 Algorithms

Englishness...
Classifying life
Removing/Sorting
and Jotto!

HW 3

Hw #3 due Monday, 11:59

Sound Lab!

Several algorithms...

Office Hrs.!

Friday, 2:30-4:30,
HMC's LAC lab...

Edward Frenkel

Is the Universe a Simulation?

FEB. 14, 2014
Sound + Starbucks!

We saw you in Lab3...
def flipflop(filename):
    """flipflop swaps the halves of an audio file
    input: filename, the name of the original file
    output: no return value, but
    this creates the sound file 'out.wav'
    and plays it"
    
    print( "Playing the original sound..."
    )
    play(filename)

    print( "Reading in the sound data..."
    )
    sound_data = [0,0]
    read_wav(filename,sound_data)
    samps = sound_data[0]
    sr = sound_data[1]

    print( "Computing new sound..."
    )
    # this gets the midpoint and calls it x
    x = len(samps)//2
    newsamps = samps[x:] + samps[:x]

    newsr = sr
    new_sound_data = [ newsamps, newsr ]

    print( "Writing out the new sound data..."
    )
    write_wav( new_sound_data, "out.wav" ) # write data to out.wav

    print( "Playing new sound...")
    play( 'out.wav' )

intro stuff – important, but less algorithmic

algorithmic stuff

"outro" stuff
Algorithms

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

HW 3

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

Sound Lab!

Several algorithms...

Office Hrs.!

Friday, 2:30-4:30,
HMC's LAC lab...

Edward Frenkel

*Is the Universe a Simulation?*

EB. 14, 2014
**BR 5 Snczx**

**Edward Frenkel**

*Love & Math: The Heart of Hidden Reality*

---

**Algorithms**

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

**HW 3**

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

**Sound Lab!**

*Several algorithms...*

**Office Hrs.**

Friday, 2:30-4:30, HMC's LAC lab...
Today in CS5:
The ♥ of CS (and CSers...)

Algorithms!
Caesar Cipher: \texttt{encipher}

\texttt{encipher(s,n)} should return the string \texttt{s} with each \textit{alphabet}ic character shifted/wrapped by \texttt{n} places in the alphabet.

\begin{align*}
\text{encipher( 'I <3 Latin' , 0 )} & \quad \text{returns} \quad 'I <3 Latin' \\
\text{encipher( 'I <3 Latin' , 1 )} & \quad \text{returns} \quad 'J <3 Mbujo' \\
\text{encipher( 'I <3 Latin' , 2 )} & \quad \text{returns} \quad 'K <3 Ncvkp' \\
\text{encipher( 'I <3 Latin' , 3 )} & \quad \text{returns} \quad 'L <3 Odwlq' \\
\text{encipher( 'I <3 Latin' , 4 )} & \quad \text{returns} \quad 'M <3 Pexmr' \\
\text{encipher( 'I <3 Latin' , 5 )} & \quad \text{returns} \quad 'N <3 Qfyns' \\
\vdots & \quad \vdots \\
\text{encipher( 'I <3 Latin' , 25 )} & \quad \text{returns} \quad 'H <3 Kzshm'
\end{align*}
ASCII and Unicode

