CS 5 Lecture 6:

max(LoLs), Dictators, and Dictionaries

Computing to the `max`

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

a *comparison* **comparison**

```
max('m+ms','kitkat')
max([0,42],[4,2])
max([4,'m+ms'],[4,'mocha'])
```

Computing with ***language***

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

Recursive snacks?

Cookies-and-Cream Oreos Are Basically Cookie Inception

Oreos are already cookies and cream, right?

"To clarify, the 'chocolayer' — the **filling** between the wafer of a **Kit Kat** — is made from cocoa liquor, sugar and a small amount of re-worked **Kit Kat**," a Nestlé U.K. spokesperson confirmed, adding, "Please note, re-worked **Kit Kat** is product which cannot be sold." Feb 14, 2019

www.today.com › food › kit-kat-bars-are-made-ground-k...

Kit Kat bars are made with ground-up Kit Kats - The Today Show

max

A recipe for life ?

and python already has it for us…

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

`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] )
       'nov'   'jan'   'mar'   'may'   'jul'   'sep'
```

ST

What if the months are in there, as well?

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

STm

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

mST

to the **max**

Want the highest price?

```
max( [475.5, 458.0, 441.3, 470.8, 532.8, 520.9] )
       'nov'   'jan'   'mar'   'may'   'jul'   'sep'
```

ST

What if the months are in there, as well?

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

STm

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

mST

Mudd's **max**?

MSt

```
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']
```

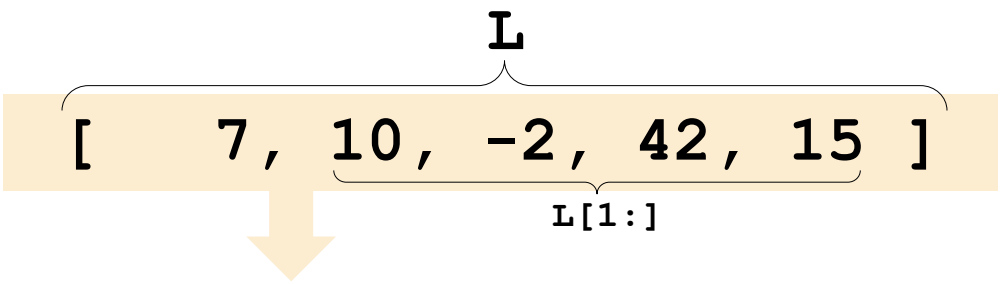
max(MSt)

Or Mudd's **min**?

min(MSt)

'CS' < 'clear'

recursive max

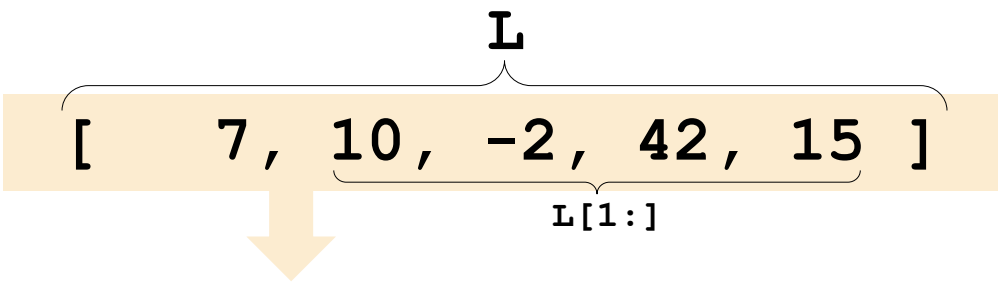


L = ['aliens', 'zap', 'hazy', 'code']

```
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
```

What two elements might be the overall max?

recursive max



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

```
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!
```

I <3 max rest!

`max` *with scrabble-score*

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

6 14 19 7

Which element has the highest scrabble score?

```
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!
```

Spacey! I like it!

`max` *with scrabble-score*

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

6 14 19 7

Which element has the highest scrabble score?

```
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!
```

`max` *with scrabble-score*

```
L = [ 'aliens', 'zap', 'hazy', 'code' ]
        6       14      19       7
```

Which element has the highest scrabble score?

Let's see if we can simplify this process... *just for* **LoLs**!

```
def maxSS( L ):
    """ returns L's hig                ble-scoring
                                       onempty list)
                        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: **LoL**

