Python and WWW
We interrupt this program to annoy you and make things generally irritating

Peter Meerwald-Stadler

April 24, 2020

Agenda

Introduction (Concepts, data types, . . .)

CGI

Strings, pattern matching & regular expressions

Session & Cookies, Screen scraping

Database & XML

Interface a database

About...

- `http://www.python.org`
- Created circa 1990 by Guido van Rossum, Benevolent Dictator for Life
- Simple, intuative, dynamic language with OO and functional elements
- Open source, current version 3.8; implementations in C, Java (Jython), .net (IronPython) and Python (PyPy)
- Version 2.7.18 is the last release for Python 2 (end-of-life)

What to do with Python?

Learn it as a first programming language
Machine Learning & AI
Scripts, CGI, glue code, embedded language, . . .
Zope (Application server)
Plone (Content Management System)
Django, TurboGears (Framework)
Mercurial (Source repository)
NumPy & SciPy (Scientific computation)
youtube.com, google.com

hello, world

```
#!/usr/bin/env python
def say_hello():
    """Just prints 'hello, world'.
    Know what Python refers to, BTW?"""
    print('hello, world')

say_hello()
```

Overview

- Basics (data types, typing, flow control, function, modules)
- Data structures (list, tuple, dictionary)
- Strings
- Classes
- Standard Library
- Exceptions

Data types

```
s = 'this is ''quoted'''
t = ''new\nline''
i = 10
b = True
j = 3.14
tuple = (1,2)
list = ['a', 'b', 3, ('d', 5)]
dictionary = {'To':  'guido', 'length':  128}
```

Flow Control

```
i = 0
while True:
    if i == 0 or i == 1:
        print('zero or one')
    elif i < 10:  print('small')
    elif i == 15:  break
    else:
        print('big')
    i += 1
print('i = %d' % (i))
```

Functions

```
def f(i):
    pass
def g(x=[]):
    return len(x)

print(g('blabla'))
print(g())
print(g(['b', 'l', 'a']))
```

Modules

```
import sys
sys.exit(0)
```

```
from sys import exit
exit(0)
```

```
from sys import *
exit(0)
```

(Duck) Typing

"If it walks like a duck and quacks like a duck, it must be a duck"

- Emphasize interface over type, allow polymorphism
- Avoid type checks, catch an exception if it fails

```
>>> def times10(x):
  return x * 10
>>> times10(3)
30
>>> times10('-')
----------
```

Lists

```
>>> l = ['spam', 'eggs', 3.14, 42]
>>> l[0]
'spam'
>>> l[-2]
3.14
>>> l[1:]
['eggs', 3.14, 42]
>>> l[:2] + ['bacon', 'pepper']
['spam', 'eggs', 'bacon', 'pepper']
>>> len(l)
4
>>> l.append(100)
['spam', 'eggs', 3.14, 42, 100]
```

List Comprehension / Looping

```
>>> v = [1,2,3,4]
>>> len(v)
4
>>> [x*x for x in v]
[1, 4, 9, 16]
>>> [x*x for x in v if x > 2]
[9,16]
>>> for i in xrange(1,10,2):  print i
1,3,5,7,9
```

Dictionaries

```
>>> weight = {'galahad':  78, 'lancelot':  80, 'arthur':  101}
>>> weight['arthur']
101
>>> weight.has_key('bjelli')
False
>>> weight['god'] = 0
>>> weight.keys()
['god','arthur', 'galahad', 'lancelot']
```

Extending a Web Server with Python

- CGI - create new process handling one request
- Fast CGI - process(es) handling many requests
- `mod_python` - interpreter running embedded in the web server
- Web Server in Python (`BaseHTTPServer`, `Twisted`, etc.)

Fast CGI

Instead of creating a new process for every request, persistent process(es) handle many requests, communication over sockets.

By Open Market, Inc., circa 1996: `http://www.fastcgi.com`

Wide adoption, but no standard.

