Teaching Computer Science with Python

Workshop #4 SIGCSE 2003

John M. Zelle
Wartburg College

Outline

- Why Python?
- Basic Structures
  - I/O
  - Assignment
  - Numbers
  - Strings
  - Functions
  - Decisions
  - Loops
  - Collections
- Software Engineering
  - Modules/Packages
  - Classes
  - Exceptions
  - Library
- Educational Apps
  - Functional Programming
  - GUIs
  - Graphics
  - Internet Programming
  - Databases
  - Op Systems
  - Others
- Python Resources

What is Python?

- Python: A free, portable, dynamically-typed, object-oriented scripting language.
- Combines software engineering features of traditional systems languages with power and flexibility of scripting languages.
- Note: Named after Monty Python's Flying Circus

Why Python?

- Traditional languages (C++, Java) evolved for large-scale programming
  - Emphasis on structure and discipline
  - Simple problems != simple programs
- Scripting languages (Perl, Python, TCL) designed for simplicity and flexibility.
  - Simple problems = simple, elegant solutions
  - More amenable to experimentation and incremental development
- Python: Ideal first language, useful throughout curriculum

First Program (Java Version)

- Assignment: Print "Hello SIGCSE" on screen

```
public class Hello{
    public static void main(String args){
        System.out.println("Hello SIGCSE");
    }
}
```

- Note: Must be in "Hello.java"

First Program (Python Version)

- Assignment: Print "Hello SIGCSE" on screen

```
print "Hello SIGCSE"
```

- Or...

```
def main():
    print "Hello SIGCSE"

main()
```

Running Python Programs

- Hybrid compiled/interpreted architecture
- Options:
  - Start Interpreter from command line (>>>)
    - Type program statements
    - Import script file
  - Start interpreter with file as command line arg
  - Configure filetype to launch interpreter on file
  - Unix pound-bang trick
  - Directly from IDE (IDLE)

"Real" Program: Chaos.py

```
#File: chaos.py
# A simple program illustrating chaotic behavior.

def main():
    print "This program illustrates a chaotic function"
    x = input("Enter a number between 0 and 1: ")
    for i in range(10):
        x = 3.9 * x * (1 - x)
        print x

main()
```

Python Features

- Comment convention "#" to end of line
- Nesting indicated by indentation
- Statements terminated by end of line
  - Explicit continuation with backslash
  - Implicit continuation to match parens
- No variable declarations
- For loop iterates through a sequence

Example Output

```
$ python chaos.py
This program illustrates a chaotic function
Enter a number between 0 and 1: .5
0.975
0.0950625
0.335499922266
0.869464925259
0.442633109113
0.962165255337
0.141972779362
0.4750843862
0.972578927537
0.104009713267
$
```

Basic Output Statement

```
print <expr1>, <expr2>, ..., <exprn>
```

- Notes:
  - Prints expressions on one line
  - Successive values separated by a space
  - Advances to next line (unless comma follows)
  - All Python built-in types have printable reps

Assignment Statements

- Simple Assignment

```
<var> = <expr>
myVar = oldValue * foo + skip
```

- Simultaneous Assignment

```
<var1>, <var2>, ... = <expr1>, <expr2>, ...
a,b = b,a
```

- Assigning Input

```
input(<prompt>)
myVar = input("Enter a number: ")
x,y = input("Enter the coordinates (x,y): ")
```

Example Program: Fibonacci

```
# fibonacci.py
# This program computes the nth Fibonacci number

n = input("Enter value of n ")

cur,prev = 1,1
for i in range(n-2):
    cur,prev = prev+cur,cur

print "The nth Fibonacci number is", cur
```

Teaching Tip: Variable Declarations

- Pluses
  - less code
  - less upfront explanation
  - eliminates "redeclaration" errors
- Minuses
  - typo on LHS of = creates new variable
  - allows variables to change type
- Bottom-line: I prefer dynamic types

Teaching Tip: Indentation as Syntax

- Pluses
  - less code clutter (; and {})
  - eliminates most common syntax errors
  - promotes and teaches proper code layout
- Minuses
  - occasional subtle error from inconsistent spacing
  - will want an indentation-aware editor
- Bottom-line: Good Python editors abound.

  This is my favorite feature.

Numeric Types

- int: Standard 32 bit integer

```
32 -3432 0
```

- long int: Indefinitely long integers

```
32L 9999999999999999
```

- floating-point: Standard double-precision float

```
3.14  2.57e-10  5E210  -3.64e+210
```

- complex: Double precision real and imaginary components

```
2+3j   4.7J  -3.5 + 4.3e-4j
```

- User-defined types (operator overloading)

Numeric Operations

- Builtins

  `+, -, *, /, %, **, abs(), round()`

- Math Library

  `pi, e, sin(), cos(), tan(), log(),`

  `log10(), ceil(), ...`

Example Numeric Program: quadratic.py

```
# quadratic.py
# Program to calculate real roots
#    of a quadratic equation

import math

a, b, c = input("Enter the coefficients (a, b, c): ")

discRoot = math.sqrt(b * b - 4 * a * c)
root1 = (-b + discRoot) / (2 * a)
root2 = (-b - discRoot) / (2 * a)

print "\nThe solutions are:", root1, root2
```

Alternative Imports

```
import math as m
discRt = m.sqrt(b * b - 4 * a * c)

from math import sqrt
discRt = sqrt(b * b - 4 * a * c)

from math import *
y = sqrt(sin(x))

from math import sqrt as squareRoot
y = squareRoot(x)
```

Teaching Tip: Integer Division

- Python follows tradition of C, C++, Java: / is overloaded
- Problem with Dynamic typing

```
average = sum / n
```

- Solution 1: Explicit type conversion

```
average = float(sum) / n
```

- Solution 2: Time Travel

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

Teaching Tip: Compound Operations

- Traditionally, Python did not support ++, --, +=, etc.
- As of Python 2.2 these are included

```
factorial = 1
for factor in range(2,n+1):
    factorial *= factor
```

- Personally prefer to avoid these shortcuts in CS1

String Datatype

- String is an immutable sequence of characters
- Literal delimited by ’ or " or """

```
s1 = ’This is a string’
s2 = "This is another"
s3 = "that’s one alright"
s4 = """This is a long string that
goes across multiple lines.
It will have embedded end of lines"""
```

- Strings are indexed
  - From the left starting at 0 or...
  - From the right using negative indexes
- A character is just a string of length 1

String Operations

```
>>>"Hello, " + " world!"
'Hello, world!'

>>> "Hello" * 3
'HelloHelloHello'

>>> greet = "Hello John"
>>> print greet[0], greet[2], greet[4]
H l o

>>> greet[4:9]
'o Joh'
>>> greet[:5]
'Hello'
>>> greet[6:]
'John'

>>> len(greet)
10
```

Example Program: Month Abbreviation

```
# month.py
#   prints the abbreviation for a month

months = "JanFebMarAprMayJunJulAugSepOctNovDec"

n = input("Enter a month number (1-12): ")
pos = (n-1)*3
monthAbbrev = months[pos:pos+3]

print "The month abbreviation is", monthAbbrev+"."
```

More String Operations

- Interactive input

```
s = raw_input("Enter your name: ")
```

- Looping through a string

```
for ch in name:
    print ch
```

- Type conversion
  - to string

```
>>> str(10)
'10'
```

  - from string

```
>>> eval('10')
10
>>> eval('3 + 4 * 7')
31
```

Standard String Library (string)

```
capitalize(s)  -- upper case first letter
capwords(s)    -- upper case each word
upper(s)       -- upper case every letter
lower(s)       -- lower case every letter

ljust(s, width)   -- left justify in width
center(s, width)  -- center in width
rjust(s, width)   -- right justify in width

count(substring, s)  -- count occurrences
find(s, substring)   -- find first occurrence
rfind(s, substring)  -- find from right end
replace(s, old, new) -- replace first occurrence

strip(s)  -- remove whitespace on both ends
rstrip(s) -- remove whitespace from end
lstrip(s) -- remove whitespace from front

split(s, char)   -- split into list of substrings
join(stringList) -- concatenate list into string
```

