Top of Form



Bottom of Form

Top of Form



**Python 2: Getting More Out of Python  
Lesson 13, Project 1**

Handed in: 12 May 2015 10:03:39PM Graded: 13 May 2015 05:57:28PM

**Here are your instructions:**

Write a program that imports the following names from a "settings" module:

RECIPIENTS   a list of (name, email-address) tuples

STARTTIME    datetime.datetime object for first message

DAYCOUNT    number of daily message generations

The program should produce a message of the format:

Date: {{date}}

From: <a href="mailto:website@example.com">website@example.com</a>

To: {{recipient}}

Message-Id: <NNNNNN>

This is a test message.

Your program should save these messages in the messages table.

Use test-driven development, and state the purpose of each test in the suite in docstrings that will eventually document your program.

Time your program for DAYCOUNTS of 1, 10, 50, 100, and 500 and plot the results (on a sheet of paper). How reliable are the timings?

Think of it like this: You are soon to go on vacation, at STARTTIME, for DAYCOUNT days, and you want your co-workers (RECIPIENTS) to continue getting your famous Joke of the Day (JOTD).

Your strategy is to store up the emails ahead of time, predated with the date they're to be sent. So if you leave on vacation on Jan 3, 2013, the first set of emails might be dated Jan 4 (each recipient gets one), then Jan 5 and so on, for DAYCOUNT days.

A good test that you have the right number is DAYCOUNT \* len(RECIPIENTS) should equal SELECT COUNT(\*) FROM jotd\_emails; that is, the total number of days you're on vacation times the number of receivers, should equal the total number of records in the table generated. Of course, this will only be true if your To: line is only to a single recipient, and not all of them separated by commas.

Storing the right date for each email will likely involve using a timedelta to increment a datetime by one day at a time for DAYCOUNT days.

Regarding timing, it's enough to count under your breath and give a sense in your remarks about how you think time might be a function of DAYCOUNT. If you have your email generating and storing function where you might conveniently go:

start = time.time()

call\_function(args)

end = time.time()

interval = end - start

print("Time to complete: ", end)

Then you could also give some hard numbers as to the relative times the program took as a function of changing DAYCOUNT. The purpose of this requirement is to look ahead to future projects where timing / profiling is a core focus.

**Your Comment:**

Pat,

Here is my Python 2 Opus :D. Included in the src folder should be to .png files that show times

plotted against number of messages generated (the one with green dots; message\_times.png)

which shows a slight exponential signature (though each time you run the code the results change

slightly, so at times it looked linear). But then I was thinking that we should really be looking at

the rate of change for each iteration relative to the one before to get a better sense. Generating

1 message is lightning fast, so the result is about 1e13 messages per second, so I put the second

.png file on a Log10 scale to get a better sense of what it does (the one with red dots; mps.png).

There is some latency as we'd expect since Python can't really know in advance how many

resources it has to allocate until it's 'go-time' (run-time), so I'm inclined to believe that 1 message

doesn't have any effect on performance, but once it creates one, it knows more are on the way

so it grabs whatever resources are free and throws them at the next iteration which it handles

10 messages with each, then as the number of messages grows, the rate of change shows that it

seems to level out.

Anyway, I'd include the graphing code in my test suite, however matplotlib is not installed on

the virtual environment (I'm assuming because numpy, which it depends on, is a bear to install

correctly without a PhD in computer science :D- damn LAPACK and BLAS libraries!)

Tear into it and let me know how to improve it. The testing script is testGoneFishin.py. The main

program (gone\_fishin.py) is fully functional as a standalone program, so my testing suites were

built to conform to the specifications for how one might run the program if they were actually

using it to store emails, etc.

Looking forward to your comments as always!

-Jason

**Items Handed In**

* [Open Project Handed In](https://students.oreillyschool.com/student/project/?/.handin/160-7927-1/com.ost.jwoloson.160.7927.1.EmailSearch_Homework.zip)

**Overall Comments:**

Hi Jason,

This is completely outstanding. You put together one of my all-time favorite implementations of this. Great going. I would tear into it, but it stands tall on its own merits, as it stands.

