CSC 110 EXAM 3 REVIEW GUIDE

ANSWER KEY INCLUDED

TOPIC LIST:

1. Reading/Writing Files and Graphics
2. String Processing
3. Dictionaries
4. Sets
5. Mutability, Objects, and References
6. Topics from previous exams
7. Anything covered in class, the PAs, or the textbook readings up until the exam

WHERE TO STUDY:

- Slides
- SOWP
- This study guide

QUESTIONS?

- Go to an Office Hour
- Send us an email

Do Note:

- Some questions are bolded and colored red. **THESE QUESTIONS ARE ON FUTURE TOPICS.** Please ignore those questions they are here to ensure proper ordering of the answer key.
- Those questions include:
    - **16-21, 29, 35-36, 38, 40-42**

REVIEW PROBLEMS:

1) Write a function named `swap`. This function should have a parameter, which is a string of alternating upper and lower case letters. The function should reverse the alternating upper case letters to lower case letters and vice versa and return this string.

   `swap("HeLlO") → "hElLo"`
   `swap("wOrLd") → "WoRlD"`

2) Write a function named `remove_odd`. This function should have a parameter, which is a string. The function should remove the characters that has an odd index.

   `remove_odd("water") → "wtr"`
   `remove_odd("Benjamin") → "Bnai"`

3) Write a function named `occurrences`. This function should have two parameters. The first is a string. The second is a single character. The function should return the amount of times that character occurs in the string.

   `occurrences("Hello world", "l") → 3`
   `occurrences("Itamar Tal is my section leader", "a") → 4`

4) Provide all the ways you can create a new:
   a) list
   b) dictionary
   c) set
   d) tuple

5) a) What function is used to get the keys from a dictionary as a list?
   b) What function is used to get the values from a dictionary as a list?
   c) What is missing from this for loop?

```
for k, v in some_dict:
    # code goes here, assume some_dict exists
```

   d) True or False: Dictionaries have NO order

6) What is the output?

```
dict = {'a':3, 'b':8, 'c':5, 'd':1}
for key in dict:
      print(key, end='')
```

   a) abcd
   b) bcad
   c) dacb
   d) Any of the above

7) Write the mappings of each dictionary key and the value for both `my_dict1` and `my_dict2`, after the following code runs.

```
my_dict1 = {}
my_dict2 = {}
states = ["california", "arizona", "utah", "washington",
          "wyoming", "nevada", "iowa", "ohio", "georgia"]
for i in range(len(states)):
      my_dict1[len(states[i])] = states[i]
for i in range(len(states)-1, -1, -1):
      my_dict2[len(states[i])] = states[i]
```

8) Write a function named `combine`. This function should have two parameter variables. The two parameters are both dictionaries, `dict1` and `dict2`. Return a new dictionary that has the keys and values of both dictionaries. If both of the dictionaries share the same key, add the values together. Do not modify the given dictionaries.

```
dict1 = {'a':5, 'b':8, 'c':12}
dict2 = {'b':2, 'c':5, 'd':7, 'e':2}
combine(dict1, dict2) → {'a':5, 'b':10, 'c':17, 'd':7, 'e':2}
```

9) Suppose you have an example file that has the following format:

```
520-555-1212 0:33
520-867-5309 1:12
520-555-1212 0:08
```

Write a function named `count_mins`. This function should ask the user for a filename that represents a person's phone log. Within the file, each line will include a phone number and an amount of time the call lasted, in hours and minutes. Use a dictionary to keep track of the total number of minutes the person has talked to with each phone number (note that the input file is in hours and minutes, but your function should track total minutes only). Once it has read in all the data from the file, it should print out how many minutes the person has spoken with each phone number and the total number of minutes used in this billing cycle.

```
Output:
520-555-1212 41 minutes
520-867-5309 72 minutes
Total 113 minutes
```

