CS 3813/718 Spring 2014

Advanced Programming

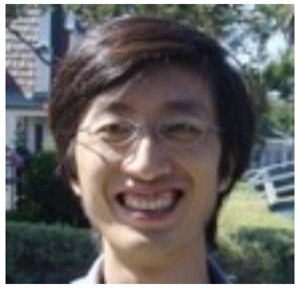
Professor Liang Huang

huang@cs.qc.cuny.edu

Course Homepage:
http://acl.cs.qc.edu/~lhuang/advprg

Logistics

- Coordinates: MW 9:15-10:30am   SB A135B
- Personnel
  - Instructor   Prof. Liang Huang   huang@cs
  - TA   Dr. Lemao Liu   lemaoliu@gmail.com
  - Admin   Xiuyi Huang   xiuyi@cs
  - for questions, always email ***both*** instructor and TA!



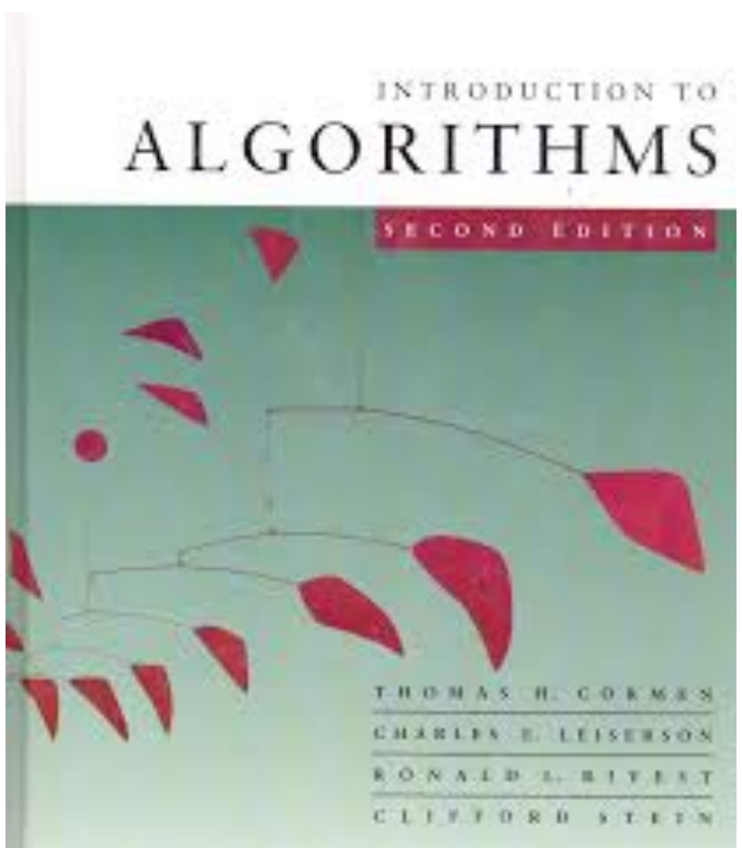
- Office Hours (subject to change!)
  - MW 10:40-11:15am   SB A227
  - additional office hours available before quizzes/exams