Example Programs: Text/ASCII Conversion

```
# Converting from text to ASCII codes
message = raw_input("Enter message to encode: ")

print "ASCII Codes:"
for ch in message:
    print ord(ch),

# Converting from ASCII codes to text
import string

inString = raw_input("Enter ASCII codes: ")

message = ""
for numStr in string.split(inString):
    message += chr(eval(numStr))

print "Decoded message:", message
```

String Formatting

- % operator inserts values into a template string (ala C printf)

```
<template-string> % (<values>)
```

- "Slots" specify width, precision, and type of value

```
%<width>.<precision><type-character>
```

- Examples

```
>>> "Hello %s %s, you owe %d" % ("Mr.", "X", 10000)
'Hello Mr. X, you owe 10000'

>>> "ans = %8.3f" % (3.14159265)
'ans =    3.142'
```

File Processing

- Opening a file

```
syntax:  <filevar> = open(<name>, <mode>)
example: infile = open("numbers.dat", "r")
```

- Reading from file

```
syntax:  <filevar>.read()
         <filevar>.readline()
         <filevar>.readlines()
example: data = infile.read()
```

- Writing to file

```
syntax:  <filevar>.write(<string>)
example: outfile.write(data)
```

Example Program: Username Creation

- Usernames are first initial and 7 chars of lastname (e.g. jzelle).

```
inf = open("names.dat", "r")
outf = open("logins.txt", "w")

for line in inf.readlines():
    first, last = line.split()
    uname = (first[0]+last[:7]).lower()
    outf.write(uname+'\n')

inf.close()
outf.close()
```

- Note use of string methods (Python 2.0 and newer)

Functions

- Example:

```
def distance(x1, y1, x2, y2):
    # Returns dist from pt (x1,y1) to pt (x2, y2)
    dx = x2 - x1
    dy = y2 - y1
    return math.sqrt(dx*dx + dy*dy)
```

- Notes:
  - Parameters are passed by value
  - Can return multiple values
  - Function with no return statement returns None
  - Allows Default values
  - Allows Keyword arguments
  - Allows variable number of arguments

Teaching Tip: Uniform Memory Model

- Python has a single data model
  - All values are objects (even primitive numbers)
  - Heap allocation with garbage collection
  - Assignment always stores a reference
  - None is a special object (not same as null)
- Pluses
  - All assignments are exactly the same
  - Parameter passing is just assignment
- Minuses
  - Need to be aware of aliasing when objects are mutable

Variable Scope

- Classic Python has only two scope levels: local and global (module level)
- Variable is created in a scope by assigning it a value
- Global declaration is necessary to indicate value assigned in function is actually a global

```
callCount = 0

def myFunc():
    global callCount
    callCount = callCount + 1
```

Decisions

```
if temp > 90:
    print "It's hot!"


if x <= 0:
    print "negative"
else:
    print "nonnegative"


if x > 8:
    print "Excellent"
elif x >= 6:
    print "Good"
elif x >= 4:
    print "Fair"
elif x >= 2:
    print "OK"
else:
    print "Poor"
```

Booleans in Python

- No Boolean type
- Conditions return 0 or 1 (for false or true, respectively)
- In Python 2.2.1 and later, True and False are defined as 1 and 0
- All Python built-in types can be used in Boolean exprs
  - numbers: 0 is false anything else is true
  - string: empty string is false, any other is true
  - None: false
- Boolean operators: and, or, not (short circuit, operational)

Loops

- For loop iterates over a sequence

```
for <variable> in <sequence>:
    <body>
```

  - sequences can be strings, lists, tuples, files, also user-defined classes
  - range function produces a numeric list
  - xrange function produces a lazy sequence

- Indefinite loops use while

```
while <condition>:
    <body>
```

- Both loops support break and continue

Loops with Else

- Python loops can have an else attached
  - Semantics: else fires if loop runs to completion (i.e. does not break)
  - Somewhat esoteric, but often quite useful

- Example:

```
for n in names:
    if n == target: break
else:
    print "Error:", target, "is not in list"
```

- I consider this an "advanced" feature

Lists: Dynamic Arrays

- Python lists are similar to vectors in Java
  - dynamically sized
  - heterogeneous
  - indexed (0..n-1) sequences
- Literals indicated with []
- Rich set of builtin operations and methods

Sequence Operations on Lists

```
>>> x = [1, "Spam", 4, "U"]
>>> len(x)
4

>>> x[3]
'U'

>>> x[1:3]
['Spam', 4]

>>> x + x
[1, 'Spam', 4, 'U', 1, 'Spam', 4, 'U']

>>> x * 2
[1, 'Spam', 4, 'U', 1, 'Spam', 4, 'U']

>>> for i in x: print i,
1 Spam 4 U
```

List are Mutable

```
>>> x = [1, 2, 3, 4]

>>> x[1] = 5
>>> x
[1, 5, 3, 4]

>>> x[1:3] = [6,7,8]
>>> x
[1, 6, 7, 8, 4]

>>> del x[2:4]
>>> x
[1, 6, 4]
```

List Methods

```
myList.append(x)    -- Add x to end of myList
myList.sort()       -- Sort myList in ascending order
myList.reverse()    -- Reverse myList
myList.index(s)     -- Returns position of first x
myList.insert(i,x) -- Insert x at position i
myList.count(x)     -- Returns count of x
myList.remove(x)    -- Deletes first occurrence of x
myList.pop(i)       -- Deletes and return ith element

x in myList         -- Membership check (sequences)
```

Example Program: Averaging a List

```
def getNums():
    nums = []
    while 1:
        xStr = raw_input("Enter a number: ")
        if not xStr:
            break
        nums.append(eval(xStr))
    return nums

def average(lst):
    sum = 0.0
    for num in lst:
        sum += num
    return sum / len(lst)

data = getNums()
print "Average =", average(data)
```

Tuples: Immutable Sequences

- Python provides an immutable sequence called tuple
- Similar to list but:
  - literals listed in () Aside: singleton (3,)
  - only sequence operations apply (+, *, len, in, iteration)
  - more efficient in some cases
- Tuples (and lists) are transparently "unpacked"

```
>>> p1 = (3,4)
>>> x1, y1 = p1
>>> x1
3
>>> y1
4
```

Dictionaries: General Mapping

- Dictionaries are a built-in type for key-value pairs (aka hashtable)
- Syntax similar to list indexing
- Rich set of builtin operations
- Very efficient implementation

Basic Dictionary Operations

```
>>> dict = { 'Python': 'Van Rossum', 'C++':'Stroustrup',
'Java':'Gosling'}

>>> dict['Python']
'Van Rossum'

>>> dict['Pascal'] = 'Wirth'

>>> dict.keys()
['Python', 'Pascal', 'Java', 'C++']

>>> dict.values()
['Van Rossum', 'Wirth', 'Gosling', 'Stroustrup']

>>> dict.items()
[('Python', 'Van Rossum'), ('Pascal', 'Wirth'), ('Java',
'Gosling'), ('C++', 'Stroustrup')]
```

More Dictionary Operations

```
del dict[k]           -- removes entry for k

dict.clear()          -- removes all entries

dict.update(dict2)    -- merges dict2 into dict

dict.has_key(k)       -- membership check for k

k in dict             -- Ditto

dict.get(k,d)         -- dict[k] returns d on failure

dict.setDefault(k,d) -- Ditto, also sets dict[k] to d
```

