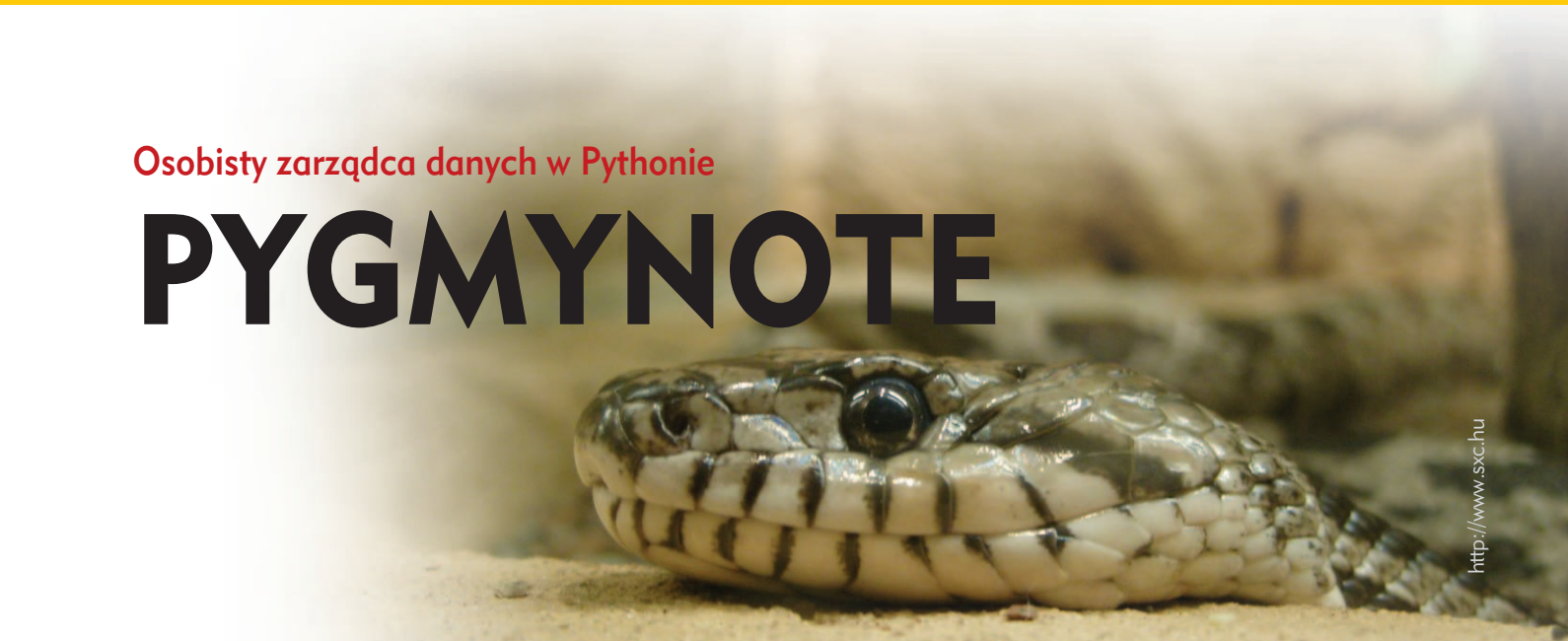
Osobisty zarządca danych w Pythonie

PYGMYNOTE

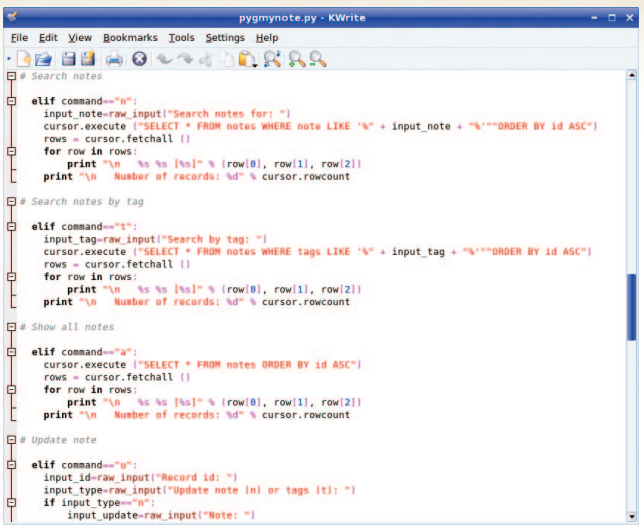


http://www.sxc.hu

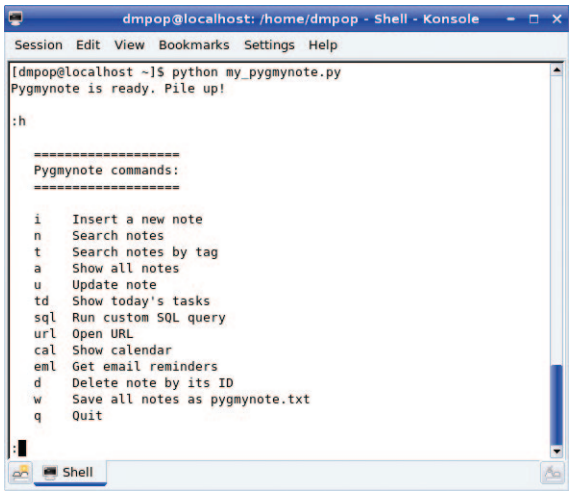
Poznajemy osobiście Pygmynote, prostego zarządcę danych w Pythonie.

DMITRI POPOV

Chociaż na rynku mamy do wyboru, do koloru osobistych zarządców danych (PIM), decyzja o tym jedynym, który zaspokoi nasze wymagania, nie jest taka łatwa, jak się wydaje. Mimo iż wypróbowałem tuziny świetnych skądinąd aplikacji PIM, wciąż nie znalazłem narzędzia, które spełniałoby kilka ważnych dla mnie warunków. Jego cechę musi bowiem stanowić lekkość, by mogło równie szybko działać na komputerze biurkowym, co na znacznie mniej wydajnym Eee PC; powinno być też łatwe w użyciu i nie wymagać wiele nauki. Musi pobierać dane skądkolwiek – od komputera biurkowego lub laptopa po jakąkolwiek maszynę za pomocą przeglądarki sieciowej. Najlepsza byłaby aplikacja pozwalająca na dzielenie się magazynowanymi w niej informacjami z innymi użytkownikami. Z pewnością powinna umożliwiać przechowywanie wszelkiego rodzaju danych: notatek, kalendarzy, wydarzeń, URL-i, przepisów itp. I na koniec najbardziej oczywiste z wymagań – nie wyobrażam sobie, żeby nie