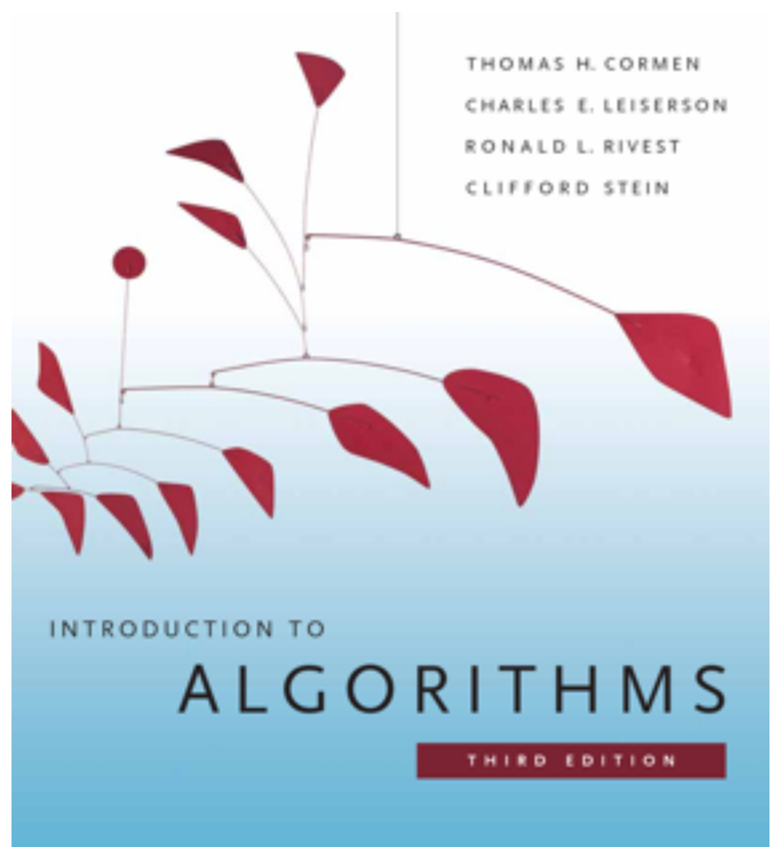
Academic Integrity

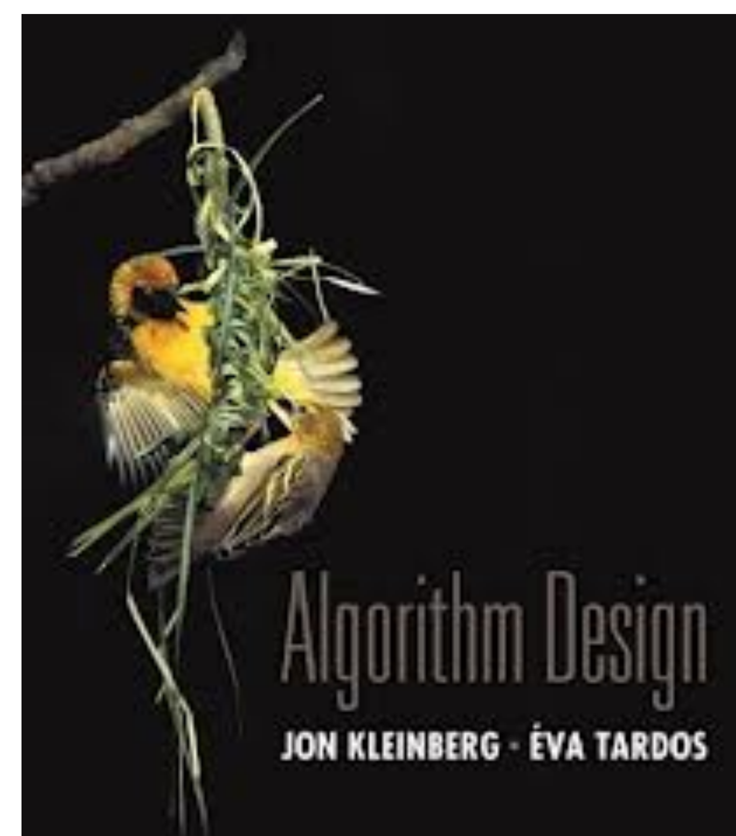
- an automatic F if you’re caught on any of these:
  - copying another person’s code for HW/Project
  - copying code from online resources for HW/Project
  - discussions with others for take-home quiz/exams
  - cheating during quizzes/exams
  - any other cheating behavior defined by University
- catching cheating is easier than you thought! :-)
- I will report every single case to the University

Textbooks

- Main Text
  - *CLRS, Introduction to Algorithms, 3rd edi. (2nd edi. is also fine)*
- Additional Textbooks
  - *Kleinberg and Tardos, Algorithm Design*
  - *How to Think Like a Computer Scientist: Learning Python*







This course is about...

- Implementation of some of the most important algorithms
  - review of data structures and algorithms
  - efficient python implementations
- Preparation for Industrial Interviews and Your Career
  - if you get an A in this course, I guarantee you a position in Google, Facebook, Amazon, or Microsoft
  - if you get an A- => summer internship w/ them
- Preparation for Programming Contests
  - many HW questions are taken from contest problems

This course is NOT about...

- programming
  - I assume you're fluent in C++ and Java
  - we will use Python but that's *not* the focus of this course
- data structures
  - I assume you're good in data structures

Tentative Schedule

1. Python Tutorial, Basic Data Structures and Sorting
   - HW 1 out on T 1/28: divide & conquer, qsort/qselect, msort, BSTs
   - Quiz 1 on W 2/5 => check if you have enough prerequisite background
   - HW 2 out on M 2/17: heaps & heapsort, hash, balanced BSTs
2. Dynamic Programming
   - HW 3 out on M 2/24 (simple chain DP); HW4 out on M 3/3 (tree DP)
   - Midterm on M 3/17
3. Graph Algorithms
   - HW 5 out on M 3/24 (DFS/BFS, SCCs)
   - Quiz 2 on M 4/9 (right before spring break)
   - HW 6 out on M 4/28 (Dijkstra, Kruskal, Prim)

Grades

- Programming Homework: 5% x 6 = 30%
  - electronic submission; work individually: only high-level discussions are OK, and you have to declare who you discussed with
  - grading is mostly black-box: you should follow strict I/O formats!
  - please work on Linux or Mac; we do not provide Windows support
  - late penalty: you can submit only one HW late for 24 hours.
- Quizzes: 10% x 2 = 20%   Midterm: 20%   Final Project: 20%
- Class Participation: 10%
  - asking/answering questions in class
  - catching/fixing bugs in slides/exams/hw & other suggestions

Why Python?

- Because it's easy and great fun!
  - only 15 years old, yet very popular now
    - a wide-range of applications, esp. in AI and Web
  - extremely easy to learn
    - many schools have shifted their intro-courses to Python
  - fast to write
    - much shorter code compared to C, C++, and Java
  - easy to read and maintain
    - more English-like syntax and a smaller semantic-gap

On to Python...

"Hello, World"

- C

```
#include <stdio.h>

int main(int argc, char ** argv)
{
   printf(“Hello, World!\n”);
}
```

- Java

```
public class Hello
{
   public static void main(String argv[])
   {
      System.out.println(“Hello, World!”);
   }
}
```

- now in Python

```
print “Hello, World!”
```

Printing an Array

```
void print_array(char* a[], int len)
{
    int i;
    for (i = 0; i < len; i++)
    {
        printf("%s\n", a[i]);
    }
}
```

has to specify `len`,
and only for one type (`char*`)

C

Python

```
for element in list:
    print element
```

only indentations
no { ... } blocks!

```
for ... in ... :
    ...
```

no C-style for-loops!

