 'e']
'''
```

Problem 6. Histogram

Write a program that prompts the user to enter a series of numbers between 0 and 10 as integers. The user will enter all numbers on a single line, and scores will be separated by spaces. You cannot assume the user will supply valid integers, nor can you assume that they will enter positive numbers, so make sure that you validate your data. You do not need to re-prompt the user once they have entered a single line of data. Once you have collected this data you should compute the following: • The largest value • The smallest value • The range of values • The mode (the value that occurs the most often) • A histogram that visualizes the frequency of each number Here's a sample running of the program.

```
Enter a series of numbers separated by spaces: -5 apple 15 1 2 3 3 4 foo 5 5 5
* -5 is out of bounds - rejecting
* apple is not a number
* 15 is out of bounds - rejecting
* 1 is valid - accepting
* 2 is valid - accepting
* 3 is valid - accepting
* 3 is valid - accepting
* 4 is valid - accepting
* foo is not a number
* 5 is valid - accepting
* 5 is valid - accepting
* 5 is valid - accepting
Largest number: 5
Smallest number: 1
Range of values: 4
Mode: 5
Frequency Histogram:
1 #
2 #
3 ##
4 #
5 ###
```

Note that your frequency histogram does not have to be sorted in ascending order for this program. Your program should support multiple modes, if they exist.



Problem 7. Smartphone
You've been hired by a large telecommunications company to write a program to help people add and remove apps from their smartphone. To do this you should write a CLASS that models a smartphone. Your class should do the following:

```
class Smartphone:

        # construct a new Smartphone
        # smartphones need to keep track of how much space they have left (integer)
        # they also need to keep track of their name (string)
        # smartphones will need some kind of internal system to keep track of all of
        # the apps that are installed, along with their size.  a list or a dictionary
        # would be useful here.
        # when a phone is constructed the 'report' method should be called (see below)
        # this method returns nothing and simply prints the desired output to the user
        def __init__(self, capacity, name):

        # add a new app to the smartphone given an appname (string) and an appsize (integer)
        # if the app is already installed, reject it.  if the phone cannot hold any additional
        # apps because the capacity has been reached, reject it.
        # this method returns nothing and simply prints the desired output to the user
        def add_app(self, appname, appsize):

        # removes an app from the phone based on appname (string)
        # if the app is not installed, reject it
        # this method returns nothing and simply prints the desired output to the user
        def remove_app(self, appname):

        # checks to see if an app is installed based on appname (string)
        # returns True if the app is installed, False if it is not
        def has_app(self, appname):

        # returns the current space available on the phone (integer)
        def get_available_space(self):

        # prints a detailed report that describes the following:
        # Name of phone
        # Capacity of phone
        # Available space
        # # of apps installed
        # a listing of all apps installed, in alphabetical order, with their sizes
        # this method returns nothing and simply prints the desired output to the user
        def report(self):
```

Test your class and make sure it works as you expect (you will need to write your own tester program for this). Next, write a program that asks the user to create a new phone and then allows them to use all of the features in your class. A sample running of this program is below:

```
Size of your new smartphone (32, 64 or 128 GB): 64
Smartphone name: Craig's iPhone
Smartphone created!
Name: Craig's iPhone
Capacity: 0 out of 64 GB
Available space: 64
Apps installed: 0

(r)eport, (a)dd app, r(e)move app or (q)uit: a
App name to add: Angry Birds
App size in GB: 10

(r)eport, (a)dd app, r(e)move app or (q)uit: r
Name: Craig's iPhone
Capacity: 10 out of 64 GB
Available space: 54
Apps installed: 1
* Angry Birds is using 10 GB

(r)eport, (a)dd app, r(e)move app or (q)uit: a
App name to add: Candy Crush
App size in GB: 15

(r)eport, (a)dd app, r(e)move app or (q)uit: a
App name to add: Facebook
App size in GB: 10

(r)eport, (a)dd app, r(e)move app or (q)uit: r
Name: Craig's iPhone
Capacity: 35 out of 64 GB
Available space: 29
Apps installed: 3
* Angry Birds is using 10 GB
* Candy Crush is using 15 GB
* Facebook is using 10 GB

(r)eport, (a)dd app, r(e)move app or (q)uit: a
App name to add: Instagram
App size in GB: 15

(r)eport, (a)dd app, r(e)move app or (q)uit: a
App name to add: TikTok
App size in GB: 15
Cannot install app, no available space

(r)eport, (a)dd app, r(e)move app or (q)uit: r
Name: Craig's iPhone
Capacity: 50 out of 64 GB
Available space: 14
Apps installed: 4
* Angry Birds is using 10 GB
* Candy Crush is using 15 GB
```

```
* Facebook is using 10 GB
* Instagram is using 15 GB

(r)eport, (a)dd app, r(e)move app or (q)uit: e
App name to remove: Facebook
App removed: Facebook

(r)eport, (a)dd app, r(e)move app or (q)uit: a
App name to add: TikTok
App size in GB: 15

(r)eport, (a)dd app, r(e)move app or (q)uit: r
Name: Craig's iPhone
Capacity: 55 out of 64 GB
Available space: 9
Apps installed: 4
* Angry Birds is using 10 GB
* Candy Crush is using 15 GB
* Instagram is using 15 GB
* TikTok is using 15 GB

(r)eport, (a)dd app, r(e)move app or (q)uit: q
Goodbye!
```

Problem 8. Bonus: What is one thing you liked about this course and one suggestion you have to improve it further? How long did you spend on each question in the test?

Python Command Index

Core Language Elements and Functions	**Module Functions**
and	random.randint()
chr	math.sqrt()
def	
elif	**String Testing Methods**
else	isalphanum()
float	isalpha()
for	isdigit()
format	islower()
global	isupper()
if	isspace()
import	find()
in	
input	**String Modification Methods**
int	rstrip()
max	lstrip()
min	lower()
not	upper()
or	capitalize()
ord	title()
print	swapcase()
range	replace()
return	
while	