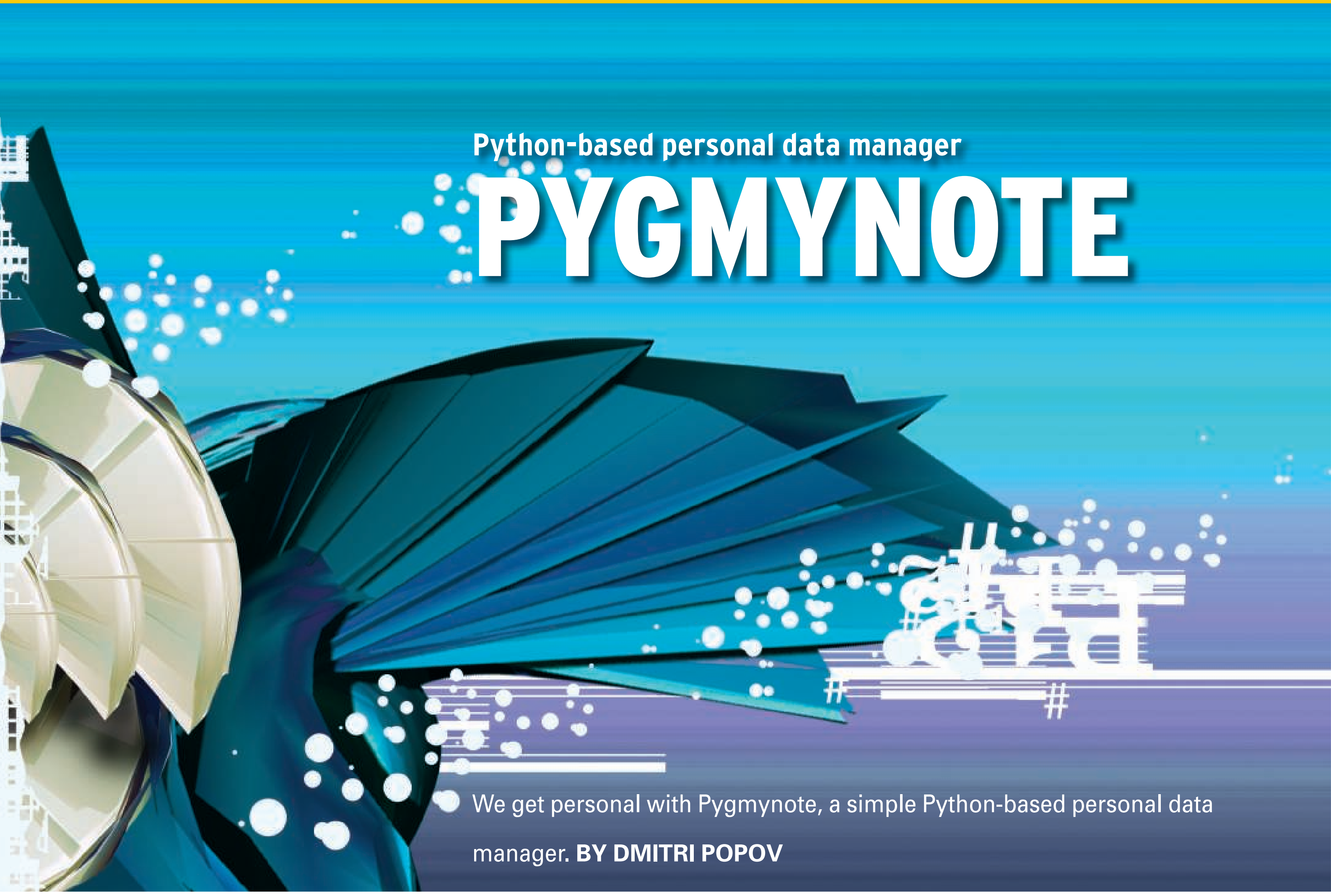


Python-based personal data manager

PYGMYNOTE

We get personal with Pygmynote, a simple Python-based personal data manager. **BY DMITRI POPOV**

Although personal information managers (PIMs) come in all shapes and colors, choosing the one that fits your needs is not as easy as it might seem. Despite trying dozens of otherwise excellent PIM applications, I still haven't found a tool that meets a few rather important requirements. It must be lightweight, so it can run equally fast on a desktop or a less-than-powerful Eee PC, and it must be easy to use with virtually no learning curve. It must be able to access data from anywhere – from a desktop machine or laptop as well as any machine via a web browser. Ideally, the application should let you share data stored in it with other users. Also, it should allow you to store virtually any kind of data: notes, calendar events, URLs, recipes, etc. Finally, the most obvious requirement is that you should be able to retrieve the data you need easily.