10) Write the function named `encode_string`. This function should have one parameter variable, which is a string. This function should return an encoded version where each alphabetical character is replaced with the number of times it occurred within the string (i.e. 'b' in 'banana' is encoded to 1 because it occurred 1 time, 'a' becomes 3, and 'n' becomes 2). Additionally, the character comparisons should be case insensitive. You MUST use a dictionary in your solution.

```
encode_string("banana") → "132323"
encode_string("Monkey money!") → "222122 22222!"
encode_string("Hi, how are you?") → "21, 211 111 121?"
```

11) Below is a snippet of code that runs through a for loop and adds values to a dictionary. After each iteration is executed, write out the contents of the dictionary:

```
dict = {}
iter_list = ["me", "you", "owie", "his", "it", "he", "you", "eh"]
```

```
for item in iter_list:
      length = len(item)
      if length not in dict:
            dict[length] = set()
      dict[length].add(item)
      # Describe what's happening in the dict at this point
print(dict)
```

12) What will be the output of the following lines of code:

```
my_set = {10, 20, 30, 40, 40, 30, 20, 10, 50}
my_dict = {10 : “dime”, 50 : “50-cent”, 100 : “dollar”}
for stuff in my_dict:
      if stuff in my_set:
            my_set.add(my_dict[stuff])
print(my_set)
print(len(my_set))
```

13) Write a function named `subset`. This function takes in two parameter variables, both which are sets. Return `True` if the first set is a subset (all elements of set A are also elements of set B) of the second set and `False` if it is not. Do not use the `issubset` function.

```
subset({1, 3}, {1, 2, 3}) → True
subset({1, 3, 2}, {1, 2, 3}) → True
subset({1, 4}, {1, 2, 3}) → False
```

14) Write a function named `unique`. The function will take a parameter, which is a list of values. Return a new list that contains only one instance of each value.

```
unique([1,2,2,3,3,3,4,5,5]) → [1,2,3,4,5]
unique([‘a’,‘b’,‘c’,‘c’,‘d’,‘a’]) → [‘a’,‘b’,‘c’,‘d’]
```

15) Fill in the loop table with the correct values. Each row should represent the values of the variables at the line with the comment #LOCATION. You don't necessarily need to fill out all of the rows on the provided table. If you don't fill up the whole table, that isn't necessarily wrong!

```
d1 = {'cash': 5000, 'bills': 150, 'tuition': 10000,
      'gas': 50, 'rent': 900, 'food': 500, 'games': 200}
c = d1['games']
for k in d1:
    a = d1[k]
    b = len(k)
    if a % 500 == 0:
        a = 1
    else:
        c += a
    print(a, b, c)
    #LOCATION
```

	a	b	c
Iteration 1			
Iteration 2			
Iteration 3			
Iteration 4			
Iteration 5			
Iteration 6			
Iteration 7			
Iteration 8			

16) What will be the output produced from the following program?

```
def funct_1(collection):
    collection.add(37.2)
    collection.add("Grab")
    collection.remove("doggy")
    return "Finished"

def funct_2(word, collection):
    grocery = []
    grocery.append(collection)
    grocery[0] = "meat"
    grocery.remove(grocery[len(grocery) - 1])
    grocery.append([55, 101])
    grocery[1:]
    return collection.lower()

def funct_3(structure):
    del structure["card"]
    structure["logger"] = "time"
    structure["moment"] = "point"
    return structure["logger"].capitalize()

def main():
    bag = {"doggy", "grab", 37.2}
    webster = {"logger": "system", "card": "great"}
    grocery = ["milk", 3.50, "butter", 2, "eggs", 3.99]

    var1 = funct_1(bag)
    var2 = funct_2(grocery, var1.upper())
    var3 = funct_3(webster)

    print("Bag: " + str(bag))
    print("Webster: " + str(webster))
```

```
    print("Grocery: " + str(grocery))
    print(var1.upper(), var2, var3)

main()
```

17) Write a function named `anagram`. This function should accept two parameters, both of which are strings. The function should check if the second string is an anagram of the first using dictionaries, returning `True` if so. Otherwise it returns `False`. You are allowed to use the `count()` function. Remember that an anagram is a string formed by rearranging the letters of another string.

```
anagram(“cinema”, “iceman”) → True
anagram(“listen”, “silent”) → True
anagram(“slam”, “mans”) → False
```

18) Write a function `film_data` which takes two dictionaries, `movie_data`, and `oscar_wins` as its parameters, where `movie_data` is a dictionary that has film information mapped to a specific year, and `oscar_wins` contains mappings of films with the amount of Oscar awards the film won. From these two dictionaries, your function should generate a summary of film data in sorted order, starting with the oldest film. The function should only generate film data for films that have won *at least one* Oscar award.

