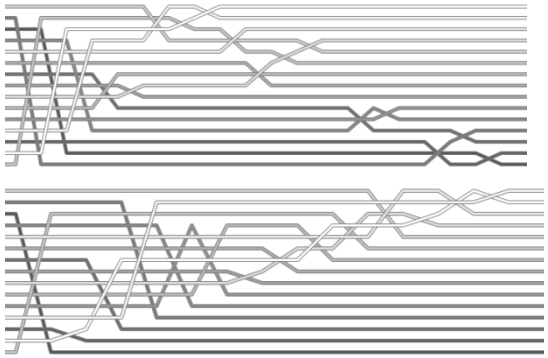
CS 42 Today

Midterm 1: DUE 9AM TOMORROW!

Allowed:

- 1 8.5x11 sheet of paper with your own writing/work.
  (Printed copies of your code done for this class OK)
- A computer for typing your solutions
  - Running of code NOT OK

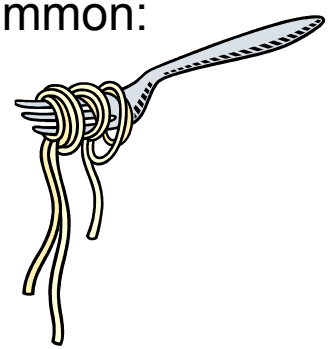


http://sortvis.org/

Can you recognize
these sorting algorithms?

I'd love to get it all **sorted** out.

What do the following have in common:



*Function*ing in Python

Some basic, built-in functions:

`abs`	absolute value
`max`, `min`, `sum`	of lists
`range`	creates lists
`round`	???

these change data from one type to another:

- `bool`
- `float`
- `int`
- `long`
- `list`
- `str`

These are the most important: `help` `dir`

*Function*ing in Python

Far more are available in separate files, or *modules*:

```
import math
```

accesses **`math.py`**'s functions

```
math.sqrt( 1764 )
```

```
dir(math)
```

lists all of **`math.py`**'s functions

```
from math import *
```

same, but without typing **`math.`** all of the time…

```
pi
```

```
sin( pi/2 )
```

help()
help modules

*Function*ing in Python

```
# my own function!
def dbl( x ):
    """ returns double its input, x """
    return 2*x
```

Functioning in Python

```
# my own function!
def dbl( x ):
    """ returns double its input, x """
    return 2*x
```

keywords

`def` starts the function
`return` stops it immediately and sends back the return value

Comments

They begin with #

Some of Python's *baggage*…

Docstrings

They become part of python's built-in help system!
With each function be sure to include one that

(1) describes overall what the function does, and
(2) explains what the inputs mean/are

*Function*ing in Python

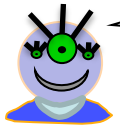
```
def undo(s):
    """ this "undoes" its string input, s """
    return 'de' + s
```

```
>>> undo('caf')
```

```
>>> undo(undo('caf'))
```

if, elif, else…

```
def foo(x):
        """This function demonstrates the use
        of if, elif, and else"""
        if x > 0 and x < 42:
                return "Small"
        elif x >= 42 and x < 100:
                return "Nice!"
        elif 100 <= x < 200:    # <- Funky!
                return "Big"
        else:
                print "That was one nasty number!"
                return "Yuck!"
```



Notice how lines with the same level of indentation are in the same code block!

Range

```
>>> range(5)
[0, 1, 2, 3, 4]
```

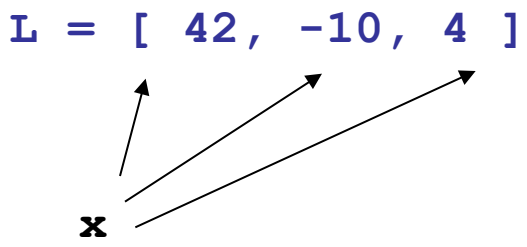
```
>>> range(5, 10)
[5, 6, 7, 8, 9]
```

```
>>> range(5, 20, 2)
[5, 7, 9, 11, 13, 15, 17, 19]
```

Two kinds of `for` loops

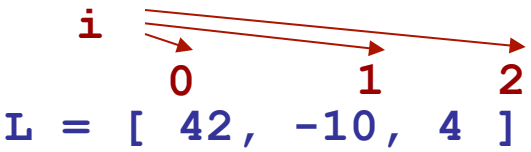
Element-based Loops

```
sum = 0

for x in L:
    sum += x
```



Index-based Loops

```
sum = 0

for i in                    :
    sum +=
```



Mutable vs. Immutable data

Changeable types:

```
dictionary

list
```

Unchangeable types:

```
tuple

string

int

float

bool
```

What's a dictionary?

