CS 61A Structure and Interpretation of Computer Programs

Summer 2014 Midterm 2

INSTRUCTIONS

- You have 2 hours to complete the exam.
- The exam is closed book, closed notes, and closed electronics, except two 8.5" × 11" cheat sheets, and The Environment Diagram Rules.
- Mark your answers ON THE EXAM ITSELF. Answers outside of the space allotted to problems will *not* be graded. If you are not sure of your answer you may wish to provide a *brief* explanation.

Full name	
SID	
Login	
TA & section time	
Name of the person to your left	
Name of the person to your right	
All the work on this exam is my own. **(please sign)**	

0. (1 points) Your thoughts?

1. **(8 points) What will Python output?**

Include all lines that the interpreter would display. If it would display a function, then write Function. If it would cause an error, write Error. Assume that you have started Python 3 and executed the following. **These are entered into Python exactly as written.**

```
class Pet:
    color = "Red"
    name = "Clifford"
    def __init__(self, num_legs):
        print("A new pet!")
        self.num_legs = num_legs
    def sleep():
        print("Zzzz")

class RubberDuck(Pet):
    color = "Yellow"
    def __init__(self):
        self.voice = print("Quack")
        Pet.name = "Daisy"
        name = "Daffy"
        self.num_legs = Pet(0).num_legs
    def debug(self):
        print("What is wrong?")
        return self.voice
```

Expression	Interactive Output
`print("Ducks are cool!")`	Ducks are cool!
`p = Pet(4)`	
`p.self.name`	
`p.sleep()`	
`q = RubberDuck()`	
`p.name + q.name`	
`print(q.debug())`	

2. (12 points) Environment Diagrams

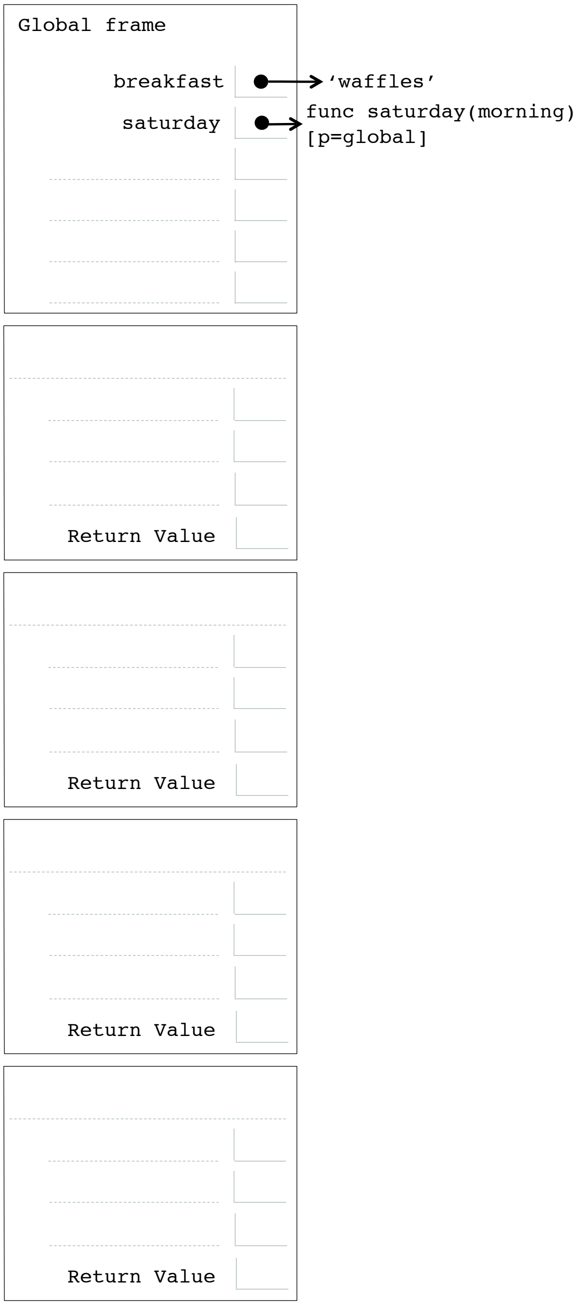
(a) (6 pt) Saturday Morning

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.* You may want to keep track of the stack on the left, but this is not required.