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

6 14 19 7

```
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!

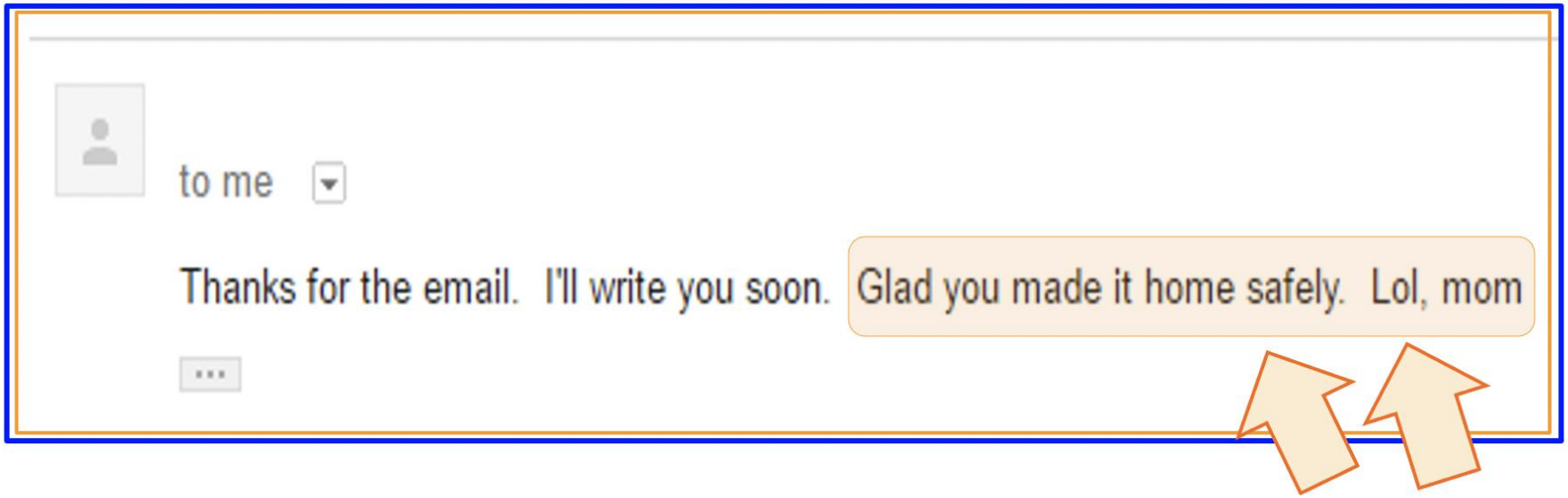
A more ***comprehensive*** solution: **LoL**

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

6 14 19 7

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

This *does* look funny!

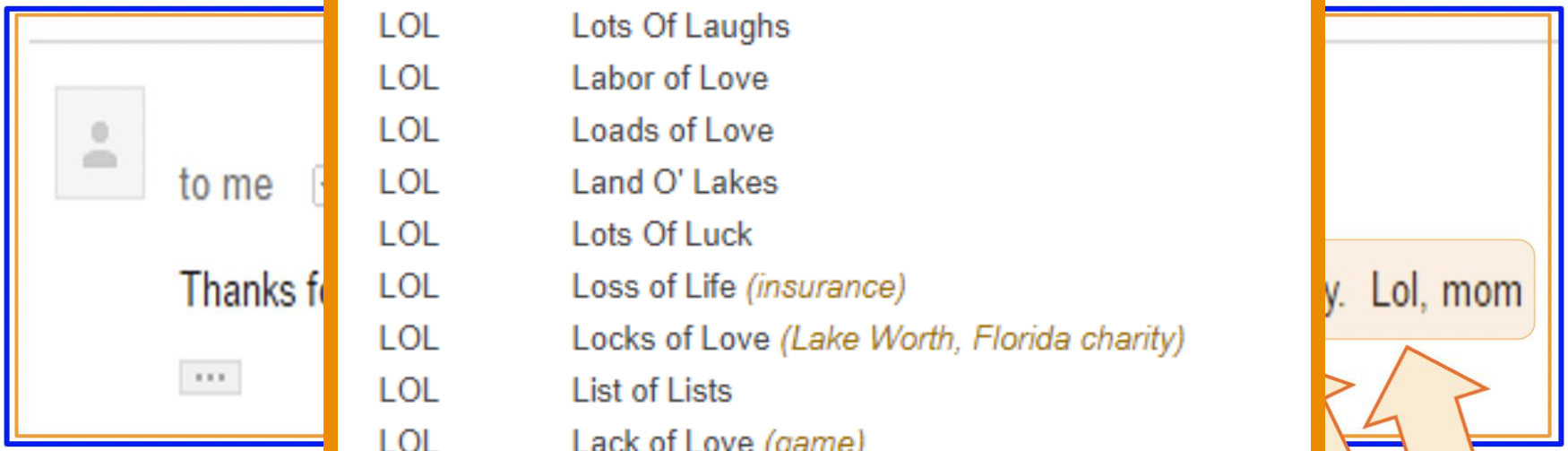


A more ***comprehensive*** solution: **`LoL`**

