# EXAM PREPARATION SECTION 2

## RECURSION AND LAMBDA FUNCTIONS

February 7 to February 8, 2018

## 1 Recursion Summary

**Recursive functions** call themselves, either directly or indirectly, in the function body

- The motivation for this is to break down the problem into **smaller, easier to solve problems**.
- In environment diagrams, each call creates its own frames and **keeps track of its own argument values**.

Recursive functions have base cases, which evaluate without relying on calls to itself, and recursive cases. When writing recursive programs, remember the three parts in the structure of a recursive function:

- One or more **base cases**, usually handling the smallest input.
- One or more ways of **reducing the problem**, and then solving the smaller problem using recursion.
- One or more ways of **using the solution to each smaller problem** to solve our larger problem.

1. **Express Yourself (Fa14 MT1 Q3a))**

A k-bonacci sequence starts with K-1 zeros and then a one. Each subsequent element is the sum of the previous K elements. The 2-bonacci sequence is the standard Fibonacci sequence. The 3-bonacci and 4-bonacci sequences each start with the following ten elements:

```
             n: 0, 1, 2, 3, 4, 5, 6,  7,  8,  9, ...
kbonacci(n, 2): 0, 1, 1, 2, 3, 5, 8, 13, 21, 35, ...
kbonacci(n, 3): 0, 0, 1, 1, 2, 4, 7, 13, 24, 44, ...
kbonacci(n, 4): 0, 0, 0, 1, 1, 2, 4,  8, 15, 29, ...
```

Fill in the blanks of the implementation of `kbonacci` below, a function that takes non-negative integer `n` and positive integer `k` and returns element $n$ of a $k$-bonacci sequence.

```
def kbonacci(n, k):
    """Return element N of a K-bonacci sequence.

    >>> kbonacci(3, 4)
    1
    >>> kbonacci(9, 4)
    29
    >>> kbonacci(4, 2)
    3
    >>> kbonacci(8, 2)
    21
    """
    if n < k - 1:
        return 0
    elif n == k - 1:
        return 1
    else:
        total = 0
        i = n - k
        while i < n:
            total = total + kbonacci(i, k)
            i = i + 1
        return total
```

2. **Combine, Reverse, and Remove (Fa14 MT1 Q3b)**
Fill in the blanks of the following functions defined together in the same file. **Assume that all arguments to all of these functions are positive integers that do not contain any zero digits**. For example, 1001 contains zero digits (not allowed), but 1221 does not (allowed). You may assume that `reverse` is correct when implementing `remove`.

```
def combine(left, right):
    """Return all of LEFT's digits followed by all of RIGHT's
       digits."""
    factor = 1
    while factor <= right:
        factor = factor * 10
    return left * factor + right

def reverse(n):
    """Return the digits of N in reverse.
    >>> reverse(122543)
    345221
    """
    if n < 10:
        return n
    else:
        return combine(n%10, reverse(n//10))

def remove(n, digit):
    """Return all digits of N that are not DIGIT, for DIGIT
       less than 10.
    >>> remove(243132, 3)
    2412
    >>> remove(remove(243132, 1), 2)
    433
    """
    removed = 0
    while n != 0:
        n, removed = n//10, 10*removed + n%10
        if removed%10 == digit:
            removed = removed//10
    return reverse(removed)
```

3. **You Complete Me (Sp15 Midterm 1 Q3d)**
Implement the `memory` function, which takes a number `x` and a single-argument function `f`. It returns a function with a peculiar behavior that you must discover from the doctests. You may only use names and call expressions in your solution. You may not write numbers or use features of Python not yet covered in the course.

```
square = lambda x: x * x
double = lambda x: 2 * x
def memory(x, f):
    """Return a higher-order function that prints its
    memories.
    >>> f = memory(3, lambda x: x)
    >>> f = f(square)
    3
    >>> f = f(double)
    9
    >>> f = f(print)
    6
    >>> f = f(square)
    3
    None
    """
    def g(h):
        print(f(x))
        return memory(x, h)
    return g
```