Example:

```
data = {1972: "(The Godfather)::Marlon Brando, Al Pacino, James Caan,
Diane Keaton", 2008: "(The Dark Knight)::Christian Bale, Heath Ledger,
Aaron Eckhart", 2016: "(Hell or High Water)::Chris Pine, Ben Foster,
Jeff Bridges", 1998: "(Saving Private Ryan)::Tom Hanks, Matt Damon, Tom
Sizemore"}

wins = { "Saving Private Ryan": 5,
         "Hell or High Water": 0,
         "The Dark Knight": 2,
```

```
            "The Godfather": 3 }

film_data(data, wins)
```

Output:

```
The Godfather - 1972 (3 Academy Awards)
Starring Marlon Brando, Al Pacino, James Caan, Diane Keaton

Saving Private Ryan - 1998 (5 Academy Awards)
Starring Tom Hanks, Matt Damon, Tom Sizemore

The Dark Knight - 2008 (2 Academy Awards)
Starring Christian Bale, Heath Ledger, Aaron Eckhart
```

19) List all the data structures you have learned into the two boxes below. And explain what the difference between Mutable and Immutable is.

Mutable	Immutable

20) Write the name of the function or keyword used to:

a) Add a value to a set
b) Delete a key/value pair from a dictionary
c) Add something to the end of a list
d) Delete a value from a set, throwing an error if the value wasn't there
e) Get all the key/value pairs from a dictionary
f) Delete a value from a set, doing nothing if the value wasn't there
g) Delete a specific value from a list

h) Delete an element at a specific index from a list

21)

Write the following function named `most_popular` that takes one parameter, which is a list, where each index is formatted as such. “`Name 1:Name 2`” In this function, you should read through the given list, find who has the most friends, and return a tuple containing that person’s information, formatted (persons_name, friend_amount).

A friend is defined as follows. Given the following list:

```
["Tony Stark:Steve Rogers", "Groot:Steve Rogers", "Thanos:Scott
Lang", "Barry Allen:Pietro Maximoff", "Tony Stark:Steve Rogers"]
```

Everything on the left of the ‘`:`’ (person 1), is a single person who is friends with the person to the right of the ‘`:`’ (person 2). Additionally, person 2 is also friends with person 1, so that relationship should be mapped as well. If a person has already been counted as a friend with another person, that relationship should not be counted again. With the list given above, the tuple containing `(“Steve Rogers”, 2)` should be returned, as although Steve Rogers appears three times, his relationship with Tony Stark is only counted once. You may assume there are at least two elements in the list, and no ties occur.

22)

What is the output for the following code:

```
a = [‘hat’, ‘mat’, ‘rat’]
print(‘rhyme’.join(a))
```

Suppose we have a set `a = {10,9,8,7}` and we execute `a.remove(14)`, what will happen?

23)

What will the following code output?

```
def func1(a, b, c, d):
    a = b
    print(a)
    b = func2(d, c, a)
    print(d)
    func3(b, a ,d)

def func2(c, b, a):
    print(a)
    print(c)
    print(b)
    return c + a + b

def func3(hello, hi, greetings):
    print(hello, hi)
    print(greetings, hello)
    hi = greetings
    print(hi, greetings)

test = “b”
func1("a", "d", test, "c")
```

24)

Write a function `output` that takes a dictionary as a parameter and writes the contents of the dictionary to a text file.