```
def maxS
    """
    """
    LoL
```

```
ce word
L   ]
```

This *does* look funny!



LOL

Also found in: Dictionary, Idioms, Encyclopedia, Wikipedia.

Category filter: Show All (90)

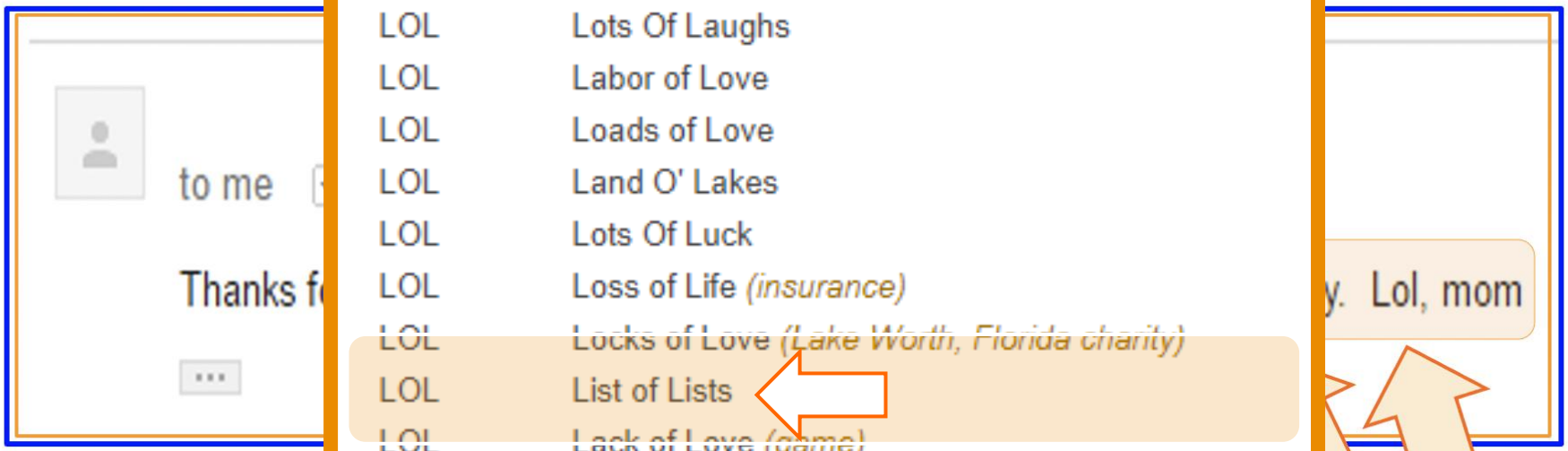
Acronym	Definition
LOL	Laugh*(ing)* Out Loud
LOL	Lots Of Love
LOL	League of Legends *(game)*
LOL	Little Old Lady
LOL	Lots Of Laughs
LOL	Labor of Love
LOL	Loads of Love
LOL	Land O' Lakes
LOL	Lots Of Luck
LOL	Loss of Life *(insurance)*
LOL	Locks of Love *(Lake Worth, Florida charity)*
LOL	List of Lists
LOL	Lack of Love *(game)*
LOL	Lowest of the Low
LOL	Lady of the Lake

A more ***comprehensive*** solution: **`LoL`**

```
def maxS
    """
    """
    LoL
```

```
e word
L   ]
```

This *does* look funny!



LOL

Also found in: Dictionary, Idioms, Encyclopedia, Wikipedia.

Category filter: Show All (90)

Acronym	Definition
LOL	Laugh*(ing)* Out Loud
LOL	Lots Of Love
LOL	League of Legends *(game)*
LOL	Little Old Lady
LOL	Lots Of Laughs
LOL	Labor of Love
LOL	Loads of Love
LOL	Land O' Lakes
LOL	Lots Of Luck
LOL	Loss of Life *(insurance)*
LOL	Locks of Love *(Lake Worth, Florida charity)*
LOL	List of Lists
LOL	Lack of Love *(game)*
LOL	Lowest of the Low
LOL	Lady of the Lake

A more ***comprehensive*** solution: **LoL**

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

6 14 19 7


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


    bestpair = max( LoL )


    return bestpair[1]
```

Let's follow the data …

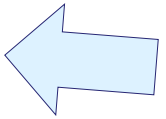
A more ***comprehensive*** solution

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

6 14 19 7

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

LoL = [[6,'aliens'], [14,'zap'], [19,'hazy'], [7,'code']]




```
    bestpair = max( LoL )
```

bestpair = [19,'hazy']

```
    return bestpair[1]
```

'hazy'

Data, followed!

Everything ... is a max problem?

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

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

```
def mystery( L ):
    """ another example - what's returned?
    """
    LoL = [ [vwl(w), w] for w in L  ]
```

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

```
    bestpair = max( LoL )
```

bestpair =

```
    return bestpair[1]
```

Let's follow the data ...

Everything ... is a max problem?

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

```
def mystery( L ):
    """ another example - what's returned?
    """
    LoL = [ [vwl(w), w] for w in L  ]
```

LoL = [[3 ,'aliens'], [1 ,'zap'], [1 ,'hazy'], [2 ,'code']]

```
    bestpair = max( LoL )
```

bestpair = [3 ,'aliens']

```
    return bestpair[1]
```

'aliens'

for the wiiin!

Data, *followed!*

Everything ... is a max problem?

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

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

```
def mystery2( L ):
    """ another example - what's returned?
    """
    LoL = [ [w[::-1], w] for w in L  ]


    bestpair = max( LoL )


    return bestpair[1]
```

Let's follow the data ...

Everything ... is a max problem?

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

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

```
def mystery2( L ):
    """ another example - what's returned?
    """
    LoL = [ [w[::-1], w] for w in L  ]
```

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

```
    bestpair = max( LoL )
```

bestpair =

```
    return bestpair[1]
```

... processing ...

Everything ... is a max problem?

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

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

```
def mystery2( L ):
    """ another example - what's returned?
    """
    LoL = [ [w[::-1], w] for w in L  ]
```

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

```
    bestpair = max( LoL )
```

bestpair = ['yzah' ,'hazy']

```
    return bestpair[1]
```

'hazy'

Data, *followed!*

Other examples...

What is **bestnumb** ?

What is **mostnumb** ?

