Work with your neighbor. (This will be graded for participation only.)

1. Write a recursive function `replace(s, a, b)` that takes as arguments the strings `s`, `a`, and `b` and returns a new string where all the occurrences of string `a` in string `s` are replaced with string `b`.

   For your solution, assume that string `a` can be of *any size*. Use `s.startswith(a)`, which returns `True` if string `s` starts with `a` and `False` otherwise.

2. Write a recursive function `get_even_positions(alist)` that returns a list consisting of the elements at the even-numbered positions of the Python list `alist` (the first element is at position 0). For example, the call

   ```
   get_even_positions([2,6,5,33,8])
   ```

   should return the list `[2,5,8]`. `<Problem continued on next page.>`
   Hints: Sometimes it can be challenging to figure out what the base case(s) and recursive case(s) should be. Before you start writing the code, consider what should be returned for each of these calls:

   ```
   get_even_positions([])              returns:
   get_even_positions([2])             returns:
   get_even_positions([2,6])           returns:
   get_even_positions([2,6,5])         returns:
   get_even_positions([2,6,5,33])      returns:
   ```

Now, before you write the base case(s) and the recursive case(s), answer these questions:

a) When the recursive call returns, what operation is performed and what types are the operands expected to be?

b) What is the type that is returned in the base case(s)?

c) Looking at the expected results, do you think you might need more than one base case?

Now write your solution.

3. Write a recursive function `max_l(alist)` that takes a list of integers `alist` and returns the largest value in `alist`. You may assume that `alist` has at least one element. (This models the behavior of the built-in `max()` function.)

<continued on next page>

4. Write a recursive function `maxmin(alist)` that takes a list of integers `alist` and returns a tuple `(max,min)` where `max` is the largest value in `alist` and `min` is the smallest value in `alist`. You may assume that `alist` has at least one element.

5. Write a function `flatten(alist)` that takes a mult-dimensional list consisting of both integers and lists and returns a list consisting of only the integer values in all of the lists. For example, the call

   ```
   flatten([[2, 4, 6], [10, [100, 200], 20], [[66], [[77, 88]], 99]]
   ```

   Would return the list below:

   ```
   [2, 4, 6, 10, 100, 200, 20, 66, 77, 88, 99]
   ```

   Note: to determine if an element is a list, use this comparison: `type(alist[0]) == list)`

6. We have written the recursive function version of `sumlist(L)` that returns the sum of the elements in `L`. Re-write the recursive part in two different ways:
   a. Recurse on the first through second to the last elements and add the last element

   b. Recurse on each half and add them together