The format of the dictionary will be:

```
{“word”: “definition”, “word2”:”definition2”}
```

The format of the output should be:

```
Word --- definition
Word2 --- definition2
```

What would the code in the following program output?

```
def adder(luke):
    luke.add('abc')
    luke.add('a')
    x = 3

def main():
    mark = {'a', 'b', 'c'}
    print(mark)
    x = 5
    adder(mark)
    x += 1
    print(mark)
    print(x)

main()
```

Example Output (user input in red):

```
File? Food.txt
What food do you want? Pasta
Your snack is on row 1 column 1 of the shelf
```

25)

Write a function called `sort` that will take in a list as a parameter and sort a list of integers. For example, if you were given a list `[ 6 , 8 , 1, 5, 0, -1 , 10]`, the return value would be `[-1, 0, 1,5, 6, 8, 10]`. You cannot use the sort method. (Hint: You should create a new list and append its minimum value then remove that value from the original list)

26)

What does the following code produce?

```
from graphics import graphics

def main():
    dict = {'r':'blue', 'l':'red', 'e':'yellow'}
    points = [200, 500, 250, 100, 400]
    sizes = [140, 90, 200, 10]
    gui = graphics(600, 600, 'practice')
    for key in dict:
        color = dict[key]
        if key == 'r':
            gui.rectangle(points[0], points[1], sizes[2], \
                sizes[1], color)
        elif key == 'l':
            gui.line(points[0], points[0], points[1], points[2],\
                color, sizes[3])
        elif key == 'e':
            gui.ellipse(points[1], points[2], sizes[1], \
                sizes[0], color)

main()
```

27)

What's the errors in the following code? How do you fix it?

```
def func(grade)
#This function is to calculate and print the grade over 60
#
#Parameter: grade is a list which contains numbers of grades

    for i in range(0, len(grade)):
     if grade[i] > 60:
```

```
                print(grade[i])
        else:
                grade.pop(i)


func([96, 88, 23, 66, 73, 89, 90])
```

28)

A student writes a function that should accept a list of integers as a parameter, called current_list, find the even integers in this list, and move those into a new list, called new_list. The new list should then be returned. Does the following code work to accomplish this purpose? If so, explain why. If not, explain why not, and offer an alternative solution.

```
def find_evens(current_list):
    new_list = []
    for index in range(len(current_list)):
        if index % 2 == 0:
            new_list.append(current_list[index])
    return new_list
```

29)

You are given the dictionary

```
{"Larry": 23, "Jill": 30, "Bob": 35, "Susy": 20, "George": 45,
"Kathryn": 25}
```

Write a function that determines the key with the max value assigned to it, and print out the key,value pair.

30)

Write a function called “remove_vowels” that takes a single parameter: a string. The function should return a string that is the same as the parameter string, but with all vowels removed. **You cannot use the replace function.**

Example outputs:

remove_vowels(“Hello there!”) returns “Hll thr!”

remove_vowels(“Nice to meet you!”) returns “Nc t mt y!”

31)

Choose the best answer for the following question. Use this dictionary as reference.

```
beat = {"golden" : "slumbers", "let it": "be", "hey" : "jude",
      "day": "life", "1": "yesterday"}
```

- Is this valid Python syntax?

  ```
  beat["1"] = []
  ```

  a) Yes, because dictionaries are mutable

  b) Yes, because values can be of any type

  c) No, because dictionaries can not be indexed

  d) No, because dictionaries key can not change

- You can create a 2D dictionary by doing the following.

  ```
  beat[{}] = ""
  ```

  a) True, because dictionaries can hold any type

  b) True, because dictionaries are global

  c) False, because 2D dictionaries can not be created

  d) False, because keys can not be mutable types

- Will the following print "hey"?.

  ```
  print(beat["jude"])
  ```

  a) Yes, because keys are accessed through values

  b) Yes, because of local type

  c) No, because it generates a Value Error

  d) No, because dictionaries can not be accessed by value

32)

Below is a Python function that has two parameter variables. The function contains several comments indicating LOCATIONs in the function. After the code, there are a few statements about this code. It is your job to determine if these statements are either always true, sometimes true, or never true. In response to each statement, you should write one of three words: always, sometimes, or never.

```
def do_something(name, age):
    """ Assume name is a non-empty string and age is an int"""
    num_val = 0
    dictionary = {}
    i = 1
    while i < age:
        # LOCATION A
        age -= 1
        if age % 2 == 0:
            name = name * 2
            # LOCATION B
            print(name + " is currently " + str(len(name)) +    "
characters long!")
        if len(name) > 10:
            # LOCATION C
            name = "Kate"
            num_val += 4
            if num_val < 4:
                dictionary["hi"] = age
        i += 2
    # LOCATION D
    print("num_val: " + str(num_val) + "\nname: " + name +
"\nage: " + str(age))
```

STATEMENTS:

i < 0 at LOCATION A.

