Język programowania Python w zastosowaniach dydaktycznych i praktycznych

Jan Jełowicki
Uniwersytet Przyrodniczy we Wrocławiu
Katedra Matematyki

XXXIX Seminarium Zastosowań Matematyki
Kobyla Góra, 20–23 września 2009

Historia — koncepcje i języki

- algorytmy (****)
- maszyny liczące (15**)
- koncepcja Turinga (193*)
- koncepcja von Neumanna (1940)
- język kompilowany linearny: Fortran (1954)
- język funkcyjny: Lisp (1958)
- interpreter: Basic (1964)
- języki proceduralne: Algol/Pascal/C (196*)
- języki wektorowe: Matlab (Fortran/C) (1985)
- języki obiektowe: C++/Delphi (199*)
- lekkie języki skryptowe: Perl/PHP/Python/Lua (199*)
- języki obiektowe dla maszyn wirtualnych: Java/C# (199*)
- lekkie języki skryptowe dla maszyn wirtualnych: Scala (200*)

Historia — paradygmaty

- linearny (asm):
  kolejność, zmienna, tablica, skok
- funkcyjny (od Lispa):
  funkcja, lista, stos, wynik
- proceduralny (od Algola):
  typ, zakres, podprogram
- obiektowy (od C++):
  obiekt, metoda, klasa

- język kompilowany:
  write once, translate once, run many (on the same system)
- język interpretowany:
  write once, translate each time you run
- maszyna wirtualna:
  write once, translate once, run many (on any system)

Języki — porównanie

Lisp

```
; suma dodatnich wyrazów
(defun sumPl (w)
  (setq s 0.0)
  (loop for x in w do
    (if (> x 0.0)
      (setq s (+ s x))
    )
  ) s
)

; test
(print (sumPl
  (list 1.0 2.0 -3.5 4.0))
)
```

Basic

```
rem suma dodatnich wyrazów
function sumPl(w as array) _
    as double
  s = 0.0
  for i = lbound(w) to _
          ubound(w)
    if w(i) > 0.0 then
      s = s + w(i)
    end if
  next i
  sumPl = s
end function

' test
sub test()
  x = array(1.0, 2.0, _
            -3.5, 4.0)
  print sumPl(x)
end sub
```

Języki — porównanie

C

```c
#include <stdio.h>
/*
    suma dodatnich wyrazów
*/
double sumPl(int n,
    double *w) {
  double s = 0.0;
  int i;
  for (i=0; i<n; i++) {
    if (w[i] > 0.0) {
      s += w[i];
    }
  }
  return s;
}
/* test */
int main() {
  double x[4] =
    {1.0, 2.0, -3.5, 4.0};
  printf("%f\n",
    sumPl(4, x));        }
```

Python

```python
def sumPl(w):
  'suma dodatnich wyrazów'
  s = 0.0
  for x in w:
    if x > 0.0:
      s += x
  return s

# test
x = [1.0, 2.0, \
     -3.5, 4.0]
print sumPl(x)
```

Kilka prowokacyjnych pytań

- Czy algorytmy się starzeją?
- Czy języki programowania się starzeją?
- Jak starzeją się konkretne języki?

Kilka prowokacyjnych pytań

- Czy algorytmy się starzeją?
  - Tak — w miarę postępu wiedzy powstają lepsze
  - Tak — w miarę postępu techniki niektóre założenia tracą aktualność
  - Nie — jeżeli coś działało, dalej będzie działać
- Czy języki programowania się starzeją?

- Jak starzeją się konkretne języki?

Kilka prowokacyjnych pytań

- Czy algorytmy się starzeją?
  - Tak — w miarę postępu wiedzy powstają lepsze
  - Tak — w miarę postępu techniki niektóre założenia tracą aktualność
  - Nie — jeżeli coś działało, dalej będzie działać
- Czy języki programowania się starzeją?
  - Tak — zmieniają się warunki i wymagania
- Jak starzeją się konkretne języki?

Kilka prowokacyjnych pytań

- Czy algorytmy się starzeją?
  - Tak — w miarę postępu wiedzy powstają lepsze
  - Tak — w miarę postępu techniki niektóre założenia tracą aktualność
  - Nie — jeżeli coś działało, dalej będzie działać
- Czy języki programowania się starzeją?
  - Tak — zmieniają się warunki i wymagania
- Jak starzeją się konkretne języki?
  - np. Lisp
  - np. Cobol
  - np. Basic
  - np. Algol
  - np. C
  - np. Matlab

Kilka prowokacyjnych pytań

- Czy inżynier powinien umieć programować?
- Czy student kierunków inżynieryjnych powinien umieć programować?
- Jaki zakres tej umiejętności jest przydatny?

Kilka prowokacyjnych pytań

- Czy inżynier powinien umieć programować?
  - Tak: dla zachowania niezależności od posiadanych środków technicznych
  - Nie: „wszystkie typowe zagadnienia już są oprogramowane”
  - Tak: aplikacje inżynieryjne mają interfejsy programistyczne
  - Nie: i tak nie dogoni gotowych aplikacji
- Czy student kierunków inżynieryjnych powinien umieć programować?
- Jaki zakres tej umiejętności jest przydatny?

Kilka prowokacyjnych pytań

- Czy inżynier powinien umieć programować?
  - Tak: dla zachowania niezależności od posiadanych środków technicznych
  - Nie: „wszystkie typowe zagadnienia już są oprogramowane”
  - Tak: aplikacje inżynieryjne mają interfejsy programistyczne
  - Nie: i tak nie dogoni gotowych aplikacji
- Czy student kierunków inżynieryjnych powinien umieć programować?
  - Tak: by kreatywnie myślał i wykorzystywał zasoby
  - Nie: bo „i tak mu się to nie przyda”
  - Tak: by przełamać to przekonanie
- Jaki zakres tej umiejętności jest przydatny?

