BR 5 Snczx

Algorithms

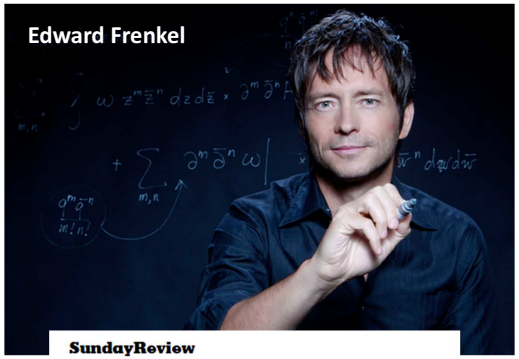
Englishness…
Classifying life
Removing/Sorting
and *Jotto!*

HW 3

Hw #3 due **Monday,** 11:59

Sound Lab!

Several algorithms …



Warm-up + cool-down from lab…

```
[2*x for x in L]
```

L = [1, 11, 21]

```
N = len(L)
[2*L[i] for i in range(N)]
```

But *why* use indices?!

```
[??? + x for x in L]
[ _________  for x in L]
```

Aargh!

L = [1, 11, 21]

```
N = len(L)
[L[i-1] + L[i] for i in range(N)]
```

Design…

remAll(e, L)

remove all e's from L

Top-down design

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

Use it!

it

'the rest'

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

keep L[0]
+ remove e from the rest

```
[5, 7, 8]
```

it

'the rest'

```
remAll('q', 'qaqqlqqiqqiiqeqqnqs')
```

drop L[0]
+ remove e from the rest

Lose it!

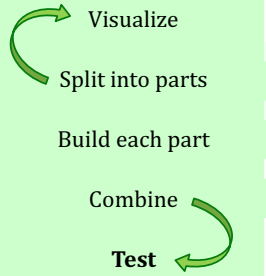
```
'aliiiens'
```

Design ~ *code*

`remAll(e, L)`

remove all e's from L

Top-down design




```
def remAll(e, 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:])
```

Use it!

Lose it!

other **rem** examples...

```
remAll(8, [7,8,9,8]) ➡ [7,9]        remAll
remOne(8, [7,8,9,8]) ➡ [7,9,8]      remOne
remUpto(8,[7,8,9,8]) ➡ [9,8]        remUpto
```

```
remAll('d', 'coded') ➡ 'coe'        remAll
remOne('d', 'coded') ➡ 'coed'       remOne
remUpto('d','coded') ➡ 'ed'         remUpto
```

Subsequences

in order, but not necessarily *adjacent*...

```
def subseq(s, sbig)   ⟶  True or False?
```

s is the subsequence to find (or not)

sbig is the bigger string in which we are looking for **s**

T or F?

```
subseq('', 'cataga') ⟶ True
subseq('ctg', 'cataga') ⟶ True
subseq('ctg', 'tacggta') ⟶
subseq('aliens', 'always frighten dragons')⟶
subseq('trogdor', 'that dragon is gone for good')
                                    ⟶
```

Here there be NO dragons!

Why are these True? or False?

from **remAll** to **remOne**

Hint: remove one thing for `remOne`!

```
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:])
```

`remOne(8, [7,8,9,8])➡ [7,9,8]` `remOne('d', 'coded')➡ 'coed'`

from **remOne** to **remUpto**

Hint: remove one *more* thing for **remUpto**!

```
def remOne(e, L):
    """Returns sequence L with one e removed.
    """
    if  len(L) == 0:
        return L

    elif  L[0] != e:
        return  L[0:1] + remOne(e, L[1:])

    else:
        return  L[1:]
```

```
remUpto(8, [7,8,9,8]) ➡ [9,8]          remUpto('d', 'coded') ➡ 'ed'
```

Subseq ~ *trying (coding) it out…*

Hint: you'll need 4 cases total for **subseq**.

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

Base case(s)

Recursive step(s)

```
subseq('alg', 'magical')
False
```

```
subseq('alg', 'twasbrillig')
True
```

Let's try it!

Subseq ~ *thinking* it out…

subseq(s, sbig)

subseq('ctg', 'tacggta')

Use it!

- or -

Lose it!

