CS 61A Structure and Interpretation of Computer Programs

Summer 2014 FINAL

INSTRUCTIONS

- You have 3 hours to complete the exam.
- The exam is closed book, closed notes, and closed electronics, except three 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. (2 points) Your thoughts? What's been fun? What are you grateful for?

1. (18 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 Cat:
    name = "meow"
    def __init__(self, fish):
        self.fish = fish

    def __iter__(self):
        while len(self.fish) > 0:
            yield lambda: self.fish[0].getname()
            self.fish = self.fish[1:]

class SuperCat(Cat):
    def __init__(self):
        self.lives = 9

    def __next__(self):
        if len(self.fish) == 0:
            raise StopIteration
        return self.fish

class Fish:
    def __init__(self, name):
        self.name = name

    def __len__(self):
        fish = ['fish1', 'fish2', 'fish3']
        return len(self.fish) + 2

    def getname(self):
        print(self.name)
```

Expression	Interactive Output
`print('Cats are cool!')`	Cats are cool!

`dory = Fish('Dory')`
`marlene = Fish('Marlene')`
`dory.name`	

`dari = Cat([dory, marlene, Fish('Nemo')])` `print(dari.fish[2].getname())`	
`fishes = iter(dari)` `next(fishes)`	
`pusheen = SuperCat()` `pusheen.fish`	
`Cat.__init__(pusheen, dari.fish)` `next(pusheen).getname()`	
`for a in pusheen:` `  a()`	
`dari.getname = Fish.getname` `dari.getname(dory)`	
`print(len(pusheen.fish))`	
`print(Fish.__len__(pusheen))`	

Expression	Interactive Output

`lst1 = [i for i in range(1, 3)]`
`lst2 = list(range(4, 6))`
`lst1 + lst2`	

`x = [[lst1], [lst2]][0]` `x[0] is lst1`	
`y = lambda: lambda lst1: lambda: lst1` `y()(x)() is lst1`	
`lst3 = lst1.append(lst2)` `print(lst3)`	
`lst1`	
`lst1[2] is lst2`	
`lst1[:1]`	
`lst1[2:] is lst1[2:]`	
`lst2[1]`	
`lst1[____] + lst2[____] + \` `lst1[____] + lst2[____]`	[4, 2, 5]

2. (8 points) Alakazam!

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.

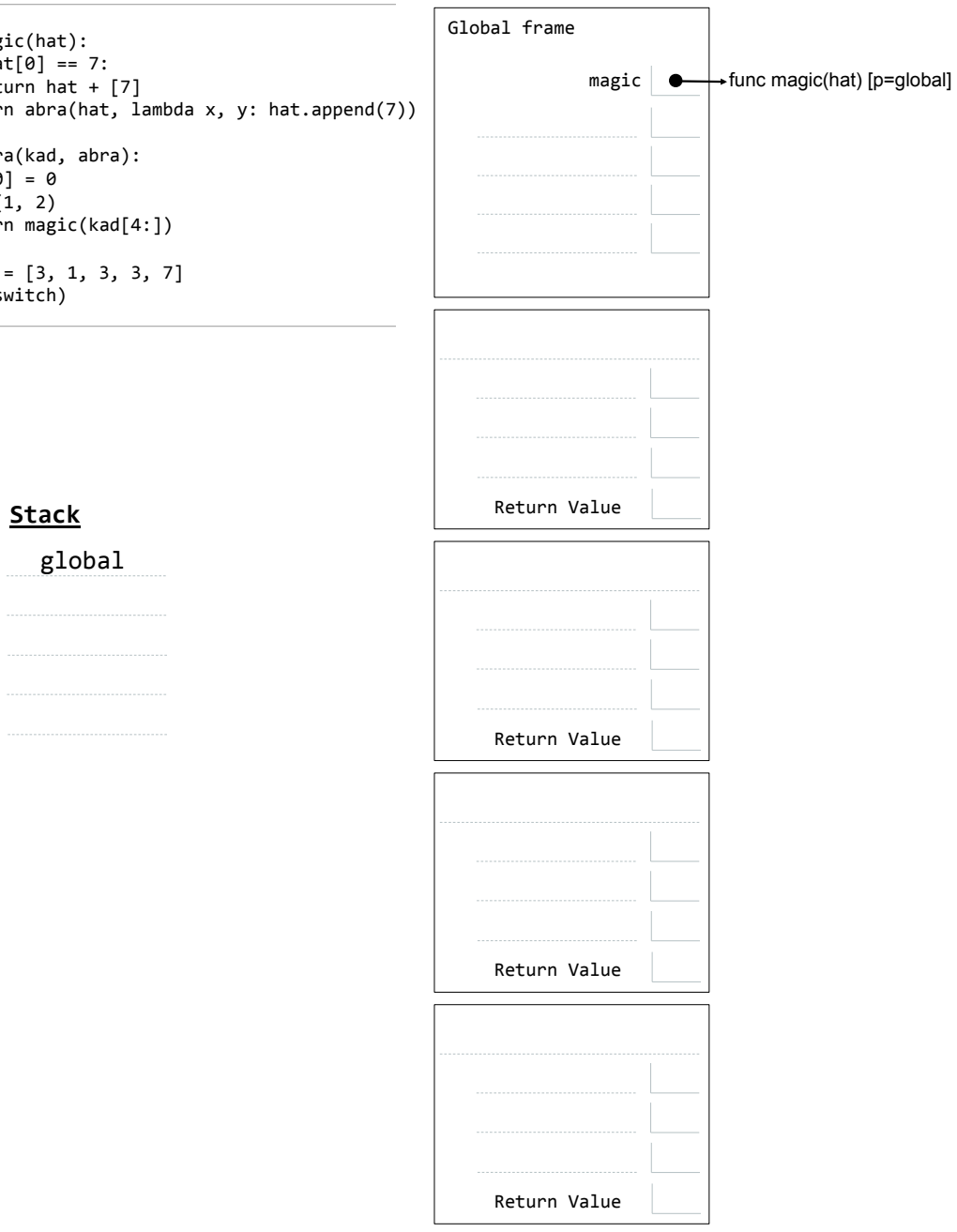
A complete answer will:

- 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.

