1. Rewrite the following expressions in a simpler or more elegant way (right of my code):

(a) # assume u and v are strings
    s = ''
    for c in u:
        s = s + c
    for c in v:
        s += c

(b) def nstars(n):
    '''Returns a string of n stars.'''
    assert n >= 0
    s = ''
    for i in range(n):
        s += '*'
    return s

(c) # assume l is a list and x is an object
    l += [ x ]

(d) # assume k and l are lists:
    for x in k:
        l += [ x ]

(e) # assume o is some object, currently unknown (hint: format)
    print("str="+str(o)+" repr="+repr(o))

2. You have six TA's in this course: Geoffrey, Javier, Julia, Louisa, Mi, and Tuan.
   Get as many of them to sign this space as you can (slight extra credit if four or more sign):
3. Let's explore the difference between `str(o)` and `repr(o)`.
   Recall that `str(o)` constructs a string from `o` while `repr(o)` constructs a representation of `o`.

   (a) Give an example of an object, `o`, where `str(o)` and `repr(o)` differ.

   (b) Interactive python is governed by a read-eval-print loop: an expression is read, it is evaluated, and the result (if any) is printed. Is the result, `o`, printed using `str(o)` or `repr(o)`? How do you know this?

   (c) The `eval(s)` function evaluates the python expression `s` and returns the result.
   Here's a hypothesis worth testing: The representation of `o`, `repr(o)`, is that it returns a string `s`, when evaluated, has the same value as `o`. That is, for any `s` in interactive python we have:
   ```python
   >>> s == eval(repr(s))
   True
   ```
   True or not? Describe different `s` you used in this experiment. (Hint: did you think of `None`?)

   (d) Suppose it were True. Why would it be useful to be able to convert an object `o` to its representation `s` and then, later, use `eval` to convert `s` back to `o`? (Hint: for example, you might wait several years between calling `repr` and `eval.`)