Listing 1: Find records with a Specific User ID

```
elif command == "my":
    cursor.execute ("SELECT * FROM notes WHERE tags LIKE '" + USERID + "'""ORDER BY id ASC")
    rows = cursor.fetchall ()
    for row in rows:
        print "\n   %s %s [%s]" % (row[0], row[1], row[2])
    print "\n   Number of records: %d" % cursor.rowcount
```

Failing to find my ideal PIM tool, I decided to write one myself using Python. The result is Pygmynote [1], a simple Python-based personal data manager (Figure 1).

The idea behind Pygmynote is rather simple. The application uses a MySQL database with a table that has only three fields: *id* (primary key that uniquely identifies each record in the database), *note*, and *tags*.

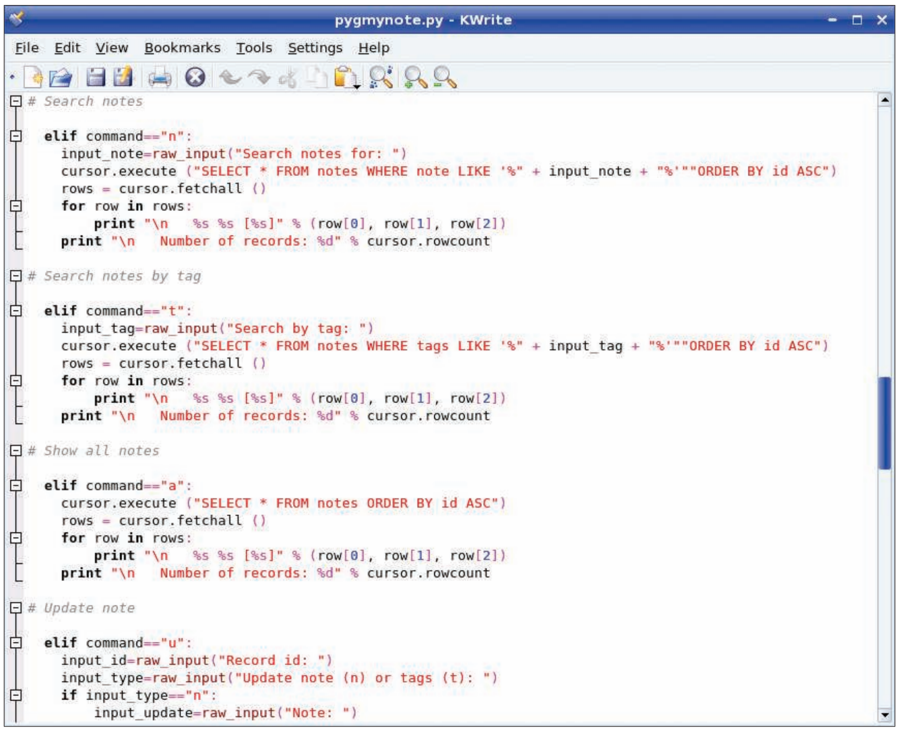
Using the available commands, you can enter the data you want in the *note* field, and then use the *tags* field to describe the data (Figure 2). For example, you can enter a recipe and tag it as "recipes dinner," or you can enter an event and specify its date in the *tags* field. Need to save an interesting link? No problem: Enter the link in the *note* field and tag it as *url*. In other words, you can store pretty much anything in Pygmynote – you just have to tag your data properly (Figure 3).