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

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

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

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

These functions *have* made me number

Matching LoLs

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

```
def maxlen(L):
```

L = [30, 40, 50]

```
def bestnumb(L):
```

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

```
def mostnumb( L ):
```

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

(B)
```
LoL = [ [count(x,L),x] for x in L ]
```

(C)
```
LoL = [ [len(x),x] for x in L ]
```

LoLs!

Name(s) _____

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`? here is a start: LoL is [[6,'aliens'], [3,'zap'], _____, _____]

2. What is `bstpr`?

3. What is returned?

Extra:

Change exactly ***three*** characters in this code so that **3** 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

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

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

Hint: Use this helper function!

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

Extra: Write the LC that *implements* this helper function!

LoLs' sols

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'

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 = [ [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: Use this helper function!

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

Extra: Write the LC that *implements* this helper function!

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 )
```

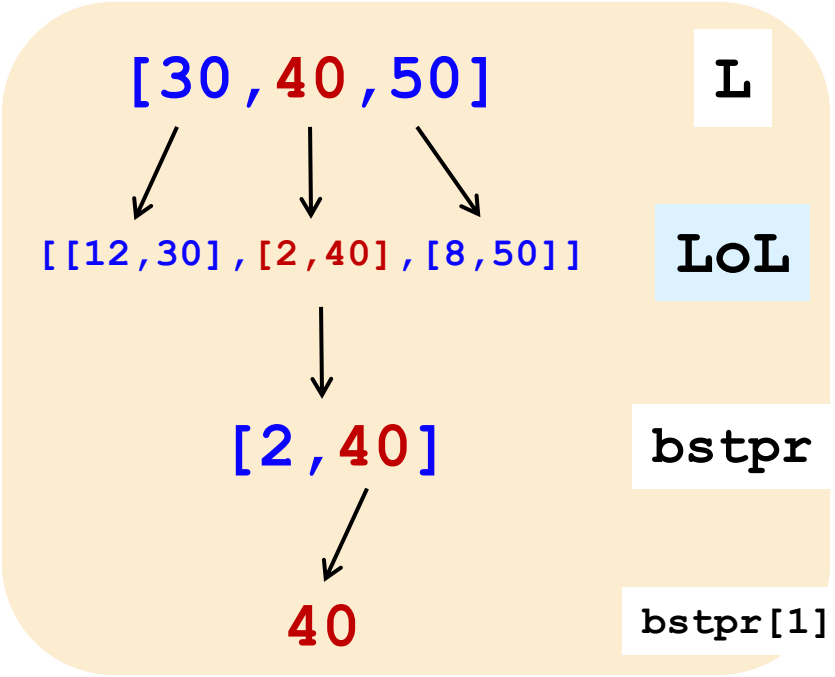
2. What is **bstpr**? [6,'aliens']

```
    return bstpr[1]
```

3. What is returned? 'aliens'

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

bestnumb



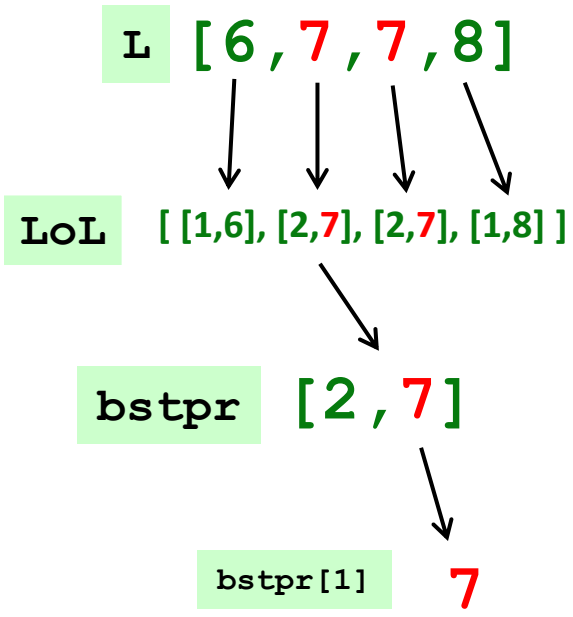
[30, 40, 50]

```
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?

Computing with *language*

ideas / meaning

language / words / phrases

strings

Python strings are here.

"alphabetic processions"

numbers / bits

Computing with *language*

ideas / meaning

open questions…

Eliza, Siri, Tay … trouble?

language / words / phrases

This week…

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

strings

how strings are represented and stored

Next week…

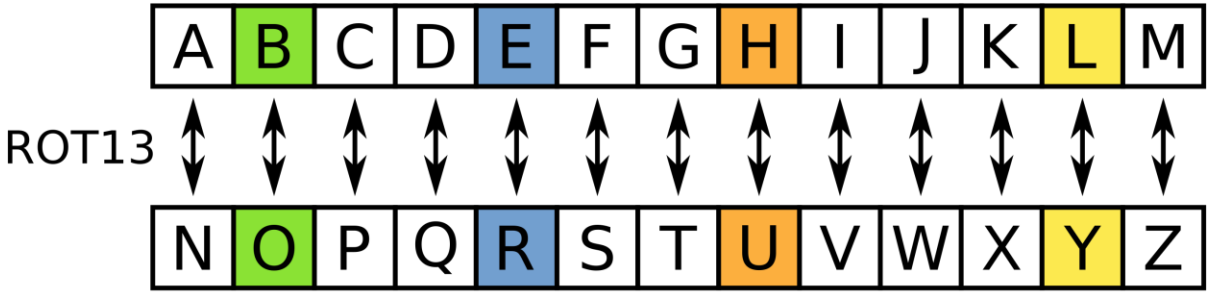
numbers / bits

A Joke…

What do you call a factory that makes okay products?

