Computing to the **max**

The not-so-subtle art of singling out the best (and worst) of anything...

> a comparison **comparison**

- `'m+ms'` ___ `'coffee'`
- `[0, 42]` ___ `[4, 2]`
- `[0, 'm+ms']` ___ `[4, 'coffee']`

Computing with **language**

- *What's in a Writ1 paper, anyway?*
- Battle-tested ciphers & how to break them...

---

Last hw?

**N-step** sleepwalking?

Turtle graphics??

Artistic renderings!!

---

This week!

*HW #4 due next Monday...*

- **pr0**: Are we *The Matrix*?
- **pr1**: Lab: *sounds good*...
- **pr2**: Sorting + Caesar!
- **ex cr**: Add'I UIOLI fun'!
hw3pr3: areas from rectangles

Areas of 4 rectangles

Areas of 8 rectangles
Area of N rectangles \textit{in the limit}
hw3pr3: Maya Lin, Architect...
Maya Lin, Artist and Computer Scientist...

"two-by-four landscape"
Maya Lin, *Artist and Computer Scientist*...

One building block, deliberately applied, *over 50,000 times*...
Building blocks == CS!

\[ y = 2x \]

\[
\text{scaledfracs}(\text{low}, \text{hi}, \text{N})
\]

\[
\text{f_of_fracs}(f, \text{low}, \text{hi}, \text{N})
\]

\[
\text{integrate}(f, \text{low}, \text{hi}, \text{N})
\]

Where are the LCs?

Areas of 8 rectangles

only a few lines...
max

A recipe for life?
The real problem is knowing what we want to maximize!

max

A recipe for life?
and python already has it for us...
max

A recipe for life?
and python already has it for us...

The real problem is knowing *what*
we want to maximize!

... or *minimize*, with *min*
to the max

Want the highest price?

```
max( [475.5, 458.0, 441.3, 470.8, 532.8, 520.9] )
```

What if the months are in there, as well?

```
max( [ [470.8,'may'], [532.8,'jul'], [520.9,'sep'] ] )
```

And I thought $54 was overpriced!

NASDAQ: GOOG
Want the highest price?

\[
\max( [475.5, 458.0, 441.3, 470.8, 532.8, 520.9] )
\]

What if the months are in there, as well?

\[
\max( [ [470.8,'may'], [532.8,'jul'], [520.9,'sep'] ] )
\]
\[
\max( [ ['may',470.8], ['jul',532.8], ['sep',520.9] ] )
\]

\textbf{Mudd's max?}

\[
L = ['Harvey', 'Mudd', 'College', 'seeks', 'to', 'educate', 'engineers', 'scientists', 'and', 'mathematicians', 'well-versed', 'in', 'all', 'of', 'these', 'areas', 'and', 'in', 'the', 'humanities', 'and', 'the', 'social', 'sciences', 'so', 'that', 'they', 'may', 'assume', 'leadership', 'in', 'their', 'fields', 'with', 'a', 'clear', 'understanding', 'of', 'the', 'impact', 'of', 'their', 'work', 'on', 'society']
\]

\textbf{Or Mudd's min?}

\[
\min(MSt)
\]

\[
\max(MSt)
\]
def max( L ):
    """ returns the max element from L
    input:  L, a nonempty list
    """
    if len(L) < 2:  return L[0]  # only 1 elem.
    maxOfRest = max(L[1:])  # max of the rest
    if L[0] > maxOfRest:
        return L[0]  # either L[0]
    else:
        return maxOfRest  # or maxOfRest!
def maxSS(L):
    """ returns L's highest scrabble-scoring element (input: L, a nonempty list) """
    if len(L) < 2:  return L[0]  # only 1 elem.