Example Program: Most Frequent Words

```
import string, sys

text = open(sys.argv[1],'r').read()
text = text.lower()
for ch in string.punctuation:
    text = text.replace(ch, ' ')

counts = {}
for w in text.split():
    counts[w] = counts.get(w,0) + 1

items = []
for w,c in counts.items():
    items.append((c,w))
items.sort()
items.reverse()

for i in range(10):
    c,w = items[i]
    print w, c
```

Python Modules

- A module can be:
  - any valid source (.py) file
  - a compiled C or C++ file
- Modules are dynamically loaded by importing
- On first import of a given module, Python:
  - Creates a new namespace for the module
  - Executes the code in the module file within the new namespace
  - Creates a name in the importer that refers to the module namespace
- from ... import ... is similar, except:
  - No name is created in the importer for the module namespace
  - Names for the specifically imported objects are created in the importer

Finding Modules

- Python looks for modules on a module search path
- Default path includes Python library location and current directory
- Path can be modified:
  - When Python is started (command line arg, env var)
  - Dynamically by the script itself (sys.path)
- Related modules can be grouped into directory structured packages

```
from OpenGL.GL import *
from OpenGL.GLUT import *
```

Useful Module Tricks

- Dual function module--import or stand-alone program

```
if __name__ == '__main__':
    runTests()
```

- Modules can be reloaded "on-the-fly"

```
reload(myModule)
```

- Module namespace is inspectable

```
>>> import string
>>> dir(string)

['_StringType', '__builtins__', '__doc__',
⋮
'uppercase', 'whitespace', 'zfill']
```

Teaching Tip: Information Hiding

- In Python, Information hiding is by convention
  - All objects declared in a module can be accessed by importers
  - Names beginning with _ are not copied over in a from...import *
- Pluses
  - Makes independent testing of modules easier
  - Eliminates visibility constraints (public, private, static, etc.)
- Minuses
  - Language does not enforce the discipline
- Bottom-line: Teaching the conventions is easier
  - The concept is introduced when students are ready for it
  - Simply saying "don’t do that" is sufficient (when grades are involved).

Python Classes: Quick Overview

- Objects in Python are class based (ala SmallTalk, C++, Java)
- Class definition similar to Java

```
class <name>:
    <method and class variable definitions>
```

- Class defines a namespace, but not a classic variable scope
  - Instance variables qualified by an object reference
  - Class variables qualified by a class or object reference
- Multiple Inheritance Allowed

Example: a generic multi-sided die

```
from random import randrange

class MSDie:

    instances = 0   # Example of a class variable

    def __init__(self, sides):
        self.sides = sides
        self.value = 1
        MSDie.instances += 1

    def roll(self):
      self.value = randrange(1, self.sides+1)

    def getValue(self):
        return self.value
```

Using a Class

```
>>> from msdie import *
>>> d1 = MSDie(6)
>>> d1.roll()
>>> d1.getValue()
6
>>> d1.roll()
>>> d1.getValue()
5
>>> d1.instances
1
>>> MSDie.instances
1
>>> d2 = MSDie(13)
>>> d2.roll()
>>> d2.value
7
>>> MSDie.instances
2
```

Example with Inheritance

```
class SettableDie(MSDie):

    def setValue(self, value):
        self.value = value

-----------------------------------------
>>> import sdie
>>> s = sdie.SettableDie(6)
>>> s.value
1
>>> s.setValue(4)
>>> s.value
4
>>> s.instances
3
```

Notes on Classes

- Data hiding is by convention
- Namespaces are inspectable

```
>>> dir(sdie.SettableDie)
['__doc__', '__init__', '__module__', 'getValue',
'instances', 'roll', 'setValue']
>>> dir(s)
['__doc__', '__init__', '__module__', 'getValue',
'instances', 'roll', 'setValue', 'sides', 'value']
```

- Attributes starting with __ are "mangled"
- Attributes starting and ending with __ are special hooks

Documentation Strings (Docstrings)

- Special attribute __doc__ in modules, classes and functions

- Python libraries are well documented

```
>>> from random import randrange
>>> print randrange.__doc__
Choose a random item from range(start, stop[, step]).

        This fixes the problem with randint() which includes the
        endpoint; in Python this is usually not what you want.
        Do not supply the 'int' and 'default' arguments.
```

- Used by interactive help utility

```
>>> help(randrange)
$ pydoc random.randrange
```

- Docstrings are easily embedded into new code
  - can provide testing framework

Another Class: Just for Fun

```
 #file: stack.py

"""Implementation of a classic
stack data structure: class Stack"""

class Stack:

    "Stack implements a classic stack with lists"

    def __init__(self): self.data = []

    def push(self, x): self.data.append(x)

    def top(self): return self.data[-1]

    def pop(self): return self.data.pop()
```

Exceptions

- Python Exception mechanism similar to Java and C++

```
try:
    foo(x,y)
    z = spam / x
except ZeroDivisionError:
    print "Can't Divide by Zero"
except FooError, data:
    print "Foo raised an error", data
except:
    print "Something went wrong"
else:
    print "It worked!"
```

- User code can raise an error

```
raise FooError, "First argument must be >= 0"
```

Python Library Overview

- Standard Library is Huge
- Example Standard Modules (besides math, string, pydoc)
  - sys: interpreter variables and interaction
  - pickle, cPickle: Object serialization
  - shelve: persistent objects
  - copy: deep copy support
  - re: regular expressions (ala Perl)
  - unittest: unit testing framework
  - cmath: complex math
  - random: various random distributions
  - os: access to OS services
  - os.path: platform independent file/directory names
  - time: time conversion, timing, sleeping
  - thread, threading: thread APIs
  - socket: low-level networking
  - select: asynchronous file and network I/O
  - Tkinter: interface to TK GUI library

Python Library Overview (cont'd)

- Standard Modules for Internet Clients/Servers
  - webbrowser: platform independent browser remote control
  - cgi: CGI client library
  - urllib, urllib2: generic utilities for fetching URLs
  - client libraries for: HTTP, FTP, POP2, IMAP, NNTP, SMTP, Telnet
  - urlparse: parsing URLs
  - server libraries for: Generic Network Server, HTTP, XMLRPC
  - Cookie: cookie parsing and manipulation
  - mimetools, MimeWriter, mimify, multifile: MIME processing tools
  - email, rfc822: email handling
  - base64, binascii, binhex, quopri, xdrlib: encoding and decoding
  - HTMLParser: parsing HTML
  - sgmllib: parsing SGML
  - xml: parser(xpat), DOM, SAX

Functional Programming Features

- Peter Norvig: ’...a dialect of LISP with "traditional" syntax.’

- FP features
  - First class functions
  - Recursion
  - Reliance on lists
  - Closures

- Python Functional Built-ins

```
lambda <args>: <expr> --> <fn-object>
map(<function>, <list>) --> <list>
filter(<function>, <list>) --> <list>
reduce(<function>, <list>) --> value
[<expr> for <vars> in <sequence> if <test>]
```

FP Example: List Processing

```
false, true = 0,1

head = lambda x: x[0]
tail = lambda x: x[1:]

def member(x, lst):
    if lst==[]:
        return false
    elif head(lst) == x:
        return true
    else:
        return member(x, tail(lst))

def reverse(lst):
    if lst==[]:
        return []
    else:
        return reverse(tail(lst)) + [head(lst)]
```

FP Example: QuickSort

- List Comprehension Combines Mapping and Filtering

```
[<expr> for <var> in <sequence> if <condition>]
[lt for lt in someList if lt < pivot]
```

- Using Comprehensions for Quicksort

```
def qsort(L):
    if len(L) <= 1: return L
    return ( qsort([lt for lt in L[1:] if lt < L[0]]) +
             L[0] +
             qsort([gt for gt in L[1:] if gt >= L[0]])
           )
```

FP Example: Closures

- Closure is a function that captures "surrounding" state
- Classic Python does not have nested scopes
- Can use default parameters to pass state into closure

```
>>> def addN(n):
...     return lambda x, a=n: x+a
...
>>> inc = addN(1)
>>> inc(3)
4
>>> addFive = addN(5)
>>> addFive(3)
8
```