Need language binding, eg. `http://jonpy.sourceforge.net/fcgi.html`

```
import jon.cgi as cgi
import jon.fcgi as fcgi
class Handler(cgi.Handler):
    def process(self, req):
        req.set_header("Content-Type", "text/plain")
        req.write("Hello, world!\n")
fcgi.Server({fcgi.FCGI_RESPONDER: Handler}).run()
```

---

mod_python

```
LoadModule python_module libexec/mod_python.so
<Directory /some/directory/htdocs/test>
    AddHandler mod_python .py
    PythonHandler mptest
    PythonDebug On
</Directory>
```

```
from mod_python import apache
def handler(req):
    req.write("Hello World!")
    return apache.OK
```

Server in Python

```
import BaseHTTPServer
import CGIHTTPServer
httpd = BaseHTTPServer.HTTPServer(\
    ('localhost', 8123), \
    CGIHTTPServer.CGIHTTPRequestHandler)
httpd.serve_forever()
```

Common Gateway Interface (CGI)

Input:

- Environment variables (QUERY_STRING, CONTENT_LENGTH, REQUEST_METHOD, REMOTE_ADDR, request header lines as HTTP_xxx)
- Command-line arguments (not used)
- stdin (for HTTP POST)

Output:

- Response to stdout

```
print('Content-type:  text/html\n')
print('<html>...')
```

- Logging to stderr (may appear in web server's log file)

cgitb and cgi Module

```
import cgitb
cgitb.enable() # special exception handler, formatted report
import cgi
form = cgi.FieldStorage()
name = form.getvalue('name') # attention!  can return a list...
addr = form.getfirst('addr', 'N/A')
print cgi.escape(addr) # converts &, <, > to HTML-safe sequences
```

FieldStorage Details

```
import cgi
form = cgi.FieldStorage() # only instantiate once
form.name
form.type # eg.  application/x-www-form-urlencoded
form.headers # dictionary of all headers
form['name'].value
form['name'].file # file handle
```

Environment Variables

```
import os
os.environ['HTTP_ACCEPT']
# text/xml,application/xml,application/xhtml+xml,...
# text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
os.environ['REMOTE_ADDR']
# 127.0.0.1
os.environ['REQUEST_METHOD']
# GET
os.environ['HTTP_USER_AGENT']
# Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.7.10) ...
```

File Upload

```
<html><body>
<form enctype="multipart/form-data" action="fileupload.py" method="post">
<p>File:  <input type="file"  name="file"></p>
<p><input type="submit" value="Upload"></p>
</form></body></html>
```

```
form = cgi.FieldStorage()
fileitem = form['file']
# strip leading path from file name to avoid directory traversal attacks
fn = os.path.basename(fileitem.filename)
open('files/' + fn, 'wb').write(fileitem.file.read())
```

String Handling

old style: `import string; string.find(str, pattern)`

new style: `str.find(substr)`

```
''%s circus'' % (''flying'') # string formatting
r''bl\a'' # raw string, escape sequences ignored
index(substr), rindex(substr) # raises ValueError
find(substr), rfind(substr) # -1 on failure
replace(old, new) # replace all substrings old with new
count(substr) # occurrences of substr
upper(), lower() # convert to upper/lower case
substr in str # True if substr part of str
strip(chars), lstrip(chars), rstrip(chars) # default chars is all whitespace
split(separator), join(separator) # default separator is all whitespace
```

Regular Expression

. . . is a string used to describe or match a set of strings, according to certain syntax rules.

Special Character	Meaning
`bla`	Matches the string `bla`
`.`	Matches any character except newline
`^`	Matches the start of the string
`$`	Matches the end of the string
`*`	Any number of occurrences of what just preceded me
`+`	One or more occurrences of what just preceded me
`|`	Either the thing before or after me
`\w, \d, \s`	Matches alphanumeric, decimal, whitespace character
`\\`	Matches backslash character
?, (), [], . . .	Option, grouping, characters, . . .

RE Examples

`.at`	hat, cat, bat, . . .
`[hc]at`	hat, cat
su+per	super, suuper, suuuper, . . .
`-?\d+`	all integers
`^\s*;`	see if ; is at beginning of string
`i\+\+|(=1)`	i++, i+=1
`((great )*grand )?((fa|mo)ther)`	father, mother, grand father, grand mother, great grand father, . . .

RE in Python

See http://www.amk.ca/python/howto/regex/.

```
import re
if re.match('su+per(duper)?', 'suuuper'):  print('matches')
ex = re.compile(r'<(?P<sometag>\w+)>$')
m = ex.match('<xml><bla>')
if not m:  print('no')
m = ex.search('<html><bla>')
if m:  print(m.groupdict()['sometag']) # 'bla'
```

File Parsing

```
import string, random
f = open('csv.txt', 'rt') # contains lines words separated with comma
for line in f:
    words = line.split(','); random.shuffle(words)
    print(string.join(words, ';'), end='') # output permuted words with ;
```

```
import fileinput, re, sys
comment = re.compile(r'^\s*;|#')
for line in fileinput.input(sys.argv[1:]):
    if comment.match(line):  continue # skip lines beginning with ; or #
    print(line, end='')
```

