Poprvé s Pythonem

Python je skriptovací (interpretovaný) programovací jazyk vybavený knihovnami pokrývajícími všemožné oblasti; jednou z nich je i numerické počítání a vizualizace. K dispozici jsou volně dostupné originální implementace pro Linux/Win/Mac (www.python.org) i distribuce s přidanými balíčky a komerční podporou (např. EPD, www.enthought.com). Jazyk zahrnuje běžné postupy imperativního (příkazově orientovaného) programování, hlavním proudem je však objektově orientované programování (OOP). Za 20 let existence (1991, autor Guido van Rossum) získal Python masovou oblibu, prý pro jednoduchost, estetickou hodnotu syntaxe a výkon, zřejmě též pro přítomnost interpretovaného OOP a dobrou interoperabilitu s jinými jazyky (C, Fortran, Java). Umožňuje tedy pružné propojování vnějších aplikací, včetně přímého volání přeložených procedur; standardně je užíván ke skriptování např. ParaView, Gimpu, Imagemagicku ad. Python spolu s balíčky NumPy a SciPy útočí i na pole působnosti systémů MATLAB/Octave, tedy interaktivní řešení numerických problémů. Evoluční vývoj Pythonu až do verze 2 (aktuálně 2.7.3) byl roku 2008 přerušen návrhem zpětně nekompatibilního Pythonu 3 (aktuálně 3.3.0). Doposud je řada balíčků a distribucí dostupných jen pro Python 2, posun k Pythonu 3 je však patrný (např. verze NumPy a SciPy v Ubuntu 12.04).

Spouštění skriptů

V Linuxu bývají oba Pythony (2 i 3) s výbavou balíčků součástí distribuce operačního systému, ve Windows může posloužit dobře vybavená pythonovská distribuce EPD (aktuálně jen s Pythonem 2). Je vhodné, když skripty v Linuxu obsahují jako první shebang řádek (př. #!/usr/bin/python) a když skripty ve Windows mají registrované přípony py nebo (pro GUI) pyw, pak lze skripty spustit jejich voláním. Dalším způsobem je uvedení skriptu jako parametru interpretu, python script.py (nebo konkrétněji python2, python3). Pro interaktivní práci lze užít jak originální interpret, tak i jeho více přizpůsobené varianty, např. ipython (Interactive Python) a idle, resp. ipython3 a idle3. Vhodné je spouštět ipython s profilem pylab, ipython --pylab. Uvnitř interpretu se skripty startují postaru jako execfile('script.py') nebo nově exec(open('script.py').read()). Skript lze analyzovat překladačem do mezikódu, from py_compile import compile; compile('script.py').

Základy

help('topic'), help(topic)	nápověda k příkazu, funkci, modulu, balíčku
#	řádkový komentář
'''description'''	(i víceřádkový) dokumentační komentář, dostupný voláním help
, ; \	oddělovač prvků v kolekcích, oddělovač příkazů na řádku, poslední znak neukončeného řádku
var=expression	přiřazení hodnoty výrazu do proměnné (nasměrování ukazatele na objekt), př. a=b=c=0
var=input(prompt)	výzva a přiřazení hodnoty vložené z klávesnice
print(expression)	výpis výrazu; interaktivně stačí jen expression, jehož hodnota je pak přiřazena symbolu _
print expression	v Pythonu 2 totéž, v Pythonu 3 zrušeno
fstring%tuple	formátovaný výpis v C stylu (sprintf): decimal %nd, float %nf, %n.nf, %n.ne, string %ns ad., př. '%3.1f**%2d = %6.1f'%(2.,10,2.**10) pro '2.0**10 = 1024.0'
fstring.format(args)	formátovaný výpis v pythonovském stylu
pass	prázdný příkaz
import this	import modulu (modul this: výpis filozofie Pythonu)
from os import system; system('cmd')	import funkce z modulu a spuštění externí aplikace
quit()	ukončení interpretu, v Linuxu též ^D, ve Windows ^Z

Datové typy

Standardní typy Pythonu jsou int, (obvykle 8bytový) float, complex, bool a str; konverzní funkce jsou stejnojmenné, př. int(3.14); float('3.14'); complex(1); bool(''); bool(' '); str(True) vrátí 3, 3.14, (1+0j), False, True a 'True'. Typ int dovoluje pracovat s velkými celými čísly, př. 10**1000, zatímco float 10.**1000 přeteče. Prázdný typ definuje jedinou hodnotu, None (prázdný ukazatel).

