Midterm Review

CS61A Summer 2016

Katya Stukalova
Jerome Baek

Announcements

- Time: 5:00PM to 8:00PM, Thursday, 7/14
- Place: 2050 VLSB (right here!)
- Check https://piazza.com/class/ipkfex1ne3p56y?cid=773
- You can bring an 8.5”x 11” cheat sheet, front and back
- These slides will be posted on Piazza

The Plan...

Pitfalls

Topics

- **Environment diagrams**
- While loops and for loops
- **Higher order functions**
- Lambda functions
- **Recursion and tree recursion**
- Orders of growth
- Lists & sequences
- Data abstraction
- **Linked lists**
- **Trees**

Environment Diagrams

Name of frame should be *intrinsic* name of function

```
def f():
    ...
def g():
    ...
f = g
f()
```

What is the name of the frame created by the last line?

g

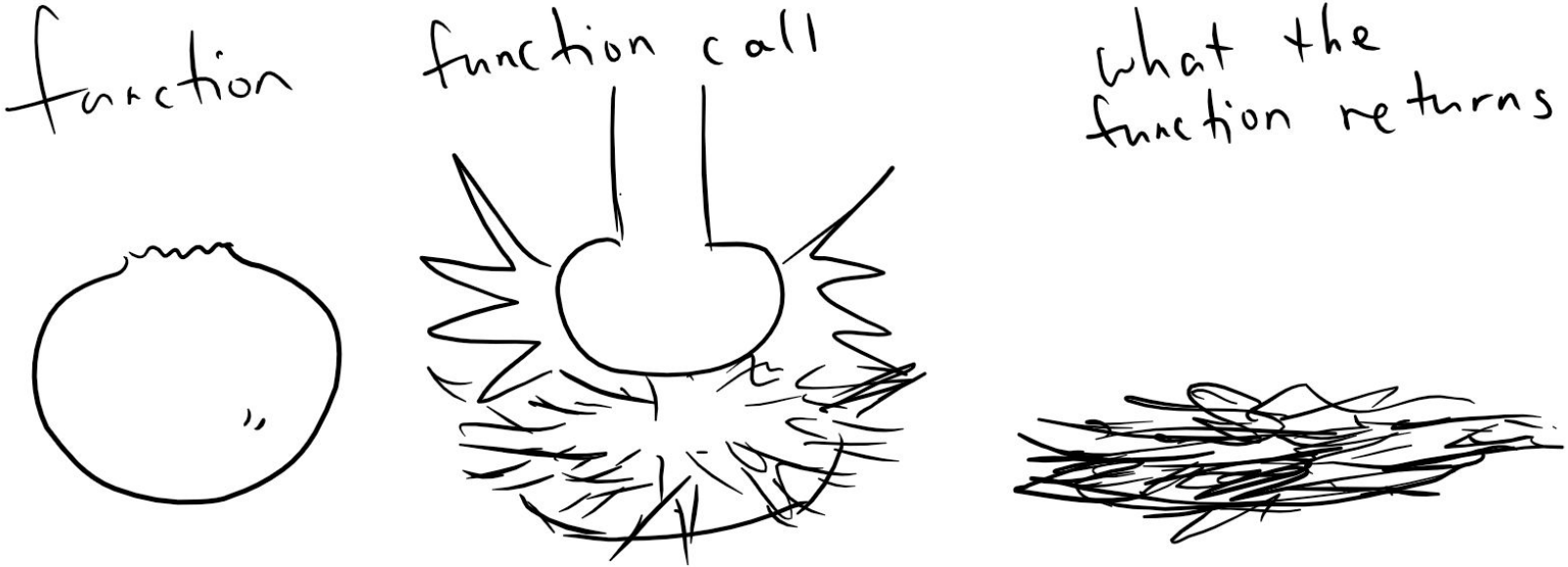
Lambda functions are defined when...

```
def f(g):
    y = 2
    return g(2)
y = 1
print(f(lambda x: x + y))
```

What number is printed? **3**

What is the parent of the lambda function? **Global**

Function Name vs. Function Call



Environment Diagram Question

```
def marvin(brain):
    cs61a = midterm+2
    return cs61a + brain(midterm+1)
def tammy(marvin):
    marvin, midterm = marvin+2, marvin+1
    return midterm // cs61a

midterm, cs61a = 3, 2
marvin(tammy)
```

Environment Diagram Pitfalls

1. Name of frame should be **intrinsic** name of function
2. Lambda functions are defined where they are **evaluated**
3. Parent frame of a function never changes once you write it down
4. Don't conflate: function name vs. function call
5. Calling a function
    a. Evaluate the operator (usually a lookup)
    b. Evaluate the operands
    c. Apply the operator on the operands (this is where you actually call the function and make a new frame)

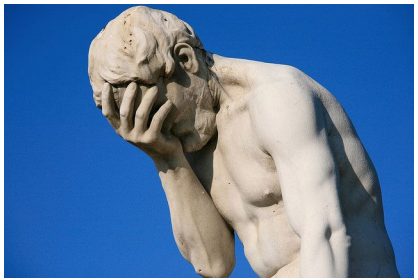
Lists and sequences

List and Sequences Pitfalls

1. Whenever you see a negative number, like -n, just replace it with len(lst) - n

```
lst[-3] == lst[len(lst)-3]

lst[-2:3] == lst[len(lst)-2:3]
```