N fngvfsnpgbel.

The punchline has been "hidden" via rot13.



chr(8834)

ASCII ⊂ Unicode

American Standard Code for Information Interchange

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

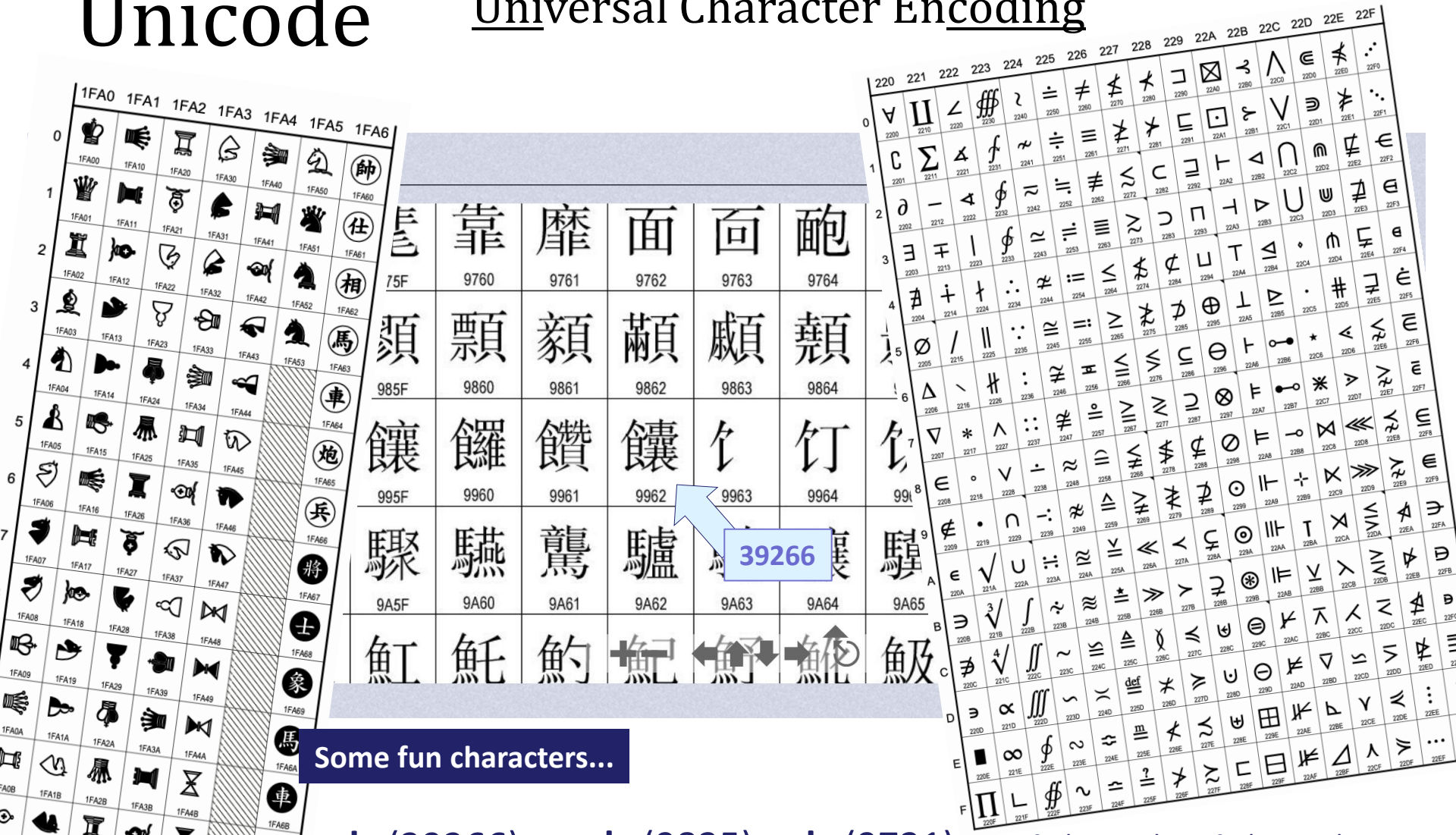
Bin	Dec	Hex	Glyph
0100 0000	64	40	@
0100 0001	65	41	A
0100 0010	66	42	B
0100 0011	67	43	C
0100 0100	68	44	D
0100 0101	69	45	E
0100 0110	70	46	F
0100 0111	71	47	G
0100 1000	72	48	H
0100 1001	73	49	I
0100 1010	74	4A	J
0100 1011	75	4B	K
0100 1100	76	4C	L
0100 1101	77	4D	M
0100 1110	78	4E	N
0100 1111	79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

Unicode

Universal Character Encoding



Some fun characters...