Kilka prowokacyjnych pytań

- Czy inżynier powinien umieć programować?
  - Tak: dla zachowania niezależności od posiadanych środków technicznych
  - Nie: „wszystkie typowe zagadnienia już są oprogramowane”
  - Tak: aplikacje inżynieryjne mają interfejsy programistyczne
  - Nie: i tak nie dogoni gotowych aplikacji
- Czy student kierunków inżynieryjnych powinien umieć programować?
  - Tak: by kreatywnie myślał i wykorzystywał zasoby
  - Nie: bo „i tak mu się to nie przyda”
  - Tak: by przełamać to przekonanie
- Jaki zakres tej umiejętności jest przydatny?
  - pisanie skryptów do konwersji danych
  - niestandardowe wizualizacje danych
  - projektowanie wielokrotnie powtarzalnych obliczeń
  - skrypty/makra dla aplikacji
  - ...

Kilka prowokacyjnych pytań

- Obraz/manipulacja czy tekst/opowiadanie?
- Uczyć samodzielności czy przyuczać do użytkowania gotowych systemów?

Kilka prowokacyjnych pytań

- Obraz/manipulacja czy tekst/opowiadanie?
  - zobaczyć i poczuć $\Rightarrow$ łatwiej zrozumieć
  - czy poprzestać na zaangażowaniu wizualno-kinestetycznym?
- Uczyć samodzielności czy przyuczać do użytkowania gotowych systemów?

Kilka prowokacyjnych pytań

- Obraz/manipulacja czy tekst/opowiadanie?
  - zobaczyć i poczuć $\Rightarrow$ łatwiej zrozumieć
  - czy poprzestać na zaangażowaniu wizualno-kinestetycznym?
  - dogłębnie zrozumieć $\equiv$ umieć opowiedzieć
  - zapisać $\Rightarrow$ móc wielokrotnie wykorzystać
- Uczyć samodzielności czy przyuczać do użytkowania gotowych systemów?

Kilka prowokacyjnych pytań

- Obraz/manipulacja czy tekst/opowiadanie?
  - zobaczyć i poczuć $\Rightarrow$ łatwiej zrozumieć
  - czy poprzestać na zaangażowaniu wizualno-kinestetycznym?
  - dogłębnie zrozumieć $\equiv$ umieć opowiedzieć
  - zapisać $\Rightarrow$ móc wielokrotnie wykorzystać
- Uczyć samodzielności czy przyuczać do użytkowania gotowych systemów?
  - niby „wszystko jest już napisane”…
  - … ale „sytuacje nietypowe” zdarzają się co chwilę
  - „wyuczona bezradność” w sytuacjach nietypowych?

Kontrowersyjna dygresja

Jan Jełowicki

Dlaczego Python?

Potrzeby	**Możliwości**	**Wybór**
skrypty systemowe	Bash, Powershell, Perl, ...	
obliczenia i wizualizacja danych	Matlab, Python, klasyczne języki kompilowane	
makropolecenia w środowiskach użytkowych	Basic, Python, Lisp, Lua, ...	
inne...	...	

(Python)

Python — historia

- autor: Guido van Rossum (obecnie Google)
- opieka: *Python Software Foundation*
- początek: 1991 r.
- aktualne wersje:
  - 2.5.4 (grudzień 2008 r.)
  - **2.6.2** (kwiecień 2009 r.)
  - 3.1.1 (sierpień 2009 r.)
- licencja: GPL

Założenia projektowe Pythona

Guido van Rossum
Computer Programming for Everybody (1999)
`http://www.python.org/doc/essays/cp4e.html`

Our plan has three components:

- *Develop a new computing curriculum suitable for high school and college students.*
- *Create better, easier to use tools for program development and analysis.*
- *Build a user community around all of the above, encouraging feedback and self-help.*

These components come together in the scientific exploration of the role of programming in next generation computing environments.

Założenia projektowe Pythona

Guido van Rossum
Computer Programming for Everybody (1999)
`http://www.python.org/doc/essays/cp4e.html`

Examples of this drive for flexibility can be seen in both present-day computing and its likely future:

- *Increasingly powerful applications for desktop and laptop computers use scripting and macro facilities.*
- *Growth of the Internet has led directly to greater need for programmability to create active and interactive Web content.*
- *End-user information appliances and networks of CPUs embedded in everyday objects — both will demand user control and personalization.*
- *Mobile and intelligent software agents will be commonplace and require customization by users.*

Założenia projektowe Pythona

`http://www.python.org/about/`

- *very clear, readable syntax*
- *strong introspection capabilities*
- *intuitive object orientation*
- *natural expression of procedural code*
- *full modularity, supporting hierarchical packages*
- *exception-based error handling*
- *very high level dynamic data types*
- *extensive standard libraries and third party modules for virtually every task*
- *extensions and modules easily written in C, C++ (or Java for Jython, or .NET languages for IronPython)*
- *embeddable within applications as a scripting interface*

Co wyróżnia Pythona?

- swoboda
  - dowolny hardware (od telefonu po mainframe)
  - dowolny system operacyjny
  - dowolny paradygmat
  - współpraca z innym oprogramowaniem
  - kilka interpreterów

Co wyróżnia Pythona?

- swoboda
  - dowolny hardware (od telefonu po mainframe)
  - dowolny system operacyjny
  - dowolny paradygmat
  - współpraca z innym oprogramowaniem
  - kilka interpreterów
- prostota
  - czytelna składnia
  - jednolitość
  - „bardzo wysoki” poziom
  - szybkość tworzenia kodu

Co wyróżnia Pythona?

- swoboda
  - dowolny hardware (od telefonu po mainframe)
  - dowolny system operacyjny
  - dowolny paradygmat
  - współpraca z innym oprogramowaniem
  - kilka interpreterów
- prostota
  - czytelna składnia
  - jednolitość
  - „bardzo wysoki” poziom
  - szybkość tworzenia kodu
- kompilacja do kodu pośredniego (*bytecode*); interpretacja w maszynie wirtualnej

Co wyróżnia Pythona?

