CSC 110 FINAL REVIEW GUIDE

Answer Key Starts on Page 14

TOPIC LIST:

1. Anything covered in videos, the PAs, or the readings, and other course material up until the final exam

WHERE TO STUDY:

- Videos
- Course book
- This study guide, and previous study guides
- Previous exams
- Prep problems

QUESTIONS?

- DISCORD
- Office Hours
- Email Ben or a TA

REVIEW PROBLEMS:

Note: these are not necessarily the questions that will be on your test. These questions were written collectively by TAs (who haven't seen the test yet) and resemble what we think you need practice with for the test. Do not use this as your only resource for studying.

That being said, feel free to jump questions, and do the ones you think will help you the most. Ask questions on Discord / office hours for additional clarification.

1) Given the PPM file below, draw (labeling the colors) the pixels of what the output would look like when zoomed in. (P.S. try not to use a color picker, you won't have one in the exam).

```
P3
3 3
255
128 128 128   255 255 0    0  255 255
64  64  64    255 128 0    0  255  0
 0   0   0    255  0  0   255 255 255
```

2) Given the list below, indicate how many items would be checked to search for each of the following numbers following the binary search algorithm discussed in class. (Bonus: What is special about this list that allows us to use binary search on it?)

index	0	1	2	3	4	5	6	7
element	-19	-4	0	6	16	25	43	85

a. 16
b. 50
c. -4
d. 6

3) Write a function `check_contains` which takes two parameter, a list and a string. If the string is present in the list, return True. Otherwise, return False. You can assume list only contain string type.

```
my_list = [“abc”, ”def”, ”ghi”]
check_contains(my_list, “abc”) → return True
check_contains(my_list, “zzz”) → return False
```

4) Given a list of strings which contain the names of video games, a publisher name, and the year of release, write a function `game_manager(games)` which takes a list of strings, `games`, as its only argument and returns a dictionary where each year of release (an integer) is mapped to a set of tuples for all of the games released that year. Each tuple consists of a game title and a publisher name.

```
info = [ “Final Fantasy VII||SCEA||1997”,
         “Mirror’s Edge||Electronic Arts||2008”,
         “GTA 4||Rockstar Games||2008”,
         “Grandia||SCEA||1997”,
         “Half Life 2||Valve||2004”]

result = gameManager(info)
print(result)
```

Output:

* Note: Since values are sets, you do not need to worry about the order of the tuples in each set

```
{1997: {(“Final Fantasy VII”, “SCEA”), (“Grandia”, “SCEA”)},
 2008: {(“Mirror’s Edge”, “Electronic Arts”),
        (“GTA 4”, “Rockstar Games”)},
 2004: {(“Half Life 2”, “Valve”)} }
```

5) a) What data types/data structures have we talked about that can't be modified (aren't mutable) ?

b) Assume that you have a dictionary named `items`. How could you add an entry in items where “`CSC`” is the key and `110` is the value?

c) How do I create an empty set?

d) What two functions could I use to remove an item from a set? What is the difference between the two?

e) What are the three modes for opening files and what do they do?

6) What are the potential errors this code can raise if there was no `try/except`? What will be the output if `x=3`, `y=5`, and `z=4`?

```
def foo(x, y):
      z = int(input(“Enter a number”))
      list = [1, 2, 5, “bye”, {7, 8, 9}, 9]
      try:
            print((x * y) < list[z])
      except:
            print(“Problem occured”)
      finally:
            print(“what happens now?”)
```

7) For the following sorts: selection sort, bubble sort, and insertion sort, list the contents of the following list after each iteration of their respective outer loop has completed, for three iterations of that loop.

```
[7, 0, -5, -5, -4, -8]
```

8) Write a function called `count_grades()` that takes a list as a parameter. The list will contain strings representing letter grades, such as `"A"`, `"B"`, `"C"`, `"D"`, or `"F"`. Your function should create a dictionary mapping these strings to the number of times they occur in the list. Your function should print out a message telling the user how many A's were found, then return the dictionary. You may **not** use the `count()` function. You may assume the list only contains valid letter grades.

For example, given the function call:

```
count_grades(["A", "F", "C", "C", "A", "B", "D", "A"])
```

Your function should produce the following output:

```
There were 3 A's.
```

9)

a) What searching technique starts from the beginning of a list and goes to the end?

b) What sorting techniques finds the smallest value and swaps it towards the front?

c) What searching technique only works if the list is sorted?