    maxOfRest = maxSS(L[1:])  # rest's max

    if L[0] > maxOfRest :  
        return L[0]  # either L[0] 
    else: 
        return maxOfRest  # or maxOfRest!
def maxSS(L):
    """ returns L's highest scrabble-scoring element (input: L, a nonempty list) """
    if len(L) < 2:
        return L[0] # only 1 elem.
    maxOfRest = maxSS(L[1:]) # rest's max

    if sScore(L[0]) > sScore(maxOfRest):
        return L[0] # either L[0]
    else:
        return maxOfRest # or maxOfRest!
def maxSS(L):
    """ returns L's highest scrabble-scoring element (input: L, a nonempty list) """
    if len(L) < 2:
        return L[0] # only 1 elem.
    maxOfRest = maxSS(L[1:]) # rest's max
    if sScore(L[0]) > sScore(maxOfRest):
        return L[0] # either L[0]
    else:
        return maxOfRest # or maxOfRest!
A more comprehensive solution: \texttt{LoL}

\begin{verbatim}
L = [ 'aliens', 'zap', 'hazy', 'code' ]

def maxSS( L ):
    """ returns L's max-scrabble-score word """
    LoL = [ [sScore(w), w] for w in L ]
    bestpair = max( LoL )
    return bestpair[1]
\end{verbatim}

This does look funny!
A more comprehensive solution

```python
L = [ 'aliens', 'zap', 'hazy', 'code' ]

def maxSS( L ):
    """ returns L's max-scrabble-score word """
    LoL = [ [sScore(w), w] for w in L ]

    bestpair = max( LoL )

    return bestpair[1]
```

This does look funny!
A more comprehensive solution

def maxSS(L):
    """ returns L's max scrabble score word """
    LoL = [ [sScore(w), w] for w in L ]
    bestpair = max(LoL)
    return bestpair[1]

L = ['aliens', 'zap', 'hazy', 'code']

This does look funny!
A more comprehensive solution

def maxSS(L):
    """ returns L's max scrabble score word """
    LoL = [ [sScore(w), w] for w in L ]
    bestpair = max(LoL)
    return bestpair[1]

This does look funny!

L = ['aliens', 'zap', 'hazy', 'code']
def maxSS(L):
    """ returns L's max-scrabble-score word """
    LoL = [ [sScore(w), w] for w in L ]
    bestpair = max(LoL)
    return bestpair[1]

I loathe hazy code!
def lastrest(L):
    """ another example – what's returned? """
    LoL = [ [w[1:], w] for w in L ]
    bestpair = max(LoL)
    return bestpair[1]
Everything ... is a max problem?

```
def lastrest(L):
    """ another example - what's returned? """
    LoL = [ [w[1:], w] for w in L ]
    bestpair = max( LoL )
    return bestpair[1]
```

```
L = ['aliens', 'zap', 'hazy', 'code']
I know the best word here... but does Python?
```
Everything ... is a max problem?

```python
L = ['aliens', 'zap', 'hazy', 'code']

def lastrest(L):
    """
    another example - what's returned?
    """
    LoL = [ [w[1:], w] for w in L ]
    bestpair = max(LoL)
    return bestpair[1]

I know the best word here... but does Python?
```
def lastrevved( L ):
    """ another example - what's returned? """

    LoL = [ [w[::-1], w] for w in L ]

    bestpair = max( LoL )

    return bestpair[1]
Everything ... is a max problem?

```python
def lastrevved(L):
    """ another example - what's returned? """
    LoL = [ [w[::-1], w] for w in L ]
    bestpair = max(LoL)
    return bestpair[1]
```

L = ['aliens', 'zap', 'hazy', 'code']

I know the best word here... but does Python?

L = ['aliens', 'zap', 'hazy', 'code']

LoL = [['sneila', 'aliens'], ['paz', 'zap'], ['yzah', 'hazy'], ['edoc', 'code']]

bestpair = max(LoL)

bestpair = ['yzah', 'hazy']

return bestpair[1]  # 'hazy'
Other examples...

What is `bestnumb`? What is `mostnumb`?

```python
>>> bestnumb([10, 20, 30, 40, 50, 60, 70])
40
```

```python
>>> bestnumb([100, 200, 300, 400])
100
```

```python
>>> bestnumb([1, 2, 3, 4, 5, 6, 7, 8, 7])
8
```

```python
>>> mostnumb([1, 2, 3, 4, 5, 6, 7, 8, 7])
7
```
**Quiz**

```python
def maxlen(L):
    LoL = [ [len(s), s] for s in L ]
    bstpr = max(LoL)
    return bstpr[1]
```

1. What is LoL?
   - Here is a start: `LoL = [ [6,'aliens'], [3,'zap'] ]`
2. What is bstpr?
   - `['aliens']`
3. What is returned?
   - `'aliens'` (corrected)

