Name:
Date: September 14th, 2017

(Tree) Recursion

Wait, what is tree recursion?

- Tree recursion is **literally** normal recursion, except you have more than one recursive call. All strategies etc in this worksheet is applicable for "both versions" of recursion.

3 parts of a recursive function:

1. **One or more recursive calls**
    a. Make a recursive call with the function on an argument that would get you closer to solving the problem. If you have one, it's "normal recursion". If you have more than one, that's "tree recursion".
2. ***Using* the recursive call(s)**
    a. Using the result of #1, solve the problem.
3. **One or more base cases**
    a. The function will eventually terminate eventually because of this.

(◕‿◕✿) PROBLEM 1: Identify the 3 parts of a recursive function within each function.

```
def has_seven(k):
    """Returns True if at least
    one of the digits of k is
    a 7, False otherwise.
    """

    if k % 10 == 7:
        return True
    elif k < 10:
        return False
    else:
        return has_seven(k // 10)
```

```
def summation(n, term):
    """Return the sum of the
    first n terms in the sequence
    defined by term.

    >>> summation(4, lambda x:
    x * x * x)
    # 1^3 + 2^3 + 3^3 + 4^3
    100
    """
    assert n >= 1
    if n == 1:
        return term(n)
    else:
        return term(n) + \
        summation(n - 1, term)
```

Note: the backslash (\) in the second return statement of summation() means:
"Python, read everything on the next line as part of this line."

How I write (normal and tree) recursive functions:

- There is no formula. :(Generally, I write the recursive call(s) first because they feel easier to me. You may find writing the base cases first easier.

- To determine the recursive calls and how to use them to get to the solution, I often try to find an **equivalent relationship** between these recursive calls and the original problem. For example:

    - multiply(a, b) == a * multiply(a, b-1)
    - gcd(a, b) == gcd(b, a % b) if a > b and a % b != 0, b otherwise.
    - fib(n) == fib(n-1) + fib(n-2)
    - ab_plus_c(a, b, c) == a + ab_plus_c(a, b - 1, c)
    - hailstone_sequence_length(n) == 1 + hailstone_sequence_length(n // 2) if n is even, 1 + hailstone_sequence_length(3 * n + 1) if odd.
    - sum_digits_of_n(n) == sum_digits_of_n(n//10) + n % 10

- **These are literally the formulas to write the entire recursive case (part 1 and 2 of recursive functions)! Seriously, for all of these we're straight up returning the right side of these equivalent relationships.** All that is left is figuring out the base cases.

- Once I figure out the recursive calls, I figure out what the recursive calls will eventually "boil down" to figure the base cases.
    - For example, for a function:

```
def example(n):
    if n == 0: return 1
    else: return example(n-1)
```

    - My recursive call is on **n-1**, which means I know **n** will eventually "boil down" to a smaller number.
    - Now, perhaps I decide that when n == 0, I definitely want to stop recursing. **So I write a base case where n == 0!**

(◕‿◕✿) **PROBLEM 2: Write an equivalent relationship for these problems. Don't worry about base cases yet.**

(2.0) Return the number of ten-pairs within positive integer n. **Assume you have function count_digit(n, digit) which returns how many times digit appears in n.**

```
>>> ten_pairs(7823952)
3
>>> ten_pairs(55055)
6
>>> ten_pairs(9641469)
6
```

Hint: ten_pairs(7823952) == ten_pairs(782395) + count_digit(782395, 8)
The value on the left **equals** the value on the right!

```
ten_pairs(n) == ____________________________________________________
```

(2.1) **(Fall 14 MT1)** Reverse all the digits in n.

```
>>> reverse(7823952)
2593287
```

```
reverse(n) == ____________________________________________________
```

(2.2) The TAs want to print handouts for their students. However, for some unfathomable reason, both the printers are broken; the first printer only prints multiples of **n1**, and the second printer only prints multiples of **n2.** Help the TAs figure out whether or not it is possible to print an exact number of handouts!

```
>>> has_sum(1, 3, 5)
False
>>> has_sum(5, 3, 5) # 0(3) + 1(5) = 5
True
>>> has_sum(11, 3, 5) # 2(3) + 1(5) = 11
True

has_sum(total, n1, n2) ==

_____________________________________________________________________
```

(2.3) I want to go up a flight of stairs that has **n** steps. I can either take **1** or **2** steps each time. How many different **ways** can I go up this flight of stairs? Assume n is positive.

```
>>> count_stair_ways(1)  #1
1
>>> count_stair_ways(2)  #11, 2
2
>>> count_stair_ways(4)  # 1111, 211, 112, 121, 22
5

count_stair_ways(n)  == ____________________________________________
```

(◕‿◕✿) PROBLEM 3: Write the base cases for the above problems!

(2.0) `ten_pairs(n)`

```
     if ___________:
          return _________
```

(2.1) `reverse(n)`

```
     if ___________:
          return _________
```

(2.2) `has_sum(total, n1, n2)`

(2.3) `count_stair_ways(n)`

You've just written the entire recursive functions! (Just separated into parts).

(◕‿◕✿) PROBLEM 4: Final Practice

(4.0) (Spring 2015 MT1) Implement the combine function, which takes a non-negative integer n, a two-argument function f, and a number result. It applies f to the first digit of n and the result of combining the rest of the digits of n by repeatedly applying f (see the doctests). If n has no digits (because it is zero), combine returns result.

```
from operator import add, mul

def combine(n, f, result):
    """Combine the digits in non-negative integer n using f.

    >>> combine(3, mul, 2) # mul(3, 2)
    6
    >>> combine(43, mul, 2) # mul(4, mul(3, 2))
    24
    >>> combine(6502, add, 3) # add(6, add(5, add(0, add(2, 3))))
    16
    >>> combine(239, pow, 0) # pow(2, pow(3, pow(9, 0)))
    8
    """
    if n == 0:

        return result

    else:

        return combine(__________, __________, ____________________________)
```

(4.1) (Albert Wu's Exam-level Practice Problems: http://albertwu.org/cs61a/review/recursion/exam.html#q1)

In game theory, a subtraction game is a simple game with two players, player 0 and player 1. At the beginning, there is a pile of n cookies. The players alternate turns; each turn, a player can take anywhere from 1 to 3 cookies. The player who takes the last cookie wins. Fill in the function can_win, which returns True if it is possible to win starting at the given number of cookies. It uses the following ideas:

- if the number of cookies is negative, it is impossible to win.
- otherwise, the current player can choose to take either 1, 2, or 3 cookies.
- evaluate each action: if that action forces the opponent to lose, then return True (since we can win)
- if none of the actions can force a win, then we can't guarantee a win.

```
def can_win(number):
    """Returns True if the current player is guaranteed a win
    starting from the given state. It is impossible to win a game
    from an invalid game state.

    >>> can_win (-1) # invalid game state
    False
    >>> can_win (3) # take all three !
    True
    >>> can_win (4)
    False
    """
```

```
def can_win(number):
    if ____________________________________________ :
        ____________________________________________
    action = ____________________________________________
    while ____________________________________________ :
        ____________________________________________
        if ____________________________________________ :
            ____________________________________________
        action ____________________________________________
    return ____________________________________________
```

Parting thoughts:

- I believe in you! This course goes by fast (remember, twice as fast as even *that school across the bay)*, and the material is very hard. I respect you for being here.
- Past exams were typically about 30% WWPD (so, environment diagrams), 35% environment diagrams, and 35% code writing.
- Please take care of yourself. Take a nap and eat a good meal before the exam.
- See you on the other side! **Solutions posted at jsluong.com around 12 pm today.**