10) Write a function `create_seating(file_name, rows, columns)` that takes in a string which acts as a file name and two integers, the rows and columns of a resulting 2D list that you should return. The file contains student names, listed all on separate lines. You should take the student names and place them in the seats, filling an entire row before moving on to the next. If you reach the end of a file and there are no more seats, print "Not enough seats!", but still return the seating arrangement. For all empty seats, put an empty string.

```
create_seating("students.txt", 2,3) → [["Heidi", "Dylan", "Aaron"],
                                        ["Meredith", "Ben", ""]]
```

Assuming `students.txt` contained:

```
Heidi
Dylan
Aaron
Meredith
Ben
```

Hint: as you place students, read a line. Readline returns an empty string when it reaches the end of a file

11) Draw an object reference diagram that represents the following code after it has been run. If any objects are garbage collected, be sure to note this.

```
names = [“Connor”,  “Aaron”,  “Adam”]
numbers = [1, 3, 4, 5]
new_list = names
new_list[2] = 8
numbers.append(“Test”)
names.append(“Nathon”)
new_list = “Hello”
numbers = names
```

12) A palindrome is a word that reads the same backward or forward. Write a function that checks if a given word is a palindrome. Character case should be ignored.

`is_palindrome("Deleveled")` → True

13) Use the following statements to determine if the following assignments will cause an error or not:

```
a = “qrstuvwxyz”
b = ( (“mm” , “hh” , “jj”) , (11, 22, 33) , [4,5,6] )
c = { “mayonnaise” : 2 , “instrument” : 3 , “patrick” : 6 }
```

a) `b[0][2] = 12` error/legal?
b) `b[2][0] = 12` error/legal?
c) `c[“patrick”] = 10` error/legal?

14) Describe the sets which are defined by the following four statements in one or more words:

a) What is the union of:
   i) The set of all english vowels
   ii) the set of all english non-vowel letters

b) What is the intersection of:
   i) platonic solids with a number of sides >=4 but < 10
   ii) platonic solids with 8 or more sides

c) What is the difference of:
   i) people under 6 feet tall
   ii) people over 5 feet tall

d) What is the symmetric difference of:
   i) months from January to April
   ii) months from March to December

15)

a) What will the following code print out?

```
a = {2, 5, 7, 8, 9}
b = {2, 4, 3, 1, 8, 10}
c = a.intersection(b)

a.add(10)
c.add(15)

d = a.difference(c)
e = b.union(a)

print(e, c)
print(e.symmetric_difference(c))
```

b) What is the difference between .add() and .union() for sets?

16) Determine the following sort based off the code. There is bubble sort, insertion sort, and selection sort.

```
def sort1(arr):
    for i in range(len(arr)):
        min_idx = i
        for j in range(i + 1, len(arr)):
            if arr[min_idx] > arr[j]:
                min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]

def sort2(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]

def sort3(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
```

17) What will the value of the `new_lst` be after the following function is run?

```
def foo(lst):
     copy_lst = []
     for item in lst:
          if item % 2 == 0:
               copy_lst.append(item * 9)
          elif item % 2 == 1:
               copy_lst.append(item * 12)
          else:
               copy_lst.append(item)
     return copy_lst
```

```
new_lst = [ 2, 1, 4, 3, 6, 5, 8, 7, 10, 9 ]
print(foo(new_lst))
```

18) How many swaps does it take to completely sort this list `[3,44,38,5,47,15,36]` by using :
    a) Selection sort
    b) Insertion sort
    c) Bubble sort

19) How many errors are in this code?

```
my_set = {}
my_set.add(1)
my_set.add("one")

my_list = {2, 4, 6, 8, 10}

for i in range(8):
    my_set.add(i)

for i in range(len(my_list)):
    tyr:
        my_list[i] = "set"
    except:
        print(my_list[i], "doesn't exist")
```

20) Given a PPM file as a parameter in the format:

```
P3

3 5

255

128 128 128   255 255   0      0 255 255

220 120  40   40   70 60      64  64  64

255 128   0    0  255  0      50  80  90

 20  40  60    0    0  0     255   0   0

255 255 255   80   20 75     120 200 250
```

Write a function called `open_ppm()` that will get the ppm file as a parameter and return a list of lists of lists like so- with the outer list representing the entire file, the inner list representing each row, and the innermost list representing each pixel.

