CS 61A Recursion and Tree Recursion, HOFs
Summer 2019 Guerrilla Section 1: July 5, 2019

1 Recursion and Tree Recursion

Questions

1.1 What are three things you find in every recursive function?

1) Base cases (at least 1 case that covers smallest form of the problem)
2) Way(s) to reduce the problem (this smaller problem is the argument in recursive call)
3) Way(s) to apply solution of each reduced problem to solve larger problem

1.2 When you write a Recursive function, you seem to call it before it has been fully defined. Why doesn't this break the Python interpreter?

The interpreter does not execute the body of the recursive function when it is defined

1.3 Below is a Python function that computes the nth Fibonacci number. Identify the three things it contains as a recursive function (from 1.1).

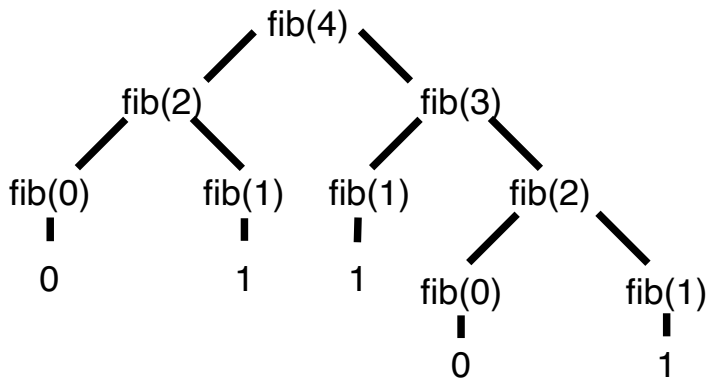
```
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-1) + fib(n-2)
```

Base cases: The two smallest Fibonacci numbers are 0 and 1

Reduce problem: Function is called on smaller inputs – fib(n-1) and fib(n-2)

Apply solution of smaller problem: The results of fib(n-1) and fib(n-2) are added together to find fib(n)

1.4 With the definition of the Fibonacci function above, draw out a diagram of the recursive calls made when **fib(4)** is called.



1.5 What does the following function **cascade2** do? What is its domain and range?

```
def cascade2(n):
    print(n)
    if n >= 10:
        cascade2(n//10)
        print(n)
```

The function cascade2 first prints all the digits of n. Next the last digit of n is cut off and n is printed again. This process continues until only the first digit of n is printed. After that, it will print the first two digits of n, then the first 3 digits of n, and so on. The function exits once all the digits of n are printed again. For example, cascade2(583) will print:

583
58
5
58
583

The domain of this function is all positive integers. The range of this function is None since there is no explicit return statement.

1.6 Consider an insect in an `M` by `N` grid. The insect starts at the bottom left corner, `(0, 0)`, and wants to end up at the top right corner `(M-1, N-1)`. The insect is only capable of moving right or up. Write a function `paths` that takes a grid length and width and returns the number of different paths the insect can take from the start to the goal. (There is a closed-form solution to this problem, but try to answer it procedurally using recursion.)

```
def paths(m, n):
    """
    >>> paths(2, 2)
    2
    >>> paths(117, 1)
    1
    """
    if m == 1 or n == 1:
        return 1
    else:
        return paths(m - 1, n) + paths(m, n - 1)
```

1.7 Write a procedure `merge(s1, s2)` which takes two sorted (smallest value first) lists and returns a single list with all of the elements of the two lists, in ascending order. Use recursion.

Hint: If you can figure out which list has the smallest element out of both, then we know that the resulting merged list will have that smallest element, followed by the merge of the two lists with the smallest item removed. Don't forget to handle the case where one list is empty!

```
def merge(s1, s2):
    """ Merges two sorted lists
    >>> merge([1, 3], [2, 4])
    [1, 2, 3, 4]
    >>> merge([1, 2], [])
    [1, 2]
    """
    if len(s1) == 0:
        return s2
    elif len(s2) == 0:
        return s1
    else:
        if s1[0] < s2[0]:
            return [s1[0]] + merge(s1[1:], s2)
        else:
            return [s2[0]] + merge(s1, s2[1:])
```

1.8 Mario needs to jump over a sequence of Piranha plants, represented as a string of spaces (no plant) and P's (plant!). He only moves forward, and he can either step (move forward one space) or jump (move forward two spaces) from each position. How many different ways can Mario traverse a level without stepping or jumping into a Piranha plant? Assume that every level begins with a space (where Mario starts) and ends with a space (where Mario must end up):

