Week 3: Strings/Style/Top-Down Design

Quick Answer

1. The following code does not work. It crashes. In just a few words, state why it crashes and then rewrite it so that it works as intended.

```python
def f(s):
    t = s
    for i in xrange(len(t)):
        if t[i] == 'Q':
            t[i] = 'R'
    return t
```

1. Very briefly, give two distinct advantages of top-down design using lots of small helper functions.

2. Give two advantages (other than the ones above) of using good style.

3. Very briefly, give one reason why magic numbers can lead to more bugs in your code.

4. True or False. The comment below should be included.

```python
#goes through each letter in the alphabet
for letter in alphabet:
    ...
```

Code Tracing

```python
def t1(s,t):
    for c in s:
        if (c.upper() not in "NO!!"):
            i = t.find(c)
            print(i,s[i],t[i],end=" ")
    t1("net","two")
```
def g(s):
    result = ""
    for i in range(len(s)):
        for j in range(i):
            if (s[j] > s[i]):
                result += s[j] + s[i]
    return result
print(g("aebdc"))

Reasoning Over Code

def f(s, t, n):
    q = ""
    for i in range(len(s)):
        q += t[(i+n)%len(t)]
    return ((len(s) == 2*len(t)) and
    (not s.startswith(t)) and (q == s))

def r(s, t):
    result = ""
    for c in t[::-1]:
        if (c.isdigit()): continue
        result += c + str(s.find(c))
    return (result == "B1C-1A3") and (len(s) == len(t))

Free Response

leastFrequentLetters:

Write the function leastFrequentLetters(s), that takes a string s, and ignoring case (so "A" and "a" are treated the same), returns a lowercase string containing the least-frequent alphabetic letters that occur in s, each included only once in the result and then in alphabetic order.

So: leastFrequentLetters("aDq efQ? FB'daf!!!") returns "be". Note that digits, punctuation, and whitespace are not letters! Also note that seeing as we have not yet covered lists, sets, maps, or efficiency, you are not expected to write the most efficient solution. Finally, if s does not contain any alphabetic characters, the result should be the empty string ("").

wordWrap
Write the function `wordWrap(text, width)` that takes a string of text (containing only lowercase letters or spaces) and a positive integer width, and returns a possibly-multiline string that matches the original string, only with line wrapping at the given width. So `wordWrap("abc", 3)` just returns "abc", but `wordWrap("abc",2)` returns a 2-line string, with "ab" on the first line and "c" on the second line. After you complete word wrapping in this way, only then: All spaces at the start and end of each resulting line should be removed, and then all remaining spaces should be converted to dashes ("-"). so they can be easily seen in the resulting string. Here are some test cases for you:

```python
assert (wordWrap("abcdefgij",4) == "abcd
efgh
ij")
assert (wordWrap("a b c de fg", 4) == "a-b
c-de
fg")
```

### largestNumber

Write the function `largestNumber(text)` that takes a string of text and returns the largest int value that occurs within that text, or None if no such value occurs. You may assume that the only numbers in the text are non-negative integers and that numbers are always composed of consecutive digits (without commas, for example). For example: `largestNumber("I saw 3 dogs, 17 cats, and 14 cows!")` returns 17 (the int value 17, not the string "17"). and `largestNumber("One person ate two hot dogs!")` returns None (the value None, not the string "None"). Hint: for code that does not quite work, you may get more partial credit if you include enough concise comments so we can see what you at least were trying to do.