python

10 things in less than 20 lines that have saved me at least 30 minutes.

python

is not named after a snake

was created by van rossum

is owned by the psf

is copyrighted but free

is not protected by gpl

has ugly icons

is still cool

python

is object-oriented

is interactive

is interpreted

is dynamically-typed

is strongly-typed

python

1. language primer
2. those 10 examples
3. conclusions and future work

python - language primer

```
 1: #!/usr/bin/env python2.4
 2: """This is my first python, isn't he cute"""
 3:
 4: import string
 5:
 6: def sayhello(peeps,wordForHomies='homies'):
 7:     """Says hello to all your peeps"""
 8:     return "I want to send shoutouts to my %s: %s, and %s" % (
 9:             wordForHomies,
10:             string.join(peeps[:-1],', '),
11:             peeps[-1])
12:
13: if __name__ == '__main__':
14:     print sayhello(
15:         ("John","Carlos","Amber","Burak","Annie"),
16:         "labmates")
17:
```

my first python:

» whitespace indicates structure

» the "main" often looks strange by convention.

» there are special places to put your inline documentation.

string » constants, classes and functions for strings. most are also found in the string object.

python - language primer

```
 6: def sayhello(peeps,wordForHomies='homies'):
 7:     """Says hello to all your peeps"""
 8:     return "I want to send shoutouts to my %s: %s, and %s" % (
 9:             wordForHomies,
10:             string.join(peeps[:-1],', '),
11:             peeps[-1])
12:
```

functions:

» declared by the def keyword.

» can include inline docs.

» can have default params.

» mind the colon.

» by default None is returned.

__doc__ » inline documentation strings can be accessed through the field __doc__, as in sayhello.__doc__

python - language primer

running the code

» scripts often act as modules.

» “the __name__ trick” is pretty much a python convention.

```
13: if __name__ == '__main__':
14:     print sayhello(
15:         ("John","Carlos","Amber","Burak","Annie"),
16:         "labmates")
17:
```

__name__ » contains the name of the current module or ‘__main__’ if the file is being called as a script.

python - language primer

```
>>> a = "gourd"
>>> dir(a)
['__add__', '__class__', '__contains__', '__delattr__', '__doc__', '__eq__',
'__ge__', '__getattribute__', '__getitem__', '__getnewargs__', '__getslice__',
'__gt__', '__hash__', '__init__', '__le__', '__len__', '__lt__', '__mod__',
'__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',
'__rmod__', '__rmul__', '__setattr__', '__str__', 'capitalize', 'center', 'count',
'decode', 'encode', 'endswith', 'expandtabs', 'find', 'index', 'isalnum', 'isalpha', 'isdigit',
'islower', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'replace', 'rfind',
'rindex', 'rjust', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title',
'translate', 'upper', 'zfill']
>>> type(a)
<type 'str'>
```

variables

» exists when you set them.

» everything is an object.

» dir() is great for exploring.

» locals() is interesting too.

python - language primer

```
>>> j = {'Sunday':0,'Monday':1}
>>> j['Tuesday']=2
>>> for k,v in j.items():
        print "%d %s" % (v,k)
0 Sunday
2 Tuesday
1 Monday
>>> j
{'Sunday': 0, 'Tuesday': 2, 'Monday': 1}
```

dictionaries

» associative arrays.
» aka hashtables.
» tuples can be used as keys.
» lists cannot.
» items() is good for loops.

python - language primer

```
>>> a = [4,5,6,7]
>>> a.append(12)
>>> a
[4, 5, 6, 7, 12]
>>> tuple(a)
(4, 5, 6, 7, 12)
>>> b = (4,5,6)
>>> b.append(12)
Traceback (most recent call last):
  File "<input>", line 1, in ?
AttributeError: 'tuple' object has no attribute 'append'
>>> b
(4, 5, 6)
>>> list(b)
[4, 5, 6]
```

lists and tuples

» tuples are immutable.
» lists are not.
» tuples are more efficient.
» lists, less.
» tuples are hashable.
» lists are not.

python - language primer

```
>>> a = "not a snake"
>>> a[6]
's'
>>> a[6:8]
'sn'
>>> a[6:]
'snake'
>>> a[:6]
'not a '
>>> a[:]
'not a snake'
```

slicing

- [start_index : end_index]
- creates sub-lists or tuples.
- and strings.
- [:] makes a copy.

python - language primer

```
>>> a = "%s/%d.jpg" % (os.getcwd(),45)
>>> a
'/Users/knorton/45.jpg'
```

formatting

» formats vars into strings.

» sort of like sprintf.

» in fact is uses the same %'s.

