CS 61A Fall 2012

Structure and Interpretation of Computer Programs

Midterm 1 Solutions

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	
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)**	

For staff use only

Q. 1	Q. 2	Q. 3	Q. 4	Total
/12	/12	/14	/12	/50

1. (12 points) The Call Express is Delayed

For each of the following call expressions, write the value to which it evaluates *and* what would be output by the interactive Python interpreter. The first two rows have been provided as examples.

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

```
from operator import add, mul
def square(x):
    return mul(x, x)

def delay(arg):
    print('delayed')
    def g():
        return arg
    return g
```

Expression	Evaluates to	Interactive Output
`square(5)`	25	25
`1/0`	Error	Error
`print(square(4))`	None	16
`square(square(print(2)))`	Error	2 Error
`print(add(3, 4), print(5))`	None	5 7 None
`delay(square)(3)`	Error	delayed Error
`add(delay(square)()(2), 1)`	5	delayed 5
`delay(delay)()(6)()`	6	delayed delayed 6

2. (12 points) Protect the Environment

(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.*

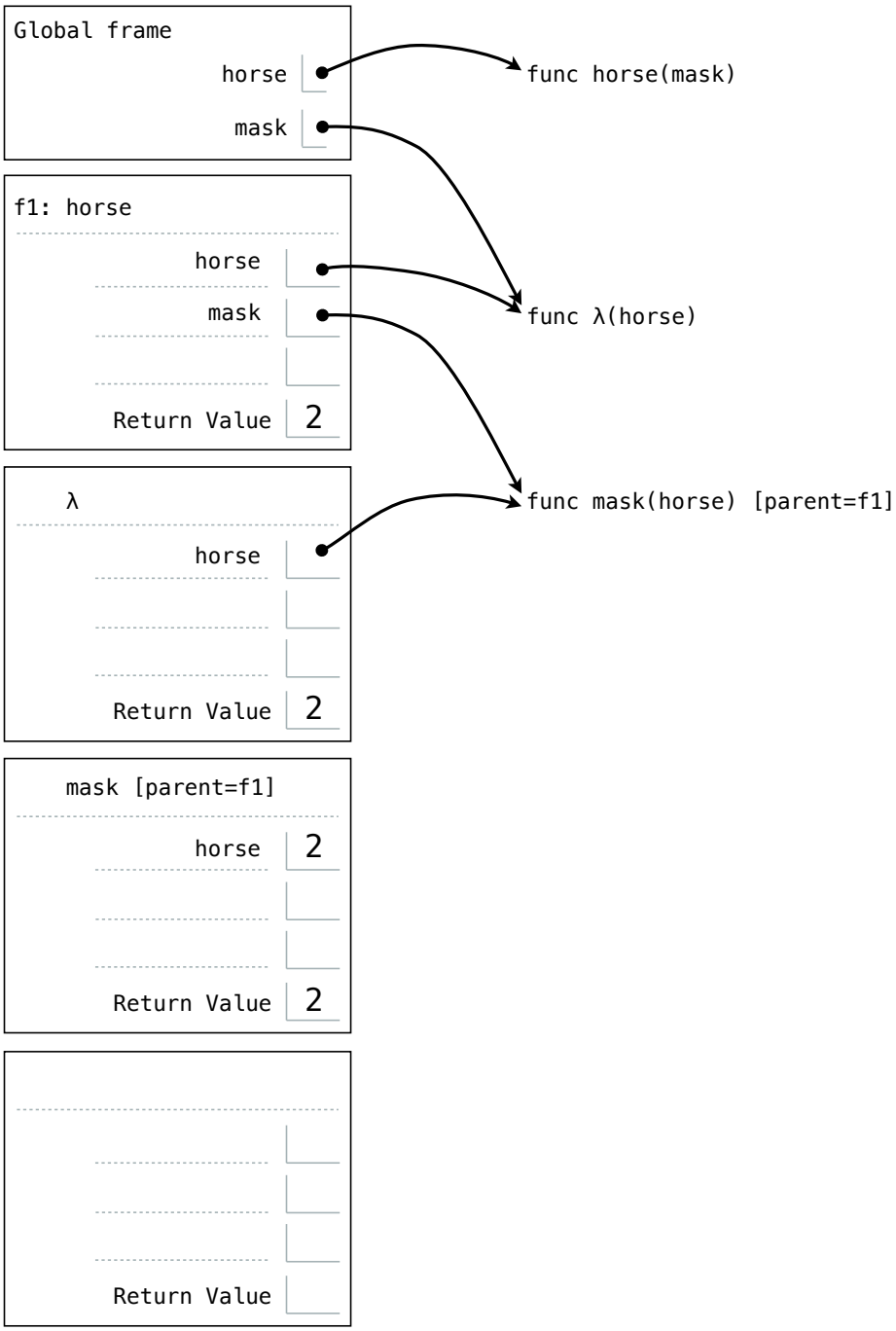
A complete answer will:

