CompSci 101 Sec02 **Exam 2** **Fall 2025**

NAME (print): _____

Netid: _____

Which Lecture section are you in:(01 at 10:05am, or 02 at 3:05pm?): _____

Community Standard Acknowledgement (signature): _____

Do NOT spend too much time on any one question.

YOU MUST WRITE YOUR NETID on EVERY PAGE THAT IS LOOSE.

Before starting, make sure your test contains 18 numbered pages, followed by the Python reference sheets. ALL PAGES must be turned in, including the reference pages.

In writing code you do not need to worry about specifying the proper `import` statements. Assume that all libraries we've discussed are imported in any code you write.

You do not need to use dictionaries for problems 2-4.

If you need more space, there are three blank pages you can write on, but you must indicate you have done so.

DO NOT WRITE any work to be graded on the Python Reference sheets.

Do **NOT** discuss this test **with anyone** until the test is handed back.

Problem	Points	Grade	Estimated Minutes
Problem 1	18 pts.		5
Problem 2 (3 parts)	12 pts.		9 (3 mins each)
Problem 3 (3 parts)	12 pts.		15 (5 mins each)
Problem 4 (5 parts)	35 pts.		35 (7 mins each)
TOTAL:	79 pts.		63 min

NETID: _____

PROBLEM 1 : (***What is the output? (18 pts, 5 minutes)***)

For the following code, write the output to the right of each print statement. The output for the first three print statements is already shown.

```
                                                OUTPUT:
seta = set([7,8,6])
print(seta)                                     {7, 6, 8}      # in any order
print(sorted(seta))                             [6, 7, 8]
print(list(seta))                               [8, 7, 6]      # in any order

#---------------------------

seta = set([2,1,3,5,6,0,3])
seta.add(9)
seta.add(5)
print(sorted(seta))

#---------------------------

seta = set([2,0,4,5,3,1,3])
for i in range(3):
    seta.remove(i)
print(sorted(seta))

#---------------------------

seta = set([2, 6, 7, 3])
setb = set([4, 2, 6, 1])
print(sorted(seta - setb))
print(sorted(setb | seta))
print(sorted(setb & seta))

#---------------------------

lst = [4, 9, 2, 'Z', 'A', 'X']
dct = {5: 'B', 3: 'H', 2: 'X', 9: 'Z'}
print(sorted(dct.values()))
print([k for k in dct if dct[k] in lst])
dct[9] = 'T'
dct[4] = 'V'
print(sorted(dct.keys()))
print(sorted(dct.values()))
```

NETID: _____

PROBLEM 2 : (*List Comprehensions (12 pts, 9 minutes)*)

For each problem, assign the variable **result** to a Python expression that calculates the answer. That means if we changed the list given, your code would still calculate the correct answer.

Each of these must be written in one line and include a list comprehension.

Here is an example.

The variable `result` should calculate the list of words from list `vehicles` that have the letter `'a'` in their word. Assume each string in the list is one word that is lowercase.

Using the list `vehicles` below, `result` would calculate the list:

`['train', 'airplane', 'car', 'longboard']`

```
vehicles = ['train', 'airplane', 'car', 'truck', 'bike', 'longboard']
result =
```

ANSWER:

```
result = [w for w in vehicles if 'a' in w]
```

The first problem is on the next page.

NETID: _____

PART A (4 pts), 3 minutes

The variable `result` should be a new list that contains each word in `lst` whose last (rightmost) letter appears elsewhere in the word. You can assume all words in `lst` contain at least one letter. The words in `result` should appear in the same order they appear in `lst`.

Using the list `lst` below, `result` would be the list: `['graded', 'onion, 'serene']`

```
lst = ['sketch', 'carrot', 'graded', 'onion', 'serene', 'get']
```

```
result =
```

PART B (4 pts), 3 minutes

The variable `result` should create a new list of integers obtained by adding 101 to the integers in `lst` whose last (rightmost) two digits are the same; all other integers in `lst` are ignored. Assume that the integers in `lst` are at least 100. The resulting numbers in `result` should appear in the same order as their original numbers appeared in `lst`.