Proměnné (ukazatele na objekty) se nedeklarují explicitně, získávají dynamický typ přiřazením hodnoty; ve jménech se rozlišuje velikost písmen. Př. i=1; a=1.; c=1+0j; S='1'; print(i,a,c,S); type(i); type(a); type(c); type(S). Proměnné lze rušit příkazem del.

K objektům strukturovaných typů patří v první řadě kolekce, konkrétně konstantní n-tice (tuple) a proměnné seznamy (list), obojí složené z prvků libovolného typu. Př. n-tice a=(1,2.,'345'); seznam b=[1,2.,'345'] nebo b=list(a); print(a,b,a[0],b[0]); nelze pak změnit a[0]=6.; lze však b[0]=6. Jak vidno, prvky kolekcí se indexují od 0; záporné hodnoty indexují od konce, b[-1] vrátí '345', počet prvků vrací len(). Seznamem jsou (v Pythonu 2) i posloupnosti

generované funkcemi range a xrange (příklad u cyklu for). Seznamem znaků jsou i řetězce, takže b[2][2] má aktuálně hodnotu '5'. V seznamech lze měnit nejen hodnoty, ale i počet prvků; zde se již nelze vyhnout postupům OOP, např. b.append(7) neboli list.append(b,7) vytvoří b[3] s hodnotou 7. Další metody (funkce vázané na objekty, zde na seznamy) jsou např. insert a remove, mnoho je řetězcových metod: př. 'Jan'.replace('Ja','Ge') + ' e '.strip() + 'RAL'.lower() pro 'General', výborně vybavená je metoda format() pro formátování řetězců. Syntaxe pro řezy v seznamech: [od_včetně:do_bez:krok]. Př. 'abeceda'[0:3]; 'abeceda'[-2:]; 'abeceda'[1::2] pro 'abe', 'da', 'bcd'. Replikátor: [1]*3; 'a'*3 pro [1,1,1], 'aaa'.
Slovníky neboli asociativní pole jsou kolekce hodnot libovolného typu indexované pomocí hodnot libovolného neměnného typu (standardní typy a n-tice). Př. a={1:1.0,2.0:'dve','tri':3}; a[1]; a[2.0]; a['tri'] pro 1.0, 'dve', 3.
Pole (i vícerozměrná) vhodná pro numerické počítání zavádí balíček NumPy a používá balíček SciPy.

Výrazy

Výrazy kombinují operandy pomocí operátorů. K aritmetickým operátorům patří běžné +-*/ (operátor / se v Pythonu 3 stal reálným dělením), operátor umocnění **, operátor celočíselného dělení // i operátory rozšířeného přiřazení v C stylu, př. i=1; i+=1. Touto cestou lze i zvětšovat seznamy, př. a=[1,2,3]; b=(4,5); a+=b. Relační operátory jsou ==, !=, <, <=, > a >= a lze je řetězit, př. x=.5; 0<x<1. K logickým operátorům se řadí not, and a or, které porovnávají operandy libovolného typu interně konvertované na bool; přitom and a or vracejí bool výsledek jen v podmíněném příkazu, jinak hodnotu rozhodujícího operandu, př. 1 and 2.; '' or 0j vrací 2. a 0j. Bool hodnoty jsou vraceny operátory is, is not pro porovnání ukazatelů, př. p=None; p is None; a operátory příslušnosti in, not in, př. 'b' in 'abc'.
Vybrané interní funkce: př. abs(3+4j); round(123.456,2); round(123.456,-2) vrací 5.0, 123.46, 100.0.
Funkce modulu math: pi, e, exp, log, log10, sqrt, hypot, pow, sin, cos, tan, acos, asin, atan, sinh, degrees, radians, copysign, factorial, fsum ad. Dostupné jsou po příkazu import math kvalifikovaným jménem, např. math.pi, nebo po from math import * bez kvalifikace, př. sin(pi/2).
Komplexní typ a jeho atributy (proměnné v objektu) a metody (funkce v objektu): př. (1+2j).real; (1+2j).imag; (1+2j).conjugate() vrací 1.0, 2.0, (1-2j). Funkce pro komplexní typ sdružuje modul cmath.