Common Log Format

```
127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700] "GET /apache_pb.gif HTTP/1.0" 200 2326
```

remote host

ident (RFC 1413)

login as which the user authenticated

time and date

request line

status code (200, 404)

bytes (content-length) transferred

Representational State Transfer (REST)

Roy Fielding, 2000
Architectural principles for hypermedia systems (eg. web)

- State and functionality divided into resources
- Uniquely addressable resources (URIs)
- Uniform interface to resources (GET, POST, headers, status codes)
- Protocol which is client/server, stateless, cacheable, layered

Sessions

need state, violates REST; eg. site login, shopping cart – solutions:

- IP address
- URL (query string)
- Hidden form fields
- Cookies
- HTTP authentication (eg. digest access authentication, RFC 2617)
- Macromedia Flash Local Stored Objects
- Client-side persistence: (ab)use JavaScript to modify browser state

Cookies

see RFC 2965, "HTTP State Management Mechanism":

Information sent from a server to a browser which the browser stores and sends back unchanged each time it accesses that server.

Cookies are set by server (ie. CGI script) or client (via JavaScript) and communicated in HTTP request (`Cookie: name=value`) and response (`Set-Cookie: name=value`) headers.

Cookies can have attributes such as `Expires` or `Max-age`, `Domain`, `Path`, `Secure`, . . .

Cookies Server-Side

```
import os, cgi
from http import cookie
try:  cookie = cookie.SimpleCookie(os.environ['HTTP_COOKIE'])
except:  cookie = cookie.SimpleCookie()
try:
    sid = cookie['sid'].value
    msg = 'Welcome back, %s!' % (sid) # returning user
except:
    cookie['sid'] = str(time.time()) # easily predicable, not good!
    msg = 'Let me introduce myself...'  # new user
print 'Content-type:  text/html'
print 'Expires:  0'
print cookie
print('\n<html>%s</html>' % (msg))
```

Cookies Client-Side

```
import urllib2, ClientCookie
cookie_file = 'mycookies'
cj = ClientCookie.LWPCookieJar()
if os.path.isfile(cookie_file):  cj.load(cookie_file)
opener = ClientCookie.build_opener( \
    ClientCookie.HTTPCookieProcessor(cj))
ClientCookie.install_opener(opener)
resp = ClientCookie.urlopen('http://google.at')
print resp.info()
for index, cookie in enumerate(cj):  print index, ':', cookie
cj.save(cookie_file)
```

Screen Scraping using Mechanize

mechanize: controllable browser with support for proxies, cookies, HTML parsing, form decoding, . . . see `http://wwwsearch.sourceforge.net/mechanize`

```
import mechanize
br = mechanize.Browser()
br.set_debug_http(True)
br.open('http://studivz.net')
br.follow_link('text=einloggen')
br.select_form(predicate=lambda form:  True)
br['email']='me@host'
br['pass']='secret'
br.submit()
print(br.title())
```

Persistance

Databases solve many issues storing data , eg. query language (SQL), multiple clients, concurrency, transactions, integrity, . . .

But: not object-oriented, not Python objects

Python offers support for object serialization: eg. pickle, shelve modules

SQLAlchemy or SQLObject: high-level, object-oriented, database-independent Python interfaces

Python DB API

Standard API for database access:

- Module interface: `connect(...)`
- Connection objects: `cursor()`, `commit()`, `rollback()`, `close()`
- Cursor objects: `rowcount`, `description`, `execute()`, `callproc()`, `close()`, `fetchone()`, `fetchmany()`, `fetchall()`, `next()`
- Constructors: `Date()`, `Time()`, `TimestampFromTick()`, `Binary()`
- Exceptions

See PEP 249: `http://www.python.org/dev/peps/pep-0249/`.

MySQL

Popular open-source RDBMS.

Has transactions and relational integrity constraints ('ACID').

Has command-line interface or web-based administration tool (eg. phpMyAdmin).

Alternatives: Postgres, SQLite, Oracle, . . .

MySQLdb Example

See http://mysql-python.sourceforge.net

```
import MySQLdb
db = MySQLdb.connect(host='localhost', user='brigitte',
    passwd='brigitte')
c = db.cursor()
c.execute('SELECT * FROM brigitte.buch')
for row in c:
    print(row)
c.execute('SELECT titel, autor FROM brigitte.buch
    WHERE erscheinungsdatum < %s', (Date(2000,1,1),))
print(c.fetchone())
```

SQL Injection

SQL injection is a security vulnerability in the database layer of an application. Unfilted user input embedded in SQL statements may be unexpectedly executed.

