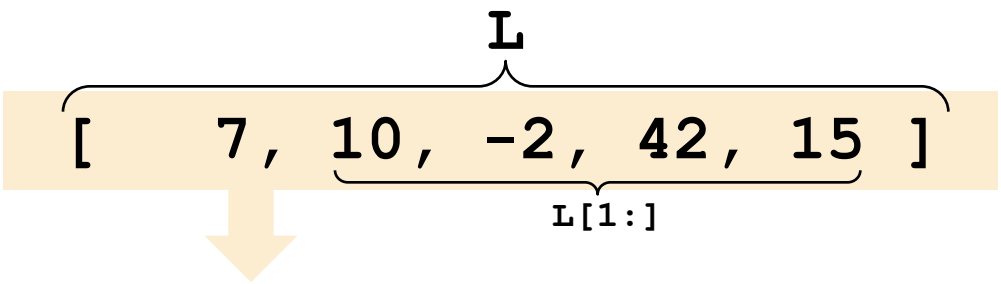
recursive max



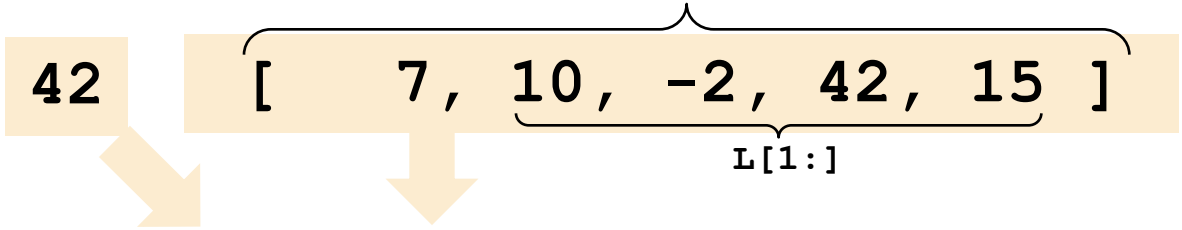

```
def max(L):
    """Returns the max element from L
       Argument: L, a nonempty list
    """
    if len(L) < 2: return L[0] # only 1 elem.

    maxOfRest = max(L[1:])    # max of the rest

    if L[0] > maxOfRest:
        return L[0]                  # either L[0]
    else:
        return maxOfRest             # or maxOfRest!
```



recursive search




```
def max(item, L):
    """Returns True if item is in L. """
    if L == []: # Empty list
        return False
    elif L[0] == item: # we found it!
        return True
    else: # keep looking
        return search(item, lst[1:])
```

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

Test yourself

```
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** is [[6, 'aliens'], [3, 'zap'], _____, _____]

2. What is `bstpr`?

3. What is returned?

L = [30, 40, 50]

Use the **LoL** method to write these two functions

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

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

Assume you have **min(LoL)** available

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

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

Hint: Define a helper function!

```
def count(e, L):
    """return # of e's in L"""
```

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


```
def maxlen(L):
    LoL = [[len(s), s] for s in L]
```

1. What is `LoL`? [[6, 'aliens'], [3, 'zap'], [4, 'hazy'], [4, 'code']]

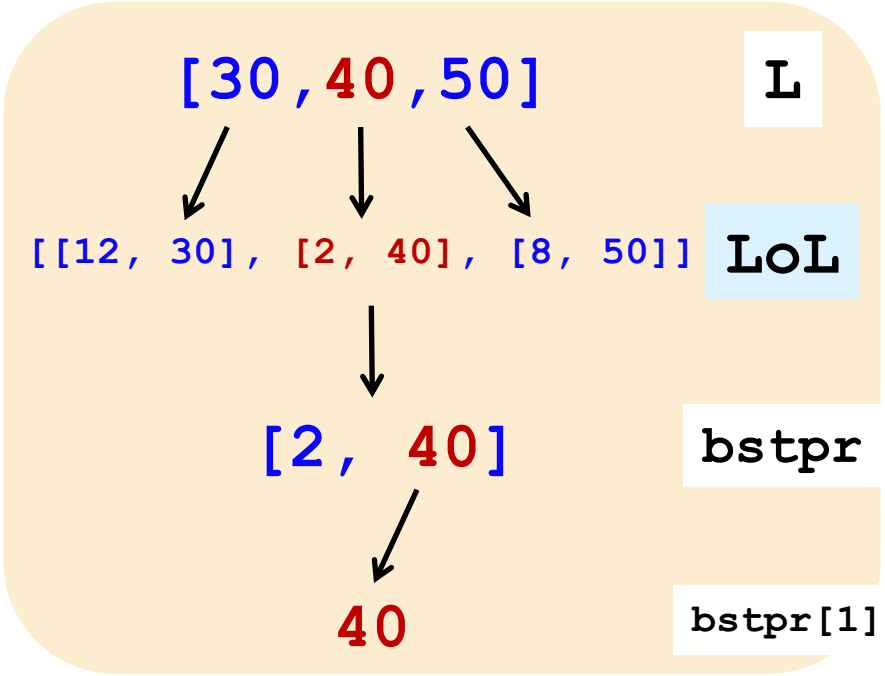
```
    bstpr = max(LoL)
```

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

