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[0]) # 1          Error
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:

```
var = { 0:  "spongebob", 1:  "patrick", 2:  "squidward", 3:  "pearl"}
if "spongebob" in var:
  var[0] = "gary"
else:
  var[1] = "gary"
var[4] = "sandy"
print(len(var)) # 1
print(sorted(var.keys())) # 2
print(sorted(var.values())) # 3
print([x[3] for x in var.items()]) # 4
```

Part B (4 points)

OUTPUT:

```
var = ("marge", "homer", ["krusty burger", "moe's tavern"])
print(len(var)) # 1
print(var[2][1]) # 2
print(set(var)) # 3
print("&".join(var[0:2])) # 4
```

Part C (3 points)

OUTPUT:

```
var1 = [1,2,2,2,4,5,5,6]
var2 = set(var1)
print(var2 - {1,2,9,10}) # 1
print(var2 & {1,2,9,10}) # 2
print(var2 ^ {1,2,9,10}) # 3
# Number 4 was omitted for Part C
```

PROBLEM 2 : (*List Comprehensions (20 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 a list that contains the remainder when each **even** number from `values` is divided by 5. Using `values` below, write the list comprehension that would store the following list into the variable `result`: `[1, 2, 4]`. This list is made by dividing 16, 12, and 14 by 5 and recording the remainders. The order of the elements does not matter.

```
values = (13, 16, 12, 17, 14)
```

```
result =
```

Part B (4 points)

The variable `result` should calculate the key-value pairs in `var` such that the key is strictly greater than 8 or the sum of the values is strictly less than 10. Using `var` below, write the list comprehension that would store the following list of tuples into the variable `result`: `[(12, [1, 12]), (5, [3, 1]), (10, [2, 2])]`. The order of the elements does not matter.

```
var = {12:  [1, 12], 5:  [3, 1], 8:  [5, 5], 10:  [2, 2]}
```

```
result =
```

Part C (4 points)

The variable `result` should calculate the list of all **unique** elements that are in `nums2` but not in `nums1`. Using `nums1` and `nums2` below, write the list comprehension that would store the following list into the variable `result`: `[7, 8, 10]`. The elements should be in sorted (ascending) order.

```
nums1 = [3, 5, 9]
nums2 = [3, 7, 8, 8, 8, 9, 10]



result =
```

Part D (4 points)

The variable `result` should calculate a list of strings from `words` that has more than 3 characters but contains the letter `"a"` at most twice. Using `words` below, write the list comprehension that would store the following list into the variable `result`: `["apple", "aahh"]`.

```
words = ["cat", "maa", "apple", "banana", "aahh"]



result =
```

Part E (4 points)

The variable `result` should calculate the list of all tuples in `var` where the 3rd element in each nested tuple is a bigger number than any other number in that nested tuple. You can assume each nested tuple is a length of 3. Using `var` below, write the list comprehension that would store the following list into the variable `result`: `[(1, 2, 5), (3, 1, 4)]`. The order of the elements does not matter.

```
var = ((1, 2, 5), (3, 1, 4), (2, 6, 3), (7, 5, 4))



result =
```

PROBLEM 3 : (*Spot the Bug (10 points) (Estimate: 8 minutes)*)

Consider the following code snippet that contains an error:

```
01   def build_dict(lst):
02      count_dict = {}
03      i = 0
04
05      while i < len(lst):
06          item = lst[i]
07          if str(item).isalpha(): # psst.... remember your reference sheets
08              count_dict[item] += 1
09
10      return count_dict
```

The function `build_dict` accepts a list of items and returns a dictionary where the keys are the alphabetical items from the list and the values are the count of how many times each item appears. The dictionary should only consider alphabetical characters in the list and ignore anything that isn't a letter. However, this function is buggy! Below are some example calls of what the function should return if there were no bugs.

Examples of function calls and expected returns.

Function call	**Return value**	**Comment**
`build_dict(["a", "b", "a"])`	{"a": 2, "b": 1}	The dictionary should count each item in the list. The letter `"a"` appears twice, and `"b"` appears once.
`build_dict(["x", "y", "z", "x", "z", 1, "#"])`	{"x": 2, "y": 1, "z": 2}	Only alphabetic characters are counted; elements `1` and `"#"` are filtered out.
`build_dict([])`	{}	An empty dictionary is returned when the input list is empty.