Example:

```
''SELECT * FROM users WHERE name = ''' + username + ''';''
```

`username = ''a' or 't'='t''`, or

```
username = ''a'; DROP TABLE users; SELECT * FROM data
    WHERE name LIKE '%''
```

```
''SELECT * FROM users WHERE id = '' + id + '';''
id = ''1; DROP TABLE users''
```

XML & SOAP

XML is a general-purpose markup language to facilitate sharing of data.

SAX (event oriented) and DOM (document as hierarchical structure) APIs; standardized (W3C DOM level 1 and 2), but not very Pythonic.

See also ElementTree: `http://effbot.org/zone/element-index.htm`.

SOAP is a protocol for exchanging XML messages, usually over HTTP, to build Web services. Remote Procedure Call (RPC) is the most common message pattern: client sends reuqest to server, server immediately replies.

WSDL (Web Service Description Language) is an XML-based language for describing Web services.

Creating XML

```
import xml.dom.minidom, xml.dom.ext, sys
class Book:
    def __init__(self, id, titel, autor):
        self.id = id; self.titel = titel; self.autor = autor
    def asDOMElement(self, doc):
        book = doc.createElement('book')
        book.setAttribute('id', str(self.id))
        book.appendChild(doc.createElement('titel')) \
            .appendChild(doc.createTextNode(self.titel))
        book.appendChild(doc.createElement('autor')) \
            .appendChild(doc.createTextNode(self.autor))
        return book
doc = xml.dom.minidom.Document()
lib_elem = doc.appendChild(doc.createElement('library'))
lib_elem.appendChild(Book(1, 'Easy Perl', 'Brigitte Jellinek').asDOMElement(doc))
lib_elem.appendChild(Book(2, 'Learning Python', 'Mark Lutz').asDOMElement(doc))
xml.dom.ext.PrettyPrint(doc, sys.stdout)
```

Parsing XML

```
import xml.dom.minidom
xml_str = '''
<library> <book id='1'>
    <titel>Easy Perl</titel>
    ...
'''
doc = xml.dom.minidom.parseString(xml_str)
books = doc.getElementsByTagName('book')
for book in books:
    print('%d:  %s, %s' % (\
        int(book.getAttribute('id')), \
        book.getElementsByTagName('titel')[0].childNodes[0].data, \
        book.getElementsByTagName('autor')[0].childNodes[0].data))
```

Example: Yahoo Weather RSS feed

See http://developer.yahoo.com/weather/.

```
import urllib2, xml.dom.minidom
WEATHER_URL = 'http://weather.yahoo.com/forecastrss?p=%s&u=%s'
def weather_for_zip(zip_code, unit='c'):
    url = WEATHER_URL % (zip_code, unit)
    dom = xml.dom.minidom.parse(urllib2.urlopen(url))
    ycond = dom.getElementsByTagName('yweather:condition')[0]
    return {'current_temp': ycond.getAttribute('temp'),
    'title': dom.getElementsByTagName('title')[0].firstChild.data}

weather = weather_for_zip('AUXX0018')
print('%s\nCurrent temperature: %s C' % \
    (weather['title'], weather['current_temp']))
```

Web Application Frameworks

Software framework to support development of dynamic web sites by providing libraries for database access, templating, session management.

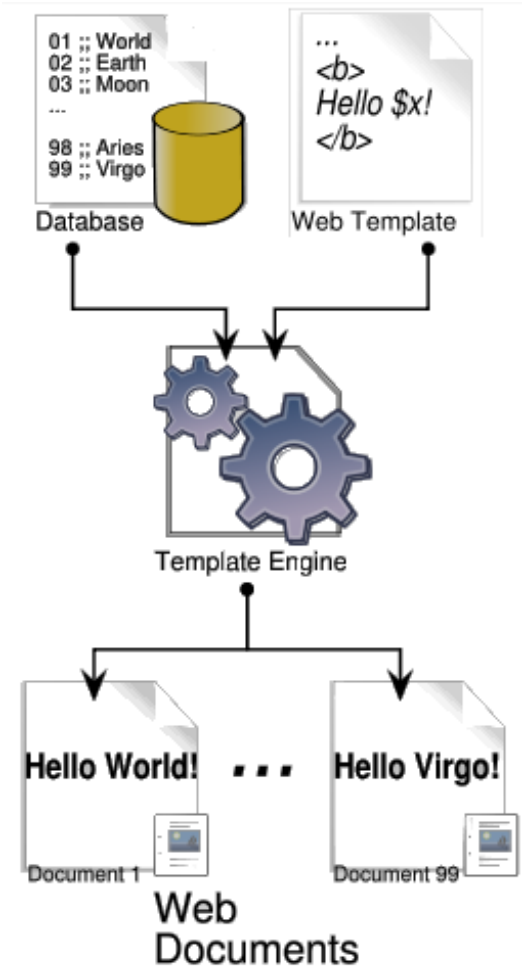
Variety for many languages: JBoss, Struts (Java), Drupal (PHP), Django (Python), Ruby on Rails.