Hint: You can get the ith character in a string 's' by using 's[i]'. For example,

```
>>> s = 'abcdefg'
>>> s[0]
'a'
>>> s[2]
'c'
```

You can find the total number of characters in a string with the built-in'len' function:

```
>>> s = 'abcdefg'
>>> len(s)
7
>>> len('')
0
```

```
def mario_number(level):
    """Return the number of ways that Mario can perform a sequence of steps
    or jumps to reach the end of the level without ever landing in a Piranha
    plant. Assume that every level begins and ends with a space.
```

```
>>> mario_number(' P P ')   # jump, jump
1
>>> mario_number(' P P  ')   # jump, jump, step
1
>>> mario_number('  P P ')  # step, jump, jump
1
>>> mario_number('   P P ') # step, step, jump, jump or jump, jump, jump
2
>>> mario_number(' P PP ')  # Mario cannot jump two plants
0
>>> mario_number('    ')    # step, jump ; jump, step ; step, step, step
3
>>> mario_number('    P    ')
9
>>> mario_number('   P    P P   P  P P    P     P ')
180
"""
```

Don't move on until you have been checked-off.

Topic Review:
Make sure you have a good understanding of these topics before getting checked off.

- Identifying the three parts of a recursive function.
- Evaluating the output of a recursive function.
- Using multiple base cases and recursive cases (tree recursion)

2 Higher Order Functions

Questions

2.1 What do lambda expressions do? Can we write all functions as lambda expressions? In what cases are lambda expressions useful?

Lambda expressions are a way to define functions in Python. Functions created with lambda expressions have no intrinsic name, unlike those created with def statements. Not every function can be written as a lambda expression. One reason for this is that the body of a lambda expression cannot contain statements, only expressions.
Lambda expression are useful in the case when we need to pass a simple function as an argument to another function.

2.2 Determine if each of the following will error:

```
>>> 1/0
```

Yes. This expression results in ZeroDivisionError.

```
>>> boom = lambda: 1/0
```

No. The body of the lambda function is not evaluated when it is defined.

```
>>> boom()
```

Yes. The name boom points to the lambda function which returns 1/0. When called it results in ZeroDivisionError.

2.3 Express the following lambda expression using a **def** statement, and the **def** statement using a lambda expression.

```
pow = lambda x, y: x**y
```

```
def pow(x, y):
    return x**y
```

