Hi, I'm Randall. Welcome to my TED talk.

It's an honor to speak to you, some of the brightest innovators from so many fields, about a problem in desperate need of your attention:

How do you end parenthetical statements with emoticons?
I can't figure out a good way.

...Linux (or BSD :)) would...
...Linux (or BSD :) ) would...

Conferences I'm banned from:
- SIGGRAPH
- EUROCRYPT
- DEF CON
- PICON
- International Astronomical Union
- Canadian Paleontology Conference
- Every American Furry Convention
- American Baking Society
- Asia Dolphin Training Convention
- TED

source: http://xkcd.com/541/
HW2, HW3 stats, Grading

- Class did well on average
- 9 on HW2, 9+ on HW3
- I was generous – (research has been busy – the extra points are courtesy of the delay)
- If you're in that range you're doing great, 9 curves to about an A-
- If you have not turned in HW3 (and have not spoken with me already) – do so immediately
- Project proposals: see me tonight re: CCLS
Agenda – using libraries in the real world

- Extra credit and project exercises
- Working with compressed files – zip creation, appending, extracting
- Recursively exploring a directory structure
- Serializing data with cPickle (a couple elegant examples)
- An introduction to the Python DBAPI – connecting to MySQL and PostgreSQL databases
- PyWin32 – leveraging (hacking) the windows common object model to access Word and Excel functionality
- An introduction to Tkinter
In class extra credit exercise: Treasure Hunt

• Use the necessary libraries to write a routine which searches for a pattern within plain text files recursively reachable from the starting directory

• Look within compressed archives

• The input to your script should be the starting directory followed by the search string

• python treasureHunt.py /home/josh treasure
In class exercise: adding functionality to your project

- At your discretion, begin adding either
  - A simple GUI framework to your project
  - Serialization capability
  - Database connectivity
- Suggestions to get you started
  - GUIs: Start / Stop buttons, an about box, a file chooser rather than command line arguments
  - Serialization, the ability to save and resume program state, send data across networks
  - Databases: reading / storing data, logging program results
Working with Compressed Files

- Python can work directly with data in zip, gzip, bz2, tars, etc
- Most libraries offer decompression on the fly (i.e., unnecessary to extract the entire archive to modify files)
- Third party libraries are available to handle pretty much any format
import zipfile

def read():
    # use r, not rb
    z = zipfile.ZipFile('test.zip', 'r')
    for filename in z.namelist():
        # slurp the uncompressed bytes into a string
        contents = z.read(filename)
        print('File:', filename)
        print(contents)

if __name__ == '__main__':
    read()
import zipfile

# create using Deflate as the default compression is deflate
z = zipfile.ZipFile("test.zip", "w")

path_1 = 'helloworld.txt'
path_2 = 'hey_dad.txt'

z.write(path_1)
z.write(path_2)

print z.namelist()

z.close()

# append

z = zipfile.ZipFile("test.zip", "a")

path_3 = 'hi_mom.txt'

z.write(path_3)

print z.namelist()

z.close()
Recursively exploring a directory structure using os.walk()

- os.walk generates a tuple for each directory rooted at the starting path
  - (dirpath, dirnames, filenames)
  - dirpath: a string holding the path of the current directory
  - dirnames: a list containing the names of subdirectories
  - filenames: a list of the names of the non-directory files in dirpath
- See: http://docs.python.org/library/os.html
Recursively explore a directory

```python
import os
from os.path import join, getsize

for current, dirs, files in os.walk('/Users/josh/):
    print 'current dir', current
    print 'contains subdirectories', dirs
    print 'contains files', files
    print 'consumes', 
    #reconstruct the path
    print sum(getsize(join(current, name)) for name in files),
    print 'bytes in', len(files), 'non-directory files'

#current dir /Users/josh/
#contains subdirectories [...] 
#contains files [...] 
#consumes 44127 bytes in 9 non-directory files 
#current dir /Users/josh/courses 
# ...
```
import re
exp = re.compile('donut')
f = '/Users/josh/Desktop/test.txt'
lines = open(f).readlines()
text = ".join(lines)
match = exp.search(text)
if match: print match.group()
Serializing Data using cPickle

- Serialization: turning data into a string of bytes which can be saved to disk, inserted into a database, sent across a network, etc.

- Useful for:
  - interprocess python communication
  - saving and restoring program state
  - storing data (caveat – never do this if you need to play well with other development environments!)