~~`for (i = 0; i < 10; i++)`~~

or even simpler:

```
print list
```

Reversing an Array

```
static int[] reverse_array(int a[])
{
   int [] temp = new int[ a.length ];
   for (int i = 0; i < len; i++)
   {
      temp [i] = a [a.length - i - 1];
   }
   return temp;
}
```

Java

Python

```
def rev(a):
   if a == []:
      return []
   else:
      return rev(a[1:]) + [a[0]]
```

def ...(...):
...

no need to specify argument and return types! python will figure it out. (dynamically typed)

a without a[0]

singleton list

or even simpler:

```
a.reverse()
```

← built-in list-processing function

Quick-sort

```java
public void sort(int low, int high)
{
    if (low >= high) return;
    int p = partition(low, high);
    sort(low, p);
    sort(p + 1, high);
}

void swap(int i, int j)
{
    int temp = a[i];
    a[i] = a[j];
    a[j] = temp;
}
```

```java
int partition(int low, int high)
{
    int pivot = a[low];
    int i = low - 1;
    int j = high + 1;
    while (i < j)
    {
        i++; while (a[i] < pivot) i++;
        j--; while (a[j] > pivot) j--;
        if (i < j) swap(i, j);
    }
    return j;
}
```

Java

Python

```python
def sort(a):
    if a == []:
        return []
    else:
        pivot = a[0]
        left  = [x for x in a if x < pivot ]
        right = [x for x in a[1:] if x >= pivot]
        return  sort(left) + [pivot] +  sort(right)
```

$$\{x \mid x \in a,\ \ x < pivot\}$$

smaller semantic-gap!

how about `return [sort(left)] + [pivot] + [sort(right)]` *got an error??*

Python is...

- a scripting language (strong in text-processing)
  - interpreted, like Perl, but much more elegant
- a very high-level language (closer to human semantics)
  - almost like pseudo-code!
- procedural (like C, Pascal, Basic, and many more)
- but also object-oriented (like C++ and Java)
- and even functional! (like ML/OCaml, LISP/Scheme, Haskell, etc.)
- from today, you should use Python for everything
  - not just for scripting, but for serious coding!

Let’s take a closer look...

Python Interpreter

- Three ways to run a Python program

1. Interactive
   - like DrJava

```
>>> for i in range(5):
...     print i,
...
0 1 2 3 4
```

2. (default) save to a file, say, `foo.py`
   - in command-line: `python foo.py`
3. add a special line pointing to the default interpreter
   - add `#!/usr/bin/env python` to the top of `foo.py`
   - make `foo.py` executable (`chmod +x foo.py`)
   - in the command-line: `./foo.py`

The right version of Python

- we will use the latest version 2.7 (e.g. 2.7.3)
  - Python 3.x is a very different experimental branch...
  - your default machine is "`cs12.cs.qc.cuny.edu`", where your default "`python`" is already 2.7
  - or you can install 2.7 on your own mac/windows
    - TA will help you with installations and versions

```
bash-2.0$ python
Python 2.7.1 (#1, Jan 22 2010, 18:59:00)
[GCC 3.3 20030304 (Apple Computer, Inc. build 1495)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

```
[<lhuang@Mac OS X:~>] which python
/Library/Frameworks/Python.framework/Versions/2.7/bin/python
```

Basic Python Syntax

Numbers and Strings

- like Java, Python has built-in (atomic) types
  - numbers (`int`, `float`), `bool`, `string`, `list`, etc.
  - numeric operators: `+ - * / ** %`

```
>>> a = 5
>>> b = 3
>>> type (5)
<type 'int'>
>>> a += 4
>>> a
9
```

```
>>> c = 1.5
>>> 5/2
2
>>> 5/2.
2.5
>>> 5 ** 2
25
```

```
>>> s = “hey”
>>> s + “ guys”
'hey guys'
>>> len(s)
3
>>> s[0]
'h'
>>> s[-1]
'y'
```

no `i++` or `++i`

```
>>> from __future__ import division
>>> 5/2
2.5
```

recommended!

Assignments and Comparisons

```
>>> a = b = 0
>>> a
0
>>> b
0

>>> a, b = 3, 5
>>> a + b
8
>>> (a, b) = (3, 5)
>>> a + b
>>> 8
>>> a, b = b, a
(swap)
```

```
>>> a = b = 0
>>> a == b
True
>>> type (3 == 5)
<type 'bool'>
>>> "my" == 'my'
True

>>> (1, 2) == (1, 2)
True

>>> 1, 2 == 1, 2
???
(1, False, 2)
```

for loops and range()

- for always iterates through a list or sequence

```
>>> sum = 0
>>> for i in range(10):
...     sum += i
...
>>> print sum
45

>>> for word in ["welcome", "to", "python"]:
...     print word,
...
welcome to python

>>> range(5), range(4,6), range(1,7,2)
([0, 1, 2, 3, 4], [4, 5], [1, 3, 5])
```

Java 1.5

```
foreach (String word : words)
    System.out.println(word)
```

`while` loops

- very similar to `while` in Java and C
  - but be careful
    - `in` behaves differently in `for` and `while`
  - `break` statement, same as in Java/C

```
>>> a, b = 0, 1
>>> while b <= 5:
...         print b
...         a, b = b, a+b
...
1
1
2
3
5
```

simultaneous assignment

fibonacci series

Conditionals

```
>>> if x < 10 and x >= 0:
...     print x, "is a digit"
...
>>> False and False or True
True
>>> not True
False
```

```
>>> if 4 > 5:
...     print "foo"
... else:
...     print "bar"
...
bar
```

```
>>> print “foo” if 4 > 5 else “bar”
...
>>> bar
```

conditional expr since Python 2.5

C/Java `printf( (4>5)? ”foo” : “bar”);`

if … elif … else

```
>>> a = "foo"
>>> if a in ["blue", "yellow", "red"]:
...       print a + " is a color"
... else:
...       if a in ["US", "China"]:
...             print a + " is a country"
...       else:
...             print "I don't know what”, a, “is!"
...
I don't know what foo is!
```

```
>>> if a in ...:
...       print ...
... elif a in ...:
...       print ...
... else:
...       print ...
```

C/Java

```
switch (a) {
    case “blue”:
    case “yellow”:
    case “red”:
        print ...; break;
    case “US”:
    case “China”:
        print ...; break;
    else:
        print ...;
}
```

break, continue and else

- break and continue borrowed from C/Java
- special else in loops
  - when loop terminated *normally* (i.e., not by break)
  - very handy in testing a set of properties

```
>>> for n in range(2, 10):
...     for x in range(2, n):
...         if n % x == 0:
...             break
...     else:
...         print n,
...
```

prime numbers

|| func(n)

C/Java