```
    return bstpr[1]
```

3. What is returned? 'aliens'

bestnumb



[30, 40, 50]

```
def bestnumb(L):
    """Returns the # closest to 42 in L"""

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

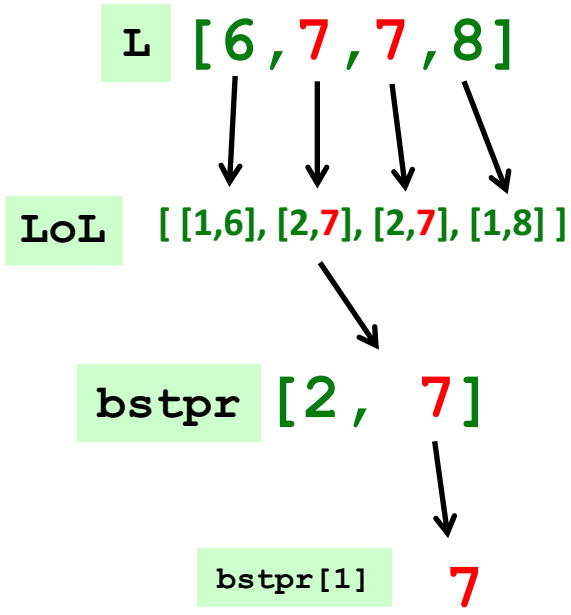
    bstpr = min(LoL)

    return bstpr[1]
```

Helper function: `count(e, L)`

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

mostnumb



[6, 7, 7, 8]

```
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?



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

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

1. What is `LoL`? [[6, 'aliens'], [3, 'zap'], [4, 'hazy'], [4, 'code']]

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

3. What is returned? 'aliens'

L = [30, 40, 50]

```
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]
```

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

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

```
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]
```

Hint: Define a helper function!

```
def count(e, L):
    LC = [1 for x in L \
            if x == e]
    return sum(LC)
```

Computing with *language*



ideas / meaning



language / words / phrases



strings

Python strings are here.



numbers / bits

Caesar Cipher: `encipher`

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

What is it doing?

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



```
encipher( 'I <3 Latin', 25 )   returns   'H <3 Kzshm'
```

Caesar Cipher: `encipher`

`encipher(s, n)` should return the string **`s`** with each ***alphabetic*** character shifted/wrapped by **`n`** places in the alphabet

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



```
encipher( 'I <3 Latin', 25 )   returns   'H <3 Kzshm'
```

ASCII

American Standard Code for Information Interchange

Binary	Dec	Hex	Glyph
0010 1111	47	2F	/
0011 0000	48	30	0
0011 0001	49	31	1

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 #

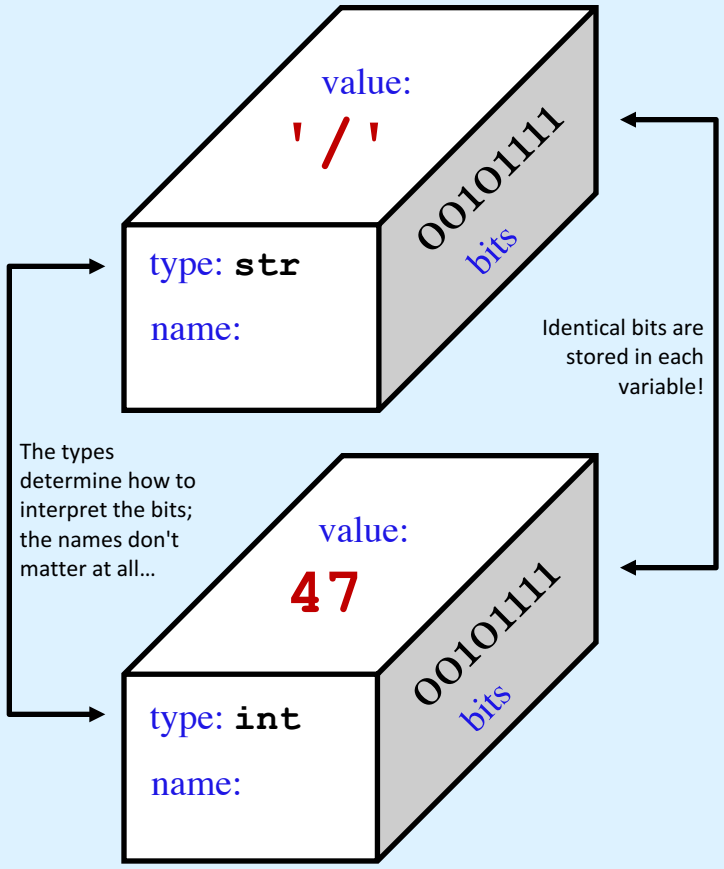
ASCII *or* Unicode…

Binary	Dec	Hex	Glyph
0010 1111	47	2F	/
0011 0000	48	30	0
0011 0001	49	31	1

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

convert # to char

`chr`



`ord`

convert char to #

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

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

Bin	Dec	Hex	Glyph
0110 0000	96	60	`
0110 0001	97	61	a
0110 0010	98	62	b
0110 0011	99	63	c
0110 0100	100	64	d
0110 0101	101	65	e
0110 0110	102	66	f
0110 0111	103	67	g
0110 1000	104	68	h
0110 1001	105	69	i
0110 1010	106	6A	j
0110 1011	107	6B	k
0110 1100	108	6C	l
0110 1101	109	6D	m
0110 1110	110	6E	n
0110 1111	111	6F	o
0111 0000	112	70	p
0111 0001	113	71	q

