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.

   ANS:

```
def replace(s, a, b):
    if s == "":
        return ""
    elif s.startswith(a):
        return b + replace(s[len(a):], a, b)
    else:
        return s[0] + replace(s[1:], a, b)
```

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]`.

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

   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?

ANS:

list concatenation

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

ANS:

list

c) Do you think you might need more than one base case?

ANS:

Yes, or one base case that covers alist lengths <= 2.

Now write your solution.

ANS:

```
def get_even_positions(alist):
    if len(alist) == 0:
        return []
    elif len(alist) <= 2:
        return [alist[0]]
    else:
        return [alist[0]] + get_even_positions(alist[2:])
```

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.)

ANS: Note the solutions assume that the argument list will have at least one element. This models the behavior of the built-in `max()` function.

```
def max_l(alist):
    if len(alist) == 1:
        return alist[0]
    else:
        value = max_l(alist[1:])
        if alist[0] >= value:
            return alist[0]
        else:

            return value
```

We could also use the built-in `max` function—not to compute the maximum of the entire list, but to find the maximum of just two numbers. That gives the following solution:

```
def max_l_v2(alist):
    if len(alist) == 1:
        return alist[0]
    else:
        return max(alist[0], max_l_v2(alist[1:]))
```

Here is an alternative solution. In the recursive cases, a new list is created for the recursive call that has eliminated the smaller of `alist[0]` and `alist[1]`.

```
def max_alist(alist):
    if len(alist) == 1:
        return alist[0]
    else:
        if alist[0] > alist[1]:
            return max_alist([alist[0]] + alist[2:])
        else:
            return max_alist(alist[1:])
```

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.

ANS:

```
def maxmin(alist):
    if len(alist) == 1:
        return (alist[0], alist[0])           #return a tuple
    else:
        max_rest, min_rest = maxmin(alist[1:])#unpack the tuple
        return (max(alist[0], max_rest),
                       min(alist[0], min_rest))
```

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)`

ANS:

```
def flatten(alist):
    if alist == []:
        return []
    elif type(alist[0]) == list:
        return flatten(alist[0]) + flatten(alist[1:])
    else:

        return [alist[0]] + flatten(alist[1:])
```

NOTE: Problem 6 was moved to ICA-21.