- swoboda
  - dowolny hardware (od telefonu po mainframe)
  - dowolny system operacyjny
  - dowolny paradygmat
  - współpraca z innym oprogramowaniem
  - kilka interpreterów
- prostota
  - czytelna składnia
  - jednolitość
  - „bardzo wysoki” poziom
  - szybkość tworzenia kodu
- kompilacja do kodu pośredniego (*bytecode*); interpretacja w maszynie wirtualnej
- bogactwo bibliotek

Co wyróżnia Pythona?

- swoboda
  - dowolny hardware (od telefonu po mainframe)
  - dowolny system operacyjny
  - dowolny paradygmat
  - współpraca z innym oprogramowaniem
  - kilka interpreterów
- prostota
  - czytelna składnia
  - jednolitość
  - „bardzo wysoki” poziom
  - szybkość tworzenia kodu
- kompilacja do kodu pośredniego (*bytecode*); interpretacja w maszynie wirtualnej
- bogactwo bibliotek
- licencja

Cechy składni Pythona

- minimalizm
- elegancja
- konsekwencja
- obiektowość
- brak sztucznych udziwnień

Język — charakterystyka

- dana jest obiektem, zmienna jest referencją
- dana ma typ, zmienna nie ma typu
- układ typograficzny kodu, wcięcia i dwukropki
- funkcje i generatory
- przekazywanie argumentów przez wartość (ale argument jest referencją)
- leniwa (późna) ewaluacja
- import modułów, przestrzenie nazw
- typy, klasy, funkcje, moduły itp. też są obiektami

Język — słowa kluczowe


```
False None True and as assert
break class continue def del elif
else except exec finally for from
global if import in is lambda
nonlocal not or pass print
raise return try while with yield
```

Język — typy elementarne

- `NoneType`: wartość specjalna `None`
- `bool`: wartości `True`, `False`
  operacje logiczne
- `int` i `long`: liczby całkowite
  działania arytmetyczne: `+ - * / ** %`
  operacje bitowe: `| ^ &`
- `float`: liczby double
  działania arytmetyczne: `+ - * / // ** %`
- `complex`: liczby zespolone

Język — typy iterowalne

- `str, unicode`: napisy, ciągi znaków

```
'abcd' "abcd"  '''abcd''' """abcd"""
```

operacje: `+ *`; indeksowanie od przodu i od końca; wycinki `[1:3]`

- `list`: listy, ciągi dowolnych obiektów

```
[1,2,5] [1, 'a', [1]] a = x; a = a + [a] # rekursja
```

operacje: `+ *`, indeksy, wycinki, ...

- `tuple`: listy niemodyfikowalne, krotki

```
(1,2,5)
```

- `set`: zbiory

```
{1,2,5}
```

- `dict`: tablice asocjacyjne, „słowniki”

```
{'a': 1, 'b': 2, 'c': 5} {1: 'p', 2: 'q', 5: 'r'}
```

- `file`: pliki i strumienie

operacje: `open(), read(), write(), close()`, ...

Język — typy niestandardowe

- `Decimal`: liczby dziesiętne
- `array`, `matrix`: wektory i macierze
- `class`: mechanizm tworzenia typów (klas obiektów)

Język — typy niestandardowe

- `Decimal`: liczby dziesiętne
- `array`, `matrix`: wektory i macierze
- `class`: mechanizm tworzenia typów (klas obiektów)
  - każdy typ danych jest zarządzany przez klasę
  - każdy obiekt jest przedstawicielem pewnej klasy
  - zmienna jest referencją do obiektu
  - obiekty bez referencji są usuwane (*garbage collector*)

Jan Jełowicki

Język — obiekty i wartości

Warianty instrukcji przypisania

```
a = x # nadaj zmiennej a wartość x

a = b = x # nadaj zmiennym a i b wartość x

a, b, c = x, y, z # jednoczesne przypisanie

a, b = b, a # zamiana wartości zmiennych
```

Przyrównywanie obiektów

```
a == b # czy wartości są takie same

a is b # czy są referencjami do tego samego obiektu
```

Usuwanie obiektów

```
del a
```

Język — konstrukcje warunkowe

Jan Jełowicki

Instrukcja warunkowa

```
if warunek:
    instrukcje
elif warunek:
    instrukcje
else:
    instrukcje
```

Wyrażenie warunkowe

```
x = a if warunek else b
```

Język — instrukcje iteracji

Iterowanie listy

```
for x in a:
    instrukcje
else:
    instrukcje
```

Wyrażenie iteracyjne (generator listy)

```
b = [ wyrażenie for x in a ]
```

Iteracja sterowana warunkiem

```
while warunek:
    instrukcje
else:
    instrukcje
```

Pomocnicze

```
break, continue, else
```

Jan Jełowicki

Język — obsługa wyjątków

Klauzula kontroli wyjątków

```
try:
    instrukcje # wykonuj ,,normalnie''
except:
    instrukcje # wykonaj w razie ,,wpadki''
finally:
    instrukcje # wykonaj pod koniec
```

Pomocnicze

```
raise
```

Język — deklaracje

Funkcje i procedury

```
def f(arg):
    instrukcje
    return x # przekazanie wyniku, zakończenie pracy
```

Generatory

```
def f(arg):
    instrukcje
    yield x # przekazanie wyniku, zatrzymanie stanu
```

Klasy i metody

```
class c():
    ...
    def p(self, arg):
        instrukcje
```

Wczytywanie bibliotek i modułów

```
import xxx
from xxx import yyy
```

Biblioteki — matematyka

- `math` (standard)
- `numpy` (podstawowe metody numeryczne)
- `scipy` (zaawansowane obliczenia naukowe)
- `sympy` (podstawowe obliczenia symboliczne)
- `cvxopt` (optymalizacja wypukła)
- `RPy` (interfejs do pakietu statystycznego R)
- ...

Biblioteki — grafika

- `matplotlib/pylab` (grafika wektorowa)
- `Gnuplot` (interfejs do programu gnuplot)
- `VPython` (wizualizacja procesów fizycznych)

- `PIL` (operacje na bitmapach, analiza obrazu)
- `ImageMagick` (interfejs do pakietu ImageMagick)
- ...

