CS 61A Structure and Interpretation of Computer Programs

Spring 2015 Midterm 1

INSTRUCTIONS

- 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 and the official 61A midterm 1 study guide attached to the back of this exam.
- Mark your answers ON THE EXAM ITSELF. If you are not sure of your answer you may wish to provide a *brief* explanation.

Last name	
First name	
SID	
Email (...@berkeley.edu)	
Login (e.g., cs61a-ta)	
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)**	

For staff use only

Q. 1	Q. 2	Q. 3	Total
/12	/14	/14	/40

1. (12 points) In-N-Out

For each of the expressions in the tables below, write the output displayed by the interactive Python interpreter when the expression is evaluated. The output may have multiple lines. No errors occur.

The first two rows have been provided as examples of the behavior of the built-in `pow` and `print` functions.

Recall: The interactive interpreter displays the value of a successfully evaluated expression, unless it is `None`.

Assume that you have started Python 3 and executed the following statements:

```
from operator import add

def re(peat):
    return print(peat, peat)

def cheap(eat):
    car, seat = re, print
    seat(car(eat))
    return double(eat)

def double(double):
    if double:
        return double + double
    elif car(double)(print)(print):
        return 1000
    else:
        return seat(3)

seat = double
car = lambda c: lambda a: lambda r: r(5, a(c))
```

Expression	Interactive Output
`pow(2, 3)`	8
`print(4, 5)`	4 5
`print(re(1+2), print(4))`	3 3 4 None None
`cheap(3)`	3 3 None Return value 6
`cheap(seat(2))`	8 8 None Return value 16

Expression	Interactive Output

`car(1)(double)(pow)`	"""c = 1
	a = func double
	r = func pow
	
	car = pow(5, double(1))
	car = pow(5, 2)"""
	25

`double(print(1))`	1 None 5 None return value 6
`car(0)(seat)(add)`	0 5 None return value 11

2. **(14 points) Supernatural**

(a) **(6 pt)** 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.

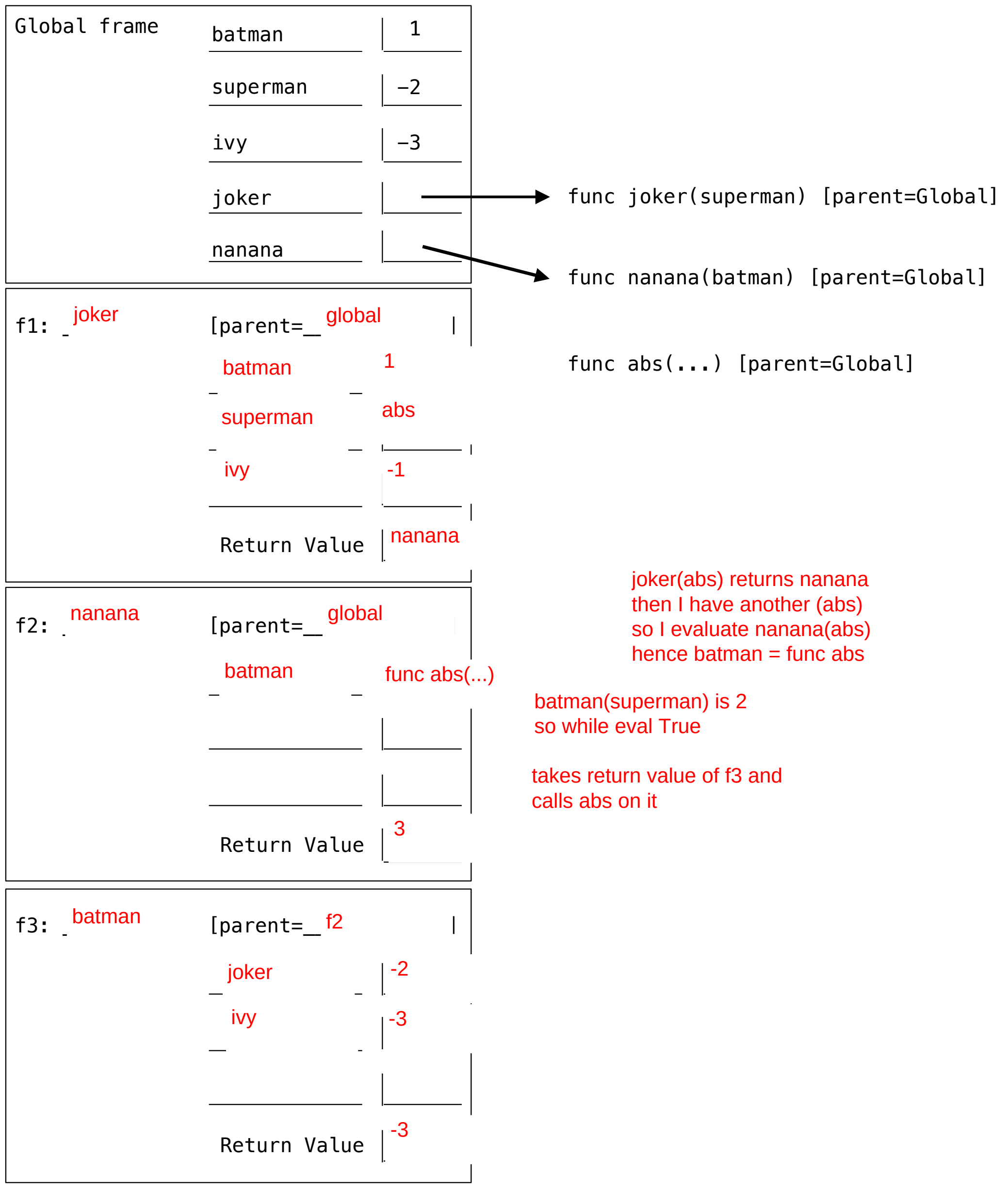
Remember: Do not add a new frame when calling a built-in function (such as `abs`).

```
batman, superman, ivy = 1, -2, -3

def nanana(batman):
    while batman(superman) > ivy:
        def batman(joker):
            return ivy
    return -ivy

def joker(superman):
    if superman(batman):
        ivy = -batman
    return nanana

joker(abs)(abs)
```



```
Global frame
    batman      | 1
    superman    | -2
    ivy         | -3
    joker       | → func joker(superman) [parent=Global]
    nanana      | → func nanana(batman) [parent=Global]

                  func abs(...) [parent=Global]

f1: joker        [parent= global]
    batman      | 1
    superman    | abs
    ivy         | -1
    Return Value | nanana

f2: nanana       [parent= global]
    batman      | func abs(...)
    Return Value | 3

f3: batman       [parent= f2]
    joker       | -2
    ivy         | -3
    Return Value | -3
```