pozwalała z łatwością odszukiwać potrzebnych danych. Ponieważ nie udało mi się znaleźć idealnego narzędzia PIM, postanowiłem sam je stworzyć, używając Pythona. Efektem tego jest Pygmynote [1], prosty, oparty na Pythonie osobisty zarządca danych (Rysunek 1).



Rysunek 1: Pygmynote to prosty skrypt w Pythonie, który możemy dowolnie modyfikować.



Rysunek 2: Pygmynote zawiera tylko kilka łatwych do zapamiętania poleceń.

Pomysł, na którym opiera się Pygmynote, jest dość prosty. Program używa bazy danych MySQL z tabelą, która zawiera tylko trzy pola: *id* (klucz główny, który niepowtarzalnie identyfikuje każdy wpis w bazie danych), *note* i *tags*. Możemy użyć dostępnych poleceń, by umieścić wybrane dane w polu *note*, a następnie opisać je za pomocą użyć pola *tags* (Rysunek 2). Na przykład możemy wprowadzić przepis i opatrzyć go znacznikiem „przepisy na obiady” lub zapisać wydarzenie i podać jego datę w polu *tags*. Po-

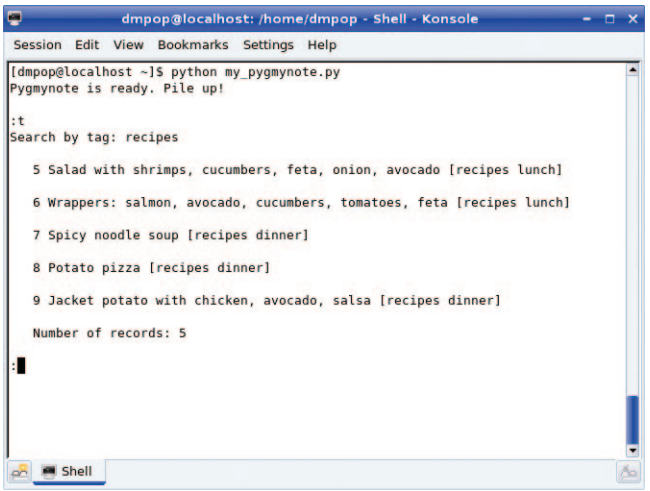
trzebujemy zapamiętać ciekawy link? Nie ma sprawy – wprowadźmy go w polu *note* i nadajmy mu znacznik *url*. Innymi słowy w Pygmynote możemy przechowywać dosłownie wszystko – musimy tylko opatrywać swoje dane odpowiednimi znacznikami (Rysunek 3).