Response:

num_val % 2 == 0 at LOCATION D.

Response:

At LOCATION C, num_val == 0.

Response:

dictionary will contain the key "hi" at LOCATION D.

Response:

At LOCATION B, len(name) == 2.

Response:

At LOCATION C, len(name) == 10.

Response:

At LOCATION A, age will refer to an integer value that is positive.

Response:

33)

A student is asked to write a python function “remove_keys(lst, dict)” that takes a list and a dictionary.
The list contains values that could be keys in the dictionary, though they don’t have to be. You can assume that the list will only contain immutable types.
The function is tasked with removing all keys from the dictionary, “dict,” that show up in the list.
Examples:

```
----- Running
list = ["1", "2", "3", "this is not a key"]
dictionary = {"1" : 1, "2" : 2, "3" : 2, "4" : 4}
print(dict)
remove_keys(list, dictionary)
print(dict)
------ Has the output
{'1': 1, '2': 2, '3': 2, '4': 4}
{'4': 4}

----- Running
keys = [1,2,3,4,"USA", 327.2]
populations_in_millions = {"USA":327.2, "CHINA":1386.0, \
"CANADA":37.59, "BRAZIL":209.3}
print(populations_in_millions )
remove_keys(keys, populations_in_millions )
print(dicpopulations_in_millions t)
------ Has the output
{'USA': 327.2, 'CHINA': 1386.0, 'CANADA': 37.59, 'BRAZIL': 209.3}
{'CHINA': 1386.0, 'CANADA': 37.59, 'BRAZIL': 209.3}
```

Will the code on the following page accomplish this? If yes then explain what it is doing, if not, then provide an alternate solution and explain where the original went wrong.

```
def remove_keys(lst, dict):
    ret = {}
    for k in dict:
        if not k in lst:
            ret[k] = dict[k]
    return ret
```

Write a function remove_vowels, which would remove all the vowels in a string. The function will take in a list of strings and return a list of strings, in which the strings will not contain any vowels.

Ex: remove_vowels([“Hello”, “Goodbye”, “Yes”,”No”])

Return: [“Hll”,”Gdby”,”Ys”,”N”]

34)

Which of the data structures that we have learned in class would be best suited to the following situations? Possible answers are List, Set, Dictionary. **Explain** your response.

a. A phonebook which contains the name of the person and their phone number.

b. Printing out the names of the students in the class in the order in which they registered for class.

c. Finding out how many unique words there are in our class textbook.

d. Creating a word count of how many times a word appears in our class textbook.

Solutions

1)
```
def swap(original):
    alternate = ""
    for letter in original:
        if letter.isupper():
            # make the letter lowercase if it is uppercase
            alternate += letter.lower()
        else:
            alternate += letter.upper()
    return alternate
```

2)
```
def remove_odd(original):
    removed = ""
    for i in range(len(original)):
        if i % 2 == 0:
            # append the even indices onto the new string
            removed += original[i]
    return removed
```

3)
```
def occurrences(sentence, character):
    count = 0
    for letter in sentence:
        if letter == character:
            count += 1
    return count
```