```
for (n=2; n<10; n++) {
    good = true;
    for (x=2; x<n; x++)
        if (n % x == 0) {
            good = false;
            break;
        }
    if (good)                if (x==n)
        printf("%d ", n);
}
```

Defining a Function `def`

- no type declarations needed! wow!
  - Python will figure it out at run-time
    - you get a run-time error for type violation
      - well, Python does not have a compile-error at all

```
>>> def fact(n):
...     if n == 0:
...         return 1
...     else:
...         return n * fact(n-1)
...
>>> fact(4)
24
```

Fibonacci Revisited

```
>>> a, b = 0, 1
>>> while b <= 5:
...         print b
...         a, b = b, a+b
...
1
1
2
3
5
```

```
def fib(n):
   if n <= 1:
      return n
   else:
      return fib (n-1) + fib (n-2)
```

conceptually cleaner, but much slower!

```
>>> fib(5)
5
>>> fib(6)
8
```

Default Values

```
>>> def add(a, L=[]):
...     return L + [a]
...
>>> add(1)
[1]

>>> add(1,1)
error!

>>> add(add(1))
[[1]]

>>> add(add(1), add(1))
???
[1, [1]]
```

lists are heterogenous!

Approaches to Typing

- ✓ strongly typed: types are strictly enforced. no implicit type conversion
- weakly typed: not strictly enforced
- statically typed: type-checking done at compile-time
- ✓ dynamically typed: types are inferred at runtime

	weak	strong
static	C, C++	Java, Pascal
dynamic	Perl, VB	Python, OCaml

Lists

heterogeneous variable-sized array

```
a = [1,'python', [2,'4']]
```

Basic List Operations

- length, subscript, and slicing

```
>>> a = [1,'python', [2,'4']]
>>> len(a)
3
>>> a[2][1]
'4'
>>> a[3]
IndexError!
>>> a[-2]
'python'
>>> a[1:2]
['python']
```

```
>>> a[0:3:2]
[1, [2, '4']]

>>> a[:-1]
[1, 'python']

>>> a[0:3:]
[1, 'python', [2, '4']]

>>> a[0::2]
[1, [2, '4']]

>>> a[::]
[1, 'python', [2, '4']]

>>> a[:]
[1, 'python', [2, '4']]
```

+, extend, +=, append

- extend (+=) and append mutates the list!

```
>>> a = [1,'python', [2,'4']]
>>> a + [2]
[1, 'python', [2, '4'], 2]
>>> a.extend([2, 3])
>>> a
[1, 'python', [2, '4'], 2, 3]
```

same as `a += [2, 3]`

```
>>> a.append('5')
>>> a
[1, 'python', [2, '4'], 2, 3, '5']
>>> a[2].append('xtra')
>>> a
[1, 'python', [2, '4', 'xtra'], 2, 3, '5']
```

Comparison and Reference

- as in Java, comparing built-in types is by value
  - by contrast, comparing objects is by reference

```
>>> [1, '2'] == [1, '2']
True
>>> a = b = [1, '2']
>>> a == b
True
>>> a is b
True
>>> b [1] = 5
>>> a
[1, 5]
>>> a = 4
>>> b
[1, 5]
>>> a is b
>>> False
```

```
>>> c = b [:]
>>> c
[1, 5]
>>> c == b
True
>>> c is b
False
```

slicing gets a shallow copy

```
>>> b[:0] = [2]
>>> b
[2, 1, 5]
>>> b[1:3]=[]
>>> b
[2]
```

insertion

deletion

```
>>> a = b
>>> b += [1]
>>> a is b
True
```

`a += b` means `a.extend(b)` NOT `a = a + b` !!

List Comprehension

```
>>> a = [1, 5, 2, 3, 4 , 6]
>>> [x*2 for x in a]
[2, 10, 4, 6, 8, 12]

>>> [x for x in a if \
... len( [y for y in a if y < x] ) == 3 ]
[4]

>>> a = range(2,10)
>>> [x*x for x in a if \
... [y for y in a if y < x and (x % y == 0)] == [] ]
???
[4, 9, 25, 49]
```

4th smallest element

square of prime numbers

List Comprehensions

```
>>> vec = [2, 4, 6]
>>> [[x,x**2] for x in vec]
[[2, 4], [4, 16], [6, 36]]

>>> [x, x**2 for x in vec]
SyntaxError: invalid syntax

>>> [(x, x**2) for x in vec]
[(2, 4), (4, 16), (6, 36)]

>>> vec1 = [2, 4, 6]
>>> vec2 = [4, 3, -9]
>>> [x*y for x in vec1 for y in vec2]
[8, 6, -18, 16, 12, -36, 24, 18, -54]
```

(cross product)

```
>>> [x+y for x in vec1 for y in vec2]
[6, 5, -7, 8, 7, -5, 10, 9, -3]
```

should use zip instead!

```
>>> [vec1[i]*vec2[i] for i in range(len(vec1))]
[8, 12, -54]
```

(dot product)

Strings

sequence of characters

Basic String Operations

- join, split, strip
- upper(), lower()

```
>>> s = " this is  a  python course. \n"
>>> words = s.split()
>>> words
['this', 'is', 'a', 'python', 'course.']
>>> s.strip()
'this is  a  python course.'
>>> " ".join(words)
'this is a python course.'
>>> "; ".join(words).split("; ")
['this', 'is', 'a', 'python', 'course.']
>>> s.upper()
' THIS IS  A  PYTHON COURSE. \n'
```

http://docs.python.org/lib/string-methods.html

Basic Search/Replace in String

```
>>> "this is a course".find("is")
2
>>> "this is a course".find("is a")
5
>>> "this is a course".find("is at")
-1

>>> "this is a course".replace("is", "was")
'thwas was a course'
>>> "this is a course".replace(" is", " was")
'this was a course'
>>> "this is a course".replace("was", "were")
'this is a course'
```

these operations are much faster than regexps!

String Formatting

