CS 61A Structure and Interpretation of Computer Programs
Summer 2019

MIDTERM SOLUTIONS

INSTRUCTIONS

- You have 3 hours to complete the exam individually.
- The exam is closed book, closed notes, closed computer, and closed calculator, except for two hand-written 8.5" × 11" crib sheet of your own creation.
- Mark your answers **on the exam itself**. We will *not* grade answers written on scratch paper.

Last (Family) Name	
First (Given) Name	
Student ID Number	
Berkeley Email	
Teaching Assistant	
Exam Room	
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`, `len`, and `abs`.
- You **may not** use example functions defined on your study guides unless a problem clearly states you can.
- 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. (10 points) WWPD

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. Each expression has at least one line of output.

- If an error occurs, write **Error**, but include all output displayed before the error.
- To display a function value, write **Function**.
- If an expression would take forever to evaluate, write **Forever**.
- If nothing is displayed, write **Nothing**

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

Assume that you have started `python3` and executed the code shown on the left first, then you evaluate each expression on the right in the order shown. Expressions evaluated by the interpreter have a cumulative effect.

```
1  d = {}
2  x = 2
3  y = 9
4  p = lambda : print('crikey')
5
6  def f(a, b):
7      if b or print(a):
8          print(a, b)
9      if b > a:
10         return a + b
11     else:
12         return "less"
13
14 def g(f, x, y):
15     def m(n):
16         if f(n):
17             m = f
18             print('bonk')
19         else:
20             m = p
21             print('ers')
22         return m(n - 1)
23     return m(x) and f(x)
24
25 def h(x):
26     if x not in d:
27         d[x] = 'yes'
28         return False
29     else:
30         return True
31
32 def m(x):
33     d = {}
34
35 def e(x):
36     print(h(x))
37     m(x)
38     print(h(x))
```

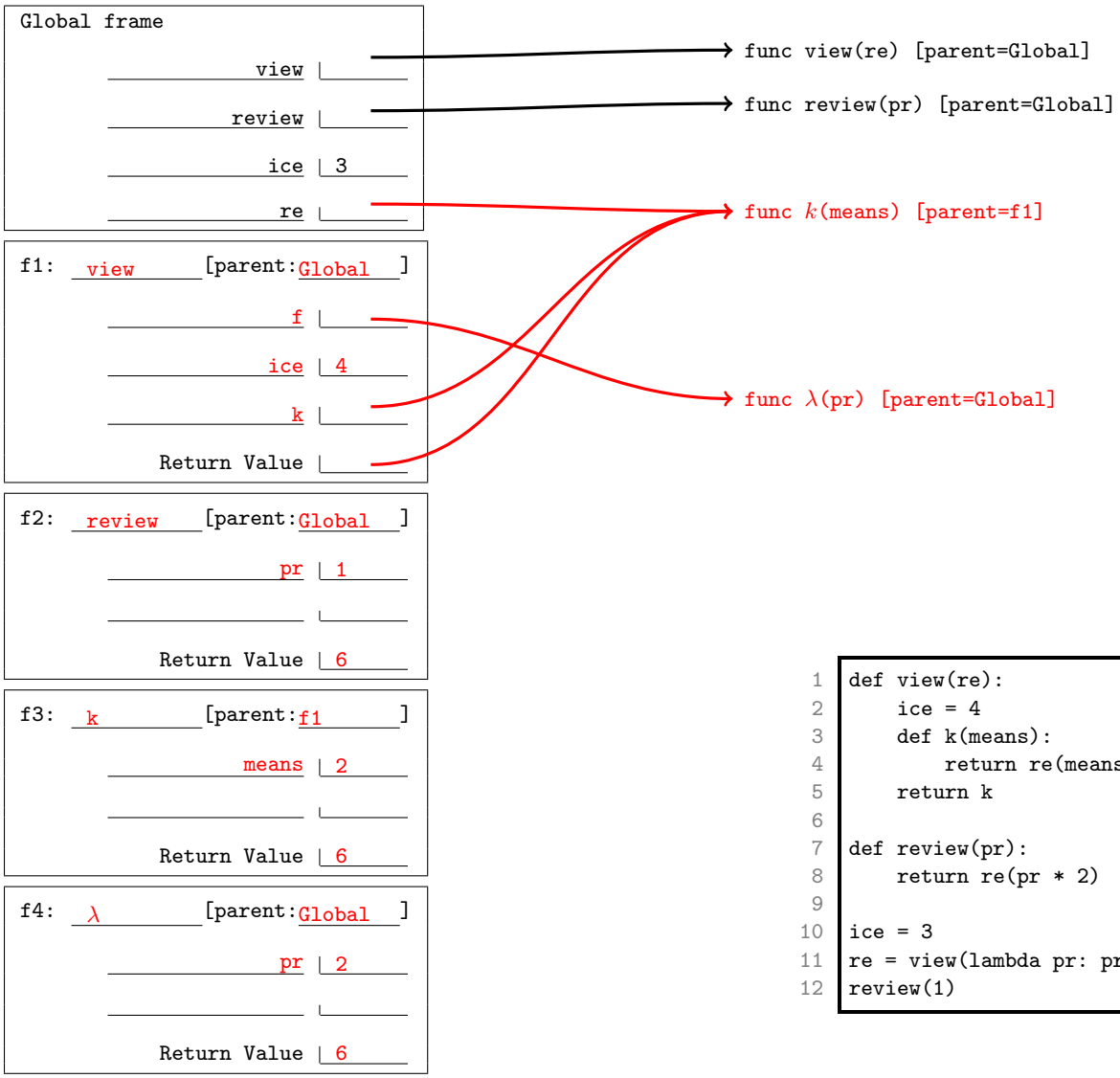
Expression	Interactive Output
`print(4, 5) + 1`	4 5 **Error**
`(p() and p()) or p()`	crikey crikey
`print(e(5))`	False True None
`f(x, y)`	2 9 11
`g(h, 4, 10)`	ers **Error**
`g(lambda x: x % 2, y, x)`	bonk 0