4)

a) To create a new empty list:

```
empty_lst = []
```

To create a new list with elements:

```
new_lst = [1, 2, 3, 4, 5]
```

```
new_lst = [0] * 5
```

b) To create a new empty dictionary:

```
empty_dict = {}
```

To create a new dictionary with elements:

```
new_dict = {'a' : 1, 'b' : 5, 'c' : 2}
```

c) To create a new empty set:

```
empty_set = set()
```

To create a new set with elements:

```
new_set = {1, 3, 2, 3, 2, 5}
```

d) To create a tuple with elements:

```
new_tup = ("name", "number")
```

5) a) `keys`

b) `values`

c) Should be `for k, v in some_dict.items()`

d) True

6) d) any of the above

7)

dict1	dict2
10 → 'washington'	7 → 'arizona'
7 → 'georgia'	4 → 'utah'
4 → 'ohio'	6 → 'nevada'
6 → 'nevada'	10 → 'california'

8)
```
def combine(dict1, dict2):
    combined = {}

    # iterate through the first dictionary and store it
    # in the new dictionary combined
    for key, value in dict1.items():
        combined[key] = value

    # iterate through the second dictionary and if the key
    # already exists, add onto it. Otherwise create the element
    for key, value in dict2.items():
        if key not in combined:
            combined[key] = value
        else:
            combined[key] += value

    return combined
```

9)
```
def count_mins():
    filename = input("Enter filename: ")
    file = open(filename, 'r')

    phone_log = {}
    for line in file:
        # Strips the last new line and splits it on space
        line = line.strip('\n').split()
        number = line[0] # accesses the phone number
        time = line[1]   # accesses the time associated

        # splits the time on ‘:’ for hours and minutes
        time = time.split(':')
        minutes = int(time[1]) # accesses the minutes as an int

        # multiply the hours with 60 to convert to minutes
        if int(time[0]) > 0:
            minutes += int(time[0]) * 60
```

```
        # stores the number and minutes in the dictionary
        if number not in phone_log:
            phone_log[number] = minutes
        else:
            phone_log[number] += minutes

    total = 0
    for key in phone_log:
        print(key, phone_log[key], "minutes")
        total += phone_log[key]
    print("Total", total, "minutes")
```

10)
```
def encode_string(original):
    dict = {}
    for letter in original:
        # if the character is an alphabetical
        # character (a-z A-Z), then make it lower case
        # and store it in the dictionary
        if letter.isalpha():
            letter = letter.lower()
            if letter not in dict:
                dict[letter] = 0
            dict[letter] += 1

    encoded = ""
    for letter in original:
        # if the character is an alphabetical character,
        # print its occurrence
        if letter.isalpha():
            encoded += str(dict[letter.lower()])
        # print the original if it is a punctuation or space
        else:
            encoded += letter

    return encoded
```

11)
```
{2: {'me'}}
```

```
{2: {'me'}, 3: {'you'}}
{2: {'me'}, 3: {'you'}, 4: {'owie'}}
{2: {'me'}, 3: {'you', 'his'}, 4: {'owie'}}
{2: {'me', 'it'}, 3: {'you', 'his'}, 4: {'owie'}}
{2: {'me', 'he', 'it'}, 3: {'you', 'his'}, 4: {'owie'}}
{2: {'me', 'he', 'it'}, 3: {'you', 'his'}, 4: {'owie'}}
{2: {'eh', 'me', 'he', 'it'}, 3: {'you', 'his'}, 4: {'owie'}}
```

12)

```
{'50-cent', 40, 10, 50, 20, 'dime', 30}
7
```

13)

```
def subset(set1, set2):
    for element in set1:
        # iterate through set1 and if there is an element
        # not in set2, it is not a subset so return False
        if element not in set2:
            return False
    # after iterating through set1 and it exits the loop,
    # all the elements of set1 is in set2
    return True
```

14)

```
def unique(lst):
    unique = set()
    for element in lst:
        unique.add(element)
    return unique
```

15)

	a	b	c
Iteration 1	1	4	200
Iteration 2	150	5	350
Iteration 3	1	7	350

Iteration 4	50	3	400
Iteration 5	900	4	1300
Iteration 6	1	4	1300
Iteration 7	200	5	1500
Iteration 8			

16)

```
Bag: {'grab', 37.2, 'Grab'}
Webster: {'logger': 'time', 'moment': 'point'}
Grocery: ['milk', 3.5, 'butter', 2, 'eggs', 3.99]
FINISHED finished Time
```