```
def magic(hat):
  if hat[0] == 7:
    return hat + [7]
  return abra(hat, lambda x, y: hat.append(7))

def abra(kad, abra):
  kad[0] = 0
  abra(1, 2)
  return magic(kad[4:])

switch = [3, 1, 3, 3, 7]
magic(switch)
```



3. (4 points) Counting Song

Let's write a song with generators! The song goes like this: `1, 1 2 1, 1 2 3 2 1 ...`. Write `counting_song`, a function which yields each line of the song represented as a list (see the doctests).

There are multiple ways to solve this.

```
def counting_song():
    """
    >>> cs = counting_song()
    >>> next(cs)
    [1]
    >>> next(cs)
    [1, 2, 1]
    >>> next(cs)
    [1, 2, 3, 2, 1]
    >>> next(cs)
    [1, 2, 3, 4, 3, 2 ,1]
    """

    song = [1]
    while True:

        yield ____________________________________________________________

        ____________________________________________________________________
```

4. (5 points) Final tale of Waldo

Write `waldo-tail`, a *tail-recursive* version of the `wheres-waldo` function you saw on Midterm 2. As a reminder, `waldo-tail` is a Scheme procedure that 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> (waldo-tail '(moe larry waldo curly))
2
STk> (waldo-tail '(1 2))
nowhere

(define (waldo-tail lst)

  (define (helper ____________________________________)

    (cond ((_________________________________________) 'nowhere) ; Base Case

          (_____________________________________________________) ; Base Case

          (else _____________________________________________)))

  (helper _____________________________________________))
```

5. (6 points) Guess-Tree

Many popular guessing games have the following format: We start with a single question, such as "Is it an animal?" If the answer is "yes", we ask a new question assuming what we're guessing is an animal ("Is it a dog?"). If the answer is "no", we ask a new question assuming what we're guessing is not an animal ("Is it a place?"). We keep repeating this process until we are ready to make a guess.

One possible representation of this game is using nested dictionaries. The dictionary has one key - the original question. Its value is another dictionary with at MOST two keys: a 'yes' key and a 'no' key (either of those keys may be missing). Each of those leads to another dictionary, and so on until we hit "Success!" or run into an empty dictionary.

