8.13: LIST ARGUMENTS

Contributed by Chuck Severance
Clinical Associate Professor (School of Information) at University of Michigan

When you pass a list to a function, the function gets a reference to the list. If the function modifies a list parameter, the caller sees the change. For example, `delete_head` removes the first element from a list:

```
def delete_head(t):
    del t[0]
```

Here's how it is used:

```
>>> letters = ['a', 'b', 'c']
>>> delete_head(letters)
>>> print(letters)
['b', 'c']
```

The parameter `t` and the variable `letters` are aliases for the same object.

It is important to distinguish between operations that modify lists and operations that create new lists. For example, the `append` method modifies a list, but the `+` operator creates a new list:

```
>>> t1 = [1, 2]
>>> t2 = t1.append(3)
>>> print(t1)
[1, 2, 3]
>>> print(t2)
None

>>> t3 = t1 + [3]
>>> print(t3)
[1, 2, 3]
>>> t2 is t3
False
```

This difference is important when you write functions that are supposed to modify lists. For example, this function *does not* delete the head of a list:

```
def bad_delete_head(t):
    t = t[1:]              # WRONG!
```

The slice operator creates a new list and the assignment makes `t` refer to it, but none of that has any effect on the list that was passed as an argument.

An alternative is to write a function that creates and returns a new list. For example, `tail` returns all but the first element of a list:

```
def tail(t):
    return t[1:]
```

This function leaves the original list unmodified. Here's how it is used:

```
>>> letters = ['a', 'b', 'c']
>>> rest = tail(letters)
>>> print(rest)
['b', 'c']
```

Exercise 1:

Write a function called `chop` that takes a list and modifies it, removing the first and last elements, and returns `None` .

Then write a function called `middle` that takes a list and returns a new list that contains all but the first and last elements.