```
>>> print “%.2f%%” % 97.2363
97.24%

>>> s = '%s has %03d quote types.' % ("Python", 2)
>>> print s
Python has 002 quote types.
```

Sequence Types

- list, tuple, str;   buffer, xrange, unicode

Operation	Result
`x in s`	`True` if an item of s is equal to x, else `False`
`x not in s`	`False` if an item of s is equal to x, else `True`
`s + t`	the concatenation of s and t
`s * n , n * s`	n shallow copies of s concatenated
`s[i]`	i'th item of s, origin 0
`s[i:j]`	slice of s from i to j
`s[i:j:k]`	slice of s from i to j with step k
`len(s)`	length of s
`min(s)`	smallest item of s
`max(s)`	largest item of s

```
>>> lists = [[]] * 3
>>> lists
[[], [], []]
```

```
>>> lists[0].append(3)
>>> lists
[[3], [3], [3]]
```

the tricky *

```
>>> [1, 2] * 3
[1, 2, 1, 2, 1, 2]

>>> [] * 3
[]

>>> [[]] * 3
[[], [], []]

>>> a = [3]
>>> b = a * 3
>>> b
[3, 3, 3]

>>> a[0] = 4
>>> b
[3, 3, 3]
```

```
>>> a = [[3]]
>>> b = a * 3
>>> b
[[3], [3], [3]]

>>> a[0][0] = 4
[[4], [4], [4]]

>>> a[0] = 5
>>> b
[[4], [4], [4]]

>>> a = [3]
>>> b = [a] * 3
>>> b
[[3], [3], [3]]
```

```
>>> a[0] = 4
>>> b
[[4], [4], [4]]

>>> b[1] = 5
>>> b
[[4], 5, [4]]

>>> b[0] += [2]
[[4, 2], 5, [4, 2]]

>>> " " * 3
"   "

>>> "_ " * 3
"_ _ _ "
```

Pythonic Styles

- do not write ... when you can write ...

do not write ...	when you can write ...
`for key in d.keys():`	`for key in d:`
`if d.has_key(key):`	`if key in d:`

`i = 0`
`for x in a:`
`    ...`
`    i += 1`
	`for i, x in enumerate(a):`

`a[0:len(a) - i]`	`a[:-i]`
`for line in \` `    sys.stdin.readlines():`	`for line in sys.stdin:`

`for x in a:`
`    print x,`
`print`
	`print " ".join(map(str, a))`

`s = ""`
`for i in range(lev):`
`    s += " "`
`print s`
	`print " " * lev`

Tuples

immutable lists

Tuples and Equality

- caveat: singleton tuple
- `==`, `is`, `is not`

```
a += (1,2)   # new copy
a += [1,2]   # in-place
```

```
>>> (1, 'a')
(1, 'a')
>>> (1)
1
>>> [1]
[1]
>>> (1,)
(1,)
>>> [1,]
[1]
>>> (5) + (6)
11
>>> (5,)+ (6,)
(5, 6)
```

```
>>> 1, 2 == 1, 2
(1, False, 2)
>>> (1, 2) == (1, 2)
True
>>> (1, 2) is (1, 2)
False
>>> "ab" is "ab"
True
>>> [1] is [1]
False
>>> 1 is 1
True
>>> True is True
True
```

Comparison

- between the same type: "lexicographical"
- between different types: arbitrary
- `cmp()`: three-way <, >, ==
  - C: `strcmp(s, t)`, Java: `a.compareTo(b)`

```
>>> (1, 'ab') < (1, 'ac')
True
>>> (1, ) < (1, 'ac')
True
>>> [1] < [1, 'ac']
True
>>> 1 < True
False
>>> True < 1
False
```

```
>>> [1] < [1, 2] < [1, 3]
True
>>> [1] == [1,] == [1.0]
True
>>> cmp ( (1, ), (1, 2) )
-1
>>> cmp ( (1, ), (1, ) )
0
>>> cmp ( (1, 2), (1, ) )
1
```

enumerate

```
>>> words = ['this', 'is', 'python']
>>> i = 0
>>> for word in words:
...     i += 1
...     print i, word
...
1 this
2 is
3 python

>>> for i, word in enumerate(words):
...     print i+1, word
...
```

- how to enumerate two lists/tuples simultaneously?

zip and _

```
>>> a = [1, 2]
>>> b = ['a', 'b']

>>> zip (a,b)
[(1, 'a'), (2, 'b')]

>>> zip(a,b,a)
[(1, 'a', 1), (2, 'b', 2)]

>>> zip ([1], b)
[(1, 'a')]
```

```
>>> a = ['p', 'q']; b = [[2, 3], [5, 6]]
>>> for i, (x, [_, y]) in enumerate(zip(a, b)):
...     print i, x, y
...
0 p 3
1 q 6
```

zip and list comprehensions

```
>>> vec1 = [2, 4, 6]
>>> vec2 = [4, 3, -9]
>>> [(x, y) for x in vec1 for y in vec2]
[(2, 4), (2, 3), (2, -9), (4, 4), (4, 3), (4, -9), (6, 4),
(6, 3), (6, -9)]

>>> [(vec1[i], vec2[i]) for i in range(len(vec1))]
[(2, 4), (4, 3), (6, -9)]

>>> sum([vec1[i]*vec2[i] for i in range(len(vec1))]
-34

>>> sum([x*y for (x,y) in zip(vec1, vec2)])
-34

>>> sum([v[0]*v[1] for v in zip(vec1, vec2)])
-34
```

how to implement zip?

binary zip: easy

```
>>> def myzip(a,b):
...     if a == [] or b == []:
...             return []
...     return [(a[0], b[0])] + myzip(a[1:], b[1:])
...

