Algorithms

Edward Frenkel

Is the Universe a Simulation?
FEB. 14, 2014

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

HW 4
due Monday, 11:59

Sound Lab!
Several algorithms...

Office Hrs.!
Today, 2-4, HMC's LAC lab...
Sound + Starbucks!

We saw you in Lab4... heard
Take-away ~ Lab4

```python
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 = [ newssamps, 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' )
```
Englishness...
Classifying life
Removing/Sorting and Jotto!

HW 4
due **Monday, 11:59**

Several algorithms...

*Is the Universe a Simulation?*

FEB. 14, 2014
**Algorithms**

*Englishness*

Classifying life

Removing/Sorting

and Jotto!

---

**HW 3**

Hw #3 due **Monday, 1:**

Sound Lab!

*Several algorithms.*
Caesar Cipher: `encipher`  

**encipher**\((s, n)\)  

should return the string \(s\) with each alphabetic character shifted/wrapped by \(n\) places in the alphabet

**Examples:**

\[
\begin{align*}
\text{encipher( 'I <3 Latin' , 0 )} & \quad \text{returns} \quad 'I <3 Latin' \\
\text{encipher( 'I <3 Latin' , 1 )} & \quad \text{returns} \quad 'J <3 Mbujo' \\
\text{encipher( 'I <3 Latin' , 2 )} & \quad \text{returns} \quad 'K <3 Ncvkp' \\
\text{encipher( 'I <3 Latin' , 3 )} & \quad \text{returns} \quad 'L <3 Odwlq' \\
\text{encipher( 'I <3 Latin' , 4 )} & \quad \text{returns} \quad 'M <3 Pexmr' \\
\text{encipher( 'I <3 Latin' , 5 )} & \quad \text{returns} \quad 'N <3 Qfyns' \\
\cdots \\
\text{encipher( 'I <3 Latin' , 25 )} & \quad \text{returns} \quad 'H <3 Kzshm'
\end{align*}
\]
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) (sp)</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 0100</td>
<td>66</td>
<td>42</td>
<td>B</td>
</tr>
<tr>
<td>0100 0101</td>
<td>67</td>
<td>43</td>
<td>C</td>
</tr>
<tr>
<td>0100 1000</td>
<td>68</td>
<td>44</td>
<td>D</td>
</tr>
<tr>
<td>0100 1001</td>
<td>69</td>
<td>45</td>
<td>E</td>
</tr>
<tr>
<td>0100 1100</td>
<td>70</td>
<td>46</td>
<td>F</td>
</tr>
<tr>
<td>0100 1101</td>
<td>71</td>
<td>47</td>
<td>G</td>
</tr>
<tr>
<td>0100 1110</td>
<td>72</td>
<td>48</td>
<td>H</td>
</tr>
<tr>
<td>0100 1111</td>
<td>73</td>
<td>49</td>
<td>I</td>
</tr>
<tr>
<td>0101 0000</td>
<td>74</td>
<td>4A</td>
<td>J</td>
</tr>
<tr>
<td>0101 0001</td>
<td>75</td>
<td>4B</td>
<td>K</td>
</tr>
<tr>
<td>0101 0100</td>
<td>76</td>
<td>4C</td>
<td>L</td>
</tr>
<tr>
<td>0101 0101</td>
<td>77</td>
<td>4D</td>
<td>M</td>
</tr>
<tr>
<td>0101 1000</td>
<td>78</td>
<td>4E</td>
<td>N</td>
</tr>
<tr>
<td>0101 1001</td>
<td>79</td>
<td>4F</td>
<td>O</td>
</tr>
<tr>
<td>0101 1100</td>
<td>80</td>
<td>50</td>
<td>P</td>
</tr>
<tr>
<td>0101 1101</td>
<td>81</td>
<td>51</td>
<td>Q</td>
</tr>
<tr>
<td>0110 0000</td>
<td>96</td>
<td>60</td>
<td>`</td>
</tr>
<tr>
<td>0110 0001</td>
<td>97</td>
<td>61</td>
<td>a</td>
</tr>
<tr>
<td>0110 0100</td>
<td>98</td>
<td>62</td>
<td>b</td>
</tr>
<tr>
<td>0110 0101</td>
<td>99</td>
<td>63</td>
<td>c</td>
</tr>
<tr>
<td>0110 1000</td>
<td>100</td>
<td>64</td>
<td>d</td>
</tr>
<tr>
<td>0110 1001</td>
<td>101</td>
<td>65</td>
<td>e</td>
</tr>
<tr>
<td>0110 1010</td>
<td>102</td>
<td>66</td>
<td>f</td>
</tr>
<tr>
<td>0110 1011</td>
<td>103</td>
<td>67</td>
<td>g</td>
</tr>
<tr>
<td>0110 1100</td>
<td>104</td>
<td>68</td>
<td>h</td>
</tr>
<tr>
<td>0110 1101</td>
<td>105</td>
<td>69</td>
<td>i</td>
</tr>
<tr>
<td>0110 1110</td>
<td>106</td>
<td>6A</td>
<td>j</td>
</tr>
<tr>
<td>0110 1111</td>
<td>107</td>
<td>6B</td>
<td>k</td>
</tr>
<tr>
<td>0111 0000</td>
<td>108</td>
<td>6C</td>
<td>l</td>
</tr>
<tr>
<td>0111 0001</td>
<td>109</td>
<td>6D</td>
<td>m</td>
</tr>
<tr>
<td>0111 0100</td>
<td>110</td>
<td>6E</td>
<td>n</td>
</tr>
<tr>
<td>0111 0101</td>
<td>111</td>
<td>6F</td>
<td>o</td>
</tr>
<tr>
<td>0111 0110</td>
<td>112</td>
<td>70</td>
<td>p</td>
</tr>
<tr>
<td>0111 0111</td>
<td>113</td>
<td>71</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!
        return c
    else:
        raise Exception("Unknown character")

