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

Computing to the max

The real problem is knowing what we want to maximize!

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’l UIOLI fun’!

Computing with language

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

Building blocks == CS!

scaledfracs(low, hi, N)
f_of_fracs(f, low, hi, N)
integrate(f, low, hi, N)

only a few lines...

Where are the LCs?

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

max
to the max

Want the highest price?

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

What if the months are in there, as well?

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

A more comprehensive solution: \textbf{LoL}

\[
\text{LoL} = [ [\text{sScore}(w), w] \text{ for } w \text{ in } L ]
\]

\[
\text{bestpair} = \text{max}( \text{LoL} )
\]

\[
\text{return bestpair[1]}
\]

**max with scrabble-score**

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

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

A more comprehensive solution

\[
\text{LoL} = [ [6, \text{'aliens'}], [14, \text{'zap'}], [19, \text{'hazy'}], [7, \text{'code'}] ]
\]

\[
\text{LoL} = [ [\text{sScore}(w), w] \text{ for } w \text{ in } L ]
\]

\[
\text{LOL} = [ [6, \text{'aliens'}], [14, \text{'zap'}], [19, \text{'hazy'}], [7, \text{'code'}] ]
\]

\[
\text{bestpair} = \text{max}( \text{LoL} )
\]

\[
\text{return bestpair[1]}
\]

\[
\text{'hazy'}
\]
**Everything** ... is a max problem?

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

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

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

LoL = [ [6,'aliens'], [3,'zap'], [4,'hazy'], [4,'code'] ]

```
bestpair = max( LoL )
bestpair =

return bestpair[1]
```

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

def maxlen( L ):
    LoL = [ [len(s),s] for s in L ]
    LoL = [ [   , 'aliens'], [   , 'zap'], [   , 'hazy'], [   , 'code'] ]
    bstpr = max( LoL )  
    return bstpr[1]
```

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

def bestnumb( L ):
    """ returns the # in L closest to 42 """
    LoL = [                       ]
    bstpr =
    return bstpr[1]
```

```
L = [ 1,2,3,4,5,6,7,8,7 ]

def mostnumb( L ):
    """ returns the item most often in L """
    LoL = [                       ]
    bstpr =
    return bstpr[1]
```

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

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

def maxlen( L ):
    LoL = [ [len(s),s] for s in L ]
    LoL = [ [   , 'aliens'], [   , 'zap'], [   , 'hazy'], [   , 'code'] ]
    bstpr = max( LoL )
    return bstpr[1]
```

```
L = [ 1,2,3,4,5,6,7,8,7 ]

def bestnumb( L ):
    """ returns the # in L closest to 42 """
    LoL = [ [   , 'aliens'], [   , 'zap'], [   , 'hazy'], [   , 'code'] ]
    bstpr =
    return bstpr[1]
```

```
L = [ 1,2,3,4,5,6,7,8,7 ]

def mostnumb( L ):
    """ returns the item most often in L """
    LoL = [                       ]
    bstpr =
    return bstpr[1]
```

**Other examples...**

<table>
<thead>
<tr>
<th>What is bestnumb?</th>
<th>What is mostnumb?</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;&gt;&gt; bestnumb( [10,20,30,40,50,60,70] )</td>
<td>40</td>
</tr>
<tr>
<td>&gt;&gt;&gt; bestnumb( [100,200,300,400] )</td>
<td>100</td>
</tr>
<tr>
<td>&gt;&gt;&gt; bestnumb( [1,2,3,4,5,6,7,8,7] )</td>
<td>8</td>
</tr>
<tr>
<td>&gt;&gt;&gt; mostnumb( [1,2,3,4,5,6,7,8,7] )</td>
<td>7</td>
</tr>
</tbody>
</table>

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

def bestnumb( L ):
    """ returns the # in L closest to 42 """
    LoL = [ [   , 'aliens'], [   , 'zap'], [   , 'hazy'], [   , 'code'] ]
    bstpr =
    return bstpr[1]
```

```
L = [ 1,2,3,4,5,6,7,8,7 ]

def mostnumb( L ):
    """ returns the item most often in L """
    LoL = [                       ]
    bstpr =
    return bstpr[1]
```

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

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

def bestnumb( L ):
    """ returns the # in L closest to 42 """
    LoL = [                       ]
    bstpr =
    return bstpr[1]
```

