CS 61A Structure and Interpretation of Computer Programs

Fall 2013 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	
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	/16	/14	/8	/50

1. (12 points) Dog Goes Woof

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.

- In the **Evaluates to** column, write the value to which the expression evaluates. If evaluation causes an error, write Error.
- In the column labeled **Interactive Output**, write all output that would be displayed during an interactive session, after entering each call expression. This output may have multiple lines. Whenever the interpreter would report an error, write Error. You *should* include any lines displayed before an error. *Reminder*: 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, mul
def square(x):
    return mul(x, x)

def dog(bird):
    def cow(tweet, moo):
        woof = bird(tweet)
        print(moo)
        return woof
    return cow

cat = dog(square)
```

Expression	Evaluates to	Interactive Output
`square(5)`	25	25
`1/0`	Error	Error
`add(square(2), mul(3, 4))`		
`print(print(print(2)))`		
`cat(3, 4)`		
`square(cat(5))`		
`cat(square(2), print(5))`		
`cat(print(square(3)), 8)`		

2. (16 points) Frog Goes Croak

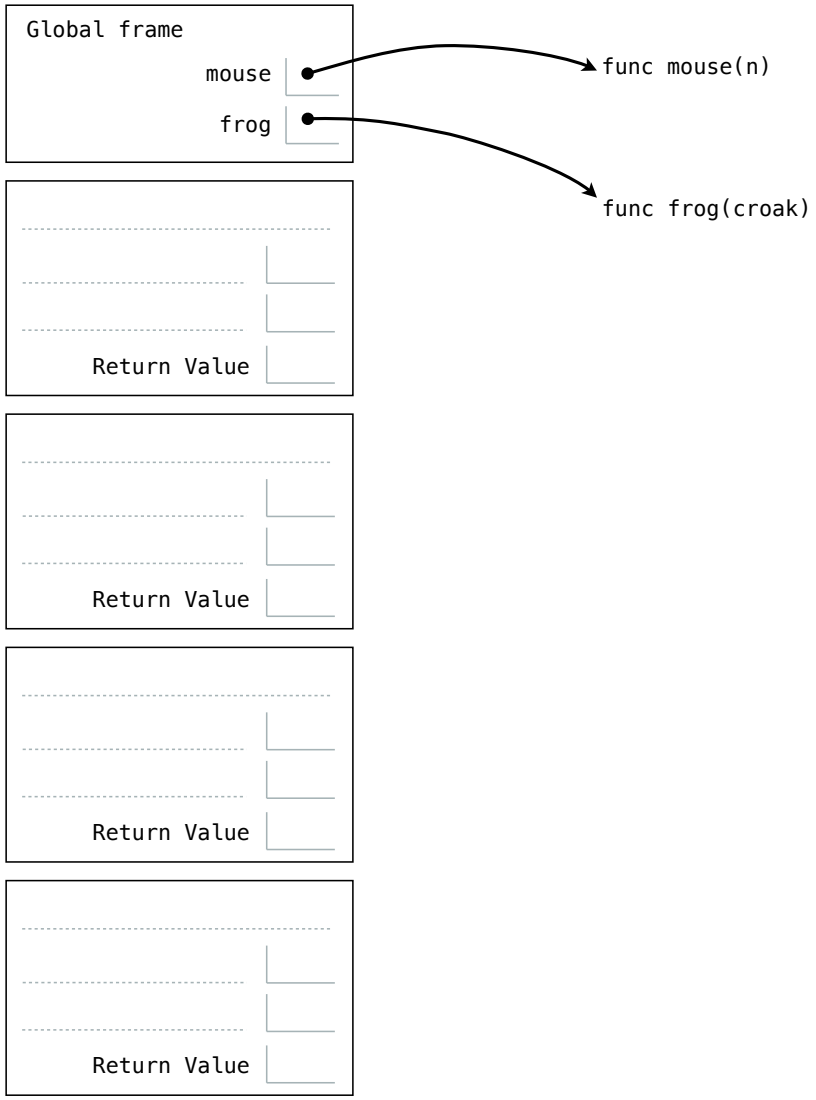
(a) (6 pt) Fill in the environment diagram that results from executing the code below until the entire program is finished, an error occurs, or all frames are filled. *You may not need to use all of the spaces or frames.* A complete answer will:

- 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 mouse(n):
    if n >= 10:
        squeak = n // 100
        n = frog(squeak) + n % 10
    return n

def frog(croak):
    if croak == 0:
        return 1
    else:
        return 10 * mouse(croak+1)

mouse(357)
```



(b) (2 pt) ***Question*:** After executing the code above, to what value will `mouse(21023508479)` evaluate?