Piętrzenie różnego typu danych w jednym programie może wydawać się nieefektywne – w końcu wiele popularnych PIM-ów polega w znacznym stopniu na umożliwianiu strukturyzowania i wyszczególniania danych. Ale prowadzi to do ciekawego paradoksu – narzucanie danym określonej struktury w końcu powoduje spadek produktywności. Na przykład właściwie każdy PIM jest wyposażony w moduł kontaktowy, który posiada szereg pól, takich jak: imię, nazwisko, kod pocztowy, ulica, miasto, adres e-mail, numer telefonu itp. Często jednak taka sztywna struktura nie jest potrzebna. Po co umieszczać nazwę ulicy i kod pocztowy w osobnych polach?

W każdym razie oszczędzilibyśmy wiele czasu i wysiłku, gdybyśmy mogli po prostu wpisać wszystkie dane kontaktowe jednej osoby w jednym polu. Tradycyjne podejście sprawia również, że program jest mniej elastyczny, ponieważ możemy przechowywać dane kontaktowe jedynie w module kontaktowym. Aby zapisać notatki, potrzebujemy kolejnego modułu, a zadania do wykonania wymagają jeszcze jednego komponentu. A co się stanie, jeśli zechcemy opatrzyć znacznikami swoje przepisy, wydarzenia i zadania? Zanim się zorientujemy, okaże się, że potrzebujemy pięciu albo siedmiu różnych modułów lub oddzielnych aplikacji, by prowadzić rejestr wszystkich swoich danych. Pygmynote próbuje uniknąć tych problemów, pozwalając na przechowywanie wszelkiego rodzaju danych. Sztuka polega na tym, by poprawnie opisać każdy rodzaj danych, co później pozwoli szybko odszukać dane wpisy.

Listing 1: Znajdź wpisy z ID danego użytkownika

```
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
```



Rysunek 3: Możemy wyszukiwać wpisy w Pygmynote po ich znacznikach.

Instalacja i konfiguracja

Ponieważ Pygmynote napisany jest w Pythonie, oczywiście w swoim systemie musimy mieć zainstalowanego Pythona. Większość (jeśli nie wszystkie) dystrybucji Linuksa posiada fabrycznie zainstalowanego Pythona, więc potrzebujemy tylko kilku pakietów, by umożliwić działanie MySQL. W Ubuntu za pomocą *sudo apt-get install mysql-server python-mysqldb* instalujemy serwer MySQL i pakiet Pythona dla MySQL. To samo w Mandrivie wykonujemy poleceniem *urpmi mysql python-mysql*. Serwer MySQL może odmówić działania pod Mandrivą z powodu nieusuniętego błędu. Na szczęście możemy rozwiązać ten problem w dość prosty sposób. Wystarczy, że wywołamy polecenie:

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

a następnie uruchomimy serwer MySQL przez Mandriva Control Center.

Następnie konfigurujemy Pygmynote tak, by mógł łączyć się z bazą danych MySQL i naszym kontem e-mail IMAP. W tym celu najpierw otwieramy skrypt *pygmynote. py* w edytorze tekstu i wprowadzamy stosowną informację w sekcjach *MySQL connection settings* i *IMAP connection settings*. Następnie za pomocą polecenia *chmod a+x python.py* doprowadzamy skrypt do stanu wykonywalności i Pygmynote jest gotowy.

By uruchomić skrypt, kliknijmy na nim dwukrotnie lub wpiszmy polecenie *python pygmynote. py* w terminalu. Następnie użyjmy polecenia *create* do stworzenia tabeli *notes* w wyszczególnionej bazie danych.

Polecenia

Pygmynote zawiera trzynaście łatwych do zapamiętania poleceń, które umożliwiają odnalezienie potrzebnych danych i szybkie przeprowadzanie pewnych operacji na wpisach. Na przykład:

- *i*: wstawia nową informację.

Pygmynote pod Windows

Co zrobić, jeśli jesteśmy w ruchu i chcemy używać Pygmynote na maszynie z Windowsami bez Pythona? Ten problem łatwo rozwiązujemy, instalując Portable Python [2] na nośniku USB. W tym celu pobierzmy najnowszą wersję Portable Python, rozpakujmy archiwum i przenieśmy powstały folder na nośnik USB. By dodać moduł MySQL dla Pythona do przenośnego środowiska, skopiujmy folder MySQL *db* do *Python25 | Lib | site-packages* (w Windowsie) do tego samego katalogu w Portable Python. Skopiujmy również następujące pliki: *_mysql.pyd*, *_mysql_exceptions.py*, *_mysql_exceptions. pyc, and _mysql_exceptions.pyo*. Teraz możemy używać Pygmynote na dowolnej maszynie z zainstalowanym Windowsem.