Příkazové konstrukce

Rozvětvení a opakování je jako jinde realizováno podmíněným příkazem if a cykly for a while, součástí jazyka je strukturovaný příkaz try pro ošetření výjimek. Nikde není třeba závorek, bloky (suites) se značí odsazováním (zvykem jsou 4 mezery) a předchází jim dvojtečka. Stručnou variantou podmíněného příkazu je podmíněný výraz. Proměnná indexovaného cyklu nabývá po řadě hodnot z kolekce, tedy např. z n-tice, seznamu nebo řetězce. V cyklech lze použít příkazy continue a break pro skok na začátek a za konec cyklu. Volitelný blok else lze použít ve všech těchto příkazech; skok break blok else přeskočí.

Př.
```
podmíněný příkaz
if boolExpr:
    suite
elif boolExpr:
    suite
...
else:
    suite
podmíněný výraz
exprT if boolExpr else exprF
```
```
indexovaný cyklus
for variable in collection:
    suite
else:
    suite
cyklus s podmínkou
while boolExpr:
    suite
else:
    suite
```
```
ošetření výjimek
try:
    suite
except exception:
    suite
...
else:
    suite
finally:
    suite
```

Kolekcí v indexovaném cyklu často bývá seznam (nově samostatný objekt) generovaný funkcemi range nebo xrange (paměťově úspornější varianta, v Pythonu 3 zrušená) nebo pole vracené funkcemi z balíčku NumPy, např. arange a linspace. Paměťová režie cyklu for s velkými seznamy je velká.

Př.
```
n=10; i=0; while i<n: i+=1; print(i)      # 1 2 ... 10
for i in range(1,n+1): print(i)           # 1 2 ... 10
import numpy as np
for i in np.arange(1,n+1): print(i)       # 1 2 ... n
for x in np.linspace(0.,1.,n+1): print(x) # 0.0 0.1 ... 1.0
```

Funkce

Python rozlišuje globální funkce, lokální funkce (vnořené v jiných funkcích), lambda funkce (bezejmenné funkce definované výrazem) a metody (funkce vázané na objekty). Globální a lokální funkce se vytvářejí pomocí příkazu def.

Př.
```
def name(arguments):
    suite
```

Proměnné ve funkcích jsou lokální, návratová hodnota (skalár nebo kolekce) se definuje příkazem pro návrat z funkce return (jinak je None). Formální argumenty mohou být inicializované, př. def f(a1,a2=2,a3=3), jsou pak volitelné. Navazovat skutečné a formální argumenty lze jak pozičně, tak pomocí klíčů (formálních jmen), př. f(1,a3=2).

Proměnný počet pozičních argumentů lze vyjádřit pomocí operátoru rozbalení posloupnosti *,

př.
```
def sumarg(*args): result=0; for arg in args: result+=arg; return result  # kolekce sčítá interní funkce sum
```

přičemž tentýž operátor u kolekce jako skutečného argumentu způsobí její rozbalení,

př.
```
a=range(1,11); print(sumarg(*a))        # nebo: print(sum(a))
```

Podobné je to s proměnným počtem klíčů a rozbalením slovníku pomocí operátoru **.

Argumenty s neměnným typem (standardní typy, n-tice) se předávají hodnotou, argumenty s proměnným typem (seznamy) odkazem.

Př.
```
def test(arg): arg+=arg; return arg
arg=1; print(test(arg)); print(arg)        # 2 1
arg=(1,); print(test(arg)); print(arg)     # (1, 1) (1,)
arg=[1]; print(test(arg)); print(arg)      # [1, 1] [1, 1]
```

Globální proměnné (vnější z pohledu funkce) lze ve funkci číst, má-li do nich funkce psát, musí je deklarovat příkazem global, jinak by se vytvořila stejnojmenná lokální proměnná. Jméno funkce je jako každé pythonovské jméno ukazatelem, může tak být bez dalšího prvkem kolekce nebo argumentem jiné funkce.

Krátké, tzv. lambda funkce, jsou místo samostatné sady příkazů definované pouhým (lambda) výrazem. S argumenty se zachází totožně jako u globálních funkcí, návratovou hodnotou může být skalár i kolekce.

Př. f=lambda x: x**2; g=lambda x,y: x**2+y**2; h=lambda x: (x,x**2,x**3); f(2),g(3,4),h(5) pro 4, 25, (5, 25, 125).

Funkce sdružené v souboru vytvářejí modul; modul je třeba před použitím importovat příkazem import module a jméno modulové funkce při volání kvalifikovat jménem modulu, module.f(), nebo zkráceně po import module as m jen m.f(). Jednotlivé funkce z modulu lze importovat pomocí from m import f, všechny funkce také pomocí from m import *; taková jména pak není třeba kvalifikovat. Z adresáře s moduly lze vytvořit balíček.

