Class 13 - Notes

Upcoming Schedule

Project 3 is due at the beginning of class on **Friday, 26 February**, but you may not submit Project 3 until completing the Orange Belt level. If you are making clear progress towards completing the Orange Belt (see below), you may turn in Project 3 on Monday, 29 February with no penalty (still eligible for automatic Green Belt promotion with an excellent Project 3).

Orange Belt Promotion

If you received a Gold Star (or better!) on Project 2, congratulations - you are now an Orange Belt!

If you received a Green Star on Project 2, the standard way to be promoted to the Orange Belt level is to do these four things:

1. Complete Udacity cs101 Lesson 6: *How to Have Infinite Power* to make sure you understand recursive definition.
2. Answer Lesson 6: *Problem Set*: Question 4 ("Deep Count").
3. Answer Lesson 6: *Problem Set 6 Starred*: Question 1 ("Family Trees")
4. Define a `list_reverse` function that takes as input a list and returns a new list with the elements of the input list in reverse order. For example, `list_reverse([1, 2, 3, 4])` should return `[4, 3, 2, 1]`. You should define *two* versions of your `list_reverse` function: one that uses a recursive call (and may not contain any `for` or `while` loops), and one that uses a loop (and may not contain a recursive call).

There is nothing to submit for 1-3. For questions 2 and 3, you should definitely try to solve these yourself and not look at the answer until you have made a serious attempt to solve it yourself. But, you may look at the answer if you are stuck. After looking at the answer, you should still write your own solution.

For question 4, **you must work alone and solve this problem without help from anyone or using any external resources** (other than the course staff).

To complete the Orange Belt, send your solution to #4 by email.

Hammering Hamming

```
def hamming_distance_rec(s1, s2):
    assert len(s1) == len(s2)
    if not s1: return 0
    else:
        return (not s1[0] == s2[0]) + hamming_distance_rec(s1[1:], s2[1:])

def hamming_distance_loop(s1, s2):
    assert len(s1) == len(s2)
    count = 0
    for i in range(len(s1)):
        count = count + (not s1[i] == s2[i])
    return count

def hamming_distance_lc(s1, s2):
    assert len(s1) == len(s2)
    return sum([not e1 == e2 for (e1, e2) in zip(s1, s2)])
```

Time Trials

This is a simple (but not super accurate) way to estimate running times.

```
import sys, time, random, string

def random_bases(n):
    return ''.join(random.choice(['A','C','G','T']) for i in range(n))

def time_hamming(n):
    s1 = random_bases(n)
    s2 = random_bases(n)
    for impl in [hamming_distance_rec,
                 hamming_distance_loop,
                 hamming_distance_lc]:
        start_time = time.clock()
        for count in range(1000):
            impl(s1, s2)
        stop_time = time.clock()
        print("Time for " + impl.__name__ + ": " +
              str(stop_time / start_time))
```

Edit Distance

```
def edit_distance(a, b):
    """
    Returns the Levenshtein distance between a and b.
    """
    if not (a and b):
        return len(a) + len(b)
    else:
        if a[0] == b[0]:
            return edit_distance(a[1:], b[1:])
        else:
            return min(1 + edit_distance(a, b[1:]),     # insert
                       1 + edit_distance(a[1:], b),     # delete
                       1 + edit_distance(a[1:], b[1:])) # mutate
```

Nested Lists

```
def list_copy(lst):
    """Returns a shallow copy of the input lst."""
    if not lst:
        return []
    else:
        return [lst[0]] + list_copy(lst[1:])

def is_list(obj):
    """Returns True iff obj is a list."""
    return isinstance(obj, list)

def list_deep_copy(lst):
    """Returns a deep copy of the input list."""
    if not lst:
        return []
    elif is_list(lst[0]):
        return [list_deep_copy(lst[0])] + list_deep_copy(lst[1:])
    else:
        return [lst[0]] + list_deep_copy(lst[1:])
```

Demonstrate the difference between `list_copy` and `list_deep_copy` with some Python code that shows how they behave differently.