(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: encipher

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

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

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

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

model for this problem: transcribe from hw#2

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

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

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

```python
>>> decipher('Uifz xpsl ju pvu xjui b qfodjm!"
```

```python
>>> decipher('gv vw dtwvg')
```

But **how**?
Decipher?

All possible decipherings

Strategies?

Algorithms?
Measuring *Englishness*

**Very English-y**
- "Call me Ishmael."
- "Attack at dawn!"
- "rainbow, table, candle"
- "quadruplicity drinks procrastination"
- "the gostak distims the doses"
- "hension, framble, bardle"
- "jufict, stofwus, lictpub"
- "itehbs, rsnevtr, khbsota"
- "epadxo, nojarpn, gdxokpw"
- "h o q dedqBzdrzqrzkzc"

**Not English-y**

**Quantifying "Englishness"?**

**Higher scores**

**Lower scores**

All of these sound good to me!
Decipher?

All possible decipherings

Max!

Score them all

"Englishness" score ~ the # of vowels
### 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 silkv`
- `wl lm tjmlw`

#### Scores

- `[6.9e-05, 'gv vw dtwvg']`
- `[3.6e-05, 'hw wx euxwh']`
- `[1.4e-07, 'ix xy fvyxi']`
- `[8.8e-11, 'jy yz gwzyj']`
- `[7.2e-10, 'kz za hxazk']`
- `[0.01503, 'la ab iybal']`
- `[3.7e-08, 'mb bc jzcbm']`
- `[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 silkv']`
- `[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']`
- `[0.02060, 'ds st aqtqd']`
- `[0.45555, 'et tu brute']`
- `[0.00011, 'fu uv csvuf']`

### Strategies?

- **Algorithms?**
  
  \[ p(l_1 l_2 l_3) \]

- **“Englishness”** based on letter-probabilities

  \[ \max_x \rho(l_1 l_2 l_3) \]
Decipher?  

Strategies?  

Algorithms?

