Another hw3pr0 option (and EC option) – so much happening this week!
The Name Challenge Continues!

Stand up – and introduce yourself to SEVEN people to your left!

Around the room: “This is T, U, V, W, X, Y and Z and I am A.”
Design...

Code?

Algorithms!
Design...

design of what?

Safiya Umoja Noble PhD 🌀 @safiyanoble · Feb 2, 2018

Shameless plug: If everyone bought one right now for themselves, and one for a friend, this book could have a chance at improving the internet for women and people marginalized by tech...

amazon.com

Algorithms of Oppression: How Search Engines Reinforce Racism
Algorithms of Oppression: How Search Engines Reinforce Racism
Design...

design of what?

graduation, class of 2021
Algorithm
Design...

\text{remAll}(e, L)

\text{remove all } e's \text{ from } L
Design...

Top-down design

- Visualize
- Split into parts
- Build each part
- Combine
- Test

remAll(e, L)
remove all e's from L

remAll(42, [5, 7, 42, 8, 42])
\[5, 7, 8\]

remAll('q', 'qaqqlqqiqqiiqueqqnqs')
'aliiiiens'
Design...

Top-down design

Visualize

Split into parts

L[0] and L[1:]

Build each part

Combine

Test

remAll(e,L)

remove all e's from L

remAll(42, [5, 7, 42, 8, 42])

[5, 7, 8]

remAll('q', 'qaqqlqqiqqiiqiqqennqs')

'aliiens'

Use it!

'it'

Lose it!
Design...

Top-down design

- Visualize
- Split into parts
- Build each part
- Combine
- Test

remAll(e, L)
- remove all e's from L

remAll(42, [5, 7, 42, 8, 42])
- keep L[0]
- + remove e from the rest
- [5, 7, 8]

remAll(\text{'q'}, \text{'aqqllqqiqiqqiiiqeqqnqs'})
- drop L[0]
- + remove e from the rest
- \text{'aliiiiens'}
Design...

Top-down design

- Visualize
- Split into parts
- Build each part
- Combine
- Test

remAll(e,L)
remove all e's from L

remAll('q','xaqqlqqiqaqqiqeunnqs
aliiens
remAll(e,L)
remove all e's from L

keep L[0]
+ remove e from the rest

drop L[0]
+ remove e from the rest

Use it!

- or -

Lose it.
Design ~ code

```
def remAll( e, L ):
    """ removes all e's from L ""
    if len(L) == 0:
        return L
    elif L[0] != e:
        return L[0:1] + remAll(e,L[1:])
    else:
        return remAll(e,L[1:])
```

If there are no elements or characters in L, we're done – return L itself!
Removal of all 'e's from L:

```python
def remAll(e, L):
    """ removes all e's from L """
    if len(L) == 0:
        return L
    elif L[0] != e:
        return L[0:1] + remAll(e, L[1:])
    else:
        return remAll(e, L[1:])
```

- **Design ~ code**

Top-down design

Re-Visualize *in syntax*!?
Design ~ code

def remAll(e, L):
    """ removes all e's from L """
    if len(L) == 0:
        return L
    elif L[0] != e:
        return L[0:] + remAll(e, L[1:])
    else:
        return remAll(e, L[1:])

remAll(7, [7, 5, 42])
Design ~ code

That's it. Algorithmic expression ~ it's what CSers do. (think we)

```python
def remAll(e, L):
    """removes all e's from L""
    if len(L) == 0:
        return L
    elif L[0] != e:
        return L[0:1] + remAll(e, L[1:])
    else:
        return remAll(e, L[1:])
```

To visualize in syntax!?
Try it... Algorithm design

```python
def remAll( e, L ):
    """ removes all e's from L ""
    if len(L) == 0:
        return L
    elif L[0] != e:
        return L[0:1] + remAll(e,L[1:])
    else:
        return remAll(e,L[1:])
```

1. Change `remAll` so that it removes only one `e` from `L`. (We could call it `remOne`.)

   ```python
   remOne(8,[7,8,9,8]) ➞ [7,9,8]
   ```

   Hint: In both 1 + 2, what's needed is mostly crossing stuff out! What stuff?

2. Make more changes to `remAll` so that it removes all of the elements up to and including the first `e` in `L`. (We could call it `remUpto`.)

   ```python
   remUpto('d','coded') ➞ 'ed'
   ```

   If `e` is not in `L`, `remUpto` should remove everything...

