A Quick Tutorial to Python

Most of the slides are taken from

http://www.cis.upenn.edu/~lhuang3/cse399-python/schedule.html

“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
```

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

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

```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)
```

smaller semantic-gap!

Python basics

- Basic types: various numbers
- Basic flow control: if/then/else, for/while
- Indentation for code blocks

Numbers and Strings

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

```
>>> a = 5
>>> b = 3
>>> a + b
8
>>> type (5)
<type 'int'>
>>> a += 4
>>> a
9
```

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

```
>>> c = 1.5
>>> c
1.5
>>> type (c+a)
<type 'float'>
>>> 5/2
2
>>> 5/2.
2.5
>>> 5 ** 2
25
```

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

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 = 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

```
>>> while i in range(5):
...     print i,
...
???

>>> while True:
...     print 1
...     break
...
1
```

Conditionals `if`

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

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

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
- 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

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)
        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
```

Basic Data Structures

- C++'s STL containers:
  - `vector, set, map, deque`, etc..
- Java's generics collections:
  - `Vector, ArrayList, HashMap`, etc.

Python has …

- **List**: variable-sized array
  - `[1, 2, 3]`
- **String**: sequence of characters
  - `’123’`
- **Tuple**: Immutable list
  - `(1, 2, 3)`
- **Dictionary**: hash map
  - `{1: ‘a’, 2: ‘b’, 3:’c’}`
- **Set**: hash set
  - `set([1, 2, 3])`

