1. The following statements are executed:

\[ x = [3.5 , -9 , 2.0, 6.7, 4, 2] \]
\[ y = ((3, 7), (4,2), (2,3)) \]
\[ z = \{1: ['one', 'un', 'uno', 'yi'], 2: ['two', 'deux', 'dos', 'er'], 3: ['three', 'trois', 'tres', 'san']\} \]

Answer the following questions about x, y and z. In some parts, if the question asks you what happens when a particular statement is executed, the right answer might be 'it results in an error'.

(a) What is the type of y? **tuple**

(b) What is \( \text{sum}(x[:2]) \)? **-5.5** Frequent incorrect answer: -3.5, because you added \( x[0]+x[1]+x[2] \).

(c) What are \( \text{len}(z) \), \( \text{len}(z[2]) \) and \( \text{len}(z[2][2]) \)? (Be careful!) **3, 4, 3** The admonition to Be Careful was for the last part. Since z is a dictionary, z[2] refers to the item in the dictionary with key 2, and since z[2] is a list, z[2][2] refers to the third item in this dictionary, which is 'dos', a string of length 3. Had z been defined with square instead of curly braces, it would have been a list, and z[2][2] would refer to 'tres', with length 4.

(d) What is the value of x after the following statement is executed?

\[ x = x + 3 \] **error** You can't add a list and an int. \( x[3] \) would have been okay, and have value \( [3.5, -9, 2.0, 6.7, 4, 2, 3] \).

(e) What is the value of x after the following statement is executed?

\[ x . \text{append}(3) \] **[3.5 , -9 , 2.0, 6.7, 4, 2,3]** Note that it is the value of x that changes, the expression \( x . \text{append}(3) \) has no value.

(f) What is the value of y after executing the following statement? (Be very careful!)

\[ y += (5, 6) \] **(3, 7), (4, 2), (2,3), 5, 6** Adding two tuples just concatenates the elements of the tuples, but of course this is designed to make you write the incorrect answer **((3, 7), (4, 2), (2,3),5, 6)**. **(I would feel bad about this trick question if it did not come packaged with a sign**
that in effect said 'Trick Question!') If you really wanted to get $((3,7),(4,2),(2,3),(5,6))$ you would write $y += ((5,6),)$

(g) What is the value of $y$ after executing the following statement?

$y[2] = (6,2)$

**Error** Tuples, like strings, are immutable, and you cannot directly assign to one of its components.

Assume for the next several question that $x,y$ and $z$ are as given at the beginning of this problem (in other words, that the statements described in the questions above have not been executed).

(h) What are the type and value of the following expression?

$[z[w[0]][2] \text{ for } w \text{ in } y \text{ if } w[0] \text{ in } z]$

**Type:** list  **Value:** ['tres', 'dos']

A few people gave the wrong type, but even if you can't decipher the list comprehension, you should know that list comprehension always gives you a list. Since the only elements $w$ of $y$ with $w[0]$ having the value of a key in $z$ are $(3,7)$ and $(2,3)$, this is equivalent to $[z[3][2],z[2][2]]$, which is ['tres', 'dos'], though I accepted ['dos','tres'] without penalty.

2. Consider the function $\text{prob2}$ defined below:

```python
def prob2(nums):
    u=[(int(math.sqrt(x)),x) for x in nums]
    v=[y for y in u if y[0]**2==y[1]]
    return [y[1] for y in v]
```

Suppose we call $\text{prob2}(x)$, where $x$ is the list

$[11, 11, 17, 9, 3, 5, 8, 11, 9, 14, 14, 2, 3, 13, 13, 6, 1, 8, 7, 6]$

(a) What are the first three elements of the list $u$ created in the first statement of the function?

$(3,11), (3,11), (4,17)$
(b) What are the first three elements of the list \( v \) created in the second statement of the function?

\[(3,9), (3,9), (1,1)\]

As it turns out, these are the only three elements of \( v \).

(c) What is the value returned by the function?

\[9,9,1\]

(d) Give a succinct description of what \( \text{prob2 (nums)} \) returns in general, when \( \text{nums} \) is a list of integers.

It returns a list of the perfect squares in \( \text{nums} \).

Some people gave me the brain-dump on this problem, describing every step of the program before getting to the point.

The textbook describes a number of common patterns for processing lists, with names like decorate, filter, undecorate, map, reduce. This one might be called decorate-filter-undecorate, or simply a complicated filter operation. We could also do it with a single list comprehension

\[\{y \text{ for } y \text{ in } \text{nums} \text{ if } \text{int(math.sqrt(y))**2==y}\}\]

3. Write a function

```
def columnsums(v)
```

where \( v \) is a list of lists of integers (i.e., a two-dimensional array). For example

\[
\begin{bmatrix}
2 & -1 & 3 \\
4 & 1 & 6 \\
1 & 1 & 0 \\
0 & -4 & 2
\end{bmatrix}
\]

which we think of as representing the array

2  1  3  
4  1  6  
1  1  0  
0  -4  2

Your function should return a list of the sums of the successive columns, so in the example above, the return value should be \([7,1,11]\). Your function should work with arrays of any width and height, but you may assume that it is only called with arrays that are properly formed (in the sense that all the rows have the same number of elements).
Here are THREE different solutions. The idea of the first one is to traverse the array going down each column in turn, accumulating the column sum and appending it to the new list.