17)

```
def anagram(string1, string2):
    dict1 = {}
    for char in string1:
        if char not in dict1:
            dict1[char] = 0
        dict1[char] += 1
    for char in string2:
        if char not in dict1:
            return False
        else:
            if string2.count(char) != dict1[char]:
                return False
    return True
```

18)

```
def film_summary(movie_data, oscar_wins):
    years = []
    for key in movie_data:
        years.append(key)
    years.sort()
```

```
for year in years:
      film_info = movie_data[year].split("::")
      film = film_info[0].strip('()')

      if oscar_wins[film] >= 1:
          award_amount = oscar_wins[film]
          actors = film_info[1].split(",")
          print(film + ' - ' + str(year) + ' (' +
             str(award_amount) + ' Academy Awards)\nStarring '
             + film_info[1] + '\n')
```

19)

Mutable	Immutable

Lists	Strings
Dictionaries	
Sets	

Mutable data structures are able to be changed without creating a new value in memory. Immutable data structures cannot be changed and whenever a change is made to an immutable object in Python, a new object is created in memory.

20)

a) add
b) del
c) append
d) remove
e) items
f) discard

g) remove

h) pop

21)

```
def most_popular(mappings):
    friendships = {}
    for mapping in mappings:
        mapping = mapping.strip("\n").split(":")
        person_1 = mapping[0]
        person_2 = mapping[1]
        if person_1 not in friendships:
            friendships[person_1] = set()
        if person_2 not in friendships:
            friendships[person_2] = set()
        friendships[person_1].add(person_2)
        friendships[person_2].add(person_1)
    maximum = -1
    person = ""
    for other_person, friends in friendships.items():
        if len(friends) > maximum:
            person = other_person
            maximum = len(friends)
    return (person, maximum)
```

22)

'hatrhymematrhymerat'

The method `join()` takes list of strings as input and prints out a string as output. It removes '', and adds the given string with join to the list.

KeyError is raised since there is no such element in the set

23)

```
d
d
c
b
c
cdb d
c cdb
c c
```

24)

```
def output(dict):
      file = open(“out.txt”, ‘w’)
      for key in dict:
            file.write(key + “ --- ” + dict[key] + ‘\n’)
      file.close()
```

```
{‘a’, ‘c’, ‘b’}
{‘a’, ‘abc’, ‘c’, ‘b’}
6
```

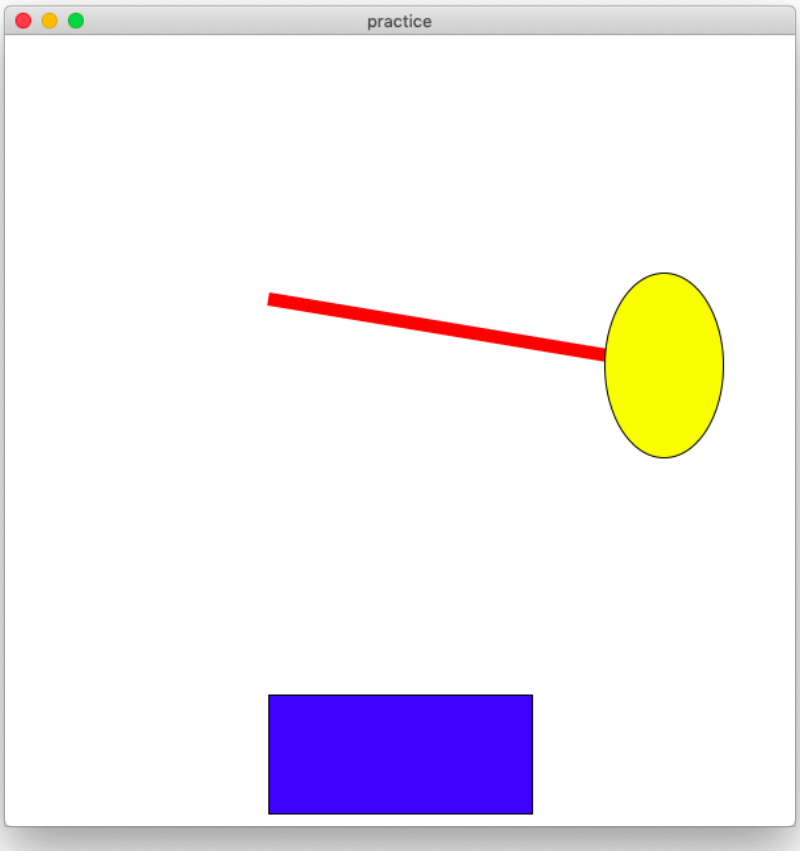
Explanation:
It is a set so it has no order
Because the set is an object, it is modified by functions acting on it
Functions acting on primitive types (like integers or floats) do not modify
things that are pointing to those primitive types outside of the function (like the variable x)