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

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 largest element

square of prime numbers

Strings

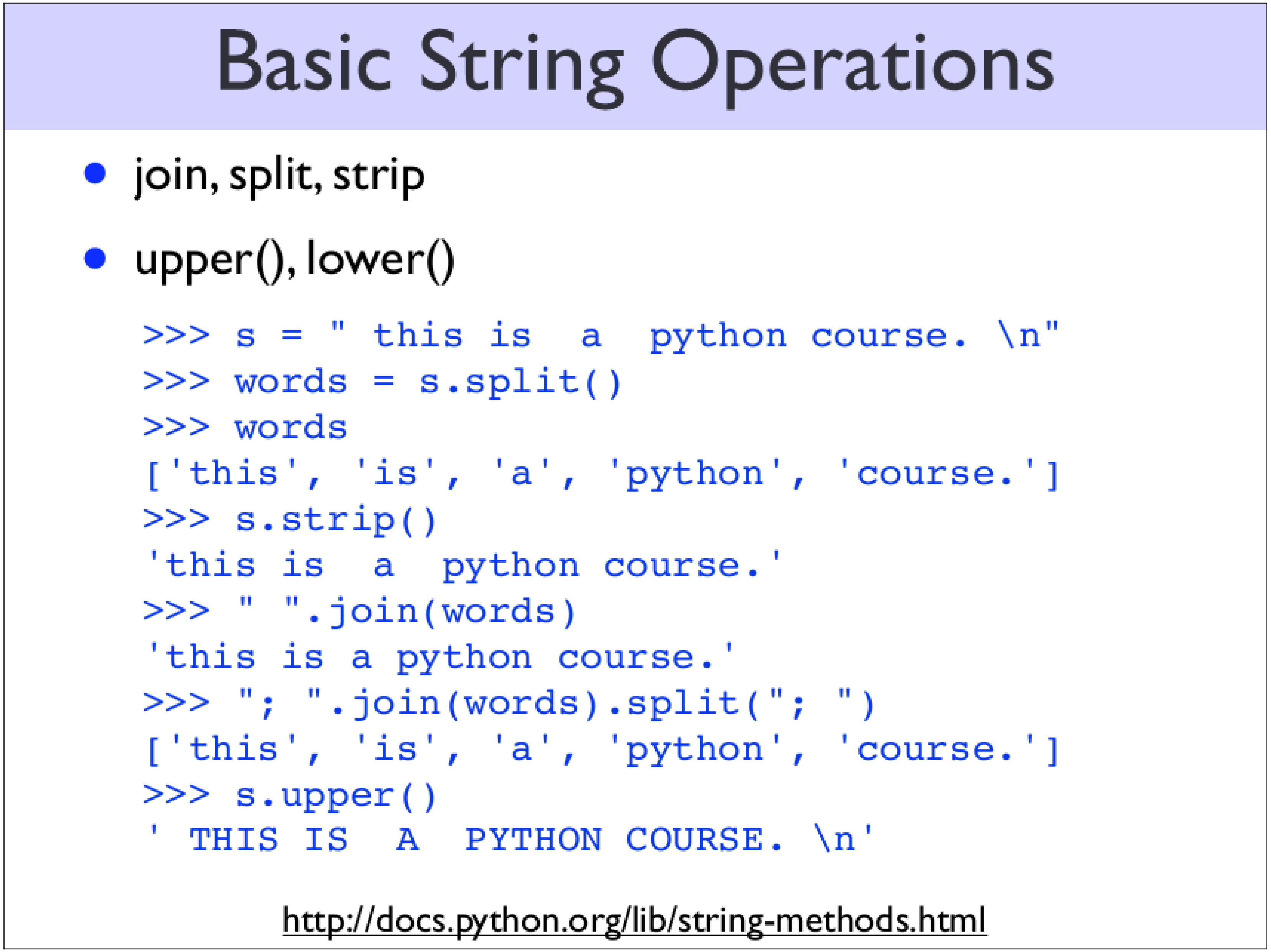
sequence of characters

String Literals

- single quotes and double quotes; escape chars
- strings are immutable!

```
>>> 'spam eggs'
'spam eggs'
>>> 'doesn't'
SyntaxError!
>>> 'doesn\'t'
"doesn't"
>>> "doesn't"
"doesn't"
>>> "doesn"t"
SyntaxError!
>>> s = "aa"
>>> s[0] ='b'
TypeError!
```

```
>>> s = "a\nb"
>>> s
'a\nb'
>>> print s
a
b
>>> "\"Yes,\" he said."
'"Yes," he said.'
>>> s = '"Isn\'t," she said.'
>>> s
'"Isn\'t," she said.'
