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

1. Write a recursive function `join_all(alist)` that takes a list `alist` and returns a string consisting of every element of `alist` concatenated together.

**ANS:**

What is the base case? Line at comment A

What is the recursive case? Line at comment B

```
def join_all(alist):
    if len(alist) == 0:       # A
        return ""
    else:
        return str(alist[0]) + join_all(alist[1:]) # B
```

2. Write a recursive function `join(alist,sep)` that takes a list `alist` and returns a string consisting of every element of `alist` concatenated together separated by the string `sep`.

**ANS:**

What is the base case? Lines at comment A and B

What is the recursive case? Line at comment C

```
def join(alist, sep):
    if len(alist) == 0:    # A
        return ""
    elif len(alist) == 1:  # B
        return alist[0]
    else:
         return alist[0] + sep + join(alist[1:], sep)  # C
```

3. Write a recursive function `sum_cols(grid, n)` that takes a list of lists of integers `grid` and integer `n` and returns the sum of column `n` in `grid`. For example, the call

```
sum_cols([[1,2,3,4], [10,20,30,40], [100,200,300,400]], 2)
```

should return `333`.

**ANS:**

```
def sum_cols(grid, n):
    print(grid)

    if len(grid) == 0:
        return 0
    else:
        return grid[0][n] + sum_cols(grid[1:], n)
```

**NOTE: Problems 4 and 5 were moved to ICA-20, but I am providing the solutions here.**

4. 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`.

   a) For your solution, assume that string a is only one character long.

**ANS:**

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

   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)
```

5. 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]`.

**Help:** 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

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:])
```