All possible decipherings

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

"Englishness" based on scrabble-scoring!

decPR(LAT)  
decPR2(LAT)  
decPR3(LAT)
Design...

The ♥ of CS (and CSers...)

Algorithms!
Design...

Code?
syntax

design of what?
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 easier to express, in an abstract way, what the software will do.
Design...

Code?

Algorithms!

design of what?
syntax
ideas!
Algorithm Design...

\[
\text{remAll}(e, L) \quad \text{remove all } e\text{'s from } L
\]
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', 'qaqqlqqiqqqiiiqeqqnsq')
.aliien.
Design...

Top-down design

- Visualize
- Split into parts, \( L[0] \) and \( L[1:] \)
- Build each part
- Combine
- Test

\[
\text{remAll}(e, L) \quad \text{remove all } e's \text{ from } L
\]

\[
\text{if } e \neq e \quad \text{Use it!}
\]

\[
\text{remAll}(42, [5, 7, 42, 8, 42])
\]

\[
[5, 7, 8]
\]

\[
\text{remAll}('q', 'qaqqlqqiqqiiiqeqqnsqqs')
\]

\[
'aliiiens'
\]
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('q', 'aqqlqqiqqiiqeqqnqs')

drop L[0]
+ remove e from the rest

'aliiiens'

Use it!

Lose it!

it

'the rest'

it

'the rest'
Design...

Top-down design

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

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

remAll('q', 'xagel')

Use it!
- or -

Lose it.

keep L[0]
+ remove e from the rest

drop L[0]
+ remove e from the rest
Allie Russell, ’12 speaking of roadside church signs...
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!?

```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(e, L)

remove all e's from L
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

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

```
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:])
```

That's it. Algorithmic expression ~ it's what CSers (think they) do.
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[1:0])
    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

Don't start yet...

Write the other cases needed for subseq...

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:])

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

Write the other cases needed for subseq...
remAll insight

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 `rem` examples...

\[
\begin{align*}
\text{remAll}(8, [7,8,9,8]) & \Rightarrow [7,9] \quad \text{remAll} \\
\text{remAll}('d', 'coded') & \Rightarrow 'coe' \quad \text{remAll}
\end{align*}
\]

\[
\begin{align*}
\text{remOne}(8, [7,8,9,8]) & \Rightarrow [7,9,8] \quad \text{remOne} \\
\text{remOne}('d', 'coded') & \Rightarrow 'coed' \quad \text{remOne}
\end{align*}
\]

\[
\begin{align*}
\text{remUpto}(8, [7,8,9,8]) & \Rightarrow [9,8] \quad \text{remUpto} \\
\text{remUpto}('d', 'coded') & \Rightarrow 'ed' \quad \text{remUpto}
\end{align*}
\]
**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
```

- `subseq('ctg', 'ctaga')` → True
- `subseq('ctg', 'tacggta')` → False
- `subseq('aliens', 'always frighten dragons')` → True
- `subseq('trogdor', 'that dragon is gone for good')` → False

Here there be NO dragons!

**Why** Are these True? or False?
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

# Challenge:
# Write the other cases needed for subseq...

# Problem:
# 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]

# Problem:
# 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'

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

# Hint:
# you'll need 3-4 cases total for subseq.
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

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

Try it out!

Challenge...

Write the other cases needed for subseq...

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. Challenge...

subseq('alg','magical')  
False

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:])
```

**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

**Challenge**

- `name(s):`
- `cs5 hrs last week`
from remAll to remOne

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

    if len(L) == 0:
        return L

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

    else:
        print("One")
        return remAll( e, L[1:] )

from remAll to remOne

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:]
```

```
from remOne to remUpto!
```

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

```python
Upto
def remUpto( e, L ):
    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'
```
def subseq( s, sbig ):
    ''' returns True if s is a subseq. of sbig;
    False otherwise. Both are strings. '''
    if s == '':
        return True
    elif sbig == '':
        return False
    elif s[0] == sbig[0]:
        return subseq(s[1:], sbig[1:])
    else:
        return subseq(s, sbig[1:])

subseq('alg','magical')  # False
subseq('alg','twasbrillig')  # True
I ❤️ NY
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
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?

