FINAL REVIEW

COMPUTER SCIENCE MENTORS 61A

April 24 to April 26, 2017

1 What Would Python Print? Iterators

1.
```
class SkipIterator:
    def __init__(self, rng, n):
        self.obj = rng
        self.skip = n

    def __iter__(self):
        return self

    def __next__(self):
        result = self.obj.curr
        self.obj.curr += self.skip
        return result

class SkippedNaturals:
    def __init__(self):
        self.curr = 0
        self.skip = 1

    def __iter__(self):
        return SkipIterator(self, self.skip)
```

Expression	Interactive Output

`p = SkippedNaturals()`
`twos = iter(p)`
`p.skip = p.skip + 1`
`threes = iter(p)`
`next(twos)`	

`next(twos)`	
`next(threes)`	
`next(threes)`	

Solution:

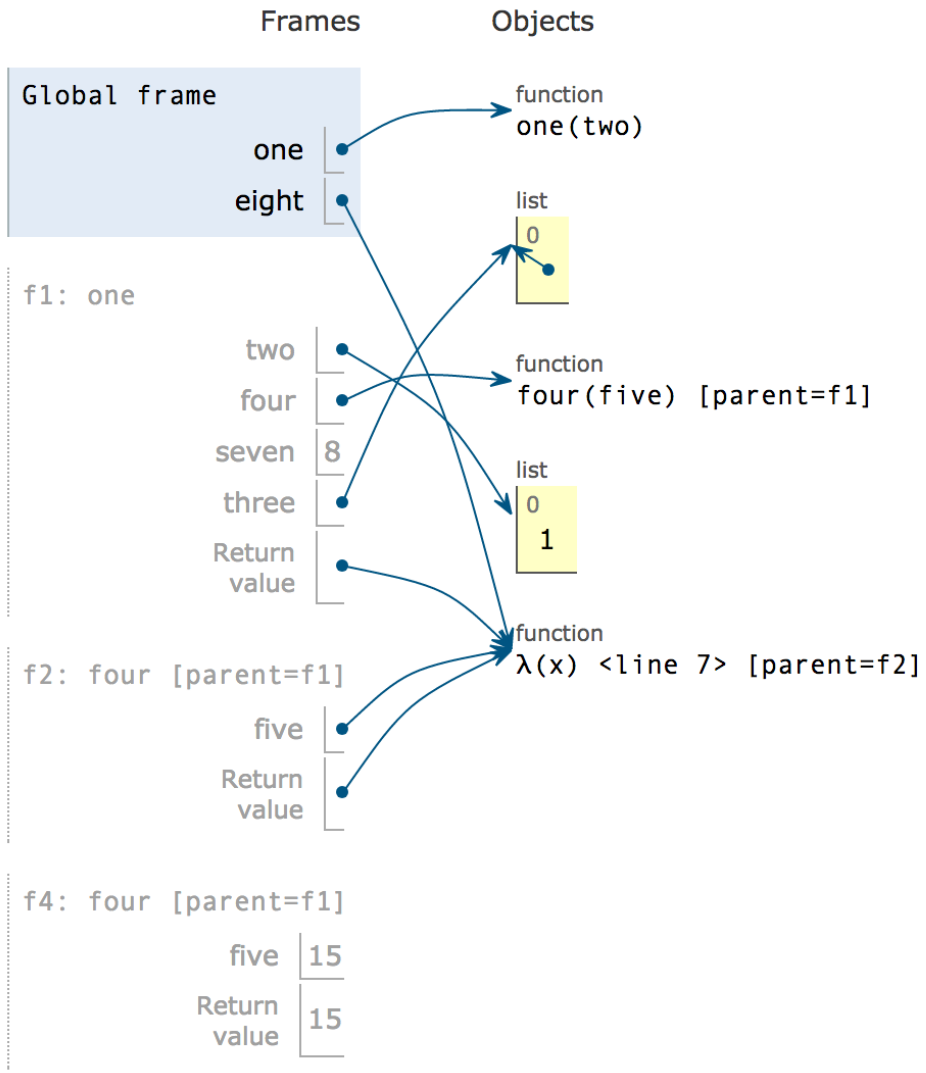
0
1
2
4

2 Environment Diagrams

2. Draw the environment diagram for the following code snippet:

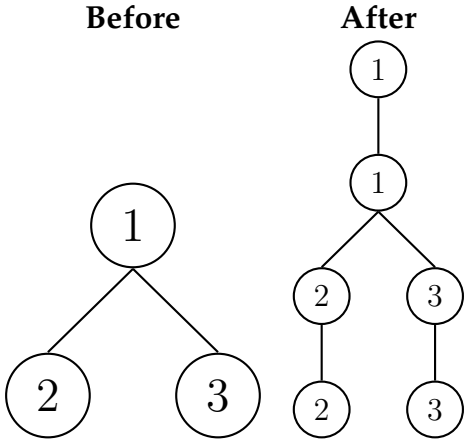
```
def one(two):
    three = two
    def four(five):
        nonlocal three
        if len(three) < 1:
            three.append(five)
            five = lambda x: four(x)
        else:
            five = seven + 7
        return five
    two = two + [1]
    seven = 8
    return four(three)

eight = one([])
print(eight(9))
```

Solution: https://goo.gl/d71WTd



3 Recursive Data Structures

3. DoubleTree hired you to architect one of their hotel expansions! As you might expect, their floor plan can be modeled as a tree and the expansion plan requires doubling each node (the patented double tree floor plan). Here's what some sample expansions look like:



Fill in the implementation for `double_tree`.

```
def double_tree(t):
    """
    Given a tree, return a new tree where entries appear
    twice.
    >>> double_tree(Tree(1))
    Tree(1, [Tree(1)])
    >>> double_tree(Tree(1, [Tree(2), Tree(3)]))
    Tree(1, [Tree(1, [Tree(2, [Tree(2)]),
                      Tree(3, [Tree(3)])
                     ])
            ])
    """
```

Solution:

```
    if t.is_leaf():
        return Tree(t.label, [Tree(t.label)])
    else:
        dbl_branches = [double_tree(c) for c in t.branches]
            return Tree(t.label,
                        [Tree(t.label, dbl_branches)])
```

4. Fill in the implementation of `double_link`.

```
