In writing code you do not need to worry about specifying the proper import statements. Do not worry about getting function or method names exactly right. Assume that all libraries and packages we have discussed are imported in any code you write.
PROBLEM 1:  (20 points)

These questions ask you to write code that references the list `words` below. The Python code you write should work with any values stored in the list `words`. You can write one line of code or many for each of the tasks below.

```
words = ["eagle", "underground", "uncopyrightable", "go", "angry", "bookkeepers", \ 
         "hungry", "america", "screeches", "aglet"]
```

Part A (3 points)
Write code to store in the variable `middles`, a list of strings, the middle letter of each of the 5-letter words in the order they appear in the list. In the example this would be ["g", "g", "l"].

Part B (5 points)
Write code to store in the variable `sameEnds`, a list of strings, those words that start and end with the same letter in sorted order from the longest to the shortest. In the example this would be ["screeches", "america", "eagle"].
Part C (6 points)
Write code to store in the variable `unique`, a list of strings, those words that do not contain repeated letters in sorted order alphabetically. In this example this would be 

```
["aglet", "angry", "go", "hungry", "uncopyrightable"]
```

Part D (6 points)
Given the additional variable `onebigword` the combines all the words in the list into one single string:

```python
>>> onebigword = ".join(words)
"eagleundergrounduncopyrightablegoangrybookkeepershungryamericascreechesaglet"
```

Write code to store in the variable `mostCommon`, a string, the most common letter in all the words. In this example this would be "e".
PROBLEM 2: (12 points)

The following problem asks you to choose a Python sequence (strings, lists/tuples, sets, or dictionaries) that would most help represent the data described.

For each scenario, state the type of sequence you feel best represents the data and justify your decision. Note, more than one sequence may make sense, so be sure to clearly explain your reasoning. If you suggest a dictionary, clearly state the types and purpose of the keys and values. If you suggest a sequence of sequences (e.g., a list of lists or set of tuples), clearly describe the type and purpose of the sub-sequences.

Consider the problem of managing a major airline corporation, organizing pilots and planes, tracking passenger information, and getting them to their flights. For the following scenarios, you can assume that pilots, planes, and passengers have unique names (or IDs), and in-flight rows are simply numbered (i.e., you do not need to come up with a complex solution to represent items uniquely, a simple string or number will do).

1. For a departing flight, organize passengers such that they board in a pre-determined order.

2. At any airport, there are different kinds of planes (Boeing 747, 787, etc.). How would you determine which kinds of planes are common across airports?
3. On a plane, organize the passengers in their seats in their rows.

4. At the airport, organize passengers based on their departure time so that those with less time can be given priority at the security checkpoint.
PROBLEM 3 : \(12\text{ points}\)

The \texttt{contestWinner} APT returns the numeric ID of the winner of the contest whose rules are simple: the winner is the contestant who solves the largest number of tasks. If contestants tie for most tasks solved, the winner is the one who was the first to have all of their tasks solved.

You are given a log of all accepted solutions as a list of int values where the \(i\)-th element of the list is the number of the contestant who submitted the \(i\)-th accepted solution.

As a reminder, this table illustrates what the function is supposed to return:

<table>
<thead>
<tr>
<th>ID</th>
<th>Call</th>
<th>Expected Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>\texttt{contestWinner([4, 7, 4, 1])}</td>
<td>4</td>
</tr>
<tr>
<td>#2</td>
<td>\texttt{contestWinner([10, 20, 30, 40, 50])}</td>
<td>10</td>
</tr>
<tr>
<td>#3</td>
<td>\texttt{contestWinner([123, 123, 456, 456, 456, 123])}</td>
<td>456</td>
</tr>
</tbody>
</table>

Consider the following \texttt{correct} solutions. For each implementation, explain what it does and why it works as a solution to the problem. Note, each solution is shorter than the one before, though not necessarily better, so your explanation should focus on why it works, not just describing the code, even though it may not seem to consider a case that a previous solution did.

Note, the lines are numbered for your convenience in referring to them in your explanation, they would not be included when the code is run.

\textbf{Part A (4 points)}

\begin{verbatim}
    def contestWinner (events):
        d = {}
        for (idx, event) in enumerate(events):
            if event not in d:
                d[event] = [ idx ]
            else:
                d[event] += [ idx ]
        most = max([ len(x) for x in d.values() ])
        winners = sorted([(v[-1], k) for (k, v) in d.items() if len(v) == most ])
        return winners[0][1]
\end{verbatim}
Part B (4 points)

```python
def contestWinner (events):
    1. most = max([ events.count(x) for x in events ])
    2. d = {}
    3. for event in events:
    4.     if event not in d:
    5.         d[event] = 0
    6.     d[event] += 1
    7.     if d[event] == most:
    8.         return event
```

Part C (4 points)

```python
def contestWinner (events):
    1. events.reverse()
    2. tuples = [ (events.count(name), events.index(name), name) for name in set(events) ]
    3. return max(tuples)[2]
```
PROBLEM 4 :  (16 points)

These questions ask you about the code from lab 6 regarding data about basketball stats that is given as a handout with the exam.

Part A (2 points)
In two of the calculations (effectiveFieldGoalPercent and freeThrowPercent), there is an if statement to protect against division by zero. In three other calculations (pointsPerMinute, gameScore, and gameScore), there is no such if statement even though the calculation contains a division. Why do you think it is not necessary in the latter calculations?

Part B (4 points)
In the function loadData, in the loop in the else statement, the new stats are added to the existing stats to create a single list of career stats (also shown below for your convenience):

```python
    currentStats = players[name]
    for (k,s) in enumerate(stats):
        currentStats[k] += s
```

Why is it not necessary to re-associate currentStats with the key in the dictionary? In other words, why does the code not need to include the following line immediately after the loop:

```python
    players[name] = currentStats
```

Your explanation can either be a drawing (like Python Tutor uses) or written.
Part C (4 points)
In the function `loadData`, in the last loop, the player’s stats are converted from a list of `ints` into a dictionary, whose keys are `strings` and whose values are `ints` (also shown below for your convenience):

```python
fieldNames = formattedLines[0].split('t')
for (k,v) in players.items():
    players[k] = dict(zip(fieldNames[1:], v))
```

Explain how this code works, specifically how the data is organized so that the functions `zip` and `dict` can be used effectively.
Your explanation can either be a drawing (like Python Tutor uses) or written.

Part D (6 points)
Write a function that takes a single parameter: a dictionary whose keys are strings, representing the names of the stats given in the file’s header row for a single player, and whose values are ints, representing the values of the corresponding stat; and return a float, the player’s points per scoring attempt. Use the following formula to calculate this stat:

\[ \text{PTS} / \left( \text{FGA} + 0.475 \times \text{FTA} \right) \]

Write the function `pointsPerScoringAttempt` below.

```python
def pointsPerScoringAttempt (stats):
    """
    returns a float representing the calculation described above
    """
```