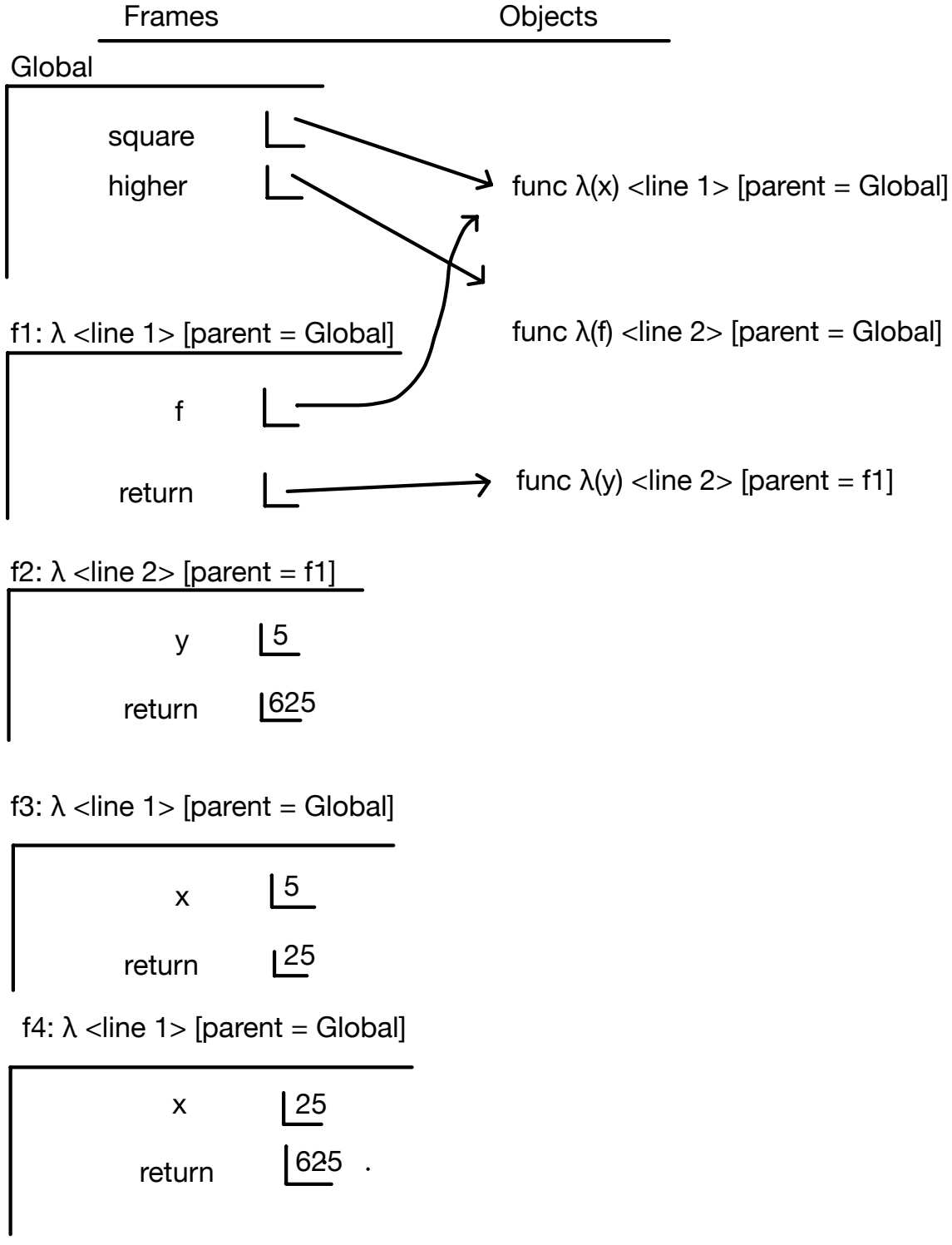
```
def foo(x):
    def f(y):
        def g(z):
            return x + y * z
        return g
    return f
```

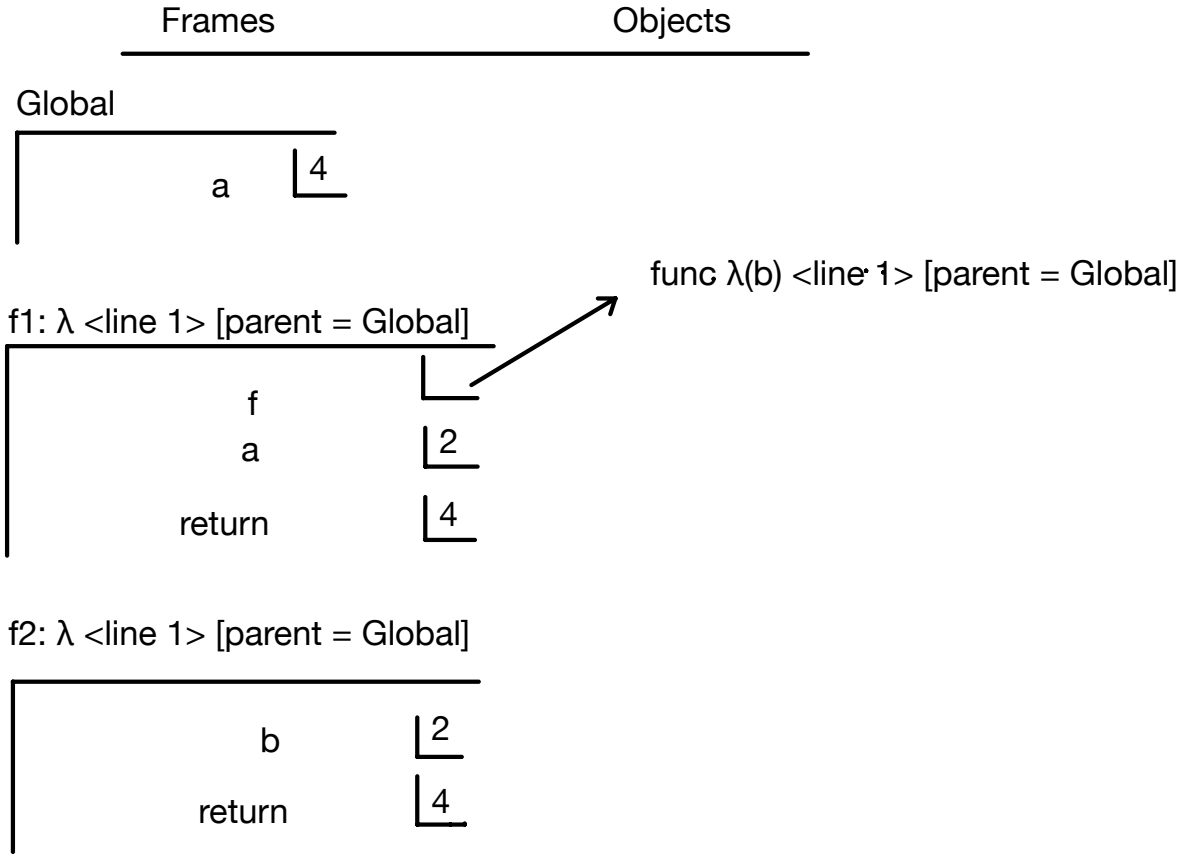
```
foo = lambda x: lambda y: lambda z: x + y * z
```