<table>
<thead>
<tr>
<th>Binary</th>
<th>Dec</th>
<th>Hex</th>
<th>Glyph</th>
</tr>
</thead>
<tbody>
<tr>
<td>0010 0000</td>
<td>32</td>
<td>20</td>
<td>(blank) ( )</td>
</tr>
<tr>
<td>0010 0001</td>
<td>33</td>
<td>21</td>
<td>!</td>
</tr>
<tr>
<td>0010 0010</td>
<td>34</td>
<td>22</td>
<td>&quot;</td>
</tr>
<tr>
<td>0010 0011</td>
<td>35</td>
<td>23</td>
<td>#</td>
</tr>
<tr>
<td>0010 0100</td>
<td>36</td>
<td>24</td>
<td>$</td>
</tr>
<tr>
<td>0010 0101</td>
<td>37</td>
<td>25</td>
<td>%</td>
</tr>
<tr>
<td>0010 0110</td>
<td>38</td>
<td>26</td>
<td>&amp;</td>
</tr>
<tr>
<td>0010 0111</td>
<td>39</td>
<td>27</td>
<td>'</td>
</tr>
<tr>
<td>0010 1000</td>
<td>40</td>
<td>28</td>
<td>(</td>
</tr>
<tr>
<td>0010 1001</td>
<td>41</td>
<td>29</td>
<td>)</td>
</tr>
<tr>
<td>0010 1010</td>
<td>42</td>
<td>2A</td>
<td>*</td>
</tr>
<tr>
<td>0010 1011</td>
<td>43</td>
<td>2B</td>
<td>+</td>
</tr>
<tr>
<td>0010 1100</td>
<td>44</td>
<td>2C</td>
<td>,</td>
</tr>
<tr>
<td>0010 1101</td>
<td>45</td>
<td>2D</td>
<td>-</td>
</tr>
<tr>
<td>0010 1110</td>
<td>46</td>
<td>2E</td>
<td>.</td>
</tr>
<tr>
<td>0010 1111</td>
<td>47</td>
<td>2F</td>
<td>/</td>
</tr>
<tr>
<td>0011 0000</td>
<td>48</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>0011 0001</td>
<td>49</td>
<td>31</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bin</th>
<th>Dec</th>
<th>Hex</th>
<th>Glyph</th>
</tr>
</thead>
<tbody>
<tr>
<td>0100 0000</td>
<td>64</td>
<td>40</td>
<td>@</td>
</tr>
<tr>
<td>0100 0001</td>
<td>65</td>
<td>41</td>
<td>A</td>
</tr>
<tr>
<td>0100 0010</td>
<td>66</td>
<td>42</td>
<td>B</td>
</tr>
<tr>
<td>0100 0011</td>
<td>67</td>
<td>43</td>
<td>C</td>
</tr>
<tr>
<td>0100 0100</td>
<td>68</td>
<td>44</td>
<td>D</td>
</tr>
<tr>
<td>0100 0101</td>
<td>69</td>
<td>45</td>
<td>E</td>
</tr>
<tr>
<td>0100 0110</td>
<td>70</td>
<td>46</td>
<td>F</td>
</tr>
<tr>
<td>0100 0111</td>
<td>71</td>
<td>47</td>
<td>G</td>
</tr>
<tr>
<td>0100 1000</td>
<td>72</td>
<td>48</td>
<td>H</td>
</tr>
<tr>
<td>0100 1001</td>
<td>73</td>
<td>49</td>
<td>I</td>
</tr>
<tr>
<td>0100 1010</td>
<td>74</td>
<td>4A</td>
<td>J</td>
</tr>
<tr>
<td>0100 1011</td>
<td>75</td>
<td>4B</td>
<td>K</td>
</tr>
<tr>
<td>0100 1100</td>
<td>76</td>
<td>4C</td>
<td>L</td>
</tr>
<tr>
<td>0100 1101</td>
<td>77</td>
<td>4D</td>
<td>M</td>
</tr>
<tr>
<td>0100 1110</td>
<td>78</td>
<td>4E</td>
<td>N</td>
</tr>
<tr>
<td>0100 1111</td>
<td>79</td>
<td>4F</td>
<td>O</td>
</tr>
<tr>
<td>0101 0000</td>
<td>80</td>
<td>50</td>
<td>P</td>
</tr>
<tr>
<td>0101 0001</td>
<td>81</td>
<td>51</td>
<td>Q</td>
</tr>
</tbody>
</table>

This is why 'CS' < 'clear'!
def rot13(c):
    """ rotates c by 13 chars, "wrapping" as needed
    NON-LETTERS don't change!
    """
    if 'a' <= c <= 'z':
        if ord(c)+13 <= ord('z'):
            return chr(ord(c)+13)
        else:
            return chr(ord(c)+13-26)
    elif 'A' <= c <= 'Z':  # upper-case test!
        same, but for 'Z'
    else:
        return c

(0) What do these tests do?
(1) What code will "wrap" to the alphabet's other side?
(2) How will upper case change? Try noting only the code differences...
(3) What if c is not a letter at all?