Piling different types of data up in one application might seem counter-productive – after all, many popular PIMs go to great lengths to help you structure and itemize your data. But this leads to an interesting paradox – imposing a certain structure on your data ultimately makes you less productive. For example, virtually any PIM has a contact module that offers a range of fields such as First Name, Last Name, ZIP, Street, City, Email, Phone, and so on.

But quite often, you don't really need this rigid structure. Why do you need to

Listing 2: Upload the pygmynote.txt file via Ftp

```
import ftplib

conn = ftplib.FTP('ftp.server','username','password')
f = open('pygmynote.txt','rb')
conn.storbinary('STOR pygmynote.txt', f)
f.close()
conn.quit()
```



```
# Search notes

  elif command=="n":
    input_note=raw_input("Search notes for: ")
    cursor.execute ("SELECT * FROM notes WHERE note LIKE '%" + input_note + "%'""ORDER BY id ASC")
    rows = cursor.fetchall ()
    for row in rows:
        print "\n   %s %s [%s]" % (row[0], row[1], row[2])
    print "\n   Number of records: %d" % cursor.rowcount

# Search notes by tag

  elif command=="t":
    input_tag=raw_input("Search by tag: ")
    cursor.execute ("SELECT * FROM notes WHERE tags LIKE '%" + input_tag + "%'""ORDER BY id ASC")
    rows = cursor.fetchall ()
    for row in rows:
        print "\n   %s %s [%s]" % (row[0], row[1], row[2])
    print "\n   Number of records: %d" % cursor.rowcount

# Show all notes

  elif command=="a":
    cursor.execute ("SELECT * FROM notes ORDER BY id ASC")
    rows = cursor.fetchall ()
    for row in rows:
        print "\n   %s %s [%s]" % (row[0], row[1], row[2])
    print "\n   Number of records: %d" % cursor.rowcount

# Update note

  elif command=="u":
    input_id=raw_input("Record id: ")
    input_type=raw_input("Update note (n) or tags (t): ")
    if input_type=="n":
        input_update=raw_input("Note: ")
```

Figure 1: Pygmynote is just a simple Python script that you can tweak to your heart's content.

put the street and zip code in separate fields?

At any rate, it would save you a lot of time and effort if you could just type all the contact info for a particular person in one field.

Also, the traditional approach makes the application less flexible because you can only use the Contacts module to store contact information. If you want to save notes, you need another module, and managing tasks and to-dos requires yet another component. And what if you also need to keep tabs on your recipes, tasks, and to-dos?

Before you know it, you need five or seven different modules or separate applications to keep track of all your data. Pygmynote attempts to solve these problems by allowing you to store any kind of data.

The trick is to describe each type of data properly, so you can quickly retrieve the records you want later.

Because Pygmynote is written in Python, you obviously need to have Python installed on your system. Most, if not all, Linux distributions come with Python pre-installed, so you only have to install a couple of packages to enable the MySQL functionality.

On Ubuntu, use *sudo apt-get install mysql-server python-mysqldb* to install the MySQL server and the Python for MySQL package.

To do the same on Mandriva, use the *urpmi mysql python-mysql* command. The MySQL server might refuse to start on Mandriva because of an unresolved bug. Fortunately, it's quite easy to fix that problem. Just execute the command

```
rpm -e mysql
rm -f /var/lib/mysql/mysql/*
/bin/hostname 127.0.0.1
urpmi mysql
```

and you can then start MySQL server via the Mandriva Control Center.

Next, you must configure Pygmynote so that it can connect to the MySQL database and your IMAP email account. First, open the *pygmynote.py* script in a text editor and enter the relevant information in the *MySQL connection settings* and *IMAP connection settings* section. After you make the script executable with the *chmod a+x python.py* command, Pygmynote is ready to go.

To launch the script, double-click on it or use the *python pygmynote.py* command in the terminal.

Then run the *create* command to create the *notes* table in the specified database.

