CS 61A Fall 2017

Final Review

Discussion 11: November 29, 2017

1 Mutation

1.1 For each row below, fill in the blanks in the output displayed by the interactive Python interpreter when the expression is evaluated. Expressions are evaluated in order, and expressions may affect later expressions.

```
>>> cats = [1, 2]
>>> dogs = [cats, cats.append(23), list(cats)]
>>> cats
```

```
>>> dogs[1] = list(dogs)
>>> dogs[1]
```

```
>>> dogs[0].append(2)
>>> cats
```

```
>>> dogs[2].extend([list(cats).pop(0), 3])
>>> dogs[3]
```

```
>>> dogs
```

1.2 (Fall 2013) Draw the environment diagram for the folowing code.

```
def miley(ray):
    def cy():
        def rus(billy):
            nonlocal cy
            cy = lambda: billy + ray
            return [1, billy]
        if len(rus(2)) == 1:
            return [3, 4]
        else:
            return [cy(), 5]
    return cy()[1]

billy = 6
miley(7)
```

2 Recursion

2.1 Write a procedure `merge(s1, s2)` which takes two sorted (smallest value first) lists and returns a single list with all of the elements of the two lists, in ascending order. Use recursion.

Hint: If you can figure out which list has the smallest element out of both, then we know that the resulting merged list will have that smallest element, followed by the merge of the two lists with the smallest item removed. Don't forget to handle the case where one list is empty!

```
def merge(s1, s2):
    """ Merges two sorted lists
    >>> merge([1, 3], [2, 4])
    [1, 2, 3, 4]
    >>> merge([1, 2], [])
    [1, 2]
    """
    if ____________________________________________________________________________:


        return s2


    elif ____________________________________________________________________________:


        return s1


    elif ____________________________________________________________________________:


        return ____________________________________________________________________


    else:


        return ____________________________________________________________________
```

2.2 Consider the subset sum problem: you are given a list of integers and a number k. Is there a subset of the list that adds up to k? For example:

```
>>> subset_sum([2, 4, 7, 3], 5)         # 2 + 3 = 5
True
>>> subset_sum([1, 9, 5, 7, 3], 2)
False
>>> subset_sum([1, 1, 5, -1], 3)
False
```

```
def subset_sum(seq, k):


    if ____________________________________________________________________________:


        return False


    elif ____________________________________________________________________________:


        return True


    else:


        return ____________________________________________________________________
```

3 Trees

3.1 Assuming that every value in `t` is a number, define `average(t)`, which returns the average of all the values in `t`. You may not need to use all the provided lines.

```
def average(t):
    """
    Returns the average value of all the nodes in t.
    >>> t0 = Tree(0, [Tree(1), Tree(2, [Tree(3)])])
    >>> average(t0)
    1.5
    >>> t1 = Tree(8, [t0, Tree(4)])
    >>> average(t1)
    3.0
    """
    def sum_helper(t):

        total, count = ____________________________________________________________


        for ____________________________________________________________________:


            ____________________________________________________________________


            ____________________________________________________________________


            ____________________________________________________________________


        return total, count


    total, count = ____________________________________________________________

    return total / count
```

4 Streams

4.1 Write a function `merge` that takes 2 sorted streams `s1` and `s2`, and returns a new sorted stream which contains all the elements from `s1` and `s2`.
Assume that both `s1` and `s2` have infinite length.

```
(define (merge s1 s2)

    (if ___________________________________________________________________

        ___________________________________________________________________

        ___________________________________________________________________))
```

4.2 (Adapted from Fall 2014) Implement `cycle` which returns a stream repeating the digits 1, 3, 0, 2, and 4, forever. Write `cons-stream` only once in your solution!
Hint: `(3+2) % 5 == 0`.

```
(define (cycle start)

    ___________________________________________________________________)
```

5 Generators

5.1 Implement `accumulate`, which takes in an `iterable` and a function `f` and yields each accumulated value from applying `f` to the running total and the next element.

```
from operator import add, mul

def accumulate(iterable, f):
    """
    >>> list(accumulate([1, 2, 3, 4, 5], add))
    [1, 3, 6, 10, 15]
    >>> list(accumulate([1, 2, 3, 4, 5], mul))
    [1, 2, 6, 24, 120]
    """
    it = iter(iterable)


    ____________________________________________________________________________


    ____________________________________________________________________________


    for ____________________________________________________________________:

        ________________________________________________________________________


        ________________________________________________________________________
```

5.2 Write a generator function that yields functions that are repeated applications of a one-argument function `f`. The first function yielded should apply `f` 0 times (the identity function), the second function yielded should apply `f` once, etc.

```
def repeated(f):
    """
    >>> double = lambda x: 2 * x
    >>> funcs = repeated(double)
    >>> identity = next(funcs)
    >>> double = next(funcs)
    >>> quad = next(funcs)
    >>> oct = next(funcs)
    >>> quad(1)
    4
    >>> oct(1)
    8
    >>> [g(1) for _, g in
    ...  zip(range(5), repeated(lambda x: 2 * x))]
    [1, 2, 4, 8, 16]
    """


    g = ____________________________________________________________________

    while True:


        ____________________________________________________________________


        ____________________________________________________________________
```

5.3 Ben Bitdiddle proposes the following alternate solution. Does it work?

```
def ben_repeated(f):
    g = lambda x: x
    while True:
        yield g
        g = lambda x: f(g(x))
```

6 SQL

6.1 You're trying to re-organize your music library! The table `tracks` below contains song titles and the corresponding album. Create another table `tracklist` with two columns: the album and a comma-separated list of all songs from that album in alphabetical order.

```
CREATE TABLE tracks AS
  SELECT "Human" AS title  , "The Definition" AS album UNION
  SELECT "Simple and Sweet", "The Definition"          UNION
  SELECT "Paper Planes"    , "Translations Through Speakers";


CREATE TABLE tracklist AS
  WITH songs(album, total) AS (

    ____________________________________________________________________________

  ),

  _____________________________________________ AS (

    ____________________________________________________________________________

    ____________________________________________________________________________

    ____________________________________________________________________________

  )

  SELECT ____________________________________________________________________________

    WHERE ____________________________________________________________________________;
```

```
sqlite3> SELECT * FROM tracklist ORDER BY album;
The Definition|Human, Simple and Sweet
Translations Through Speakers|Paper Planes
```