joker(abs) returns nanana
then I have another (abs)
so I evaluate nanana(abs)
hence batman = func abs

batman(superman) is 2
so while eval True

takes return value of f3 and
calls abs on it

joker(abs)(abs) returns 3

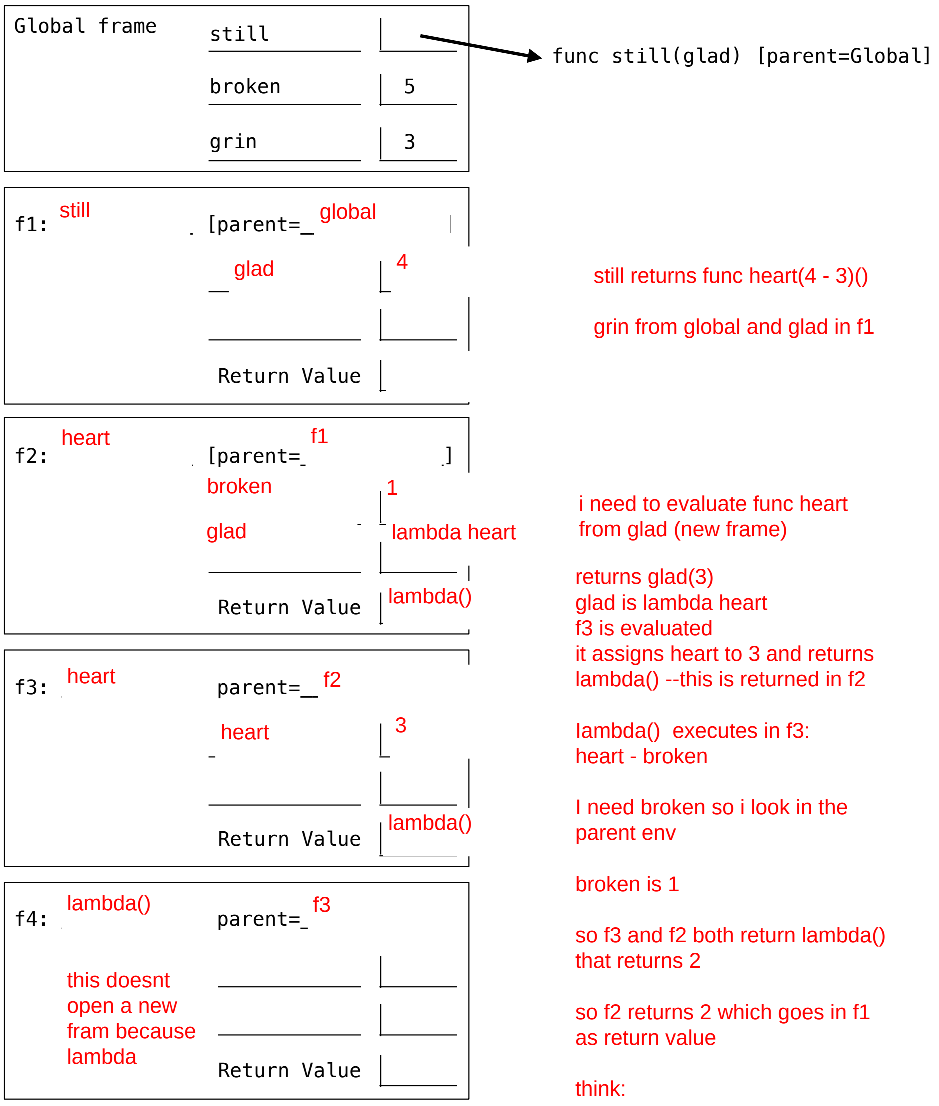
(b) **(8 pt)** Fill in the environment diagram that results from executing the code below until the entire program is finished, an error occurs, or all frames are filled. *You may not need to use all of the spaces or frames.*