See `http://en.wikipedia.org/wiki/Comparison_of_web_application_frameworks`.

Typical Features

- Use Model View Controller (MVC) architectural pattern to separate data model, business rules and user interface.
- Object-Relational mapping. Map objects to DB tuples.
- URL mapping. Rewrite URLs to support REST. Map eg. `?cat=1&pageid=3` to `/interests/chocolate`.
- Template system. Reduce number of static pages.
- Caching. Ajax. Automatic configuration.

Template System



Model / View / Controller

Typical process flow:

- The user interacts with the HTML interface in some way (Form, Link).
- A controller handles the input event from the user interface, often via a registered handler or callback. (Web server, URL matching, CGI).
- The controller accesses the model (DB), possibly updating it in a way appropriate to the user's action.
- A view uses the model to generate an appropriate user interface (Template). The model has no direct knowledge of the view.
- The user interface waits for further user interactions, which begins the cycle anew.

Django

See `http://www.djangoproject.com`.
BSD license since 2005.
DRY, MVC & CRUD.

Supports MySQL, Postgres, SQLite, MSSQL.
Built-in development server. Deploy on Apache mod_python or FCGI.
Automatically generates administration interface (CRUD).
Automatically reloads code on change.
Command-line management tool: `django-admin.py`, `manage.py`.

Django: Model

```
from django.db import models
class Poll(models.Model):
    question = models.CharField(maxlength=200)
    pub_date = models.DateTimeField('date published')
class Choice(models.Model):
    poll = models.ForeignKey(Poll, edit_inline=models.TABULAR, num_in_admin=3)
    choice = models.CharField(maxlength=200, core=True)
    votes = models.IntegerField(core=True)
```

```
> python manage.py sql polls
CREATE TABLE "polls_poll" (
    "id" integer NOT NULL PRIMARY KEY,
    "question" varchar(200) NOT NULL, "pub_date" datetime NOT NULL );
CREATE TABLE "polls_choice" (
    "id" integer NOT NULL PRIMARY KEY,
    "poll_id" integer NOT NULL REFERENCES "polls_poll" ("id"),
    "choice" varchar(200) NOT NULL,
    "votes" integer NOT NULL );
```

Django: URL matching

Resolve request `http://www.mysite.com/polls/1/results` by calling `mysite/polls/views.py:results(req, 1)`.

```
from django.conf.urls.defaults import *
urlpatterns = patterns('mysite.polls.views',
    (r'^$', 'index'),
    (r'^(?P<poll_id>\d+)/$', 'detail'),
    (r'^(?P<poll_id>\d+)/results/$', 'results'),
    (r'^(?P<poll_id>\d+)/vote/$', 'vote'),
)
```

Django: Controller / View

```
# polls/views.py
from django.shortcuts import render_to_response,get_object_or_404

from mysite.polls.models import Choice, Poll
def index(request):
    latest_poll_list = Poll.objects.all().order_by('-pub_date')[:5]
    return render_to_response('polls/index.html',
        {'latest_poll_list':  latest_poll_list})
def results(request, poll_id):
    p = get_object_or_404(Poll, pk=poll_id)
    return render_to_response('polls/results.html', {'poll':  p})
```

Django: Templates

Generate HTML for `http://www.mysite.com/polls/1/results` request.

```
<!-- polls/results.html -->
<h1>{{ poll.question }}</h1>
<ul>
{% for choice in poll.choice_set.all %}
    <li>{{ choice.choice }} – {{ choice.votes }}
        vote{{ choice.votes|pluralize }}</li>
{% endfor %}
</ul>
```

Bibliography

Mark Lutz, David Ascher, Learning Python, 2nd ed., O'Reilly, 2003.
Alex Martinelli, Python in a nutshell, 2nd ed., O'Reilly, 2006.