Biblioteki — użytkowe środowisko graficzne

GUI

- `tkinter` (standard)
- `gtk`: GTK+
- `QT`: Trolltech QT
- `wxWidgets`
- Windows API — jedyne nieprzenośne

RAD

- QtDesigner (dla QT)
- Glade (dla GTK+ i wxWidgets)

Biblioteki — system

- `system`, `os` (standard)
- `urllib`, `sockets`: sieć (standard)
- kryptografia (standard)
- XML: DOM, SAX (standard)
- serwisy i aplikacje WWW: Django, Pylons, Cherry

Biblioteki — bazy danych

- SQLite (standard)
- PostgreSQL
- MySQL
- Firebird
- ODBC
- ...

Biblioteki — komunikacja z aplikacjami

Przykłady

- COM (Microsoft) — komunikacja z aplikacjami
- DBus (UNIX) — komunikacja z aplikacjami
- UNO (OpenOffice) — komunikacja z aplikacją, język makropoleceń
- Gnumeric — język makropoleceń
- Blender — język makropoleceń
- GIMP — język makropoleceń
- Inkscape — język makropoleceń
- ...

Interpretery i platformy wykonawcze

- praktycznie każdy współczesny OS (CPython)
- .NET (IronPython)
- Java (Jython)
- PyPy (eksperymentalny interpreter Pythona napisany w Pythonie)
- py2exe (tworzenie aplikacji Windows z kodu Pythona)

Środowisko

- lekkie edytory: SciTE, Notepad++, BlueFish, MC, ...
- silne uniwersalne edytory: Emacs, Vim, ...
- kompletne IDE z debuggerem: IDLE, PythonWin, Komodo, WingIDE, Eclipse, ...
- samodzielne debuggery: pgdb, WinPDb, ...

Python jako narzędzie dla inżyniera

- tworzenie skryptów uruchamianych z poziomu systemu operacyjnego
- konwersja danych
- tworzenie programów obliczeniowych
- wizualizacja niestandardowych danych
- integracja wielu niezależnych komponentów w jednym projekcie
- makropolecenia w niektórych aplikacjach

Python w dydaktyce

Wrocław (poza wydziałami informatycznymi)

- Wydział Fizyki Uniwersytetu Wrocławskiego (co najmniej od 2005 r.)
- Wydział Chemii Politechniki Wrocławskiej (co najmniej od 2005 r.)
- ...

Wydział IKŚiG UP we Wrocławiu
(J. Jełowicki, A. Machowczyk)

- testy i przymiarki (od 2005 r.)
- wykorzystanie podczas zajęć (od wiosny 2008 r.)
- materiały dydaktyczne dla studentów:

  `http://karnet.up.wroc.pl/~jasj/cwiczenia/kwpp.html`

Python w nauce

Jan Jełowicki

Wrocław

- Wrocławskie Centrum Sieciowo-Superkomputerowe (co najmniej od 2008 r.)
- ...

Wydział IKŚiG UP we Wrocławiu
(M. Grządziel, J. Jełowicki)

- wykorzystanie pakietu optymalizacji wypukłej CvxOpt (od wiosny 2009 r.)

Ciąg Fibonacciego, wersja naiwna

Jan Jełowicki


```
#!/usr/bin/env python

def fib(n):
    if (n<2):
        return 1
    else:
        return fib(n-1) + fib(n-2)

n = 200
x = []
# niestety, ta petla szybko ,,zatka sie''
# z uwagi na zlozonosc obliczen
for in in range(n):
    x.append[fib(i)]

print x
```

Ciąg Fibonacciego, wersja iteracyjna

Jan Jełowicki


```
#!/usr/bin/env python

n = 200
x = [1, 1]
for i in range(2,n):
    x.append(x[i-1] + x[i-2])

print x
```

Ciąg Fibonacciego — generator


```
#!/usr/bin/env python

def genfib():
    a, b = 1, 1
    while 1:
        yield a
        a, b = b, a+b

fib = genfib()
n = 200
x = [ fib.next() for i in range(n) ]

print x
```

Ciąg Fibonacciego, wersja funkcyjna

Jan Jełowicki


```
#!/usr/bin/env python

def listfib(n):
    return (n+1)*[1] if n<2 \
           else nextfib(listfib(n-1))

def nextfib(fibs):
    return fibs + [ sum(fibs[-2:]) ]

x = listfib(200)

print x
```

Obliczenia numeryczne: całkowanie


```
#!/usr/bin/env python
import math
from scipy import integrate as calka

def f(x):
    return math.exp(-x**2)

# metoda Romberga z biblioteki, funkcja zadeklarowana
a = 0.0
b = math.pi
q = calka.romberg(f, a, b)
print q
# metoda Romberga z biblioteki, funkcja anonimowa
a = 0.0
b = math.pi
q = calka.romberg(lambda(x): math.exp(-x**2), a, b)
print q
```

```
$ python romberg.py

0.886219059173
0.886219059173
```

Obliczenia numeryczne: algebra

Jan Jełowicki


```
#!/usr/bin/env python
import numpy

m = numpy.matrix([[1.0, 0.0], [2.0, 2.0]])

# wyznaczene macierzy odwrotnej
mm = m**-1

print m
print mm
print m * mm
```

```
$ python matrix.py

[[ 1.  0.]
 [ 2.  2.]]
[[ 1.   0. ]
 [-1.   0.5]]
[[ 1.  0.]
 [ 0.  1.]]
```