Python GUI Options

- Lots of GUI Toolkits fitted for Python
  - Tkinter
  - wxPython
  - PythonWin
  - PyQt
  - pyFLTK
  - pyKDE
  - VTK
  - Java Swing (Jython)
  - Lots of others...
- Best Cross-Platform Options: TKinter, wxPython
- Defacto GUI: Tkinter

About Tkinter

- Allows Python to Use TK Toolkit from TCL/TK

- Pluses
  - Cross-platform
  - Very easy to learn and use
  - Comes with Python
  - Event loop integrated into interpreter
  - Excellent text and canvas widgets

- Minuses
  - Small widget set
  - Relies on TCL layer
    - can be sluggish
    - more layers to understand

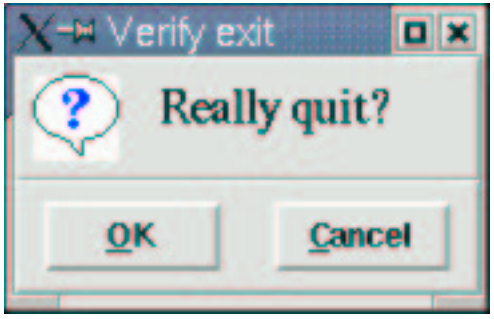
Example Program: Hello World, GUI-style

```
from Tkinter import *

root = Tk()
root.title("Hello GUI")
Label(root, text='Hello SIGCSE',
            font='times 32 bold').pack()
root.mainloop()
```

Example Program: Quitter

```
from Tkinter import *
from tkMessageBox import askokcancel
import sys

def quit():
    ans = askokcancel('Verify exit', "Really quit?")
    if ans:
        sys.exit()

root = Tk()
root.title("Quitter")
b = Button(root, text="Don't do it",
                 font="times 24 normal",
                 command = quit)
b.pack()
root.mainloop()
```

Example Program: Quitter (Screenshots)



Example Program: OO Quitter

```
class Quitter:

    def __init__(self, master):
        self.qb = Button(master,
                         text = "Don't do it!",
                         command = self.quit)
        self.qb.pack()

    def quit(self):
        ans = askokcancel("Verify exit",
                          "Really quit?")
        if ans: sys.exit()

root = Tk()
root.title("Quitter 2")
Quitter(root).mainloop()
```

Example Program: Simple Editor

```
class Editor(Frame):

    def __init__(self, root):
        self.root = root
        Frame.__init__(self, root)
        self.pack()
        self.text = ScrolledText(self,
                                 font="times 24 normal")
        self.text.pack()
        self.filename = None
        self.buildMenus()

    def buildMenus(self):..

    def onSave(self):...

    def onExit(self):...
```

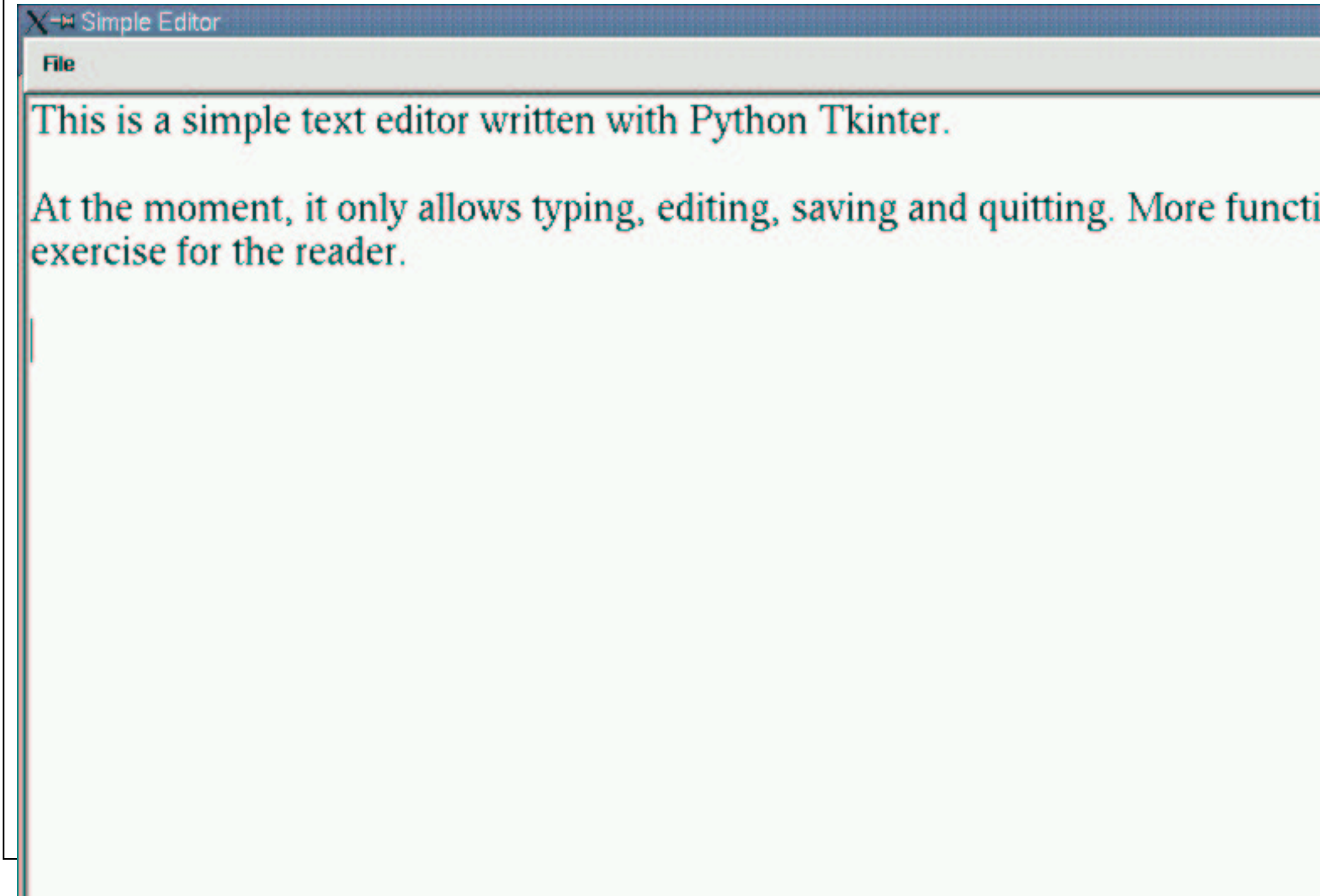
Example Program: Simple Editor(cont'd)

```
def buildMenus(self):
    menubar = Menu(self.root)
    self.root.config(menu=menubar)
    filemenu = Menu(menubar,tearoff=0)
    filemenu.add_command(label="New...", command=None)
    filemenu.add_command(label="Open...", command=None)
    filemenu.add_separator()
    filemenu.add_command(label = "Save",
                         command=self.onSave)
    filemenu.add_command(label="Save as...",
                         command=None)
    filemenu.add_separator()
    filemenu.add_command(label="Exit",
                         command=self.onExit)
    menubar.add_cascade(label="File", menu=filemenu)
```

Example Program: Simple Editor (cont'd)

```
    def onSave(self):
        if not self.filename:
            filename = asksaveasfilename()
        if not filename:
            return
        self.filename=filename
        file = open(filename, 'w')
        file.write(self.text.get("1.0",END))
        file.close()


    def onExit(self):
        ans = askokcancel('Verify exit',
                          'Really quit?')
        if ans: sys.exit()


root = Tk()
root.title("Simple Editor")
Editor(root).mainloop()
```

Example Program: Simple Editor (screen)