4. **You Complete Me (Sp15 Midterm 1 Q3b)**
Add parentheses and single-digit integers in the blanks below so that the expression on the second line evaluates to 2015. **You may only add parentheses and single-digit integers.** You may leave some blanks empty.

```
lamb = lambda lamb: lambda: lamb + lamb

lamb(1000)()() + (lambda b, c: b()() * b()() -
   c()())(lamb(2)(), 1)()
```

## 2 Environment Diagrams

1. **Frog Goes Croak (Fa13 Midterm 1 Q2a)**
   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:

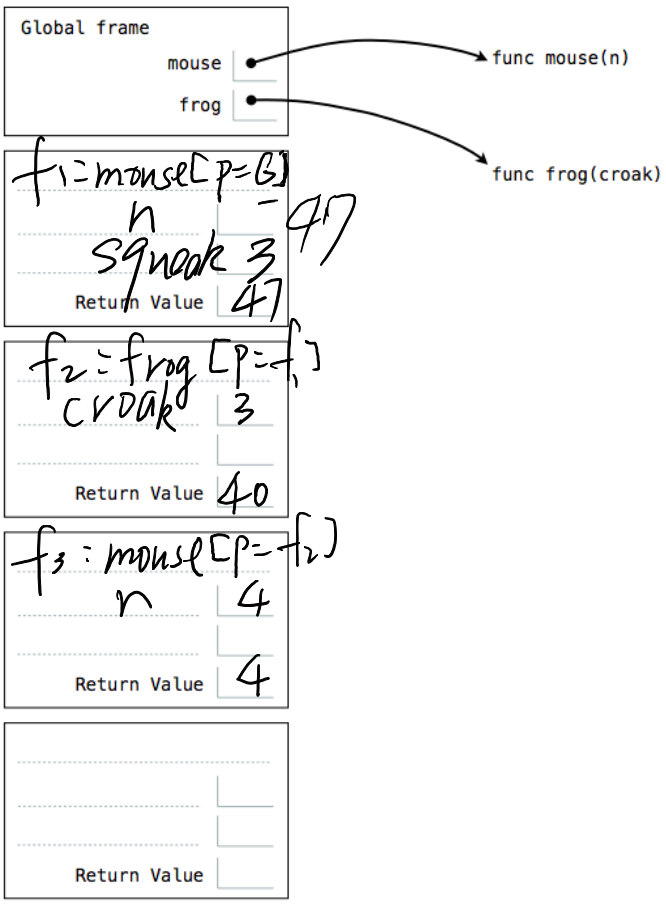
   - Add all missing names and parent annotations to all local frames.
   - Add all missing values created or referenced during execution.
   - Show the return value for each local frame.

   *You must list all bindings in the order they first appear in the frame.*

```
def mouse(n):
    if n >= 10:
        squeak = n // 100
        n = frog(squeak) + n % 10
    return n

def frog(croak):
    if croak == 0:
        return 1
    else:
        return 10 * mouse(croak+1)

mouse(357)
```




**(b) (2 pt) ***Question***:** After executing the code above, to what value will `mouse(21023508479)` evaluate?

31459

## 3 What Would Python Print

1. **"If (s)he can wield the Hammer..." (Su15 Midterm 1 Q1)**
   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".

   The first three rows have been provided as examples.

   *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:

```
from operator import add

avengers = 6

def vision(avengers):
    print(avengers)
    return avengers + 1

def hawkeye(thor, hulk):
    love = lambda black_widow: add(black_widow, hulk)
    return thor(love)

def hammer(worthy, stone):
    if worthy(stone) < stone:
        return stone
    elif worthy(stone) > stone:
        return -stone
    return 0

capt = lambda iron_man: iron_man(avengers)
```

| Expression | Interactive Output |
|---|---|
| `pow(2, 3)` | 8 |
| `print(4, 5) + 1` | 4 5<br>Error |
| `capt(vision)` | 6<br>7 |
| `print(print(1), vision(2))` | 1<br>2<br>None 3 |
| `hawkeye(hammer, 3)` | Error |
| `hawkeye(capt, 3)` | 9 |
| `hammer(lambda ultron: ultron, -1)` | 0 |
| `hammer(vision, avengers)` | 6<br>6<br>-6 |