
COMPUTER SCIENCE 61A

April 27, 2017

1 Mutable Sequences

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
```

2 Environment Diagram

1. (Fall 2012) Draw the environment diagram.

```
def box(a):
    def box(b):
        def box(c):
            nonlocal a
            a = a + c
            return (a, b)
        return box
    gift = box(1)
    return (gift(2), gift(3))
box(4)
```

3 Object-Oriented Programming

1. Assume these commands are entered in order. What would Python output?

```
>>> class Foo:
...     def __init__(self, a):
...         self.a = a
...     def garply(self):
...         return self.baz(self.a)
>>> class Bar(Foo):
...     a = 1
...     def baz(self, val):
...         return val
>>> f = Foo(4)
>>> b = Bar(3)
>>> f.a

>>> b.a

>>> f.garply()

>>> b.garply()

>>> b.a = 9
>>> b.garply()

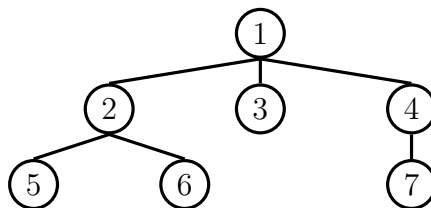
>>> f.baz = lambda val: val * val
>>> f.garply()
```

4 Mutable Linked Lists and Trees

1. Write a recursive function `flip_two` that takes as input a linked list `lnk` and mutates `lnk` so that every pair is flipped.

```
def flip_two(lnk):  
    """  
    >>> one_lnk = Link(1)  
    >>> flip_two(one_lnk)  
    >>> one_lnk  
    Link(1)  
    >>> lnk = Link(1, Link(2, Link(3, Link(4, Link(5))))  
    >>> flip_two(lnk)  
    >>> lnk  
    Link(2, Link(1, Link(4, Link(3, Link(5))))  
    """
```

2. Write a function `flatten` that given a `Tree t`, will return a linked list of the elements of `t`, ordered by level. Entries on the same level should be ordered from left to right. For example, the following tree will return the linked list `<1 2 3 4 5 6 7>`.



```
def flatten(t):
```

5 Scheme

1. Consider the following Scheme tree data abstraction.

```
(define (make-tree label branches) (cons label branches))
(define (label tree) (car tree))
(define (branches tree) (cdr tree))
(define tree 'below-example)
;
;           5
;   +-----+-----+
;   |         |         |
;   6         7         2
;   +---+---+   |   +---+---+
;   |     |     |   |   |   |
;   9     8     1   6   4
;
;                   |
;                   |
;                   3
```

Write a procedure `tree-sums` that takes a tree of numbers (like the one above) and outputs a list of sums from following each possible path from label to leaf.

Hint: You may find the `flatten` procedure helpful.

```
(define (flatten lst)
  (cond ((null? lst) nil)
        ((list? (car lst)) (append (flatten (car lst)) (
                                     flatten (cdr lst))))
        (else (cons (car lst) (flatten (cdr lst))))))

(define (tree-sums tree)

  (if _____

      _____

      (map (lambda (x) _____)

           _____)))
```

```
scm> (flatten '(0 (1) ((2)) (3 ((4))))))
(0 1 2 3 4)
scm> (tree-sums tree)
(20 19 13 16 11)
```

6 Streams

1. (Summer 2012 Final) What are the first five values in the following stream?

```
def my_stream():
    def compute_rest():
        return add_streams(filter_stream(lambda x: x % 2 == 0,
            my_stream()), map_stream(lambda x: x + 2,
            my_stream()))
    return Stream(2, compute_rest)
```

2. (Fall 2014) Define `cycle` that returns a Stream repeating the digits 1, 3, 0, 2, and 4. Hint: $(3+2) \% 5$ equals 0.

```
def cycle(start=1):
    """ Return a stream repeating 1, 3, 0, 2, 4 forever .
    >>> first_k(cycle(), 12) # Return the first 12 elements as
        a list
    [1, 3, 0, 2, 4, 1, 3, 0, 2, 4, 1, 3]
    """
    def compute_rest():
        return _____
    return Stream(_____, _____)
```

7 Generators

1. 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):
```

```
    """
```

```
    >>> [g(1) for _, g in
    ...   zip(range(5), repeated(double))]
    [1, 2, 4, 8, 16]
```

```
    """
```

```
    g = _____
```

```
    while True:
```

```
        _____
```

```
        _____
```

2. 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))
```

8 SQL

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.

```

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

```