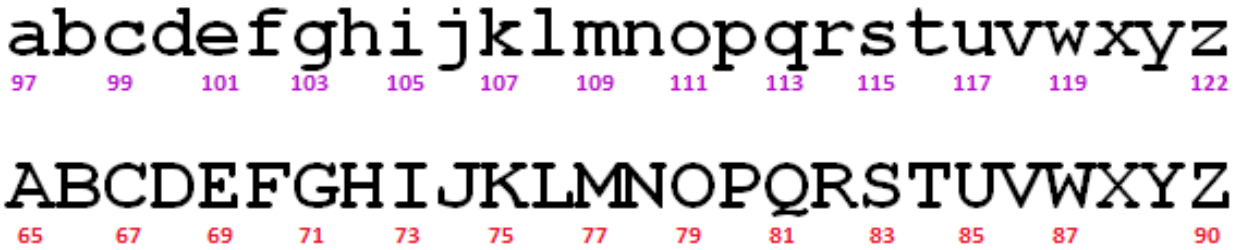
chr(39266) **chr**(9835) **chr**(9731) **chr**(19977) + **chr**(30524)

extra terrestrial: 外星人

My favorite is chr(1661)

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

abcdefghijklmnopqrstuvwxyz

97 99 101 103 105 107 109 111 113 115 117 119 122 **ord** value

chr value

ABCDEFGHIJKLMNOPQRSTUVWXYZ

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':     # uppercase 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':     # uppercase 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?

Look it up!


```
{'a': 'n',  'A': 'N',  'b': 'o',  'B': 'O',
 'c': 'p',  'C': 'P',  'd': 'q',  'D': 'Q',
 'e': 'r',  'E': 'R',  'f': 's',  'F': 'S',
 'g': 't',  'G': 'T',  'h': 'u',  'H': 'U',
 'i': 'v',  'I': 'V',  'j': 'w',  'J': 'W',
 'k': 'x',  'K': 'X',  'l': 'y',  'L': 'Y',
 'm': 'z',  'M': 'Z',  'n': 'a',  'N': 'A',
 'o': 'b',  'O': 'B',  'p': 'c',  'P': 'C',
 'q': 'd',  'Q': 'D',  'r': 'e',  'R': 'E',
 's': 'f',  'S': 'F',  't': 'g',  'T': 'G',
 'u': 'h',  'U': 'H',  'v': 'i',  'V': 'I',
 'w': 'j',  'W': 'J',  'x': 'k',  'X': 'K',
 'y': 'l',  'Y': 'L',  'z': 'm',  'Z': 'M'}
```

I should look this up...

Language? Dictionaries!

keys values

dictionary

```
D = { "induction": "self-reference in math",
      "recursion": "self-reference in cs",
        "flexion": "self-reference everywhere else",
             42: "the answer",
     "dictionary": "a cs lookup table, like this!"
}
```

Language? Dictionaries!

keys values

dictionary
```
D = { "induction": "self-reference in math",
      "recursion": "self-reference in cs",
        "flexion": "self-reference everywhere else",
             42: "the answer",
     "dictionary": "a cs lookup table, like this!"
}
```

---

key value

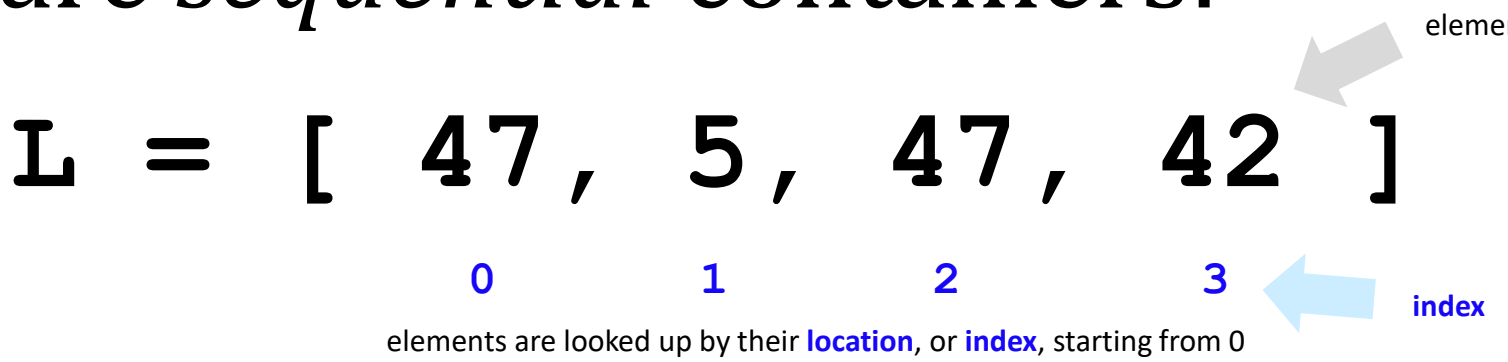
```
D["recursion"]  == "self-reference in cs"
```

