

CSE 399-004, Spring 2006

# Python Programming

Handout I (Lectures 1 and 2)

handouts available online at:

[www.seas.upenn.edu/~cse39905/schedule.html](http://www.seas.upenn.edu/~cse39905/schedule.html)

# Logistics

- Personnel

- Instructor      Liang Huang      [lhuang3@cis.upenn.edu](mailto:lhuang3@cis.upenn.edu)
- TA                Bill Kandylas      [kandylas@cis.upenn.edu](mailto:kandylas@cis.upenn.edu)
- Administrator   Jennifer Finley      [jfinley@cis.upenn.edu](mailto:jfinley@cis.upenn.edu)

- Coordinates

- March 13 - April 21 (6 weeks, after spring break)
- Lecture            MW 1-2 PM    Towne 309
- Lab/Recitation    F      1-2 PM    Moore 207
- Office Hours      R      3-4 PM    Levine 565

# Logistics - cont'd

- Homepage [www.seas.upenn.edu/~cse39905/](http://www.seas.upenn.edu/~cse39905/)
  - schedule, syllabus, homework, handouts, etc.
- Newsgroup [upenn.cis.cse399-005](http://upenn.cis.cse399-005)
  - announcements, Q/A
- Course Email [cse39905@seas](mailto:cse39905@seas)
  - to reach both the instructor and the TA
- Blackboard [courseweb.library.upenn.edu](http://courseweb.library.upenn.edu)
  - grades and announcements

# Textbooks (for reference)

- Textbooks

we will not follow any textbook.

- *Dive into Python*

- by Mark Pilgrim

- *How to Think Like a Computer Scientist: Learning Python*

- by Allen B. Downey, Jeffrey Elkner and Chris Meyers

- Tutorials

- (Official) *Python Tutorial*

- by Guido van Rossum (inventor of Python)

- *A Quick, Painless Tutorial on the Python Language*

- by Norm Matloff

# Homework

- 6 weekly programming assignments
  - to be completed individually
  - usually out on Mondays and due on Sundays
  - submit your work through the “turnin” program
  - late policy: 24 hours: 25% off; 48 hours: 50% off; no points afterwards
  - input/output format will be **strictly** enforced
    - there will be sample I/O files online
      - make sure your programs pass them before submission
      - otherwise you will get 0 for this problem

# Grades

- 60% Homework, 10% Quiz, 25% Final Exam, and 5% Participation
- Quiz and Final Exam
  - Both during labs, on Mar. 31 and Apr. 21, respectively
  - closed book but open to one Letter sheet of notes
- Participation
  - Labs are optional but recommended
    - we teach some additional material and help with HW
  - you will be rewarded for catching bugs in course materials (handouts, textbooks, homework, exams)

# Before and After this course

- Before this course
  - CSE 120 (Intro-to-CS with Java): **required**
    - **CSE 399 is not an intro-to-programming course!**
  - CSE 121 (Data Structures with Java): **recommended**
    - **you may take it in parallel**
- After this course
  - CSE 391 (Artificial Intelligence)
  - CIS 530 (Computational Linguistics)
  - and many more to come...

# Course Outline

[www.seas.upenn.edu/~cse39905/schedule.html](http://www.seas.upenn.edu/~cse39905/schedule.html)

- Two parts (3 weeks each, separated by Quiz)
  1. Python Basics
    - Syntax, Control Flow
    - Data Structures (Lists, Dictionaries, Tuples, etc.)
    - Regular Expressions and String Processing
    - File I/O and Exception Handling
  2. Object-Oriented and Functional Programming
    - OOP (Objects, Inheritance, Linked Lists, Trees, etc.)
    - FP (map, filter, reduction, iterator/generator,  $\lambda$ -function)

**On to Python...**

# Why Python?

- Because it's easy and great fun!
  - less than 10 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

# Python is...

- a scripting language (strong in text-processing)
  - interpreted, like Perl, but much more elegant
- a **very** high-level language (closer to human)
  - like Java, unlike C or Assembly
- procedural
  - like C, Pascal, Basic, and many more
- but also object-oriented
  - like C++ and Java
- and even functional! (like ML, Scheme, Haskell, etc.)

