Practical Questions

CSCA48 – Week 6

Trace the following functions with a variety of input, and describe their functionality in a single sentence.

1.
```
def mystery(n):
    if n == 4:
        result = n
    else:
        result = 2 * mystery(n+1)
    return result
```

2.
```
def mystery(a, b):
    if b < 1:
        return 0
    if b % 2 == 0:
        return mystery(a+a, b/2)
    return mystery(a+a, b/2) + a
```

3.
```
def mystery(s):
    if len(s) == 0:
        result =  “”
    elif len(s) == 1:
        result = s
    elif s[0] == s[1]:
        result = mystery(s[1:])
    else:
        result = s[0] + mystery(s[1:])
    return result
```

4.
```
def mystery(x, y):
    if y == 0:
        result = 0
    else:
        result = x + mystery(x, y-1)
    return result
```
   What does mystery do? Describe in one English sentence.

5. **(Challenge)** (Hint: do part 4 first.)

```
def mystery(x, y):
    if y == 0:
        result =  0
    else:
        result = x + mystery(x, y-1)
    return result


def mystery2(a, b):
    if  b == 0:
        result =  1
    else:
        result = mystery(a, mystery2(a, b-1))
    return result
```

6. **(Challenge)**

```
from math import *
def mystery(a, b, r):
    if b >= a:
        return [] # return empty list
    c = ceil(a + (a-b)*r) - 1
    d = floor(a + (b-a)*r)
    return mystery(c, a, r) + [(a, b)] + mystery(b, d+1, r) + [(c, d)]
```

What is the output of mystery(20, 10, 0.2), mystery(7, 1, 0.5), and mystery(6, 2, 0.6)?

Complete the following functions. All functions must work recursively.

a) def rec_len(L):
'''Return the total number of non-list elements within nested list L.
For example, rec_len([1, 'two', [[], [[5]]]]) should return 3. '''

b) def nest_level(L):
'''Return the maximum "nesting level" of nested list L, that is, how many levels of sub-lists are in L, at the "deepest" point within L.
For example, nest_level([1, [[], [[2]]], 'three']) should return 4 because element 2 is in a list ([2]) within a list ([[2]]) within a list ([[], [[2]]]) within the original list. '''

c) def rev(n):
'''Return the result of reversing the digits in integer n. For example, rev(512) should return 215. '''

d) def add_commas(n):
'''Return a string version of integer n with commas added. For example, add_comma(15866321) should return "15,866,321". '''

e) def flatten(L):
'''Takes a list as a parameter and flattens it: if any items are themselves lists, their contents are retrieved from those nested lists. For example, flatten([1,[2[3,4]],5]) should return [1, 2, 3, 4, 5]. '''

f) def num1s(n):

'''Given a nonnegative int n, yields the number of 1's in its binary representation. For example, suppose n = 17. The binary representation of 17 is 10001, so the function would return 2.

The following facts may be useful:

(1) The rightmost bit of the binary representation of n is 1 if and only if n is odd.

(2) For n > 0, the binary representation of n is: (binary representation of n/2) followed by (the rightmost bit of the binary representation of n).'''

g) Write a function that given a integer K, print the first K rows of Pascal's Triangle. Print each row with values separated by space. For example, K = 5, then it prints

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1

h) The parentheses below line up for each boolean subexpression. Note that the entire body is a single return statement.
Fill in the blanks with **or**, and, **subL**, **seqL**, **subL[1:]**, and **seqL[1:]** so that the code works as advertised. You may use each of them as often as you need to.

```
def is_sub_sequence(subL, seqL):
    '''Return whether the items in list subL appear in list seqL in the same order. A
    list is a sub-sequence of itself, and the empty list if a sub-sequence of every list.
    For examples:
        is_sub_sequence([1, 3], [1, 2, 3, 4]) should evaluate to True.
        is_sub_sequence([3, 1], [1, 2, 3, 4]) should evaluate to False. '''

    return (
            len(subL) == 0 _____________
                (len(seqL) > 0 _____________
                    (is_sub_sequence(_____________, _____________) _____________
                        (subL[0] == seqL[0] _____________
                            is_sub_sequence(_____________, _____________) ) ) ) )
```

i) A word is considered elfish if it contains the letters: e, l, and f in it, in any order. For example, we would say that the following words are elfish: whiteleaf, tasteful, unfriendly, and waffles, because they each contain those letters. Write a function called elfish, given a word and return true if that word is elfish, false otherwise.

j) Write a recursive function called to_words that takes a single *int* parameter and returns a *str* that contains the digits in the *int* in English.
For example, the call
to_words(23561)
would result in the following:
'two three five six one '
Assume the following list of *str*s is defined in the file that contains the function you are writing.
NUMBERS = ["zero", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine"]
Hint: Use the / and % operators.

k) <u>(Challenge)</u>
We define super digit of an integer x using the following rules:

- Iff x has only 1 digit, then its upper digit is x
- Otherwise, the super digit of x is equal to the super digit of the digit-sum of x. Here, digit-sum of a number us defined as the sum of its digits.

For example, super digit of 9875 will be calculated as:

super-digit(9875) = super-digit(9+8+7+5)
= super-digit(29)
= super-digit(2+9)
= super-digit(11)
= super-digit(1+1)
= super-digit(2)
= 2.

Write a function that calculate and return the super digit of P, given two integer n and k. P is created when number n is concatenated k times. That is, if n = 148 and k = 3, so P = 148148148

super-digit(9875) = super-digit(148148148)
= super-digit(1+4+8+1+4+8+1+4+8)
= super-digit(39)
= super-digit(3+9)
= super-digit(12)
= super-digit(1+2)
= super-digit(3)
= 3.

l) (Challenge)

Consider the following puzzle. You have exactly two containers to use: one can hold at most three liters of water and the other can hold at most five liters of water. The containers have no markings on them to tell you how much water they currently contain. You have an infinite supply of water at your disposal with which to fill the containers. Both containers start empty. How can you get exactly four liters of water in the file-liter container?

Write a recursive function that solves this puzzle by printing out all possible solutions. None of your solutions should allow the same state space to be reached more than once (i.e. you should never enter infinite recursion). Note that I do not want you to actually give me a list of moves that represent a solution to the puzzle (that's that computers are for).

Discussion Questions

1) When we create a recursive function that doesn't have or misses its base case, the exception we get is called a 'stack overflow'. Why do we use that name? What does this have to do (if anything) with the stacks we've been learning about in lecture?
2) Can every program we write recursively also be written non-recursively? What about the other way around? Can you come up with a general algorithm to convert from one to the other?
3) Is recursion more or less efficient than iterative code? Can you come up with an example where you could write a recursive and a non-recursive version of the same code? Test them to see which one runs faster (hint: Try some of the easy ones we've been doing).

Logic Question (Challenge)

In the game Tetris, each piece has exactly 4 squares, and each row is 10 squares long. So in theory, it's possible to completely clear the board after your first 5 pieces are placed. How many arrangements of pieces can complete this task? That is, how many ways are there to arrange the classic Tetris pieces into a square that is 10 blocks wide by 2 blocks high?