2. In list slicing, if you go out of bounds, you DON'T error, you just return as much as you can
3. List slicing ALWAYS returns a list

Deep Length

The function deep_len takes a deep list as input and returns the deep length of the list. Fill in the blanks.

```
"""
>>> deep_len([1, 2, 3]) # normal list
3
>>> x = [1, [2, 3], 4] # deep list
>>> deep_len(x)
4
>>> x = [[1, [1, 1]], 1, [1, 1]]
>>> deep_len(x)
6
"""
```

```
def deep_len(lst):
    if not lst:
        return 0
    elif type(lst[0]) == list:
        return deep_len(lst[0]) + deep_len(lst[1:])
    else:
        return 1 + deep_len(lst[1:])
```

Higher Order Functions

How are the following pieces of code different?

What would Python display for each?

```
t = “surprise!”
def outer(t):
    def inner():
        print(t)
    return inner
outer(“boo!”)()
```

```
t = “surprise!”
def inner():
    print(t)
def outer(t):
    return inner
outer(“boo!”)()
```

Draw env diagrams to see what's different!

PythonTutor

Fun Multiply

```
“””
>>> def func_a(num):
...     return num + 1
>>> func_b1 = fun_mult(func_a, 3)
>>> func_b1(2)
4
>>> func_b2 = fun_mult(func_a, -2)
>>> func_b2(-3)
>>> func_b3 = fun_mult(func_a, -1)
>>> func_b3(4)
>>> func_b4 = fun_mult(func_a, 0)
>>> func_b4(3)
6
>>> func_b5 = fun_mult(func_a, 1)
>>> func_b5(4)
24
”””
```

Fill in the blanks so that the doctests pass.

```
def fun_mult(func):
    def func_b(stop):
        i = __start___
        product = 1
        if start < 0:
            return ___None___
        if start > stop:
            return _func_a(start)_____
        while i < stop:
            product = product * func_a(i)
            i += 1
        return __product__
    return func_b
```

Higher Order Functions Pitfalls

1. Function name vs. function call
2. Parent of the function is the frame in which the function was defined
3. Don't be freaked out by things like f(3)(2)(6)

Recursion and Tree Recursion

Recursive Remove

```
def remove (n , digit ):
    """
    Return a number that is identical to n, but with all instances
    of digit removed. Assume that DIGIT is a positive integer less
    than 10.
    """
    if n == 0:
        return 0
    if n % 10 == digit:
        return remove(n // 10, digit)
    else:
        return n % 10 + 10 * remove(n // 10, digit)
```

FooBar

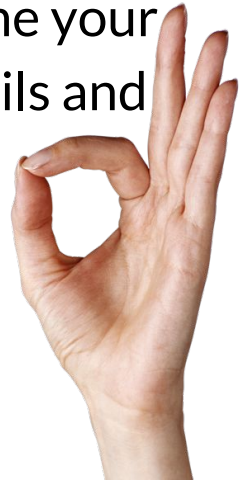
Write the function foobar that behaves as follows:

```
>>> foobar(0)
"foo"
>>> foobar(1)
"foobar"
>>> foobar(2)
"foobarbar"
>>> foobar(3)
"foobarbarfoo"
>>> foobar(4)
"foobarbarfoobar"
>>> foobar(14)
"foobarbarfoobarbarfoobarbarfoobarbarfoobarbar"
```

```
def foobar(n):
  if n == 0:
    return "foo"
  elif n % 3 == 0:
    return foobar(n-1) + "foo"
  else:
    return foobar(n-1) + "bar"
```

Recursion Pitfalls

1. **PLEASE** consider the TYPE of input and output to the function
2. A recursive function must **ALWAYS** return a value of the same type!!!
    a. **BAD**: returning `first(link)` when you should return a linked list
3. Take the leap of faith! Be confident - thought is recursive. Assume your solution is correct and you'll be correct. Assume your solution fails and you will fail.
4. The input to the recursive call MUST be closer to the base case
    a. Otherwise, you get stuck in recursive calls forever!



Addup

Write a function that takes as input a number `n` and a list of numbers `lst` and returns True if we can find a subsequence of lst that sums up to n

```
>>> addup(10, [1, 2, 3, 4, 5])
True
>>> addup(8, [1, 2, 3, 4, 5])
True
>>> addup(-1, [1, 2, 3, 4, 5])
False
>>> addup(100, [1, 2, 3, 4, 5])
False
```

```
def addup(n, lst):
    if n == 0:
        return True

    if lst == []:
        return False

    else:
        first, rest = lst[0], lst[1:]
        return addup(n-first, rest) or \
            addup(n, rest)
```

Tree Recursion Tips

1. **LOOK AT** the TYPE of input and output to the function
    a. BAD: calling `f(children(tree))` when `f` takes in a tree
2. A recursive function must ALWAYS return a value of the same type!!!
3. Think of the *logic* of the function, think of what the function *should* return, **take the leap of faith**!

