CS 61A Structure and Interpretation of Computer Programs

Summer 2015

MIDTERM 2

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 and official CS 61A midterm 2 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	
BearFacts 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)**	

1. (10 points) Shut up and take my money

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. If an error occurs, write "Error".

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

Assume that you have started `python3` and executed the following statements:

```
class Singer:
  msg = 'Shut up and '
  count = 1
  lst = []

  def __init__(self, name, x):
    self.x = x

  def sing(self):
    Singer.count = Singer.count + 1
    print(self.msg + self.x)
    self.lst.append(self.count)
```

```
class Group(Singer):
  def __init__(self, s1, s2):
    self.s1 = s1
    self.s2 = s2

  def sing(self):
    self.count = self.count + 1
    self.s1.sing()
    self.s2.sing()
    self.s1.x, self.s2.x = self.s2.x, self.s1.x
    self.lst.append(self.count)

w_moon = Singer('WALK THE MOON', 'dance')
rihanna = Singer('Rihanna', 'drive')
w_ri = Group(w_moon, rihanna)
w_ri.lst = []
```

For the following lines, assume they are executed in the same interactive session; **the result of executing one line may affect the result of executing all following lines**. The statements on the right are executed after the statements on the left.

Expression	Interactive Output
`pow(2, 3)`	8
`print(4, 5) + 1`	4 5 Error
`w_moon.lst is w_ri.lst`	
`w_moon.sing()`	
`w_moon.count`	
`Group.count`	

Expression	Interactive Output
`w_ri.s2.sing()`	
`w_ri.sing()`	
`Group.sing(w_moon)`	
`Group.sing(w_ri)`	
`min(rihanna.lst, key=lambda x: -x)`	
`w_ri.lst`	

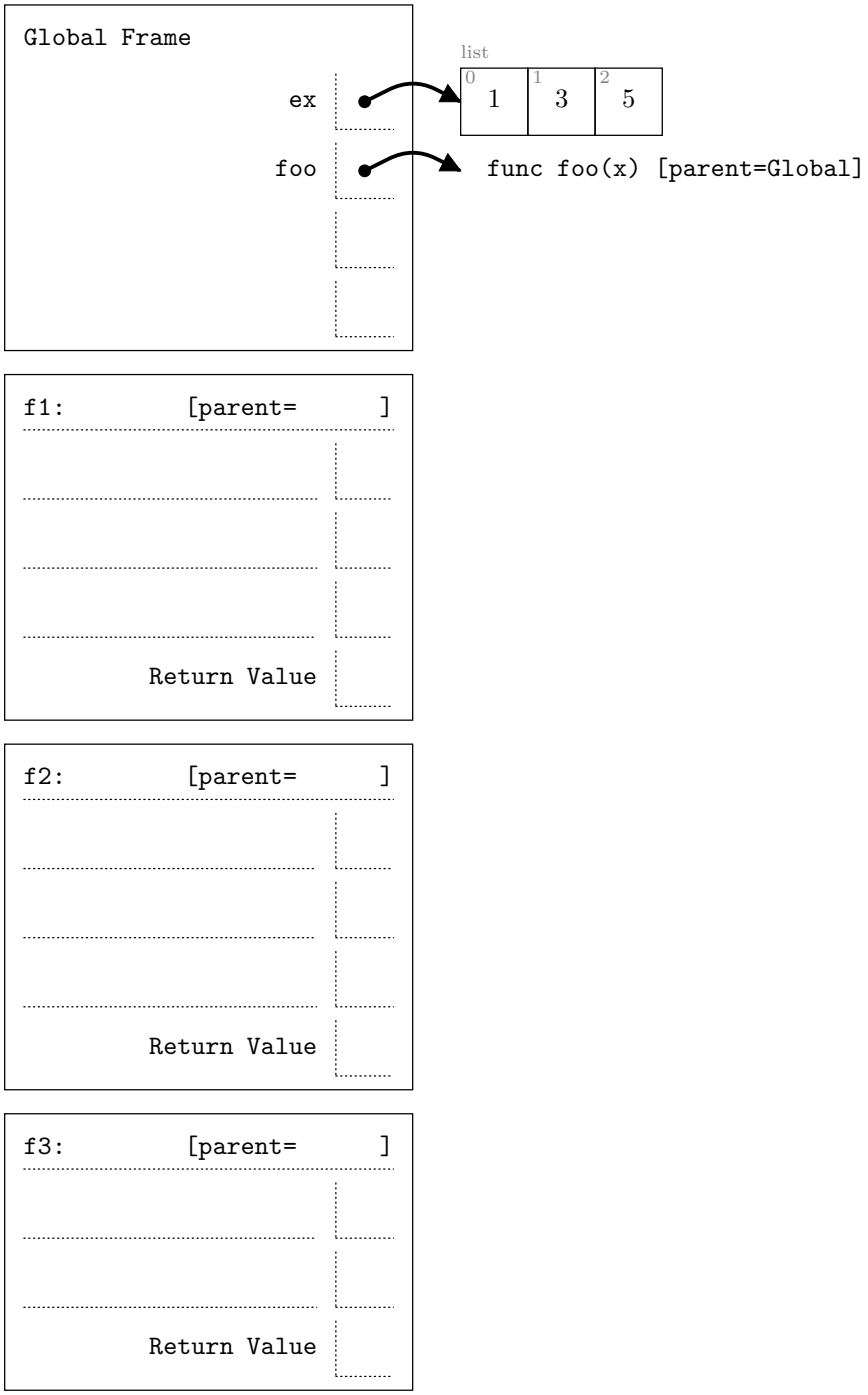
2. (15 points) Fly, you foos!