Answer the following two questions regarding the above code.

a) What is the actual return value (if any) when the following function call is executed for the buggy code? If nothing is returned, write `None`. If an error is displayed before return executes, name or describe the error.

```
build_dict([1, 2, 3])
```

b) Using the space below, rewrite the function with the error(s) removed. **You are required to use a `while` loop to solve this problem.** Don't hate me, I needed to use a `while` loop somewhere.

PROBLEM 4 : (*Total Amount Spent by Item (15 points) (Estimated time: 10 minutes)*)

Write the function `calculate_total` to compute the total amount spent on each item, given some data. The function should accept one argument called `purchases`, which is a list of strings. Each string follows the format `"item:quantity:price"`. The function should return a dictionary where the keys are the items and the values are the total amount spent on those items. An item may show up multiple times in a list. You do not need to round numbers and you can assume `purchases` is not an empty list.

Examples of function calls and expected returns:

Function call	**Return value**	**Comment**
`calculate_total(["apple:5:0.50"])`	`{"apple": 2.5}`	Buying 5 apples at $0.50 each results in a total of $2.50 spent on apples.
`calculate_total(["apple:1:0.50", "apple:2:0.50"])`	`{"apple": 1.5}`	The total for apples accumulates as (1 * 0.50) + (2 * 0.50) which equals 1.5.
`calculate_total(["banana:3:0.15", "orange:2:1.00", "pear:3:1.00"])`	`{"banana": 0.45, "orange": 2.0, "pear": 3.0}`	Calculation: (3 * 0.15) for bananas, (2 * 1.00) for oranges, and (3 * 1.00) for pears.

Complete the function below.

```
def calculate_total(purchases):
```

PROBLEM 5 : (*Unique Long Words Filter (15 points) (Estimated time: 15 minutes)*)

Write the function `filter_words` to process a list of sentences and return a set of unique words that meet specific criteria. The function parameter `sentences` is a list of strings, where each string is a sentence containing multiple words. The function should return a list containing unique words that are longer than 5 characters and start with a vowel. Return the words in sorted (alphabetical) order. You can assume all lowercase strings and that `sentences` will not be an empty list.

Examples of function calls and expected returns:

Function call	**Return value**	**Comment**
`filter_words(["the elephant walked", "an elephant stomped "])`	`["elephant"]`	`"elephant"` is the only word that is longer than 5 characters and starts with a vowel. Only one copy of `"elephant"` is in the list.
`filter_words(["amazing cobblers are available", "under the umbrella", "lazy dog"])`	`["amazing", "available", "umbrella"]`	All words in the set start with a vowel and are longer than 5 characters. They are in sorted (alphabetical) order.
`filter_words(["quick fox"])`	`[]`	No words meet the criteria.

Complete the function below.

```
def filter_words(sentences):
```

PROBLEM 6 : (*School with Total Wins (15 points) (Estimated time: 15 minutes)*)

Write the function `most_wins` to determine the total number of wins for each school, given data about individual game scores. The function parameter is `records`, which is a list of strings such that each string follows the format `"school:score1,score2,...,scoreN"`. The function should return a list of tuples, where each tuple contains a school name and the total amount of wins for that school. If a school appears multiple times in the list, its total number of wins should be accumulated across all occurrences. If the list is empty, the function should return an empty list. The order of the elements does not matter.

Examples of function calls and expected returns:

Function call	**Return value**	**Comment**
`most_wins(["Duke:10,5,7"])`	`[("Duke", 22)]`	Duke has a total of $(10 + 5 + 7) = 22$ points
`most_wins(["UNC:3,7", "Duke:8,4"])`	`[("UNC", 10), ("Duke", 12)]`	UNC has a total of 10, Duke has 12.
`most_wins(["NCSU:5", "UNC:6", "NCSU:4"])`	`[("NCSU", 9), ("UNC", 6)]`	NCSU's total is accumulated as $(5 + 4) = 9$, and UNC has 6.
`most_wins([])`	`[]`	Return an empty list for an empty input.

Complete the function below.

```
def most_wins(records):
```