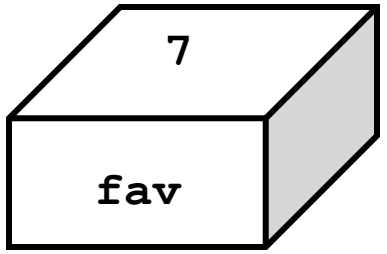
I guess I'll have to look it up!

Functions and (immutable) Variables

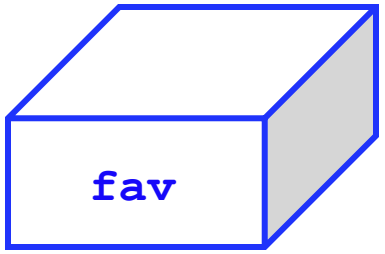
```
def fact(a):
    """non-recursive factorial"""
    result = 1
    while a > 0:
        result *= a
        a -= 1
    return result

>>> x = 5
>>> y = fact(x)
>>> x
??
```

“Pass By Value”

```
def main()
    """ calls conform """
    print " Welcome to Conformity, Inc. "

    fav = 7
    conform(fav)

    print " My favorite number is", fav
```

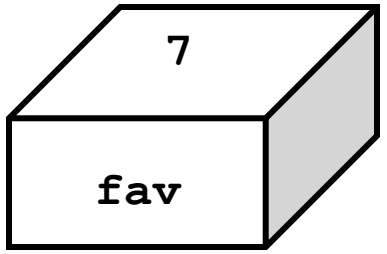


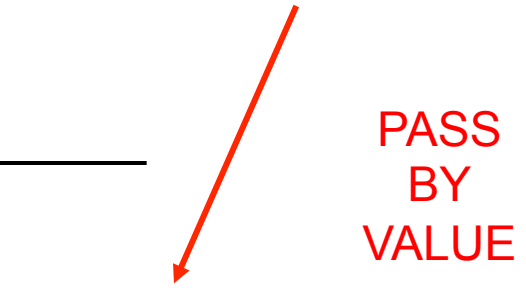
```
def conform(fav)
    """ sets input to 42 """
    fav = 42
    return fav
```



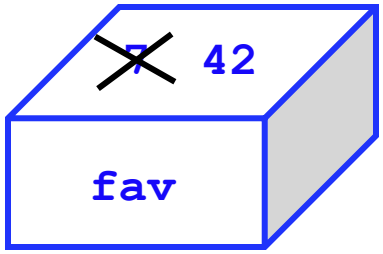
“Pass By Value”

```
def main()
    """ calls conform """
    print " Welcome to Conformity, Inc. "

    fav = 7
    conform(fav)

    print " My favorite number is", fav
```






```
def conform(fav)
    """ sets input to 42 """
    fav = 42
    return fav
```