A complete answer will:

- Add all missing names, labels, and parent annotations to all local frames.
- Add all missing values created during execution.
- Show the return value for each local frame.
- The first function created by `lambda` should be labeled λ_1, the next one should be λ_2, and so on.

```
breakfast = 'waffles'
def saturday(morning):
    def breakfast(cereal):
        nonlocal breakfast
        breakfast = cereal
    breakfast(morning)
    return breakfast
saturday(lambda morning: breakfast)('cereal')
```



Stack

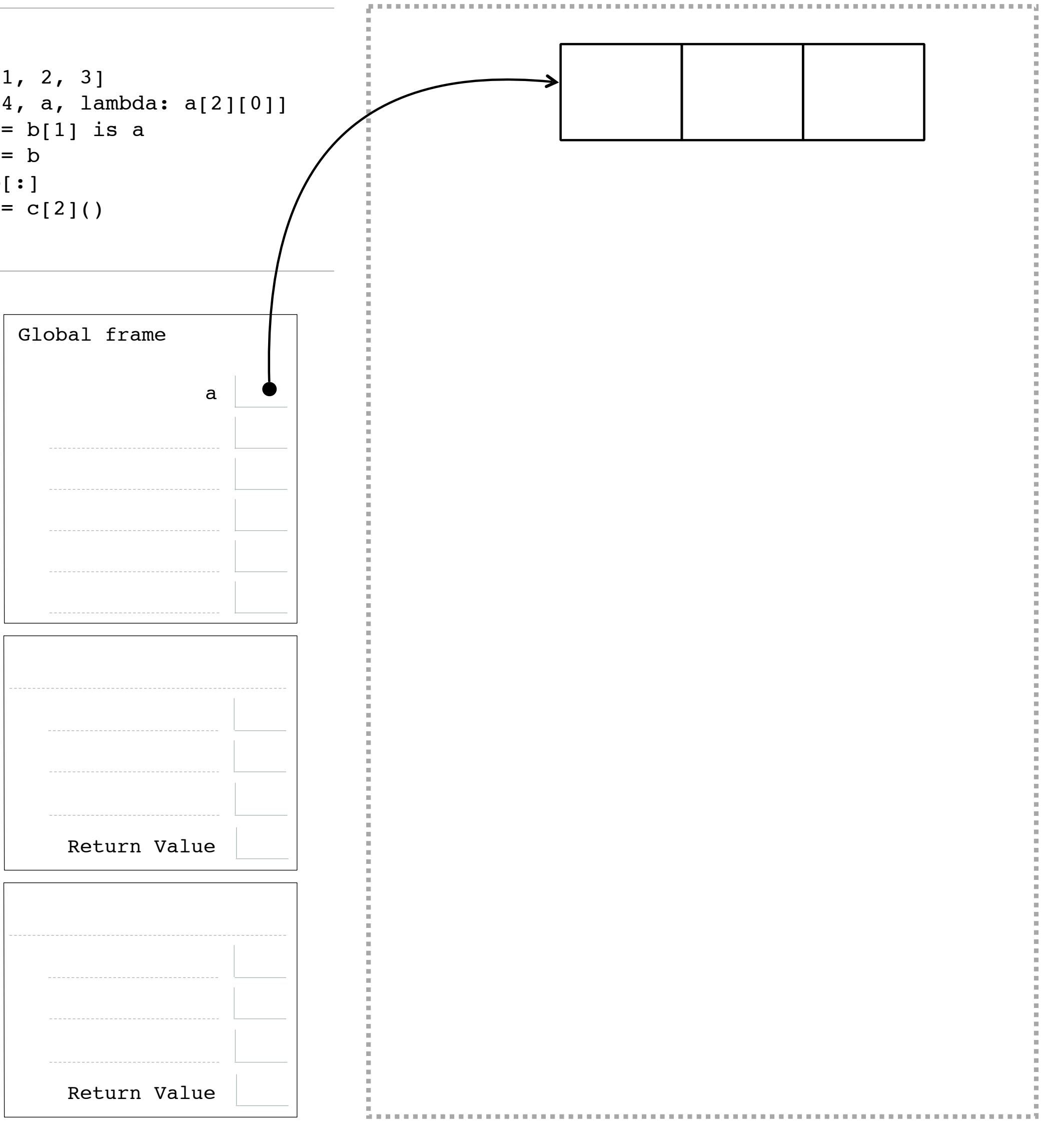
global

(b) (6 pt) Box and Pointer

Fill in the enviroment 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, labels, and parent annotations to all local frames.
- Add all missing values created during execution. *This may include more box-and-pointer diagrams.*
- Show the return value for each local frame.
- The first function created by `lambda` should be labeled λ_1, the next one should be λ_2, and so on.

```
a = [1, 2, 3]
b = [4, a, lambda: a[2][0]]
a[0] = b[1] is a
a[2] = b
c = b[:]
a[1] = c[2]()
```



3. (5 points) Scanning

We all know the higher order functions `map`, `filter`, and `reduce`. Today we're going to talk about their not-quite-so-famous fourth sibling, `scan`. Scan is like `reduce`, only instead of accumulating the result into a single value, `scan` returns a list that contains all the intermediate values in reducing the list.

Cross out lines from the implementation of the `scan` function below so that all doctests pass **and the implementation contains as few lines of code as possible**. You may want to look at the return statement first. **Do not cross out any docstrings or doctests.**

```
def scan(f, lst, start):
    """Returns a list containing the intermediate values of reducing the list.

    >>> scan(add, [1, 2, 3, 4], 0)
    [1, 3, 6, 10]
    >>> scan(mul, [3, 2, 1, 0], 10)
    [30, 60, 60, 0]
    """

