PROBLEM 1 : (*What will Python display? (24 points) (Estimate: 8 minutes)*)

For each code snippet, determine the output next to its corresponding `print` statement. If an error is produced, write `"Error"`. Example outputs are provided for the initial statements.

```
seta = set([6,3,2])
print(seta) # 1              {6, 3, 2}
print(list(seta)) # 2        [6, 3, 2]
print(sorted(seta)) # 3      [2, 3, 6]
print(tuple(seta)) # 4       (2, 3, 6)
```

Part A (4 points)

OUTPUT:

```
nums = (4, 2, 3, 2, 4)
print(len(nums)) # 1
print(nums[1]) # 2
newNums = set(nums)
print(len(newNums)) # 3
print(newNums[1]) # 4
```

Part B (4 points)

OUTPUT:

```
seta = {"A", "b", "c", 1, 2, 3}
setb = {"A", "B", "Z", 1, 2}
print(seta | setb) # 1
print(seta & setb) # 2
print(seta - setb) # 3
print(seta ∧ setb) # 4
```

Part C (4 points)

OUTPUT:

```
d = {"w":  24, "a":  18, "m":  45}
d["w"] = 10
d["z"] = 30
print(18 in d) # 1
print(d["w"]) # 2
print(d.items()) # 3
print(sorted(d.values())) # 4
```

PROBLEM 2 : (*List Comprehensions (16 pts) (Estimate: 10 minutes)*)

Write a list comprehension to solve the given problems. Your solutions should be flexible enough to work with different inputs. That means if we changed the values inside the lists, sets, tuples, or dictionaries given, your code would still calculate the correct answer. Assume strings are lowercase and sequences have at least one item.

Each of these must be written in one line (using a list comprehension) for full credit. You can still get partial credit if you can solve the problem in more than one line (even if no list comprehension is used).

Here is an example:

The variable result should calculate the list of words from `vehicles` that have the letter `"a"` in their word. Using `vehicles` below, write the list comprehension that would store the following list into the variable `result`: `["train", "airplane", "car", "longboard"]`.

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

ANSWER:

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

Part A (4 points)

The variable result should calculate the list of words of length 5 that appear in both `lst1` and `lst2`. Using `lst1` and `lst2` below, write the list comprehension that would store the following list into the variable `result`: `["magic", "genie"]`. The order of the words does not matter.

```
lst1 = ["unicorn", "magic", "dragons", "genie"]
lst2 = ["genie", "dragons", "rainbow", "magic"]


result =
```

Part B (4 points)

The variable result should calculate the list of strings from `words` (shown below) that contain non-repeated letters. Using `words` below, write the list comprehension that would store the following list into the variable `result`: `["sprint", "boxing"]`. The order of the words does not matter.

```
words = ["sprint", "tennis", "volleyball", "boxing"]


result =
```

Part C (4 points)

The variable result should calculate the list of letters from `word` (shown below) that are non-unique letters. Using `word` below, write the list comprehension that would store the following list into the variable `result`: `["o", "o", "k", "k", "e", "e", "e"]`. The order of the letters does not matter.

```
word = "bookkeeper"


result =
```

Part D (4 points)

The variable `result` should calculate the sorted (in increasing order) list of the values in the dictionary `d` that are greater than or equal to 10. Using the dictionary `d` below, write the list comprehension that would store the following list into the variable `result`: `[10, 30, 40]`.

```
d = {"wizard":  10, "orc":  40, "elf":  5, "knight":  30, "hobbit":  2}


result =
```

PROBLEM 3 : (***Spot the Bug (15 points) (Estimate: 15 minutes)***)

Consider the following code snippet that contains an error:

```
01   def findFirstEven(nums):
02      i = 0
03      while nums[i] % 2 != 0 and i < len(nums):
04          print("Checking number at index:", i)
05          i += 1
06      print("First even number found at index:", i)
07
08   if __name__ == "__main__":
09      numbers = [1, 9, 5, 7]
10     findFirstEven(numbers)
```

This function intends to find and print the index of the first even number in a list. If there are no even numbers in the list, then it will print `"There are no even numbers."` But this function is buggy!
The expected output (assuming the code worked correctly) is:

```
Checking number at index: 0
Checking number at index: 1
Checking number at index: 2
Checking number at index: 3
There are no even numbers.
```

Answer the following questions regarding the above code.

a) What line number contains the function call?

b) What line number contains the function definition header?

c) What is the name of the argument?

d) What is the name of the parameter?

e) If the buggy code were to execute, what would be the value of `i` when the first iteration of the loop finished executing? Now, assume all the bugs were removed and the code worked as intended, what would be the value of `i` when the first iteration of the loop with the corrected code finished executing?

Actual value of `i` after the first iteration (using the buggy version):

Expected value of `i` after the first iteration (if bug(s) were removed):

f) If the buggy code were to execute, what would be the value of `i` when the last iteration of the loop finished executing? That is, the last loop iteration that is able to successfully finish iterating before crashing. Now, assume all the bugs were removed and the code worked as intended, what would be the value of `i` when the last iteration of the loop with the corrected code finished executing?

