`max()`, `min()`, and Lists of Lists; ASCII Codes and the Caesar Cipher

Computer Science 111
Boston University

Vahid Azadeh-Ranjbar, Ph.D.

based in part on notes from the CS-for-All curriculum developed at Harvey Mudd College

`max()` and `min()`

- `max(values)`: returns the largest value in a list of values

```
>>> max([4, 10, 2])
10
>>> max(['all', 'students', 'love', 'recursion'])
'students'
```

- `min(values)`: returns the smallest value in a list of values

```
>>> min([4, 10, 2])
2
>>> min(['all', 'students', 'love', 'recursion'])
'all'
```

Lists of Lists

- Recall that the elements of a list can themselves be lists:

```
[[124, 'Jaws'], [150, 'Lincoln'], [115, 'E.T.']]
```

- When you apply `max()`/`min()` to a list of lists, the comparisons are based on the ***first*** element of each sublist:

```
>>> max([[124, 'Jaws'], [150, 'Lincoln'], [115, 'E.T.']])
[150, 'Lincoln']

>>> min([[124, 'Jaws'], [150, 'Lincoln'], [115, 'E.T.']])
[115, 'E.T.']
```

Finding a Maximum Stock Price

```
>>> max([578.7, 596.0, 586.9])
          'jun'     'jul'     'aug'
596.0
```

- To determine the month in which the max occurred, use a *list of lists*!

```
>>> max([[578.7,'jun'], [596.0,'jul'], [586.9,'aug']])
[596.0, 'jul']

>>> max([['jun',578.7], ['jul',596.0], ['aug',586.9]])
???
```

Finding a Maximum Stock Price

```
>>> max([578.7, 596.0, 586.9])
          'jun'     'jul'     'aug'
596.0
```

- To determine the month in which the max occurred, use a *list of lists*!

```
>>> max([[578.7,'jun'], [596.0,'jul'], [586.9,'aug']])
[596.0, 'jul']
>>> max([['jun',578.7], ['jul',596.0], ['aug',586.9]])
['jun', 578.7]        # not what we want!
```

Problem Solving Using LCs and Lists of Lists

- Sample problem: finding the ***shortest*** word in a list of words.

```
words = ['always', 'come', 'to', 'class']
```

1. Use a list comprehension to build a list of lists:

```
scored_words = [[len(w), w] for w in words]
# for the above words, we get:
# [[6, 'always'], [4, 'come'], [2, 'to'], [5, 'class']]
```

2. Use min/max to find the correct sublist:

```
min_pair = min(scored_words)
# for the above words, we get: [2, 'to']
```

3. Use indexing to get just the desired value:

```
min_pair[1]
```

Problem Solving Using LCs and Lists of Lists (cont.)

- Here's a function that works for an arbitrary list of words:

```
def shortest_word(words):
    """ returns the shortest word from the input
        list of words
    """
    scored_words = [[len(w), w] for w in words]
    min_pair = min(scored_words)
    return min_pair[1]
```

Finding the Best Scrabble Word

- Assume we have:
  - a list of possible Scrabble words

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

  - a `scrabble_score()` function like the one from PS 2
- To find the best word:
  - form a ***list of lists*** using a list comprehension

    ```
    scored_words = [[scrabble_score(w), w] for w in words]

    ## for the above words, we get the following:
    # [[6, 'aliens'], [9, `zap'], [19, 'hazy'], [5, 'code']]
    ```

  - use `max()` to get the best `[score, word]` sublist:

    ```
    bestpair = max(scored_words)

    ## for the above words, we get the following:
    # [19, 'hazy']
    ```

  - use indexing to extract the word: `bestpair[1]`

best_word()

```
def best_word(words):
    """ returns the word from the input list of words
        with the best Scrabble score
    """
    scored_words = [[scrabble_score(w), w] for w in words]
    bestpair = max(scored_words)
    return bestpair[1]
```

How Would Your Complete This Function?

```
def longest_word(words):
    """ returns the string that is the longest
        word from the input list of words
    """
    scored_words = _____________________________
    bestpair = max(scored_words)
    return _________________
```

	first blank	second blank
A.	`[[w, len(w)] for w in words]`	`bestpair[0]`
B.	`[[len(w), w] for w in words]`	`bestpair[0]`
C.	`[[w, len(w)] for w in words]`	`bestpair[1]`
D.	`[[len(w), w] for w in words]`	`bestpair[1]`
E.	more than one of these would work	