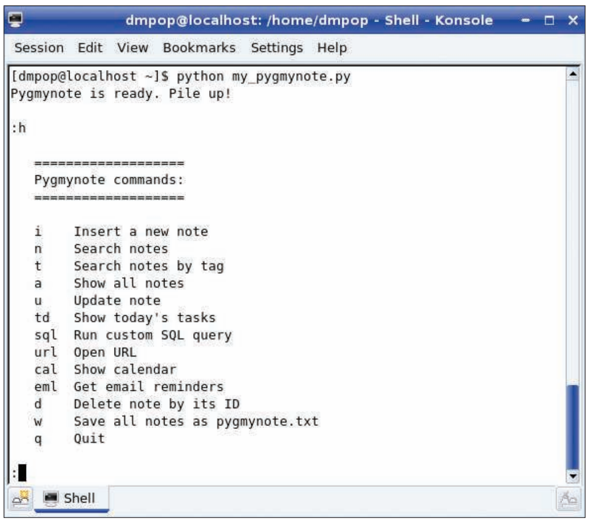
Commands

Pygmynote sports 13 easy-to-remember commands that allow you to find the data you need and perform actions with the records quickly. For example:

- *i*: Inserts a new note.
- *a*: Lists all records in the database.
- *td*: Finds all records containing the current date ("today") in the *tags* field, which quickly allows you to view the list of tasks and events scheduled for today.

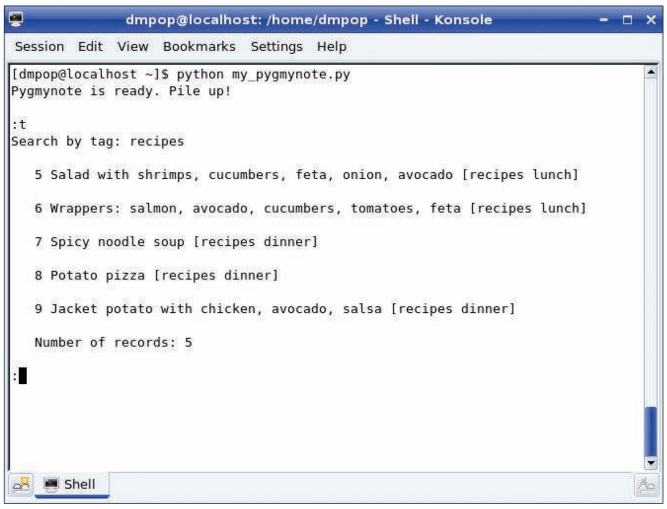
Pygmynote on Windows

What if you are on the move and you want to use Pygmynote on a Windows machine without Python? This problem is easy to solve by installing Portable Python [2] on a USB stick. To do this, download the latest version of Portable Python, unzip the downloaded archive, and move the resulting folder to the USB stick. To add the MySQL for Python module to the portable environment, copy the MySQL *db* folder inside the *Python25\Lib\site-packages* (on Windows) to the same directory in Portable Python. Also copy the following files: *_mysql.pyd*, *_mysql_exceptions.py*, *_mysql_exceptions.pyc*, and *_mysql_exceptions.pyo*. Now you can use Pygmynote on any Windows machine.


dmpop@localhost: /home/dmpop - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
[dmpop@localhost ~]$ python my_pygmynote.py
Pygmynote is ready. Pile up!

:h

   ===================
   Pygmynote commands:
   ===================

   i    Insert a new note
   n    Search notes
   t    Search notes by tag
   a    Show all notes
   u    Update note
   td   Show today's tasks
   sql  Run custom SQL query
   url  Open URL
   cal  Show calendar
   eml  Get email reminders
   d    Delete note by its ID
   w    Save all notes as pygmynote.txt
   q    Quit

:
```

Figure 2: Pygmynote contains just a few easy-to-remember commands.


dmpop@localhost: /home/dmpop - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
[dmpop@localhost ~]$ python my_pygmynote.py
Pygmynote is ready. Pile up!

:t
Search by tag: recipes

   5 Salad with shrimps, cucumbers, feta, onion, avocado [recipes lunch]

   6 Wrappers: salmon, avocado, cucumbers, tomatoes, feta [recipes lunch]

   7 Spicy noodle soup [recipes dinner]

   8 Potato pizza [recipes dinner]

   9 Jacket potato with chicken, avocado, salsa [recipes dinner]

   Number of records: 5

:
```

Figure 3: You can search records in Pygmynote by their tags.