Ex: `[[[128,128,128,], [255,255,0], [0,255,255], ... ]]`

21) Given the following function and function calls, determine the output:

```
def rand_output(i, j):
  while (i != 0 and j != 0):
       i = i // j
       j = (j - 1) // 2
       print(str(i) + " " + str(j) + " ", end='')
  print(i)
```

Function Call

```
rand_output(5, 0)
rand_output(3, 2)
rand_output(16, 5)
rand_output(80, 9)
rand_output(1600, 40)
```

22) Write a function `differ_by(s1, s2)` that takes two strings `s1` and `s2` . This function calculates the number of positions where `s1` and `s2` differ and should return this value. You may assume that `s1` and `s2` will always be the same length.

```
differ_by("Megan", "Regan")→ 1
differ_ by("abcd", "abef") → 2
```

23) Write a function named `increment()` where you will be given a 2D list containing integers as a parameter. Iterate through the entire list and increment each number by 10. Keep in mind that inner lists may or may not vary in size.
For example:

```
list  = [[1,2,3],
         [4,5],
         [6]]
increment(list)
```

```
 print(list)
```

The output should look like:

```
[[11,12,13],[14, 15],[16]]
```

24) Write a function named `second_largest()` which will take in a dictionary containing strings as keys and positive integers as their values. Iterate through the dictionary and return the key with the second largest value.

    For example:

```
dictionary = {"not": 23, "fond": 45, "of": 33, "cheesecake": 10}
key = second_largest(dictionary)
print(key)
```

    The output should be:

```
of
```

25) Write a function named `remove_duplicates()` which will take in a list containing either strings or integers as a parameter. Iterate through the list and remove all duplicates. You are not allowed to create any new lists.

    For example:

```
list = ["one", "two", "three", "four", "four", "one", "five", "two"]
remove_duplicates(list)
print(list)
```

    The output should be:

```
["one", "two", "three" "four", "five"]
```

26) Write a function named `count_odd()` that takes a single list of integers as a parameter, called `num_list`. The function must return a dictionary that maps each odd value in `num_list` to the number of times it appears in `num_list`.

For example:

```
num_list = [-7, 0, 2, 8, 13, 13, -7, 6, 33, 6, -7, 17, 18]
check_return = count_odd(num_list)
print(check_return)
```

The output should be:

```
{-7: 3, 13: 2, 33: 1, 17: 1}
```

27) Write a function called swap_mappings() that takes a single dictionary as a parameter, called orig_dict. Orig_dict maps names of students (represented as strings) to their GPA (represented as floats). This function should return a dictionary that maps GPAs to a list of students that mapped to it in orig_dict.

For example:

```
orig_dict = {“Emma”: 3.7, “Jayden”: 3.2, “Lyra”: 3.5, “Greg”: 3.5}
check_return = swap_mappings(orig_dict)
```

The output should be:

```
{3.7: [“Emma”], 3.2: [“Jayden”], 3.5: [“Lyra”, “Greg”]}
```

28) Write a function named multi_csv(filename) that has one parameter for the file name. It stores each line in the file into a list and prints out the list. Also, you should be able to store the files in a dictionary considering that the name is a key and the rest of it should be in a list saved as a value.

file.csv

```
Olivia,515-123-4567,230343,AZ
Emma,515-123-4568,234203,CA
```

Output :
CSV file in a list:
[['Olivia', '515-123-4567', '230343', 'AZ'], ['Emma', '515-123-4568', '234203', 'CA']]
CSV file in a dictionary:
{'Olivia': ['515-123-4567', '230343', 'AZ'], 'Emma': ['515-123-4568', '234203', 'CA']}

29)

Write a function named common_sets. This function should have two parameters, each set will contain strings only. This function should determine the strings that appeared in two of the sets and only appeared in one of the sets.