---

```python
def bestnumb(L):
    """ returns the # in L closest to 42 ""
    LoL = [ abs(42-x) for x in L ]
    bstpr = min(LoL)
    return bstpr[1]
```

Hint: Python has `abs(x)` built-in

Extra!

```python
def mostnumb(L):
    """ returns the item most often in L ""
    LoL = [ Count(s,L) ≤ 3 for s in L ]
    bstpr = max(LoL)
    return bstpr[1]
```

Hint: Use this helper function!

```python
def count(e,L):
    """ return # of e's in L ""
    LC = [ 1 for x in L if x == e ]
    return sum(LC)
```

Extra!

Change exactly three characters in this code so that 3 is returned.

L = ['aliens', 'zap', 'hazy', 'code']

Use the LoL method to write these two functions

L = [30, 40, 50]

Name(s)
Quiz

L = ['aliens', 'zap', 'hazy', 'code']

```python
def maxlen(L):
    LoL = [ [len(s), s] for s in L ]
    bstpr = max(LoL)
    return bstpr[1]
```

1. What is LoL?  
   ```python
   ```

2. What is bstpr?  
   ```python
   [6, 'aliens']
   ```

3. What is returned?  
   ```python
   'aliens'
   ```

L = [30, 40, 50]

```python
def bestnumb(L):
    """ returns the # in L closest to 42 """
    LoL = [ [abs(x-42), x] for x in L ]
    bstpr = min(LoL)
    return bstpr[1]
```

```
L = [30, 40, 50]
```

```python
def mostnumb(L):
    """ returns the item most often in L """
    LoL = [ [count(e,L), e] for e in L ]
    bstpr = max(LoL)
    return bstpr[1]
```

L = [3, 4, 5, 6, 6, 7, 7]

```
L = [3, 4, 5, 6, 6, 7, 7]
```

```python
def count(e,L):
    """ return # of e's in L """
    LC = [ 1 for x in L if x == e ]
    return sum(LC)
```

```
L = [30, 40, 50]
```

```
L = [3, 4, 5, 6, 6, 7, 7]
```

```
```

```
L = [30, 40, 50]
```

```
L = [3, 4, 5, 6, 6, 7, 7]
```

```
L = [3, 4, 5, 6, 6, 7, 7]
```

```
L = [3, 4, 5, 6, 6, 7, 7]
```
def maxlen(L):
    LoL = [[len(s), s] for s in L]
    bstpr = max(LoL)
    return bstpr[1]

L = [ 'aliens', 'zap', 'hazy', 'code' ]


bstpr = max( LoL )

2. What is bstpr?  [6, 'aliens']

return bstpr[1]

3. What is returned?  'aliens'

Extra!  Change exactly three characters in this code so that 3 is returned.
def bestnumb(L):
    """ returns the # closest to 42 in L """

    LoL = [ [abs(x-42),x] for x in L ]

    bstpr = min( LoL )

    return bstpr[1]
def count( e, L ):
    """ returns the # of e's in L """
    LC = [ 1 for x in L if x==e ]
    return sum( LC )

def mostnumb( L ):
    """ returns the item most often in L """
    LoL = [ [count(e,L),e] for e in L ]
    bstpr = max( LoL )
    return bstpr[1]

Could you use x here instead of e?
```python
def count(e, L):
    '''returns the # of e's in L '''
    LC = [1 for x in L if x==e]
    return sum(LC)

def mostnumb(L):
    ''' returns the item most often in L '''
    LoL = [[count(e,L), e] for e in L]
    bstpr = max(LoL)
    return bstpr[1]
```

Could you use x here instead of e?
Today's lab: *big data*?

Any guesses as to what *kind* of data this is?

I find your lack of faith in this data disturbing.
Today's lab: *sound* data!

what are the vertical and horizontal axes here?
Lab3 ~ Sound

Continuous variation of air pressure vs. time

Samples taken every 1/22050th of a second (or some sampling rate)

Each sample is measured on a loudness scale from -32,768 to 32,767. (This fits into 2 bytes.)

These two bytes are called a frame. Raw audio data - such as what is written to the surface of a CD - is simply a list of these frames.