>>> print s
"Isn't," she said.
```



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

Tuples

immutable lists

Tuples and Equality

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

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

```
>>> (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
```

Dictionaries

(heterogeneous) hash maps

Constructing Dicts

- key : value pairs

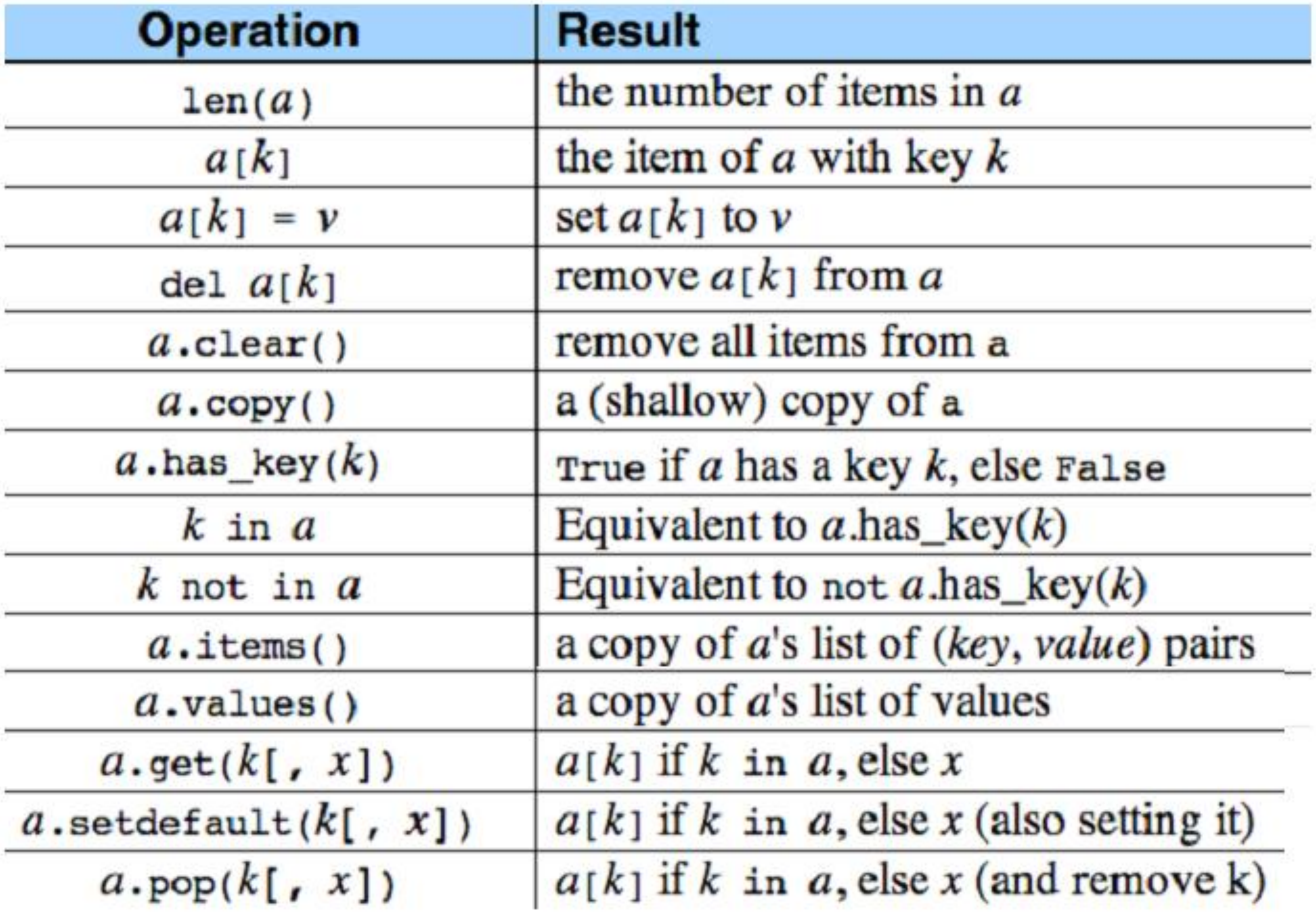
```
>>> d = {'a': 1, 'b': 2, 'c': 1}
>>> d['b']
2
>>> d['b'] = 3
>>> d['b']
3
>>> d['e']
KeyError!
>>> d.has_key('a')
True
>>> 'a' in d
True
>>> d.keys()
['a', 'c', 'b']
>>> d.values()
[1, 1, 3]
```