Extra: How would you rotate n places, instead of 13?
Caesar Cipher: \textit{encipher}

```python
>>> encipher('Bzdrzq bhogdq? H oqdedq Bzdrzq rzkzc.', 25)
'Aycqyp agnfcp? G npcdcp Aycqyp qyjyb.'
```

```python
>>> encipher('Bzdrzq bhogdq? H oqdedq Bzdrzq rzkzc.', 15)
'Qosgof qwdvsf? W dfstsf Qosgof gozor.'
```

```python
>>> encipher('Bzdrzq bhogdq? H oqdedq Bzdrzq rzkzc.', 4)
'Fdhvdu flskhu? L suhihu Fdhvdu vdodg.'
```

```python
>>> encipher('Bzdrzq bhogdq? H oqdedq Bzdrzq rzkzc.', 1)
'Caesar cipher? I prefer Caesar salad.'
```

```python
>>> encipher('Hu lkbjhapvu pz doha ylthpuz hmaly dl mvynla \lclyfaopun dl ohcl slhyulk.', 19)
'An education is what remains after we forget everything we have learned.'
```
Caesar Cipher: **decipher**

```python
>>> decipher('Bzdrzq bhogd? H oqdedq Bzdrzq rzkzc.')
'Caesar cipher? I prefer Caesar salad.'

>>> decipher('Hu lkbjhapvu pz doha ylthpuz hmaly dl mvynla \lclyfaopun dl ohcl slhyulk.')
'An education is what remains after we forget everything we have learned.'

>>> decipher('Uifz xpsl ju pvu xjui b qfodjm!')

>>> decipher('gv vw dtwvg')

But **how**?
Decipher?

Strategies?

All possible decipherings

Algorithms?
Measuring *Englishness*

Very English-y

- "Call me Ishmael."
- "Attack at dawn!"
- "rainbow, table, candle"
- "Yow! Legally-imposed CULTURE-reduction is CABBAGE-BRAINED!"
- "quadruplicity drinks procrastination"
- "Hold the newsreader's nose squarely, waiter, or friendly milk will countermand my trousers."
- "the gostak distims the doshes"
- "hension, framble, bardle"
- "jufict, stofwus, lictpub"
- "itehbs, rsnevtr, khbsota"
- "epadxo, nojarpn, gdxokpw"
- "h o q dedqBzdrzqrzkzc"

All of these sound good to me!

lower scores

Not English-y

quantifying "Englishness"?

higher scores
Decipher?

All possible decipherings

Strategies?

Algorithms?

"Englishness " score ~ the #-of-vowels

Score them all

max!

[4, 'la ab iybal']
### Decipher?

All possible decipherings:
- gv vw dtwvg
- hw wx euxwh
- ix xy fvyxi
- jy yz gwzyj
- kz za hxazk
- la ab iybal
- mb bc jzcbm
- nc cd kadcn
- od de lbedo
- pe ef mcfep
- qf fg ndgfq
- rg gh oehgr
- sh hi pfihs
- ti ij qgjit
- uj jk rhkju
- vk kl silky
- wl lm tjmlw

### Strategies?

### Algorithms?

"Englishness" based on letter-probabilities:
- 0.45555, 'et tu brute'
- 0.00011, 'fu uv csvuf'

Scores:
- 0.00524, 'nc cd kadcn'
- 0.29041, 'od de lbedo'
- 0.00874, 'pe ef mcfep'
- 7.3e-07, 'qf fg ndgfq'
- 0.06410, 'rg gh oehgr'
- 0.11955, 'sh hi pfihs'
- 3.1e-06, 'ti ij qgjit'
- 1.1e-08, 'uj jk rhkju'
- 2.6e-05, 'vk kl silky'
- 0.00012, 'wl lm tjmlw'
- 3.1e-06, 'xm mn uknmx'
- 0.02011, 'yn no vlony'
- 1.5e-06, 'zo op wmpoz'
- 1.9e-07, 'ap pq xnqpa'
- 5.7e-08, 'bq qr yorqb'
- 0.00024, 'cr rs zpsrc'
- 0.00660, 'ds st aqtsd'
- 0.45555, 'et tu brute'
- 0.00011, 'fu uv csvuf'

Max: [0.45555, 'et tu brute']
Decipher?

Strategies?