- Options:
  - cPickle = python extension coded in C, fast – usually available, use this
  - pickle = pure python, same thing, slower, 100% compatible, always available
  - No size limitations on serializable objects
Serializing Data using cPickle
(w.r.t naming: software engineers are not poets)

- Basic operations: dump and load
- Dump: store arbitrary data structure
  - Supports text and binary forms (where might you need text data?)
- Load
  - Compatibility is guaranteed from one Python release to the next
  - Machine and implementation independent
- In between dumps and loads you can
  - Store data to a file, compress, send over a network, etc
Simple example

```python
import cPickle
donuts = {'homer': 12, 'lisa': 0, 'marge': 1, 'bart': 6}

# serialize to text
text = cPickle.dumps(donuts)

# binary string, faster, more compact
bytes = cPickle.dumps(donuts, 2)

print bytes

# bring it back
restored_donuts = cPickle.loads(text)
restored_donuts_2 = cPickle.loads(bytes)

print restored_donuts_2
```
Store, compress, and load objects using a generator

```python
import cPickle, gzip

def store(f_name, *objs):
    out = gzip.open(f_name, 'wb')
    for object in objs:
        cPickle.dump(object, out, 2)
    out.close()

def retriever(f_name):
    in_file = gzip.open(f_name, 'rb')
    while True:
        try:
            yield cPickle.load(in_file)
        except EOFError:
            ...
    in_file.close()
```
Pickling class instances and mixed types

```
foo = 'hello pickle world'
Bar = [7, 5, 1, 9]
x = range(5)

store('test.dat', foo, Bar, x)
x = retriever('test.dat')

while True:
    try:
        print x.next()
    except StopIteration:
        Break

hello pickle world
[7, 5, 1, 9]
[0, 1, 2, 3, 4]

class Donut():
    def __init__(self):
        self.tasty = True
        self.homer = 'hungry'
        self.cost = {'jelly': 5, 'orange': 1}

d = Donut()
store('test.dat', d)
x = retriever('test.dat')
y = x.next()
print y
print y.tasty

<__main__.Donut instance at 0x78cd8>
True
```
The Python Database API (DBAPI)

- Python specifies a common interface for database access, but the standard library does not include a RDBMS (relation db management sys) module - why?
- Designed to encourage similarity between database implementations – pick a module, same patterns apply
  - Defines common connection objects, cursor objects, types, etc
Implementations

- There are many free third-party modules (including XML db support).
- Pretty much these all work the same way programatically.
- Differences are mostly in SQL variations.
- PostgreSQL: http://www.initd.org/
- PostgreSQL: http://pybrary.net/pg8000/
- MySQL http://sourceforge.net/projects/mysql-python
- MSSQL: http://pymssql.sourceforge.net/
DBAPI Pattern

- Download and install the DBAPI implementation
- Import the module and call the connect function (when you're finished, remember to close it)
- Specify the server address, port, database, and authentication
- Get a cursor, use it to execute SQL (cursors are emulated for DBs which do not support them)
- Fetch results as a sequence of tuples, one tuple per row in the result, where tuple indexes correspond to columns
- Cursors pretty much work the way you expect in other languages, just with less code.
- Standard methods on cursors: fetchone(), fetchmany(), fetchall()
import MySQLdb

# create a connection object
con = MySQLdb.connect('127.0.0.1',
                      port=3306, user='tomato',
                      passwd='squish', db='test')

# get a cursor
cursor = con.cursor()

# some quick sql
sql = 'SELECT * FROM Simpsons

# execute and fetch
cursor.execute(sql)

results = cursor.fetchall()

print results

# close the connection
con.close()
A more elegant way to access columns

def fields(cursor):
    """ Given a DB API 2.0 cursor object that has been executed, \n    returns a dictionary that maps \n    each field name to a column index, 0 and up"""

    results = {}
    for column, description in enumerate(cursor.description):
        results[description[0]] = column

    return results

Source: The Python CookBook – O'Reilly
Insert some data into a PostgreSQL database

```python
from pg8000 import DBAPI
conn = DBAPI.connect(host='localhost', user="tomato",
password="squish")
cursor = conn.cursor()
cursor.execute(
    'CREATE TEMPORARY TABLE book (id SERIAL, title TEXT)'
)
cursor.execute("INSERT INTO book (title) VALUES (%s), (%s) 
    RETURNING id, title",("Ender's Game", "Speaker for the Dead"))
for row in cursor:
    id, title = row
    print "id = %s, title = %s" % (id, title)
#id = 1, title = Ender's Game
#id = 2, title = Speaker for the Dead
conn.commit()
cursor.close()
conn.close()
```
Living in a Windows World

- Proprietary formats abound (.xlsx, .docx, etc)
- Often times you need to access data stored in them, or create them on the fly
- Couple of ways to do that
  - Leverage the Win32 API and the common object model (trivial, but annoying)
  - Leverage libraries with specific format knowledge (case by case)
- Note on corporate environments
  - make yourself look good.
PyWin32 winapi and COM

