CS 61A Structure and Interpretation of Computer Programs
Spring 2018

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 CS 61A midterm 1 study guide.
- Mark your answers **on the exam itself**. We will *not* grade answers written on scratch paper.

Last name	
First name	
Student ID number	
CalCentral email (`_@berkeley.edu`)	
TA	
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)**	

POLICIES & CLARIFICATIONS

- If you need to use the restroom, bring your phone and exam to the front of the room.
- You may use built-in Python functions that do not require import, such as `min`, `max`, `pow`, and `abs`. You **may not** use functions defined on your study guide unless clearly specified in the question.
- For fill-in-the blank coding problems, we will only grade work written in the provided blanks. You may only write one Python statement per blank line, and it must be indented to the level that the blank is indented.
- Unless otherwise specified, you are allowed to reference functions defined in previous parts of the same question.

1. (12 points) Frame of Thrones *(All are in Scope: WWPD, Higher-Order Functions)*

For each of the expressions in the table below, write the output displayed by the interactive Python interpreter when the expression is evaluated. The output may have multiple lines. If an error occurs, write "Error", but include all output displayed before the error. To display a function value, write "Function". The first two rows have been provided as examples.

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

Assume that you have first started `python3` and executed the statements on the left.

```
from operator import add, sub

def winterfell(a, b):
    a
    b
    return b(a+1, b(a))

da, ny = 20, 18

while da > ny:
    da = ny
    da, ny = ny + 1, da + 3

def tar(gar, yen):
    if print(yen):
        print(yen + 1)
    return gar(yen)

def st(ar, k=None):
    return lambda a, y: ar(y, a)

night = st(sub)
king = st(st(pow))

def jon(sn, ow):
    print(ow)
    jon = sn(ow)
    print(ow)
    return jon

def snow(ow):
    def tarly(snow):
        return ow + snow
    ow += 2
    return tarly
```

Expression	Interactive Output
`sub(pow(10, 2), 1)`	99
`print(4, 5) + 1`	4 5 Error
`(print(2) or 3) // (0 or 1)`	
`winterfell(2, print)`	
`ny`	
`tar(lambda x: x-7, 8)`	
`night(king(2, 3), 4)`	
`jon(snow(5), 2)`	

2. (8 points) Stranger Frames *(All are in Scope: Environment Diagrams, Higher-Order Functions)*

Fill in the environment diagram that results from executing the code on the right 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 lucas(mike):
    return will

def dustin(lucas):
    will = 1
    def dustin(mike):
        will = 2
        return lucas
    return lambda mad: dustin(3)(will)

will = 5 + 6
lucas = dustin(lucas)
lucas(max)
```

```
Global frame      lucas    |
                  dustin   |
                  will     |

f1: dustin   [parent= Global ]
              ________ |
              ________ |
              ________ |
              ________ |
          Return Value |

f2: ________ [parent=________]
              ________ |
              ________ |
          Return Value |

f3: ________ [parent=________]
              ________ |
              ________ |
          Return Value |

f4: ________ [parent=________]
              ________ |
              ________ |
          Return Value |
```

func dustin(lucas) [parent=Global]

func lucas(mike) [parent=Global]

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

3. (10 points) Choose Wisely

(a) (4 pt) *(All are in Scope: Control)* Implement `sum_some`, which takes a non-negative integer `n` and a function `p`. It returns the sum of all the digits `d` for which `p` returns a true value when given `d` as an argument. Assume that the function `p` takes a single digit `d` (from 0 to 9) and returns either `True` or `False`.

```
def sum_some(n, p):
    """Return the sum of the digits of N for which P returns a true value.

    >>> even = lambda d: d % 2 == 0
    >>> big = lambda d: d > 5
    >>> sum_some(124567, even)     # Sum the even digits: 2 + 4 + 6
    12
    >>> sum_some(124567, big)      # Sum the big digits: 6 + 7
    13
    """
    total = 0

    while __________________________________________________________________________________:

        if __________________________________________________________________________________:

            ________________________________________________________________________________

        ____________________________________________________________________________________

    return total
```

(b) (4 pt) *(All are in Scope: Tree Recursion)* Implement `sum_largest`, which takes non-negative integers `n` & `k`. It sums the largest `k` digits of `n`.

```
def sum_largest(n, k):
    """Return the sum of the K largest digits of N.

    >>> sum_largest(2018, 2)    # 2 and 8 are the two largest digits (larger than 0 and 1).
    10
    >>> sum_largest(12345, 10)  # There are only five digits, so all are included in the sum.
    15
    """
    if _____________________________________________________________________________________:

        return 0

    a = ____________________________________________________________________________________

    b = ____________________________________________________________________________________

    return ____________________________________________________________________________(a, b)
```

(c) (2 pt) *(All are in Scope: Lambda Expressions)* Complete the expression below by **only adding parentheses** so that the whole expression evaluates to 2018. Each blank should be filled with one or more parentheses.

```
(lambda a, x: x + (lambda y: lambda z: y+z+1000)(1000 _____ 10 _____ 5, _____ lambda: 8 _____ )
```

4. (10 points) Editor

Definitions. An *edit* is a pure function that takes a non-negative integer and returns a non-negative integer. An *editor* for a non-negative integer `n` is a function that takes an *edit*, applies it to `n`, displays the result, and then returns an editor for the result.

(a) (3 pt) *(At least one of these is out of Scope: Self Reference, Higher-Order Functions)* Implement `make_editor`, which takes a non-negative integer `n` and a one-argument function `pr`. It returns an *editor* for `n` that uses `pr` to display the result of each *edit*.

(b) (5 pt) *(All are in Scope: Recursion, Higher-Order Functions)* Implement `insert`, which takes a single digit `d` (from 0 to 9) and a non-negative position `k`. It returns an *edit* that inserts `d` into its argument `n` at position `k`, where `k` counts the number of digits from the end of `n`. **Assume that `k` is not larger than the number of digits in `n`. Your solution must be recursive.**

(c) (2 pt) *(All are in Scope: Lambda Expressions)* Implement `delete`, which takes a non-negative integer `k` and returns an *edit* that deletes the last `k` digits of its argument `n`. You may use `pow` or `**` in your solution.

```
def make_editor(n, pr):
    """Return an editor for N.

    >>> f = make_editor(2018, lambda n: print('n is now', n))
    >>> f = f(delete(3))     # delete the last 3 digits from the end of 2018
    n is now 2
    >>> f = f(insert(4, 0))  # insert digit 4 at the end of 2 (position 0)
    n is now 24
    >>> f = f(insert(3, 1))  # insert digit 3 in the middle of 24 (position 1)
    n is now 234
    >>> f = f(insert(1, 3))  # insert digit 1 at the start of 234 (position 3)
    n is now 1234
    >>> f = make_editor(123, print)(delete(10))  # delete 10 digits from the end of 123
    0
    """
    def editor(edit):

        result = ____________________________________________________________________________

        _____________________________________________________________________________________

        _____________________________________________________________________________________

    return editor

def insert(d, k):

    def edit(n):

        if _________________________________________________________________________________:

            return _____________ + 10 * _____________________________________________________

        else:

            return _____________ + 10 * insert ______________________________________________

    return edit

delete = ____________________________________________________________________________________
```

No more questions.