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()
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
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 \(\text{len(lst)} - n\)

   \[
   \text{lst}\left[-3\right] == \text{lst}[\text{len(lst)}-3] \\
   \text{lst}\left[-2:3\right] == \text{lst}[\text{len(lst)}-2:3]
   \]

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 \textit{ALWAYS} returns a list
The function deep_len takes a deep list as input and returns the deep length of the list. Fill in the blanks.

```python
def deep_len(lst):
    if not lst:
        return ______________
    elif type(lst[0]) == list:
        return _______________ + ________________
    else:
        return ______________
```

```bash
>>> 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
```

Deep Length
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
Fill in the blanks so that the doctests pass.

```python
def fun_mult(func_a, start):
    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
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)
Write the function `foobar` that behaves as follows:

```python
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!
Write a function that takes as input a number \( n \) and a list of numbers \( \text{lst} \) and returns True if we can find a subsequence of \( \text{lst} \) 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)

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

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

Constant!

Constant!

\textbf{Constant!}

\textbf{n^3}
Linked Lists
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`. Assume that all entries in `nums` are distinct.

[Hint: practice your list comprehensions! :)]

```python
def count(lnk, nums):
    if lnk == 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)
```
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

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
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`.

```python
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, kept_children)
    return tree(entry(t), kept_children)
```
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.

```python
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!