2. (10 points) Environment Diagram

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.




```
1  def view(re):
2      ice = 4
3      def k(means):
4          return re(means)
5      return k
6
7  def review(pr):
8      return re(pr * 2)
9
10 ice = 3
11 re = view(lambda pr: pr * ice)
12 review(1)
```

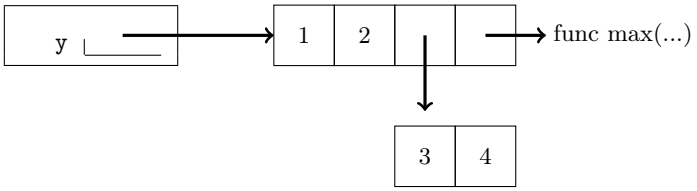
3. (8 points) Boxing Day

Draw box-and-pointer diagrams for the state of the lists after executing each block of code below. Each box should contain a value or a pointer to an object, as in an environment diagram.

You don't need to write index numbers or the word `list`. Please erase or cross out any boxes or pointers that are not part of a final diagram.

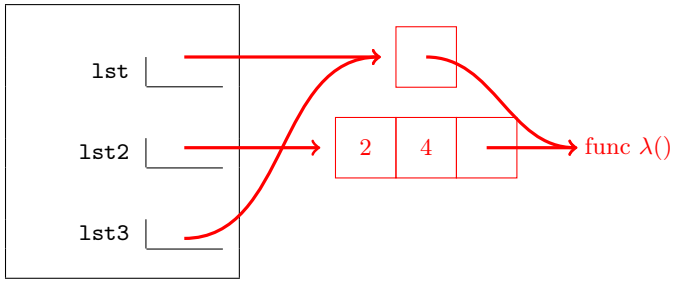
An example is given below:

```
1  y = [1, 2, [3, 4], max]
```



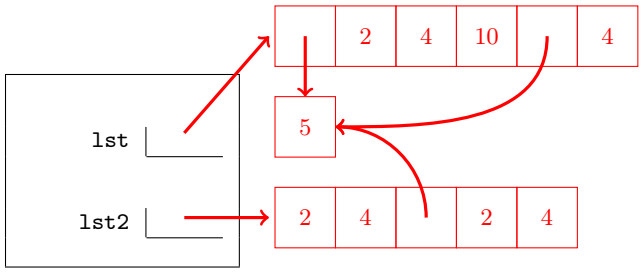
(a) (3 pt)

```
1  lst = [2, 4, lambda: lst]
2  lst2 = lst
3  lst = lst[2:]
4  lst3 = lst[0]()
```



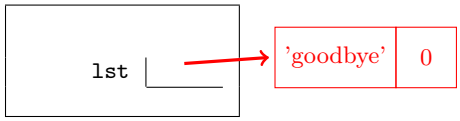
(b) (3 pt)

```
1  lst = [[5], 2, 4, 10]
2  lst2 = lst[1:3] + lst[:3]
3  for n in lst2[:2]:
4      lst.append(lst2[n])
```



(c) (2 pt)

```
1  lst = ['goodbye', 0, None, 8, 'hello', 1]
2  while lst.pop():
3      x = lst.pop()
4      if x:
5          lst.pop()
6      else:
7          lst.append('three')
```



4. (2 points) Conceptual Questions

There are three problems here, which cover topics from this week's lecture. Each problem is worth 1 point, but you can only earn a maximum of 2 points on this problem, so you only need to know two answers.