Real Tkinter Application: PySol

Computer Graphics

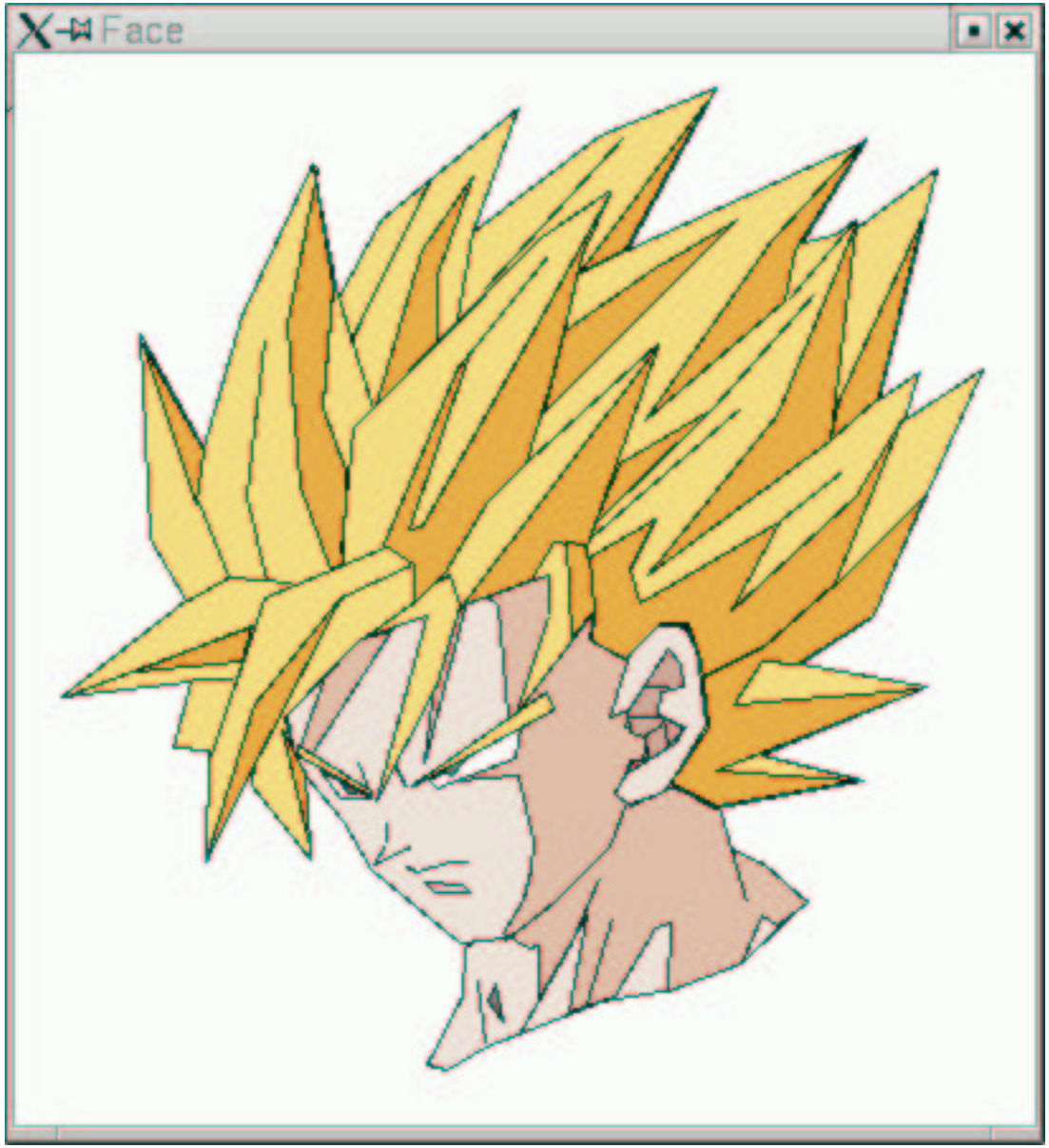
- "Baby" Graphics Package in CS1
  - "Hides" the event loop
  - Provides OO 2D primitives for drawing
  - Input via mouse click and entry box
  - Students implement own GUI widgets
- Upper-Level Courses Use Python Add-Ins
  - VPython simple 3D visualization package
  - PyOpenGL -- wrapper over OpenGL API

Baby Graphics: triangle.py

```
from graphics import *  # our custom graphics

win = GraphWin("Draw a Triangle")
win.setCoords(0.0, 0.0, 10.0, 10.0)
message = Text(Point(5, 0.5), "Click on three points")
message.draw(win)
p1 = win.getMouse()
p1.draw(win)
p2 = win.getMouse()
p2.draw(win)
p3 = win.getMouse()
p3.draw(win)
triangle = Polygon(p1,p2,p3)
triangle.setFill("peachpuff")
triangle.setOutline("cyan")
triangle.draw(win)
message.setText("Click anywhere to quit.")
win.getMouse()
```