The rest of this email is pretty long. So that things don't get lost in the clutter, be on the outlook for:

-a recipe

-an alternative implementation

-a contact

-congratulations

I know what you mean about numpy. It's a total pain in the ass to get going and much can go wrong. I have successfully installed it on a few of my virtuals, most recently on Ubuntu (under VirtualBox), in order to get pandas up and running. With the usual disclaimers, here is the recipe I created as I went along:

Strategy: Work from requirements.txt from a iPython notebook server, figure out what's needed, but not already part of nupic. Avoid conflicts at all costs.

Additional core system requirement:

sudo apt-get install libfreetype6-devpip install --no-deps -r 1.7.1

#cd pandas

#python setup.py build

#python setup.py install

installed pandas with no deps to keep current version of nupy in play:

pip install pandas --no-deps

The dependencies are on the main pandas home page:

http://pandas.pydata.org/pandas-docs/dev/install.html#installing-from-source

...thinking we'll install them individually.

AND IT WORKS.

====================

I don't know if you are interested, but I got the most interesting email yesterday. In part it reads:

"Hi Pat,

I saw that you teach python courses at the O’Reilly School of Technology and I wanted to reach out to you and see if there are any python developers that have recently finished or are about to finish courses that you would recommend I reach out to? I am working on multiple junior and senior level python developer positions for a national healthcare company. We just placed someone with them who received their python certificate through your program and he was extremely knowledgeable. If there’s anyone that comes to mind that you’d recommend I reach out to, I would greatly appreciate it! "

... if you are interested, I would be happy to recommend that these guys contact you.

====================

For your inspection and amusement, here's an alternative implementation of this project from another student. This approach uses a joke server (api.icndb.com) to provide an unlimited number of Chuck Norris jokes (and who thought Chuck Norris could be funny?)

##joke\_service.py

'''

Provides a joke service, HTTPJokeService, which retrieve jokes from an API.

HTTPJokeService is currently tightly coupled to api.icndb.com.

'''

import json

import html

from urllib import request

from urllib.parse import urlencode

class JokeError(Exception): pass

class HTTPJokeService(object):

def \_\_init\_\_(self, source, options):

'''Initializes the service. The first argument is the source API, and

the second is a dictionary of GET parameters that should be used.'''

self.source = source

self.options = options

def get\_jokes(self, jokes=1):

'''Retrieve a list of jokes from the service, where "jokes" is the

number of jokes to retrieve. By default, returns 1 joke.'''

# The icndb API likes to use "[]" to denote a list of parameter values,

# so don't quote those characters.

options = urlencode(self.options, safe='[]')

url = '{0}/{1}?{2}'.format(self.source, jokes, options)

joke\_bytes = request.urlopen(url).read()

jokes = json.loads(joke\_bytes.decode('utf-8'))

if jokes['type'] != 'success':

raise JokeError('Error retrieving jokes from the API')

# Some of the jokes have HTML entities in them, like """, so use

# html.unescape to decode them.

return [html.unescape(joke['joke']) for joke in jokes['value']]

######jotd.py

'''

Generates a number of joke-of-the-day emails and stores them in a message

database using the maildb module.

The parameters for this module, such as the sender, recipients, number of jokes,

etc., are all stored in the settings module.

'''

from datetime import timedelta

from email.mime.text import MIMEText

from email.utils import make\_msgid

from joke\_service import HTTPJokeService

from maildb import store

from settings import (SENDER, RECIPIENTS, STARTTIME, DAYCOUNT,

JOKESOURCE, JOKEOPTIONS)

SENDER\_NAME = SENDER[0]

SENDER\_EMAIL = SENDER[1]

DATETIME\_FORMAT = '%a, %d %b %Y %H:%M:%S %z'

def \_get\_jokes():

'Retrieve a number of jokes from the joke service equal to DAYCOUNT'

js = HTTPJokeService(JOKESOURCE, JOKEOPTIONS)

return js.get\_jokes(DAYCOUNT)

def \_generate\_message(name, email, dt, payload):