Play('swnotry.wav')  # run demo()
flipflop('swnotry.wav')
play('swfaith.wav')
changeSpeed('swfaith.wav', 44100)
reverse('swfaith.wav')
play('spam.wav')
reverse('spam.wav')
Lab 4's key challenge...

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 – not important

important stuff

"outro" stuff
Computing with *language*

- ideas / meaning
- language / words / phrases
- strings
- numbers / bits

Python strings are here.
"alphabetic processions"
Computing with *language*

- ideas / meaning
- language / words / phrases
- strings
- numbers / bits

This week...

processing language – *how English-y is it?*

open questions in AI ...

Eliza, Siri, Tay ... trouble?

how strings are represented and stored

Next week...
Caesar Cipher: \texttt{encipher}

\texttt{encipher(s, n)}

\begin{itemize}
\item \texttt{encipher( 'I <3 Latin', 0 )} \quad \Rightarrow \quad \texttt{'I <3 Latin'}
\item \texttt{encipher( 'I <3 Latin', 1 )} \quad \Rightarrow \quad \texttt{'J <3 Mbujo'}
\item \texttt{encipher( 'I <3 Latin', 2 )} \quad \Rightarrow \quad \texttt{'K <3 Ncvkp'}
\item \texttt{encipher( 'I <3 Latin', 3 )} \quad \Rightarrow \quad \texttt{'L <3 Odwlq'}
\item \texttt{encipher( 'I <3 Latin', 4 )} \quad \Rightarrow \quad \texttt{'M <3 Pexmr'}
\item \texttt{encipher( 'I <3 Latin', 5 )} \quad \Rightarrow \quad \texttt{'N <3 Qfyns'}
\item \ldots
\item \texttt{encipher( 'I <3 Latin', 25 )} \quad \Rightarrow \quad \texttt{'H <3 Kzshm'}
\end{itemize}
Caesar Cipher: \texttt{encipher}

\begin{itemize}
  \item \texttt{encipher(s,n)} \textit{should return the string} \texttt{s} \textit{with each alphabetic character shifted/wrapped by} \texttt{n} \textit{places in the alphabet}
  \item \texttt{encipher( 'I \textless 3 Latin', 0 )} \quad \text{returns} \quad \textquote{I \textless 3 Latin'}
  \item \texttt{encipher( 'I \textless 3 Latin', 1 )} \quad \text{returns} \quad \textquote{J \textless 3 Mbujo'}
  \item \texttt{encipher( 'I \textless 3 Latin', 2 )} \quad \text{returns} \quad \textquote{K \textless 3 Ncvkp'}
  \item \texttt{encipher( 'I \textless 3 Latin', 3 )} \quad \text{returns} \quad \textquote{L \textless 3 Odwlq'}
  \item \texttt{encipher( 'I \textless 3 Latin', 4 )} \quad \text{returns} \quad \textquote{M \textless 3 Pexmr'}
  \item \texttt{encipher( 'I \textless 3 Latin', 5 )} \quad \text{returns} \quad \textquote{N \textless 3 Qfyns'}
  \item \hspace{1cm} \ldots
  \item \hspace{1cm} \ldots
  \item \texttt{encipher( 'I \textless 3 Latin', 25 )} \quad \text{returns} \quad \textquote{H \textless 3 Kzshm'}
\end{itemize}
The SAME bits represent an integer or a string, depending on type: `int` or `str`
Unicode

Universal Character Encoding

Some fun characters...

chr(156265)  chr(9835)  chr(9731)

u"I <3 ‮ Wow! ‭ Unicode..."

My favorite is chr(1661)

on Win10: chcp 65001
The SAME bits represent an integer or a string, depending on type: \texttt{int} or \texttt{str}

<table>
<thead>
<tr>
<th>Binary</th>
<th>Dec</th>
<th>Hex</th>
<th>Glyph</th>
</tr>
</thead>
<tbody>
<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>

1 byte
8 bits

Identical bits are stored in each variable!