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
```

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)

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

Sets

identity maps, unordered collections

Construction and Operations

- sets do not have a special syntactic form
  - unlike [] for lists, () for tuples and {} for dicts
- construction from lists, tuples, dicts (keys), and strs
- in, not in, add, remove

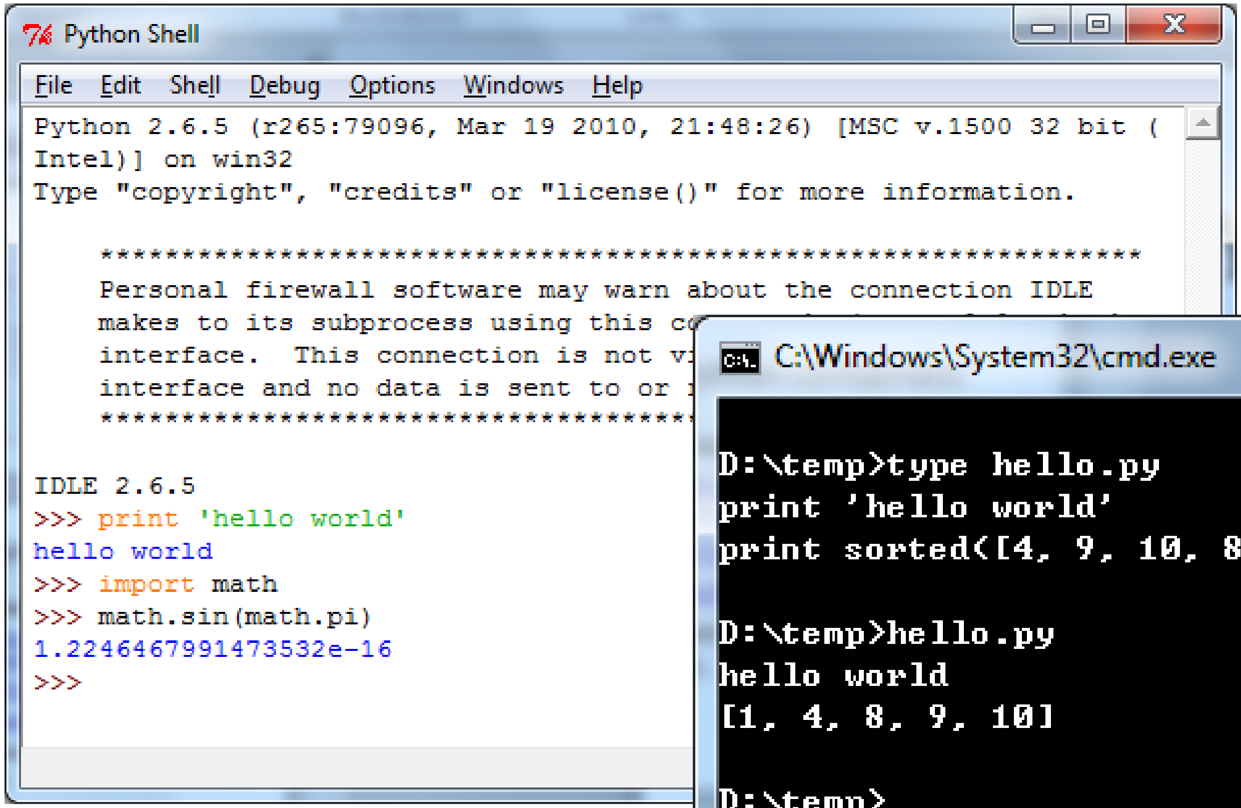
```
>>> a = set((1,2))
>>> a
set([1, 2])
>>> b = set([1,2])
>>> a == b
True
>>> c = set({1:'a', 2:'b'})
>>> c
set([1, 2])
```

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

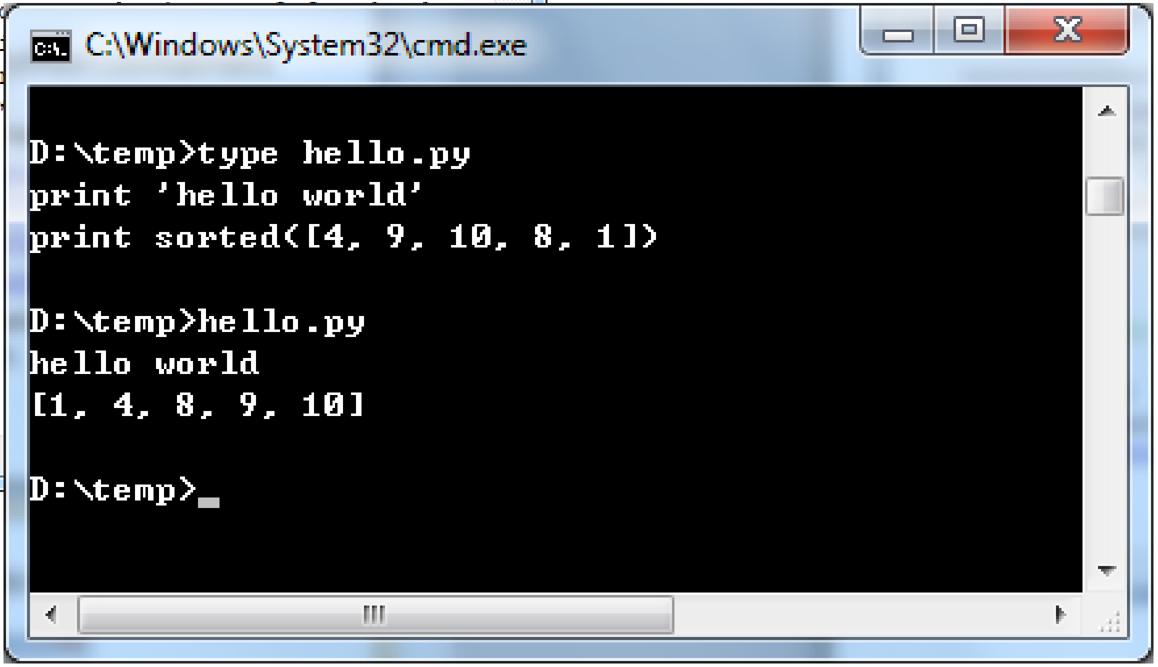
More on Python containers

- Study the first three handouts on
  - http://www.cis.upenn.edu/~lhuang3/cse399-python/schedule.html
- Study the official Python tutorial
- Code Like a Pythonista: Idiomatic Python
- Python Howto: **sorting** and **functional programming**

Ways to run Python



