1 Lists

A **list** is a sequence, like a string. Whereas a string is a sequence of characters, a list can contain items of any type. The things in a list are called **elements** or **items**. One creates a list by enclosing a comma-separated sequence of values in brackets. The values can be of any type: `int`, `float`, `bool`, `str`, even other lists!

```
>>> L = [43, 2, 8]                          # list of ints
>>> L2 = ['a list', 'of', 'strs']           # list of strs
>>> L3 = [5, 'hello', 4.3, False ]          # list of mixed types
>>> L4 = ['hi', 5, ['another', 'list!'], 12] # list w/ nested list
```

Many of the operators we applied to strings can be applied to lists:

```
>>> L + L2                                  # concatenation
[43, 2, 8, 'a list', 'of', 'strs']
>>> ['a', 'b'] * 3                          # copying
['a', 'b', 'a', 'b', 'a', 'b']             # the in operator
>>> 'of' in L2
True
>>> 'a' in L2
False
>>> L[0]                                    # indexing
43
>>> L[-2]                                   # negative indexing
2
```

The functions `min`, `max`, `len`, and `sum` can be applied to lists. The function `len` returns the number of items in a list. Keep in mind that a nested list counts as a single item.

```
>>> len(L4)                    # nested list counts as one item
4
```

Function `sum` expects a list of numbers (runtime errors occur for lists with non-numerical values).

```
>>> sum(L)
53
>>> sum(L2)
Traceback (most recent call last):
  File '<stdin>', line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

2 Lists in loops

Just like with a string, you can use a **for** loop to iterate over the items in a list. This loop

```
for item in L2:
    print item
```

will print this:

```
a list
of
strs
```

In fact, you have already been looping over lists since the `range` function returns a list of numbers. When we need to know the index of an item, we will use a `range` loop to iterate over the indices of list items. For example,

```
for i in range(len(L2)):
      print i, L2[i]
```

prints this

```
0 a list
1 of
2 strs
```

3   Lists are mutable

Unlike strings, lists are *mutable*.

```
>>> s = 'strings cannot be mutated'
>>> s[0] = 'S'
Traceback (most recent call last):
  File '<stdin>', line 1, in <module>
TypeError: 'str' object does not support item assignment
>>> L2[0] = "a mutable list"
>>> L2
['a mutable list', 'of', 'strs']
```

4   Exercises

Solutions are presented in class and also included in the online version of this handout.

1. Write a function `find_min` that takes a list of numbers and returns the smallest number. You cannot use `min`.

   > **Solution:**
   >
   > ```
   > def find_min(L):
   >     '''(list of int) -> list of int
   >     Given a list of numbers L, return the indexes of the
   >     smallest number.
   >     >>> find_min([12, 9, 10])
   >     9
   > ```

```
>>> find_min([12, 9, 10, 9])
9
>>> find_min([-12, 9, 10, -12, -12])
-12
'''
min_value = L[0]  # or some large number
for i in range(len(L)):
    if L[i] < min_value:
        min_value = L[i]
return min_value
```

2. Write a function `index_of_min` that takes a list of numbers and returns the *index* of the smallest number. (Challenge edition: don't use the `min` function.)

Solution:

```
def index_of_min(L):
    '''(list of int) -> int
    Given a list of numbers L, return the index of the
    smallest number.
    >>> index_of_min([12, 9, 10])
    1
    >>> index_of_min([12, 9, 10, 7])
    3
    >>> index_of_min([-12, 9, 10])
    0
    '''
    min_idx = 0
    for i in range(len(L)):
        if L[i] < L[min_idx]:
            min_idx = i
    return min_idx
```

3. Write a function `indexes_of_min` that takes a list of numbers and returns a list containing the *indexes* of the smallest number. Example: on input `[30,23,45,23]` it should return `[1,3]`.

Solution:

```
def indexes_of_min(L):
    '''(list of int) -> list of int
    Given a list of numbers L, return the indexes of the
    smallest number.
```

```
>>> indexes_of_min([12, 9, 10])
[1]
>>> indexes_of_min([12, 9, 10, 9])
[1, 3]
>>> indexes_of_min([-12, 9, 10, -12, -12])
[0, 3, 4]
'''
min_indexes = []
min_value = L[0]   # or some large number
for i in range(len(L)):
    if L[i] < min_value:
        min_indexes = [i]
        min_value = L[i]
    elif L[i] == min_value:
        min_indexes += [i]
return min_indexes
```

4. Write a function `make_list` that takes a string and returns a list containing the characters in the string in the same order.

Solution:

```
def make_list(s):
    '''(str) -> list of str
    Given string s, return a list containing the
    characters in s.
    >>> make_list("abcxyz")
    ['a', 'b', 'c', 'x', 'y', 'z']
    '''
    L = []
    for ch in s:
        L = L + [ch]
    return L
```

5. Write a function `avg` that takes a list of numbers and returns their average. Write two versions, one that uses `sum` and one that does not. If the list is empty return 0.

Solution:

```
def avg(L):
    '''(list of int) -> float
    Given list L of numbers, return the average.
```

```
>>> avg([1,2,3])
2.0
'''
if len(L) == 0:
    return 0

# version 1:
# return float(sum(L)) / len(L)

# version 2:
tot = 0.0
for item in L:
    tot += item
return tot / len(L)
```