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([7,2,5])
print(seta)                                         {7, 2, 5}   # in any order
print(list(seta))                                   [7, 2, 5]   # in any order
print(sorted(seta))                                 [2, 5, 7]

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

seta = set([4,8,5,8,5,8,5,4,5])
seta.add(5)
seta.add(1)
print(sorted(seta))

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

seta = set([6,5,3,5,6])
seta.remove(5)
print(sorted(seta))

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

seta = set([5, 7, 2, 1])
setb = set([6, 2, 1, 5])
print(sorted(seta & setb))
print(sorted(setb ^ seta))
print(sorted(setb - seta))

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

lst = ["S", "P", "B", 7, 8, 1]
dict = {"G":1, "B":6, "W":7, "S":5}
print(sorted(dict.keys()))
print([k for k in dict if dict[k] in lst])
dict["K"] = 2
dict["W"] = 6
print(sorted(dict.keys()))
print(sorted(dict.values()))
```

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.

PART A (4 pts), 3 minutes

The variable **result** should calculate the list of words from `lst` that contain the string `'et'` in its middle ("et" cannot start the word, and the word cannot end with "et".) 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 calculate the list: `['retro', 'fetch']`

```
lst = ['etch', 'caret', 'retro', 'ether', 'cat', 'fetch', 'stern']


result =
```

PART B (4 pts), 3 minutes

The variable **result** should create a new list of the numbers from `lst` doubled, but only if the original number was exactly two digits. The resulting numbers in `result` should appear in the same order their original number appeared in `lst`.

Using the list `lst` below, `result` would calculate the list (note 11, 75, 43 and 16 are each two digits, so each of them are doubled):

```
[22, 150, 86, 32]
```

```
lst = [8, 11, 75, 43, 3, 16]


result =
```

PART C (4 pts), 3 minutes

The variable **result** should calculate the list of all the words from `lst` that end with 't' and are either of length 2 or length greater than 5. You can assume all words in `lst` contain at least one letter, and the words in `result` are in the same order they were in `lst`.

Using the list `lst` below, `result` would calculate the list: `['it', 'carrot', 'output']`

```
lst = ['cat', 'turtle', 'it', 'carrot', 'star', 'chart', 'output', 'top']
```

```
result =
```

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 resulting 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 'a' 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.

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 first letter of the word and 2) the word with the first letter of the word replaced by the last letter of the word. 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:

```
[('r', 'nun'), ('a', 'a'), ('f', 'wew'), ('m', 'siles'), ('f', 'ror'), ('e', 'exercise')]
```

Note the first word in phrase is 'run', it's first letter is 'r', and the word with the first letter of 'run' replaced with the last letter of the word is 'nun', resulting in the tuple ('r', 'nun').

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

```
phrase = "run a few miles for exercise"
```

PART B (4 pts), 5 minutes

You are given the list of strings named **courses**, and the list of integers named **size**. Assume the two lists have the same number of elements. A course in the kth position of courses corresponds to the integer in the kth position in size, and the integer means the size or enrollment of that course. Assume the course names are unique. Calculate the list of the courses from **courses** that have enrollment greater than 140. Those courses should appear in the same order they appear in the list courses. The answer should be stored in the variable **result**.

Using the lists below, `result`'s calculated value would be: `['CompSci 101', 'Econ 101', 'STA 199']` .

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

```
courses = ['CompSci 101', 'Econ 101', 'Chem 101', 'STA 199', 'PubPol 155']
size = [142, 230, 132, 334, 140]
```

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 either in phrase1 or phrase2, but not in both. The answer should be stored in the variable **result**.

Using the strings `phrase1` and `phrase2` below, `result`'s value would be: `['blue', 'dark', 'light', 'purple', 'yellow']`

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

```
phrase1 = "red blue red purple orange blue green"
phrase2 = "light red dark orange light yellow dark green"
```

PROBLEM 4 : (*I need a new backpack! (35 pts, 35 minutes)*)

There is a new startup that makes a new backpack and stores are ordering the backpack to sell at their store. Each store negotiates with the startup the amount they must pay for each backpack. This problem is about data related to this particular backpack that is sold at different stores.

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

The first four problems have datalist as one of the parameters. The parameter datalist is a list of lists, with each inner list having the following four items: 1) a string representing the name of a store, 2) a string representing the state abbreviation where the store is located, 3) the price the store pays for the backpacks, as a float and 4) a list of integers of orders of the backpack.

We will assume there is only one line in the file for each store.

For example, assume datalist is the lists of lists shown below. Note that the first item in the first inner list is the store "Outdoor Best Prices", the second item is "PA" meaning the store is in Pennslyvania (as "PA" is its state abbreviation), the third item is 5.5, meaning this store pays $5.50 for each backpack it orders to sell, and the fourth item is a list of four orders. The store ordered 8 backpacks, and then 10 backpacks, and then 12 backpacks, and then 20 backpacks.