Please make sure to fill in the bubble completely when answering. Each question has only one right answer.

(a) (1 pt) Mutable Functions

Declaring a variable nonlocal allows you to modify a binding for that variable in:

◯ Any parent frame ◯ The global frame ● Any parent frame other than global ◯ The current frame

(b) (1 pt) Growth

Out of these options, what's the best way to measure the performance of a program?

◯ Run the program many times and calculate the average time it takes to run
● Count the number of operations executed by the program for different inputs
◯ Count how many while and for loops the code contains

(c) (1 pt) Iterators and Generators

Select the statement that is ***not*** true. A generator function is a function that...

● is an iterator ◯ returns an iterator ◯ returns a generator ◯ contains the "yield" keyword

5. (4 points) Negative Feedback

Write a function `clear_negatives` that takes in a list containing integers, and returns a list which is identical to the original, but in each case where a negative integer appears, delete a number of elements from the list equivalent to the magnitude of the negative integer.

For example, if -4 occurs in the list, remove a total of four elements: -4 and the next three elements. See the doctests for examples.

Your solution ***must*** use recursion, and should not modify the input list.

```
def clear_negatives(lst):
    """
    >>> clear_negatives([1, 0, 6])
    [1, 0, 6]
    >>> clear_negatives([3, -2, 1, 6, 2])
    [3, 6, 2]
    >>> lst = [-3, -4, 5, 5]
    >>> clear_negatives(lst)
    [5]
    >>> lst
    [-3, -4, 5, 5]
    """

    if not lst:

        return []

    elif lst[0] < 0:

        return clear_negatives(lst[-lst[0]:])

    else:

        return [lst[0]] + clear_negatives(lst[1:])
```

6. (6 points) Your Guess Is As Good As Mine

Let's play a guessing game! In order to do this, we'll use higher order functions.

Write a function, `make_guess`, which takes in a number that we want someone to try to guess, and returns a guessing function.

A *guessing function* is a one-argument function which takes in a number. If the number passed in is the number we wanted to guess, it will return the number of incorrect guesses made prior to the correct guess. Otherwise, it returns another guessing function, which keeps track of the fact that we've made an incorrect guess.

Solutions which use lists, object mutation, nonlocal, or global will receive **no credit.**

```
def make_guess(n):
    """Makes a guessing function for n.

    >>> guesser = make_guess(10)
    >>> guess1 = guesser(6)
    >>> guess2 = guess1(7)
    >>> guess3 = guess2(8)
    >>> guess3(10)
    3
    >>> guess2(10)
    2
    >>> a = make_guess(5)(1)(2)(3)(4)(5)
    >>> a
    4
    """
    def update_guess(num_incorrect):
        def new_guess(x):
            if x == n:
                return num_incorrect
            else:
                return update_guess(num_incorrect + 1)
        return new_guess
    return update_guess(0)
```

7. (12 points) Scrabbled Eggs

In this problem, we will build (part of) the ultimate Scrabble AI! Scrabble is a game where, given a collection of letters that each have a point value assigned, you try to make the word with the greatest score.

For example, if "a" and "s" are both worth 1 point, and "b" and "c" are both worth 2 points, the word "cab" would get 5 points, and "sass" would get 4 points.

(a) (8 pt) Implement the function `is_subseq` which takes two strings, `w1` and `w2`, and returns `True` if `w1` is a subsequence of `w2`.

Recall from Lab 5 that a string is a subsequence of another string if all the letters in the first string appear in the second string in the same order, although not necessarily directly next to each other. Put another way, you can transform the first string to the second string just by adding letters at any position in the first string.