Př.
```
from random import randint; print(randint(1,6))  # 1 ... 6 v náhodném pořadí
```

Python Imaging Library (PIL): obrázky

Typický pythonovský pracovní postup:

1. import balíčku, 2. vytvoření objektu, 3. použití metody (funkce třídy) nebo atributu (proměnné třídy).

Př.
```
import Image # from PIL import Image
Image.open(infile).save(outfile)
im=Image.open(infile); im.size; im.rotate(30); _.show(); _.save(outfile)
```

Funkce: new, open, eval, merge, ...

Metody: crop, filter, resize, rotate, save, show, split, transform, transpose, ...

Atributy: format, size, ...

NumPy: pole a lineární algebra

NumPy zavádí do Pythonu datový typ ndarray pro pole, jedno- i vícerozměrná, v paměti ukládaná kompaktně (2D pole po řádcích). Pole se indexují od 0, předávají se (do funkcí i v přiřazovacím příkazu) odkazem. Udržují pythonovskou syntaxi, př. a=array([[1,2],[3,4]]). Přetížené operace se provádějí po prvcích: př. násobení a*b, též multiply(a,b). Maticové násobení: dot(a,b), a.dot(b). Typ prvků: standardní typy (int, float, complex, ...) a int8, int16, int32, ..., float32, float64, complex64, complex128 ad.
Typ matrix, odvozený od ndarray, je bližší pojetí MATLABu. Pole typu matrix je vždy 2D; vektor je 1xn řádková matice. Matice se inicializují konverzí pole, matrix(pole) vs. asmatrix(pole) neboli mat(pole). Nabízí se emulace matlabovské syntaxe, mat('[1 2; 3,4]'), bmat pro skládání z maticových bloků, bmat('[a;b]'). Násobení po prvcích: multiply(a,b), maticové násobení: a*b. Inverzní matice: a.I, pro pole: mat(a).I. Konverze na pole: array(a) vs. asarray(a) neboli a.A. Funkce modulu numpy.matlib vracejí typ matrix.

Ukázky

ipython --pylab
ruční import: import numpy as np; from numpy import *
nápověda: dir(np); help(np.zeros); np.zeros?
vytvoření pole: a=(1,2); b=[3,4]; c=array(a); c=array(b); d=array((a,b)) # array(a,b) nelze
výpis: a; b; print c; print(d) # (1,2), [3,4], array([3,4]), array([[1,2],[3,4]])
funkce: type(a); type(b); type(c); type(d) # tuple, list, numpy.ndarray, numpy.ndarray
atributy c.dim; c.size; c.shape; d.size; d.shape # 1, 2, (2,), 4, (2,2)
metody: d.transpose() neboli d.T # [[1,3],[2,4]]
operace: 2*a; 2*b; 2*c; c*c; c**c # (1,2,1,2) [3,4,3,4], [6,8], [9,16], [27,256]
matice: zeros((2,3)); empty((2,3),dtype=int) # inicializace float64 nulami nebo int32 čímkoliv; nikoliv zeros(2,3)
a=ones((2,3),dtype=complex); a[0,0]=2; a[1][1]=3.; a.imag[1,2]=4; print(a) # vše complex
eye; diag; random.rand ad.
posloupnosti: arange(3); arange(3.); arange(1.,3.); arange(1,3,.5) # range jen s int: range(3.) nelze
sítě: mgrid[...]/meshgrid(...)
ogrid[...]/linspace(...)
přetvarování: a=arange(1,25); b=a.reshape(4,6); c=a.reshape(2,3,4) # 2D pole po řádcích
řezy: a[0:2] pro 1,2, a[-2:] pro 23,24, a[0:4:2] pro 1,3, a[-1:-3:-1] pro 24,23
c[:,0,0]; c[0,:,0]; c[0,0,:] # [1,13], [1,5,9], [1,2,3,4]
b[0]; c[0]; c[0,0] # 1..6 (1x6), 1..12 (3x4), 1..4 (1x4)
fortranské řazení po sloupcích: b=a.reshape(4,6,order='F')
vektorizace skalární funkce pro n-tice/seznamy: (lambda x:x**2)(3); vectorize(lambda x:x**2)((3,4,5)) # 9, [9,16,25]
totéž pro pole: (lambda x:x**2)(array((3,4,5)))

Pole v přiřazovacích příkazech

Pole (i jejich řezy) se přiřazují odkazem, jde o přiřazení ukazatelů; přiřazení hodnotou provádí metoda copy().
Př. a=arange(0,3); b=a; b[1]=10; print(a[1],b[1]) # 10 10
a=arange(0,3); b=a.copy(); b[1]=10; print(a[1],b[1]) # 1 10

SciPy: numerické metody

import scipy as sp; dir(sp); help(sp.zeros); sp.zeros?

