CS417 Lab 17

Getting Started

Create a folder for your work. Then, go to `mycourses.unh.edu`, find CS417, find Modules, and find the lab. Then download these files:

- `recursion.py`.
- `test_output.txt`. Your program's output should match this.

Recursion

In this lab, you will practice working with recursive functions. A well-written recursive function will be given a problem to solve, and will handle it in two ways:

1. The *BASE* case : if the problem is small, solve it directly.

2. The *RECURSIVE* case : if the problem is larger, call yourself, with a smaller problem. Then use the result of that call, to solve the over-all problem.

For example, here is a recursive function that computes the sum of the values in a list. It is passed a list.

```
        def sum_list( alist ):
            # Base case: if there are no values to add up,
            # the sum is zero
            if len(alist) == 0:
                return 0

            # Recursive case: get the tail's sum,
            # and add to it the head value
            else:
                sum_tail = sum( alist[1:] )
                total = alist[0] + sum_tail
                return total
```

Notice that there are no loops! All the looping happens through recursion.

.

Exercises

In the exercises that follow, you will be slicing a list in various ways, getting a small part, and a big part.

- Slicing method #1: head and tail

  a. `a_list[0]` : this is the head of the list (its first value)
  b. `a_list[1:]` : this is the tail of the list (all the other values)

- Slicing method #2: ends and core

  a. `a_list[0]` and `a_list[-1]` are the front and back of the list
  b. `a_list[1:-1]` is the core of the list

- Slicing method #3: front and "caboose"

  a. `a_list[:-1]` : this is the front of the list (minus the last value)
  b. `a_list[-1]` : this is the caboose

---

1. Implement `count_odds(alist)`, which counts how many odd numbers are odd in alist.

   - BASE case: if the list has length 0, there are no odd numbers. Return 0.
   - RECURSIVE case: otherwise, call `count_odds`, and pass it the tail of the list `alist[1:]`. Save the result in `tail_count`.

     Finally, check the head value. If it is odd, return `1 + tail_count` .

     Otherwise, it isn't odd, so return `tail_count`.

---

2. Implement `is_palindrome( s )`. The parameter `s` is a string.

   A palindrome is a string that is the same when reversed. Here are some examples:

   ```
       madam
       eve
       racecar
   ```

   The length of the string `len(s)` is the size of the problem. Here are the two cases:

   - base case: if the length is 0 or 1, the string is a palindrome, by default. Return `True`.
   - recursive case #1: otherwise, check if the first and last letters in the string are different. If so, the string is not a palindrome. Return `False`.
   - recursive case #2: the first and last letters match. Let's test the center of the list. Call `is_palindrome`, and pass it the core of the string. Save the returned value into a variable. Return that variable.

3. Implement `max_value( alist )`, which finds the biggest value in `alist`. Assume the list has length 1 or more.

    - base case: if `alist` has length 1, return the head (and only) value.
    - recursive case: otherwise, call `max_value`, and pass it the tail of `alist`. Save the returned value into `max_tail`.

      Compare the head value to `max_tail`. Whichever is larger, return that.

---

4. Implement `last_index(alist, value)`. This returns the last index in `alist` where `value` occurs.

    For example, `last_index( [5,2,4,5,6,2], 5 )` should return 3.

    - base case: if `alist` has zero length, return -1 (indicating that `value` does not occur).
    - base case: otherwise, if the **last entry** of `alist` is equal to `value`, then return `len(alist)-1`.
    - Recursive case: call `last_index()`, and pass it a slice of the list, without the last entry (front minus caboose). Save the returned value into a variable. Return that variable.

---

5. 10% Bonus: Implement `is_sorted(alist)`. It checks whether the values are in increasing order, and returns `True` or `False` accordingly.

    - base case: if the list is empty, it is sorted, by default. Return `True`.
    - recursive case: otherwise, check the first two values. If they are not in ascending order, return `False`.

      Otherwise, the first two values are OK. Check the tail: call `is_sorted`, and pass it the tail of the list. Return whatever is returned by that call.

Turning in your work

To submit your work, go to `mycourses.unh.edu`, find cs417, and the lab, and upload `recursion.py`. You can submit again until midnight, with no lateness penalty.