Baby Graphics: Triangle Screenshot

Baby Graphics: Example Face

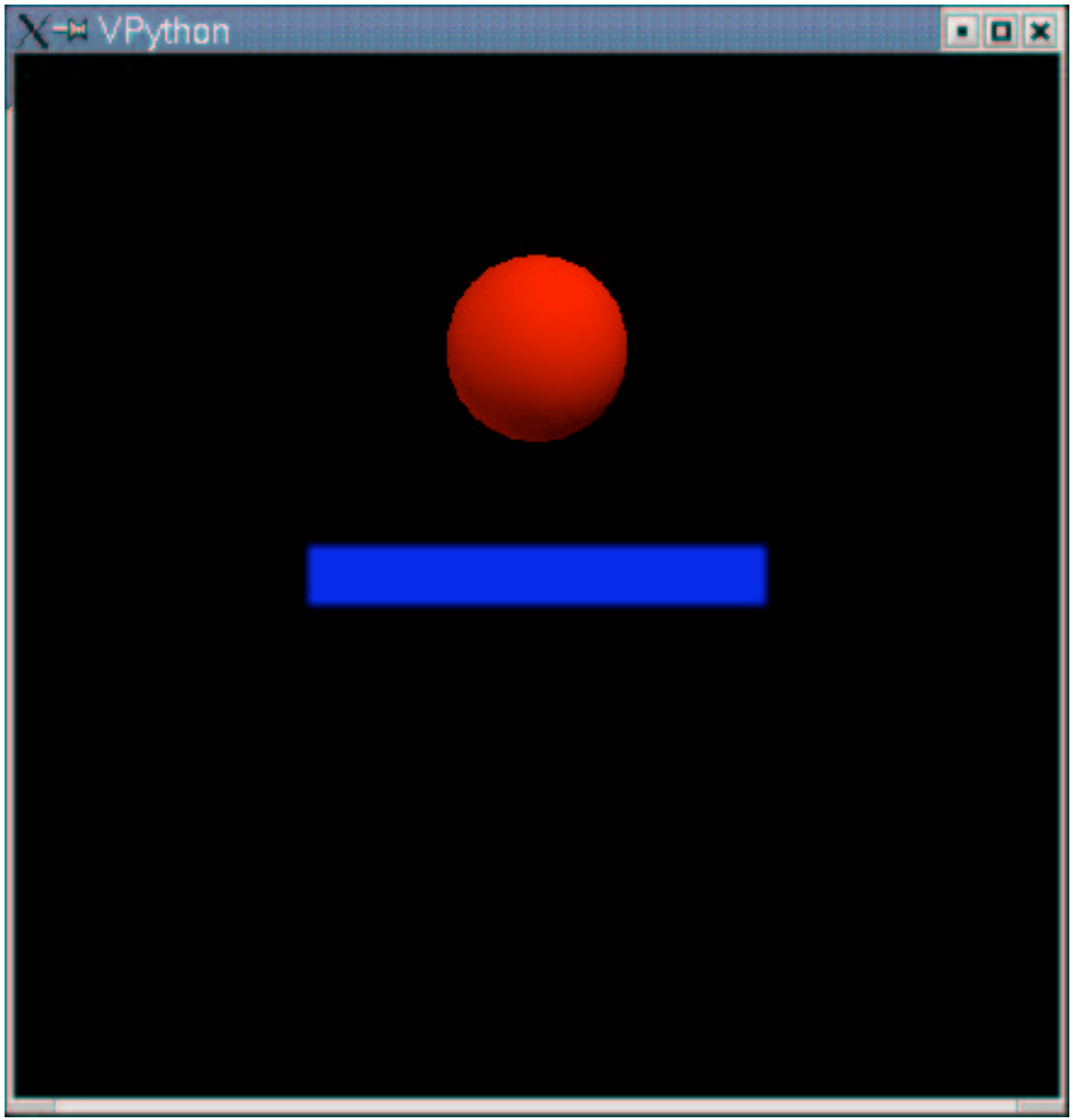


Baby Graphics: Blackjack Project

VPython Example: Bounce

```
from visual import *

floor = box(length=4, height=0.5,
            width=4, color=color.blue)

ball = sphere(pos=(0,4,0), color=color.red)
ball.velocity = vector(0,-1,0)

scene.autoscale=0
dt = 0.01
while 1:
    rate(100)
    ball.pos = ball.pos + ball.velocity*dt
    if ball.y < 1:
        ball.velocity.y = -ball.velocity.y
    else:
        ball.velocity.y = ball.velocity.y - 9.8*dt
```

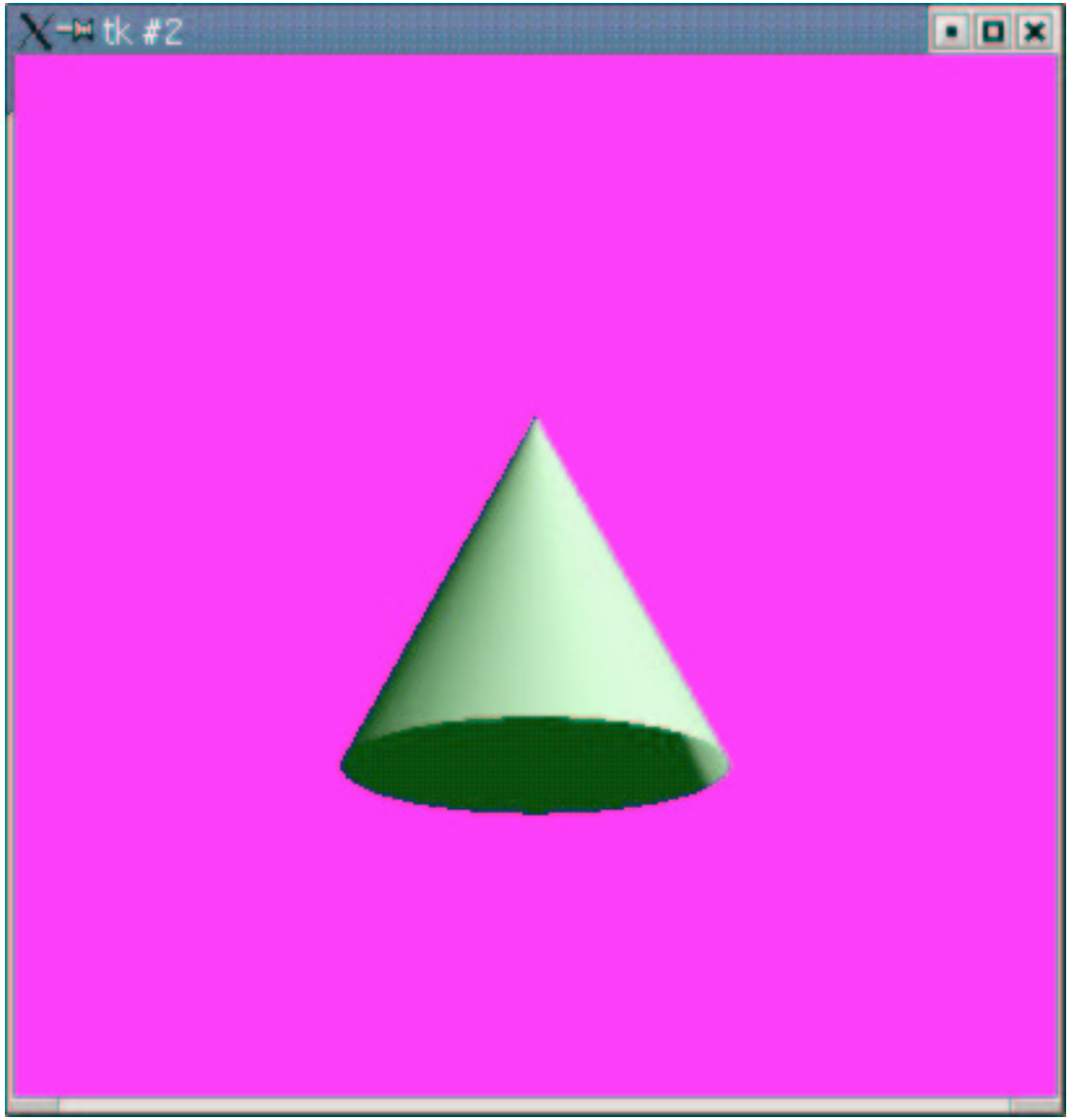
VPython Example: Screenshot



PyOpenGL Example: GLUT Cone

```
def init():
  glMaterialfv(GL_FRONT, GL_AMBIENT, [0.2, 0.2, 0.2, 1.0])
  glMaterialfv(GL_FRONT, GL_SHININESS, 50.0)
  glLightfv(GL_LIGHT0, GL_AMBIENT, [0.0, 1.0, 0.0, 1.0])
  glLightfv(GL_LIGHT0, GL_POSITION, [1.0, 1.0, 1.0, 0.0]);
  glLightModelfv(GL_LIGHT_MODEL_AMBIENT, [0.2, 0.2, 0.2, 1.0])
  glEnable(GL_LIGHTING); glEnable(GL_LIGHT0)
  glDepthFunc(GL_LESS)
  glEnable(GL_DEPTH_TEST)

def redraw(o):
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
  glPushMatrix()
  glTranslatef(0, -1, 0)
  glRotatef(250, 1, 0, 0)
  glutSolidCone(1, 2, 50, 10)
  glPopMatrix()

def main():
  o = Opengl(width = 400, height = 400, double = 1, depth = 1)
  o.redraw = redraw
  o.pack(side = TOP, expand = YES, fill = BOTH)
  init()
  o.mainloop()

main()
```

PyOpenGL Example: GLUT Cone (Screen)



Internet Programming

- Use socket library to teach protocol fundamentals
- Server Side Technologies
  - Build HTTP server using library
  - CGI programs
  - Custom Application Framework (with XML?)
  - Database manipulation
- Client Side Technologies
  - Build standard client (e.g. email, web browser,etc)
  - Novel html application (e.g. spider, site grabber, etc.)
  - Novel web application (with XML?)
- Client-Server GUI

Example Project: Chat Server

- Three modules
  - chat server
  - talk client
  - listen client
- Problem is to devise a protocol to allow
  - Any number of (anonymous) listeners
  - Any number of talkers identified by nickname
  - Clean method of shutting down listeners

Chat Server Shell

```
import socket, sys

def server(port=2001):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.bind(("", port))
    s.listen(10)
    listeners = []  # list of listener sockets
    print "SCServer started on port", port
    while 1:
        conn, address = s.accept()
        message = ""
        while "\n" not in message:
            message = message + conn.recv(1)
        if message[0] in "lL":
            listeners.append(conn)
        elif message[0] in "tT":
            for lsock in listeners:
                lsock.send(message[1:])

if __name__ == "__main__": server(eval(sys.argv[1]))
```

Chat Clients

```
def talk(machine, port):
    while 1:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        data = raw_input(">>> ")
        s.connect((machine, port))
        s.send("t"+data+"\n")
        s.close()

if __name__ == "__main__": talk(sys.argv[1], eval(sys.argv[2]))

-------------------------------------------------------------
def listen(machine, port):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((machine, port))
    s.send("listen\n")
    try:
        while 1:
            mess = ""
            while not "\n" in mess:
                mess = mess + s.recv(1)
            print mess[:-1]
    finally:
        s.close()

if __name__ == "__main__": listen(sys.argv[1], eval(sys.argv[2]))
```

Example Project: Web Site from Scratch

- Goal: Create a complete functioning website using only
  - A text editor
  - An image editor
  - A Python interpreter
- Must Create Both Content and Server
- Python Provides TCP Server Framework
- BTW: Python Also Has HTTP Server!