- COM: older protocol enabling interprocess communication and dynamic object creation
  - Umbrella term: encompasses OLE (object linking and embedding)
- Language neutral, the idea is to allow use of objects with minimal knowledge of their internal structure
- Well supported in windows
- References:
  - http://www.microsoft.com/com/default.mspx
  - http://sourceforge.net/projects/pywin32/
Leverage Win32 API to set file attributes

```python
import win32com, win32api, os

filename = 'foo.txt'
f = open(filename, 'w')
f.close()
win32api.SetFileAttributes(filename, win32com.FILE_ATTRIBUTE_HIDDEN)

#FILE_ATTRIBUTE_READONLY
#FILE_ATTRIBUTE_NORMAL
```

Converting word documents to plaintext via COM (icky, but great when you need it)

```python
import os, sys, win32com.client
word = win32com.client.Dispatch("Word.Application")
try:
    for current_dir, sub_dirs, files in os.walk(sys.argv[1]):
        for filename in files:
            if not (filename.endswith('.doc')): continue
            doc = os.path.abspath(os.path.join(path, filename))
            print 'converting %s' % doc
            word.Documents.Open(doc)
            new_name = doc[:-3] + 'txt'
            word.ActiveDocument.SaveAs(new_name, \
                FileFormat=win32com.client.constants.wdFormatText)
            word.ActiveDocument.Close()
finally:
    word.Quit()
```
Programatically control Excel via COM

# reference an Excel instance
import win32com.client
app = \ win32com.client.Dispatch("Excel.Application")
# set visible if you like
app.Visible=1
# create a workbook
book = app.Workbooks.Add()
# get a reference to the first sheet
sheet = book.Sheets(1)
# extract the value from a cell
val = xlSheet.Cells(1,1).Value
# insert a formula
sheet.Cells(1,2).Value = "5"
sheet.Cells(1,3).Value = "=a2*2"
...

- Corporate environments LOVE this
- Nice if the output of your script appears in a prettily formatted excel sheet, yes?
- Icky – but trivial to code with the right documentation
Python Programming On Win32

- Detailed resources available
- You can integrate code pretty deeply
- Customers will love you
- FYI: you can work w/ Open Office standards (xml based) without any special libraries
Tkinter GUIs

- GUIs are programmed via toolkits, which supply prebuilt controls (aka widgets)
- Widgets range from simple buttons to complex graphs
- There are many toolkits available for Python
  - Many are cross platform – _ignore anything that's not_
- Most popular toolkit is probably wxpython
- Tkinter is included by default
Tkinter

- Object oriented Python wrapper for TK
- Tk – basic cross platform toolkit originally written for Tcl, wrappers available for Ruby and Perl as well.
- Works on Windows / Mac / Linux
- FYI: when debugging gui applications, run them as a standalone from the command line – interference issues running one tk gui w/in another
- We're interested in simple, functional guis
Tkinter Fundamentals

• Basic pattern
• Import Tkinter
• Create, configure, and position the widget
• Enter the Tk mainloop at which point your code becomes event driven
• Terminology
  • layout manager – responsible for geometric on screen layout
  • Pack() - hands control of the widget to the layout man.
import sys, Tkinter
Tkinter.Label(text='hello gui world').pack()
Tkinter.Button(text='bye', command=sys.exit).pack()
Tkinter.mainloop()

- The calls to label and button instantiate the widgets
- We do not specify parent windows, so these are placed on the applications top level
- Notice the command field, this may be any callable object
<table>
<thead>
<tr>
<th>Widget</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Button</td>
<td>A simple button, used to execute a command or other operation.</td>
</tr>
<tr>
<td>Canvas</td>
<td>Structured graphics. This widget can be used to draw graphs and plots, create graphics editors, and to implement custom widgets.</td>
</tr>
<tr>
<td>Checkbutton</td>
<td>Represents a variable that can have two distinct values. Clicking the button toggles between the values.</td>
</tr>
<tr>
<td>Entry</td>
<td>A text entry field.</td>
</tr>
<tr>
<td>Frame</td>
<td>A container widget. The frame can have a border and a background, and is used to group other widgets when creating an application or dialog layout.</td>
</tr>
<tr>
<td>Label</td>
<td>Displays a text or an image.</td>
</tr>
<tr>
<td>Listbox</td>
<td>Displays a list of alternatives. The listbox can be configured to get radiobutton or checklist behavior.</td>
</tr>
<tr>
<td>Menu</td>
<td>A menu pane. Used to implement pulldown and popup menus.</td>
</tr>
<tr>
<td>Menubutton</td>
<td>A menubutton. Used to implement pulldown menus.</td>
</tr>
<tr>
<td>Message</td>
<td>Display a text. Similar to the label widget, but can automatically wrap text to a given width or aspect ratio.</td>
</tr>
<tr>
<td>Radiobutton</td>
<td>Represents one value of a variable that can have one of many values. Clicking the button sets the variable to that value, and clears all other radiobuttons associated with the same variable.</td>
</tr>
<tr>
<td>Scale</td>
<td>Allows you to set a numerical value by dragging a &quot;slider&quot;.</td>
</tr>
<tr>
<td>Scrollbar</td>
<td>Standard scrollbars for use with canvas, entry, listbox, and text widgets.</td>
</tr>
<tr>
<td>Text</td>
<td>Formatted text display. Allows you to display and edit text with various styles and attributes. Also supports embedded images and windows.</td>
</tr>
<tr>
<td>Toplevel</td>
<td>A container widget displayed as a separate, top-level window.</td>
</tr>
</tbody>
</table>
import tkMessageBox
name = 'foo.txt'
try:
  fp = open(name)
except:
  tkMessageBox.showwarning("Open file",
  "Cannot open this file\n(%s)" % name)
import tkMessageBox
print tkMessageBox.askquestion("Fire the missiles?")