The types determine how to interpret the bits; the names don't matter at all...
ASCII and Unicode

<table>
<thead>
<tr>
<th>chr</th>
<th>ord</th>
<th>convert char to #</th>
</tr>
</thead>
</table>

ASCII Table:

<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>

Unicode 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>
<tr>
<td>0101 0010</td>
<td>82</td>
<td>52</td>
<td>R</td>
</tr>
<tr>
<td>0101 0011</td>
<td>83</td>
<td>53</td>
<td>S</td>
</tr>
<tr>
<td>0101 0100</td>
<td>84</td>
<td>54</td>
<td>T</td>
</tr>
<tr>
<td>0101 0101</td>
<td>85</td>
<td>55</td>
<td>U</td>
</tr>
<tr>
<td>0101 0110</td>
<td>86</td>
<td>56</td>
<td>V</td>
</tr>
<tr>
<td>0101 0111</td>
<td>87</td>
<td>57</td>
<td>W</td>
</tr>
<tr>
<td>0101 1000</td>
<td>88</td>
<td>58</td>
<td>X</td>
</tr>
<tr>
<td>0101 1001</td>
<td>89</td>
<td>59</td>
<td>Y</td>
</tr>
<tr>
<td>0101 1010</td>
<td>90</td>
<td>60</td>
<td>Z</td>
</tr>
<tr>
<td>0101 1011</td>
<td>91</td>
<td>61</td>
<td>a</td>
</tr>
<tr>
<td>0101 1100</td>
<td>92</td>
<td>62</td>
<td>b</td>
</tr>
<tr>
<td>0101 1101</td>
<td>93</td>
<td>63</td>
<td>c</td>
</tr>
<tr>
<td>0101 1110</td>
<td>94</td>
<td>64</td>
<td>d</td>
</tr>
<tr>
<td>0101 1111</td>
<td>95</td>
<td>65</td>
<td>e</td>
</tr>
<tr>
<td>0110 0000</td>
<td>96</td>
<td>66</td>
<td>f</td>
</tr>
<tr>
<td>0110 0001</td>
<td>97</td>
<td>67</td>
<td>g</td>
</tr>
<tr>
<td>0110 0010</td>
<td>98</td>
<td>68</td>
<td>h</td>
</tr>
<tr>
<td>0110 0011</td>
<td>99</td>
<td>69</td>
<td>i</td>
</tr>
<tr>
<td>0110 0100</td>
<td>100</td>
<td>6A</td>
<td>j</td>
</tr>
<tr>
<td>0110 0101</td>
<td>101</td>
<td>6B</td>
<td>k</td>
</tr>
<tr>
<td>0110 0110</td>
<td>102</td>
<td>6C</td>
<td>l</td>
</tr>
<tr>
<td>0110 0111</td>
<td>103</td>
<td>6D</td>
<td>m</td>
</tr>
<tr>
<td>0110 1000</td>
<td>104</td>
<td>6E</td>
<td>n</td>
</tr>
<tr>
<td>0110 1001</td>
<td>105</td>
<td>6F</td>
<td>o</td>
</tr>
<tr>
<td>0110 1010</td>
<td>106</td>
<td>70</td>
<td>p</td>
</tr>
<tr>
<td>0110 1011</td>
<td>107</td>
<td>71</td>
<td>q</td>
</tr>
<tr>
<td>0110 1100</td>
<td>108</td>
<td>72</td>
<td>r</td>
</tr>
<tr>
<td>0110 1101</td>
<td>109</td>
<td>73</td>
<td>s</td>
</tr>
<tr>
<td>0110 1110</td>
<td>110</td>
<td>74</td>
<td>t</td>
</tr>
<tr>
<td>0110 1111</td>
<td>111</td>
<td>75</td>
<td>u</td>
</tr>
<tr>
<td>0111 0000</td>
<td>112</td>
<td>76</td>
<td>v</td>
</tr>
<tr>
<td>0111 0001</td>
<td>113</td>
<td>77</td>
<td>w</td>
</tr>
<tr>
<td>0111 0010</td>
<td>114</td>
<td>78</td>
<td>x</td>
</tr>
<tr>
<td>0111 0011</td>
<td>115</td>
<td>79</td>
<td>y</td>
</tr>
<tr>
<td>0111 0100</td>
<td>116</td>
<td>7A</td>
<td>z</td>
</tr>
<tr>
<td>0111 0101</td>
<td>117</td>
<td>7B</td>
<td></td>
</tr>
<tr>
<td>0111 0110</td>
<td>118</td>
<td>7C</td>
<td>_</td>
</tr>
<tr>
<td>0111 0111</td>
<td>119</td>
<td>7D</td>
<td>`</td>
</tr>
<tr>
<td>0111 1000</td>
<td>120</td>
<td>7E</td>
<td></td>
</tr>
<tr>
<td>0111 1001</td>
<td>121</td>
<td>7F</td>
<td>\</td>
</tr>
</tbody>
</table>

