CSC 106 – Winter 2010 – Midterm Exam 2
Friday, February 26th

Name:

Problem 1: For each of the following items, say whether they are syntactically correct Python lists. If yes, say what's the length of the list; if no, explain why not.

1. `[1, 2, 3]`
   Yes. Length is 3.

2. `[1 2 3]`
   No. Commas between elements are missing.

3. `[[]]`
   Yes. Length is 1. The list contains one element, namely `[]`.

4. `[1, 2] + [3, 4]`
   Yes. The value of this expression is a list of length 4.

5. `[1, 2] + 3`
   No. You can't concatenate a list and a number.

6. `[]`
   Yes. Length is 0.

7. `[1,2,3],4,5]` No. The brackets aren't balanced. That is, there is either an opening bracket missing somewhere or an extra closing bracket in the middle.

Problem 2: For the following problems, assume that a non-empty list has been defined and is associated with the variable name `l`. Also note that when I say i'th element I mean what is "normally" considered the i'th element (starting with the first element at the beginning of the list); I do not mean the element with the index i.

- Write an expression that retrieves the last element of the list.
  `l[-1]`

- Write an expression that retrieves the i'th element of the list.
  `l[i-1]`

- Write an expression that yields the sub-list of elements from the i'th element to the k'th element. Both the i'th and the k'th element should be included.
  `l[i-1:k]`

- Write a statement that assigns the value associated with the variable `new_value` to the i'th element.
  `l[i-1] = new_value`

Problem 3: The following piece of code is supposed to print the elements of a list one per line. Instead, it produces an exception. Explain what is going wrong. How can it be fixed?

```
words = ['house', 'sun', 'dog', 'tree', 'car']
for w in words:
    print words[w]
```

The for-loop goes through the list called `words` and assigns each element of the list to the variable `w` one after the other. Since the list contains strings, the value of `w` is thus always a string. However, inside the for-loop, `w` is used as if it was an index (`words[w]`). This causes the error message.

To fix this problem and have the code behave as desired, use `print w` instead of `print words[w]`.

Problem 4: Write (in Python) a function called `filter_empty` that takes a list as parameter. The function should return a new list that is the same as the input list except that it does not contain any empty lists as elements. Here are some examples of how the function should behave:

```
>>> filter_empty([1,2,3])
[1,2,3]
>>> filter_empty([1,[],3])
[1,3]
```

```
def filter_empty(list):
    filtered_list = []
    for ele in list:
        if ele != []:
            filtered_list += [ele]
    return filtered_list
```

Problem 5: Given a list (any list; you don't know anything about this list except that it is a list), how would you go about checking whether a given value is in this list? Specify (in English!) an algorithm that checks whether a given value is an element of a given list.

If we don't know whether or not the list is sorted, we need to use linear search. That is, we go through every element in the list and compare it to the given value. If we come to an element that is equal to the given value, we can immediately return True. Otherwise, we keep on going through the list until we reach the end. If we get to the end, that means that no element in the list was equal to the given value, and we return False.

Explain how binary search works and how it takes advantage of a list being sorted to make searching for a given value in a list faster.

Binary search checks the element in the middle of the list. If that element is the same as the given value, return True.

If it is not equal to the given value, check whether it is smaller or greater than the given value. Since the list is ordered, this will tell us which half of the list the given value is in. If the middle element is smaller than the given value, we know that the given value has to be in the second half; if it is greater, the given value has to be in the first half.

We then repeat the process for the half of the list which has to contain the given value (if the list contains the given value at all).

This strategy uses the information that the list is ordered to rule out one half of the (remaining) list in each step without even looking at it. This means that binary search never looks at all elements of a list whereas linear search, in the worst case, has to look at all list elements.