# “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!"
```

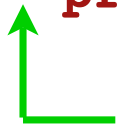
# Printing an Array

```
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

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



only indentations  
no { ... } blocks!

or even simpler:

```
print list
```

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

no C-style for-loops!

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

Python

# 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

```
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)

Python

or even simpler:

a without a[0]      singleton list

a.reverse() ← built-in list-processing function

# Quick-sort

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

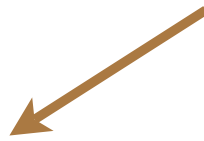
void swap(int i, int j)
{
    int temp = a[i];
    a[i] = a[j];
    a[j] = temp;
}

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

```
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)
```

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



Python

smaller semantic-gap!

**Take a closer look...**

# Python Interpreter

- Three ways to run a Python program

1. Interactive

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

- like DrJava

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

- e.g. `#!/usr/bin/python` at the beginning of `foo.py`
- make `foo.py` executable (`chmod +x foo.py`)
- in command-line: `./foo.py`

# The right version of Python

- like Java, Python is still under active development
- we will use the latest version 2.4 (or 2.4.2)
  - our default machine is “`eniac-1.seas.upenn.edu`”, where default “`python`” is 2.3.4
  - to use the latest python on eniac-1, you can either
    - “`python2.4.2`”, or
    - append one of the following to your “`~/.bashrc`” file:
      - `export PATH=/usr/local/python/2.4.2/:$PATH`
      - `alias python='python2.4.2'`

best solution



demo

# Install your own Python

- alternatively, you can install python 2.4.2 on your own Windows/Mac/Linux machines from [www.python.org](http://www.python.org)
- see “resources” page for instructions [www.seas.upenn.edu/~cse39905/resources.html](http://www.seas.upenn.edu/~cse39905/resources.html)
- we will help with python installations at this Friday’s Lab. you can bring your laptop if you have problems.

```
bash-2.0$ python
```

```
Python 2.4 (#1, Jan 22 2005, 18:59:00)
```

```
[GCC 3.3 20030304 (Apple Computer, Inc. build 1495)] on darwin
```

```
Type "help", "copyright", "credits" or "license" for more information.
```

```
>>>
```

# 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
>>> 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)
```

# Raw Input and `print`

```
>>> a = raw_input("please input a number: ")
please input a number: 6
>>> a
'6'
>>> a = int(raw_input("please input a number: "))
please input a number: 6
>>> a
6
>>> a = int(raw_input("please input a number: "))
please input a number: six
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ValueError: invalid literal for int(): six
```

```
>>> print 5,6; print 7
5 6
7
```

# for loops and range()

- **for** always iterates through a list or sequence

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

```
>>> print sum
45
```

## Java 1.5

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

```
>>> 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])
```

# 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

```
for (n=2; n<10; n++) {
    good = true;
    for (x=2; x<n; x++)
        if (n % x == 0) {
C/Java    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
```

# 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]]
```

# 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, Scheme

# Lecture 2

## Lists, Strings, Files

[www.seas.upenn.edu/~cse39905](http://www.seas.upenn.edu/~cse39905)

# Recap...

- in Lecture 1 we covered
  - background
  - basic Python syntax
    - conditionals, loops, function definitions
    - raw input and print
  - basic Python typing

# Today

- lists
  - operations, address mode, comprehension
- sequence types
- strings
  - join, split, conversion, formatting
- import and standard I/O

# 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]
```

# 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

>>> b[1:] = [3, 4]
>>> b
[1, 3, 4]
>>> b[:0] = b
>>> b
[1, 3, 4, 1, 3, 4]
>>> b[1:3]=[]
>>> b
[1, 1, 3, 4]
```

slicing gets a shallow copy

insertion

deletion

# 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 \
```

4th largest element

```
... 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]
```

square of prime numbers

# Sequence Types

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

Operation	Result
$x \text{ in } s$	True if an item of $s$ is equal to $x$ , else False
$x \text{ not in } s$	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 $k$
$\text{len}(s)$	length of $s$
$\text{min}(s)$	smallest item of $s$
$\text{max}(s)$	largest item of $s$

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

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

# 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'
```

# Basic *import* and I/O

# import and I/O

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

```
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()
```

file copy

note this comma!