>>> myzip([1,2], ['a','b'])
[(1, 'a'), (2, 'b')]
>>> myzip([1,2], ['b'])
[(1, 'b')]
```

how to deal with arbitrarily many arguments?

Dictionaries

(heterogeneous) hash maps

Constructing Dicts

- key : value pairs

```
>>> d = {'a': 1, 'b': 2, 'c': 1}
>>> d['b']
2
>>> d['b'] = 3
>>> d['b']
3
>>> d['e']
KeyError!
>>> d.has_key('a')
True
>>> 'a' in d
True
>>> d.keys()
['a', 'c', 'b']
>>> d.values()
[1, 1, 3]
```

Other Constructions

- zipping, list comprehension, keyword argument
- dump to a list of tuples

```
>>> d = {'a': 1, 'b': 2, 'c': 1}
>>> keys = ['b', 'c', 'a']
>>> values = [2,  1,   1 ]
>>> e = dict (zip (keys, values))
>>> d == e
True
>>> d.items()
[('a', 1), ('c', 1), ('b', 2)]

>>> f = dict( [(x, x**2) for x in values] )
>>> f
{1: 1, 2: 4}

>>> g = dict(a=1, b=2, c=1)
>>> g == d
True
```

default values

- counting frequencies

```
>>> def incr(d, key):
...     if key not in d:
...             d[key] = 1
...     else:
...             d[key] += 1
...

>>> def incr(d, key):
...     d[key] = d.get(key, 0) + 1
...

>>> incr(d, 'z')
>>> d
{'a': 1, 'c': 1, 'b': 2, 'z': 1}
>>> incr(d, 'b')
>>> d
{'a': 1, 'c': 1, 'b': 3, 'z': 1}
```

defaultdict

- best feature introduced in Python 2.5

```
>>> from collections import defaultdict
>>> d = defaultdict(int)
>>> d['a']
0
>>> d['b'] += 1
>>> d
{'a': 0, 'b': 1}

>>> d = defaultdict(list)
>>> d['b'] += [1]
>>> d
{'b': [1]}

>>> d = defaultdict(lambda : <expr>)
```

Mapping Type

Operation	Result
`len(`a`)`	the number of items in a
a`[`k`]`	the item of a with key k
a`[`k`] = `v	set a`[`k`]` to v
`del `a`[`k`]`	remove a`[`k`]` from a
a`.clear()`	remove all items from `a`
a`.copy()`	a (shallow) copy of `a`
a`.has_key(`k`)`	`True` if a has a key k, else `False`
k `in` a	Equivalent to a.has_key(k)
k `not in` a	Equivalent to `not` a.has_key(k)
a`.items()`	a copy of a's list of (*key*, *value*) pairs
a`.values()`	a copy of a's list of values
a`.get(`k`[, `x`])`	a`[`k`]` if k `in` a, else x
a`.setdefault(`k`[, `x`])`	a`[`k`]` if k `in` a, else x (also setting it)
a`.pop(`k`[, `x`])`	a`[`k`]` if k `in` a, else x (and remove k)

defaultdict behaves like setdefault, not get (following STL)

http://docs.python.org/lib/typesmapping.html

Sets

identity maps, unordered collection

Sets

- [] for lists, () for tuples, {} for dicts, and {} for sets (2.7)
- construction from lists, tuples, dicts (keys), and strs
- in, not in, add, remove

```
>>> a = {1, 2}
a
>> set([1, 2])
>>> a = set((1,2))
>>> a
set([1, 2])
>>> b = set([1,2])
>>> a == b
True
>>> c = set({1:'a', 2:'b'})
>>> c
set([1, 2])
```

```
>>> type({})
'dict'   # not set!

>>> a = set()
>>> 1 in a
False
>>> a.add(1)
>>> a.add('b')
>>> a
set([1, 'b'])
>>> a.remove(1)
>>> a
set(['b'])
```

Set Operations

- union, intersection, difference, is_subset, etc..

demo

```
>>> a = set('abracadabra')
>>> b = set('alacazam')
>>> a
set(['a', 'r', 'b', 'c', 'd'])
>>> a - b
set(['r', 'd', 'b'])
>>> a | b
set(['a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'])
>>> a & b
set(['a', 'c'])
>>> a ^ b
set(['r', 'd', 'b', 'm', 'z', 'l'])
>>> a |= b
>>> a
set(['a', 'c', 'b', 'd', 'm', 'l', 'r', 'z'])
```

set and frozenset type

Operation	Equivalent	Result
`len(s)`		cardinality of set *s*
`x in s`		test *x* for membership in *s*
`x not in s`		test *x* for non-membership in *s*
`s.issubset(t)`	`s <= t`	test whether every element in *s* is in *t*
`s.issuperset(t)`	`s >= t`	test whether every element in *t* is in *s*
`s.union(t)`	$s \mid t$	new set with elements from both *s* and *t*
`s.intersection(t)`	*s* & *t*	new set with elements common to *s* and *t*
`s.difference(t)`	*s* - *t*	new set with elements in *s* but not in *t*
`s.symmetric_difference(t)`	*s* ^ *t*	new set with elements in either *s* or *t* but not both
`s.copy()`		new set with a shallow copy of *s*
`s.update(t)`	$s \mid= t$	return set *s* with elements added from *t*
`s.intersection_update(t)`	*s* &= *t*	return set *s* keeping only elements also found in *t*
`s.difference_update(t)`	*s* -= *t*	return set *s* after removing elements found in *t*
`s.symmetric_difference_update(t)`	*s* ^= *t*	return set *s* with elements from *s* or *t* but not both
`s.add(x)`		add element *x* to set *s*
`s.remove(x)`		remove *x* from set *s*; raises KeyError if not present
`s.discard(x)`		removes *x* from set *s* if present
`s.pop()`		remove and return an arbitrary element from *s*; rais
`s.clear()`		remove all elements from set *s*

Basic `import` and I/O

`import` and I/O

- similar to `import` in Java
- File I/O much easier than Java

demo

