

University of Pittsburgh

CS 0007 Fall 2009 Test 3

There are four questions for a total of 100 points.

We can't help you answer the questions. Otherwise, it gets too noisy, and some people would get more information than others.

Please just follow the instructions, and use your best judgment.

In case you want to use it, here is "help" information on `str.find`:

```
find(...)  
S.find(sub [,start [,end]]) -> int
```

Return the lowest index in `S` where substring `sub` is found, such that `sub` is contained within `s[start,end]`. Optional arguments `start` and `end` are interpreted as in slice notation.

Return `-1` on failure.

Good luck!

1. (25 points)

Write a program that prompts the user for two strings, and then prints a list of all of the positions in which the second string appears in the first one, where the matches do not overlap. First, complete the following function according to the docstring description. Then, write a main program that calls your function.

```
def non_overlapping_stringmatch(text, pattern):
    """Find all instances of the string 'pattern' in the string 'text' but
    where the matches do not overlap. Return a list containing the
    starting indices of all matches. For example,
    non_overlapping_stringmatch('bbbbbcbbbb', 'bbb') should return
    [0, 6]. Hint: you'll need to take care of one
    special case!

    You may use str.find if you want to."""

    indices = []
    index = text.find(pattern, 0)
    pattern_length = len(pattern)
    while index != -1:
        indices.append(index)
        if pattern_length > 0:
            # special case, where substr is ""
            # there is only one match, position 0
            # this was tricky, so points were not deducted
            index = text.find(pattern_length, index + pattern_length)
        else:
            index = -1

    return indices

if __name__ == '__main__':
    mytext = raw_input('Please enter some text')
    mypattern = raw_input('Please enter a pattern')
    print non_overlapping_stringmatch(mytext, mypattern)
```

2. (25 points) Complete the following function according to its docstring description.

```
def pictures_with_face(faces, person):
    """ 'faces' is a dictionary as in assignment 3: each key is the name
        of a picture, and each value is a list of people who appear in
        that picture. 'person' is the name of a person. Return a list
        of all the pictures that 'person' appears in. Return the
        empty list if 'person' does not appear in any pictures. """
```

```
def pictures_with_face(faces, person):
    pics = []
    for k in faces.keys():
        if person in faces[k]:
            pics.append(k)
    return pics
```

3. (25 points) Below, give the requested information.

(a) Suppose file “temperature.dat” has the following contents:

Average spring monthly temperatures in Dubuque, Iowa,
from 1964 and 1965, n=6

```
24    25    30
16    19    24
```

And, we have this function definition:

```
def input_temperature_data(filename):
    f = open(filename)
    f.readline()
    f.readline()
    f.readline()
    data = []
    line = f.readline()
    while line != "":
        nums = line.split()
        i_nums = []
        for n in nums:
            i_nums.append(int(n))
        data.append(i_nums)
        line = f.readline()
    f.close()
    return data
```

Q: What does the following statement print?

```
print input_temperature_data('temperature.dat')
```

```
ans = [[24, 25, 30], [16, 19, 24]]
```

```
(b) def fun_1(L):
    x = []
    for l in L:
        x.append(len(l))
    return x
```

Q: What is printed by the print statement below?

```
x = [[0,4], ['hello', 'there', 'bear'], [[0], ['hi'], [3.45, 6.44, 7.22], ['aeiou']], []]
print fun_1(x)
```

```
ans = [2, 3, 4, 0]
```

```
(c) def fun_2(dict):
    new = {}
    for (x, y) in dict.items():
        if y in new:
            new[y].append(x)
        else:
            new[y] = [x]
    return new
```

Q: What is printed by the print statement below?

```
d = {'Fluffy': 'Noah', 'Rover': 'Sally', 'Spot': 'Jen', \
     'Fishy': 'Noah', 'Crocky' : 'Jen'}
print fun_2(d)
```

```
ans = {'Jen': ['Spot', 'Crocky'], 'Sally': ['Rover'], 'Noah': ['Fluffy', 'Fishy']}
```

```
(d) def fun_3(lst,x):  
    newlist = []  
    i = 0  
    while len(lst[i]) > x:  
        newlist.append(lst[i])  
        i = i+1  
    return newlist
```

Q: What is printed by the print statement below? Be careful!

```
x = fun_3(['aaaaa','fffff','q','1111111111','33333'],2)  
print x
```

```
ans = ['aaaaa', 'fffff']
```

4. (25 points)

For this question, you will write two functions that could have been used in “wannabe.py”, the code from class that builds a dictionary that remembers what words follow others in an input text, and then uses that dictionary to generate nonsense text that is the same style.

We’ll use an example to remind you of the dictionary:

Suppose the following is an item in the dictionary:

```
('and', 'Happy', 'day'): ['and', 'but']
```

From this example, we know that ‘context_length’ is 3, and ‘and Happy day’ appears twice in the input text, once followed by ‘and’ and once followed by ‘but’.

Complete the following functions according to their docstring descriptions.

```
(a) def initialize(context_length):
    """ Given positive integer 'context_length', return a tuple
        containing that many empty strings. """

    context = (',',)
    for i in range(context_length-1):
        context = context + (',',)
    return context

another solution a student had:
    return (',',) * context_length
```

```
(b) def update(word_dict, context, word):
    ''' 'word_dict' is a dictionary as described above, 'context' is
        a tuple of words that is the same length as the keys of
        'word_dict', and 'word' is a word that was found to
        follow 'context' in the input text. Update the 'context'
        entry of 'word_dict' with 'word' '''

    if context in word_dict:
        word_dict[context].append(word)
    else:
        word_dict[context] = [word]
```