Subbalíčky

fft/fft2/fftn, integrate, interpolate, io, lib, linalg, optimize, sparse, special, stats, test, utils

Lineární algebra

import scipy.linalg as LA; help(LA); LA?
inverze inv(a), pseudoinverze pinv(), hodnost matrix_rank(a), ...
solvery: solve(a,b), lstsq(a,b); př. solve(a,b); dot(a,_) # vrátí b
rozklady: [P,L,U]=LA.lu(a); dot(dot(P,L),U) # vrátí a
U,S,Vh=LA.svd(a)
ad.

Jiné

fft, ifft, fft2, ..., rfft, irfft, ...
scipy.integrate.ode(f).set_integrator('dopri5') analogie ode45 (initial value ODE problem with RK4/5)

Matplotlib: 2D grafika

3D v počátcích (alternativa: mlab). Emulace MATLABu.

Ukázky

```
ipython --pylab
nebo import matplotlib as m
x=linspace(0,1,101); y=sqrt(x); z=x**(1./3)
plot(x,y,color='green',lw=3) # další klíče: label
plot(x,y,'r-',x,z,'go') # Matlab styl: červená čára, zelená kolečka
# show() # v interaktivní relaci se provede samo
clf(); cla() # clear figure, clear axes
xlabel('x'); ylabel('functions'); title('Title'); legend(...); tick_params(axis, direction, ...)
metody textových objektů: tx=xlabel(...); tx.set_color('b'), tx.set_weight('bold'), ...
text(xx,yy,string); figtext(xx,yy,string) # přidá texty k osám nebo do obrázku
axes((0,0,1,1)); axis('off')
grid(True)
savefig('fig') # default png, export též do jpg, pdf, eps, svg; parametry: dpi aj.
figure(n); close(n); close('all') # otevře nové okno, zavře okno, zavře všechna okna
subplot() # subplot v aktuálním okně
fig=figure(); f1=fig.add_subplot(...); f1.set_title(...); f1.colorbar(...); f1.plot(...); show()
fig=imread(file); imshow(fig); fig=mean(fig,2); imshow(fig); gray()
```

Přehled instalovaných verzí

	Ubuntu 10.04	Ubuntu 12.04	Windows EPD	poslední verze 04/2013	
python2	2.6.5	2.7.3	2.7.3	2.7.3	
python3	3.1.2	3.2.3	–	3.3.0	
numpy	1.3.0	1.6.1	1.6.1	1.7.0	s Python 3 od 1.5.0
scipy	0.7.0	0.9.0	0.10.1	0.11.0	s Python 3 od 0.9.0
matplotlib	0.99.1.1	1.1.1rc	1.1.0	1.2.1	s Python 3 od 1.2

Ruční instalace: cd package; python setup.py build; python setup.py install

Odkazy a dokumentace

www

Python home, python.org, česká stránka python.cz
Enthought Python Distribution, www.enthought.com
IPython, ipython.org
Rozdíly Pythonu 2 a 3, python3porting.com/differences.html
Jemný úvod k Pythonu, melkor.dnp.fmph.uniba.sk/~zenis/prirucky/python/Python-s
První jazyk: Python, geon.wz.cz/PrvniJazykPython
Numpy home, www.numpy.org
SciPy home, www.scipy.org, www.scipy.org/Cookbook, www.scipy.org/NumPy_for_Matlab_Users
Matplotlib, http://www.scipy.org/Cookbook/Matplotlib
PIL, www.pythonware.com/library/pil/handbook/image.htm

PDF

Guido van Rossum (otec zakladatel) and Fred L. Drake, Jr., Python tutorial, The Python language reference, Python setup and usage, The Python library reference ad.
NumPy and SciPy User and Reference guides
Hans Petter Langtangen, Python scripting for computational science
M. Scott Shell, An introduction to Python for scientific computing

L. H., 17. 4. 2013