    start = []

    start = 0

    accumulated = f(start)

    accumulated = start

    def closure(item):

        nonlocal accumulated

        nonlocal start

        accumulated = f(item)

        accumulated += f(item)

        accumulated = f(accumulated, item)

        accumulated += f(accumulated, item)

        return accumulated

        return start + accumulated

        return item + accumulated

    return list(map(f(lst)))

    return list(map(f, lst))

    return list(map(closure(lst)))

    return list(map(closure, lst))
```

4. (4 points) What would Python output

Include all lines that the interpreter would display. If it would display a function, then write Function. If it would cause an error, write Error. Assume that you have started Python 3 and executed the following. **These are entered into Python exactly as written.**

```
class SkipIterator:
    """Iterates over a range starting from the beginning and
    skipping every nth element.
    """
    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:
    """Iterable class for positive integers. """
    def __init__(self):
        self.curr = 0
        self.skip = 1

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

Expression	Interactive Output
`print("Skipping Rope")`	Skipping Rope

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

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

5. (3 points) Interpretation

Select which function(s) you would have to modify in order to add the new syntax features in Calculator. **For full credit, you must justify your answers with at most two sentences.**

(a) (1 pt) `=` (equality checker) – e.g. `(= 3 1)` returns False

`calc_eval`   `calc_apply`   Both   Neither

Justification:

(b) (1 pt) `or` – e.g. `(or (= 5 2) (= 2 2) (\ 1 0))` returns True

`calc_eval`   `calc_apply`   Both   Neither

Justification:

(c) (1 pt) Creating and calling lambdas (Assume `define` has been implemented.) – e.g.