Using the list `lst` below, `result` would calculate the list:

`[1500, 334, 523]`

```
lst = [1399, 390, 101, 201, 233, 747, 422]
```

```
result =
```

NETID: _____

PART C (4 pts), 3 minutes

The variable **result** should be a new list of the last half of each word in `lst` with even length; ignore all odd-length words in `lst`. You can assume all words in `lst` contain at least one letter. The words in `result` are should appear in the same order their original words appeared in `lst`.

Using the list `lst` below, `result` would calculate the list: `['ecake', 'rot', 'ved']`

```
lst = ['cheesecake', 'carrot', 'handtowel', 'halved', 'oatmeal']
```

```
result =
```

NETID: _____

FIRST BLANK PAGE. IF YOU NEED MORE SPACE (MUST TURN IN)

There are still more problems to do!

NETID: _____

PROBLEM 3 : (*Short code (12 pts, 15 minutes)*)

For each of these problems, calculate the answer with code, such that the variable `result`'s value should be the desired answer.

If we changed the list(s) or string(s) given your code should still calculate the correct answer.

You can write your answer in more than one line, and with more than one variable, but be sure `result`'s value should be the answer.

Here is an example.

Calculate the list of words from list `vehicles` that have the letter in their word. Assume each string in the list is one word that is lowercase. The answer should be stored in the variable **result**.

Using the list `vehicles` below, `result`'s value would be `['train', 'airplane', 'car']`.

```
vehicles = ['train', 'airplane', 'car', 'truck', 'bike']

result =
```

ANSWER:

```
result = [w for w in vehicles if 'a' in w]
```

ALTERNATIVE ANSWER:

```
result = [ ]
for w in vehicles:
    if 'a' in w:
        result.append(w)
```

With both of these the answer is calculated and stored in result.

This problem has three parts. The three problems start on the next page.

NETID: _____

PART A (4 pts), 5 minutes

Given the string of words named **phrase**, calculate a list of tuples, where each tuple represents a word from phrase and has two items, 1) the last letter of the word, and 2) the first letter added immediately before the last letter. The tuple for each word should appear in the same order the words appear in phrase. The answer should be stored in the variable **result**.

Using the string `phrase` below, `result`'s value would be:

```
[('e', 'thte'), ('e', 'littlle'), ('g', 'egeg'), ('s', 'layls'),
('n', 'oon'), ('a', 'aa'), ('f', 'lealf')]
```

```
phrase = "the little egg lays on a leaf"
```

Note the first word in phrase is `'the'`, it's last letter is `'e'`, and the word with the first letter of 'the' added immediately before the last letter of the word is `'thte'`, resulting in the tuple `('e', 'thte')`. For the word `'a'`, its only letter is both the first and last in the word.

Write your code below and be sure `result`'s value is the answer.

NETID: _____

PART B (4 pts), 5 minutes

You are looking to buy a new laptop that has as much memory (in gigabytes) as possible without overspending. You are given a list of integers named `memory` and a list of floats named `prices`. Assume the two lists have the same number of elements. An integer in the k-th position of `memory` corresponds to the float in the k-th position of `prices`, which represent a laptop with that amount of memory for that price (in dollars). Calculate the maximum amount of memory available for a price of at most 1500 dollars. The answer should be stored in the variable `result`.

Using the lists below, `result`'s calculated value would be 16:

Write your code below and be sure `result`'s value is the answer.

```
memory = [32, 16, 8, 8, 16, 16]
prices = [2499.99, 1777.77, 655.50, 1122.98, 1425.00, 1900.19]
```

NETID: _____

PART C (4 pts), 5 minutes

You are given two strings of words named `phrase1` and `phrase2`.

Calculate a list of sorted unique words that are in `phrase2` but not `phase1`. The answer should be stored in the variable `result`.

Using the strings `phrase1` and `phrase2` below, `result`'s value would be:

```
['four', 'one', 'pound', 'two']
```

Write your code below and be sure result's value is the answer.

```
phrase1 = "nine zero three five seven six eight"
phrase2 = "eight two one six nine pound nine four one"
```

NETID: _____

SECOND BLANK PAGE IF YOU NEED MORE SPACE (MUST TURN IN), STILL ONE MORE PROBLEM TO DO!

