Midterm 1 Review (Questions from past exams)

What Would Python Print

```
def jazz(hands):
      if hands < out:
            return hands * 5
      else:
            return jazz(hands // 2) + 1

def twist(shout, it, out=7):
      while shout:
            shout , out = it(shout), print(shout , out)
      return lambda out: print(shout , out)

hands, out = 2, 3
```

```
>>> print(None, print(None))

>>> jazz(5)

>>> (lambda out: jazz(8))(9)

>>> twist(2, lambda x: x-2)(4)

>>> twist(5, print)(out)

>>> twist(6, lambda hands: hands-out, 2)(-1)
```

Environment Diagrams

(Fall 2012)

```
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)

horse(mask)
```

(Fall 2015)

```
def inside(out):
    anger = lambda fear: fear(disgust)
    fear = lambda disgust: anger(out)
    disgust = 3
    fear(5)

fear, disgust = 2, 4
inside(lambda fear: fear + disgust)
```

(Spring 2014)

```
pop, ice, yam = 1, 2, 3
def x(sochi):
    pop = 4
    return lambda ice: sochi(ice)

mask = x(lambda yam: lambda: yam*pop*ice)

horse = mask(5)()
```

Lambdas

(Spring 2015)

Fill in the blanks so that the expression evaluates to 2015

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

lamb(1000)_____ + (lambda b, c: b____ * b____ - c____)(lamb(____), 1)____
```

(Spring 2014)

The boolean values True and False are actually instances of a kind of integer that Python treats as if they represented our logical notions of true and false. Suppose instead we use functions to represent boolean values. That is, we'll represent "true" with a value true (lower case) that is a function, and likewise "false" with a function false:

```
true = lambda a, b: a
false = lambda a, b: b
```

The function py_truth is supposed to convert these values into the normal Python True and False. Fill in the blanks so as to make the functions below conform to their comments. Do not use any conditional expressions or statements (if, and, or, not, while), and do not call py_truth.

```
def py_truth(p):
"""The Python boolean value represented by P, a functional truth value. >>>
py_truth(true)
True
>>> py_truth(false)
False
"""
return p(True, False)

def functional_and(p1, p2): """

>>> py_truth(functional_and(true, true)) True
>>> py_truth(functional_and(false, true)) False

>>> py_truth(functional_and(true, false)) False
>>> py_truth(functional_and(false, false)) False

"""
return _____
```

Iteration Recursion

(Fall 2014)
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(_____________________________ , _____________________________)


def remove(n, digit):
"""Return all digits of N that are not DIGIT, for DIGIT less than 10.
      >>>  remove (243132 , 3)
      2412
      >>>  remove (243132 , 2)
      4313
      >>> remove(remove(243132, 1), 2)
      433
      """
      removed = 0
      while n != 0:
            _________ , _________ = ___________________ , ___________________

            if ____________________________________________________________:
                  removed = ____________________________________________________
      return reverse(removed)
```

(Spring 2016)

Suppose we have a sequence of quantities that we want to multiply together, but can only multiply two at a time. We can express the various ways of doing so by counting the number of different ways to parenthesize the sequence. For example, here are the possibilities for products of 1, 2, 3, 4 and 5 elements:

Product	a	ab	abc	abcd	abcde		
Count	1	1	2	5	14		
Parenthesizations	a	ab	a(bc) (ab)c	a(b(cd)) a((bc)d) (ab)(cd) (a(bc))d ((ab)c)d	a(b(c(de))) a(b((cd)e)) a((bc)(de)) a((b(cd))e) a(((bc)d)e)	(ab)(c(de)) (ab)((cd)e) (a(bc))(de) ((ab)c)(de) (a(b(cd)))e	(a((bc)d))e ((ab)(cd))e ((a(bc))d)e (((ab)c)d)e

Assume, as in the table above, that we don't want to reorder elements.
Define a function count_groupings that takes a positive integer n and returns the number of ways of parenthesizing the product of n numbers. (You might not need to use all lines.)

```
def count_groupings(n):
    """For N >= 1, the number of distinct parenthesizations of a product of N
    items.
    >>>  count_groupings(1) 1
    >>>  count_groupings(2) 1
    >>>  count_groupings(3) 2
    >>>  count_groupings(4) 5
    >>>  count_groupings(5) 14
    """
    if n == 1:

        return ____________________________________________________________

    ____________________________________________________________________________

    i = ________________________________________________________________________

    while ____________________________________________________________:


    ____________________________________________________________________________

    i += 1

    return ____________________________________________________________________
```

Extras

Environment Diagrams

(Spring 2015)

```
def still(glad):
    def heart(broken):
        glad = lambda heart: lambda: heart - broken
        return glad(grin)
    return heart(glad - grin)()

broken, grin = 5, 3
still(broken - 1)
```

(Fall 2013)

Fill in the blanks of the implementation of differs_by_one_digit below, a function that takes two positive integers m and n and returns whether m and n differ in exactly one digit. If m and n have different numbers of digits, then differs_by_one_digit(m, n) always returns False.

```
def differs_by_one_digit(m, n):
      """Return True if and only if m and n have the same number of digits,
      and they differ by exactly one digit.
      You may assume that m and n are positive integers.
      >>> differs_by_one_digit(3467, 3427) # 3rd digit differs
      True
      >>> differs_by_one_digit(2013, 2011) # Last digit differs
      True
      >>> differs_by_one_digit(1013,  2013) # First digit differs
      True
      >>> differs_by_one_digit(5, 2) # Only digit differs
      True
      >>> differs_by_one_digit(2013, 2013) # No digit differs
      False
      >>> differs_by_one_digit(1013, 2011) # Both first and last differ
      False
      >>> differs_by_one_digit(3102, 2013) # All digits differ
      False
      >>> differs_by_one_digit(1, 21) # Different digit count
      False
      >>> differs_by_one_digit(21, 1)  # Different digit count
      False
      """
      diffs = 0

      while m > 0:

            if _____________________________:

                  return False

            m, t = m // 10, m % 10

            n, v = n // 10, n % 10

            if _____________________________:
```

```
            diffs = _____________________________________

    return _____________________________________
```

(Fall 2014)

The if_fn returns a two-argument function that can be used to select among alternatives, similar to an if statement. Fill in the return expression of factorial so that it is defined correctly for non-negative arguments. You may only use the names if_fn, condition, a, b, n, factorial, base, and recursive and parentheses in your expression (no numbers, operators, etc.).

```
def if_fn(condition):
    if condition:
        return lambda a, b: a
    else:
        return lambda a, b: b

def factorial(n):
    """Compute N! for non-negative N. N! = 1 * 2 * 3 *...* N.
    >>>  factorial (3)
    6
    >>>  factorial (5)
    120
    >>>  factorial (0)
    1
    """
    def base():
        return 1
    def recursive():
        return n * factorial(n-1)

    return _____________________________________________
```