key value

```
D[42]  == "the answer"
```

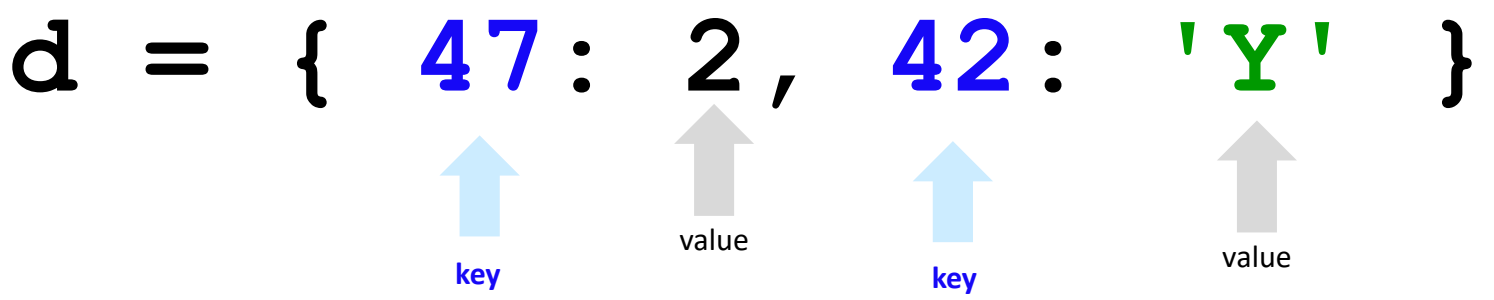
Dictionaries are **lookup tables**!
Looking up a **key** provides the table's **value**.

Lists are *sequential* containers:



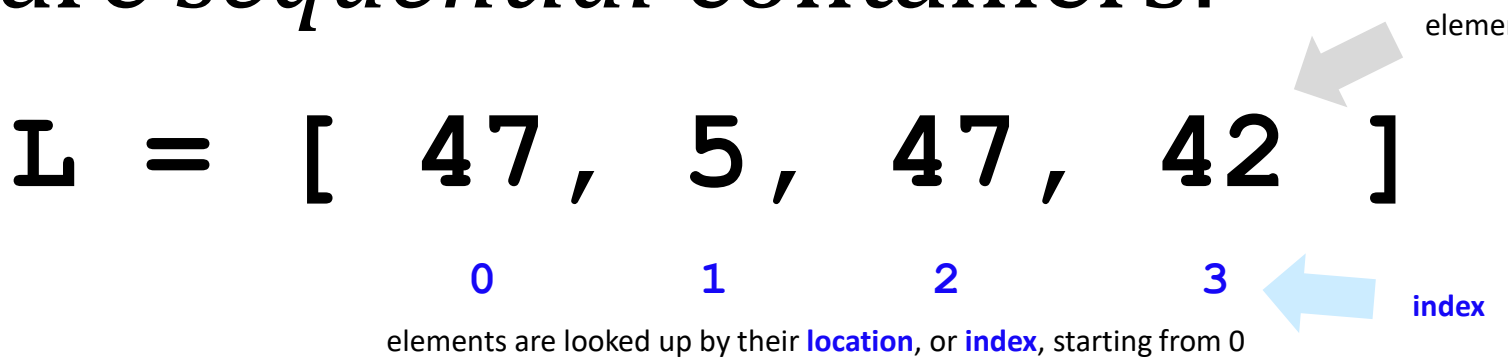
elements are looked up by their **location**, or **index**, starting from 0

Dictionaries are *arbitrary* containers:



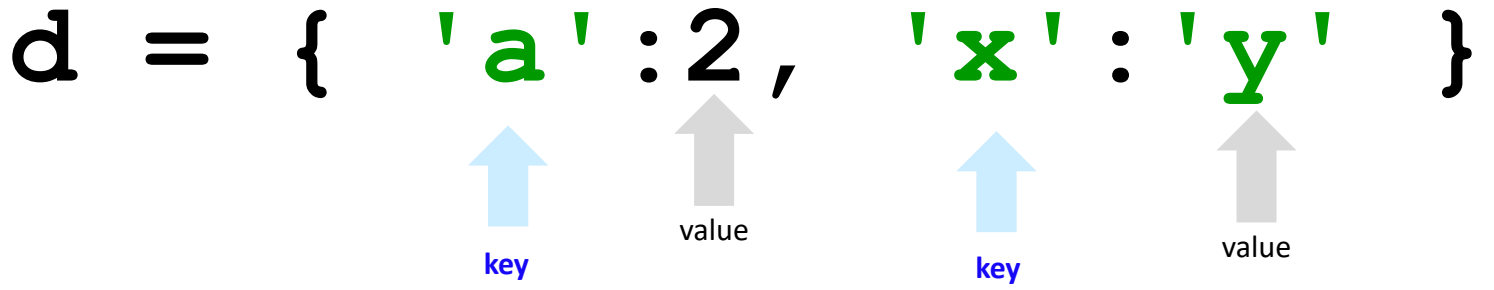
elements (or *values*) are looked up by a **key** starting anywhere you want! **Keys** don't have to be ints!

Lists are *sequential* containers:



elements are looked up by their **location**, or **index**, starting from 0

Dictionaries are *arbitrary* containers:



elements (or *values*) are looked up by a **key** starting anywhere you want! **Keys** don't have to be ints!

Look it up!