def double_link(lst):
    """
    Using mutation, replaces the second in each pair of items
    with the first. The first of each pair stays as is.
    >>> double_link(Link(1, Link(2, Link(3, Link(4)))))
    Link(1, Link(1, Link(3, Link(3))))
    >>> double_link(
            Link('c', Link('s', Link(6, Link(1, Link('a'))))))
        )
    Link('c', Link('c', Link(6, Link(6, Link('a')))))
    """
    if ____________________________________________:
        return ____________________________________
    ____________________________________________
    ____________________________________________
    return ____________________________________
```

Solution:

```
    if lst is Link.empty or lst.rest is Link.empty:
        return lst
    lst.rest.first = lst.first
    double_link(lst.rest.rest)
    return lst
```

5. Fill in the implementation of `shuffle`.

```
def shuffle(lst):
    """
    Swaps each pair of items in a linked list.
    >>> shuffle(Link(1, Link(2, Link(3, Link(4)))))
    Link(2, Link(1, Link(4, Link(3))))
    >>> shuffle(
            Link('s', Link('c', Link(1, Link(6, Link('a'))))))
        )
    Link('c', Link('s', Link(6, Link(1, Link('a')))))
    """
    if _____________________________________________________
        return _____________________________________________
    new_head = lst.rest
    lst.rest = _________________________________________________
    ____________________________________________________________
    return _____________________________________________________
```

Solution:

```
    if lst == Link.empty or lst.rest == Link.empty:
        return lst
    new_head = lst.rest
    lst.rest = shuffle(new_head.rest)
    new_head.rest = lst
    return new_head
```

4 Scheme

6. Write a Scheme function insert that creates a new list that would result from inserting an item into an existing list at the given index. Assume that the given index is between 0 and the length of the original list, inclusive.

```
(define (insert lst item index)




)
```

Solution:

```
(define (insert lst item index)
  (if (= index 0)
    (cons item lst)
    (cons (car lst) (insert (cdr lst) item (- index 1))))
)
```

Extra: Write this as a tail recursive function. Assume append is tail recursive.