3. Write the other cases needed for `subseq`...

```python
def subseq( s, sbig ):
    """ returns True if s is a subseq. of sbig, False otherwise. Both are strings. ""
    if s == '':
        return True
    elif
```

Challenge...

```python
subseq('alg','magical')
False
subseq('alg','twasbrillig')
True
```
def remAll( e, L ):
    """ removes all e's from L """
    if len(L) == 0:
        return L
    elif L[0] != e:
        return L[0:1] + remAll(e,L[1:])
    else:
        return remAll(e,L[1:])

1. Change remAll so that it removes only one e from L. (We could call it remOne.)

2. Make more changes to remAll so that it removes all of the elements up to and including the first e in L. (We could call it remUpto.)

def subseq( s, sbig ) :
    """ returns True if s is a subseq. of sbig, False otherwise. Both are strings. """
    if s == '':
        return True
    elif s != sbig[:len(s)]:
        return False
    else:
        return subseq(s[1:], sbig[1:])

3. Write the other cases needed for subseq...
Try it... Algorithm design

def remAll( e, L ):
    """ removes all e's from L """
    if len(L) == 0:
        return L
    elif L[0] != e:
        return L[0:1] + remAll(e,L[1:])
    else:
        return remAll(e,L[1:])

def subseq( s, sbig ):
    """ returns True if s is a subseq. of sbig, False otherwise. Both are strings. """
    if s == '':
        return True
    elif

# Change remAll so that it removes only one e from L. (We could call it remOne.)
remOne(8,[7,8,9,8]) → [7,9,8]

# Make more changes to remAll so that it removes all of the elements up to and including the first e in L. (We could call it remUpto.)
remUpto('d','coded') → 'ed'

# Challenge... Write the other cases needed for subseq...
subseq('alg','magical')
False

subseq('alg','twasbrillig')
True
def subseq( s, sbig ):
    """ returns True if s is a subseq. of sbig;
    False otherwise. Both are strings.  """
    if s == '':
        return True
    elif s[0] not in sbig:
        return False
    elif s[0] == sbig[0]:
        return subseq( s[1:], sbig[1:] )
    else:
        return subseq( s[0:], sbig[1:] )
What is a small (initial) piece of the problem?
How would we describe it in terms of the inputs?

What is left after handling this piece?
Are there other functions we will need?
def subseq( s, sbig ):
    """ returns True if s is a subseq. of sbig; False otherwise. Both are strings. """
    if s == '':
        return True
    elif s[0] not in sbig:
        return False
    elif s[0] == sbig[0]:
        return subseq( s[1:], sbig[1:] )
    else:
        return subseq( s[0:], sbig[1:] )
def subseq( s, sbig ):
    """ returns True if s is a subseq. of sbig;
    False otherwise. Both are strings. """
    if s == '':
        return True
    elif s[0] not in sbig:
        return False
    else:
        return subseq(s[1:], remUpto(s[0], sbig))
def subseq( s, sbig ):
    """ returns True if s is a subseq. of sbig; False otherwise. Both are strings. """
    if s == '':
        return True
    elif s[0] not in sbig:
        return False
    else:
        return subseq(s[1:], remUpto(s[0], sbig))

Where are the *useit* and *loseit* here?

"Use it or lose it"
What's the *problem*?!

Top-down design

- Visualize
- Split into parts
- Build each part
- Combine
- Test

Which *one* of these steps is the most important?
What's the problem?!

Top-down design

Visualize

Split into parts

Build each part

Combine

Test

understanding what the problem demands!!

I want some examples!
hw3pr2: use-it-or-lose-it algorithm design

Longest Common Subsequence

Jotto Score counting

binary list and general list sorting

exact_change making
**Practice:**  *edit distance*

```python
def ed(s1, s2):
```

Spell Checking challenge: what words are *close* to the non-word you typed?

**Base cases:**
- `s1 == ''` → `len(s2)` that many insertions
- `s2 == ''` → `len(s1)` that many deletions
- `s1 == s2` → return 0
- `s1[0] == s2[0]`  

```python
else:
    insert_cost = 
    delete_cost = 
    substitute_cost =
    return min(insert_cost, delete_cost, substitute_cost)
```

```
Speeding things up: Memoization

Spell Checking challenge: what words are *close* to the non—word you typed?
decipher( 'Weet bksa ed xecumeha 3!' )

kxn rkfo k qbokd goouoxn ...
decipher( 'Weet bksa ed xecumeha 3!' )

Good luck on homework 3!

kxn rkfo k qbokd goouoxn ...

and have a great weekend ...