How Would Your Complete This Function?

```
def longest_word(words):
    """ returns the string that is the longest
        word from the input list of words
    """
    scored_words = _____________________________
    bestpair = max(scored_words)
    return _________________
```

	first blank	second blank
A.	`[[w, len(w)] for w in words]`	`bestpair[0]`
B.	`[[len(w), w] for w in words]`	`bestpair[0]`
C.	`[[w, len(w)] for w in words]`	`bestpair[1]`
D.	**`[[len(w), w] for w in words]`**	**`bestpair[1]`**
E.	more than one of these would work	

ASCII

American Standard Code for Information Interchange

- Strings are sequences of characters. `'hello'`
- Individual characters are actually stored as integers.
- ASCII specifies the mapping between characters and integers.

character	ASCII value
`'A'`	65
`'B'`	66
`'C'`	67
...	
`'a'`	97
`'b'`	98
`'c'`	99
...	

Converting Between Characters and Numbers

ASCII values

```
abcdefghijklmnopqrstuvwxyz
97  99  101 103 105 107 109 111 113 115 117 119 122

ABCDEFGHIJKLMNOPQRSTUVWXYZ
65  67  69  71  73  75  77  79  81  83  85  87  90
```

Conversion functions

`ord(c)`	***input:*** a one-character string, `c` ***returns:*** an integer, the ASCII value of `c`
`chr(n)`	***input:*** an integer ASCII value ***returns:*** the one-character string for that ASCII value

Examples

```
>>> ord('e')
101
>>> ord('G')
71
```

```
>>> chr(101)
'e'
>>> chr(71)
'G'
```

Encryption

original message → *encrypted message*

`'my password is foobar'` → `'pb sdvvzrug lv irredu'`

Caesar Cipher Encryption

- Each letter is shifted/"rotated" forward by some number of places.

- Example: a shift/rotation of 3

```
'a' → 'd'
'b' → 'e'
'c' → 'f'
```

etc.

Caesar Cipher Encryption

- Each letter is shifted/"rotated" forward by some number of places.

- Example: a shift/rotation of 3

```
'a' → 'd'              'A' → 'D'
'b' → 'e'              'B' → 'E'
'c' → 'f'              'C' → 'F'
```

etc.

original message *encrypted message*

```
'my password is foobar' → 'pb sdvvzrug lv irredu'
```

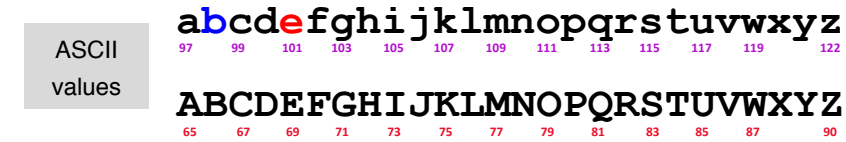
- Non-alphabetic characters are left alone.
- We "wrap around" as needed.

```
'x' → 'a'              'X' → 'A'
'y' → 'b'              'Y' → 'B'
```

etc.

Implementing a Shift in Python

ASCII values



- `ord()` and addition gives the ASCII code of the shifted letter:

```
>>> ord('b')
98
>>> ord('b') + 3      # in general, ord(c) + shift
101
```

- `chr()` turns it back into a letter:

```
>>> chr(ord('b') + 3)
'e'
```

Caesar Cipher in PS 3

- You will write an `encipher` function:

```
>>> encipher('hello!', 1)
'ifmmp!'
>>> encipher('hello!', 2)
'jgnnq!'
>>> encipher('hello!', 4)
'lipps!'
```

- "Wrap around" as needed.
  - upper-case letters wrap to upper; lower-case to lower

```
>>> encipher('XYZ xyz', 3)
'ABC abc'
```

What Should This Code Output?

```
secret = encipher('Caesar? Wow!', 5)
print(secret)
```

A. `Hfjxfw? Btb!`
B. `Hfjxfw? Wtw!`
C. `Geiwev? Asa!`
D. `Geiwev? Wsw!`
E. none of these

What Should This Code Output?

```
secret = encipher('Caesar? Wow!', 5)
print(secret)
```