'Create the message that will be stored in the database'

message = MIMEText(payload)

message['From'] = '{0} <{1}>'.format(SENDER\_NAME, SENDER\_EMAIL)

message['To'] = '{0} <{1}>'.format(name, email)

message['Message-ID'] = make\_msgid()

message['Date'] = dt.strftime(DATETIME\_FORMAT)

message['Subject'] = dt.strftime('JOTD: %Y-%m-%d')

store(message)

def generate\_messages():

'''Create one message for each day for each recipient. We are going to rely

on another program to actually send the emails, so store them in the message

table for now.'''

payloads = \_get\_jokes()

date = STARTTIME

for day in range(0, DAYCOUNT):

for name, email in RECIPIENTS:

\_generate\_message(name, email, date, payloads[day])

date += timedelta(days=1)

####maildb.py

'''

Email message handling module: contains logic to store email messages using a

MySQL relational database.

'''

from database import login\_info

import mysql.connector as msc

from email import message\_from\_string

from email.utils import parsedate\_tz, mktime\_tz, parseaddr

from datetime import datetime, timedelta

MESSAGE\_TABLE = 'message'

TBLDEF = '''

CREATE TABLE IF NOT EXISTS {0} (

msgID INTEGER AUTO\_INCREMENT PRIMARY KEY,

msgMessageID VARCHAR(128),

msgDate DATETIME,

msgSenderName VARCHAR(128),

msgSenderAddress VARCHAR(128),

msgText LONGTEXT

)'''

conn = msc.Connect(\*\*login\_info)

curs = conn.cursor()

curs.execute(TBLDEF.format(MESSAGE\_TABLE))

def store(msg):

'Stores an email message, if necessary, returning its primary key'

retrieval = '''

SELECT msgID FROM {0} WHERE msgMessageId = %s

'''.format(MESSAGE\_TABLE)

insertion = '''

INSERT INTO {0} (

msgMessageID, msgDate, msgSenderName, msgSenderAddress, msgText)

VALUES

(%s, %s, %s, %s, %s)

'''.format(MESSAGE\_TABLE)

message\_id = msg['message-id']

curs.execute(retrieval, (message\_id,))

result = curs.fetchone()

if result:

return result[0]

date = msg['date']

dt = datetime.fromtimestamp(mktime\_tz(parsedate\_tz(date)))

text = msg.as\_string()

name, email = parseaddr(msg['from'])

curs.execute(insertion, (message\_id, dt, name, email, text))

conn.commit()

curs.execute(retrieval, (message\_id,))

return curs.fetchone()[0]

def msg\_by\_id(id):

'''

Return the (presumably singleton) message whose primary key is given or

raise KeyError if no such message\_exists.

'''

query = '''

SELECT msgID, msgText FROM {0} WHERE msgID = %s

'''.format(MESSAGE\_TABLE)

curs.execute(query, (id,))

result = curs.fetchone()

if not result:

raise KeyError('ID {0} not found in store'.format(id))

id, text = result

msg = message\_from\_string(text)

return id, msg

def msg\_by\_message\_id(message\_id):

'''

Return the (presumably singleton) message whose "Message-ID" is given or

raise KeyError if no such message exists.

'''

query = '''

SELECT msgID, msgText FROM {0} WHERE msgMessageID = %s

'''.format(MESSAGE\_TABLE)

curs.execute(query, (message\_id,))

result = curs.fetchone()

if not result:

raise KeyError('ID {0} not found in store'.format(id))

id, text = result

msg = message\_from\_string(text)

return id, msg

def msgs(mindate=None, maxdate=None, namesearch=None, addsearch=None):

'''

Return a list of all messages sent on or after mindate and on or before

maxdate.

If mindate is not specified, there is no lower bound on the date, and

similarly if maxdate is not specified, no upper bound.

If namesearch is given, the result set is restricted to messages with sender

names containing that string.

If addsearch is given, the result set is restricted to messages with email

addresses containing that string.