Another possible representation is as a `BinaryTree`. Your task is to write `make_guess_tree`, which takes in a nested dictionary representation and converts it into a `BinaryTree` representation:

```
class BinaryTree:
    def __init__(self, entry, left=None, right=None):
        self.entry = entry
        self.left = left
        self.right = right
def make_guess_tree(guess):
    """
    results = {'Is it an animal?': \
           {'yes': {'Is it a dog?': \
                   {'yes': 'Success!'}
                   }, \
            'no': {'Is it a place?': \
                   {'no': {'Is it a person?': \
                               {}},
                   'yes': {'Is it LA?': \
                               {}}
                   }}}}
    >>> t = make_guess_tree(results)
    >>> t.pretty_print()
                         Is it an animal?
                        /                \
             Is it a dog?                 Is it a place?
            /                            /              \
    Success!                    Is it LA?                Is it a person?
    """

    if type(guess) != type({}):

        ____________________________________________________________
    question = list(guess.keys())[0] #gets out question

    new_tree = ____________________________________________________

    if ____________________________________________________________

        ____________________________________________________________

    if ____________________________________________________________

        ____________________________________________________________

    ____________________________________________________________
```

6. (3 points) What Will Scheme Output? (Streams)

For each of the following Scheme expressions, write the Scheme value to which it evaluates. If evaluation causes an error, write `ERROR`. If evaluation never completes, write `FOREVER`. Assume that you have started `STk` and executed the following statements in order.

```
(define ones (cons-stream 1 ones))
(define a
  (cons-stream 1
    (cons-stream 3 b)))
(define b
  (cons-stream 2 a))
(define happy
  (cons-stream 2
    (stream-map * happy (stream-cdr happy))))
```

Expression	Interactive Output
`(ss ones)`	`(1 1 1 1 1 1 1 1 1 1 ...)`
`(ss (interleave a b))`	
`(stream-car happy)`	
`(stream-car (stream-cdr happy))`	

7. (5 points) Skipping Streams

Define a stream of streams where every `i`th element in the larger stream is a stream of the multiples of `i`. Assume that the stream `naturals` and the function `add-streams` that we defined in the Streams discussion are given to you, in addition to the predefined functions in `STk` (such as `stream-map`, `stream-filter`, `interleave`, etc.) **Do not add in any additional lines. Do not use any additional `define` expressions (but you may use `lambda` expressions).**

```
STk> (ss naturals)
(1 2 3 4 5 6 7 8 9 10 ...)
STk> (ss (add-streams naturals naturals))
(2 4 6 8 10 12 14 16 18 20 ...)
STk> (ss nth-multiples 4)
((1 2 3 4 ...) (2 4 6 8 ...) (3 6 9 12 ...) (4 8 12 16 ...) ...)
STk> (ss (stream-car (stream-cdr nth-multiples)))
(2 4 6 8 10 12 14 16 18 20 ...)

(define nth-multiples

  ------------------------------------------------------------------------

  ------------------------------------------------------------------------

  ------------------------------------------------------------------------

  ------------------------------------------------------------------------)
```

8. (6 points) Everyday I'm Shufflin

Define `shuffle`, which takes in a linked list and uses mutation to create a linked list with every pair of elements in the original list swapped (see the doctests). Your function **must return a linked list**, but that list must be created using only mutation of the `rest` attribute.

You may NOT call the `Link` constructor. You may NOT assign to the `first` attribute. The point of this problem is to mutate `rest` attributes. USE RECURSION.

```
def shuffle(lst):
    """
    >>> 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 ____________________________________________________________________
```

9. (5 points) Searching the Depths

Write `member` for deep lists in `Logic`. You may assume there is at most one occurrence of the element we are searching for. **Hint:** Consider how we could define separate facts to check for the element in either the `car` of the list or the `cdr` of the list.

```
logic> (query (member 5 ((((((((((5))))))))))))
Success!
logic> (query (member 4 (3 2 (1 4))))
Success!
logic> (query (member ?what ()))
Failed.
```