What is a small (initial) piece of the problem?
How would we describe it in terms of the arguments?

What is left after handling this piece?
Are there other functions we will need?

Top-down design

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

hw3pr2: *use it or lose it*

Longest **C**ommon **S**ubsequence LCS(S, T)

'HUMAN'

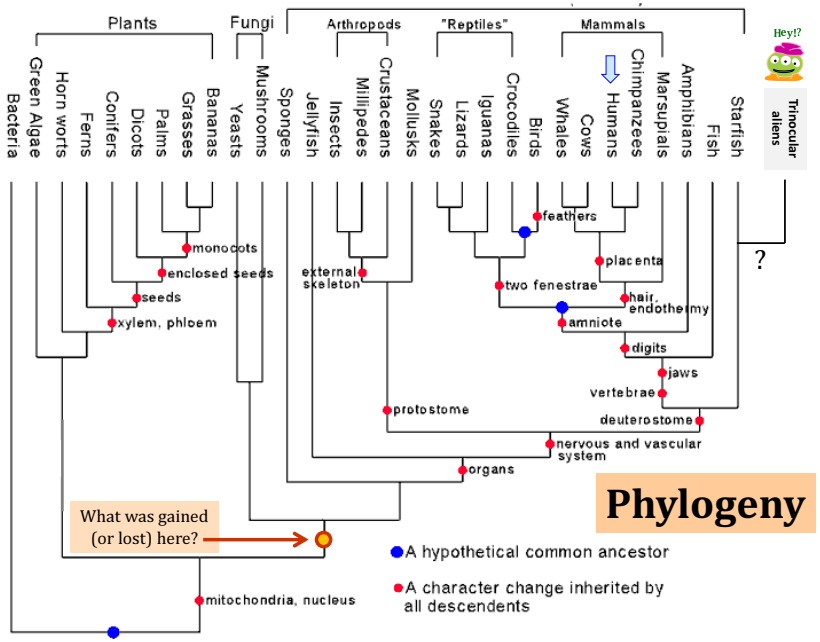
'CHIMPANZEE'

'CGCTGAGCTAGGCA…'

'ATCCTAGGTAACTG…'

+10^9 more

Eye oneder if this haz other aplications?



also in hw4pr2: *Jotto* !

jscore(S, T)

'robot'

'otter'

These are two cute

jscore('robot', 'otter') →

Let's try it!

should return the jotto score for any strings **s1** and **s2**

jscore(s1,s2)

jscore('robot', 'otter') → 3
jscore('geese', 'seems') → 3
jscore('fluff', 'lulls') → 2
jscore('pears', 'diner') →
jscore('xylyl', 'slyly') →

Extra! Which of these 10 is the *cruellest* hidden jotto word?

Use it! Lose it! **remOne**

should return a new list that is the sorted version of the input **L**

sort(L)

sort([42,5,7]) → [5,7,42]
sort([42,7]) → [7,42]
sort([42]) → [42]
sort([ ]) →
blsort([1,0,1]) →

binary-list sort: same as sort, but all of the #s are 0 or 1

don't write any code for these...

should return the Longest Common Subsequence of strings **S** and **T**

LCS(S,T)

LCS('ctga', 'tagca') → 'tga'
LCS('tga', 'taacg') → 'ta' (or 'tg')
LCS('tga', 'a') → 'a'
LCS('gattaca', 'ctctgcgat') →

4 chars

do try the examples + brainstorm

Brainstorm algorithms for these problems -- what **helper functions** might help each one?

returns True if ***any*** subset of elements in L add up to t; returns False otherwise

exact_change(t,L)

exact_change(42, [25,30,2,5]) → False
exact_change(42, [40,17,1,7]) →
exact_change(42, [18,21,22]) →
exact_change(42, [22,16,3,2,17]) →
exact_change(20, [16,3,2,17]) →

Name(s):

Quiz

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:])
```

Hint: In both cases, all that's needed is *crossing stuff out*! *What stuff?*

1

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]**

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**.)

remUpto('d','coded') ➡ **'ed'**

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

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

Challenge...

3

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

```
subseq('alg', 'magical')
False

subseq('alg', 'twasbrillig')
True
```