```
datalist = [
      ['Outdoor Best Prices', 'PA', 5.5, [8, 10, 12, 20]],
      ['School Supplies', 'SC', 5.25, [6, 5, 7]],
      ['Your Outfitter', 'NC', 5.6, [10]],
      ["Bob's Sales", 'VA', 6.1, [4, 6]],
      ["Minerva's Things", 'NC', 5.29, [15, 20, 10, 8]],
      ['School Stuff', 'SC', 5.3, [8, 7]],
      ['The Backpack Store', 'PA', 5.4, [12, 8, 8]],
      ['Hiking High', 'SC', 5.15, [15, 10, 8]]
  ]
```

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

Part A (7 pts, 7 minutes)

Write the function named **numberOrders** 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 **store** representing the name of a store.

We repeat the format of datalist, a list of lists, with each inner list having the following four items: 1) a string representing the name of a store, 2) a string representing the state abbreviation where the store is located, 3) the price the store pays for the backpacks, as a float and 4) a list of integers of orders of the backpack.

This function returns the total number of backpacks the store has ordered.

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**
`numberOrders(datalist, "School Stuff")`	15
`numberOrders(datalist, "Outdoor Best Prices")`	50
`numberOrders(datalist, "Your Outfitter")`	10

Complete the function below.

```
def numberOrders(datalist, store):
```

Part B (7 pts, 7 minutes)

Write the function named **storesFromState** 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 **state**.

We repeat the format of datalist, a list of lists, with each inner list having the following four items: 1) a string representing the name of a store, 2) a string representing the state abbreviation where the store is located, 3) the price the store pays for the backpacks, as a float and 4) a list of integers of orders of the backpack.

This function returns a sorted list of stores from datalist that are in the specified state. We give several examples of calls to this function.

call	**returns**
`storesFromState(datalist, "SC")`	`['Hiking High','School Stuff','School Supplies']`
`storesFromState(datalist, "NC")`	`["Minerva's Things", 'Your Outfitter']`
`storesFromState(datalist, "GA")`	`[]`

Complete the function below.

```
def storesFromState(datalist, state):
```

Part C (7 pts, 7 minutes)

Write the function named **largestTotal** 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 the following four items: 1) a string representing the name of a store, 2) a string representing the state abbreviation where the store is located, 3) the price the store pays for the backpacks, as a float and 4) a list of integers of orders of the backpack.

This function returns the name of the store that has spent the most funds on these backpacks. For example, assume datalist is the list on the first page of this problem, then the call `largestTotal(datalist)` returns `Minerva's Things`. This store bought 53 backpacks at a cost of $5.29 per backpack, which was the largest amount spent from any store on these backpacks.

Complete the function below.

```
def largestTotal(datalist):
```

Part D (7 pts, 8 minutes)

Write the function named **missingStores** that has two parameters, one named **datalist**, which is a list of lists in the format described earlier, and one named **storesWant**, a list of strings of stores the company wants to buy their backpacks.

We repeat the format of datalist, a list of lists, with each inner list having the following four items: 1) a string representing the name of a store, 2) a string representing the state abbreviation where the store is located, 3) the price the store pays for the backpacks, as a float and 4) a list of integers of orders of the backpack.

This function returns a sorted list of stores from **storesWant** that have not bought any of their backpacks yet. For example, assume datalist is the list on the first page of this problem, and storesWant is the list below: `storesWant = ["Appalachian Trail Store", "Your Outfitter", "ABCs of Hiking","Hiking High"]`

then the call `missingStores(datalist,storesWant)` returns the list `['ABCs of Hiking', 'Appalachian Trail Store']`.

Complete the function below.

```
def missingStores(datalist,storesWant):
```

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 name of a store, 2) a colon, 3) the abbreviation of a state 4) a colon, 5) the price the store pays for each backpack 6) a colon and 7) one or more integers separated by a dash.

Here is an example of the first two lines from a file. The first line represents the store Outdoor Best Prices in the state PA. They paid $5.50 for each backpack and they placed 4 orders for backpacks, buying 8, then 10, then 12, and then 20.

```
Outdoor Best Prices:PA:5.50:8-10-12-20
School Supplies:SC:5.25:6-5-7
[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.

```
[ ['Outdoor Best Prices', 'PA', 5.5, [8, 10, 12, 20]], ... 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:
```

Python Reference Sheet for Compsci 101, Exam 2, Spring 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`	