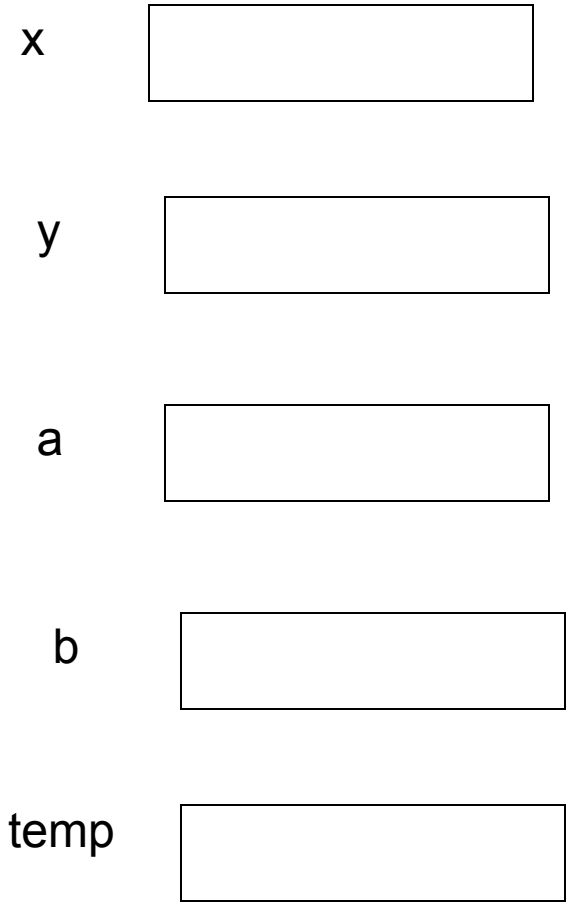


“Pass by value” means that data is **copied** when sent to a method

Functions and (immutable) Variables

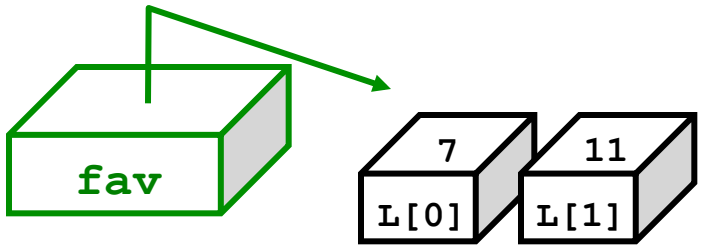
The fine print: This is not the “truth”… but it’s close enough

```
def swap(a, b):
    temp = a
    a = b
    b = temp

>>> x = 5
>>> y = 10
>>> swap(x, y)
>>> print x, y
??
```

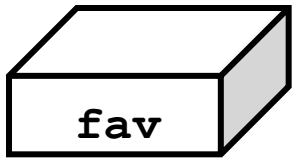


Passing ***lists*** by value…

```
def main()
    """ calls conform2 """
    print " Welcome to Conformity, Inc. "
    fav = [ 7, 11 ]
    conform2(fav)
    print " My favorite numbers are", fav
```



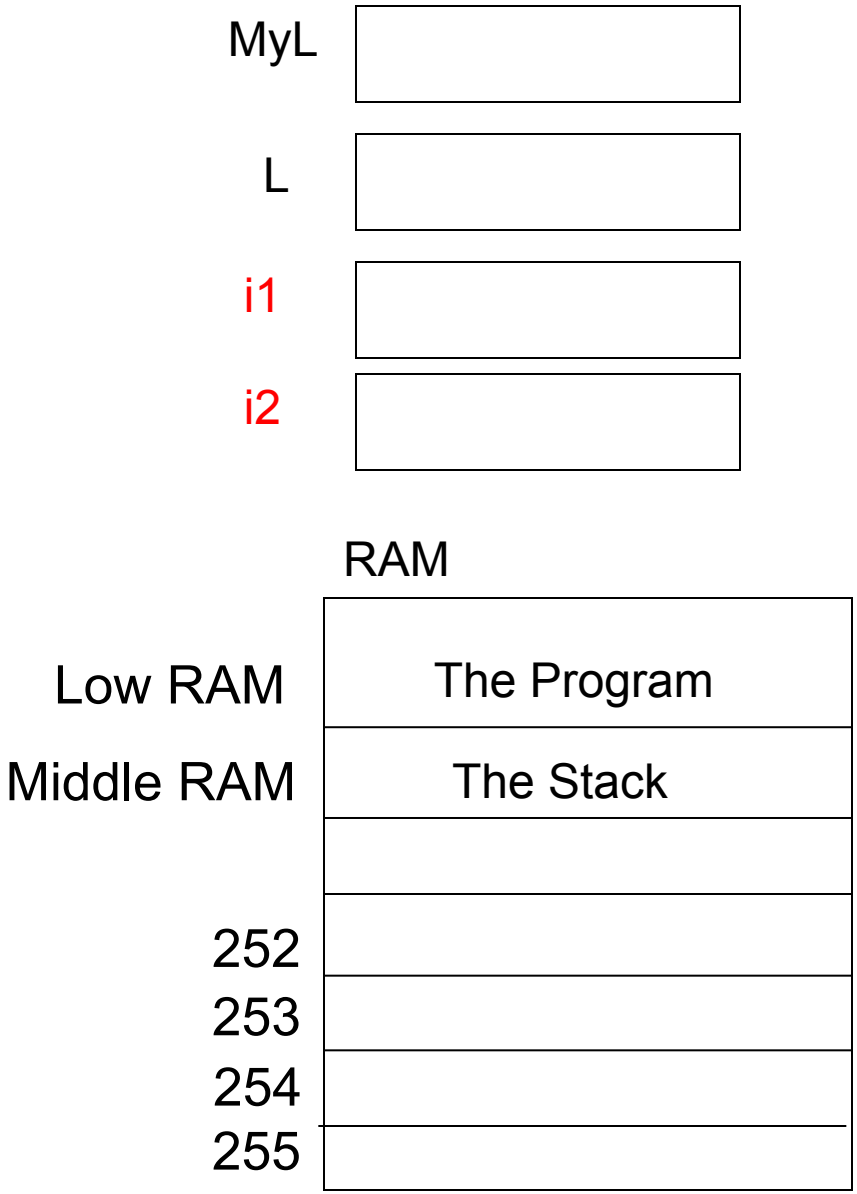
```
def conform2(fav)
    """ sets all of fav to 42 """
    fav[0] = 42
    fav[1] = 42
```



