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)	45
print(re(1+2), print(4))	3 3 4 None none
cheap(3)	3 3 None 6
cheap(seat(2))	4 4 None 8

Expression	Interactive Output
car(1)(double)(pow)	25
double(print(1))	1 None 5 none 6 6
car(0)(seat)(add)	0 5 none 11

2. (14 points) Supernatural

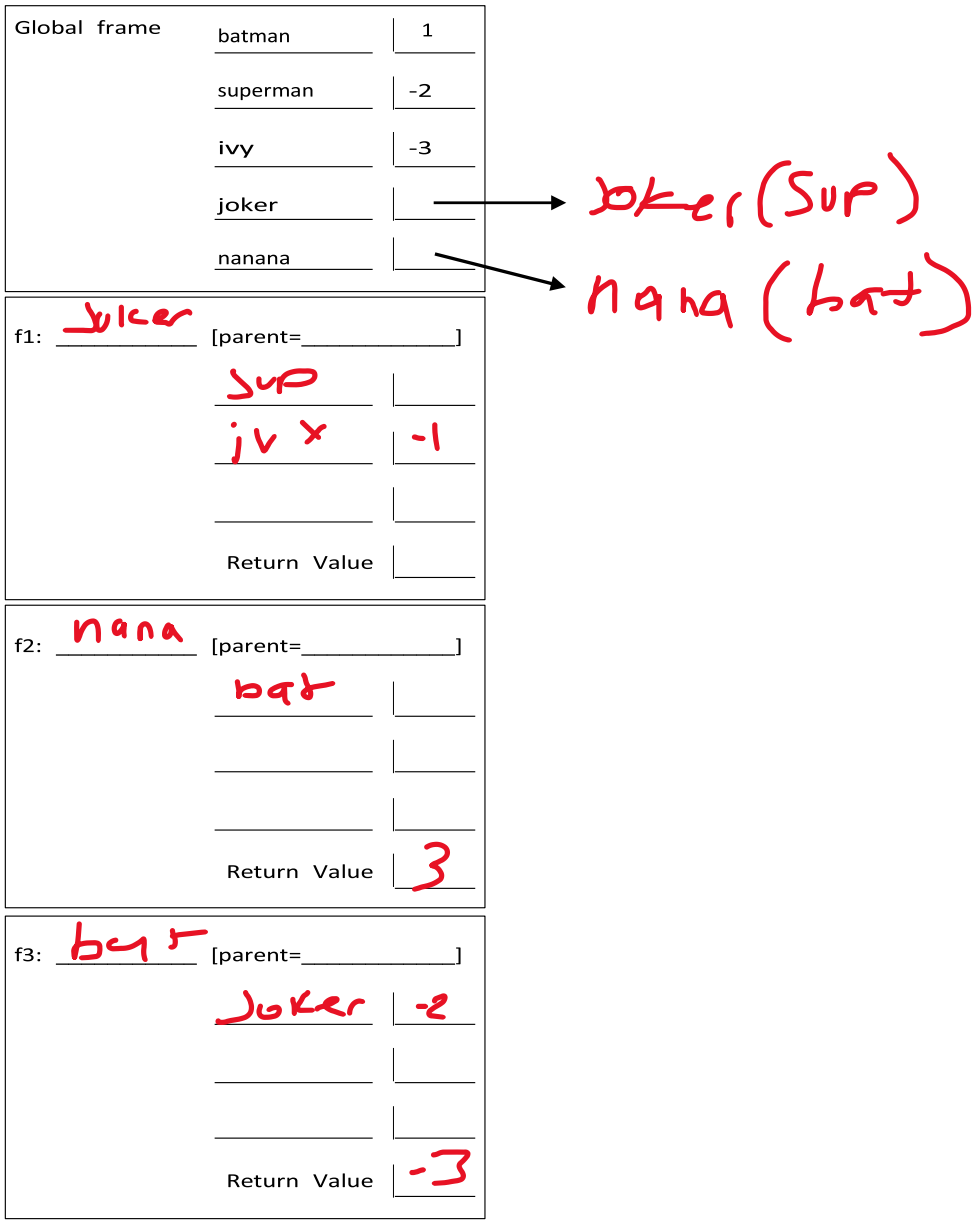
(a) (6 pt) Fill in the environment diagram that results from executing the code below until the entire program is finished, an error occurs, or all frames are filled. *You may not need to use all of the spaces or frames.*