Actual value of `i` after the last iteration (using the buggy version):

Expected value of `i` after the last iteration (if bug(s) were removed):

g) What is the actual output (if any) when this buggy program is executed? If nothing is displayed, write `"nothing"`. If an error is displayed, name or describe the error.

h) What line(s) of code have error(s)? There may be more than one line of code with an error.

i) Using the space below, rewrite the function with the error(s) removed.

PROBLEM 4 : (*Seating Chart (12 points) (Estimated time: 7 minutes)*)

A small theater has a seating arrangement organized in a grid pattern, where each seat is referenced by its row and column numbers. Write the function `findSeat` to check if a specific seat is available. The function should accept two arguments: a list of lists named `seatingChart`, where each inner list represents a row of seats, and a tuple named `seat` representing the desired seat coordinates (row, column). Each seat can either be available, occupied, or not even exist (out of bounds). The function should return True if the seat is available and False otherwise.

The indices for rows and columns start at 0. Be sure to check that the seat's coordinates are within the bounds of the seating chart to avoid index errors. If it's not, return False.

Examples of function calls and expected returns:

Function call	**Return value**	**Comment**
`findSeat([["O"], ["O"], ["O"], ["A"]], (3, 0))`	`True`	This theater has 4 rows of seats such that each row contains 1 seat each.(3,0) means we are searching for the seat in row position 3, column 0. Since that seat is assigned the value "A", it is Available, so return True.
`findSeat([["A", "O"], ["A", "A"], ["O", "A"]], (0, 1))`	`False`	This theater has 3 rows of seats such that each row contains 2 seats each. (0,1) means we are searching for the seat in row position 0, column 1. Since that seat is assigned the value "O", it is Occupied, so return False.
`findSeat([["A", "O"], ["A", "A"], ["O", "A"]], (0, 3))`	`False`	This theater has 3 rows of seats such that each row contains 2 seats each. (1,3) means we are searching for the seat in row position 1, column 3. Since there is no column 3, this seat does not exist, so return False.
`findSeat([["A", "O", "A", "A"], ["A", "A", "O", "O"], ["A", "O", "A", "O"], ["A", "A", "A", "O"]], (2, 2))`	`True`	This theater has 4 rows of seats such that each row contains 4 seats each. (2,2) means we are searching for the seat in row position 2, column 2. Since that seat is assigned the value "A", it is available, so return True.

Complete the function on the next page.

```
def findSeat(seatingChart, seat):
```

PROBLEM 5 : (***Words Collector (15 points) (Estimate: 10 minutes)***)

In an effort to build a unique word collection, you are tasked with writing a function named `uniqueWordsCollector`, which takes a list of strings named `lines` and an integer named `minSize`. This function should analyze `lines`, where each string contains multiple words separated by spaces. The goal is to extract all unique words that have at least `minSize` characters and return them as a set. Assume all characters are lowercase and no punctuation. The order of the returned values does not matter.

Examples of function calls and expected returns:

Function call	**Return value**	**Comment**
`uniqueWordsCollector(["quick brown fox", "sleepy brown dog"], 5)`	{`"sleepy"`, `"quick"`, `"brown"`}	The following strings consist of a length of at least 5 characters: `"sleepy"`, `"quick"`, and `"brown"`. The following strings contain less than 5 characters: `"fox"` and `"dog"`. The string `"brown"` is repeated, so we only keep one instance of it.
`uniqueWordsCollector(["hello world", "to be or not to be", "you got this"], 3)`	{`"hello"`, `"this"`, `"got"`, `"not"`, `"world"`, `"you"`}	The following strings consist of a length of at least 3 characters: `"hello"`, `"this"`, `"got"`, `"not"`, `"world"`, and `"you"`. The following strings contain less than 3 characters: `"to"`, `"be"`, and `"or"`.

Complete the function below.

```
def uniqueWordsCollector(lines, minSize):
```

PROBLEM 6 : (***Baking Contest (17 points) (Estimated time: 15 minutes)***)

In a baking contest, each item is scored solely on taste by judges. Write the function `bestBaker` to determine the winner. The function takes a list of tuples named `scores`, each tuple representing a contestant's taste score given by a judge in the format: `(name, taste_score)`. The total score for a contestant is the sum of all their taste scores, which are integers. The function should return the name of the contestant with the highest total taste score. Assume there will be no ties.

Examples of function calls and expected returns:

Function call	**Return value**	**Comment**
`bestBaker([("Charles",10), ("Conor",8), ("Charles",10), ("Conor",5)])`	`"Charles"`	Charles has a total taste score of 20, while Conor's is 13. So Charles wins.
`bestBaker([("Abigail",2), ("Elisabeth",4), ("Ayda",2)])`	`"Elisabeth"`	The total taste score for Abigail is 2, Elisabeth is 4, and Ayda is 2. This means Elisabeth won with the highest combined score.

Complete the function below.

```
def bestBaker(scores):
```