What gets passed by value here?

Functions and Mutable Types

```
def swap(L, i1, i2):
    temp = L[i1]
    L[i1] = L[i2]
    L[i2] = temp

>>> MyL = [2, 3, 4, 1]
>>> swap(myL, 0, 3)
>>> print myL
??
```



Reference vs. Value

Mutable types:

`dictionary`

`list`

Unmutable types:

`tuple`

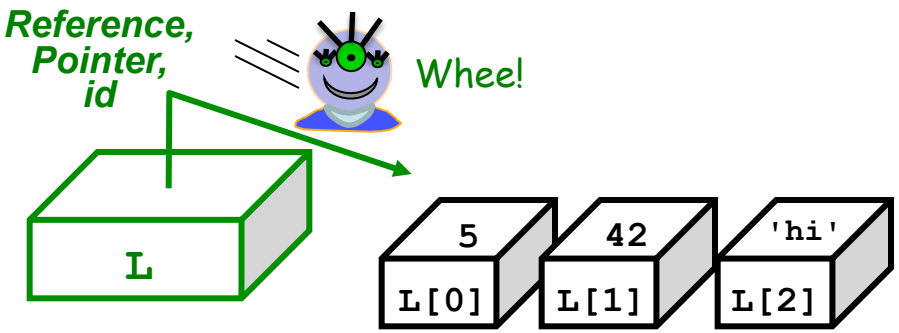
`float`

`string`

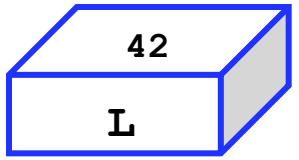
`bool`

`int`

`L = [5,42,'hi']`

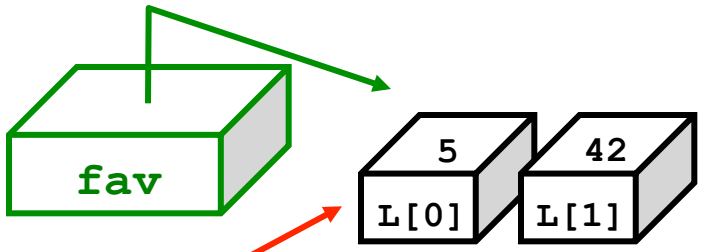


`L = 42`

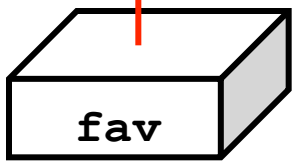


Passing ***lists*** by value…

```
def main()
    """ calls conform2 """
    print " Welcome to Conformity, Inc. "
    fav = [ 7, 11 ]
    conform2(fav)
    print " My favorite numbers are", fav
```



```
def conform2(fav)
    """ sets all of fav to 42 """
    fav[0] = 42
    fav[1] = 42
```



The **reference** is copied!

can change data elsewhere!

The conclusion

You can change **the contents of lists** in functions that take those lists as input.