» format % (tuple)

python - language primer

```
 1: #!/usr/local/bin/python
 2: class SomethingThatPlays:
 3:     def play(self):
 4:     print "do da do da"
 5:
 6: class Song(SomethingThatPlays):
 7:     def __init__(self,title,artist):
 8:     self.__title = title
 9:     self.__artist = artist
10:
11:     def __str__(self):
12:     return "%s by %s" % (self.__title,self.__artist)
13:
14:     def get_artist(self):
15:     return self.__artist
16:
17:     def get_title(self):
18:     return self.__title
19:
20:
```

classes

» declared with class keyword.

» self is always the first param.

» privates begin with __.

» __str__ is like java toString.

python - language primer

```
 1: #!/usr/local/bin/python
 2:
 3:
 4: try:
 5:     fh = open('not a file')
 6:     print fh.read()
 7:     fh.close()
 8: except IOError, (errno,strerror):
 9:     print 'zzzt: %s' % strerror
10:     raise IOError, (errno,"It's so broken")
11:
```

exceptions

» try … except

» try … finally

» can't have try … except … finally, but you can nest.

python

10 examples

python - examples

```
 1: #!/usr/bin/env python2.4
 2:
 3: from urllib import urlopen
 4: from HTMLParser import HTMLParser
 5:
 6: class LinkFinder(HTMLParser):
 7:     def __init__(self):
 8:         HTMLParser.__init__(self)
 9:         self.links = []
10:
11:     def handle_starttag(self,tag,attr):
12:         if tag.lower()=='a':
13:             attr = dict(attr)
14:             if attr.has_key('href'):
15:                 self.links.append(attr['href'])
16:
17: def get_links(url):
18:     f = LinkFinder()
19:     f.feed(urlopen(url).read())
20:     return f.links
21:
```

simple web scraper

» get_links will return a list of all the links found on the specified web page.

» LinkFinder is a subclass of HTMLParser that keeps track of links as they are found in the html.

» I use similar scripts to leech, er, archive things from download sites.

urllib » provides simple means for fetching data over the web.

HTMLParser » includes a base class for parsing html into individual elements.

python - examples

```
 1: #!/usr/bin/env python2.4
 2:
 3: import SimpleHTTPServer
 4: import SocketServer
 5:
 6: PORT=9090
 7:
 8: httpd = SocketServer.TCPServer(
 9:     ("",PORT),
10:     SimpleHTTPServer.SimpleHTTPRequestHandler)
11: httpd.serve_forever()
12:
```

simple web server

» will server up files in the current directory.

» add a custom handler to add functionality beyond static pages.

» just remember this isn't an industrial strength web server.

» i use this while designing web sites to preview pages.

SimpleHTTPServer

» provides a really basic web server.

SocketServer » is a general-purpose, modular TCP server.

python - examples

```
 1: #!/usr/bin/python
 2:
 3: from random import randint
 4: from SimpleXMLRPCServer import *
 5:
 6: def register_functions(handler):
 7:     handler.register_function(randint);
 8:     handler.register_introspection_functions()
 9:
10: if os.environ.has_key('SERVER_SOFTWARE'):
11:     handler = CGIXMLRPCRequestHandler()
12:     register_functions(handler)
13:     handler.handle_request()
14: else:
15:     handler = SimpleXMLRPCServer(("",8080))
16:     register_functions(handler)
17:     handler.serve_forever()
18:
```

simple xmlrpc server

» implements a web service for generating random integers.

» will work as either a cgi script or a stand alone server.

» register_functions tells the xmlrpc handler which functions to make available as web services.

» the imac cluster in plw uses this to lauch applications on all the machines.

random » includes many functions for seeding, and generating random numbers.

SimpleXMLRPCServer » provides classes for handling xmlrpc requests either by cgi or in the provided stand-along server.

python - examples

```
 1: #!/usr/bin/env python2.4
 2:
 3: from xml.dom.minidom import parseString
 4: from urllib import urlopen
 5:
 6: dom = parseString(
 7:     urlopen("http://boingboing.net/index.rdf").read())
 8:
 9: for t in [ \
10:     x.firstChild.nodeValue for x in \
11:     dom.getElementsByTagName('title')]:
12:     print t
13:
```

rss summarizer

» retreives the titles of current articles on boingboing.net.

» retreives and rss file and parses it as xml.

» uses standard dom functions to find all the title elements.

minidom » designed to be a lightweight version of the document object model interface.

python - examples

```
1: import tarfile
2:
3: def readlines(tf):
4:     untar = lambda x:tarfile.open(x,'r|gz')
5:     for log in [l for l in untar(tf).getmembers() if l.isfile()]
6:         for line in untar(tf).extractfile(log).readlines():
7:             yield line
8:
```

digging in tar files

» reads log files that are stored in a zipped tar file.

» uses a generator to keep from returning a really large list.