Algorithms?

decPR(LAT)
decPR2(LAT)
decPR3(LAT)

All possible decipherings

Using the **LoL** technique to score each rotation's "Englishness"

"Englishness" based on scrabble-scoring!

[27, 'gv vw dtwvg']
[38, 'hw wx euxwh']
[42, 'ix xy fvyxi']
[54, 'jy yz gwzyj']
[54, 'kz za hxazk']
[16, 'la ab iybal']
[39, 'mb bc jzcbm']
[21, 'nc cd kadcn']
[16, 'od de lbedo']
[pe ef mcfep]
[33, 'ti ij qgjit']
[41, 'uj jk rhkju']
[27, 'vk kl silkv']
[26, 'wl lm tjmlw']
[33, 'xm mn uknmx']
[18, 'yn no vlony']
[36, 'zo op wmpoz']
[40, 'ap pq xnqpa']
[43, 'bq qr yorqb']
[24, 'cr rs zpsrc']
[20, 'de et aqtsd']
[11, 'et tu brute']
[25, 'fu uv csvuf']

S c o r e s

S c o r e s

S c o r e s

S c o r e s
Design...

design of what?

The ♥ of CS
(and CSers...)

Algorithms!
Design...

design of what?

Code?
syntax
The Economist explains
What is code?
Sep 8th 2015, 23:50 BY T.S.

FROM lifts to cars to airliners to smartphones, modern civilisation is powered by software, the digital instructions that allow computers, and the devices they control, to perform calculations and respond to their surroundings. How did that software get there? Someone had to write it. But code, the sequences of symbols painstakingly created by programmers, is not quite the same as software, the sequences of instructions that computers execute. So what exactly is it?

Coding, or programming, is a way of writing instructions for computers that bridges the gap between how humans like to express themselves and how computers actually work. Programming languages, of which there are hundreds, cannot generally be executed by computers directly. Instead, programs written in a particular "high level" language such as C++, Python or Java are translated by a special piece of software (a compiler or an interpreter) into low-level instructions which a computer can actually run. In some cases programmers write software in low-level instructions directly, but this is fiddly. It is usually much easier to use a high-level programming language, because such languages make it
Design...

Code?

Algorithms!
Algorithm
Design...

remAll(e,L)
remove all e's 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', 'qaqqlqqiqqiiqeqqnqs')
'aliiiiens'
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','qaqqlqqiqqqiqeqqqnqs')
'aliiens'

Use it!
L[0] and L[1:]
'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])
[5, 7, 8]

keep L[0]
+ remove e from the rest

remAll('q', 'aqqlqqiqqliqqiiqeqqnqs')
'aliiiens'

drop L[0]
+ remove e from the rest

Lose it!
Design...

Top-down design

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

Use it!

- or -

Lose it.

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

remAll('q',"alice")
Allie Russell, '12

speaking of roadside church signs...
Allie Russell, '12
Design ~ code

Top-down design

Re-Visualize *in syntax!*?