```
Python 2.6.5 (r265:79096, Mar 19 2010, 21:48:26) [MSC v.1500 32 bit (
Intel)] on win32
Type "copyright", "credits" or "license()" for more information.

    ****************************************************************
    Personal firewall software may warn about the connection IDLE
    makes to its subprocess using this co
    interface.  This connection is not vi
    interface and no data is sent to or r
    ************************************

IDLE 2.6.5
>>> print 'hello world'
hello world
>>> import math
>>> math.sin(math.pi)
1.2246467991473532e-16
>>>
```

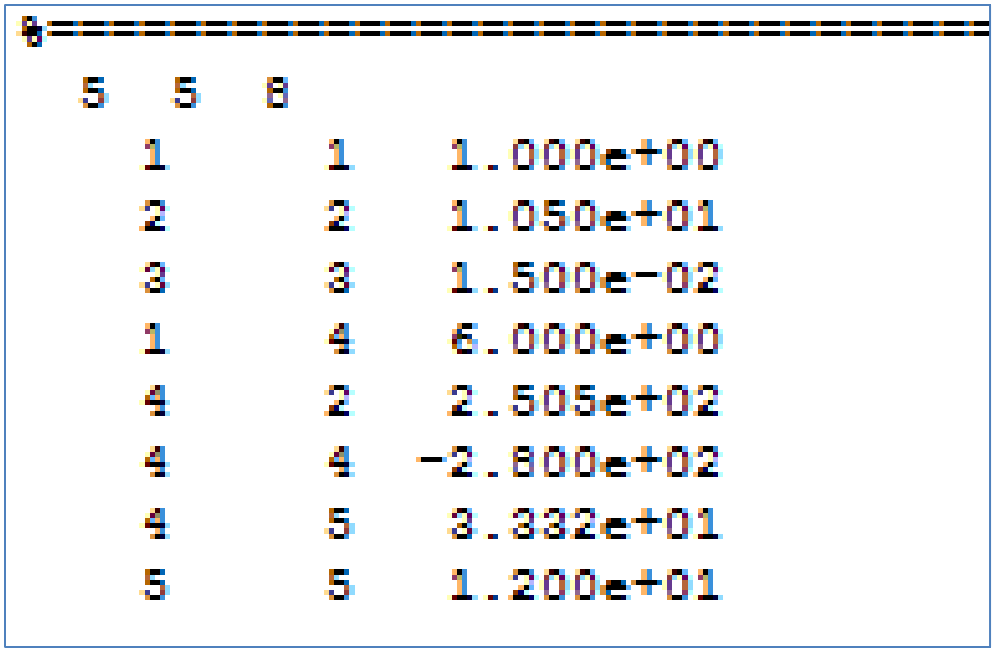


```
D:\temp>type hello.py
print 'hello world'
print sorted([4, 9, 10, 8, 1])

D:\temp>hello.py
hello world
[1, 4, 8, 9, 10]

D:\temp>_
```

A Complete Example: sparse matrix format conversion

Input Format:
(Matrix Market Sparse)

```
1      0      0      6      0
0    10.5     0      0      0
0      0    .015     0      0
0   250.5     0    -280   33.32
0      0      0      0    12
```



```
%================================
  5    5    8
       1         1     1.000e+00
       2         2     1.050e+01
       3         3     1.500e-02
       1         4     6.000e+00
       4         2     2.505e+02
       4         4    -2.800e+02
       4         5     3.332e+01
       5         5     1.200e+01
```

Output Format:

```
1:1   4:6
2:10.5
3:0.015
2:250.5   4:-280    5:33.32
5:12
```

```
# rows is a dictionary with
# key as row number
# value as non-zero entries, which are also dictionaries
rows = {}

# read all lines in the input files as *a list of strings*
lines = open(input_file, 'r').readlines()
firstLine = True
for line in lines:
    line = line.strip()
    # skip empty and comment lines
    if len(line) ==0 or line.startswith('%'):
        continue
    else:
        if firstLine:
            firstLine = False
            continue
        else:
            # split the line
            s = line.split()
            # read the three numbers in the line
            i, j, v = int(s[0]), int(s[1]), float(s[2])
            if i not in rows:
                rows[i] = {}
            rows[i][j] = v
```

```
rows =
{1: {1: 1.0, 4: 6.0},
 2: {2: 10.5},
 3: {3: 0.015},
 4: {2: 250.5, 4: -280.0, 5: 33.32},
 5: {5: 12.0}}
```

```python
# open a file for writing
out = open(output_file, 'w')

# for each row
for row in range(1, len(rows)+1):
    first = True
    # for non-zero columns (in ascending order)
    for col in sorted(rows[row].keys()):
        if first:
            first = False
        else:
            out.write(' ')
        # write col:value pairs
        out.write('%d:%g'%(col, rows[row][col]))
    out.write('\n')
out.close()
```

```
output_file:

1:1 4:6
2:10.5
3:0.015
2:250.5 4:-280 5:33.32
5:12
```