NETID: _____

PROBLEM 4 : (***Movie showings (35 pts, 35 minutes)***)

This problem is about data related to favorite movies.

A student-ran club runs an annual fundraising event, showing movies in Griffith Film Theater. Last year, they recorded survey data from other students that attended. You'll help them analyze this data ahead of this year's new fundraising event.

There are five functions to write in this part. Your functions should work for any valid data, not just the examples shown.

Most of the problems have `datalist` as one of the parameters. The parameter `datalist` is a list of lists, with each inner list representing information about one person and three or more of their favorite movies in order by most favorite first. More specifically, each inner list has:

1. a string representing a student's NetID,
2. an integer representing the student's donation at the previous movie event,
3. a float representing their probability of attending the new event (between 0 and 100), and
4. a list of at least three strings of the person's favorite movies in order by most favorite first.

Assume there is only one line in the file for each student.

For example, assume `datalist` is the lists of lists shown below. Note that the first item in the first inner list is the data for student with NetID `kwn234`, the second item is `50` meaning they donated 50 dollars at the last event, the third item is `65.5`, meaning that there is a 65.5% chance the student attends the new event, and the fourth item is a list of three of their favorite movies in decreasing order of favorites. Their favorite movie is Inception, followed by Titanic, and then Avatar.

```
datalist = [
  ['kwn234',  50, 65.5, ['Inception', 'Titanic', 'Avatar']],
  ['msm020',  10, 42.0, ['Parasite', 'Jaws', 'Inception', 'Memento']],
  ['fmh245',   5, 35.2, ['Gladiator', 'Amelie', 'Dune', 'Parasite']],
  ['ppw123',   0, 29.9, ['Amelie', 'Parasite', 'Gladiator', 'Avatar']],
  ['lol423',   0, 90.2, ['Amelie', 'Inception', 'Jaws']],
  ['rof667', 100, 93.4, ['Casablanca', 'Parasite', 'Amelie', 'Jaws']],
  ['nty621',  25, 44.4, ['Casablanca', 'Inception', 'Citizen Kane', '300']],
  ['xyz115',   9, 31.2, ['Avatar', 'Parasite', 'Casablanca']],
  ['wal966',  40, 80.4, ['Gladiator', 'Amelie', 'Inception', 'Parasite', 'Jaws']]
]
```

Go to the next page to start Part A of this problem.

NETID: _____

Part A (7 pts, 7 minutes)

Write the function named **topFaves** that has two parameters. The first parameter is named **datalist**, which is a list of lists in the format described earlier, and the second parameter is a string named **movie** representing a movie title.

We repeat the format of datalist, a list of lists, with each inner list having 1) a string representing a student's NetID, 2) an integer representing the student's donation at the previous movie event, 3) a float representing their probability of attending the new event (between 0 and 100), 4) a list of at least three strings of the person's favorite movies in order by most favorite first.

This function returns the total number of times that the given **movie** is either the most favorite or second-most favorite among the students.

For example, assume datalist is the list of lists shown on the first page of this problem. We give several examples of calls to this function.

call	**returns**
`topFaves(datalist, "Jaws")`	`1`
`topFaves(datalist, "Amelie")`	`4`
`topFaves(datalist, "Inception")`	`3`

Complete the function below.

```
def topFaves(datalist, movie):
```

NETID: _____

Part B (7 pts, 7 minutes)

Write the function named **likelyFunds** that has two parameters. The first parameter is named **datalist**, which is a list of lists in the format described earlier, and the second parameter is a float named **chance**.

We repeat the format of datalist, a list of lists, with each inner list having 1) a string representing a student's NetID, 2) an integer representing the student's donation at the previous movie event, 3) a float representing their probability of attending the new event (between 0 and 100), 4) a list of at least three strings of the person's favorite movies in order by most favorite first.

This function returns a sorted list of donations (in increasing order) from datalist for which the student has a probability of attending at least **chance**. We give several examples of calls to this function.

call	**returns**
`likelyFunds(datalist, 90.0)`	`[0, 100]`
`likelyFunds(datalist, 50.5)`	`[0, 40, 50, 100]`
`likelyFunds(datalist, 0.0)`	`[0, 0, 5, 9, 10, 25, 40, 50, 100]`

Complete the function below.

```
def likelyFunds(datalist, chance):
```

Part C (7 pts, 7 minutes)

Write the function named **movieLovers** that has one parameter named **datalist**, which is a list of lists in the format described earlier.