Solution:

```
(define (insert lst item index)
  (define (helper lst index so-far)
    (if (or (null? lst) (= index 0))
      (append so-far (cons item lst))
      (helper (cdr lst) (- index 1)
              (append so-far (list (car lst))))
    )
  )
  (helper lst index nil)
)
```

5 Interpreters

7. Circle the number of calls to `scheme_eval` and `scheme_apply` for the code below.

```
(define (square x) (* x x))
(+ (square 3) (- 3 2))
```

Calls to `scheme_eval` (circle one)	2	5	14	24
Calls to `scheme_apply` (circle one)	1	2	3	4

Solution: 14 for eval, 4 for apply.

6 Recursive Select in SQL

8. Create a `mod_seven` table that has two columns, a number from 0 to 100 and then its value mod 7.
   Hint: You can create a table first with all of the initial data you will build from, and then build the `mod_seven` table.

Solution:

```
with
    base(n) as (
        select 0 union
        select n+1 from base where n+1<7
    ),
    mod_seven (n, value) as (
        select n, n from base union
        select n+7, value from mod_seven where n+7<=100
    )
select * from mod_seven;

ALTERNATIVE SOLUTION WITH MODULO OPERATOR
with
    mod_seven (n, value) as (
        select 0, 0 union
        select n+1, (n+1)%7 from mod_seven where n<100
     )
select * from mod_seven;

ALTERNATIVE SOLUTION WITH ONE TABLE
(This could be a pre-step to approaching the original
   solution.)
with
    mod_seven (n, value) as (
        select 0, 0 union
        select 1, 1 union
        select 2, 2 union
        select 3, 3 union
        select 4, 4 union
        select 5, 5 union
        select 6, 6 union
        select n+7, value from mod_seven where n+7 <= 100
    )
select * from mod_seven;
```

7 Iterators, Generators, and Streams

9. Write a generator that will take in two iterators and will compare the first element of each iterator and yield the smaller of the two values.

```
def interleave(iter1, iter2):
    """
    >>> gen = interleave(iter([1, 3, 5, 7, 9]),
                         iter([2, 4, 6, 8, 10]))
    >>> for elem in gen:
    ...     print(elem)
    ...
    1
    2
    3
    4
    5
    6
    7
    8
    9
    """
```

Solution:

```
t1, t2 = next(iter1), next(iter2)
while True:
    if t1 > t2:
        yield t2
        t2 = next(iter2)
    else:
        yield t1
        t1 = next(iter1)
```

10. **Stream Supreme**

(a) You and your friends are preparing for the 61A final by streaming lectures. You get tired and want to take a rest from studying but first you realize you are hungry so you check the refrigerator. You notice that **every other food** in there is stale! Write a function that takes in a list of foods and outputs a stream that contains all your stale food. **Bonus:** Count all the puns in this question!

```
def stale_foods(foods):
```

Solution:

```
def stale_foods(foods):
  if foods is Link.empty or foods.rest is Link.empty:
    return Stream.empty
  return Stream(foods.first, lambda: stale_foods(foods.
     rest.rest))
```

(b) Can you magically find a way to have infinite food? Find a way to cycle through the foods so that when the last food is exhausted, the stream loops back to the first food.

```
def food_stream(foods):
```

Solution:

```
def food_stream(foods):
  def compute_rest():
    curr = foods
    while curr.rest is not Link.empty:
      curr = curr.rest
      curr.rest = foods.first
    return food_stream(foods)
  return Stream(foods.first, compute_rest)
```

Solution:

```
# Alternate:
def food_stream(foods):
  def exhaust_link(lnk):
    if lnk is Link.empty:
      return exhaust_link(foods)
    return Stream(lnk.first, lambda: exhaust_link(lnk.rest)
      )
  return exhaust_link(foods)
```