» this technique has more than paid for itself, i've tested it up to 80M lines of data.

tarfile » provides functions and classes for inspecting and accessing tar files.

python - examples

```
 1: #!/usr/bin/env python2.4
 2:
 3: import re
 4: from syslog import readlines
 5:
 6: users=[]
 7: p = re.compile("(Illegal user|Failed password for) (\S+)")
 8: for l in readlines('system.log.tgz'):
 9:     m = p.search(l)
10:     if m:
11:         user = m.groups()[1]
12:         if not user in users:
13:             users.append(user)
14:
15: print "I found %d usernames in the logs" % len(users)
16:
```

regular expressions

» uses the function from the last example.

» checks each line to see if it fits a pattern.

» extracts the username if the line was about a failed login attempt.

» notice the in keyword can search lists for you.

re » python's regular expression module. regular expressions provide very powerful text pattern matching.

python - examples

```
 1: #!/usr/bin/env python2.4
 2:
 3: import re, os, pickle
 4: from syslog import readlines
 5:
 6: if os.path.exists('usernames.p'):
 7:     users = pickle.load(open('usernames.p','rb'))
 8: else:
 9:     users=[]
10:     p = re.compile("(Illegal user|Failed password for) (\S+)")
11:     for l in readlines('system.log.tgz'):
12:         m = p.search(l)
13:         if m:
14:             user = m.groups()[1]
15:             if not user in users:
16:                 users.append(user)
17:     pickle.dump(users,open('usernames.p','wb'))
18:
19: print "I found %d usernames in the logs" % len(users)
20:
```

pickling data

» pickling just means serializing a data structure to be loaded up later.

» this is the same functionality as the last example only it uses pickle to cache results.

os » provides an interface to miscellaneous operating system functions.

pickle » provides object serialization for saving objects in files or sending them over the network.

python - examples

```
 1: #include <Python.h>
 2:
 3: int getmacraddr(char* szBuffer, int nBuffer);
 4:
 5: PyObject *wrap_getmacaddr(PyObject *self,PyObject *args) {
 6:     char    addr[255];
 7:
 8:     if (getmacraddr(addr,255) != 0) {
 9:         Py_INCREF(Py_None);
10:         return Py_None;
11:     }
12:     return PyString_FromString(addr);
13: }
14:
15: static PyMethodDef methods[] = {
16:     {"getMacAddress", wrap_getmacaddr, METH_VARARGS},
17:     {NULL,NULL}};
18:
19: void initnetwork(void) {
20:     Py_InitModule("network",methods);
21: }
22:
```

native extensions

» python tries to make native extensions easy.

» init<modulename> is used to tell python what functions are in the module.

Python.h » this includes everything you need to work the native side of python.

python - examples

```
1: from distutils.core import setup, Extension
2:
3: setup(
4:     ext_modules=[
5:         Extension("network",
6:             sources = ['network.c','getmacaddr.c'])])
7:
```

native extensions

» python can even take care of compiling the native code for you.

» setup provides a fairly complete build and install system.

» this builds our native code into a shared library that python can load seamlessly as a module.

distutils » a set of tools that are meant to take the guess work out of building and installing python modules.

python - examples

```
 1: #!/usr/bin/env python
 2:
 3: import sys, os, string
 4: from CoreGraphics import *
 5:
 6: g_template="typical"
 7: def py2pdf(f,tmpl=g_template):
 8:     (stdin,stdout) = os.popen2( \
 9: "/usr/local/bin/webcpp %s -" \
10: " -c=/usr/local/share/webcpp/%s.Scs2 -l -t=4" \
11:  % (f,tmpl))
12:     dat = CGDataProviderCreateWithString(stdout.read())
13:     rec = CGRectMake(0,0,792,612)
14:     pdf = CGPDFContextCreateWithFilename("%s.pdf" % f,rec)
15:     pdf.beginPage(rec)
16:     pdf.drawHTMLTextInRect(dat,rec,12.0)
17:     pdf.endPage()
18:     pdf.finish()
19:
20: if __name__ == '__main__':
21:     map(py2pdf,sys.argv[1:])
22:
```

the making of ...

» i needed a way to get nice pdf's of the code i had for this talk.

» webcpp is used to create colorful html by calling the illustrious os.popen2 function.

» notice how this file serves as both a script and a module.

CoreGraphics » OSX only module that provides bindings to some CoreGraphics functionality.

python - goodies

python.org
python.org
the python hub.

activestate
activestate.com
good python distributions for all major platforms.

pythonmac.org
pythonmac.org
pythonmac.org/packages
the place to go for osx packages.

bob ippolito's python stuff
undefined.org/python
very good python distributions for osx; the one i currently use.

dive into python
diveintopython.org
mark pilgrim's free online python book.

python 2.4.1 docs
python.org/doc/2.4.1
python documentation is generally fair to good.

python software fondation
http://python.org/psf/
benevolent owners of python.

pygame
pygame.org
graphics and game modules all bundled into one.