H

unshifted: abcdefghijklmnopqrstuvwxyz
shifted by 5: fghijklmnopqrstuvwxyzabcde

unshifted: ABCDEFGHIJKLMNOPQRSTUVWXYZ
shifted by 5: FGHIJKLMNOPQRSTUVWXYZABCDE

A. `Hfjxfw? Btb!`
B. `Hfjxfw? Wtw!`
C. `Geiwev? Asa!`
D. `Geiwev? Wsw!`
E. none of these

What Should This Code Output?

```
secret = encipher('Caesar? Wow!', 5)
print(secret)
```

Hf

unshifted: abcdefghijklmnopqrstuvwxyz
shifted by 5: fghijklmnopqrstuvwxyzabcde

unshifted: ABCDEFGHIJKLMNOPQRSTUVWXYZ
shifted by 5: FGHIJKLMNOPQRSTUVWXYZABCDE

A. `Hfjxfw? Btb!`
B. `Hfjxfw? Wtw!`
C. `Geiwev? Asa!`
D. `Geiwev? Wsw!`
E. none of these

What Should This Code Output?

```
secret = encipher('Caesar? Wow!', 5)
print(secret)
```

Hfjxfw? B

unshifted: abcdefghijklmnopqrstuvwxyz
shifted by 5: fghijklmnopqrstuvwxyzabcde

unshifted: ABCDEFGHIJKLMNOPQRSTUVWXYZ
shifted by 5: FGHIJKLMNOPQRSTUVWXYZABCDE

A. Hfjxfw? Btb!
B. Hfjxfw? Wtw!
C. Geiwev? Asa!
D. Geiwev? Wsw!
E. none of these

What Should This Code Output?

```
secret = encipher('Caesar? Wow!', 5)
print(secret)
```

Hfjxfw? Btb!

unshifted: abcdefghijklmnopqrstuvwxyz
shifted by 5: fghijklmnopqrstuvwxyzabcde

unshifted: ABCDEFGHIJKLMNOPQRSTUVWXYZ
shifted by 5: FGHIJKLMNOPQRSTUVWXYZABCDE

A. **Hfjxfw? Btb!**
B. Hfjxfw? Wtw!
C. Geiwev? Asa!
D. Geiwev? Wsw!
E. none of these

Caesar Cipher with a Shift/Rotation of 13

- 'a' → 'n' 'n' → 'a'
  'b' → 'o' 'o' → 'b'
  'c' → 'p' 'p' → 'c'
  etc.

- Using `chr()` and `ord()`:

```
>>> chr(ord('a') + 13)
'n'
>>> chr(ord('P') + 13 - 26)      # wrap around!!
'C'
```

- Can use the following to determine if c is lower-case:

```
if 'a' <= c <= 'z':
```

- Can use the following to determine if c is upper-case:

```
if 'A' <= c <= 'Z':
```

Caesar Cipher with a Shift/Rotation of 13

```
def rot13(c):
    """ rotate c forward by 13 characters,
        wrapping as needed; only letters change
    """
    if 'a' <= c <= 'z':           # lower-case
        new_ord = ord(c) + 13
        if new_ord > ord('z'):
            new_ord = _______________
    elif 'A' <= c <= 'Z':         # upper-case
        new_ord = ord(c) + 13
        if ___________________:
            ________________________
    else:                         # non-alpha
        ___________________

    return _________________
```

Caesar Cipher with a Shift/Rotation of 13

```
def rot13(c):
    """ rotate c forward by 13 characters,
        wrapping as needed; only letters change
    """
    if 'a' <= c <= 'z':           # lower-case
        new_ord = ord(c) + 13
        if new_ord > ord('z'):
            new_ord = new_ord - 26
    elif 'A' <= c <= 'Z':         # upper-case
        new_ord = ord(c) + 13
        if new_ord > ord('Z'):
            new_ord = new_ord - 26
    else:                         # non-alpha
        new_ord = ord(c)

    return chr(new_ord)
```