TCP Server Example: httpPeek

```
RESPONSE = """HTTP/1.0 200 OK
Connection: Close
Content_type: text/html


<HTML>
<PRE>
 %s
</PRE>
</HTML>"""

class EchoHandler(StreamRequestHandler):

    def handle(self):
        lines = []
        while 1:
            line = self.rfile.readline()
            if line == "\r\n": break       #empty line at end of header
            lines.append(line)
        self.wfile.write(RESPONSE % "".join(lines))

def start(port):
    server = TCPServer(("",port), EchoHandler)
    server.serve_forever()

if __name__ == "__main__": start(int(sys.argv[1]))
```

Example Assignment: CGI Scripting

- CGI -- Common Gateway Interface
- HTTP Server Options
  - Configure global server (e.g. Apache)
  - Use Python CGIHTTPServer
- Scripts are Just Programs
  - Input from env variables and stdin
  - Output to stdout
- Python Provides
  - Standard module (cgi) to parse form input
  - Add-ons to produce nice HTML

Simple CGI Script

```
import cgi

RESPONSE="""Content-type: text/html


<HTML>
<HEAD>
<TITLE>%s</TITLE>
</HEAD>
<PRE>
<BODY>
 %s
</PRE>
</BODY>"""

form = cgi.FieldStorage()
content = []
for name in form.keys():
    content.append("Name: %s  value: %s" % (name, form[name].value))
content.append("Done")

print RESPONSE %("Form Echo", "\n".join(content))
```

Databases

- Modules Available for Every Major DB
  - ODBC drivers
  - MySQL
  - PostgreSQL
  - Commercial DBs (Oracle and friends)
- Pure Python DB: Gadfly
- Uses
  - Backend for web applications
  - Interactive query engine
  - DB Application Development

Datbase Example: PostgreSQL

```
import pg    # PostgreSQL database module
from pprint import pprint  # pretty printing

QUERY="""SELECT customer_name, balance
   FROM account, depositor
   WHERE account.balance > 500
        and account.account_number=depositer.account_number"""

db = pg.connect(dbname='bank', host='localhost', user='zelle')
res = db.query(QUERY)
print res.ntuples()
pprint(res.getresult())
pprint(res.dictresult())
pprint(res.listfields())

-------------------------------------------------------------

4
[('Johnson', 900.0), ('Jones', 750.0), ('Lindsay', 700.0)]
[{'customer_name': 'Johnson', 'balance': 900.0},
 {'customer_name': 'Jones', 'balance': 750.0},
 {'customer_name': 'Lindsay', 'balance': 700.0}]
('customer_name', 'balance')
```

Operating Systems

- OS Course is Perfect for Systems Language...
  - IF you're implementing an OS
- Python Excels for
  - Experimenting with system calls
  - Concurrent programming (processes and threads)
  - Simulations (queuing, paging, etc.)
  - Algorithm animations
- Appropriateness Depends on Type of Course

POSIX Process Calls

```
# fork -- create a (duplicate) process
if os.fork() == 0:
   print "in child"
else:
   print "in parent"

# exec -- overlay the process with another executable
os.execl("/bin/more", "more", "foo.txt")  # note: no 0 terminator
os.execvp(sys.argv[0], sys.argv)

# sleep -- put process to sleep for specified time
time.sleep(n)

# exit -- terminate process
sys.exit(0)

# wait -- wait for termination of child
pid, status = wait()  # no arguments, returns a pair of values
print "Returned status:", status/256

# getpid -- return process id
myId = os.getpid()
```

POSIX Signals

```
# signal -- installs a signal handler
signal.signal(number, handlerFn)

# pause -- put process to sleep until signal is received
signal.pause()

-------------------------------------------------------------
import signal

def handler(n, traceback):
    print "Caught signal:", n

for i in range(1,31):
    if i != 9 and i != 19:
        signal.signal(i, handler)

print "I'm a tough process, you can't kill me!"
for i in range(1,6):
    signal.pause()
    print "Hit number", i
print "You sunk my battleship"
```

Example Assignment: Process Sieve

- Implement Sieve of Eratosthenes using pipeline of processes
- Each process filters out numbers divisible by its prime
- Process pipeline grows as each prime is found

Process Sieve Code

```
def main(n):
    pipe = spawnNode()
    for i in range(2,n):
        os.write(pipe, str(i)+'\n')
    os.write(pipe, "-1\n")
    os.wait()

def spawnNode():
    readEnd, writeEnd = os.pipe()
    if os.fork() == 0:  # Code for newly created node
        os.close(writeEnd); sieveNode(readEnd); sys.exit(0)
    return writeEnd

def sieveNode(pipeIn):
    myIn = os.fdopen(pipeIn)  # Turn pipe into regular file
    myNum = eval(myIn.readline())
    print "[%d]: %d" % (os.getpid(),myNum)
    myOut = None
    while 1:
        candidate = eval(myIn.readline())
        if candidate == -1: break
        if candidate % myNum != 0:     # not divisible, send down pipe
            if not myOut: myOut = spawnNode()
            os.write(myOut, str(candidate)+'\n')
    if myOut:
        os.write(myOut,'-1\n')
        os.wait()
```

Threads

- Python Provides Two Libraries
  - thread -- basic thread functionality
  - threading -- Object-based threading model
- Basic Thread Facilities
  - Create new thread to run a function
  - Create simple locks (binary semaphore)
- Assignments:
  - Create more sophisticated structures
  - Concurrency/Synchronization problems