'yes'
import Tkinter

root = Tkinter.Tk()

menubar = Tkinter.Menu()

def handle_click(menu, entry):
    print menu, entry

filemenu = Tkinter.Menu()

for x in 'Homer', 'Marge', 'Lisa', 'Bart':
    filemenu.add_command(label=x, command=lambda x=x:handle_click('Simpsons', x))

menubar.add_cascade(label='Simpsons', menu=filemenu)

root.config(menu=menubar)

Tkinter.mainloop()
Tkinter Variable Objects

- The Tkinter module supplies classes whose members are variables
  - DoubleVar, IntVar, StringVar
- You can pass a StringVar as the textvariable parameter for a widget
- When you do this, that widget will display changes to the variable
Tkinter Variable Objects

```python
import Tkinter

root = Tkinter.Tk()

tv = Tkinter.StringVar()
Tkinter.Label(textvariable=tv).pack()
Tkinter.Entry(textvariable=tv).pack()
tv.set('Hello tk variable world')
Tkinter.Button(text='Exit', command=root.quit).pack()
Tkinter.mainloop()
```
Tkinter Lists, Images, and Clicks
Tkinter Lists, Images, and Clicks

```python
import os, Tkinter

root = Tkinter.Tk()
L = Tkinter.Listbox(selectmode=Tkinter.SINGLE)
imgdict = {}
path = '/Users/josh/Desktop'
for name in os.listdir(path):
    if not name[-3:] == 'gif': continue
    imgpath = os.path.join(path, name)
    img = Tkinter.PhotoImage(file=imgpath)
    imgdict[name] = img
    L.insert(Tkinter.END, name)
L.pack()
...
```
Tkinter Lists, Images, and Clicks

...  

label = Tkinter.Label()
label.pack()

def list_entry_clicked(*ignore):
    name = L.get(L.curselection()[0])
    label.config(image=imgdict[name])

L.bind('<ButtonRelease-1>', list_entry_clicked)
root.mainloop()
Tkinter Lists and Scroll bars

Listbox can display textual items, selection capability
L.delete(0, END) # clear the box
L.insert(End, foo) # insert a string to the back

```python
import Tkinter
S = Tkinter.Scrollbar()
L = Tkinter.Listbox()
S.pack(side=Tkinter.RIGHT, fill=Tkinter.Y)
L.pack()
S.config(command=L.yview)
L.config(yscrollcommand=S.set)
for i in range(100):
    L.insert(Tkinter.END, str(i))
Tkinter.mainloop()
```
from Tkinter import *
root = Tk()
root.title('Radiobutton')
opts = [('Option 1', 1), ('Option 2', 2), ('Option 3', 3),
        ('Option 4', 4), ('Option 5', 5), ('Option 6', 6)]
var = IntVar()
def which():
    print var.get(), 'selected'
for text, value in opts:
    Radiobutton(root, text=text, value=value,
                variable=var, command=which).pack()
var.set(3)
root.mainloop()

4 Selected
5 Selected
from Tkinter import *

class LittleWrapper:
    def __init__(self, parent):
        frame = Frame(parent)
        frame.pack()
        self.button = \
            Button(frame, text="Bye", command=frame.quit)
        self.button.pack(side=LEFT)
        self.hi_there = \
            Button(frame, text="Hello", command=self.say_hi)
        self.hi_there.pack(side=LEFT)

    def say_hi(self):
        print "hi there, everyone!"

root = Tk()
app = LittleWrapper(root)
root.mainloop()
Frames

class LittleWrapper:
    def __init__(self, master):
        frame = Frame(master)
        frame.pack()

• The constructor (the __init__ method) is called with a parent widget (the master), to which it adds a number of child widgets.

• The constructor starts by creating a Frame widget. A frame is a simple container, and is in this case only used to hold the other two widgets.
Tkinter References

- http://docs.python.org/library/tkinter.html
- http://wiki.python.org/moin/TkInter