ABSTRACT LIST FUNCTION AND ACCUMULATIVE RECURSION
REMINDER

• Assignment 05 due Wednesday, Oct. 23rd at 10:00AM

• Midterm is on Nov. 4th at 7 PM
  – Midterm reference sheet will be posted on Piazza & course webpage.

• Please write design recipe first

• Can’t guarantee to reply late night emails.
ABSTRACT LIST FUNCTIONS: MAP, FILTER

• map
  – applies function to each element in list
  list(map(fun_name, L))

• filter
  – matches the elements in list for which function returns True.
  list(filter(fun_name, L))

Note: fun_name must have only one parameter/argument.

Note: map and filter both return an iterator, and we need to convert that to a list
map and filter can also be applied to strings.
Lambda

\[
\text{lambda } x, y, \ldots, z: \text{ body here}
\]

Parameters of lambda (no brackets)

Example:

```python
def non_zero(numlist):
    return list(filter(lambda x: x != 0, numlist))

def triple(numlist):
    return list(map(lambda x: x * 3, numlist))
```
We can apply `map` and `filter` to strings with lambda

For example:

sentence is a string consisting of various characters

```python
def just_letter(sentence):
    loc = list(filter(lambda c: c.isalpha(), sentence))
    return ''.join(loc)
```
RECURSION

Types:
• Structural recursion

We’ve been using this so far.

• Accumulative recursion

New one we are going to learn!!!
DIFFERENCE BETWEEN THE TWO

• **Structural Recursion:**
  – Break problems into smaller problems using the recursive definition of our data.
  – recursive subproblem is always **one step closer** to a base case
  – Uses a recursive template

• **Accumulative Recursion:**
  – Recursion with an accumulator(s)

But these two are not necessarily independent!
def acc_fn(remaining, acc):
    if (base_case of remaining):
        return ... acc ...
    else:
        ...
        return acc_fn(updated_remaining, updated_acc)

def fn(lst):
    return acc_fn(initial_remaining, initial_acc)

• Accumulators “keep track” of something so that you can quickly return the expected result
• Sometimes, you may need more than one accumulator.
QUESTION 1

Develop an accumulatively recursive function `list_to_num` that consumes a nonempty list, `digits`, of integers between 0 and 9, and returns the number corresponding to `digits`.

For example,

- `list_to_num([9, 0, 8])` => 908
- `list_to_num([8, 6])` => 86
- `list_to_num([0, 6, 0])` => 60
QUESTION 2

Write an accumulatively recursive function `count_max` that consumes a nonempty list of integers `alon` and returns the number of times the largest integer in `alon` appears.

**Note:** `max` cannot be used in this question.

For example,

```
count_max([1, 3, 5, 4, 2, 3, 3, 3, 5]) => 2
```

since the largest element of the list, 5, appears twice. Your function should pass through the list only once.
Write a function `choose_by_colour` that consumes a list of Card (hand) and a string "red" or "black" (colour) and returns a list of the values of the Card in hand of the appropriate colour (spades and clubs are "black", hearts and diamonds are "red").

For example,

```python
choose_by_colour([[1,'hearts'],
                  [9,'spades'],
                  [3,'diamonds']], 'red')
⇒ [1,3]
```

Use abstract list functions to solve this problem.
Write a function sanitize that consumes a string, s, and returns a similar string but with any non-alphanumeric characters removed. Write this function using abstract list functions that operate on the consumed string.

• For example: `sanitize("@Test@")` => "Test"