Obliczenia symboliczne: różniczkowanie

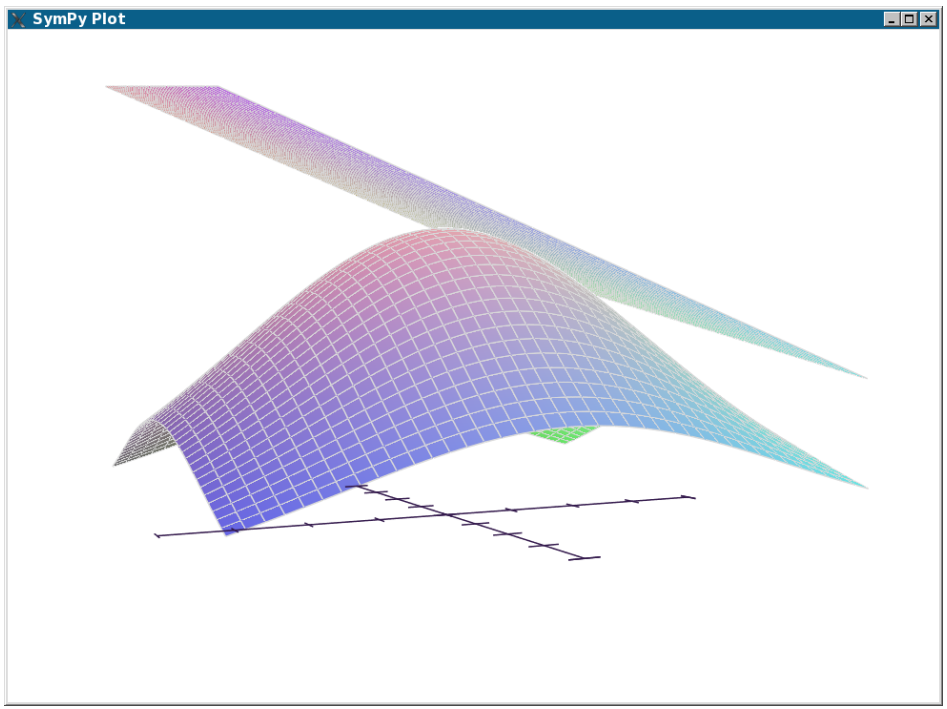
Jan Jełowicki

```
#!/usr/bin/env python
import sympy
x, y = sympy.symbols('xy')
a, b = 1.0, 1.0
z = sympy.exp(-a*x**2-b*y**2)
# pochodna funkcji 2 zmiennych
dz = [ sympy.diff(z, p) for p in [x,y] ]
x0, y0 = 0.1, 0.2
z0 = z.subs(x, x0).subs(y, y0)
dz0 = [dz[i].subs(x, x0).subs(y,y0) for i in range(2)]
# punkt na plaszczyznie stycznej w (x0, y0)
zprim = z0 + dz0[0]*(x-x0) + dz0[1]*(y-y0)

print z  # ,,wzor'' funkcji
print dz  # ,,wzor'' pochodnej
print z0, float(z0) # wartosc w punkcie (symb. i num.)
print dz0, map(float, dz0) # pochodna w punkcie
print zprim # ,,wzor'' rozniczki

sympy.Plot(z).append(zprim)     # grafika
```

Obliczenia symboliczne: różniczkowanie — wyniki


```
$ python symbolic.py
```

```
exp(-y**2 - 1.0*x**2)
[-2.0*x*exp(-y**2 - 1.0*x**2), -2*y*exp(-y**2 - 1.0*x**2)]
exp(-0.05) 0.951229424501
[-0.2*exp(-0.05), -0.4*exp(-0.05)] [-0.1902458849001428, -0.3804917698002856]
0.2*(0.1 - x)*exp(-0.05) + 0.4*(0.2 - y)*exp(-0.05) + exp(-0.05)
```



Konwersja danych: wejście

Dane opadowe dekadami

```
 0.4   1.0   0.6   0.0   0.0   0.0   0.0   0.0   0.0   4.2
 1.0   0.1   1.3   0.6   0.0   0.0   0.0   0.0   0.0   0.0
 0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.3   3.7
 0.0
 0.0   0.0   2.0   3.8   4.5   2.1   3.7  24.8   5.4   1.8
 5.5   3.1   0.4   0.8   0.4   1.5   0.4   1.6   4.4   2.1
 2.9   0.2   0.0   0.4   0.3   0.1   0.0   0.0
 0.0   0.2   0.0   0.0   1.5   0.0   0.0   0.0   0.0   0.0
 0.0   0.0   0.0   4.0   0.0   0.0   0.0   0.0   0.0   0.0
 0.1   1.1   0.0   0.1   6.2   0.0   0.0   0.0   0.0   0.0
 0.0
 0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.1   0.0
 0.0   5.6   0.0   0.0   0.0   0.0   0.0   8.6   6.1   0.0
 0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.4   0.0   0.0
 0.0   0.0   0.0   0.0   0.0   6.5   0.0   2.7   0.0   0.0
 0.0   0.0   5.6   3.2   0.0   3.1   0.0   2.7   0.0   0.0
 0.0   0.0   3.4   0.0   0.7   4.6   9.0   0.0   0.0   0.0
 1.3
 0.0   1.3   0.0   0.0   0.0   0.0   3.9   0.8   1.2   9.9
19.2   0.7  12.6   1.2   0.0   0.0   0.0   0.5   0.0   0.0
22.7  10.7   1.7  18.9   5.1   0.3   1.8   0.0   0.0   0.0
 0.0   0.0   0.0   0.0   0.0   5.5   2.3   0.8   0.9   0.0
 3.2   0.5  28.4   0.1  17.1   0.0   0.0  13.2   5.3   0.0
 2.0   0.0   0.0   0.0   0.0   0.0   0.0   1.0   0.0   0.0
 1.7
 0.0   2.4   3.7   0.0   0.0   6.1   0.0   7.8   1.3   0.0
26.5   0.0  10.4   0.0   0.1   0.0   0.0   0.2  19.8   0.0
 7.4   0.0   6.4  12.0   8.0   1.2   0.0   2.4   0.0   8.5
 8.1
 0.0   0.0   0.0   0.0   0.0  11.2   0.6   2.0   0.0   0.0
 0.0   0.0   0.0   2.2   0.0   0.0   0.0   3.0   1.0   0.0
 0.4   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
 0.0   0.0   0.0   6.0  12.2  10.2  10.7  10.9   0.3   0.0
```