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

ASCII *and* Unicode

convert # to char

chr

ord

convert char to #

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

Ex1

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

Ex2

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

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

Bin	Dec	Hex	Glyph
0110 0001	97	61	a
0110 0010	98	62	b
0110 0011	99	63	c
0110 0100	100	64	d
0110 0101	101	65	e
0110 0110	102	66	f
0110 0111	103	67	g
0110 1000	104	68	h
0110 1001	105	69	i
0110 1010	106	6A	j
0110 1011	107	6B	k
0110 1100	108	6C	l
0110 1101	109	6D	m
0110 1110	110	6E	n
0110 1111	111	6F	o
0111 0000	112	70	p
0111 0001	113	71	q

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

Rot13

a useful and illustrative starting point...




```
rot13('a') should output 'n'
rot13('M') should output 'Z'
```

adding 13

```
rot13('n') should output 'a'
rot13('W') should output 'J'
```

wrapping

```
rot13(' ') should output ' '
rot13('<') should output '<'
```

spaces + other characters

ASCII and Unicode

chr value

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
97	99	101	103	105	107	109	111	113	115	117	119	122													

ord value

chr value

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
65	67	69	71	73	75	77	79	81	83	85	87	90													

ord value

convert # to char

chr



ord

convert char to #

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

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

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

how do we wrap?

Writing Rot13

any single character, `c`

```
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(                          )

    elif 'A' <= c <= 'Z':     # upper-case test!
                              )

    else:
                              )
```

(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?

Writing Rot13

any single character, `c`

```
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?

use n instead of 13

Extra: How would you rotate `n` places, instead of 13?



Caesar

Caesar Cipher: **encipher**



Brutus

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

s1

```
>>> 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 vdodg.'
```

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

```
>>> 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.'
```

s2



Caesar

Caesar Cipher: ***decipher***



Brutus

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

S1

S2

```
>>> 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!')
```

PL

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

How!?

Which is more difficult computationally?

Decipher?

Strategies?

Algorithms?

Decipher?

Strategies?

Algorithms?

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
```

Decipher?

Strategies?

Algorithms?

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'],
[1, 'vk kl
                v'],
[0
                w'],
                ],
                ony'],
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']
```

quantifying Englishness?

Decipher?

Strategies?

Algorithms?

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'],
[2, 'uj jk rhkju'],
[0, 'vk kl silkv'],
[1, 'wl lm tjmlw'],
[3, 'xm mn uknmx'],
[2, 'yn no vlony'],
[1, 'zo op wmpoz'],
[0, 'ap pq xnqpa'],
[1, 'bq qr yorqb'],
[0, 'cr rs zpsrc'],
[1, 'ds st aqtsd'],
[4, 'et tu brute'],
[3, 'fu uv csvuf']
```

yields the "most English" phrase

Measuring *Englishness*

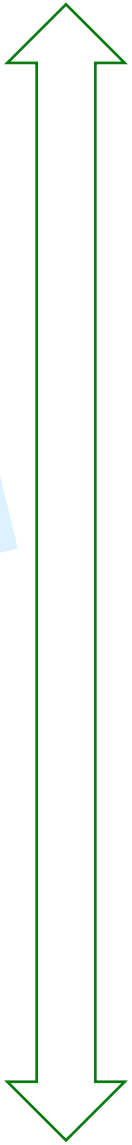
Very English-y

higher scores

quantifying Englishness?

lower scores

Not 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!



Decipher?

Strategies?

Algorithms?

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'],
[2, 'uj jk rhkju'],
[1, 'vk kl silkv'],
[1, 'wl lm tjmlw'],
[2, 'xm mn uknmx'],
[2, 'yn no vlony'],
[2, 'zo op wmpoz'],
[1, 'ap pq xnqpa'],
[1, 'bq qr yorqb'],
[0, 'cr rs zpsrc'],
[1, 'ds st aqtsd'],
[4, 'et tu brute'],
[3, 'fu uv csvuf']
```

"Englishness" score based on # of vowels

Decipher?

Strategies?

Algorithms?

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-  silkv'],
        tjmlw'],
        uknmx'],
        lony'],
        wmpoz'],
        'ap pq xnqpa'],
[5.7e-08, 'bq qr yorqb'],
[0.00024, 'cr rs zpsrc'],
[0.02060, 'ds st aqtsd'],
max! [0.45555, 'et tu brute'],
[0.00011, 'fu uv csvuf']
```

"Englishness" based on letter probabilities