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.

```
def still(glad):
    def heart(broken):
        glad = lambda heart: lambda: heart-broken
        return glad(grin)
    return heart(glad-grin)()

broken, grin = 5, 3
still(broken-1)
```



Global frame
still → func still(glad) [parent=Global]
broken 5
grin 3

f1: still [parent=global]
glad 4
Return Value

f2: heart [parent=f1]
broken 1
glad lambda heart
Return Value lambda()

f3: heart parent=f2
heart 3
Return Value lambda()

f4: lambda() parent=f3
this doesnt open a new fram because lambda
Return Value

still returns func heart(4 - 3)()

grin from global and glad in f1

i need to evaluate func heart from glad (new frame)

returns glad(3)
glad is lambda heart
f3 is evaluated
it assigns heart to 3 and returns lambda() --this is returned in f2

lambda() executes in f3:
heart - broken

I need broken so i look in the parent env

broken is 1

so f3 and f2 both return lambda() that returns 2

so f2 returns 2 which goes in f1 as return value

think:
heart is the function that returns the function that returns heart - broken

3. (14 points) You Complete Me

(a) (4 pt) Implement the `longest_increasing_suffix` function, which returns the longest suffix (end) of a positive integer that consists of strictly increasing digits.

```
def longest_increasing_suffix(n):
    """Return the longest increasing suffix of a positive integer n.

    >>> longest_increasing_suffix(63134)
    134
    >>> longest_increasing_suffix(233)
    3
    >>> longest_increasing_suffix(5689)
    5689
    >>> longest_increasing_suffix(568901) # 01 is the suffix, displayed as 1
    1
    """

    m, suffix, k = 10, 0, 1

    while n:
                                         n
        ______________________________________________, last = n // 10, n % 10

               last < m
        if ______________________________________________________________________:

                      last                      last*k + suffix
            m, suffix, k = __              , ___________________________, 10 * k

        else:

            return suffix

    return suffix
```

(b) (3 pt) Add parentheses and single-digit integers in the blanks below so that the expression on the second line evaluates to 2015. **You may only add parentheses and single-digit integers.** You may leave some blanks empty.

```
lamb = lambda lamb: lambda: lamb + lamb
                                                   lambda functions are hard! no time..

lamb(1000)______ + (lambda b, c: b______ * b______ - c______)(lamb(______), 1)______
```

(c) **(3 pt)** Implement the `combine` function, which takes a non-negative integer `n`, a two-argument function `f`, and a number `result`. It applies `f` to the first digit of `n` and the result of combining the rest of the digits of `n` by repeatedly applying `f` (see the doctests). If `n` has no digits (because it is zero), `combine` returns `result`.

```
from operator import add, mul

def combine(n, f, result):
    """Combine the digits in non-negative integer n using f.

    >>> combine(3, mul, 2) # mul(3, 2)
    6                                                        4 is n // 10
    >>> combine(43, mul, 2) # mul(4, mul(3, 2))              and so on        3 is n % 10
    24
    >>> combine(6502, add, 3) # add(6, add(5, add(0, add(2, 3))))
    16
    >>> combine(239, pow, 0) # pow(2, pow(3, pow(9, 0)))
    8
    """
    if n == 0:

        return result

    else:
                              n // 10         f                 f(n % 10, result)
        return combine(___________, ___________, ______________________________)
```

(d) **(4 pt)** Implement the `memory` function, which takes a number `x` and a single-argument function `f`. It returns a function with a peculiar behavior that you must discover from the doctests. **You may only use names and call expressions in your solution. You may not write numbers or use features of Python not yet covered in the course.**

```
square = lambda x: x * x
double = lambda x: 2 * x

def memory(x, f):
    """Return a higher-order function that prints its memories.

    >>> f = memory(3, lambda x: x)
    >>> f = f(square)
    3
    >>> f = f(double)
    9
    >>> f = f(print)
    6
    >>> f = f(square)
    3
    None
    """                   no time
    def g(h):

        print(_____________________________________________________________)


        return ____________________________________________________________

    return g
```