"People have forgotten this truth," the fox said. "But you mustn't forget it. You become responsible forever for what you've tamed." — Antoine de Saint-Exupéry, The Little Prince

(c) (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.*

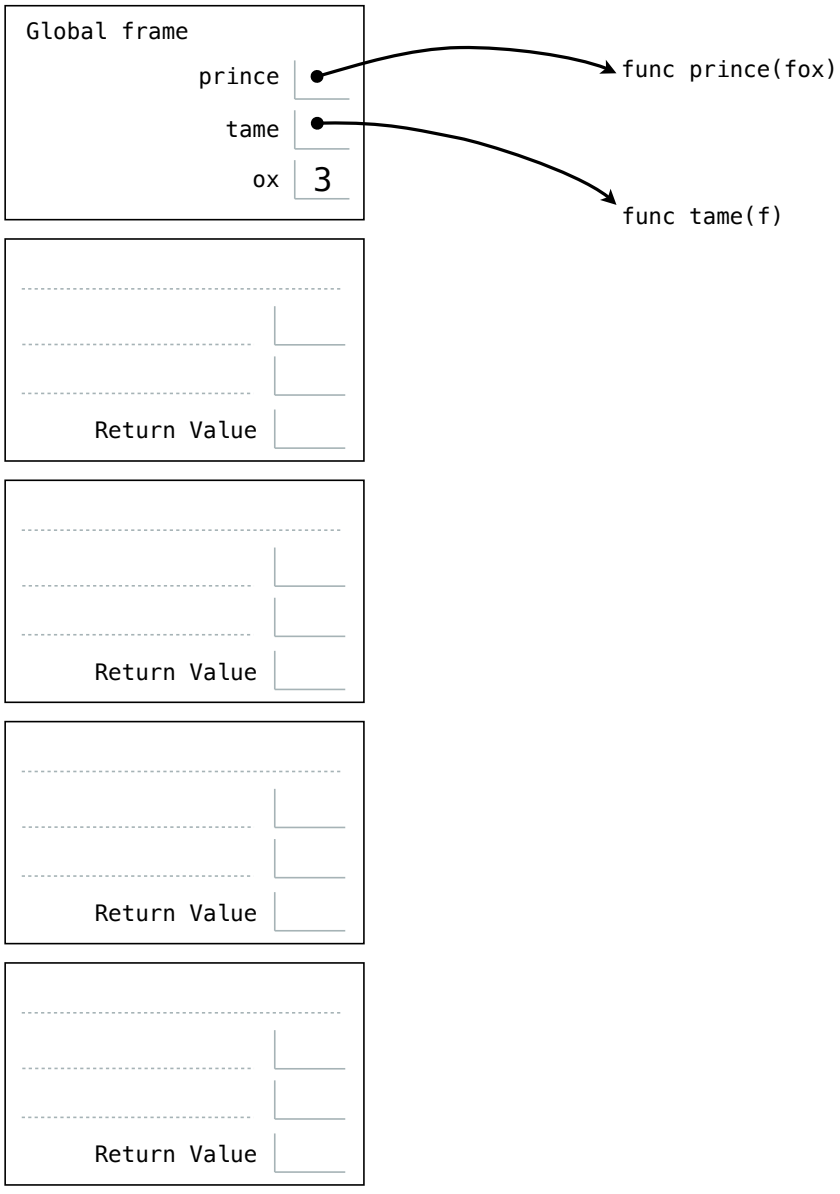
A complete answer will:

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

```
def prince(fox):
    def fox(fox):
        return lambda x: fox
    return fox(5)

def tame(f):
    return f(ox)(4)

ox = 3
tame(prince)
```