We repeat the format of datalist, a list of lists, with each inner list having 1) a string representing a student's NetID, 2) an integer representing the student's donation at the previous movie event, 3) a float representing their probability of attending the new event (between 0 and 100), 4) a list of at least three strings of the person's favorite movies in order by most favorite first.

This function returns the sorted list of NetIDs of students that have the most favorite movies *that are not favorites of anyone else*. For example, assume `datalist` is the list on the first page of this problem, then the call `movieLovers(datalist)` returns `['nty621']`. This student has `'Citizen Kane'` and `'300'` as favorite movies and no other student does, so they have two favorites that are not favorites of any other students. All other students have zero or one favorite movies that are not favorites of any other students.

Complete the function below.

```
def movieLovers(datalist):
```

Part D (7 pts, 8 minutes)

Write the function named **favesThese** that has two parameters, one named **datalist**, which is a list of lists in the format described earlier, and one named **movies** which is a list of strings of movie titles.

We repeat the format of datalist, a list of lists, with each inner list having 1) a string representing a student's NetID, 2) an integer representing the student's donation at the previous movie event, 3) a float representing their probability of attending the new event (between 0 and 100), 4) a list of at least three strings of the person's favorite movies in order by most favorite first.

This function returns a sorted list of NetIDs of students from **favesThese** that like **at least one** movie in the given list **movies**. For example, assume datalist is the list on the first page of this problem, and movies is the list below: `movies = ['Jaws', 'Casablanca']`

then the call `favesThese(datalist,movies)` returns the list `['lol423', 'msm020', 'nty621', 'rof667', 'wal966', 'xyz115']`.

Complete the function below.

```
def favesThese(datalist, movies):
```

NETID: _____

Part E (7 pts, 7 minutes)

Write the function named **processFile** that has one parameter named **filename**, that is the name of a file with data described in the following format. This function reads in the file and returns a list of lists described on the first page of this problem.

Each line in filename has the following format. 1) The student's NetID, 2) a hyphen, 3) their previous donation 4) a hyphen, 5) the likelihood that student attends the new event, 6) a hyphen, and 7) three or more phrases separated by a colon.

The first line represents student with NetID `kwn234`, who donated 50 dollars at the last event, has a 65.5% chance the student attends the new event, and has favorite movies as Inception, followed by Titanic, and then Avatar (in that order).

```
kwn234-50-65.5-Inception:Titanic:Avatar
msm020-10-42.0-Parasite:Jaws:Inception:Memento
[rest of file not shown]
```

The function processFile opens and reads the file and returns the list of lists described earlier. For example if the file above is passed to processFile, then processFile returns the list below.

```
[ ['kwn234', 50, 65.5, ['Inception', 'Titanic', 'Avatar']], ... REST OF LISTS NOT SHOWN ]
```

Complete the function below that has been started for you.

```
def processFile(filename):
    f = open(filename)
    datalist = [ ]
    for line in f:
```

NETID: _____

BLANK PAGE 3 IF YOU NEED MORE SPACE (MUST TURN IN)

Python Reference Sheet for Compsci 101, Exam 2, Fall 2025

DO NOT WRITE ANYTHING TO BE GRADED ON THESE REFERENCE SHEETS!!