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

```
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)

horse(mask)
```



(b) (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:

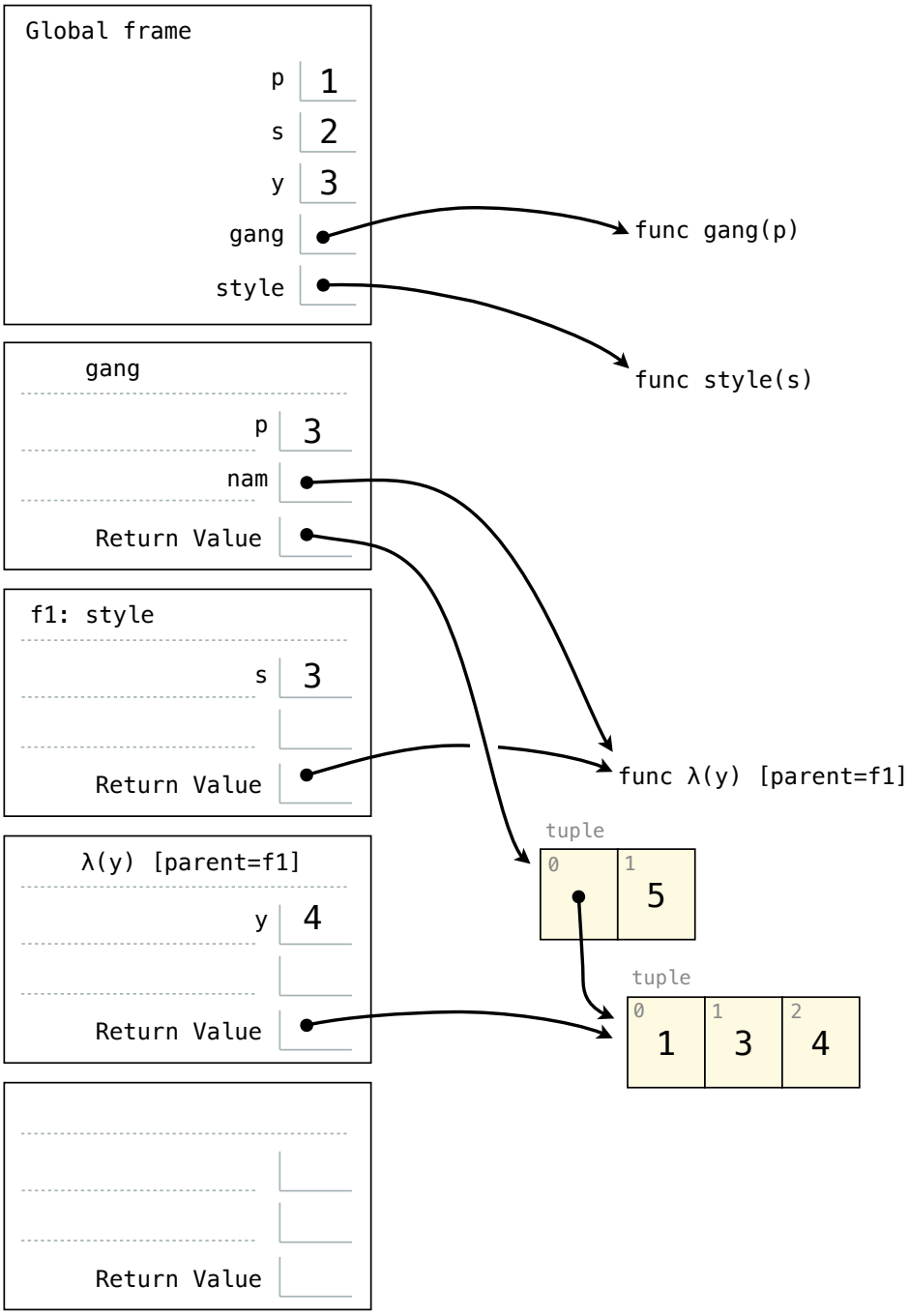
- Add all missing names, labels, and parent annotations to all local frames.
- Add all missing values created during execution.
- Show the return value for each local frame.

```
p, s, y = 1, 2, 3

def gang(p):
    nam = style(p)
    return (nam(4), 5)

def style(s):
    return lambda y: (p, s, y)

gang(3)
```



