National University of Singapore
School of Computing
CS1010S: Programming Methodology

Extra Practice 4 Solutions

Help

Yes, we put these functions here so that you can refer to them easily.

```
def sum(term, a, next, b):
    if a > b:
        return 0
    else:
        return term(a) + sum(term, next(a), next, b)

def fold(op, f, n):
    if n == 0:
        return f(0)
    else:
        return op(f(n), fold(op, f, n-1))

def compose(f, g):
    return lambda x: f(g(x))

def thrice(f):
    return compose(compose(f, f), f)
```

Question 1

This question tests you about the left-right evaluation.

```
def new_if(predicate, then, otherwise):
    if predicate:
        then
    else:
        otherwise

def p(x):
    new_if(x > 5, print(x), p(x+1))

p(1) # Infinite print; 'otherwise' part always evaluates first.
```

Question 2

It's time for a simple function nesting! What's the output of the code below?

```
print((lambda x: lambda y: 2*x)(3)(4))
# (lambda x: lambda y: 2*x)(3)(4)
# = (lambda y: 6)(4) = 6
```

Question 3

Let's apply the same idea from Mission 3 :D

```
print(thrice(thrice)(lambda x: x-1)(27))
# thrice(thrice)(lambda x: x-1)(27)
# = thrice(thrice(thrice(lambda x: x-1)))(27)
# = thrice(thrice(lambda x: x-3))(27)
# = thrice(lambda x: x-9)(27)
# = (lambda x: x-27)(27) = 0
```

Question 4

Nested function calls. Can you do it?

```
foo = lambda y: lambda x: x(y)
add1 = lambda x: x+1
# This means foo(y)(x) = x(y) and add1(x) = x+1

print(foo(add1(2))(foo)(add1))
# foo(add1(2))(foo)(add1)
# = foo(3)(foo)(add1)
# = foo(3)(add1)
# = add1(3) = 4
```

Question 5

Nested variable scopes. Can you do it?

```
def foo(x):
    def bar(x, y):
        return lambda y: y(x)
    return lambda y: bar(x, y)
# This means foo(x)(y) = bar(x, y) = lambda y: y(x)
#                                  = lambda z: z(x) <- y is a dummy variable
# In conclusion, foo(x)(y)(z) = z(x)

print(foo(lambda x: x**3)(lambda x: x**2)(lambda x: x)(4))
# foo(lambda x: x**3)(lambda x: x**2)(lambda x: x)(4)
# = (lambda x: x)(lambda x: x**3)(4)
# = (lambda x: x**3)(4) = 64
```

Question 6

Try to do each subquestion within **5 minutes**. You are to use the **`fold`** function in all of the following questions.

1. Define a function **`between`** that takes in a word as an input and returns a new string with "*" placed in between all consecutive words that are identical.
   Sample Tests:

```
>>> between("happy")
'hap*py'
>>> between("oookayy")
'o*o*okay*y'
```

   Solution:

```
def between(word):
    # Define a function that takes in the index and see
    # if word[idx] == word[idx+1] then add a star
    def add_star(idx):
        if word[idx] == word[idx+1]:
            return word[idx] + "*"
        return word[idx]

    return fold(lambda x, y: x + y,                  # op
                lambda x: add_star(len(word)-1-x) \
                          if x != 0 else word[-1],   # f
                len(word)-1)                         # n
```

2. Define a function **`check_vowel`** that takes in a word as an input and returns **`True`** if there is at least one vowel in the word or **`False`** otherwise. This is case-sensitive.
   Sample Tests:

```
>>> check_vowel("qwertyjkl")
True
>>> check_vowel("482jfn")
False
```

   Solution:

```
def check_vowel(word):
    return fold(lambda x, y: x or y,             # op
                lambda x: word[x] in "aeiou",    # f
                len(word) - 1)                   # n
```

3. We have a tuple that contains some integers. Define a function **`largest`** that takes in a tuple and returns the largest integer.
   Sample Tests:

```
>>> largest((4, 2, 6, 2, 1))
6
>>> largest((0, 0, 0, 0, 0))
0
```

   Solution:

```
    def largest(tup):
        return fold(lambda x, y: max(x, y), lambda x: tup[x], len(tup) - 1)
```

Question 7

Given a tuple containing numbers, find the number of combination of numbers that you can use from the tuple that can add up to a target number. Define this function **no_of_ways** that takes in a tuple and a target number, and return the number of ways you can hit this target. You can only use each number once. You do not need to use any higher-order functions for this.

Sample Tests:

```
>>> no_of_ways((4, 2, 5), 8)
0                                  # no way you can make the number 8
>>> no_of_ways((4, 2, 5, 3), 7)
2                                  # either 2 + 5 or 4 + 3 gives 7
>>> no_of_ways((4, 2, 5, 3, 5, 1), 5)
4                                  # either 4 + 1, 2 + 3, 5 or 5 gives 5
```

Solution:

```
# Very similar to the counting change problem
def no_of_ways(tup, num):
    if num <= 0 or tup == ():
        return 0
    elif len(tup) == 1 and tup[0] == num:
        return 1
    return int(tup[0] == num) + no_of_ways(tup[1:], num - tup[0]) \
           + no_of_ways(tup[1:], num)
```

Question 8

Define a function that returns the sum of cubes for integers from 1 to n, using **accumulate**. Recall the definition of **accumulate** below.

$$a_1 = a, a_n \leq b$$

$$\text{accumulate}(\oplus, \text{base}, f, a, \text{next}, b) : (f(a_1) \oplus (f(a_2) \oplus (... \oplus (f(a_n) \oplus \text{base})...)))$$

Solution:

```
def sum_of_cubes(n):
    return accumulate(lambda x, y: x + y,        # op
                      0, lambda x: x**3,         # base, f
                      1, lambda x: x + 1, n)     # a, next, b
```

Question 9

Define the double factorial function using **`accumulate`**.

$$n!! = \begin{cases} (n)(n-2)(n-4)\cdots(4)(2) & \text{if } \frac{n}{2} \in \mathbb{Z}^+ \\ (n)(n-2)(n-4)\cdots(3)(1) & \text{if } \frac{n+1}{2} \in \mathbb{Z}^+ \end{cases}$$

Solution:

```
def double_factorial(n):
    return accumulate(lambda x, y: x*y,                  # op
                      1, lambda x: x,                    # base, f
                      2 - (n % 2), lambda x: x + 2, n)   # a, next, b

# Idea:
# base if definitely 1 when we want to set this is a product
# of a sequence.
# Note that next should be an increment by 2, but now it leaves
# us the variable a, which might seem unintuitive, but the idea
# is just that we want a = 1 when n is odd and a = 2 when n is even.
```

Solution compiled by Russell Saerang.