(a) (7 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 only need to show the final state of each frame. *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  ex = [1, 3, 5]
2
3  def foo(x):
4      bar[x] = x
5      bar.append(len(bar))
6      return bar
7
8  bar = [1, [2]]
9
10 foo(0)
11 foo(2)
12 foobar = bar[:]
13 foobar[1][0] = foobar
```

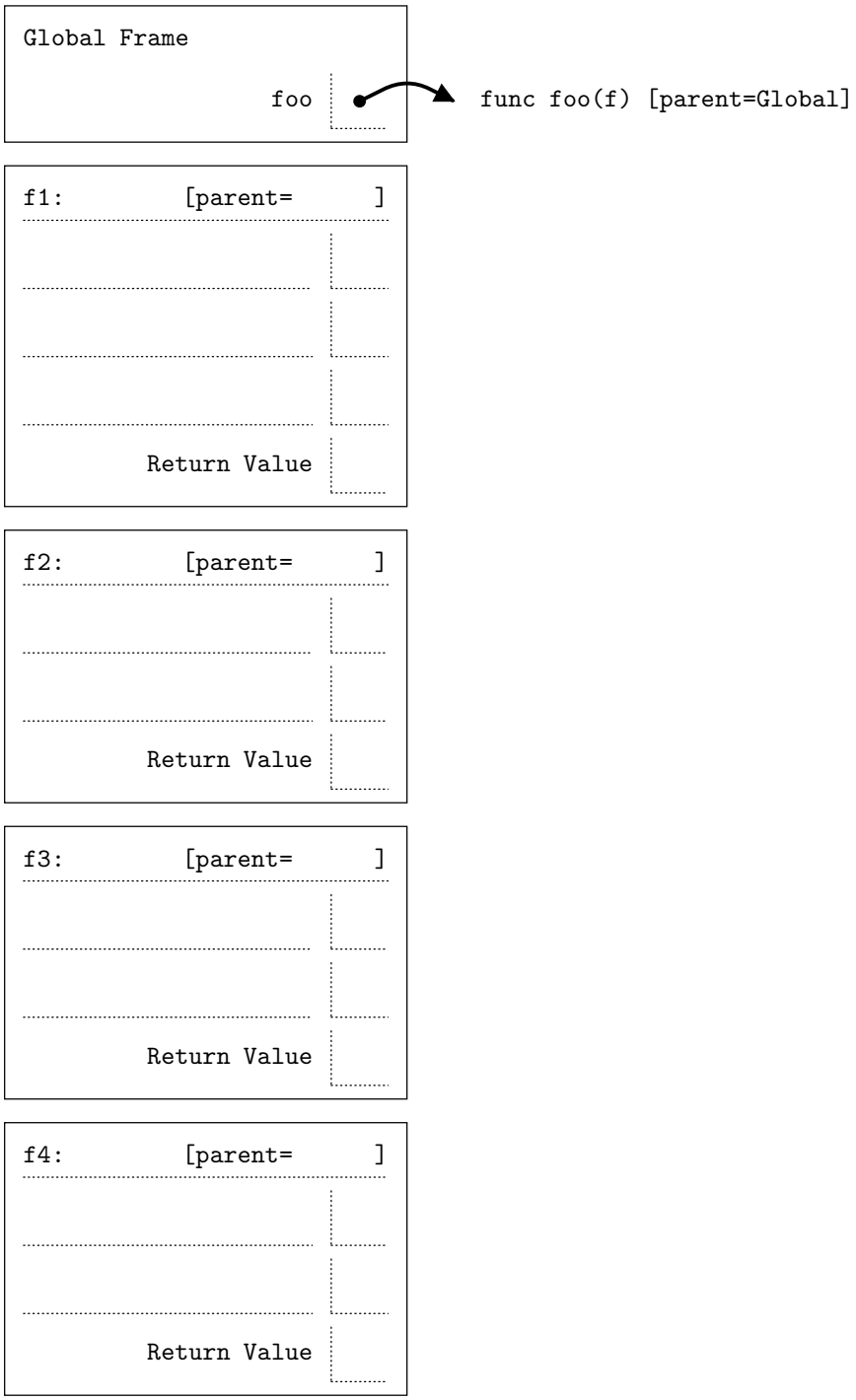


(b) **(8 pt)** Fill in the environment diagram that results from executing the code below until the entire program is finished, an error occurs, or all frames are filled. You only need to show the final state of each frame. *You may not need to use all of the spaces or frames.*

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 foo(f):
2      def bar(x):
3          return f(x) + f(x)
4
5      def f(x):
6          nonlocal f
7          f = lambda x: x + 2
8          return x - 1
9
10     return bar
11
12 foo(lambda x: 2*x)(2)
```