3. (14 points) Sequences

(a) (2 pt) Fill in the blanks so that the final call expression below evaluates to a **tuple** value.

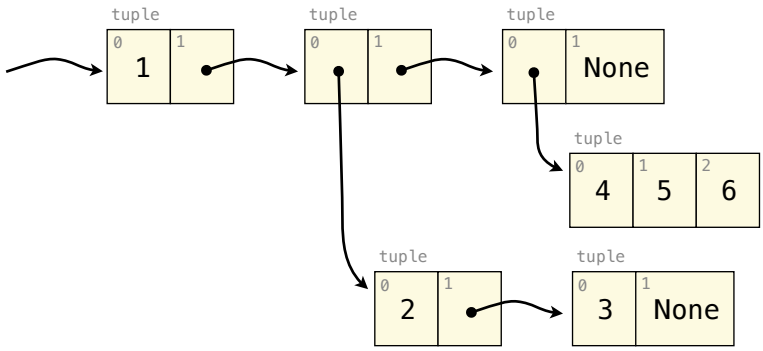
```
def pair(x):
    if x == 30:
        return lambda: (1, 2, 3)
    else:
        return lambda: 4


(lambda_____________, soda: hall_____________)(pair, "sequence")
```

(lambda hall, soda: hall(30)())(pair, "sequence")

(b) (2 pt) Draw a box and pointer diagram for the following `rlist`:

```
a = rlist(1, rlist(rlist(2, (3, empty_rlist)), rlist((4, 5, 6), empty_rlist)))
```



(c) (2 pt) What is the element at index 1 of this rlist, returned by `getitem_rlist(a, 1)`?

(2, (3, None))

```
def getitem_rlist(s, i):
    """Return the element at index i of recursive list s."""
    while i > 0:
        s, i = rest(s), i - 1
    return first(s)
```

(d) (2 pt) What is the length of this rlist, returned by `len_rlist(a)`?

3

```
def len_rlist(s):
    """Return the length of recursive list s."""
    length = 0
    while s != empty_rlist:
        s, length = rest(s), length + 1
    return length
```

(e) (6 pt) When the `int` constructor is called on a `float` value, it "truncates toward zero," meaning that it returns the largest integer less than any positive argument, or the least integer greater than any negative argument. For example:

```
>>> int(2)
2
>>> int(2.7)
2
>>> int(-1.5)
-1
```

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

```
def alt(f, g, z):
    while g(z) > 0 and z != 5:
        f, g = g, f
        z = g(z)
    return z

def grow(x):
    return int((x * 3) / 2)

def shrink(x):
    return x - 2

def flip(x):
    return int(10 / (x-2))
```

For each of the following call expressions, write the value to which it evaluates. If evaluation causes an error, write ERROR. If evaluation would run forever, write FOREVER.