Subseq ~ thinking it out...

```
subseq( s, sbig )
```

```
subseq('ctg', 'tacggta')
```

```
s[0]    sbig[0]
```

Use it!

- or -

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:])

remOne(8, [7, 8, 9, 8]) => [7, 9, 8]
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
    Quiz

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...

Pass those in and up...
Design ~ (*code*)

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

wrench!
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!
hw4pr2: use it or lose it

Longest Common Subsequence

Jotto Score counting

binary list and general list sorting

exact_change making
hw4pr2: use it or lose it

Longest Common Subsequence

LCS(S, T)

'HUMAN'

'CGCTGAGCTAGGCA...'

'CHIMPANZEE'

'ATCCTAGGTAACTG...'

+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 ?!?

<table>
<thead>
<tr>
<th>Ruler</th>
<th>Metridium</th>
<th>A. sulcata</th>
<th>Hematodinium</th>
<th>S. raphanus</th>
<th>N. virens</th>
<th>L. latreillii</th>
<th>Modiolus</th>
<th>S. solidissima</th>
<th>Pagurus</th>
<th>Emerita</th>
<th>Coelotes</th>
<th>F. heteroclitus</th>
<th>Chrysops</th>
<th>D. simulans</th>
<th>S. purpuratus</th>
<th>A. forbesi</th>
<th>G. rhodei</th>
<th>A. crucifera</th>
<th>M. portucalensis</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AATTACCAATCCACGCAGCTAGCTGACTGACAGAAGAAATAAACATACACAGGTCTCT</td>
<td>AATTACCAATCCACGCAGCTAGCTGACTGACAGAAGAAATAAACATACACAGGTCTCT</td>
<td>AATTACCAATCCACGCAGCTAGCTGACTGACAGAAGAAATAAACATACACAGGTCTCT</td>
<td>AATTACCAATCCACGCAGCTAGCTGACTGACAGAAGAAATAAACATACACAGGTCTCT</td>
<td>AATTACCAATCCACGCAGCTAGCTGACTGACAGAAGAAATAAACATACACAGGTCTCT</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Screenshot from the ClustalX multiple subsequence alignment tool...
What was gained (or lost) here?
What was gained (or lost) here?

Mourning species...?

Night-loving species!

caffeine cinema chocolate

Phylogeny

Hey!? Trinocular aliens
Subsequences @ 5Cs

host: figs

parasites: wasps

matching two phylogenies together
*Jane's source data:* 100s of species, 6 continents ...

*Largest co-phylogeny ever computed (in 2012)*
also in hw4pr2: *Jotto*!

a word-guessing game...

```
jscore( S, T )
```
also in hw4pr2: !

Let's try it!

These are two cute

'robot'  'otter'

jscore( 'robot', 'otter' ) →

jscore( S, T )  in general...
also in hw3pr2: \( \text{sort} + \text{exact\_change} \)

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

returns an ascending list

returns \text{True} or \text{False}

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

```
jscore(s1,s2)
```

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

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

```
sort( L )
```

```
sort( [42,5,7] ) \rightarrow [5,7,42]
sort( [42,7] ) \rightarrow [7,42]
sort( [42] ) \rightarrow [42]
sort( [ ] )
blsort( [1,0,1] )
```

```
LCS(S,T)
```

```
LCS('ctga', 'tagca') \rightarrow 'tga'
LCS('tga', 'taacg') \rightarrow 'ta' (or 'tg')
LCS('tga', 'a') \rightarrow 'a'
LCS('gattaca', 'ctctgctgat')
```

```
exact_change(t, L)
```

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

```
exact_change( 42, [25,30,2,5] ) \rightarrow False
exact_change( 42, [22,16,3,2,17] ) \rightarrow 
exact_change( 42, [18,21,22] ) \rightarrow 
exact_change( 42, [40,17,1,7] ) \rightarrow 
exact_change( 20, [16,3,2,17] ) \rightarrow 
```
should return the jotto score for any strings \( s_1 \) and \( s_2 \)