- *u*: Updates the existing record.
- *d*: Deletes a record by specifying its ID.
- *url*: Handles records containing URLs in the *note* field. Just execute the command, enter the ID number of the desired record, and Pygmynote launches the URL in the default browser.
- *w*: Saves all the records in the tab-separated *pygmynote.txt* file. With this command you can back up your Pygmynote data or import it into any application that supports the tab-separated text format, such as OpenOffice.org Calc.
- *eml*: Retrieves email reminders. For example, if you don't have access to Pygmynote, also you can send an email to yourself containing a specific keyword, such as “Pygmynote” or “Reminder,” in the subject line. (You can specify the desired keyword in the *IMAP connection settings* section of the *pygmynote.py* script.) You can send yourself email reminders, like “Pygmynote: Remember to buy milk,” “Pygmynote: Doctor's appointment,” or “Reminder: Pay bills.” You can then use the *eml* command to view the messages containing the keyword.
- *h*: Shows you a list of all Pygmynote commands.

Tweaking Pygmynote

Because Pygmynote is just a simple Python script, you can tweak it to fit your particular needs or add new functionality. For example, say you plan to use Pygmynote in a multi-user environment and you would like to give users the ability to view their own tasks quickly. Simply add the *USERID* = '' option to the *MySQL connection settings* section in the script. Then copy an existing *elif* block and modify it as shown Listing 1.

The *my* command can be whatever string you like. The next time you add a record, you can add your user ID to the *tags* field (e.g., “dp”), and view all the records with your user ID by executing the *my* command.

The great thing about Python is that it comes with a lot of modules that allow you to add nifty features to Pygmynote without too much code wizardry. For example, the *w* command lets you save data stored in Pygmynote as a text file, but what if you want to upload the file to a remote server for off-site backup? The *ftplib* module allows you to do so with just a few lines of code (Listing 2).

The ability to generate an RSS feed containing shared records can be another useful feature, especially in a multi-user environment. You have several ways to generate an RSS feed in Python. The code in Listing 3 does this using the *xml.etree.cElementTree* module. The main advantage of this solution is that the *xml.etree.cElementTree* module is part of Python 2.5, so you don't need to install any additional software. As you might have figured out, the code publishes only records containing “rss” in the *tags* field by using the *SELECT * FROM notes WHERE tags LIKE '%rss%'* SQL statement. This way, you can specify which records you want to share by simply adding “rss” to the *tags* field.

Final Word

Obviously, Pygmynote is not the most sophisticated tool out there, but it provides an alternative approach to managing personal data. Because it's written in Python and sports a rather simple structure, you easily can adapt and extend the script to meet your needs. ■

Listing 3: RSS Feeds from Pygmynote records

```
import xml.etree.cElementTree as ET

cursor = conn.cursor ()
cursor.execute ("SELECT * FROM notes WHERE tags LIKE '%rss%'")
rows = cursor.fetchall ()

RSSroot = ET.Element( 'rss', {'version':'2.0'} )
RSSchannel = ET.SubElement( RSSroot, 'channel' )
ET.SubElement( RSSchannel, 'title' ).text = 'Pygmynote Feed'
ET.SubElement( RSSchannel, 'link' ).text = 'http://localhost/rss.xml'
ET.SubElement( RSSchannel, 'description' ).text = 'The feed generated by
Pygmynote'
for row in rows:
        RSSitem = ET.SubElement ( RSSchannel, 'item' )
        ET.SubElement( RSSitem, 'title' ).text = row[1]
        ET.SubElement( RSSitem, 'description' ).text = row[2]
        RSSfeed = ET.ElementTree(RSSroot)
        RSSfeed.write("rss.xml")
```

INFO

[1] Pygmynote: *pygmynote.googlecode.com*

[2] Portable Python: *http://www.portablepython.com/*

THE AUTHOR

Dmitri Popov holds a degree in Russian language and computer linguistics. He has been writing exclusively about Linux and open source software for several years, and his articles have appeared in Danish, British, North American, German, and Russian magazines and websites.