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)
```



func joker(superman) [parent=Global]

func nanana(batman)
[parent=Global]

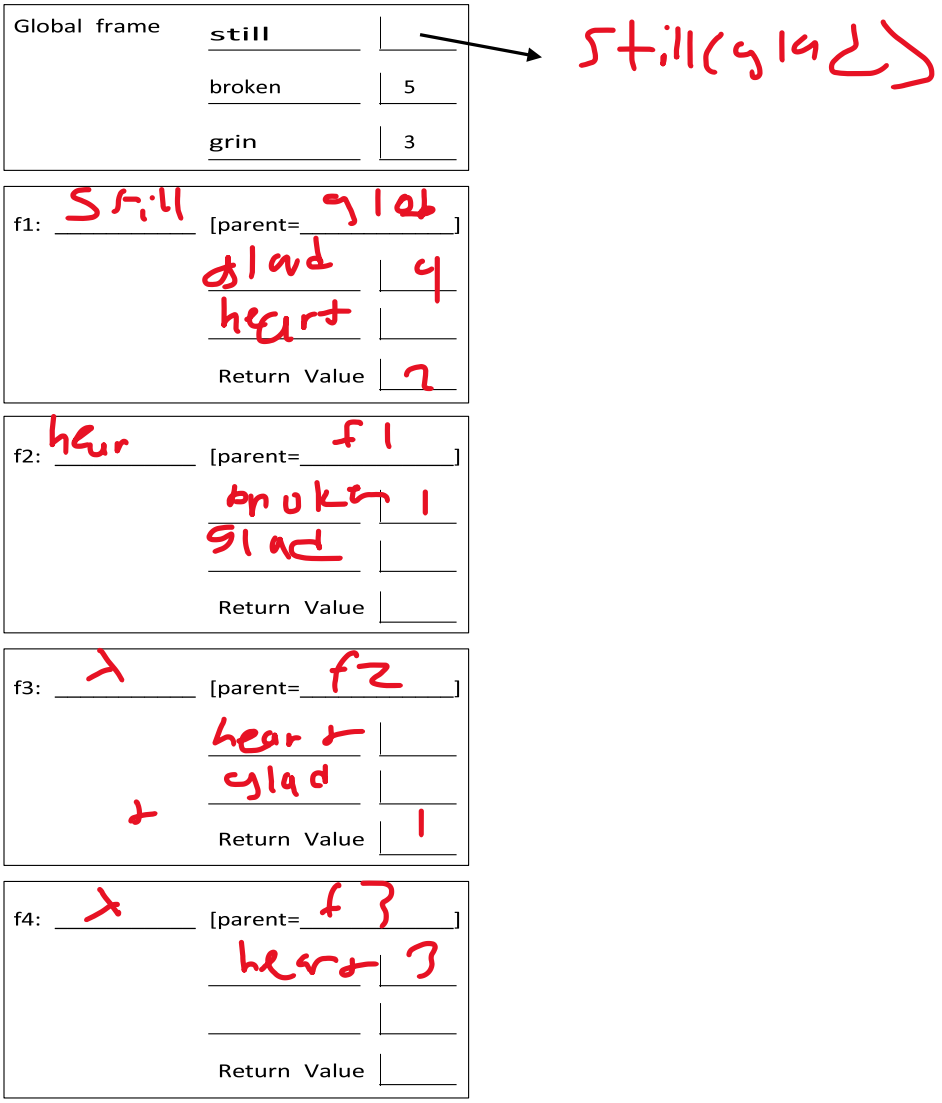
func abs(...) [parent=Global]

(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)
```




```
1
2
3 func still(glad) [parent=Global]
4
5
6
7
8
9
```

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


                                        if last < m:


            m, suffix, k = last, suffix 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


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
    >>> combine(43, mul, 2) # mul(4, mul(3, 2))
    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:

                                    return combine(n// 10,f,f(n%1,result))
```

(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 """ def
    g(h):

                                    print(f(x))


        return memory(x,h)

         return g
```