Mathematical Operators		
Symbol	**Meaning**	**Example**
+	addition	4 + 5 = 9
-	subtraction	9 - 5 = 4
*	multiplication	3*5 = 15
/ and //	division	6/3 = 2.0 6/4 = 1.5 6//4 = 1
%	mod/remainder	5 % 3 = 2
**	exponentiation	3**2 = 9, 2**3 = 8
String Operators		
+	concatenation	"ab"+"cd"="abcd"
*	repeat	"xo"*3 = "xoxoxo"
Comparison Operators		
==	is equal to	3 == 3 is True
!=	is not equal to	3 != 3 is False
>=	is greater than or equal to	4 >= 3 is True
<=	is less than or equal to	4 <= 3 is False
>	is strictly greater than	4 > 3 is True
<	is strictly less than	3 < 3 is False
Boolean Operators		
x=5		
not	flips/negates the value of a bool	(not x == 5) is False
and	returns True only if both parts of it are True	(x > 3 and x < 7) is True (x > 3 and x > 7) is False
or	returns True if at least one part of it is True	(x < 3 or x > 7) is False (x < 3 or x < 7) is True
Type Conversion Functions		
int(x)	turn x into an integer value	int("123") == 123 int(5.8) == 5
	int can fail, e.g., int("abc") raises an error	
float(x)	turn x into an float value	float("2.46") == 2.46
	float can fail, e.g., float("abc") raises an error	
str(x)	turn x into a string value	str(432) == "432"
type(x)	the type of x	type(1) == int type(1.2) == float
String Index and Splicing		

`s="colorful"`		
		Example
s[x]	index a character	s[0] == 'c' s[-3] == 'f' s[5] == 'f'
s[x:y]	splice of string, substring from index x up to but not including index y	s[2:5] == 'lor' s[:5] == 'color' s[4:-1] == 'rfu' s[5:] == 'ful'

String Functions		
`s="colorful"`		
Name	**Returns**	**Example**
.find(str)	index of first occurrence	s.find("o") == 1
		s.find("e") == -1
.rfind(str)	index of last occurrence	s.rfind("o") == 3
		s.rfind("e") == -1
.index(str)	same as .find(str), error if str not in string	s.index("o") == 1
.count(str)	number of occurrences	s.count("o") == 2 s.count("r") == 1 s.count("e") == 0
.strip()	copy with leading/trailing whitespace removed	" big ".strip() == "big"
.split()	list of "words" in s	"big bad dog".split() == ["big","bad", "dog"]
.split(",")	list of "items " in s that are separated by a comma *In general can split on any string, not just a comma, e.g., s.split(":") will split on a colon and s.split("gat") will split on the string "gat".*	"this,old,man".split(",") == ["this", "old", "man"]
' '.join(lst)	concatenate elements of lst, a list of strings, separated by ' ' or any string	':'.join(['a','b','c']) == "a:b:c"
.startswith(str)	boolean if starts with string	s.startswith("color") == True s.startswith("cool") == False
.endswith(str)	boolean if ends with string	s.endswith("ful") == True s.endswith("color") == False
.upper()	uppercase of s	s.upper() == "COLORFUL"
.lower()	lowercase of s	"HELLO".lower() == "hello"
.isupper()	boolean is uppercase	'A'.isupper() == True 'a'.isupper() == False
.islower()	boolean is lowercase	'A'.islower() == False 'a'.islower() == True
.isalpha()	boolean is alphabetic character	'3'.isalpha() == False '?'.isalpha() == False 'z'.isalpha() == True
.capitalize()	capitalized s	s.capitalize() == "Colorful"

Miscellaneous Functions

help(x)	documentation for module x	
len(x)	length of sequence x, e.g., String or List	len("duke") == 4
list(str)	a list of the characters from string str	list("cards") == ['c','a','r','d','s']
sorted(x)	return list that is sorted version of sequence/iterable x, doesn't change x	sorted("cat") == ['a','c','t']
range(x)	a list of integers starting at 0 and going up to but not including x	range(5) == [0, 1, 2, 3, 4]
range(start, stop)	a list of integers starting at start and going up to but not including stop	range(3, 7) == [3, 4, 5, 6]
range(start, stop, inc)	a list of integers starting at start and going up to but not including stop with increment inc	range(3, 9, 2) == [3, 5, 7]
min(x, y, z)	minimum value of all arguments	min(3, 1, 2) == 1 min("z", "b", "a") == "a"
max(x, y, z)	maximum value of all arguments	max(3, 1, 2) == 3 max("z", "b", "a") == "z"
abs(x)	absolute value of the int or float x	abs(-33) == 33 abs(-33.5) == 33.5