Deciphering an Enciphered Text

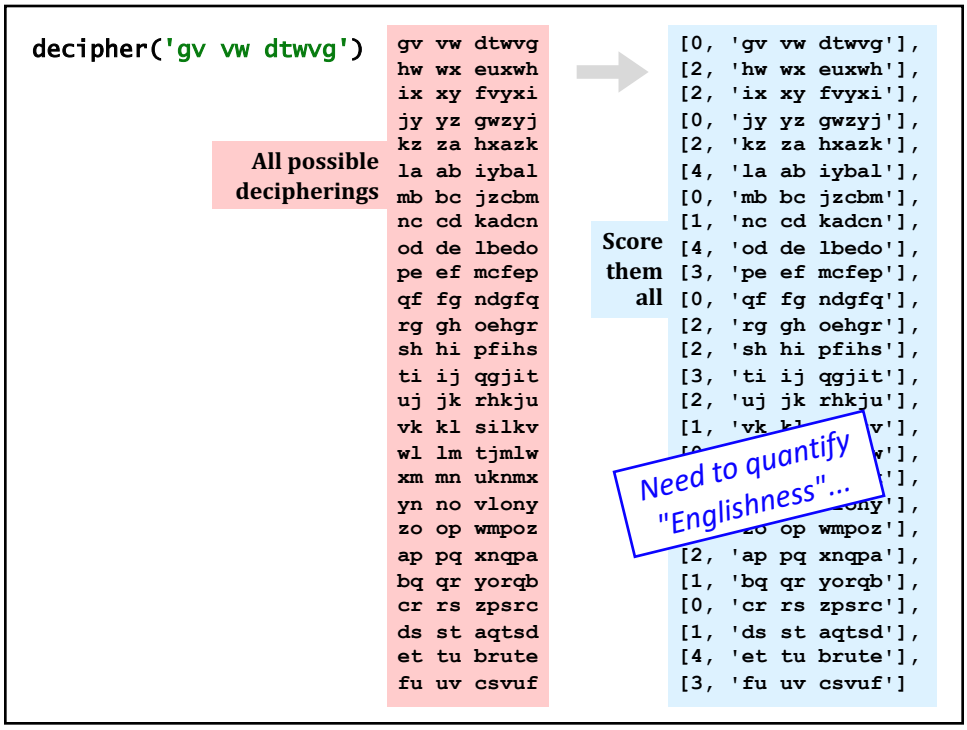
- You will write a function for this as well.