The idea of the second is to simultaneously compute the sum in each column, initially setting a row of 0's, then traversing the array row by row and adding as we go along.

The third solution is the basically the same idea as the first, using list comprehension at two levels.

def columnsum1(v):
    height = len(v)
    width = len(v[0])
    result=[]
    for col in range(width):
        accumsum=0
        for row in range(height):
            accumsum+=v[row][col]
        result.append(accumsum)
    return result

def columnsum2(v):
    height = len(v)
    width = len(v[0])
    result=[0]*width
    for row in range(height):
        for col in range(width):
            result[col]+=v[row][col]
    return result

def columnsum3(v):
    height = len(v)
    width = len(v[0])
    return[sum([v[row][col] for row in range(height)]) for col in range(width)]
def binsearch(thelist, target):
    count = 0
    low = 0
    high = len(thelist) - 1
    while low <= high:
        mid = (low + high) / 2
        if thelist[mid] == target:
            return mid
        elif thelist[mid] < target:
            low = mid + 1
        else:
            # if you're here, then thelist[mid] > target
            high = mid - 1
    return -1

def binsearch_new(thelist, target):
    low = 0
    high = len(thelist) - 1
    while low <= high:
        mid = (low + high) / 2
        if thelist[mid] == target:
            return (thelist[mid], thelist[mid + 1])
        elif thelist[mid] < target:
            low = mid + 1
        else:
            high = mid - 1
    if low > mid:
        return (thelist[mid], thelist[low])
    else:
        return (thelist[high], thelist[mid])

Figure 1: The original binary search

Figure 2: The modified binary search
4. Displayed on the next page is the original version of the binary search function for searching a sorted list, as presented in class, along with a modified version. As you may recall, the original implementation returns the index of the target in the list, if the target item is present, and returns -1 otherwise.

In the modified version the return values have been altered. What the resulting function does is actually quite useful, but it suffers from a serious defect, as you should discover in part (c) below.

Suppose the modified function is called with

\[1,8,27,64,125,216,343,729,1000]\n
as the first argument.

(a) What is the return value if the function is called with 343 as the second argument?

\[(343,729)\]

In the event the item is found, this just returns the tuple consisting of the item and the following item.

(b) ...with 600 as the second argument??

\[(343,729)\]

Look carefully at what happens with the values low, high and mid during the search for 600. At some point we have low=6, high =7 with the target trapped between elements 6 and 7 of the list. In the next step we set mid to 6. Since the value is too low, we set low to 7. So now low, mid and high are all 7, and we exit the loop with high=6, low = mid = 7, so the function returns (thelist[high],thelist[mid]), or (343,729).

(c) ...with 1500 as the second argument?

error

Eventually we get low=high=mid=8, and since we have thelist[mid]<target, we set low=9 and exit the loop. Since low>mid, the function tries to return (thelist[mid],thelist[low]), but thelist[low] is out of the range of indices of thelist.
5. We represent a collection of Boston-area restaurants as a Python dictionary `di`. Here is a typical entry (the restaurant is real---the data comes from Yelp).

"Sandrine's" : ('Cambridge', 'French', '$$$', 3.5)

That is, the key of each entry is the name of the restaurant, and the value field is a tuple consisting of: the location, the type of food, an expense rating (the 3 dollar signs), and an average of reviewer ratings (3.5 stars).

(a) My favorite restaurant is *Mom's Eat at Joe's*. I would like to see if others agree with me, so I want to write a Python statement or statements that will print out the star rating of this restaurant. The code you write should not cause an Python-generated error message on the chance that this restaurant is not in the database, but instead print 'Restaurant not found'.

```python
def get_rating(name):
    if name in di:
        return di[name][-1]
    else:
        return 'Restaurant not found'
```

Most students overthought this one, and tried to give me a function that worked for a general restaurant passed as a parameter, but I only wanted a brief fragment of code.

(b) I want to impress someone by taking them to a fancy Italian place in the North End. (I don't care if the restaurant is good, I just want my guest to see me spending a lot of money.) Write a statement or statements that prints a list of the names of all the restaurants with location 'North End', food type 'Italian', and at least 4 dollar signs. *This list should be sorted in alphabetical order.*

```python
s = [r for r in di if di[r][0] == 'North End' and di[r][1] == 'Italian' and len(di[r][2]) >= 4]
s.sort()
print s
```

Many students assumed 4 stars was the highest possible and wrote `==` in place of `>=`. Others wrote "'North End' in di[r]" in place of "di[r] == 'North End'". That's ok, but if there were a district in the city called 'Italian', it might not work! And some did not use the `len` function and wrote `di[r][2] == '$$$$'`, which actually works!

Some forgot that `s.sort()` does not have a value, and thus print `s.sort()` does not work.