For example, "put" and "cot" are both subsequences of "computer", but "car" and "rot" are not.

```
def is_subseq(w1, w2):
    """ Returns True if w1 is a subsequence of w2 and False otherwise.

    >>> is_subseq("word", "word")
    True
    >>> is_subseq("compute", "computer")
    True
    >>> is_subseq("put", "computer")
    True
    >>> is_subseq("computer", "put")
    False
    >>> is_subseq("sin", "science")
    True
    >>> is_subseq("nice", "science")
    False
    >>> is_subseq("boot", "bottle")
    False
    >>> is_subseq("a", "bottle")
    False
    """

    if not w1:
        return True
    elif not w2:
        return False
    else:
        with_elem = w1[0] == w2[0] and is_subseq(w1[1:], w2[1:])
        without_elem = is_subseq(w1, w2[1:])
        return with_elem or without_elem
```

(b) (4 pt) Now, implemment `scrabbler` which takes in a string `chars`, a list of strings `words`, and a dictionary `values` which maps letters to point values. It should return a dictionary where each key is a word in `words` which can be formed from the letters in `chars` and each value is the point value of that word.

For the purposes of this problem, we will only consider words we can form by using letters in `chars` in the same order they appear in the string. Assume `values` contains all the letters across valid words as keys. You may additionally assume `is_subseq` from the previous part has been implemented correctly.

```
def scrabbler(chars, words, values):

    """
    >>> words = ["easy", "as", "pie"]
    >>> values = {"e": 2, "a": 2, "s": 1, "p": 3, "i": 2, "y": 4}
    >>> scrabbler("heuaiosby", words, values)
    {'easy': 9, 'as': 3}
    >>> scrabbler("piayse", words, values)
    {'pie': 7, 'as': 3}
    """

    result = {}

    for w in words:

        if is_subseq(w, chars):

            total = sum([values[c] for c in w])

            result[w] = total

    return result
```

8. (8 points) Leaf It To Me

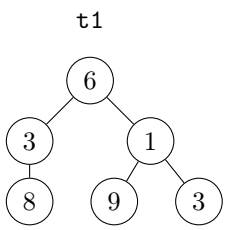
Write a function `max_path` which takes in a tree `t` with positive labels and a number `k`, and returns the path with length at most `k` for which the sum of the labels in the path is greatest. If there are multiple paths with the greatest sum, return the leftmost one.

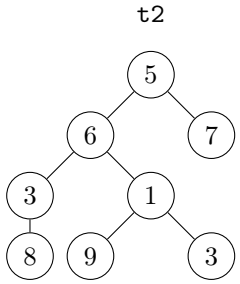
The path does *not* have to start at the root of the tree - the path can contain any top-to-bottom sequence of nodes.

```
def tree(label, branches=[]):
    return [label] + list(branches)

def is_leaf(t):
    return not branches(t)

def label(t):
    return t[0]

def branches(t):
    return t[1:]
```

The tree data abstraction is provided here for your reference. Do not violate the abstraction barrier!

```
def max_path(t, k):
    """ Return a list of the labels on any path in tree t of length at most k with the greatest sum

    >>> t1 = tree(6, [tree(3, [tree(8)]), tree(1, [tree(9), tree(3)])])
    >>> max_path(t1, 3)
    [6, 3, 8]
    >>> max_path(t1, 2)
    [3, 8]
    >>> t2 = tree(5, [t1, tree(7)])
    >>> max_path(t2, 1)
    [9]
    >>> max_path(t2, 2)
    [5, 7]
    >>> max_path(t2, 3)
    [6, 3, 8]
    >>> max_path(t1, 4)
    [6, 3, 8]
    """
    def helper(t, k, on_path):
        if k == 0:
            return []
        elif is_leaf(t):
            return [label(t)]
        a = [[label(t)] + helper(b, k - 1, True) for b in branches(t)]
        if on_path:
            return max(a, key = sum)
        else:
            b = [helper(b, k, False) for b in branches(t)]
            return max(a + b, key = sum)
    return helper(t, k, False)
```





9. (0 points) The Prisoner's Dilemma

In this extra credit problem, you may choose one of two options.

- Mark the choice to "Betray" and write a positive integer in the blank below. The one student who writes the *smallest, unique positive integer* will receive *two* (2) extra credit points but only if fewer than 90% of students choose the next option.
- Mark the choice to "Work Together". If at least 90% of students choose this option, all students who chose this option will receive *one* (1) extra credit point and those who marked the choice to "Betray" will receive zero (0) extra credit points.

Will you *work together*? Or will you *betray* your fellow students? It is up to you.

○ Betray _____

● Work Together

10. (0 points) The End

(a) (0 pt) Any feedback for us on how this exam went / how the course is going so far?

(b) (0 pt) Use this space to draw a picture, write a note, or otherwise express yourself.