```
(define square (lambda (x) (* x x)))
(square 4)
```

`calc_eval`   `calc_apply`   Both   Neither

Justification:

6. (5 points) Waldo's Revenge Scheme

Write wheres-waldo, a scheme procedure which takes in a scheme list and outputs the index of waldo if the symbol `waldo` exists in the list. Otherwise, it outputs the symbol `nowhere`.

```
STk> (wheres-waldo '(moe larry waldo curly))
2
STk> (wheres-waldo '(1 2))
nowhere
```

```
(define (wheres-waldo lst)
  (cond ((null? lst) 'nowhere)

        (_____________________________________________)

        (else

          (let ((found-him _____________________________________________ ))

            (if (equal? 'nowhere found-him)


                _____________________________________________

                _____________________________________________ )))))
```

7. (7 points) Generatree

Here's an implementation of a Binary Search Tree

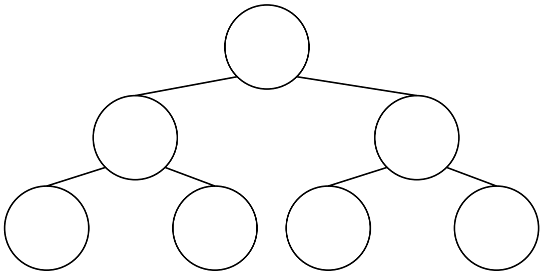
```
class BST:
    def __init__(self, datum, left=None, right=None):
        self.datum = datum
        self.left = left
        self.right = right
```

(a) (1 pt) Draw A Tree

Use the diagram below to reflect the tree generated by the following line:

```
BST(10, BST(5, BST(1)), BST(42))
```

You may not need to use all of the circles.



(b) (6 pt) 3 .. 2 .. 1 - Generate Paths!

Now let's add a `paths` method to the BST class. It will return a generator that yields all of the paths from the root of the tree to a leaf. Each path is represented as a list containing the individual datums.

```
def paths(self):
    """Return a generator for all of the paths from the root to a leaf.

    >>> tree = BST(10, BST(5, BST(1)), BST(42))
    >>> gen = tree.paths()
    >>> next(gen)
    [10, 5, 1]
    >>> for path in gen:
    ...   print(path)
    ...
    [10, 42]
    """

    if not self.right and not self.left:

        ______________________________________________________

    if ______________________________________________________

        for ______________________________________________________

            ______________________________________________________

    if ______________________________________________________

        for ______________________________________________________

            ______________________________________________________
```

8. (5 points) Dealer always wins

We want to play a card game and we must evenly deal out all of the cards to each player. We have a linked list (`Link`) of cards. In this case, cards are represented as numbers.

`deal_deck` returns a Python list of linked lists of cards for each player (reverse order of how the cards were dealt—older cards on the bottom) and a linked list of the extra cards (in the original order).

Do not call the Link constructor.

```
def deal_deck(linked_list, num_of_players):
    """Deals out a deck of cards.

    >>> deck = Link(1, Link(2, Link(3, Link(4, Link(5, Link(6, \
    Link(7, Link(8, Link(9, Link(10))))))))))
    >>> list_of_cards, remainder = deal_deck(deck, 4)
    >>> list_of_cards
    [Link(5, Link(1)), Link(6, Link(2)), Link(7, Link(3)), Link(8, Link(4))]
    >>> remainder
    Link(9, Link(10))
    """
    # Create a list containing each player's hand.
    hands = [Link.empty for i in range(num_of_players)]
    # Give each player the right number of cards.
    for i in range(len(linked_list)//num_of_players):

        # For each player

        for ______________________________________________________

            linked_list, card = linked_list.rest, linked_list

            # Put the card in the player's hand

            ______________________________________________________

            ______________________________________________________

    return ______________________________________________________
```

9. (3 points) (Extra Credit) Social Implications

(a) (1 pt) Describe what software rot is.

(b) (2 pt) Modern cryptography methods are mathematically sound. Give two ways that an attacker could still steal information.