Konwersja danych: wyniki

Daty i dane opadowe w tabeli

```
1946 sty 1 0.4
1946 sty 2 1.0
1946 sty 3 0.6
1946 sty 4 0.0
1946 sty 5 0.0
1946 sty 6 0.0
1946 sty 7 0.0
1946 sty 8 0.0
1946 sty 9 0.0
1946 sty 10 4.2
1946 sty 11 1.0
1946 sty 12 0.1
1946 sty 13 1.3
1946 sty 14 0.6
1946 sty 15 0.0
1946 sty 16 0.0
1946 sty 17 0.0
1946 sty 18 0.0
1946 sty 19 0.0
1946 sty 20 0.0
1946 sty 21 0.0
1946 sty 22 0.0
1946 sty 23 0.0
1946 sty 24 0.0
1946 sty 25 0.0
1946 sty 26 0.0
1946 sty 27 0.0
1946 sty 28 0.0
1946 sty 29 0.3
1946 sty 30 3.7
1946 sty 31 0.0
1946 lut 1 0.0
1946 lut 2 0.0
```

Konwersja danych: kod


```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
""" przeformatowanie zbioru danych """
import os
import easygui
dekady = (4, 3, 4, 3, 4, 3, 4, 4, 3, 4, 3, 4)
miesiace = ('sty', 'lut', 'mar', 'kwi', 'maj', 'cze',
            'lip', 'sie', 'wrz', 'paz', 'lis', 'gru')

def skanuj(katalog, nazwa):
    plik = open(katalog +  nazwa)
    d = 0
    m = 0
    i = 0
    for dekada in plik.readlines():
        dekada = dekada.split()
        for dana in dekada:
            i += 1
            wyj.write('%s\t%s\t%i\t%s\n' % \
                (nazwa[:-4], miesiace[m], i, dana))
        d += 1
        if d == dekady[m]:
            d = 0
            m += 1
            i = 0
```

Obliczenia hydrauliczne: wejście

Jan Jełowicki

Obrys przekroju hydrometrycznego

```
# Bystrzyca 64.300
# D Z
86.67 219.872
92.25 217.284
95.66 216.566
95.89 216.207
96.94 216.149
97.81 216.104
98.84 216.142
100.04 216.142
101.15 216.124
102.32 216.091
103.32 216.119
104.45 216.117
105.75 216.058
106.66 216.017
107.66 216.069
108.68 216.029
109.51 216.138
110.52 216.207
110.76 216.625
110.76 216.624
120.92 216.831
121.29 218.008
126.55 218.884
```

Obliczenia hydrauliczne: wyniki

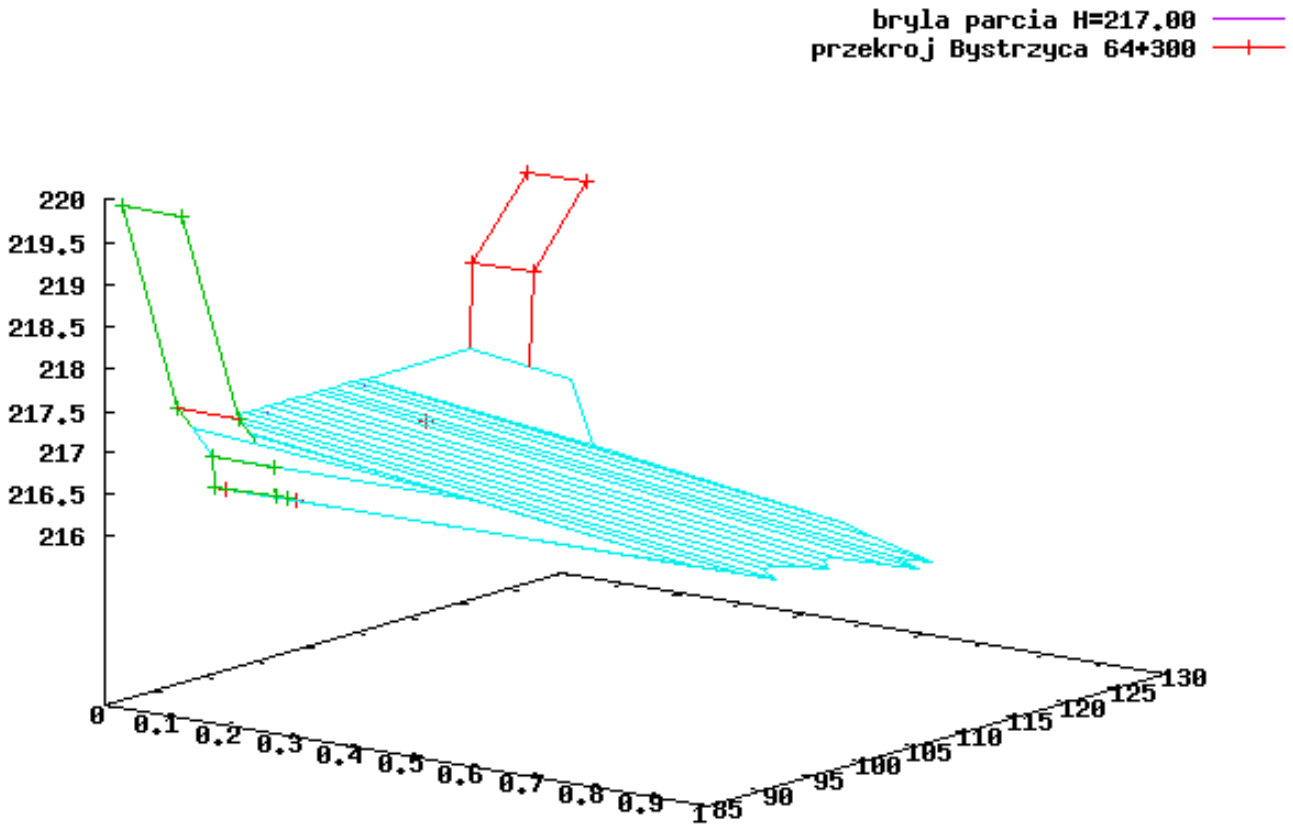
Jan Jełowicki

Stablicowane wartości parcia w przekroju

```
# Bystrzyca 64.300
# H Parcie
216.017000 0.000000
216.209750 1.743050
216.402500 12.696081
216.595250 34.486864
216.788000 68.131890
216.980750 118.888119
217.173500 189.514673
217.366250 280.730965
217.559000 393.086252
217.751750 526.939080
217.944500 682.636554
218.137250 860.565956
218.330000 1061.575078
218.522750 1286.805746
218.715500 1537.404550
218.908250 1814.517800
219.101000 2119.092289
219.293750 2451.524890
219.486500 2812.118539
219.679250 3201.176174
219.872000 3619.000731
```