```
import sys
for line in sys.stdin:
   print line.split()
```

or

```
from sys import *
for line in stdin:
   print line.split()
```

```
import System;
```

Java

```
import System.*;
```

```
>>> f = open("my.in", "rt")
>>> g = open("my.out", "wt")
>>> for line in f:
...     print >> g, line,
... g.close()
```

to read a line:

```
line = f.readline()
```

to read all the lines:

```
lines = f.readlines()
```

file copy

note this comma!

`import` and `__main__`

foo.py

- multiple source files (modules)
  - C: `#include "my.h"`
  - Java: `import My`
- demo

demo

```
def pp(a):
    print " ".join(a)

if __name__ == "__main__":
    from sys import *
    a = stdin.readline()
    pp (a.split())
```

- handy for debugging

```
>>> import foo
>>> pp([1,2,3])
1 2 3
```

interactive

Quiz

- Palindromes

abcba

  - read in a string from standard input, and print `True` if it is a palindrome, print `False` if otherwise

```
def palindrome(s):
    if  len(s) <= 1 :
        return True
    return  s[0] == s[-1] and palindrome( s[1:-1] )

if __name__ == '__main__':

    import sys
    s = sys.stdin.readline(). strip()
    print palindrome(s)
```

Functional Programming

map and filter

- intuition: function as data
- we have already seen functional programming a lot!
  - list comprehension, custom comparison function

```
map(f, a)

filter(p, a)
```

```
[ f(x) for x in a ]

[ x for x in a if p(x) ]
```

```
map(f, filter(p, a))
```

```
[ f(x) for x in a if p(x) ]
```

```
>>> map(int, ['1','2'])
[1, 2]
>>> " ".join(map(str, [1,2]))
1 2
```

```
>>> def is_even(x):
...     return x % 2 == 0
...
>>> filter(is_even, [-1, 0])
[0]
```

demo

lambda

- map/filter in one line for custom functions?
  - "anonymous inline function"
- borrowed from LISP, Scheme, ML, OCaml

```
>>> f = lambda x: x*2
>>> f(1)
2
>>> map (lambda x: x**2, [1, 2])
[1, 4]
>>> filter (lambda x: x > 0, [-1, 1])
[1]
>>> g = lambda x,y : x+y
>>> g(5,6)
11
>>> map (lambda (x,y): x+y, [(1,2), (3,4)])
[3, 7]
```

demo

more on lambda

```
>>> f = lambda : "good!"
>>> f
<function <lambda> at 0x381730>
>>> f()
'good!'
```

lazy evaluation

```
>>> a = [5, 1, 2, 6, 4]
>>> a.sort(lambda x,y : y - x)
>>> a
[6, 5, 4, 2, 1]
```

custom comparison

```
>>> a = defaultdict(lambda : 5)
>>> a[1]
5
>>> a = defaultdict(lambda : defaultdict(int))
>>> a[1]['b']
0
```

demo

Basic Sorting

```
>>> a = [5, 2, 3, 1, 4]
>>> a.sort()
>>> print a
[1, 2, 3, 4, 5]

>>> a = [5, 2, 3, 1, 4]
>>> a.sort(reverse=True)
>>> a
[5, 4, 3, 2, 1]

>>> a = [5, 2, 3, 1, 4]
>>> a.sort()
>>> a.reverse()
>>> a
[5, 4, 3, 2, 1]
```

sort() is in-place,
but sorted() returns new copy

```
>>> a = [5, 2, 3, 1, 4]
>>> sorted(a)
[1, 2, 3, 4, 5]
>>> a
[5, 2, 3, 1, 4]
```

Built-in and custom cmp

```
>>> a = [5, 2, 3, 1, 4]
>>> def mycmp(a, b):
        return b-a

>>> sorted(a, mycmp)
[5, 4, 3, 2, 1]

>>> sorted(a, lambda x,y: y-x)
[5, 4, 3, 2, 1]

>>> a = zip([1,2,3], [6,4,5])
>>> a.sort(lambda (_,y1), (__, y2): y1-y2)
>>> a
[(2, 4), (3, 5), (1, 6)]
>>> a.sort(lambda (_,y1), (_, y2): y1-y2)
SyntaxError: duplicate argument '_' in function definition
```

demo

Sorting by Keys or Key mappings

```
>>> a = "This is a test string from Andrew".split()
>>> a.sort(key=str.lower)
>>> a
['a', 'Andrew', 'from', 'is', 'string', 'test', 'This']

>>> import operator
>>> L = [('c', 2), ('d', 1), ('a', 4), ('b', 3), ('b', 1)]

>>> L.sort(key=operator.itemgetter(1))
>>> L
[('d', 1), ('b', 1), ('c', 2), ('b', 3), ('a', 4)]

>>> sorted(L, key=operator.itemgetter(1, 0))
[('b', 1), ('d', 1), ('c', 2), ('b', 3), ('a', 4)]

>>> operator.itemgetter(1,0)((1, 2, 3))
(2, 1)
```

demo

sort by two keys

lambda for key mappings

- you can use lambda for both custom cmp and key map

```
>>> a = "This is a test string from Andrew".split()
>>> a.sort(lambda x, y: cmp(x.lower(), y.lower()))
>>> a
['a', 'Andrew', 'from', 'is', 'string', 'test', 'This']

>>> a.sort(key=lambda x: x.lower())

>>> L = [('c', 2), ('d', 1), ('a', 4), ('b', 3), ('b', 1)]

>>> L.sort(key=lambda (_, y): y)
>>> L
[('d', 1), ('b', 1), ('c', 2), ('b', 3), ('a', 4)]

>>> sorted(L, key=lambda (x, y): (y, x))
[('b', 1), ('d', 1), ('c', 2), ('b', 3), ('a', 4)]
```

Decorate-Sort-Undecorate