```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:])

If there are no elements or characters in L, we're done –
return L itself!
```
Design ~ code

Top-down design

Re-Visualize in syntax!? [7, 5, 42]

```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!?  

```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:])

If it is e, LOSE it (don't keep it in the return value)
AND still remove all of the e's from the rest of L!
```

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

[7, 5, 42]
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:])
```python
def remAll(e, L):
    # Removes all copies of e 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:])
```

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

Don't start yet...

Algorithm design

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'

3. Write the other cases needed for `subseq`...
   - `subseq('alg', 'magical')` → False
   - `subseq('alg', 'twasbrillig')` → True

Name(s):
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:])

Algorithm design

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'

3
Write the other cases needed for subseq...

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

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:])

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

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

Name(s):

Algorithm design

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

remOne(8,[7,8,9,8]) \rightarrow [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') \rightarrow 'ed'

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

Write the other cases needed for subseq...

subseq('alg','magical')
False

subseq('alg','twasbrillig')
True
**remAll** insight

```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:])
```

```
remAll(8, [7,8,9,8]) ➞ [7,9]
```

sharpening our model for where + how actions happen...
other \texttt{rem} examples...

\texttt{remAll}(8, [7,8,9,8]) \Rightarrow [7,9] \quad \texttt{remAll}

\texttt{remAll}(\texttt{d}', \texttt{coded}') \Rightarrow \texttt{coe}' \quad \texttt{remAll}

\texttt{remOne}(8, [7,8,9,8]) \Rightarrow [7,9,8] \quad \texttt{remOne}

\texttt{remOne}(\texttt{d}', \texttt{coded}') \Rightarrow \texttt{coed}' \quad \texttt{remOne}

\texttt{remUpto}(8,[7,8,9,8]) \Rightarrow [9,8] \quad \texttt{remUpto}

\texttt{remUpto}(\texttt{d}',\texttt{coded}') \Rightarrow \texttt{ed}' \quad \texttt{remUpto}
Subsequences

```python
def subseq(s, sbig)

s is the subsequence to find (or not)

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

<table>
<thead>
<tr>
<th>subseq('', 'cataga') → True</th>
<th>subseq('ctg', 'cataga') → True</th>
</tr>
</thead>
<tbody>
<tr>
<td>subseq('ctg', 'tacggta') →</td>
<td>subseq('aliens', 'always frighten dragons') →</td>
</tr>
<tr>
<td>subseq('trogdor', 'that dragon is gone for good') →</td>
<td></td>
</tr>
</tbody>
</table>

Why Are these True? or False?

In order, but not necessarily adjacent...

Here there be NO dragons!
Name(s):

Quiz

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:])

# Algorithm design

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

Hint: remove one thing for remOne!

Hint: remove one more thing for remUpto!

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

3

Write the other cases needed for subseq...

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

```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:])

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

Try it out!

```python
remOne(8, [7, 8, 9, 8])  ⇒  [7, 9, 8]
remUpto('d', 'coded')  ⇒  'ed'
```

Algorithm design

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

Challenge...

```python
remUpto('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:])

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] in sbig:
        return subseq(s[1:],sbig)
    else:
        return False

Name(s):

Algorithm design

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'

3. Write the other cases needed for subseq...
   - subseq('alg','magical') ➞ True
   - subseq('alg','twasbrillig') ➞ False

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

Hint: you'll need 3-4 cases total for subseq.
from `remAll` to `remOne`

```python
def remAll(e, L):
    """ returns seq. L with all e's rmovd """
    if len(L) == 0:
        return L
    elif L[0] != e:
        return L[0:1] + remAll(e, L[1:]
    else:
        return remAll(e, L[1:]

from remAll to remOne

One
def remAll(e, L):
    """ returns seq. L with all e's rmovd """
    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

```python
def remOne(e, L):
    ''' returns seq. 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'
```

Hint: remove one more thing for remUpto!
def subseq( s, sbig ):
    """ returns True if s is a subseq. of sbig; False otherwise. Both are strings. """
    if s == '':
        return True

subseq('alg','magical')
False

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

```
subseq('alg','magical')  # False
subseq('alg','twasbrillig')  # True
```
Subseq ~ thinking it out...

\[
\text{subseq( } s, \text{ sbig } \text{ )}
\]

\[
\text{subseq('ctg', 'tacggta')}
\]

\[
\text{s[0]} \quad \text{subseq[0]}
\]

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?

- or -

Use it!

Lose it!
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 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

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

remOne(8,[7,8,9,8]) \rightarrow [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') \rightarrow 'ed'

3
Write the other cases needed for subseq...

subseq('alg','magical')
False

subseq('alg','twasbrillig')
True
Design ~ (code)

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

can take some "getting used to"...?
... at this time in prior CS5 ...
random lols

Anna Marburger via cs.hmc.edu
to dodds

This is from this past Wednesday

Zachary Dodds <dodds@cs.hmc.edu>

Feb 14 (2 days ago)

Thanks for this, Anna – this is great! (though I do think I only understand a fraction of what's going on...)

Best wishes!