For example, if this code were run:

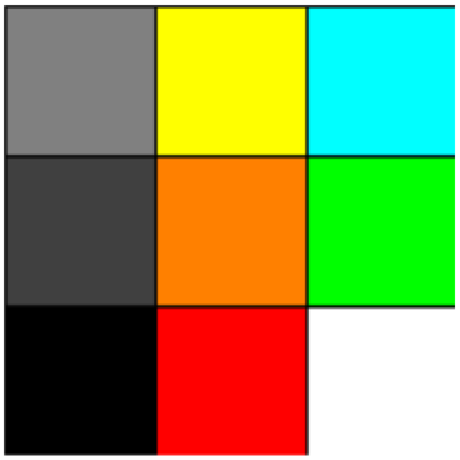
```
set1 = {"B","PP","S"}
set2 = {"PP","S","B"}
common_sets(set1, set2)
```

The output :

```
Appeared once 0
Appeared twice 3
```

Solutions

1)



2)

a) 3
b) 4
c) 2
d) 1

Bonus: it's in sorted order

3)
```
def check_contains(my_list, str):
    for elements in my_list:
        if elements == str:
            return True
    return False
```

4)
```
def game_manager(games):
    res = {}
    for game in games:
        info = game.split("||")
        title, publisher, year = info[0], info[1], int(info[2])
        if year in res:
            res[year].add((title, publisher))
        else:
            res[year] = {(title, publisher)}
    return res
```

5)
   a) tuple, string
   b) my_dict[“CSC”] = 110
   c) set()
   d) remove(), discard(). They both remove an item from a set but remove() throws an exception if the item doesn’t exist in the set. Discard doesn’t throw any exception if an item doesn’t exist in the set making it the safer function to use in general.
   e) “r” read. Reads in the file
      “w” write. Overwrites and existing file or creates a new file. Used for writing information to a file.
      “a” append. Used for adding information to the end of a file.

6) `Error 1: if x and y are not ints/doubles → TypeError: can't multiply sequence by non-int of type 'str'`

   `Error 2: If z is not convertible to an int → TypeError: list indices must be integers or slices, not str`

   `Error 3: If z>5 → IndexError: list index out of range`

   `Error 4: If z==3 or z==4 → TypeError: '<' not supported between instances of 'int' and 'str' and TypeError: '<' not supported between instances of 'int' and 'set', respectively`

   Output:
   ```
   Problem occured
   what happens now?
   ```

7)
   ```
   ORIGINAL = [7, 0, -5, -5, -4, -8]
   Selection
   ----------------------------------
   ```

```
[-8, 0, -5, -5, -4, 7]  Iteration 1
[-8, -5, 0, -5, -4, 7]  Iteration 2
[-8, -5, -5, 0, -4, 7]  Iteration 3

Bubble
-----------------------------------
[0, -5, -5, -4, -8, 7]  Iteration 1
[-5, -5, -4, -8, 0, 7]  Iteration 2
[-5, -5, -8, -4, 0, 7]  Iteration 3

Insertion
-----------------------------------
[0, 7, -5, -5, -4, -8]  Iteration 1
[-5, 0, 7, -5, -4, -8]  Iteration 2
[-5, -5, 0, 7, -4, -8]  Iteration 3
```

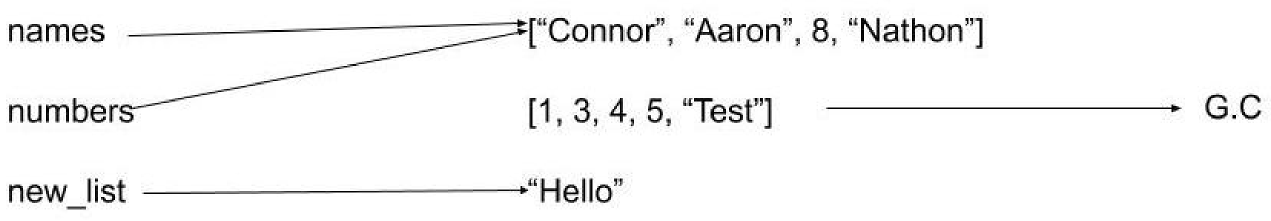
8)
```
def count_grades(grades_list):
     grades_dict = {}
     for grade in grades_list:
          if grade not in grades_dict:
               grades_dict[grade] = 0
          grades_dict[grade] += 1
     print("There were", grades_dict["A"], "A's.")
     return grades_dict
```

9)

a) Sequential
b) Selection
c) Binary

10)
```
def create_seating(file_name, rows, columns):
    seats = []
    file = open(file_name, ‘r’)
    for i in range(rows):
        row = []
        for j in range(columns):
            row.append(file.readline().strip('\n'))
        seats.append(row)
    if len(file.readline()) != 0:
        print("Not enough seats!")
    return seats
```

11)



12)
```
def is_palindrome(string):
    revStr = string[::-1]
    if (revStr == string):
        return True
    return False
```

13)

a) Error
b) Legal
c) Legal

14)

a) the English Alphabet
b) octahedron
c) people under 5 feet tall
d) all months except March and April

15)

a) `{1, 2, 3, 4, 5, 7, 8, 9, 10} {8, 2, 15}`
   `{1, 3, 4, 5, 7, 9, 10, 15}`
b) The .add() method adds a value to the existing set, while the .union() method creates a new set with the contents of the two sets put together.