3. (5 points) Counter intuitive

(a) (3 pt) Implement the `__init__` method of the `Counter` class. The constructor for `Counter` takes in a list of strings and stores a dictionary called `counts` whose keys are elements of the list and whose values are the number of times each element appeared.

See the doctests for an example. **Your solution should not require more than 5 additional lines, and your solution does not need to use all 5 lines.**

```
class Counter:
    """
    >>> dog = Counter(['this', 'is', 'spot', 'see', 'spot', 'run'])
    >>> dog.counts['this']
    1
    >>> dog.counts['spot']
    2
    >>> 'catdog' in dog.counts
    False
    """

    def __init__(self, lst):

        self.counts = {}


        ------------------------------------------------------------------


        ------------------------------------------------------------------


        ------------------------------------------------------------------


        ------------------------------------------------------------------


        ------------------------------------------------------------------
```

(b) (2 pt) It would be convenient if we could use indexing notation on our `Counter` objects instead of having to access its `counts` attribute:

```
>>> sappy = Counter(['romeo', 'romeo', 'wherefore', 'art', 'thou', 'romeo'])
>>> sappy['romeo']
3
>>> sappy['thou']
1
>>> sappy['paris']
0
```

Notice that, because `'paris'` did not appear as an element in the input to `Counter`, we get 0 when we use it as a key (instead of getting a `KeyError`).

