**MOCK EXAM *for* MIDTERM 3 SOLUTIONS**

# MOCK MIDTERM 3

Computer Science 61A . November 22, 2015 . alvinwan.com/cs61a

- You have 2 hours to complete the exam.
- The exam is closed book, closed notes, closed computer, closed calculator, except one hand-written 8.5" × 11" crib sheet of your own creation
- Mark your answers ON THE EXAM ITSELF. If you are not sure of your answer you may wish to provide a brief explanation.

| | |
|---|---|
| First Name | Alvin |
| Last Name | Wan |
| SID | |
| Email (...@berkeley.edu) | |
| Login (e.g., cs61a-ta) | |
| TA & section time | |
| Name of person to your left | |
| Name of person to your right | |
| *All the work on this exam is my own.* **(please sign)** | |

**0. (0 Points)** On a scale of Turkey to DeNero, how do you feel? Turkey :( _____:/_____ :) DeNero!

**MOCK EXAM *for* MIDTERM 3 SOLUTIONS**

## 01. (6 Points) PROPER TURKEYS ONLY

For each of the expressions in the table below, write the output displayed by the interactive Python interpreter when the expression is evaluated. **The output may have multiple lines**. Expressions are evaluated in order, and **expressions may affect later expressions**.

Whenever the interpreter would report an error, write `Error`. If execution would take forever, write `Forever`. **`f(*[1, 2])` is the same as `f(1, 2)`, and popping an empty list causes an `IndexError`.** Assume that you have started Python 3 and executed the following statements:

```
class Turkeyland:

    turkeys = ['premium', 'fresh', 'young', 'cage-free', 'grain-fed',
               'certified humane']

    def breed(self, turkey, get_next):
        turkeys = Turkeyland.turkeys
        while turkeys and turkey != 'certified humane':
            yield get_next(turkey)
            turkey = turkeys.pop(0)

    def dinner(self, candidate, dinner):
        [food for food in self.breed(self.turkeys.pop(), dinner)]
        return candidate
```

Source of labels deemed "false advertising" and "meaningful": NPR Article for Turkey labels

| Expression | Interactive Output |
|---|---|
| `good = Turkeyland()`<br>`bad = Turkeyland()`<br>`turkeys = list(bad.breed(`<br>`  'false advertising:', print))` | `false advertising:`<br>`premium`<br>`fresh`<br>`young`<br>`cage-free`<br>`grain-fed` |
| `Turkeyland.turkeys.append(`<br>`  'Animal Welfare approved')`<br>`print('look for:',`<br>`  good.dinner(good.turkeys[0],`<br>`  bad.dinner))` | `look for: Animal Welfare approved` |
| `print(*turkeys) #bad turkeys=>errors!`<br>`bad.dinner(turkeys[0], bad.dinner)` | `None None None None None None`<br>`IndexError (or Error)` |

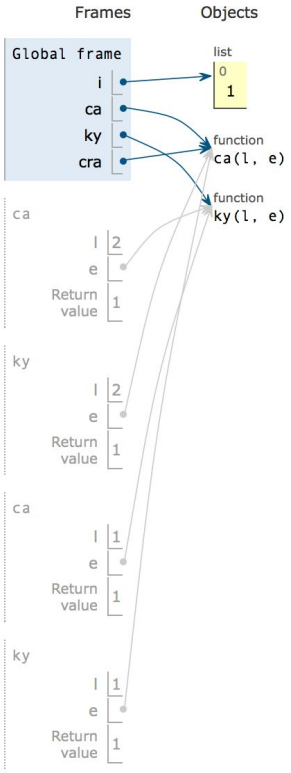
**MOCK EXAM *for* MIDTERM 3 SOLUTIONS**

## 02. (10 Points) CALE RECURSION

(a) **(8 points)** Fill in the environment diagram that results from executing the code below until the entire program is finished, an error occurs, or all frames are filled. You may not need to use all of the spaces or frames.

