CSE 399-004, Spring 2006
Python Programming
Handout 2 (lectures 3 & 4)

Restructuring the Course
- only 4 assignments!
- grades
  - HW: 50%
  - quiz 15% and final 25%
  - participation: 10%
- schedule
  - functional programming will be touched earlier
  - more like “linguistics of programming languages”
  - some applications and special topics in the 2nd half

Optional Projects
- students do a mini-research
- on extensions and applications of Python, or
- comparing Python with other languages
- produce
  - a website of annotated resources
  - a tutorial (written form)
  - a 20 minute presentation (teaching)
- grading
  - [-5, 15] extra credit points

Project Ideas
- transfer learning by comparison (==, <, >):
  - Python vs. Ruby (syntax/semantics)
  - Python vs. OCaml (list proc., functional)
  - Python vs. Scheme (list proc., functional)
  - Python vs. Perl (scripting)
- interface
  - between Python and C (pyrex)
  - between Python and Java (Jython)
- applications and extensions
  - Python and the Web
  - Python and AI

Today
- import and __main__
- Tuples
- Dictionaries
- Sets

import and __main__
- multiple source files (modules)
  - C: #include "my.h"
  - Java: import My
- demo
- handy for debugging
- one problem in HW 2 will use this method
  - I specify the signature, instead of I/O format
  - when grading, we will just import your program

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

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

immutable lists

Tuples and Equality

• caveat: singleton tuple

• \(==\), \(\text{is}\), \(\text{is not}\)

\[
\begin{align*}
\text{>>> (1, 'a')} & \text{ (1, 'a')} \\
\text{>>> (1)} & \text{ 1} \\
\text{>>> [1]} & \text{ [1]} \\
\text{>>> (1,)} & \text{ (1,)} \\
\text{>>> [1,]} & \text{ [1,]} \\
\text{>>> (5) + (6)} & \text{ 11} \\
\text{>>> (5,) + (6,)} & \text{ (5, 6)} \\
\text{>>> (1, 2) == (1, 2)} & \text{ True} \\
\text{>>> (1, 2) is (1, 2)} & \text{ False} \\
\text{>>> "ab" is "ab"} & \text{ True} \\
\text{>>> [1] is [1]} & \text{ False} \\
\text{>>> 1 is 1} & \text{ True} \\
\text{>>> True is True} & \text{ True}
\end{align*}
\]

Comparison

• between the same type: “lexicographical”

• between different types: arbitrary

• \text{cmp()}\): three-way \(<\), \(>\), \(==\)

• C: \text{strcmp(s, t)}, Java: \text{a.compareTo(b)}

\[
\begin{align*}
\text{>>> (1, 'ab') < (1, 'ac')} & \text{ True} \\
\text{>>> (1,) < (1, 'ac')} & \text{ True} \\
\text{>>> [1] < [1, 'ac']} & \text{ True} \\
\text{>>> 1 < True} & \text{ False} \\
\text{>>> True < 1} & \text{ False}
\end{align*}
\]

\[
\begin{align*}
\text{>>> [1] < [1, 2] < [1, 3]} & \text{ True} \\
\text{>>> [1] == [1] == [1.0]} & \text{ True} \\
\text{>>> cmp ( (1, ), (1, 2) )} & \text{ -1} \\
\text{>>> cmp ( (1, ), (1, ) )} & \text{ 0} \\
\text{>>> cmp ( (1, 2), (1, ) )} & \text{ 1}
\end{align*}
\]

enumerate

\[
\begin{align*}
\text{>>> words = ['this', 'is', 'python']} \\
\text{>>> i = 0} \\
\text{>>> for word in words:} \\
\text{... i += 1} \\
\text{... print i, word} \\
\text{... 1 this} \\
\text{... 2 is} \\
\text{... 3 python} \\
\text{>>> for i, word in enumerate(words):} \\
\text{... print i+1, word} \\
\text{... 2 this} \\
\text{... 3 is} \\
\text{... 4 python}
\end{align*}
\]

• how to enumerate two lists/tuples simultaneously?

zip and _

\[
\begin{align*}
\text{>>> a = [1, 2]} \\
\text{>>> b = ['a', 'b']} \\
\text{>>> zip (a,b)} & \text{ [(1, 'a'), (2, 'b')]} \\
\text{>>> zip(a,b,a)} & \text{ [(1, 'a', 1), (2, 'b', 2)]} \\
\text{>>> zip ([1], b)} & \text{ [(1, 'a')]} \\
\text{>>> a = [(1, 2, 3)], (4, [5, 6])} \\
\text{>>> for i, (x, [_, y]) in enumerate(a):} \\
\text{... print i, x, y} \\
\text{... 0 1 3} \\
\text{... 1 4 6}
\end{align*}
\]

how to implement zip?

binary zip: easy

\[
\begin{align*}
\text{>>> def myzip(a,b):} \\
\text{... if a == [] or b == []:} \\
\text{... return []} \\
\text{... return [(a[0], b[0])] + myzip(a[1:], b[1:])} \\
\text{...}
\end{align*}
\]

\[
\begin{align*}
\text{>>> myzip([1], [1, 'b'])} & \text{ [(1, 'b')]} \\
\text{>>> myzip([1,2], ['a', 'b'])} & \text{ [[1, 'a'], [2, 'b']]} \\
\text{>>> myzip([1,2], ['b'])} & \text{ [[1, 'b']]} \\
\end{align*}
\]

how to deal with arbitrarily many arguments?
Social Network

Alice ——— Rachel

Bob ——— Tom ——— Leo

```python
>>> people = ['A', 'B', 'R', 'T', 'L']
>>> friends = [('A', 'B'), ('B', 'T'), ('T', 'R'), ('R', 'A'), ('L', 'T')]

>>> def intro(person):
...     for another in people:
...         if (person, another) in friends or (another, person) in friends:
...             print person, "this is", another

>>> intro(another)
```