This is why 'CS' < 'clear'!
### ASCII and Unicode

#### Convert # to char

```python
[ [i, chr(i)] for i in range(128) ]
```

#### Convert char to #

```python
[ ord(i) for i in '**** CS! ****' ]
```

---

*This is why 'CS' < 'clear'!*

---

Julius spr'15
Rot13

rot13('a') should output 'n'
rot13('M') should output 'Z'
rot13('n') should output 'a'
rot13('W') should output 'J'
rot13(' ') should output ' '
rot13('<') should output '<'

A useful and illustrative starting point...
### ASCII and Unicode

<table>
<thead>
<tr>
<th>chr value</th>
<th>ord value</th>
</tr>
</thead>
<tbody>
<tr>
<td>abcdefghijklmnopqrstuvwxyz</td>
<td>97 99 101 103 105 107 109 111 113 115 117 119 122</td>
</tr>
<tr>
<td>ABCDEFGHIJKLMNOPQRSTUVWXYZ</td>
<td>65 67 69 71 73 75 77 79 81 83 85 87 90</td>
</tr>
</tbody>
</table>

**What is `ord('U') // 2`?**

**What is `chr(ord('i')+13)`?**

**What is `chr(ord('W')+13)`?**

**how do we wrap?**
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+13)
    elif 'A' <= c <= 'Z':
        # upper-case test!
        return chr(ord(c)+13)
    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?
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 using '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?
rot13's surprising history...
Caesar Cipher: `encipher`

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

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

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

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

```python
>>> encipher('Hu lkbjhapvu pz doha ylthpuz hmaly dl mvylna \
       '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."

>>> decipher('Hu lkbjhapvu pz doha ylthpuz hmaly dl mvynla \n  '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')
```

Which is more difficult computationally?
Decipher?

Strategies?

*Algorithms*?
Decipher?

Strategies?

Algorithms?

All possible decipherings
Decipher?

All possible decipherings

Strategies?

Algorithms?

quantifying Englishness?

Score them all
Decipher?

All possible decipherings

Max!

Score them all

yields the "most English" phrase

Strategies?

Algorithms?
Measuring *Englishness*

**Very English-y**
- higher scores
- "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 doses"
- "hension, framble, bardle"
- "jufict, stofwus, lictpub"
- "itehbs, rsnevtr, khbsota"
- "epadxo, nojarpn, gdxokpw"
- "h o q dedqBzdrzqzrkzc"

**Not English-y**
- lower scores
- "All of these sound good to me!"
Decipher?

Strategies?

Algorithms?

All possible decipherings

Score them all

"Englishness" score based on # of vowels

max!

\[[4, \text{'la ab iybal']},
\[0, \text{''mb bc jzcbm'']},
\[1, \text{''nc cd kadcn'']},
\[4, \text{''od de lbedo'']},
\[3, \text{''pe ef mcfep'']},
\[0, \text{''qf fg ndgfq'']},
\[2, \text{''rg gh oehgr'']},
\[2, \text{''sh hi pfihs'']},
\[3, \text{''ti ij qgjit'']},
\[2, \text{''uj jk rhkju'']},
\[1, \text{''vk kl slikv'']},
\[0, \text{''wl lm tjmlw'']},
\[xm mn uknmx''],
\[yn no vlony''],
\[zo op wmpoz''],
\[ap pq xnqpa''],
\[bq qr yorqb''],
\[cr rs zpsrc''],
\[ds st aqtsd''],
\[et tu brute''],
\[fu uv csvuf'']

Decipher?

Strategies?

Algorithms?

"Englishness" based on letter-probabilities
Earbuds are helpful for lab - unless you really like Darth Vader!

We'll see you in Lab4!