Thinking about strings.

1. Questions?

2. Pre- and postconditions.
   (a) Postcondition: what this procedure can be expected to do.
   (b) Precondition: what must true for the postcondition to be achieved.

3. Module testing:
   (a) Add example usage to doc-comments.
   (b) Add the following:
       ```python
       from doctest import testmod
testmod()
       ```
       This will report errors in assumptions made in doc-comments.

4. String objects. You can learn more about the `str` class with `pydoc3 str`
   (a) Strings look similar to (but are not) lists of single characters.
      i. They have order, they can be indexed (including slices), and you can iterate across them.
      ii. They have length. You can determine their length with the `len` function.
      iii. Slices return (possibly smaller or empty) objects of the same type. (Unlike other languages, there is no “character” type; instead, python has single character strings.)
      iv. The `+` operator performs concatenation.
   (b) Unlike lists, strings are immutable.
      i. You can not change the characters in a string. Instead, you build a new string that reflects the change.
   (c) The `str(o)` function is a constructor for the string class. It takes an object (of any type) and returns the string version of it.
   (d) The `repr(o)` function returns a string that is a representation of `o`.
      i. When you `s+o` a string (`s`) with an object (`o`), the object is converted to a string with `str(o)`. This conversion is informally called a cast.
   (e) Strings are objects. All objects `o` have methods `m` that act directly on the object by calling `o.m(arguments)`. Here are string methods you should familiarize yourself with (remember: these return new values; they do not mutate `s`):
      i. `s.lower()`, `s.upper()`. These functions convert case.
ii. `s.lstrip()`, `s.rstrip()`, or `s.strip()`. Remove whitespace from left, right, or both ends.

iii. `s.split()`. Split a string into a list of words.

iv. `s.join(l)`. Join a list of strings using `s`.

v. `s.find(t)` (or `s.rfind(t)`) and `s.count(t)`.
   Return the first (or last) position of string `t` in `s`. Returns -1 if not found.

   The `count(t)` method returns the number of (non-overlapping) occurrences of `t` in `s`.

vi. `s.replace(old,new)`. Replace all instances of `old` with `new` in string `s`.

vii. `s.isspace()` (or `islower`, `isupper`, `isalpha`, `isdigit`, `isalnum`).
   Returns True if `s` is not empty and `s` is composed of white space (or lowercase, uppercase, or alphabetic letters, or digits, or either letters or digits).

There are others! See `pydoc3 str`.

(f) The `s.format(*args)` method. A quick way to build strings with particular form (see `pydoc3 FORMATTING` for more details):

i. First, `*args` means: zero or more arguments. Thus, format takes zero or more arguments as in:
   ```
   >>> print("Hello, you {} world{}").format("silly","!"))
   Hello, you silly world!
   >>> print("Hello, {}.").format("you silly world!"))
   Hello, you silly world!
   ```
   If you have a list, `l`, then `*l` means put the elements of `l` in as arguments, here:
   ```
   >>> l = ['you', 'silly', 'world!']
   >>> print(*l) # note resulting spaces:
   you silly world!
   ```
   ```
   >>> print('Hello, {} {} {}'.format(*l))
   Hello, you silly world!
   ```
   Wow!

ii. For every pair of braces `{}`, format consumes one argument. The argument is converted to a string (with `str`) and concatenated with the remaining parts of the format string. See above.

iii. If, in the braces, you include a number/position, that indicates which argument you wish to use:
   ```
   >>> print("Hello, {1} {2} {0}".format('you','silly','world!'))
   Hello, silly world! you
   ```
   Positions must be used for all arguments, or none.

iv. You may append a `!s` or `!r` to indicate you want to use `str` or `repr` to convert the argument:
   ```
   >>> print('Hello, {1} {!r} {!s}'.format('you','silly','world!'))
   Hello, you 'silly' world!
   ```
   (This may be important if you consider using `eval(s)`.)

v. Many types can be controlled with a `width` and/or a `precision`. See `pydoc3 FORMATTING` for more details on this.

(g) The `eval(s)` function invokes the Python interpreter on `s` and returns the result. Wowza.