2.4 Draw Environment Diagrams for the following lines of code

```
square = lambda x: x * x
higher = lambda f: lambda y: f(f(y))
higher(square)(5)

a = (lambda f, a: f(a))(lambda b: b * b, 2)
```




Frames
Objects
Global
a
4
func λ(b) <line 1> [parent = Global]
f1: λ <line 1> [parent = Global]
f
a
2
return
4
f2: λ <line 1> [parent = Global]
b
2
return
4

2.5 Write **make skipper**, which takes in a number n and outputs a function. When this function takes in a number x, it prints out all the numbers between 0 and x, skipping every nth number (meaning skip any value that is a multiple of n).

```
def make_skipper(n):
    """
    >>> a = make_skipper(2)
    >>> a(5)
    1
    3
    5
    """
```

```
def print_nums(x):
  i = 1

  while i <= x:
    if i % n != 0:
      print(i)
    i = i + 1

return print_nums
```

2.6 Write a function that takes in a function `cond` and a number `n` and prints numbers from `1` to `n` where calling `cond` on that number returns `True`.

```
def keep_ints(cond, n):
    """Print out all integers 1..i..n where cond(i) is true

    >>> def is_even(x):
    ...     # Even numbers have remainder 0 when divided by 2.
    ...     return x % 2 == 0
    >>> keep_ints(is_even, 5)
    2
    4
    """
```

```
i = 1

while i <= n:
  if cond(i):
    print(i)
  i = i + 1
```

2.7 Write a function similar to `keep_ints` like before, but now it takes in a number `n` and returns a function that has one parameter `cond`. The returned function prints out numbers from 1 to `n` where calling `cond` on that number returns `True`.

```
def make_keeper(n):
    """Returns a function which takes one parameter cond and prints out
    all integers 1..i..n where calling cond(i) returns True.

    >>> def is_even(x):
    ...     # Even numbers have remainder 0 when divided by 2.
    ...     return x % 2 == 0
    >>> make_keeper(5)(is_even)
    2
    4
    """
    def print_nums(cond):
        i = 1

        while i <= n:
            if cond(i):
                print(i)
            i = i + 1

    return print_nums
```

Don't move on until you have been checked-off.

Topic Review:
Make sure you have a good understanding of these topics before getting checked off.

- The difference between **lambda** and **def**.
- Higher order lambda functions.
- Analyzing functions that return functions.

Optional

2.8 Write **make_alternator** which takes in two functions, f and g, and outputs a function. When this function takes in a number x, it prints out all the numbers between 1 and x, applying the function f to every odd-indexed number and g to every even-indexed number before printing.

```
def make_alternator(f, g):
    """
    >>> a = make_alternator(lambda x: x * x, lambda x: x + 4)
    >>> a(5)
    1
    6
    9
    8
    25
    >>> b = make_alternator(lambda x: x * 2, lambda x: x + 2)
    >>> b(4)
    2
    4
    6
    6
    """
|\begin{solution}
\begin{verbatim}
    def alternator(n):
        i = 1
        while i <= n:
            if i % 2 == 1:
                print(f(i))
            else:
                print(g(i))
            i += 1
    return alternator
\end{verbatim}
\end{solution}|
```