Wizualizacja bryły parcia: wyniki

Jan Jełowicki



Obliczenia hydrauliczne: kod

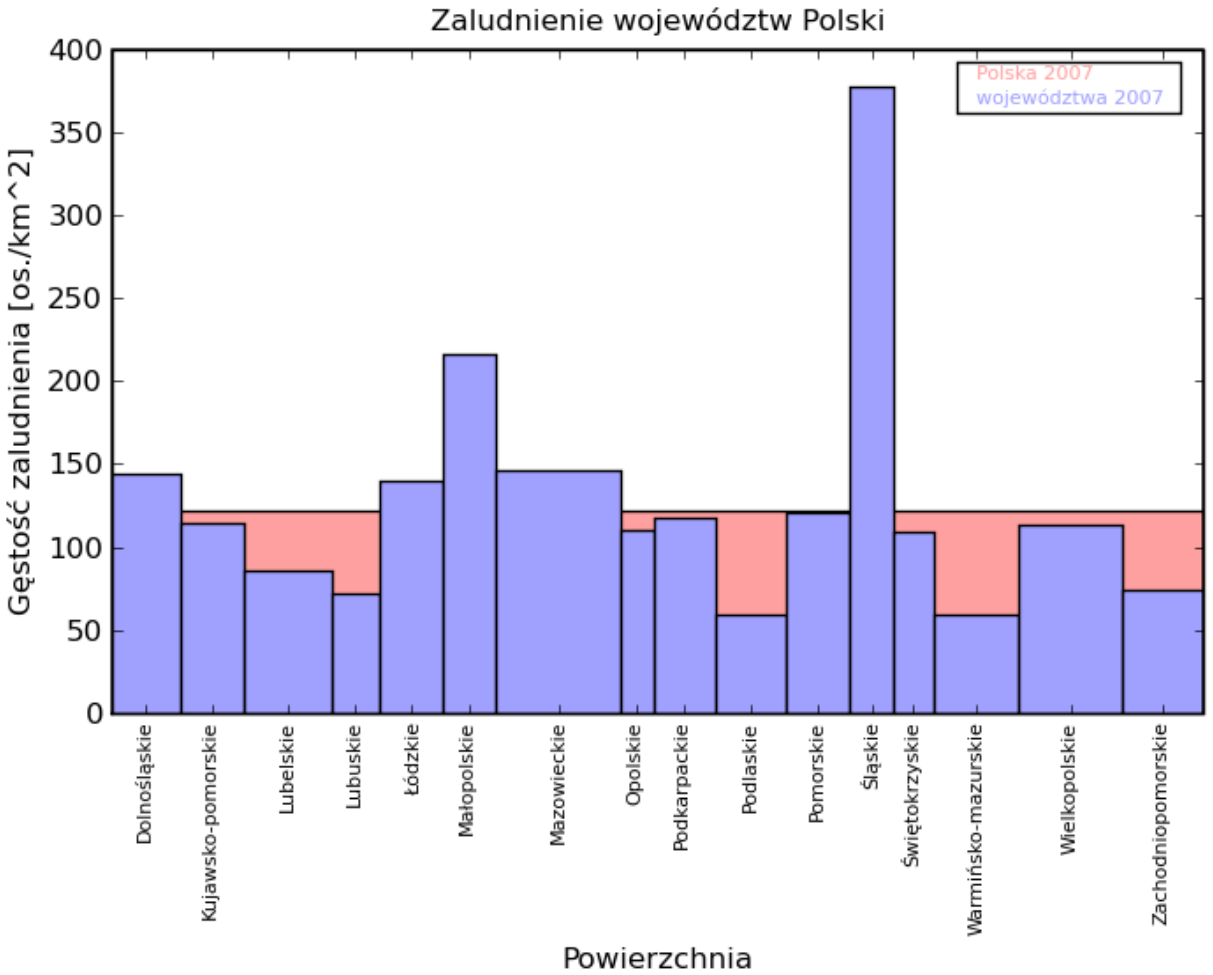

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

""" tworzenie wykresu przy użyciu programu Gnuplot
    za pośrednictwem pakietu Gnuplot-py
"""

import Gnuplot
import win32com.client as w32c
import easygui

def daneDZzPlikuTSV(nazwa):
    'pobiera dane z pliku tekstowego'
    f = open(nazwa)
    nn = f.readline()[1:-1].lstrip()
    dane = []
    for wiersz in f.readlines():
        if wiersz[0] != '#':
            dane.append(map(float,wiersz.split()))
    f.close()
    return nn, dane

def otworzArkusz(nazwa):
    sesja = w32c.Dispatch('Excel.Application')
    skoroszyt = sesja.Workbooks.Open(nazwa)
```

Raport z bazy danych

Dane wejściowe: relacyjna baza danych (`xmp/dane/admPL.db`)

Raport z bazy danych: wyniki

Jan Jełowicki



Raport z bazy danych: kod


```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
""" tworzenie wykresu na podstawie danych z bazy SQLite
    przy użyciu pakietu Matplotlib/Pylab """
import sqlite3
from matplotlib import pylab
from matplotlib.font_manager import FontProperties

baza = 'dane/admPL.db'
# parametry zapytania
q = ''' select wojewodztwo as "Województwo",
    powierzchnia as "Powierzchnia",
    ludnosc as "Ludność"
from wojewodztwa
    join wojpow on wojewodztwa.klwoj = wojpow.klwoj
    join wojlud on wojewodztwa.klwoj = wojlud.klwoj
where rok = ?; '''
rok = 2007
# nawiązanie połączenia
conn = sqlite3.connect(baza)
cursor = conn.cursor()
cursor.execute(q, (rok,))