The .add() method can only be used as set.add(single_value), while the .union() method can only be used as set.union(another_set).

16) sort1 = Selection Sort
sort2 = Bubble sort
sort3 = Insertion Sort

17) `[18, 12, 36, 36, 54, 60, 72, 84, 90, 108]`

18)

a) Selection sort: 5
b) Insertion sort: 9
c) Bubble sort: 9

*note: Go to here to better visualize these sorting algorithms!

19) 7 errors in this code:

```
my_set = {} #This defines a dictionary, not a set

my_set.add(1) #Dictionaries don't have an add method

my_list = {2,4,6,8,10} #This is a set, not a list

my_set.add(i) #Still no add method in dictionaries

tyr: #Misspelled try

my_list[i] = "set" #Can't index into sets, they have no order

print(my_list[i], "doesn't exist") #IndexError: list index out of
                                   #range error
```

20)
```
def open_ppm(file):
    file = open(file, ‘r’)
    file.readline()
    file.readline()
    file.readline()
    ppm = []
    for line in file:
        line = line.strip(‘\n’).split()
```

```
            row = []
            for i in range(0, len(line), 3):
                pixel = [line[i], line[i + 1], line[i + 2]]
                row.append(pixel)
            ppm.append(row)
        return ppm
```

21) 5
    1 0 1
    3 2 1 0 1
    8 4 2 1 2 0 2
    40 19 2 9 0 4 0

22)
```
def differ_by(s1, s2):
    count = 0
    for i in range(len(s1)):
        if s1[i] != s2[i]:
            count += 1
    return count
```

23)

```
def increment(param):
   for i in range(len(param)):
       list = param[i]
       for j in range(len(list)):
           list[j] += 10
```

24)

```
def second_largest(dictionary):
   largest_key = ''
   second_largest_key = ''
   maximum = -1
   for key, val in dictionary.items():
       if val > maximum:
           prev_largest = largest_key
           largest_key = key
           second_largest_key = prev_largest
           maximum = dictionary[largest_key]
       if val < maximum and (second_largest_key == '' or val >
dictionary[second_largest_key]):
           second_largest_key = key
   return second_largest_key
```

25)

```
def remove_duplicates(list):
   final_set = set()
   i = 0
   while i < len(list):
       elem = list[i]
       if elem in final_set:
           list.pop(i)
       else:
           final_set.add(elem)
           i+=1
```

26)

```
def count_odd(num_list):
    ret_odd = {}
    for num in num_list:
        if num % 2 != 0:
            if num not in ret_odd:
                ret_odd[num] = 1
            else:
                ret_odd[num] += 1
    return ret_odd
```

27)

```
def swap_mappings(orig_dict):
    ret_dict = {}
    for key in orig_dict:
        value = orig_dict[key]
        if value not in ret_dict:
            ret_dict[value] = [key]
        else:
            ret_dict[value].append(key)
    return ret_dict
```

28)

```
def multi_csv(filename):
   # name, phone number , Id, state
   file = open(filename,"r")
   reader = file.readlines()
   data_list = []
   for line in reader:
       lines = line.strip().split(",")
       data_list.append(lines)
   dictionary = {}
   for row in data_list:
       dictionary[row[0]] = row[1:]
   print("CSV file in a list:\n")
   print(data_list)
   print("CSV file in a dictionary:\n")
   print(dictionary)
multi_csv("file.csv")
```

29)

```
def common_sets(set_one, set_two):
   elements = {}
   appearances = {1:0, 2:0}
   combined_sets = set_one.union(set_two)
   for value in combined_sets:
       elements[value] = 0
   for item in elements:
       if item in set_one:
           elements[item] += 1
       if item in set_two:
           elements[item] += 1

   for count in elements.values():
       appearances[count] += 1
   print("Appeared once", appearances[1])
   print("Appeared twice", appearances[2])


set1 = {"B","PP","S"}
set2 = {"PP","S","B"}
common_sets(set1, set2)
```