- `alt(grow, shrink, 3)`

  2

- `alt(grow, shrink, 4)`

  Forever

- `alt(flip, shrink, 3)`

  1

4. (12 points) In Verse

The inverse of some function F	*There once was a rhyming device*
is a function of argument X	*That was built to make any sound, twice,*
that returns you the Y,	*But used orthography*
such that when you apply	*And not phonology*
F to Y you recover the X.	*To decide if a rhyme would suffice.*

An invertible function is a function that takes and returns a single numeric value, is differentiable, and never returns the same value for two different arguments. Some examples:

```
def double(y):
    """Return twice the value of y."""
    return 2 * y

def cube(y):
    """Return y raised to the third power."""
    return pow(y, 3)

def pow2(y):
    """Return 2 raised to the power of y."""
    return pow(2, y)
```

(a) (4 pt) Implement a function `invert` that takes an invertible function argument and returns its inverse. You may call `find_root`, `newton_update`, `approx_deriv`, and/or `iter_improve`. You **cannot** use any assignment, conditional, `while`, or `for` statements.

```
def invert(f):
    """Return the inverse of invertible function f.

    >>> halve = invert(double)
    >>> halve(12)
    6.0
    >>> cube_root = invert(cube)
    >>> cube_root(27)
    3.0
    >>> log2 = invert(pow2)
    >>> log2(32)
    5.0
    """

    def g(x):
        return find_root(lambda y: f(y) - x)
    return g
```

A sight rhyme is a pair of words that do not rhyme, but have the same endings, such as *device* and *office*. Two numbers that end in the same digit can be sight rhymes. For example:

- `(13, 53)` are pronounced *thirteen* and *fifty-three*, despite both ending with the same one's digit `3`.
- `(0, 30)` are pronounced *zero* and *thirty*, despite both ending with the same one's digit `0`.

(b) (4 pt) A `numpair` is a pair of integers that have the same one's digit. Fill in the **two** missing expressions in the constructor below, which takes two non-negative integers less than 100, asserts that they have the same one's digit, and returns a numpair represented as a pair of tens digits and the shared one's digit.

```
from operator import floordiv, mod  # Use these functions or // and %

def numpair(first, second):
    """Return a numpair as a pair of ten's digits and a shared one's digit.

    >>> numpair(23, 53)
    ((2, 5), 3)
    >>> numpair(67, 7)
    ((6, 0), 7)
    """
    assert first % 10 == second % 10

    return ((first // 10, second // 10), first % 10)
```

(c) (4 pt) Fill in **four** missing expressions below so that `sight_rhyme` returns whether the numbers in a `numpair p` **do not** end with the same sound when *pronounced*. Your implementation **cannot** depend on the *representation* of a `numpair`; use selector functions. You **cannot** use the boolean operators `and` or `or`.

```
def ones(p):
    return p[1]
def first_tens(p):
    return p[0][0]
def second_tens(p):
    return p[0][1]
def sight_rhyme(p):
    """Return whether the two numbers in a numpair do not rhyme.

    >>> sight_rhyme(numpair(13, 53))
    True
    >>> sight_rhyme(numpair(0, 30))
    True
    >>> sight_rhyme(numpair(53, 23))
    False
    """
    if (first_tens(p) == 1)+(second_tens(p) == 1) >= 1:

        return first_tens(p) != second_tens(p)
    elif ones(p) == 0:
        if first_tens(p) == 0:

            return second_tens(p) != 0
        else:

            return second_tens(p) == 0
    else:
        return False
```