Implement the `__getitem__` method to support this sort of behavior for `Counter` objects. You may assume that the `__init__` method is implemented correctly.

Your solution should not require more than 4 lines, and your solution does not need to use all 4 lines.

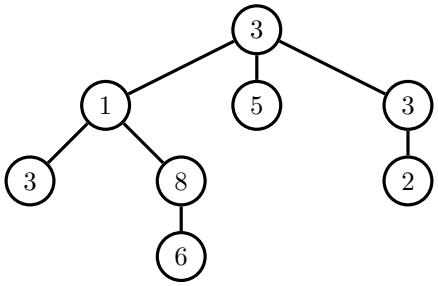
```
class Counter:
    # Code from previous question

    def __getitem__(self, key):
        """
        >>> hamlet = Counter(['to', 'be', 'or', 'not', 'to', 'be'])
        >>> hamlet['be']
        2
        >>> hamlet['not']
        1
        >>> hamlet['whiny']
        0
        """
```

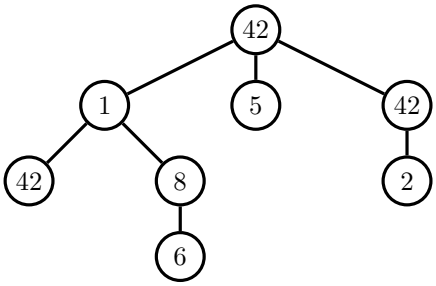
---

4. (7 points) Trees? What a reLEAF!

(a) (4 pt) Implement `find_and_replace(t, old, new)`, where `t` is an instance of the `Tree` class. The function `find_and_replace` mutates `t` such that every entry that is equal to `old` is replaced with `new`.



`t`, an instance of `Tree`



`find_and_replace(t, 3, 42)` mutates `t` to look like this.

The `Tree` class has been provided for you below. **Your solution should not require more than 7 lines, and you do not need to use all 7 lines.**

```
class Tree:
    def __init__(self, entry, subtrees=[]):
        self.entry = entry
        self.subtrees = list(subtrees)
    def is_leaf(self):
        return not self.subtrees

def find_and_replace(t, old, new):
```

(b) **(3 pt)** Implement `is_bst(b)`, which takes an instance of the `BinaryTree` class and returns `True` if it is a valid **binary search tree**. By definition, an empty `BinaryTree` is a valid binary search tree.

You may use the provided `tree_max` function (which calculates the largest element in a `BinaryTree`) and `tree_min` function (which calculates the smallest element in a `BinaryTree`. The input tree to `tree_max` and `tree_min` is not allowed to be empty.

The `BinaryTree` class has also been provided for you below.

```
class BinaryTree:
    empty = ()
    def __init__(self, entry, left=empty, right=empty):
        self.entry = entry
        self.left = left
        self.right = right

def tree_max(b):
    """Returns the largest element in a non-empty BinaryTree."""
    # Implementation omitted.

def tree_min(b):
    """Returns the smallest element in a non-empty BinaryTree."""
    # Implementation omitted.

def is_bst(b):


    if b == BinaryTree.empty:


        return True


    elif b.right != BinaryTree.empty and _______________________________:


        return False


    elif b.left != BinaryTree.empty and _______________________________:


        return False


    else:


        return ____________________________________________________________
```

5. (13 points) The weakest link

(a) (2 pt) For the following questions, assume that the following generator function is defined:

```
def naturals():
    i = 1
    while True:
        yield i
        i += 1
```

Implement a generator function called `filter(iterable, fn)` that only yields elements of `iterable` for which `fn` returns `True`.