Orders of growth

Orders of Growth Tips

1. There is no sure and fast way to determine the order of growth of a function.
2. Read the function definition carefully and make sure you understand exactly what the function is doing.

Find the Orders of Growth

```
def f(n):
    if not not not False:
        return
    else:
        return f(n - 1)
```

Constant!

```
def fun(x):
    for i in range(x):
        for j in range(x * x):
            if j == 4:
                return -1
            print("fun!")
```

Constant!

```
def belgian_waffle(n):
    i = 0
    sum = 0
    while i < n:
        for j in range (n**2):
            sum +=1
            i += sum
        return sum
```

n^3

Linked Lists

Count

Define a function count which takes in a linked list, lnk, and a list of numbers, nums, and returns the number of values in nums that appear in lnk [Hint: practice your list comprehensions! :)]

```
def count(lnk, nums):
    if lst == empty:
        return 0
    if first(lnk) in nums:
        return 1 + count(rest(lnk),\
                         [x for x in nums if x != first(lnk)])
    return count(rest(lnk), nums)
```

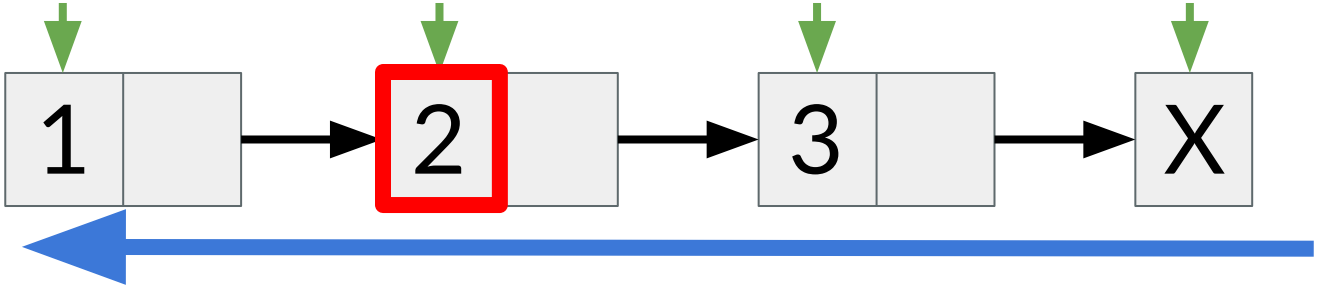
Kth to Last.

Write a function that returns the kth to last element of a linked list.

```
def kth_to_last(l):
    """
    >>> lst = link(1, link(2, link(3)))
    >>> kth_to_last(lst, 0)
    3
    >>> kth_to_last(lst, 1)
    2
    >>> print(kth_to_last(lst, 5))
    None
    """
```

Here's an approach

Recurse until you hit the empty list



k = 1

Once k is 0, you must return the first element of the current list

When you return back to the front of the list through your recursive calls, decrement k by 1

Kth to Last.

```
def kth_last(lst, k):
    def unwind_rewind(lst):
        if lst == empty:
            return k, None
        previous_k, kth_element = unwind_rewind(rest(lst))
        if previous_k == 0:
            return previous_k - 1, first(lst)
        else:
            return previous_k - 1, kth_element
    return unwind_rewind(lst)[1]
```

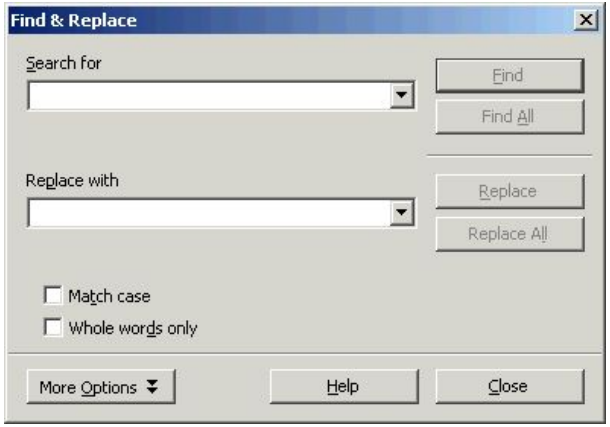
Trees

Tree Tips

1. Children of a tree is a list of trees
2. Recursive calls go vertically in the tree, for loops go horizontally

Find and Replace

Implement the function find_and_replace which takes in a tree t, and two values, old and new. The function returns a tree that is identical to the original, but with all instances of old replaced with new.




```
def find_and_replace(t, old, new):
    kept_children = []
    for c in children(t):
        kept_children += find_and_replace(c, old, new)
    if entry(t) == old:
        return tree(new, branches)
    return tree(entry(t), branches)
```

Binary Tree

Write a function that takes in a tree, t, and returns True if every node has at most two children and False otherwise.

```
def is_binary_tree(t):
    if len(children(t)) > 2:
        return False
    final_result = True
    for c in children(c):
        final_result = final_result and is_binary_tree(c)
    return final_result
```

Thanks for coming!

Good luck on the midterm!