```
L = [ 1,2,3,4,5,6,7,8,7 ]

def mostnumb( L ):
    """ returns the item most often in L """
    LoL = [                       ]
    bstpr =
    return bstpr[1]
```
**bestnumb**

```python
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]
```

**mostnumb**

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

```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]
```

---

**Today's lab: big data?**

![Graph showing stock price data]

Looks like stock prices, but much wavier!

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

---

**Today's lab: sound data!**

![Graph showing audio data]

What are the vertical and horizontal axes here?
Lab 3 ~ Sound

- Physics: continuous variation of air pressure vs. time
- Sampling: samples taken every 1/22050th of a second (or some sampling rate)
- Quantization: Each sample is measured on a loudness scale from -32,768 to 32,767. (This fits into 2 bytes.)
- Storage: 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.
- Pulse code modulation = PCM data

Lab 4's key challenge...

- Intro stuff – not important
- Important stuff
- "Outro" stuff

Computing with language

- Ideas / meaning
- Language / words / phrases
- Strings: Python strings are here. "alphabetic processions"
- Numbers / bits

Caesar Cipher: encipher

```python
encipher(s, n)
```

What is it doing?

```plaintext
cipher( 'I <3 Latin' , 0 ) → 'I <3 Latin'
cipher( 'I <3 Latin' , 1 ) → 'J <3 Mbujo'
cipher( 'I <3 Latin' , 2 ) → 'K <3 Ncvkp'
cipher( 'I <3 Latin' , 3 ) → 'L <3 Odwlq'
cipher( 'I <3 Latin' , 4 ) → 'M <3 Pexmr'
cipher( 'I <3 Latin' , 5 ) → 'N <3 Qfyns'
...```

```plaintext
cipher( 'I <3 Latin' , 25 ) → 'H <3 Kzshm'
```
ASCII
American Standard Code for Information Interchange

<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

The SAME bits represent an integer or a string, depending on type: `int` or `str`

convert # to char
`chr`

ord
convert char to #

ASCII or Unicode...

<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

The SAME bits represent an integer or a string, depending on type: `int` or `str`

over in memory...

ASCII and Unicode

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

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

**ASCII and Unicode**

**Writing Rot13**

```python
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))
    elif 'A' <= c <= 'Z':  # upper-case test!
        return chr(c)
    else:
        return c
```

**Caesar Cipher: encipher**

```python
>>> encipher('Bzdrzq bhogdq? H oqdedq Bzdrzq rzkzc.',25)
'Aycqyp agnfcp? G npdcnp Aycqyp qyjyb.'
>>> encipher('Bzdrzq bhogdq? H oqdedq Bzdrzq rzkzc.',15)
'Qosgof qwvvsf? W dfstsf Qosgof gozor.'
>>> encipher('Bzdrzq bhogdq? H oqdedq Bzdrzq rzkzc.',4)
'Fdhvdu flskhu? L suhihu Fdhvdu vdogd.'
>>> encipher('Bzdrzq bhogdq? H oqdedq Bzdrzq rzkzc.',1)
'Caesar cipher? I prefer Caesar salad.'
```

"An education is what remains after we forget everything we have learned."
Caesar Cipher: `decipher`

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

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

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

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

Decipher?

Strategies?

Algorithms?

Measuring Englishness

 duasfjd,

higher scores

quantifying Englishness?

lower scores

"Call me Ishmael."  "Attack at dawn!"

"rainbow, table, candle"

"Yow! Legally-imposed CULTURE-reduction is CABBAGE-BRAINED!"

"quadruplicity drinks procrastination"

"Hold the new reader’s nose squarely, waiter, or friendly milk will countermand my trousers."

"the gostak distms the doses"

"hension, framble, bardle"

"jufict, stofwus, lietpub"

"itehbs, rsnevt, khsota"

"epadxo, nojarpn, gdxokpw"

"h o q dedqBzdrzq rzkc"
We'll see you in Lab4!

Earbuds are helpful for lab - unless you *really* like Darth Vader!
def **maxlen**(*L*):
    LoL = [ [len(s), s] for s in *L* ]
    bstpr = max(LoL)
    return bstpr[1]

def **bestnumb**(*L*):
    """returns the # in *L* closest to 42 """
    LoL = [  ]
    bstpr =
    return bstpr[1]

def **mostnumb**(*L*):
    """returns the item most often in *L"""
    LoL = [  ]
    bstpr =
    return bstpr[1]

---

Extra!

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

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

**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)
```