Writing Rot13

rot13dict =

```
{'a': 'n',  'A': 'N',  'b': 'o',  'B': 'O',
 'c': 'p',  'C': 'P',  'd': 'q',  'D': 'Q',
 'e': 'r',  'E': 'R',  'f': 's',  'F': 'S',
 'g': 't',  'G': 'T',  'h': 'u',  'H': 'U',
 'i': 'v',  'I': 'V',  'j': 'w',  'J': 'W',
 'k': 'x',  'K': 'X',  'l': 'y',  'L': 'Y',
 'm': 'z',  'M': 'Z',  'n': 'a',  'N': 'A',
 'o': 'b',  'O': 'B',  'p': 'c',  'P': 'C',
 'q': 'd',  'Q': 'D',  'r': 'e',  'R': 'E',
 's': 'f',  'S': 'F',  't': 'g',  'T': 'G',
 'u': 'h',  'U': 'H',  'v': 'i',  'V': 'I',
 'w': 'j',  'W': 'J',  'x': 'k',  'X': 'K',
 'y': 'l',  'Y': 'L',  'z': 'm',  'Z': 'M'}
```

```
def rot13alt( c ):
    """ rotates c by 13 chars, "wrapping" as needed
        NON-LETTERS DO NOT CHANGE!"""
    if c in rot13dict:
        return rot13dict[c]
    else:
        return c
```

Or use modulo!


```
def rot13mod( c ):
    """ rotates c by 13 chars, "wrapping" as needed
        NON-LETTERS DO NOT CHANGE!
    """
    lc = c.lower()
    if 'a' <= lc <= 'z':
        index = ord(lc) - ord('a')
        new_index = (index + 13) % 26
        delta = new_index - index
        return chr( ord(c) + delta )
    else:
        return c
```

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.',0)
```

```
>>> encipher('Bzdrzq bhogdq? H oqdedq Bzdrzq rzkzc.',1)
```

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.',0)
'Bzdrzq bhogdq? H oqdedq Bzdrzq rzkzc.'
```

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

Caesar Cipher: `encipher`

`encipher(s,N)` returns 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'
```

Caesar Cipher: `encipher`

`encipher(s,N)` returns 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'
```

"…si qua occultius perferenda erant, per notas scripsit, id est sic structo litterarum ordine, ut nullum verbum effici posset; quae si qui investigare et persequi velit, quartam elementorum litteram, **id est D pro A** et perinde reliquas commutet…"

- Suetonius, *De Vitae Caesar*

"…if any were to be conveyed more secretly, he wrote in notes, that is, in such a structured order of letters that no word could be made; that is, he **exchanges D for A** and exchanges the rest in the same manner…"

- Suetonius, *The Life of Caesar*

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.'
```

But encipher is only *half* of the challenge...

```
>>> 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

LAT

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

How!?

Which is more "computationally challenging"? **`encipher`** or **`decipher`**?

Decipher?

Strategies?

Algorithms?

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
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
```

Strategies?

Algorithms?

Decipher?

Strategies?

Algorithms?

decPR(LAT)
decPR2(LAT)
decPR3(LAT)

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 silkv'],
[0, 'wl lm tjml
[1, 'xm
```

What score could quantify "English-ness"?

What is this "scored stuff" an example of?

Measuring *Englishness*

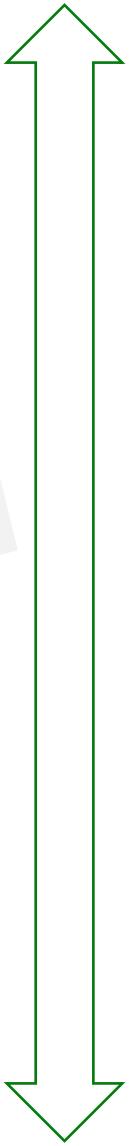
Very English-y

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 wmpo
ap
aqtsd
e
f
```

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'],
[0, 'wl lm tjmlw'],
[1, 'xm mn uknmx'],
[2, 'yn no vlony'],
[3, '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']
```

"Englishness" score based on #-of-vowels

This is a LoL!

decPR(LAT)
decPR2(LAT)
decPR3(LAT)

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
fu uv csvuf
```

Scores

```
[[27, 'gv vw dtwvg'],
 [38, 'hw wx euxwh'],
 [42, 'ix xy fvyxi'],
 [54, 'jy yz gwzyj'],
 [54, 'kz za hxazk'],
 [16, 'la ab iybal'],
 [39, 'mb bc jzcbm'],
 [21, 'nc cd kadcn'],
 [14, 'od de lbedo'],
 [23, 'pe ef mcfep'],
 [39, 'qf fg ndgfq'],
 [18, 'rg gh oehgr'],
 [23, 'sh hi pfihs'],
 [33, 'ti ij qgjit'],
 [41, 'uj jk rhkju'],
 [27, 'vk kl silkv'],
 [26, 'wl lm tjmlw'],
 [33, 'xm mn uknmx'],
 [18, 'yn no vlony'],
 [36, 'zo op wmpoz'],
 [40, 'ap pq xnqpa'],
 [43, 'bq qr yorqb'],
 [24, 'cr rs zpsrc'],
 [20, 'ds st aqtsd'],
 [11, 'et tu brute'],
 [23, 'fu uv csvuf']]
```

"Englishness" based on scrabble-scores

LoL !!

decPR(LAT)
decPR2(LAT)
decPR3(LAT)