'''

query = 'SELECT msgID, msgText FROM {0}'.format(MESSAGE\_TABLE)

clauses, data = [], []

if mindate:

clauses.append('msgDate >= %s')

data.append(mindate)

if maxdate:

clauses.append('msgDate <= %s')

data.append(maxdate + timedelta(days=1))

if namesearch:

clauses.append('msgSenderName LIKE %s')

data.append(namesearch)

if addsearch:

clauses.append('msgSenderAddress LIKE %s')

data.append(addsearch)

if clauses:

query += ' WHERE ' + ' AND '.join(clauses)

curs.execute(query, tuple(data))

return [(id, message\_from\_string(text)) for id, text in curs.fetchall()]

###settings.py

import datetime

SENDER = ('Seth', 'seth@gmail.com')

RECIPIENTS = [('Seth', 'seth@gmail.com'), ('John', 'seth@live.ca')]

STARTTIME = datetime.datetime.now()

DAYCOUNT = 7

# Options for the joke API. Set to retrieve Chuck Norris jokes, because it was

# the most useful API I could find. Retrieves only nerdy jokes for hilarity/NSFW

# reasons.

JOKESOURCE = 'http://api.icndb.com/jokes/random'

JOKEOPTIONS = {'limitTo': '[nerdy]'}

####test\_jotd.py

import unittest

import uuid

from datetime import timedelta

from mysql import connector

import jotd

import maildb

from settings import DAYCOUNT, RECIPIENTS, STARTTIME, SENDER

from database import login\_info

class JotdTest(unittest.TestCase):

def setUp(self):

maildb.MESSAGE\_TABLE = 'message\_{0}'.format(

str(uuid.uuid4()).replace('-', '')

)

self.connection = connector.Connect(\*\*login\_info)

self.cursor = self.connection.cursor()

self.cursor.execute(maildb.TBLDEF.format(maildb.MESSAGE\_TABLE))

jotd.generate\_messages()

def tearDown(self):

self.cursor.execute(

'DROP TABLE IF EXISTS {0}'.format(maildb.MESSAGE\_TABLE)

)

self.connection.commit()

self.cursor.close()

self.connection.close()

def test\_correct\_number\_of\_emails\_generated(self):

'''Ensure that the correct number of emails was generated. The number of

emails should be equal to the number of recipients multiplied by the

number of days.'''

message = ('{0} email(s) should have been generated, '

'but {1} email(s) were generated')

num\_recipients = len(RECIPIENTS)

num\_days = DAYCOUNT

num\_expected = num\_recipients \* num\_days

query = 'SELECT COUNT(\*) FROM {0}'.format(maildb.MESSAGE\_TABLE)

self.cursor.execute(query)

num\_emails = self.cursor.fetchone()[0]

self.assertEqual(num\_expected, num\_emails,

message.format(num\_expected, num\_emails))

def test\_correct\_email\_dates\_generated(self):

'''Ensure that the correct number of emails were generated for each

date, and that the correct dates were used. For each date there should

be a number of emails equal to the number of recipients.'''

message = ('{0} email(s) should have been generated for {1}, '

'but {2} email(s) where generated.')

start\_date = STARTTIME.date()

dates = [start\_date + timedelta(days=i) for i in range(0, DAYCOUNT)]

num\_recipients = len(RECIPIENTS)

query = '''

SELECT COUNT(\*) FROM {0} WHERE DATE(msgDate) = %s

'''.format(maildb.MESSAGE\_TABLE)

for date in dates:

self.cursor.execute(query, (date,))

num\_emails = self.cursor.fetchone()[0]

self.assertEqual(num\_recipients, num\_emails,

message.format(num\_recipients, date, num\_emails))

def test\_correct\_sender(self):

'''Ensure that the sender in the settings file was used when the emails

were generated.'''

message = ('The sender name "{0[0]} <{0[1]}>" should have been used, '

'but "{1[0]} <{1[1]}>" was found instead')

query = '''

SELECT DISTINCT msgSenderName, msgSenderAddress FROM {0}

'''.format(maildb.MESSAGE\_TABLE)

self.cursor.execute(query)

results = self.cursor.fetchall()

for found\_address in results:

self.assertEqual(SENDER, found\_address,

message.format(SENDER, found\_address))

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

====================

Congratulations on your performance in this class and on your strong finish. You rock!

-Pat

**Grade:**

Great

[Take Me Back](ostreturn:)

© 2014, O'Reilly Media, Inc. All rights reserved.

Bottom of Form