25)

```
def sort(list_int):
      sort_list = []
```

```
while (len(list_int) > 0):
      min = list_int[0]
      for i in list_int:
            if min > i:
                  min = i
      sort_list.append(min)
      list_int.remove(min)
return sort_list
```

26)



27)

Because pop will remove the element but the index i still keep increase, so we will use while loop.

```
- def func(grade)
- #This function is to calculate and print the grade over 60
- #
- #Parameter: grade is a list which contains numbers of grades
-
-     i=0
-     while i < len(grade):
-      if grade[i] > 60:
-              print(grade[i])
-              i+=1
-      else:
-              grade.pop(i)
-
-
- func([96, 88, 23, 66, 73, 89, 90])
```

28)

This code works functionally, but it will not work to serve the purpose described in the question. Rather than adding the even elements from current_list, it adds the elements of current_list that have even indexes. An example fix is as follows:

```
def find_evens(current_list):
    new_list = []
    for item in current_list:
        if item % 2 == 0:
            new_list.append(item)
    print(new_list)
    return new_list
```

29)

```
def find_max():
     new_dict = {"Larry": 23, "Jill": 30, "Bob": 35, "Susy": 20,
"George": 45, "Kathryn": 25}

     value_temp = 0
     key_temp = ""

     for key in new_dict:
     if new_dict[key] > value_temp:
          value_temp = new_dict[key]
          key_temp = key

     print(key_temp,new_dict[key_temp])

find_max()
```

30)

```
def remove_vowels(string):
     result = “”
     for ch in string:
          if ch not in “aeiouAEIOU”:
               result+=ch
     return result
```

31)

- A
- D
- D

32) STATEMENTS:

i < 0 at LOCATION A.

Response: NEVER

num_val % 2 == 0 at LOCATION D.

Response: ALWAYS

At LOCATION C, num_val == 0.

Response: SOMETIMES

dictionary will contain the key "hi" at LOCATION D.

Response: NEVER

At LOCATION B, len(name) == 2.

Response: SOMETIMES

At LOCATION C, len(name) == 10.

Response: NEVER

At LOCATION A, age will refer to an integer value that is positive.

Response: ALWAYS

49)
No, the provided code will not work.
This is because it is returning a new dictionary , which the requested function modified the dictionary that was passed as a parameter.

An alternative solution that would work correctly would be:

```
def remove_keys(lst, dict):
    for k in lst:
        if k in dict:
            del dict[k]
```

33)

```
def remove_vowels(a):
      vowels = ['a','e','i','o','u']
      for i in range(len(a)):
            word = a[i]
            new_word = ""
            for character in word:
                  if character not in vowels:
                        new_word += character
            a[i] = new_word
      return a
```

34)

a) A Dictionary. The name would be the key and the number would be the value. Good because you don't have to loop over lists finding the indexes and then grabbing the information.
b) A list. Whenever order matters, you should almost always think of a list.
c) A Set. The best part about sets is that they won't contain duplicates! If you throw 2 billion "hello" words into a set, the set will only contain one item of the word "hello".
d) A dictionary. The key would be the word, and the value would be the word count. Classic example of a dictionary! If we used lists, we would have to store the words at identical indexes, etc and this wouldn't be as simple as using a dictionary.