Corrected Version of DFS

Alice ——— Rachel

Bob ——— Tom ——— Leo

```python
>>> people = ['A', 'B', 'R', 'T', 'L']
>>> friends = [('A', 'B'), ('B', 'T'), ('T', 'R'), ('R', 'A'), ('L', 'T')]

>>> already = []

>>> def intro(person):
...     already.append(person)
...     for another in people:
...         if another not in already:
...             if (person, another) in friends or (another, person) in friends:
...                 print person, "this is", another
...                 intro(another)

>>> intro(another)
```

Python for Fun

- excellent book online
  - http://www.ibiblio.org/obp/py4fun/
- ASCII video game
- GUI
- Hanoi, Trees and Search
- SQL
- LISP and Prolog in Python
- Natural Language Processing
- and many more...

Projects

- considered part of the lecture
- will be represented in HW and Final
- final moved to Monday May 1st

```
<table>
<thead>
<tr>
<th>Monday</th>
<th>Wednesday</th>
<th>Friday</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apr. 10-14</td>
<td>Polynomials</td>
<td>HW3 Solutions</td>
</tr>
<tr>
<td>Apr. 17 - 21</td>
<td>Scheme (Vaitics M.)</td>
<td>Jython (Ravi C.)</td>
</tr>
<tr>
<td></td>
<td>OCanal (Matt G.)</td>
<td>Web Applications (Andrew M.)</td>
</tr>
<tr>
<td></td>
<td>HW4 out</td>
<td>Wrap-up/TA Course evaluation</td>
</tr>
</tbody>
</table>

May 1 Final 10-11am
Meyerson Hall B3
```

Netta’s qsort

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

>>> qsort([5,3,4,2,6,1])

[[[[5], 6, [6]], 4, [4]], 3, [[], 3, []]], 5, [[], 6, []]]
```

Netta’s Algorithm

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

>>> qsort([5,3,4,2,6,1])

[[[[5], 6, [6]], 4, [4]], 3, [[], 3, []]], 5, [[], 6, []]]
```

Netta’s qsort

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

>>> qsort([5,3,4,2,6,1])

[[[[5], 6, [6]], 4, [4]], 3, [[], 3, []]], 5, [[], 6, []]]
```

Netta’s Algorithm

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

>>> qsort([5,3,4,2,6,1])

[[[[5], 6, [6]], 4, [4]], 3, [[], 3, []]], 5, [[], 6, []]]
```
Dictionaries

(heterogeneous) hash maps

Constructing Dicts

- key : value pairs
  ```python
  >>> 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
  ```python
  >>> 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

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>len(a)</td>
<td>the number of items in a</td>
</tr>
<tr>
<td>a[k]</td>
<td>the item of a with key k</td>
</tr>
<tr>
<td>a[k] = v</td>
<td>set a[k] to v</td>
</tr>
<tr>
<td>del a[k]</td>
<td>remove a[k] from a</td>
</tr>
<tr>
<td>a.clear()</td>
<td>remove all items from a</td>
</tr>
<tr>
<td>a.copy()</td>
<td>a (shallow) copy of a</td>
</tr>
<tr>
<td>a.has_key(k)</td>
<td>True if a has a key k, else False</td>
</tr>
<tr>
<td>k in a</td>
<td>Equivalent to a.has_key(k)</td>
</tr>
<tr>
<td>k not in a</td>
<td>Equivalent to not a.has_key(k)</td>
</tr>
<tr>
<td>a.items()</td>
<td>a copy of a’s list of (key, value) pairs</td>
</tr>
<tr>
<td>a.values()</td>
<td>a copy of a’s list of values</td>
</tr>
<tr>
<td>a.get(k, x)</td>
<td>a[k] if k in a, else x</td>
</tr>
<tr>
<td>a.setdefault(k, x)</td>
<td>a(k) if k in a, else x (also setting it)</td>
</tr>
<tr>
<td>a.pop(k, x)</td>
<td>a[k] if k in a, else x (and remove k)</td>
</tr>
</tbody>
</table>

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

Implementation

- lists, tuples, and dicts are all implemented by hashing
- strings are implemented as arrays
- lists, tuples, and strings
  - random access: $O(1)$
  - insertion/deletion/in: $O(n)$
- dict
  - in/random access: $O(1)$
  - insertion/deletion: $O(1)$
- no linear ordering
Sorting

Basic Sorting

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

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

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

Built-in and Custom cmp

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

```python
>>> a = [5, 2, 3, 1, 4]
>>> def reverse_numeric(x, y):
...     return y-x
... >>> a.sort(reverse_numeric)
... >>> a
[5, 4, 3, 2, 1]
```

Sorting by Keys

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

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

Decorate-Sort-Undecorate

```python
>>> 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']
```

- Very General
- Faster than custom cmp
- stable sort