Zach
Anna Marburger via cs.hmc.edu

Feb 12 (4 days ago)

This is from this past Wednesday.

Zachary Dodds <dodds@cs.hmc.edu>

Feb 14 (2 days ago)

Thanks for this, Anna -- this is great! (though I do think I only understand a fraction of what's going on...)

Best wishes!

Zach

Anna Marburger via cs.hmc.edu

Feb 15 (1 day ago)

Welcome to how I feel about CS 5
Design ~ (code)

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

can take some "getting used to"...?

million "troubles" vs. million "triumps"
Top-down
Visualize
Test
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:]):

Split into parts
Build each part
Combine
remAll(e, L)
remove all e's from L
Teal...!
Top-down design
Visualize
Test

```python
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:])
```

Split into parts
Build each part
Combine

At the Career Fair!
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:])

Top-down design
Visualize
Test

Split into parts
Build each part
Combine
remAll(e, L)
remove all e’s from L

At the Career Fair!
What's the **problem**?!

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*

**Longest Common Subsequence**

**Jotto Score** counting

**binary list and general list sorting**

**exact_change** making
**hw3pr2:** use it or lose it

**Longest Common Subsequence**

LCS( S, T )

- 'HUMAN'
- 'CHIMPANZEE'
- 'CGCTGAGCTAGGCA...'
- 'ATCCTAGGTAACTG...'

Additional: +10^9 more

Eye oneder if this haz other aplications?
Why LCS?

Algorithmic challenge: How to find the best common subsequences among these very big genome strings?!!
What was gained (or lost) here?

Phylogeny
What was gained (or lost) here?

Night-loving species!

Mourning species...?

What was gained (or lost) here?

cinema

caffeine

chocolate

coffee

coding
What was gained (or lost) here?

NIGHT-LOVING SPECIES!

CINEMA
CAFÉINE
COFFEE

CHOCOLATE
CODING
~ 5CS! ~
Subsequences @ 5Cs

host: figs

parasites: wasps

matching two phylogenies together

together!
Jane's source data: 100s of species, 6 continents ...

Largest co-phylogeny ever computed
also in hw3pr2: *Jotto*!

*a word-guessing game...*

\[
\text{jscore}(S, T)
\]
These are two cute 'robot' 'otter'

Let's try it!

```
jscore( 'robot', 'otter' )
```

```
jscore( S, T )
```
also in hw3pr2: sort + exact_change

\[
\text{sort( [42,5,7] )} \quad \rightarrow \\
\text{sort( [42,7] )} \quad \rightarrow \\
\text{sort( [42] )} \quad \rightarrow \\
\]

returns an ascending list
returns True or False

\[
\text{exact_change( 42, [25,30,2,5] )} \quad \rightarrow \\
\text{exact_change( 42, [25,30,2,15] )} \quad \rightarrow \\
\]
should return the jotto score for any strings \( s_1 \) and \( s_2 \)

\[
\text{jscore}(s_1, s_2)
\]

\[
\begin{align*}
\text{jscore('robot', 'otter')} & \rightarrow 3 \\
\text{jscore('geese', 'seems')} & \rightarrow 3 \\
\text{jscore('fluff', 'lulls')} & \rightarrow 2 \\
\text{jscore('pears', 'diner')} & \rightarrow 2 \\
\text{jscore('xylyl', 'slyly')} & \rightarrow 2
\end{align*}
\]

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

Use it! Lose it!

should return a new list that is the sorted version of the input \( L \)

\[
\text{sort}( L )
\]

\[
\begin{align*}
\text{sort( [42,5,7] )} & \rightarrow [5,7,42] \\
\text{sort( [42,7] )} & \rightarrow [7,42] \\
\text{sort( [42] )} & \rightarrow [42] \\
\text{blsort( [1,0,1] )} & \rightarrow \\
\text{sort( [] )} & \rightarrow 
\end{align*}
\]

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

Brainstorm algorithms for these problems. What helper functions might help for each...