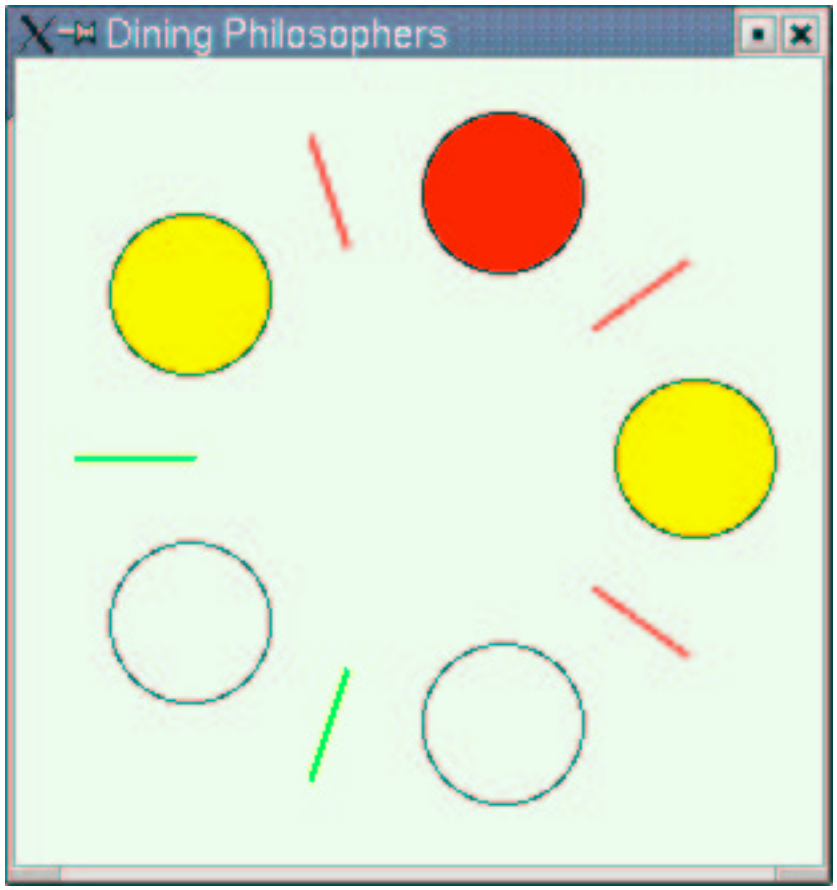
Example Assignment: Counting Semaphores

```
class Semaphore:
    def __init__(self, value = 0):
        self.count = value
        self.queue = []
        self.mutex = thread.allocate_lock()

    def wait(self):
        self.mutex.acquire()
        self.count = self.count - 1
        if self.count < 0:
            wlock = thread.allocate_lock()
            wlock.acquire()
            self.queue.append(wlock)
            self.mutex.release()
            wlock.acquire()  # suspend on new lock
        else:
            self.mutex.release()

    def signal(self):
        self.mutex.acquire()
        self.count = self.count + 1
        if self.count <= 0:
            wlock = self.queue[0]
            del self.queue[0]
            wlock.release() # let the waiting thread go
        self.mutex.release()
```

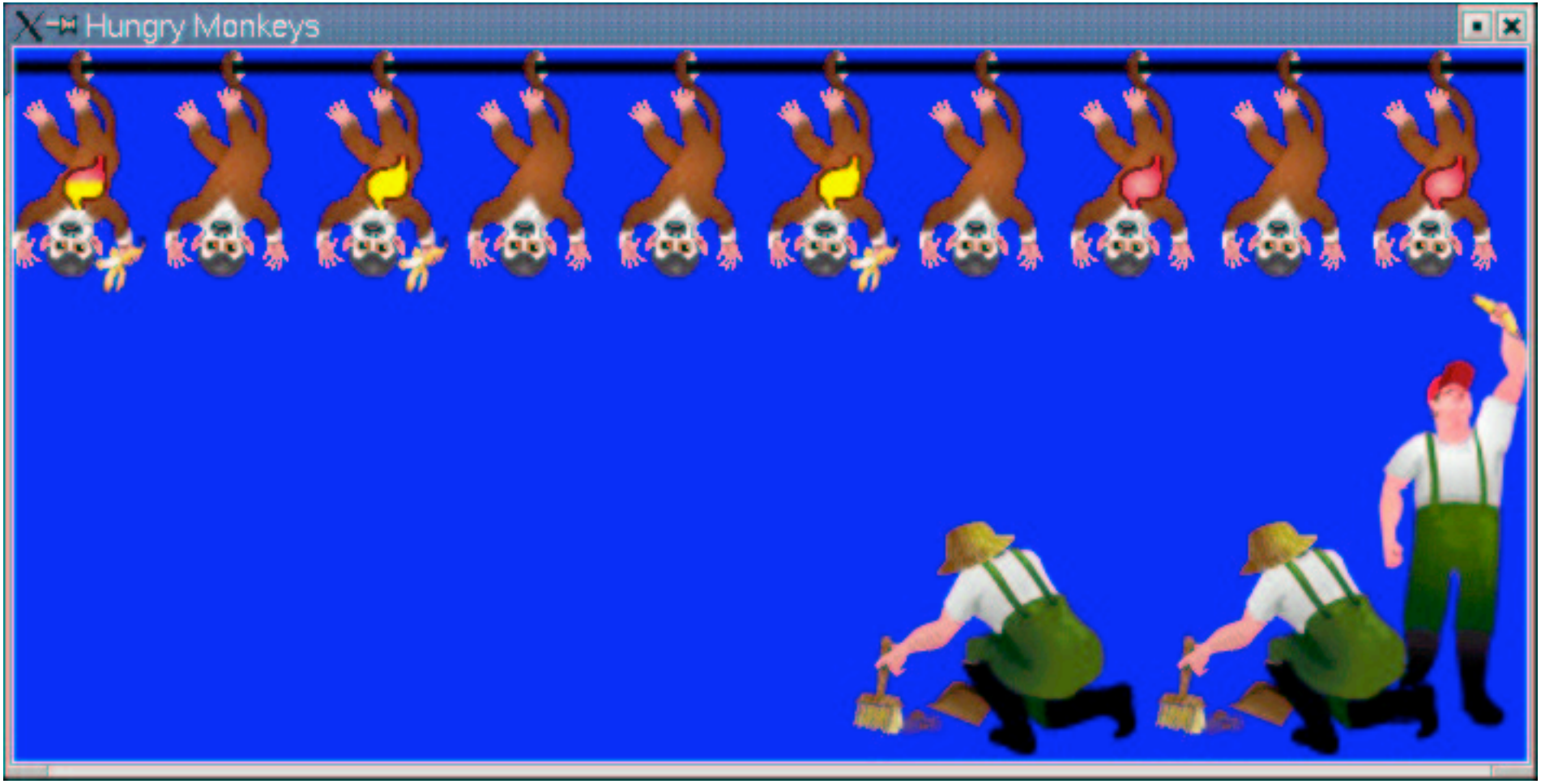
Example Program: Dining Philosophers

```
NUM_PHILOSOPHERS = 5; THINKMAX = 6; EATMAX = 2

def philosopher(n, forks, display):
    f1, f2 = n, (n+1)% NUM_PHILOSOPHERS
    display.setPhil(n, "thinking")
    while 1:  #infinite loop
        time.sleep(randint(0,THINKMAX))
        display.setPhil(n, "hungry")
        forks[f1].wait()
        display.setFork(f1,"inuse")
        time.sleep(1)
        forks[f2].wait()
        display.setFork(f2, "inuse"); display.setPhil(n, "eating")
        time.sleep(randint(1,EATMAX))
        display.setPhil(n, "thinking"); display.setFork(f2,"free")
        forks[f2].signal()
        display.setFork(f1,"free")
        forks[f1].signal()

d = DPDisplay(NUM_PHILOSOPHERS)
forks = []
for i in range(NUM_PHILOSOPHERS):
    forks.append(Semaphore(1)); d.setFork(i,"free")
for i in range(NUM_PHILOSOPHERS):
    thread.start_new_thread(philosopher, (i,forks, d))
d.pause()
```

Dining Philosophers (Screenshot)



Student Project: Hungry Monkeys



Python in Other Classes

- Virtually any class with a programming component can benefit from Python's power and simplicity
- Even in less obvious cases
  - Numerical Analysis: NumPy
  - Parallel Processing: pyPVM
  - 3D Games: Crystal Space
- Options for specialized libraries
  - Check to see if someone has "ported" them (or do it yourself)
  - C, C++ libraries: Wrap with SWIG
  - Java libraries: Use as-is with Jython
  - Use Inter-Process Communication

Python Resources

- Textbooks (CS1, CS2)
  - "Python: How to Program," Deitel, Deitel, Liperi, Weidermann, andLiperi (Prentice Hall)
  - "How to Think Like a Computer Scientist: Learning with Python," Downey, Elkner, and Meyers (Green Tea Press)
  - "Python Programming: An Introduction to Computer Science," Zelle http://mcsp.wartburg.edu/zelle/PythonCS1_Draft.pdf
- Technical Python Books
  - Too many to list, see Python web site and Amazon
- Python Web Sites
  - www.python.org -- The site for everything Pythonic
  - www.vex.net/parnassus/ -- Searchable database of Python add-ons

Python Rocks!

You'll Never Go Back