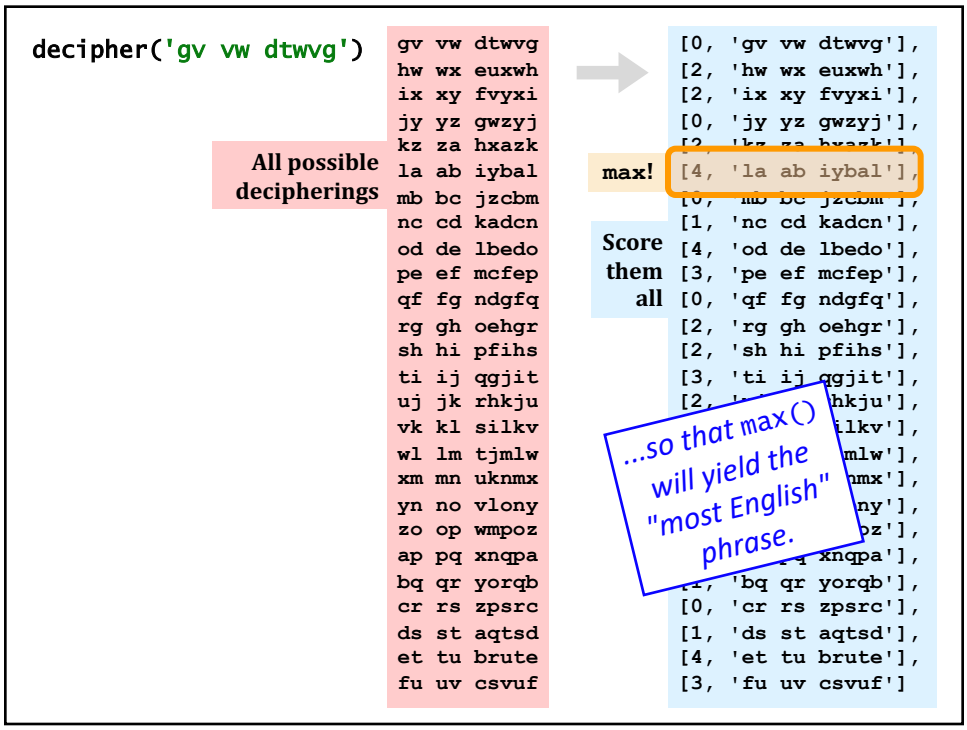
```
>>> decipher('Bzdrzq bhogdq? H oqdedq Bzdrzq rzkzc.')
'Caesar cipher? I prefer Caesar salad.'

>>> decipher('Bomebcsyx sc pexnkwoxdkv')
'Recursion is fundamental'

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

- `decipher` only takes a string.
  - no shift/rotation amount is given!
- How can it determine the correct "deciphering"?


decipher('gv vw dtwvg')
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
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
Score them all
[0, 'gv vw dtwvg'],
[2, 'hw wx euxwh'],
[2, 'ix xy fvyxi'],
[0, 'jy yz gwzyj'],
[2, 'kz za hxazk'],
[4, 'la ab iybal'],
[0, 'mb bc jzcbm'],
[1, 'nc cd kadcn'],
[4, 'od de lbedo'],
[3, 'pe ef mcfep'],
[0, 'qf fg ndgfq'],
[2, 'rg gh oehgr'],
[2, 'sh hi pfihs'],
[3, 'ti ij qgjit'],
[2, 'uj jk rhkju'],
Need to quantify "Englishness"...
[2, 'ap pq xnqpa'],
[1, 'bq qr yorqb'],
[0, 'cr rs zpsrc'],
[1, 'ds st aqtsd'],
[4, 'et tu brute'],
[3, 'fu uv csvuf']


decipher('gv vw dtwvg')
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
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
max!
Score them all
[0, 'gv vw dtwvg'],
[2, 'hw wx euxwh'],
[2, 'ix xy fvyxi'],
[0, 'jy yz gwzyj'],
[4, 'la ab iybal'],
[1, 'nc cd kadcn'],
[4, 'od de lbedo'],
[3, 'pe ef mcfep'],
[0, 'qf fg ndgfq'],
[2, 'rg gh oehgr'],
[2, 'sh hi pfihs'],
[3, 'ti ij qgjit'],
...so that max() will yield the "most English" phrase.
[2, 'ap pq xnqpa'],
[1, 'bq qr yorqb'],
[0, 'cr rs zpsrc'],
[1, 'ds st aqtsd'],
[4, 'et tu brute'],
[3, 'fu uv csvuf']

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

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
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
```

Score them all

```
[0, 'gv vw dtwvg'],
[2, 'hw wx euxwh'],
[2, 'ix xy fvyxi'],
[0, 'jy yz gwzyj'],
[2, 'kz za hxazk'],
max! [4, 'la ab iybal'],
[0, 'mb bc jzcbm'],
[1, 'nc cd kadcn'],
[4, 'od de lbedo'],
[3, 'pe ef mcfep'],
[0, 'qf fg ndgfq'],
[2, 'rg gh oehgr'],
[2, 'sh hi pfihs'],
[3, 'ti ij qgjit'],
[2, 'uj jk rhkju'],
[2, 'vk kl silkv'],
[3, 'wl lm tjmlw'],
[2, 'xm mn uknmx'],
[2, 'yn no vlony'],
[2, 'zo op wmpoz'],
[2, 'ap pq xnqpa'],
[1, 'bq qr yorqb'],
[0, 'cr rs zpsrc'],
[1, 'ds st aqtsd'],
[4, 'et tu brute'],
[3, 'fu uv csvuf']
```

A score based on # of vowels doesn't work for this phrase.

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

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
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
```

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-0?, 'vk kl silkv'],
[3.?e-0?, 'wl lm tjmlw'],
[2.?e-0?, 'xm mn uknmx'],
[2.?e-0?, 'yn no vlony'],
[2.?e-0?, 'zo op wmpoz'],
[2.?e-0?, 'ap pq xnqpa'],
[5.7e-08, 'bq qr yorqb'],
[0.00024, 'cr rs zpsrc'],
[1.?e-0?, 'ds st aqtsd'],
max! [0.45555, 'et tu brute'],
[0.00011, 'fu uv csvuf']
```

A score based on letter frequencies/ probabilities does!