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

\[
\text{jscore}('robot', 'otter') \rightarrow 3
\]

\[
\text{jscore}('geese', 'seems') \rightarrow 3
\]

\[
\text{jscore}('fluff', 'lulls') \rightarrow 2
\]

\[
\text{jscore}('pears', 'diner') \rightarrow 2
\]

\[
\text{jscore}('xylyl', 'slyly') \rightarrow 4
\]

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

---

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

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

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

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

\[
\text{sort}([42]) \rightarrow [42]
\]

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

\[
\text{blsort}([1,0,1]) \rightarrow [0,1,1]
\]

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

---

should return the Longest Common Subsequence of strings \( S \) and \( T \)

\[
\text{LCS}(S,T)
\]

\[
\text{LCS}('ctga', 'tagca') \rightarrow 'tga'
\]

\[
\text{LCS}('tga', 'taacg') \rightarrow 'ta' (or 'tg')
\]

\[
\text{LCS}('tga', 'a') \rightarrow 'a'
\]

\[
\text{LCS}('gattaca', 'ctctgcgat') \rightarrow 'ttca'
\]

4 chars

---

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

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

\[
\text{exact_change}(42, [25,30,2,5]) \rightarrow False
\]

\[
\text{exact_change}(42, [22,16,3,2,17]) \rightarrow True
\]

\[
\text{exact_change}(42, [18,21,22]) \rightarrow False
\]

\[
\text{exact_change}(42, [40,17,1,7]) \rightarrow False
\]

\[
\text{exact_change}(20, [16,3,2,17]) \rightarrow True
\]

---

Brainstorm algorithms for these problems -- what helper functions???

Use it! Lose it! Use it! Lose it! Lose it! Lose it! Lose it! Lose it!

Use it! Lose it! Lose it! Lose it! Lose it! Lose it! Lose it! Lose it! Lose it! Lose it! Lose it!
should return the jotto score for any strings $s_1$ and $s_2$

$$\text{jscore}(s_1, s_2)$$

$\text{jscore}(\text{'robot'}, \text{'otter'}) \rightarrow 3$
$\text{jscore}(\text{'geese'}, \text{'seems'}) \rightarrow 3$
$\text{jscore}(\text{'fluff'}, \text{'lulls'}) \rightarrow 2$
$\text{jscore}(\text{'pears'}, \text{'diner'}) \rightarrow 2$
$\text{jscore}(\text{'xylyl'}, \text{'slyly'}) \rightarrow 4$

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

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

$$\text{sort}(L)$$

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

Should return the Longest Common Subsequence of strings $S$ and $T$

$$\text{LCS}(S, T)$$

$\text{LCS(}'ctga', \text{'tagca}') \rightarrow \text{'tga'}$
$\text{LCS(}'tga', \text{'taacg'}') \rightarrow \text{'ta' (or 'tg')}$
$\text{LCS(}'tga', \text{'a'}\) \rightarrow \text{'a'}$
$\text{LCS(}'gattaca', \text{'ctctgcgat'}\) \rightarrow \text{'ttca'}$

4 chars

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

$$\text{blsort}([1,0,1]) \rightarrow [0,1,1]$$

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

$$\text{exact_change}(t, L)$$

| $\text{exact_change}(42, [25,30,2,5]) \rightarrow$ | False |
| $\text{exact_change}(42, [22,16,3,2,17]) \rightarrow$ | True |
| $\text{exact_change}(42, [18,21,22]) \rightarrow$ | False |
| $\text{exact_change}(42, [40,17,1,7]) \rightarrow$ | False |
| $\text{exact_change}(20, [16,3,2,17]) \rightarrow$ | True |

... and here
decipher( 'Weet bksa ed xecumeha 4!' )

kxn rkfo k qbokd goouoxn ...
Good luck on homework 4!

and have a great weekend ...