```
>>> words = "This is a test string from Andrew.".split()

>>> deco = [ (word.lower(), i, word) for i, word in \
... enumerate(words) ]

>>> deco.sort()

>>> new_words = [ word for _, _, word in deco ]

>>> print new_words
['a', 'Andrew.', 'from', 'is', 'string', 'test', 'This']
```

demo

- Most General
- Faster than custom cmp (or custom key map) -- why?
- stable sort (by supplying index)

Sorting: Summary

- 3 ways: key mapping, custom cmp function, decoration
- decoration is most general, key mapping least general
- decoration is faster than key mapping & cmp function
  - decoration only needs O(n) key mappings
  - other two need O(nlogn) key mappings -- or $O(n^2)$ for insertsort
  - real difference when key mapping is slow
- decoration is stable

Memoized Recursion v1

- Fibonacci revisited

```
def fib(n):
    a, b = 1, 1
    for _ in range(n-1):
        a, b = b, a+b
    return b
```

```
def fib(n):
    if n <= 1:
        return n
    else:
        return fib (n-1) + fib (n-2)
```

```
fibs = {0:1, 1:1}
def fib(n):
    if n in fibs:
        return fibs[n]
    fibs[n] = fib(n-1) + fib(n-2)
    return fibs[n]
```

can we get rid of the global variable?

Memoized Recursion v2

- Fibonacci revisited

```
def fib(n):
    a, b = 1, 1
    for _ in range(n-1):
        a, b = b, a+b
    return b
```

```
def fib(n):
    if n <= 1:
        return n
    else:
        return fib (n-1) + fib (n-2)
```

```
def fib(n, fibs={0:1, 1:1}):
    if n not in fibs:
        fibs[n] = fib(n-1, fibs) + fib(n-2, fibs)
    return fibs[n]
```

Memoized Recursion v3

- Fibonacci revisited

```
def fib(n):
    a, b = 1, 1
    for _ in range(n-1):
        a, b = b, a+b
    return b
```

```
>>> fib(3)
1 {1: 1}
0 {0: 1, 1: 1}
2 {0: 1, 1: 1, 2: 2}
3 {0: 1, 1: 1, 2: 2, 3: 3}
3
>>> fib(2)
2
>>> print fibs
Error!
```

draw the tree!

```
def fib(n, fibs={0:1, 1:1}):
    if n not in fibs:
        fibs[n] = fib(n-1) + fib(n-2)
#       print n, fibs
    return fibs[n]
```

the `fibs` variable has a weird closure!! feature or bug?

most people think it's a bug, but Python inventor argues it's a feature.

Memoized Recursion v4

- Fibonacci revisited

```
def fib(n):
    a, b = 1, 1
    for _ in range(n-1):
        a, b = b, a+b
    return b
```

```
>>> fib(4)
{0: 1, 1: 1, 2: 2}
{0: 1, 1: 1, 2: 2, 3: 3}
{0: 1, 1: 1, 2: 2, 3: 3, 4: 5}
5
>>> fib(3)
{0: 1, 1: 1, 2: 2}
{0: 1, 1: 1, 2: 2, 3: 3}
3
```

```
def fib(n, fibs=None):
    if fibs is None:
        fibs = {0:1, 1:1}
    if n not in fibs:
        fibs[n] = fib(n-1, fibs) + fib(n-2, fibs)
#       print n, fibs
    return fibs[n]
```

this is so far the cleanest way to avoid this bug.

Mutable types are not hashable

- mutables: list, dict, set
- immutables: tuple, string, int, float, frozenset, ...
  - only recursively immutable objects are hashable
- your own class objects are hashable (but be careful...)

```
>>> {{1}: 2}
TypeError: unhashable type: 'set'

>>> {{1:2}: 2}
TypeError: unhashable type: 'dict'

>>> {frozenset([1]): 2}
{frozenset([1]): 2}

>>> {frozenset([1, [2]]): 2}
TypeError: unhashable type: 'list'
```

Implementation / Speed

- lists, tuples, and strings
  - random access: O(1)
  - insertion/deletion/in: O(n)
- dict and set
  - in/random access: almost O(1)
  - insertion/deletion: almost O(1)
  - but no linear ordering!
- try %timeit in ipython!

```
$ ipython

In [22]: b = set(range(100))

In [23]: %timeit 78 in b
10000000 loops, best of 3: 58 ns per loop

In [24]: %timeit 8 in b
10000000 loops, best of 3: 58.2 ns per loop

In [25]: c = range(100)

In [26]: %timeit 78 in c
1000000 loops, best of 3: 1.45 us per loop

In [28]: %timeit 0 in c
10000000 loops, best of 3: 52 ns per loop
```

Pythonic Styles

- do not write ... when you can write ...

<table>
<tr><td><pre><code>for key in d.keys():</code></pre></td><td><pre><code>for key in d:</code></pre></td></tr>
<tr><td><pre><code>if d.has_key(key):</code></pre></td><td><pre><code>if key in d:</code></pre></td></tr>
<tr><td><pre><code>i = 0
for x in a:
    ...
    i += 1</code></pre></td><td><pre><code>for i, x in enumerate(a):</code></pre></td></tr>
<tr><td><pre><code>a[0:len(a) - i]</code></pre></td><td><pre><code>a[:-i]</code></pre></td></tr>
<tr><td><pre><code>for line in \
    sys.stdin.readlines():</code></pre></td><td><pre><code>for line in sys.stdin:</code></pre></td></tr>
<tr><td><pre><code>for x in a:
    print x,
print</code></pre></td><td><pre><code>print " ".join(map(str, a))</code></pre></td></tr>
<tr><td><pre><code>s = ""
for i in range(lev):
    s += " "
print s</code></pre></td><td><pre><code>print " " * lev</code></pre></td></tr>
</table>