See the doctests for expected behavior. **You may not use the built-in `filter` function or list comprehensions.**

Your solution should not require more than 3 lines, and you do not need to use all 3 lines.

```
def filter(iterable, fn):
    """
    >>> is_even = lambda x: x % 2 == 0
    >>> list(filter(range(5), is_even))
    [0, 2, 4]

    >>> all_odd = (2 * y - 1 for y in range(5))     # Generator object
    >>> list(filter(all_odd, is_even))
    []

    >>> s = filter(naturals(), is_even)
    >>> next(s)
    2
    >>> next(s)
    4
    """
```

(b) (5 pt) Implement an iterator class called `Filter`. The `__init__` method for `Filter` takes an iterable and a one-argument function that either returns `True` or `False`. The `Filter` iterator represents a sequence that only contains elements of the iterable for which the predicate function returns `True`.

See the doctests for expected behavior. **You may not use the built-in `filter` function or list comprehensions.**

Your solution should not require more than 6 lines, and you do not need to use all 6 lines.

```
class Filter:
    """
    >>> is_even = lambda x: x % 2 == 0
    >>> list(Filter(range(5), is_even))
    [0, 2, 4]

    >>> all_odd = (2 * y - 1 for y in range(5))     # Generator object
    >>> list(Filter(all_odd, is_even))
    []

    >>> s = Filter(naturals(), is_even)
    >>> next(s)
    2
    >>> next(s)
    4
    """

    def __init__(self, iterable, fn):


        self.fn = fn


        ------------------------------------------------------------


    def __iter__(self):

        return self

    def __next__(self):


        ------------------------------------------------------------


        ------------------------------------------------------------


        ------------------------------------------------------------


        ------------------------------------------------------------


        ------------------------------------------------------------
```

(c) (4 pt) Implement `trim(s, index)`, where `s` is an instance of the `Link` class and `index` is a non-negative integer. The `trim` function mutates `s` so that all elements that occur after the `index` are removed. The `trim` function should also return a linked list of all of the removed elements. If the `index` exceeds the linked list, raise an `IndexError`. **Your solution must use recursion; you may not use iteration.**

You may NOT assume that `__getitem__` and `__len__` have been implemented.

See the doctest for expected behavior. **Your solution should not require more than 8 lines, and you do not need to use all 8 lines.**

```
class Link:
    empty = ()
    def __init__(self, first, rest=empty):
        self.first = first
        self.rest = rest

def trim(s, index):
    """
    >>> s1 = Link(1, Link(2, Link(3)))
    >>> trim(s1, 0)                  # Remove and return items after index 0
    Link(2, Link(3))
    >>> s1                           # s1 has been mutated
    Link(1)
    >>> s2 = Link(1, Link(2, Link(3)))
    >>> trim(s2, 2) == Link.empty    # No elements after index 2
    True
    >>> s2
    Link(1, Link(2, Link(3)))
    >>> s3 = Link(1, Link(2, Link(3)))
    >>> trim(s3, 3)                  # s3 has no index 3
    IndexError
    """
```

(d) (2 pt) Consider the following linked list functions:

```
def append(link, value):
    """Mutates link by adding value to the end of link."""
    if link.rest is Link.empty:
        link.rest = Link(value)
    else:
        append(link.rest, value)

def extend(link1, link2):
    """Mutates link1 so that all elements of link2 are added to the end
    of link1.
    """
    while link2 is not Link.empty:
        append(link1, link2.first)
        link2 = link2.rest
```

Circle the order of growth that best describes the runtime of calling `append`, where n is the number of elements in the input `link`.

$O(1)$ $O(\log n)$ $O(n)$ $O(n^2)$ $O(2^n)$

Assuming the two input linked lists to `extend` both contain n elements, circle the order of growth that best describes the runtime of calling `extend`.

$O(1)$ $O(\log n)$ $O(n)$ $O(n^2)$ $O(2^n)$

6. (0 points) A second chance

In each of the two boxes below, write a positive integer. If one of the numbers you pick is the lowest unique integer in the class, you get one extra credit point. In other words, you get two chances to write the smallest positive integer that you think no one else will write.