3. (14 points) Elephant Goes Toot

(a) (4 pt) Fill in the blanks of the implementation of `differs_by_one_digit` below, a function that takes two positive integers `m` and `n` and returns whether `m` and `n` differ in exactly one digit. If `m` and `n` have different numbers of digits, then `differs_by_one_digit(m, n)` always returns `False`.

```
def differs_by_one_digit(m, n):
    """Return True if and only if m and n have the same number of digits,
    and they differ by exactly one digit.

    You may assume that m and n are positive integers.

    >>> differs_by_one_digit(3467, 3427) # 3rd digit differs
    True
    >>> differs_by_one_digit(2013, 2011) # Last digit differs
    True
    >>> differs_by_one_digit(1013, 2013) # First digit differs
    True
    >>> differs_by_one_digit(5, 2)       # Only digit differs
    True
    >>> differs_by_one_digit(2013, 2013) # No digit differs
    False
    >>> differs_by_one_digit(1013, 2011) # Both first and last differ
    False
    >>> differs_by_one_digit(3102, 2013) # All digits differ
    False
    >>> differs_by_one_digit(1, 21)      # Different number of digits
    False
    >>> differs_by_one_digit(1, 12)      # Different number of digits
    False
    >>> differs_by_one_digit(21, 1)      # Different number of digits
    False
    >>> differs_by_one_digit(12, 1)      # Different number of digits
    False
    """
    diffs = 0
    while m > 0:

        if ____________________________________________________:

            return False

        m, t = m // 10, m % 10

        n, v = n // 10, n % 10

        if ____________________________________________________:

            diffs = ____________________________________________

    return ____________________________________________________
```

(b) **(3 pt)** Using only the numeral 5, the numeral 2, the name `mul`, commas, and parentheses, complete the final expression below so that it evaluates to 15.

```
from operator import add, mul
def f(x):
    def g(y):
        def h(f):
            return f(add(1, x), y)
        return h
    return g


f(_____________________________________________________________)
```

(c) **(3 pt)** Fill in the blanks below with expressions so that the final expression evaluates to the string `"onetwothree"`. *Reminder*: The expression `'a' + 'bc'` evaluates to `'abc'`.

```
ring = lambda ________: ding(____________________________)(_____________________)

ring(lambda x: lambda y: x() + "two" + y)
```

(d) **(4 pt)** The CS61A staff has developed a formula for determining what a fox might say. Given three strings, a start, a middle, and an end, a fox will say the start string, followed by the middle string repeated a number of times, followed by the end string. These parts are all separated by single hyphens.

Complete the definition of `fox_says`, which takes the three string parts of the fox's statement (`start`, `middle`, and `end`) and a positive integer `num` indicating how many times to repeat `middle`. It returns a string.

You **cannot** use any `for` or `while` statements. Use recursion in `repeat`. Moreover, you **cannot** use string operations other than the `+` operator to concatenate strings together.

```
def fox_says(start, middle, end, num):
    """
    >>> fox_says('wa', 'pa', 'pow', 3)
    'wa-pa-pa-pa-pow'
    >>> fox_says('fraka', 'kaka', 'kow', 4)
    'fraka-kaka-kaka-kaka-kaka-kow'
    """
    def repeat(k):









    return start + '-' + repeat(num) + '-' + end
```

4. (8 points) What Does Newton Say?

Your partner has implemented a function `derivative` that takes a single-argument differentiable real-valued function `f` and a real number `x` and returns the derivative of `f` evaluated at `x`. You don't know how she did it, but you find that it works perfectly.

```
def derivative(f, x):
    """Return f'(x), the derivative of f at x.

    >>> derivative(lambda x: x*x, 4)    # derivative of x*x is 2*x
    8
    >>> derivative(lambda x: x*x*x, 4)  # derivative of x*x*x is 3*x*x
    48
    """
    # Mystery implementation!
```

(a) (4 pt) Complete a new implementation of `find_zero` below so that it takes only one argument, a differentiable function `f`. You may use `derivative` above, along with `newton_update` and `improve` from your study guide. You **cannot** use any assignment (`=`), conditional (`if`), `for`, or `while` statements.

```
def find_zero(f):
    """Return a zero of the function f.

    >>> def cube_root(a):
    ...     return find_zero(lambda x: x*x*x - a)
    ...
    >>> cube_root(729)
    9.0
    """
    def near_zero(x):
        return approx_eq(f(x), 0)
```

(b) (4 pt) The function `equal` takes two differentiable single-argument functions `f` and `g` and returns an `x` for which `f(x)` is equal to `g(x)`. Implement the support function `equal_update` that completes the implementation. You may use `derivative` above, along with `newton_update` from your study guide. You **cannot** use any assignment (`=`), conditional (`if`), `for`, or `while` statements.

```
def equal(f, g):
    """Return an x for which f(x) == g(x).

    >>> def cube(x):
    ...     return x * x * x
    ...
    >>> def plus_six(x):
    ...     return x + 6
    ...
    >>> equal(cube, plus_six)
    2.0
    """
    def close(x):
        return approx_eq(f(x), g(x))
    return improve(equal_update(f, g), close)

def equal_update(f, g):
    """Return an update function that completes the implementation of equal."""
```