- *a*: wyświetla wszystkie wpisy w bazie danych.
- *td*: „today” znajduje wszystkie wpisy zawierające aktualną datę w polu *tags*, co pozwala szybko przeglądać listę zadań i wydarzeń zaplanowanych na dany dzień.
- *u*: pozwala zaktualizować istniejące wpisy.
- *d*: umożliwia usunięcie rekordu poprzez podanie jego ID.
- *url*: przydaje się, kiedy mamy do czynienia z wpisami zawierającymi URL-e w polu *note*. Po prostu uruchamiamy polecenie i wpisujemy numer ID danego wpisu, a Pygmynote uruchamia ten URL w domyślnej przeglądarce.
- *w*: to sprytne polecenie pozwala zapisać wszystkie wpisy w pliku z wartościami oddzielonymi tabulatorem *pygmynote.txt*. Możemy go używać, by tworzyć kopie zapasowe danych Pygmynote lub importować je do każdej aplikacji, która obsługuje format tekstu z wartościami oddzielonymi tabulatorem, takiej jak OpenOffice. org Calc.
- *eml*: umożliwia odbieranie przypomnień e-mail. Na przykład kiedy nie mamy dostępu do Pygmynote, możemy wysłać do siebie e-mail zawierający konkretne słowo kluczowe, takie jak „Pygmynote” lub „Przypomnienie”, w polu tytułu (możemy podać żądane słowo kluczowe w sekcji *IMAP connection settings* skryptu *pygmynote.py*). Możemy wysyłać do siebie e-maile z przypomnieniami, na przykład: „Pygmynote: Pamiętaj, by kupić mleko”, „Pygmynote: Wizyta u lekarza” lub „Przypomnienie: Opłacić rachunki”.
- *eml*: za pomocą tego polecenia w Pygmynote przeglądamy wiadomości zawierające konkretne słowo kluczowe.
- *h*: dzięki niemu możemy w każdej chwili przeglądać listę wszystkich poleceń.

Listing 2: Zapisywanie pliku pygmynote.txt przez 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 ()
```

Udoskonalanie Pygmynote

Ponieważ Pygmynote jest tylko prostym programem w Pythonie, możemy łatwo podrasować go, by spełniał nasze specyficzne potrzeby, lub dodawać nowe funkcjonalności. Załóżmy, że planujemy używać Pygmynote w środowisku wielu użytkowników i chcielibyśmy dać użytkownikom możliwość szybkiego przeglądania własnych zadań. Po prostu dodajmy opcję *USERID=””* do sekcji *MySQL connection settings* w programie. Następnie skopiujmy istniejący blok *elif* i zmodyfikujmy go tak, jak pokazuje Listing 1.

Możemy zastąpić polecenie *my* jakimkolwiek ciągiem tekstowym. Kiedy następnym razem dany użytkownik będzie dodawał wpis, może dodać swoje ID użytkownika w polu *tags* (na przykład „dp”) i przeglądać wszystkie wpisy z tym ID użytkownika, używając polecenia *my*.

Ogromną zaletą Pythona jest fakt, że posiada on wiele modułów pozwalających dodawać sprytne funkcje do Pygmynote bez wielkiego hokus-pokus z kodem. Na przykład polecenie *w* pozwala zapisać dane przechowywane w Pygmynote jako plik tekstowy. Ale co zrobić, jeśli chcemy zapisać plik na zdalnym serwerze, by stworzyć zdalną kopię zapasową? Moduł *ftplib* pozwala zrobić to za pomocą zaledwie kilku linijek kodu (Listing 2).

Kolejną przydatną, zwłaszcza w środowisku wielu użytkowników, cechą jest zdolność do tworzenia kanału RSS zawierającego współużytkowane wpisy. W Pythonie istnieje kilka sposobów tworzenia kanału RSS. Kod w Listingu 3 wykonuje to za pomocą modułu *xml.etree.cElementTree*. Główną zaletą tego rozwiązania jest fakt, iż moduł *xml.etree.cElementTree* stanowi część biblioteki standardowej Pythona 2.5, więc nie ma potrzeby instalowania dodatkowego oprogramowania.

Jak się zapewne domyślamy, kod wyświetla jedynie wpisy zawierające „rss” w polu *tags*, używając wyrażenia SQL *SELECT * FROM notes WHERE tags LIKE '%rss%'*. Dzięki temu możemy określić, które wpisy chcemy współdzielić, po prostu dodając „rss” w polu *tags*.

Listing 3: Kanały RSS z wpisów Pygmynote

```
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")
```

Podsumowanie

Oczywiście Pygmynote nie jest najbardziej wyrafinowanym narzędziem, ale proponuje alternatywne podejście do zarządzania osobistymi danymi. Ponieważ został napisany w Pythonie, a jego struktura jest relatywnie prosta, możemy go łatwo adaptować i rozszerzać tak, by spełniał nasze potrzeby. ■

INFO

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

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