National University of Singapore
School of Computing
CS1010X: Programming Methodology
Semester II, 2020/2021

Solutions for Recitation 3
Higher Order Functions

Python

1. *lambda* - `lambda` *params:* `expression`
   Creates an anonymous function equivalent to:

   ```
   def function(params):
       return expression
   ```

Definitions

Instructor notes: We go through the problems first before discussing the definitions

The following are two higher-order functions discussed in lecture:

```
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))
```

Note: it is not necessary to memorize these defintions, or even the names of these functions. Definitions of such functions (if they are used) will be given in an Appendix for examinations. What you need to be able to do is to read the definition for such functions and understand what they do and be able to use them.

Problems

1. Evaluate the return values of the following expressions:

   Instructor notes: Highlight that function call requires input arguments to be evaluated first. Hence evaluation proceeds in an left-most, inner-most order.

   (a)
   ```
   x = 2
   def f():
       x = 5
       y = x + 5
       return x + y
   f() + x
   ```

   Answer: 17

   (b)
   ```
   x = 2
   y = 3
   def g(x):
       y = x + 5
       x = 7
       return x + y
   g(y)+y
   ```

   Answer: 18

   (c)
   ```
   x = 4
   def foo(x):
       return x(3)
   foo(lambda x: x+1)
   ```

   Answer: 4

   (d)
   ```
   x = 5
   def bar(x, y):
       return y(x)
   bar(4, lambda x: x**2)
   ```

   Answer: 16

2. Write a function `my_sum` that computes the following sum, for $n \geq 1$:

   $$1 \times 2 + 2 \times 3 + \cdots + n \times (n+1)$$

   Answer:

   ```
   def my_sum(n):
       if n == 1:
           return 2
       else:
           return n * (n + 1) + my_sum(n - 1)
   ```

3. Is the function `my_sum` as defined above a recursive process or an iterative process? What is the order of growth in time and in space?

   Answer: Recursive. Time: O(n), space: O(n).

4. If your answer in Question 2 is a recursive process, re-write `my_sum` as an iterative process. If your answer in Question 2 is an iterative process, re-write `my_sum` as a recursive process. What is the new order of growth in time and space?

   Answer:

   ```
   def my_sum(n):
       sum = 0
       for i in range(1,n+1):
           sum = sum + i * (i + 1)
       return sum
   ```

   Alternative:

   ```
   def my_sum(n):
       sum = 0
       while n != 0:
           sum = sum + n * (n + 1)
           n = n - 1
       return sum
   ```

   Time: O(n), space: O(1).

5. We can also define `my_sum` in terms of the higher-order function `sum`. Complete the definition of `my_sum` below. You cannot change the definition of `sum`; you may only call it with appropriate arguments.

   Instructor notes: Before trying to define `my_sum`, we first need to understand that the higher-order function `sum` computes $t(a)+t(n(a))+t(n(n(a)))+\cdots+t(n(\overset{i}{\cdots}(n(a))))$ where $n(\overset{i}{\cdots}(n(a))) \leq b$ and $(n(\overset{i+1}{\cdots}(n(a)))) > b$.

   And since we are computing $(1\times 2)+(2\times 3)+\cdots+(n\times(n+1))$, it follows to match the terms up, taking care of the ending condition

   ```
   T1: lambda x: x * (x + 1)

   T2: 1

   T3: lambda x: x + 1

   T4: n
   ```

6. Suppose instead we define `my_sum` in terms of the higher-order function `fold`. Complete the definition of `my_sum` below.

   Instructor notes: As before, we first need to understand that the higher-order function fold computes $f(n)\oplus(f(n-1)\oplus(f(n-2))\cdots(f(1)\oplus f(0)))$ where $a\oplus b$ denotes $op(a,b)$

   And since we are computing $(1\times 2)+(2\times 3)+\cdots+(n\times(n+1))$, it follows to match the terms up, taking care of the ending condition.

   Note that strictly following the precedence of operations, it should be written as $(n\times(n+1))+\cdots((3\times 2)+((2\times 3)+(1\times 2))))\ldots)$

   ```
   def my_sum(n):
       return fold(<T1>, <T2>, <T3>)
   ```

```
T1: lambda x, y: x + y

T2: lambda x: (x + 1) * (x + 2)

T3: n - 1
```

In this solution, we are matching $f(0)$ to 1×2, $f(1)$ to 2×3, and so forth, until $f(n-1)$ to $n \times (n+1)$.

Alternative:

```
def my_sum(n):
    return fold(<T1>, <T2>, <T3>)

T1: lambda x, y: x + y

T2: lambda x: x * (x + 1)

T3: n
```

In this solution, because $f(0) = 0$, we are matching $f(1)$ to 1×2, $f(1)$ to 2×3, and so forth, until $f(n)$ to $n \times (n + 1)$.

7. Write an iterative version of `sum`.

   Instructor notes: Ok if no time to discuss this. See if students can figure out a straightforward way to translate recursion to iteration and vice versa.

   Answer:

```
def sum(term, a, next, b):
    total = 0
    while a <= b:
        total = total + term(a)
        a = next(a)
   return total
```

8. **Homework:** Write an iterative version of `fold`.

   Answer:

```
def fold(op, f, n):
    res = f(0)
    curr = 1
    while curr <= n:
        res = op(f(curr), res)
        curr = curr + 1
    return res

# if your answer is significantly different, you may want to check
# boundary cases: fold(op, f, 0)
# order of operations: op(f(n), ... op(f(3), op(f(2), op(f(1) ,f(0))))...)
```