A complete answer will:

- Add all missing names and parent annotations to all local frames.
- Add all missing values created or referenced during execution.
- Show the return value for each local frame.




http://goo.gl/1IwcNp

```
nums = [0]
def ca(l, e):
    try:
        return e(l, ca)
    except IndexError:
        return ky(3*l+1, ca)

def ky(L, e):
    print(L)
    if L == 1:
        return int(L)
    nums[0] = L // 2
    return e(nums[L % 2], ky)

ca(2, ky)
cra = ca
```

(b) **(1 point)** In a sentence, describe the *printed output* of `cra(n, ky)` where integer n > 0.

hailstone

(c) **(+1 point)** Assume tail recursive frames are closed in Python. Express the total number of frames *open* just before the first return statement is reached, as a function n where integer n > 0 passed to `cra` in `cra(n, ky)`, using big-O notation.

O(1), as both ca and ky are tail recursive.

## 03. (10 Points) EXSTREAM

---

(a) **(4 points)** Complete `filtonacci`, which generates a fibonacci stream containing only integers that satisfy the given `filter`.

```
; >>> (stream (lambda (x) (= 0 (modulo x 2))))
; >>> (slice stream 0 10)
; (0 8 34 144 610 2584 10946 46368 196418 832040)

(define (filtonacci-stream pred a b)

  (if (pred a)

    (cons-stream a (filtonacci-stream pred b (+ a b)))

    (filtonacci-stream pred b (+ a b)))


(define (filtonacci pred) (filtonacci-stream pred 0 1))
```

(b) **(6 points)** Complete `cross-zip`, which zips two lists together. The beginning of list a will be joined with the end of list b, with list a's order maintained. Assume that the lists a and b will always be of the same length. See the samples below for expected behavior.

```
; >>> (cross-zip '(1 2 3) '(4 5 6))
; ((1 6) (2 5) (3 4))

(define (cross-zip-helper a b)

  (if (null? a) (cons nil b)

    (let ((rv (cross-zip-helper (cdr a) b)))

      (cons

            (cons (list (car a) (car (cdr rv))) (car rv))

            (cdr (cdr rv))))))

(define (cross-zip a b) (car (cross-zip-helper a b))) ; <= why (car...)?

; Explanation: cross-zip-helper always returns a list, where the first
element is the result we want. The rest of that list is the rest of b.
```

---

## 04. (14 Points) I-I-I-ITERATE!

(a) **(6 points)** Write `reverse_link_gen`, a generator that traverses a linked list in reverse.

```
def reverse_link_gen(link):
    """A generator that yields a linked list in reverse order.
    >>> gen = reverse_link_gen(Link(1, Link(2, Link(3, Link(4, Link(5))))))
    >>> for x in gen:
    ...     print(x)
    5
    4
    3
    2
    1
    """
    if link:

        return

    for x in reverse_link_gen(link.rest):

        yield x

    yield link.first
```

(b) **(8 points)** Create an "amorphous" iterator, which never terminates and instead assumes the identity of the next iterator once its current iterator has terminated. This iterator will take a series of other **iterables** upon creation. **Note**: Ensure that your code still works if an iterator raises `StopIteration` on the first iteration. In the doctest below, note the empty string. You may assume at least one of the iterables is not empty.

```
>>> amIter = AmorphousIterator('abc', (1, 2, 3), '', [(), [], {}])
>>> results = []
>>> for i, a in enumerate(amIter):
...     if i > 11:
...         break
...     results.append(a)
>>> print(*results)
a b c 1 2 3 () [] {} a b c
```

## MOCK EXAM *for* MIDTERM 3 SOLUTIONS

```
class AmorphousIterator:

    def __init__(self, *iterables):
        self.iterables = list(iterables)
        self.get_iterator = self.cycle()
        self.iterator = next(self.get_iterator)

    def cycle(self):
        for i in self.iterables:
            yield iter(i)
            self.iterables.append(i)

    def __next__(self):
        try:
            return next(self.iterator)
        except StopIteration:
            self.iterator = next(self.get_iterator)
            return next(self)

    def __iter__(self):
        return self
```