(actually, lists or any mutable objects)

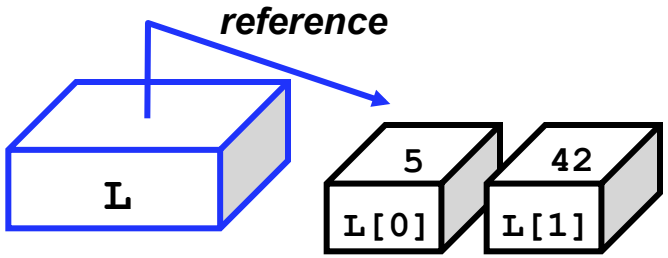
Those changes will be visible **everywhere**

(immutable objects are safe, however)

Lists vs. Dictionaries

If I had a dictionary, I guess
I could look up what it was!

Lists are not perfect…



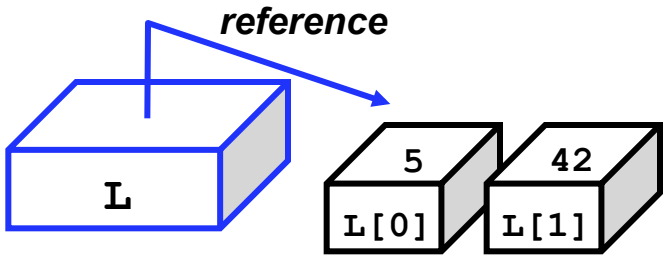
Lists vs. Dictionaries

If I had a dictionary, I guess
I could look up what it was!

Lists are not perfect…

You can't choose what to name data.

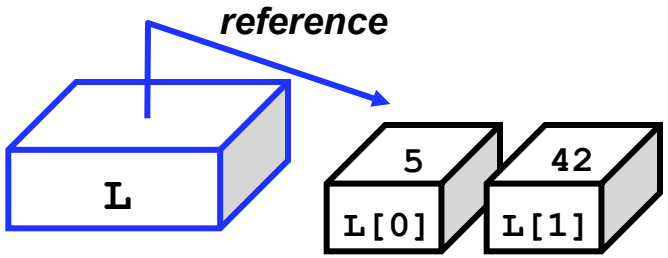
`L[0], L[1], …`



Lists vs. Dictionaries

If I had a dictionary, I guess
I could look up what it was!

Lists are not perfect…



You can't choose what to name data.

```
L[0], L[1], …
```

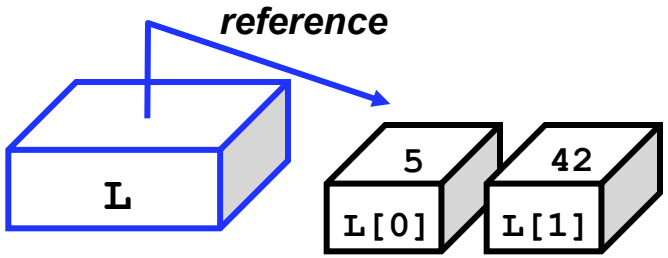
You have to start at `0`.

```
L[1988] = 'dragon'
L[1989] = 'snake'
```

Lists vs. Dictionaries

If I had a dictionary, I guess
I could look up what it was!

Lists are not perfect…



You can't choose what to name data.

```
L[0], L[1], …
```

You have to start at 0.

```
L[1988] = 'dragon'
L[1989] = 'snake'
```

Some operations can be slow for big lists …

```
if 'dragon' in L:
```

Lists vs. Dictionaries

In Python a ***dictionary*** is a set of **`key`** - **`value`** pairs.

```
>>> d = {}
>>> d[1988] = 'dragon'
>>> d[1989] = 'snake'
>>> d
{1988: 'dragon', 1989: 'snake'}
>>> d[1988]
'dragon'
>>> d[1987]
key error
```

It's a list where the index can be ***any immutable-type key***.

Lists vs. Dictionaries

In Python a ***dictionary*** is a set of **`key`** - **`value`** pairs.

```
>>> d = {}
>>> d[1988] = 'dragon'
>>> d[1989] = 'snake'
>>> d
{1988: 'dragon', 1989: 'snake'}
>>> d[1988]
'dragon'
>>> d[1987]
key error
```

creates an empty dictionary, `d`

`1988` is the key
`'dragon'` is the value

`1989` is the key
`'snake'` is the value

Anyone seen this before?

Retrieve data as with lists…

or almost !

It's a list where the index can be ***any immutable-type key***.