# obróbka odpowiedzi
ludPL = 0
```

Zagadnienie optymalizacyjne

Poszukujemy minimum funkcji $f(x) = 2x_1 - 3x_2 + 2x_3$

przy ograniczeniach $x_i \geqslant 0 \quad$ dla $1 \leqslant i \leqslant 3$

$x_1 + x_2 + x_3 \leqslant 2$

$2x_1 - x_2 + 2x_3 \geqslant 1$

```
#!/usr/bin/env python
from cvxopt import matrix, solvers
c = matrix([2.0, -3.0, 2.0])
g = matrix([[-1., 0., 0., 1.,-2.],
            [ 0.,-1., 0., 1., 1.],
            [ 0., 0.,-1., 1.,-2.] ])
h = matrix( [ 0., 0., 0., 2.,-1.])
x = solvers.lp(c, g, h)
print x
xsol = x['x']
print 'x=', xsol
print c.T * xsol
```

```
$ python cvxlp.py
Optimal solution found.
{'status': 'optimal', 'dual slack': 5.982877108339868e-10, 'residual as primal infeasibility cert
x= [ 5.00e-01] [ 1.00e+00] [ 5.00e-01]

[-1.00e+00]
```

Kompletne aplikacje — przykłady

- anaconda: instalator systemu RedHat Linux
- SAGE: obszerny pakiet integrujący wiele niezależnych aplikacji obliczeniowych
- VInka: korektor typograficzny w pakiecie OpenOffice
- ...

Wnioski — pros

- Python to język o wyjątkowo spójnej i czytelnej składni
- nadaje się do typowych zastosowań w zakresie obróbki danych i obliczeń
- ma wyjątkowo bogaty zestaw bibliotek
- przekracza granice: dobrze integruje się z różnymi środowiskami
- pozostawia wyjątkowo wiele swobody, jeżeli chodzi o styl pracy
- jest popularny (już tak) i szybko się rozwija
- jako taki jest wartościowym narzędziem pomocniczym dla studenta i dla inżyniera

Wnioski — contras

- słabo nadaje się do zarządzania obszernym kodem (rzędu dziesiątków tysięcy wierszy)
- nie zawsze jest szybki (choć można korzystać z wydajnych bibliotek lub używać JIT — *just-in-time compiler*)
- nie zawsze jest wbudowany w interesującą nas aplikację (choć może mieć dla niej bibliotekę)
- nie jest samodzielny (interpreter napisany w C), ale to nie jest istotne dla użytkowników
- jako język skryptowy nie jest zaprojektowany do tworzenia zamkniętych aplikacji (choć można to robić, np. `py2exe`, Jython)

Literatura

- G. van Rossum i inni: *Python Documentation*; `http://www.python.org/doc`; 1990–2009
- G. van Rossum i inni: *Dokumentacja Pythona*; `http://docs.python.org.pl`; 1990–2003
- M. Pilgrim i inni: *Zanurkuj w Pythonie*; `http://pl.wikibooks.org/wiki/Python`; 2000–2007
- A. B. Downey: *Think Python. An Introduction to Software Design*; `http://www.greenteapress.com/thinkpython`; 2002–2008
- P. Norton i inni: *Python od podstaw*; 2006, Helion, Gliwice
- M. Lutz, D. Ascher: *Python. Wprowadzenie*; wydanie III 2009, Helion, Gliwice
- Kursy i artykuły *online*: `http://python.org.pl/kursy,jezyka.html`
- Dokumentacja *online*: `http://python.org.pl/python.html`

Środowisko (wybór)

- Interpreter Pythona i podstawowe biblioteki systemowe: `http://www.python.org/ftp/python`
- Alternatywne interpretery: Jython (Java), IronPython (.NET), ActivePython, PyPy

- Edytor SciTE: `http://www.scintilla.org/SciTE`
- Edytor SPE: `http://pythonide.blogspot.com/`
- Edytor Emacs: `http://www.gnu.org/software/emacs/`
- Debugger WinPDb: `http://www.digitalpeers.com/pythondebugger/`
- Edytor, debugger i biblioteki PythonWin32 do współpracy z systemem Windows: `http://sourceforge.net/projects/pywin32/`
- Edytor i debugger WingIDE: `http://wingware.com/downloads/wingide-101`
- Edytor i debugger Komodo Edit: `http://www.komodo.com`

Biblioteki użytkowe (wybór)

- TkInter: w podstawowej instalacji
- PyGTK+: `http://www.pygtk.org`
- PyQt: `http://www.riverbankcomputing.co.uk/software/pyqt`
- WxPython: `http://www.wxpython.org`
- interfejs graficzny EasyGUI: `http://www.ferg.org/easygui`
- NumPy: `http://www.scipy.org/`
- SciPy: `http://www.scipy.org/`
- SymPy: `http://www.sympy.org/`
- optymalizacja wypukła CVXOPT: `http://abel.ee.ucla.edu/cvxopt/`
- pakiet SAGE: `http://www.sagemath.org/`
- baza SQLite: w podstawowej instalacji
- baza PostgreSQL: `PsycoPG`
- bazy ODBC: `http://code.google.com/p/pyodbc/`
- bazy MySQL, Firebird, ...
- grafika prezentacyjna i wektorowa Matplotlib: `http://matplotlib.sourceforge.net/`
- grafika wektorowa SDXF: `http://pypi.python.org/pypi/SDXF`
- grafiki prezentacyjna i wektorowa Gnuplot-py: `http://gnuplot-py.sourceforge.net/`
- grafika rastrowa Python Imaging Library: `www.pythonware.com/products/pil`
- PythonWin32 do współpracy z systemem Windows: `http://sourceforge.net/projects/pywin32/`
- PyUNO do współpracy z OpenOffice: `http://www.openoffice.org`

Dziękuję