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

\[
\text{exact_change}( t, L )
\]

\[
\begin{align*}
\text{exact_change( 42, [25,30,2,5] )} & \rightarrow \text{False} \\
\text{exact_change( 42, [22,16,3,2,17] )} & \rightarrow \text{False} \\
\text{exact_change( 42, [18,21,22] )} & \rightarrow \text{False} \\
\text{exact_change( 42, [40,17,1,7] )} & \rightarrow \text{False} \\
\text{exact_change( 20, [16,3,2,17] )} & \rightarrow \text{False}
\end{align*}
\]
Don't write any code for these…

**jscore(s1,s2)**

- `jscore('robot', 'otter')` → 3
- `jscore('geese', 'seems')` → 3
- `jscore('fluff', 'lulls')` → 2
- `jscore('pears', 'diner')` → 2
- `jscore('xylyl', 'slyly')` → 4

**sort( L )**

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

**LCS(S,T)**

- `LCS('ctga', 'tagca')` → 'tga'
- `LCS('tga', 'taacg')` → 'ta' (or 'tg')
- `LCS('tga', 'a')` → 'a'
- `LCS('gattaca', 'ctctgcgat')` → 'ttca' (4 chars)

**exact_change(t,L)**

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

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

Use it!  Lose it!

Don't write any code for these…

Use it!  Lose it!

Do try the examples + brainstorm

Brainstorm algorithms for these problems. What **helper functions??** might help for each…

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

**exact_change(t,L)**

Use it!  Lose it!
Don't write any code for these…

### jscore

<table>
<thead>
<tr>
<th>s1</th>
<th>s2</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>'robot'</td>
<td>'otter'</td>
<td>3</td>
</tr>
<tr>
<td>'geese'</td>
<td>'seems'</td>
<td>3</td>
</tr>
<tr>
<td>'fluff'</td>
<td>'lulls'</td>
<td>2</td>
</tr>
<tr>
<td>'pears'</td>
<td>'diner'</td>
<td>2</td>
</tr>
<tr>
<td>'xylyl'</td>
<td>'slyly'</td>
<td>4</td>
</tr>
</tbody>
</table>

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

### sort

<table>
<thead>
<tr>
<th>L</th>
<th>Sorted L</th>
</tr>
</thead>
<tbody>
<tr>
<td>[42, 5, 7]</td>
<td>[5, 7, 42]</td>
</tr>
<tr>
<td>[42, 7]</td>
<td>[7, 42]</td>
</tr>
<tr>
<td>[42]</td>
<td>[42]</td>
</tr>
<tr>
<td>[]</td>
<td>[]</td>
</tr>
<tr>
<td>[1, 0, 1]</td>
<td>[0, 1, 1]</td>
</tr>
</tbody>
</table>

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

### LCS

<table>
<thead>
<tr>
<th>S</th>
<th>T</th>
<th>LCS</th>
</tr>
</thead>
<tbody>
<tr>
<td>'ctga'</td>
<td>'tagca'</td>
<td>'tga'</td>
</tr>
<tr>
<td>'tga'</td>
<td>'taacg'</td>
<td>'ta' (or 'tg')</td>
</tr>
<tr>
<td>'tga'</td>
<td>'a'</td>
<td>'a'</td>
</tr>
<tr>
<td>'gattaca'</td>
<td>'ctctgca'</td>
<td>'ttca'</td>
</tr>
</tbody>
</table>

**4 chars**

### exact_change

<table>
<thead>
<tr>
<th>t, L</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>42, [25, 30, 2, 5]</td>
<td>False</td>
</tr>
<tr>
<td>42, [22, 16, 3, 2, 17]</td>
<td>True</td>
</tr>
<tr>
<td>42, [18, 21, 22]</td>
<td>False</td>
</tr>
<tr>
<td>42, [40, 17, 1, 7]</td>
<td>False</td>
</tr>
<tr>
<td>20, [16, 3, 2, 17]</td>
<td>True</td>
</tr>
</tbody>
</table>

**Brainstorm algorithms** for these problems -- what helper functions?? might help for each?
decipher( 'Weet bksa ed xecumeha 3!' )
decipher( 'Weet bksa ed xecumeha 3!' )

Good luck on homework 3!

kxn rkfo k qbokd goouoxn ...

and have a great weekend ...