More on dictionaries

Dictionaries have lots of built-in methods:

```
>>> d = {1988: 'dragon', 1989: 'snake'}
>>> d.keys()
[ 1989, 1988 ]
>>> d.has_key( 1988 )
True
>>> d.has_key( 1969 )
False
>>> d.pop( 1988 )
'dragon'
```

get all keys

check if a key is present

delete a key
(and its value)

A challenge…

```
prov = { 'BC': 0, 'AB': 0, … }


def provinceChallenge( prov ):
    """ prov is a dictionary of Canada's provinces
        -- the challenge is to name them all! """

    while 0 in prov.values():
        guess = raw_input("Name a province: ")
        if prov.has_key( guess ) == False:
            print 'Try again...'
        elif prov[guess] == 0:
            print 'Yes!'
            prov[guess] += 1
        else:
            print 'Already guessed...'

    print 'Phew!'
```

help?!

Name:

“Quiz”

1. Change this code so that it keeps track of you ***how many times*** you've guessed each item.
2. Then change it so it keeps track of all incorrect guesses in a single dictionary entry

```
def provinceChallenge( prov ):

   while 0 in prov.values():
      guess = raw_input("Guess: ")

      if prov.has_key( guess ) == False:
         print 'Try again...'


      elif prov[guess] == 0:
         print 'Yes!'
         prov[guess] += 1


      else:
         print 'Already guessed...'
```

- first, count guesses
- then, track incorrect provinces

```
def provinceChallenge( prov ):

    while '0' in prov.values():
        guess = raw_input("Guess: ")
        if prov.has_key( guess ) == False:
            print 'Try again...'




        elif prov[guess] == 0:
            print 'Yes!'
            prov[guess] += 1


        else:
            print 'Already guessed...'
```

Mutable vs. immutable objects

Lists are ***mutable*** objects.

(So are dictionaries.)

```
>>> L = [2,1,3]
>>> L.sort()
>>> L
[1,2,3]
```

no return value

L has changed.

Strings are ***immutable*** objects.

```
>>> s = 'string'
>>> s.replace('st','')
'ring'
>>> s
'string'
```

returns a NEW string

s has NOT changed

Why Sort?

Sorting Algorithms

the *fruit flies* of complexity theory...

Checksort

check permutations until sorted

List of length n

c	a	b

Check:

3 1 2
3 2 1
1 2 3
1 3 2
2 1 3
2 3 1

"bogosort"

SlowSort !

pull # s from a hat
with replacement…

List of length n

c	a	b

Slow:

1 1 1
1 1 2
1 1 3
1 2 1
1 2 2
1 2 3
1 3 1
1 3 2
1 3 3
2 1 1
2 1 2
2 1 3
2 2 1
2 2 2
2 2 3
2 3 1
2 3 2
2 3 3
3 1 1
3 1 2
3 1 3

Sorting Algorithms, In Python

```
def checkSort( L ):
    ''' sort the list by permuting it and checking for sorted
        returns a sorted version of L.  Does not modify L'''
    # keep a list with the current permutation
    currentPermList = range(len(L))
    sortedList = copy(L)
    while not isSorted(sortedList):
        updatePermList(currentPermList)
        for i in range(len(L)):
            sortedList[i] = L[currentPermList[i]]
    return sortedList

def isSorted( L ):
    ''' check to see if a list is sorted '''
    if len(L) == 0:
        return True
    for i in range(len(L)-1):
        if L[i+1] < L[i]:
            return False
    return True
```

Sorting Algorithms: Can we do better?

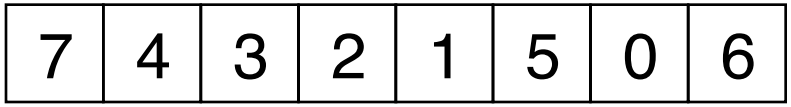
Yes!

Fun for all ages

Sorting Algorithms

The fruit flies of complexity theory...

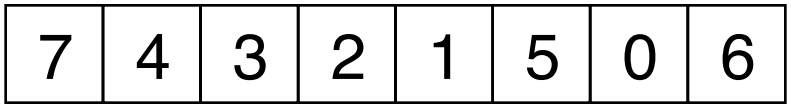
Minsort
choose the min each iteration



An algorithm of a similar sort...

Insertion Sort

insert elements in order to the left



vs. minsort ?

better than $O(N^2)$?