_____

_____

_____

_____

_____

10. (6 points) Subset Sum

Define `subset_sum`, a function which takes in a list of numbers `lst` and a number `n`, and returns a subset of the elements in `lst` that adds up to `n` if such a subset exists, otherwise it returns `None`. If there are multiple subsets, return any one of them. **Hint:** We can choose either to keep the first element of the `lst` in our subset, or we can choose to not have the first element in our subset.

```
def subset_sum(lst, n):
    """
    >>> sorted(subset_sum([4, 8, 3], 11))
    [3, 8]
    >>> print(subset_sum([4, 8, 3], 10))
    None
    >>> sorted(subset_sum([5, 8, 10, 7, 23], 40))
    [7, 10, 23]
    """

    if ____________________________________________________________
        return []

    elif __________________________________________________________
        return None


    no_first_element = ____________________________________________

    if no_first_element is not None:

        return ____________________________________________________


    using_first_element = _________________________________________

    if using_first_element is not None:

        return ____________________________________________________
```

11. (3 points) Concurrency

These two threads are running concurrently starting with `x = 1, y = 1`,

```
>>> x = x + y                    >>> y = x ** 2
>>> y = x - 1                    >>> x = y + 1
```

1) Line a1 read `x`
2) Line a1 read `y`
3) Line a1 write `x`
4) Line a2 read `x`
5) Line a2 write `y`
6) Line b1 read `x`
7) Line b1 write `y`
8) Line b2 read `y`
9) Line b2 write `x`

Determine whether `x = 5, y = 5` are possible values after the code is executed. If not possible, circle "impossible". If possible, list the order in which the 9 steps are taken by their step numbers.

Impossible

---- ---- ---- ---- ---- ---- ---- ---- ----

12. (9 points) Interpreters: Implementing Special Forms

In the Scheme project, you implemented several *special forms*, such as `if`, `and`, `begin`, and `or`. Now we're going to look at a new special form: `when`. A `when` expression takes in a `condition` and any number of other subexpressions, like this:

```
(when <condition>
      <exp>
      <exp>
      ...
      <exp>)
```

If `condition` is true, **all** of the following subexpressions are evaluated **in order** and the value of the `when` is the value of the last subexpression. If it is false, **none** of them are evaluated and the value of the `when` is `okay`. For example, `(when (= 1 1) (+ 2 3) (* 1 2))` would first evaluate `(+ 2 3)` and then `(* 1 2)`.

(a) (2 pt) Equivalent Scheme Expression Rewrite the `when` expression below into **another Scheme expression** which uses **only** the special forms you **already** implemented in your project. That is, create a Scheme expression which does the same thing as this `when` expression **without** using `when`. You should use `if` in your answer.

```
(when (= days-left 0)
      (print 'im-free)
      'jk-final)
```

You may or may not need to use all of the lines provided.

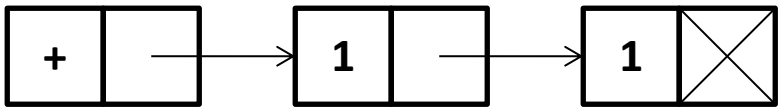
```
(_____________________________________________________________________________

     ____________________________________________________________________________

     ____________________________________________________________________________

     ____________________________________________________________________________)
```

(b) (2 pt) Box and Pointer Remember that `do_when_form`, like the other do_something_form functions in the Scheme project, takes in `vals` and the `env` to evaluate in. We will be drawing the box-and-pointer diagram for `vals` above. As an example, the box-and-pointer diagram for `'(+ 1 1)` would be



In the example from the description, `vals` would be `'((= 1 1) (+ 2 3) (* 1 2))`. Draw the box-and-pointer diagram for this list in the space provided below.

(c) (3 pt) Implementing When Now implement `do_when_form`. Assume that the other parts of `scheme.py` have already been modified to accommodate this new special form. You may not need to use all of the lines provided. You do not need to worry about tail recursion. Remember that `do_when_form` must return *two* things - a Scheme expression or value and an environment or `None`.

```
def do_when_form(vals, env):

    ------------------------------------------------------

    ------------------------------------------------------
```

(d) (2 pt) Implementing Another Special Form Now let's implement another special form `until`, which takes in a `condition` and a series of expressions, and evaluates the expressions in order only if the `condition` is **NOT** true. Implement `do_until_form` using `do_when_form`. (Remember that Scheme has a built-in `not` function which your interpreter can evaluate!)

```
def do_until_form(vals, env):

    new_expr = ____________________________________________________
    return do_when_form(new_expr, env)
```