List index, splicing and concatenation (+)		
`lst =[3, 6, 8, 1, 7]`		
		Example
lst[x]	index an element	lst[0] == 3 lst[-1] == 7
lst[x:y]	splice of list, sublist from index x up to but not including index y	lst[1:3] == [6, 8] lst[:4] == [3, 6, 8, 1] lst[3:] == [1,7]
+ operator	concatenate two lists	[3,4] + [1,3,2] == [3,4,1,3,2]

List Functions		
`lst =[3, 6, 8, 1, 7]`		
sum(lst)	returns sum of elements in list `lst`	sum([1,2,4]) == 7
max(lst)	returns maximal element in `lst`	max([5,3,1,7,2]) == 7
lst.append(...)	append an element to lst, changing lst	[1,2,3].append(8) == [1,2,3,8]
lst.insert(pos,elt)	append `elt` to `lst` at position `pos`, changing `lst`	[1,2,3].insert(1,8) == [1,8,2,3]
lst.extend(lst2)	append every element of lst2 to lst	[1,2,3].extend([8,9]) == [1,2,3,8,9]
lst.remove(elt)	remove first occurence of `elt` from `lst`	[1,2,3,2,3,2].remove(2) == [1,3,2,3,2]
lst.sort()	sorts the elements of `lst`	lst = [3,6,8,1,7] lst.sort() lst is now [1, 3, 6, 7, 8]
lst.index(elt)	return index of elt in `lst`, error if elt not in lst	[1,5,3,8].index(5) == 1
lst.count(elt)	return number of occurrences of elt in `lst`	[1,2,1,2,3].count(1) == 2
lst.pop()	remove and return last element in `lst`, so has side-effect of altering list and returns value.	lst = [3,6,8,1,7] x = lst.pop() x is 7, lst is [3,6,8,1]

lst.pop(index)	remove and return element at position index in `lst`, so has side-effect of altering list and returns value. Default index is last value.	lst = [3,6,8,1,7] x = lst.pop(1) x is 6, lst is [3,8,1,7]
Math Functions (import math)		
math.pi	3.1415926535897931	
math.sqrt(num)	returns square root of num as float	math.sqrt(9) == 3.0
File Functions		
open("filename")	opens a file, returns file object	f = open("foo.txt")
open("filename",mode)	specify mode of 'r', 'a', 'w', return file object	f = open("foo.txt", "a")
f.read()	returns the entire file as one string	s = f.read()
Random Functions (import random)		
random.choice(list_of_choices)	returns a random element from list_of_choices. Gives an error if list_of_choices has length 0.	
random.randint(start, end)	Returns a random integer between start and end. Unlike range() and list slicing, the largest value it can return is end, not end-1.	
random.random()	Returns a random float between 0 and 1.	
Set Functions		
set(lst)	returns a set of the elements from list `lst`	
s.add(item)	adds the item into the set, and returns nothing.	
s.update(lst)	adds the elements in the list `lst` into the set, and returns nothing.	
s.remove(item)	removes the item from the set, error if item not there.	
s.union(t)	returns new set representing s UNION t, i.e., all elements in either s OR t, t can be any iterable (e.g., a list)	
s.intersection(t)	returns new set representing s INTERSECT t, i.e., only elements in both s AND t, t can be any iterable (e.g., a list)	
s.difference(t)	returns new set representing s difference t, i.e., elements in s that are not in t	
s.symmetric_difference(t)	returns new set representing elements in s or t, but not in both	
s | t	returns/evaluates to union of s and t, both must be sets.	
s & t	returns/evaluates to intersection of s and t, both must be sets.	
s - t	returns/evaluates to set with all elements in s that are *not* in t	
s ^ t	returns/evaluates to set with all elements from s and t that are *not* in both s and t	
Dictionary Functions		
d[key]	returns the value associated with `key`, error if `key` not in dictionary `d`	
d.get(key)	returns value associated with `key`, returns None if `key` not in dictionary `d`	
d.get(key,default)	returns value associated with `key`, returns `default` if `key` not in `d`	
d.keys()	returns a list/view of the keys in dictionary	
d.values()	returns a list/view of values in dictionary	
d.items()	returns a list/view of tuples, (key,item) pairs from dictionary	
d.update(dict)	updates the dictionary with another dictionary `dict`	