

Poprvé s Pythonem

Python je skriptovací (interpretovaný) programovací jazyk vybavený knihovnamí 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 přidáním balíčků a komerční podporou (např. **Enthought Canopy**, 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 2.7.9) byl roku 2008 přerušen návrhem zpětně nekompatibilního **Pythonu 3** (aktuálně 3.4.3). 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 **Enthought Canopy** (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 řádce, 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 module as m` import modulu s volitelnou zkratkou, př. (filozofie Pythonu) `import this; import math; math.pi`
`from module import id` import jména (objektu, funkce aj.) z modulu, př. `from math import *; pi`
př. funkce pro spuštění externí aplikace: `from os import system; system('cmd')`
`quit()` ukončení interpretu, v Linuxu též `^D`, ve Windows `^Z`
`QUIT(); a=1; A` Python rozlišuje velikost písmen, `QUIT` není `quit` a `A` není `a`

Datové typy

– Jednoduché pythonovské typy jsou (4 nebo 8bytové) `int`, (celočíslný neomezený) `long`, (obvykle 8bytové) `float`, `complex` a `bool`; konverzní funkce jsou stejnojmenné, př. `int(3.14)`; `float('3.14')`; `complex(1)`; `bool('')`; `bool('')` vrátí `3`, `3.14`, `(1+0j)`, `False` a `True`. Typ `long` dovoluje pracovat s velkými celými čísly, př. `10**1000`, zatímco `float 10.**1000` přeteče. Prázdný typ definuje jedinou hodnotu, `None` (prázdný ukazatel).
– Ke strukturovaným typům patří v první řadě **sekvence** (druhy **kolekcí**), konkrétně neměnné **n-tice** (tuple) a měnitelné **seznamy** (list), obojí sdružující prvky 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 sekvencí se indexují od 0; záporné hodnoty indexují od konce, `b[-1]` vrátí `'345'`, počet prvků vrací `len()`. Jednoprvková n-tice se píše např. `(1,)`, neboť `(1)` je `int` výraz; v některých kontextech lze závorky n-tic vynechat, př. `a=1,2,3`. Seznamem je (v Pythonu 2) posloupnost

generovaná funkcí **range** (příklad u cyklu **for**). V seznamech (nikoliv v n-ticích) lze měnit hodnoty i počty 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**. Řezy v sekvencích: **[od_včetně:do_bez:krok]**. Př. **a=[0,1,2,3]**; **a[0:2]**; **a[-2:]**; **a[::2]** pro **[0,1]**, **[2,3]**, **[0,2]**. Replikátor: **[1]*3** pro **[1,1,1]**.

– **Řetězce** (string) jsou samostatným typem odvozeným od n-tic. Typ **str** je vystavěn na 1bytových znacích, typ **unicode** na 2bytových, př. **a=str(123)**; **b=unicode(123)**; **a**; **b** pro **'123'**, **u'123'**. Hodnotu řetězce jako neměnné n-tice nelze změnit, takže **a[1]** je **'2'** a **a[1]='4'** nelze přiřadit. Je mnoho řetězcových metod (neměnicích starý, ale vytvářejících nový řetězec): př. **'Jan'.replace('Ja','Ge') + 'e'.strip() + 'RAL'.lower()** pro **'General'**, výborně vybavená je metoda **format()** pro formátování řetězců. Řezy a replikátory v řetězcích: **'abceda'[0:3]**; **'abceda'[-2:]**; **'abceda'[1::2]** pro **'abe'**, **'da'**, **'bcd'**, **'a'*3** pro **'aaa'**.

– **Slovníky** (dictionary) 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, pro slovníky s řetězcovými indexy též: **a=dict(a=1,b=2,c=3)**; **a['a']** pro 1. Užitečné mohou být metody **keys()** a **items()**, vracejí index a pár index-hodnota. Slovníky bez indexů jsou **množiny** (set), př. **a={1.0,'dve',3}**, pro které jsou k dispozici běžné množinové operace.

– **Pole** (array, i vícerozměrná) jsou rovněž pythonovským typem, ale v podobě používané pro numerické počítání je jinak zavádí balíček **NumPy** a používá mj. balíček **SciPy**.

– **Proměnné** (ukazatele na objekty) se nedeclarují explicitně, získávají **dynamický typ** od přiřazovaného objektu. Př. **i=1**; **a=1.**; **c=1+0j**; **s='1'**; **print(i,a,c,s)**; **type(i)**; **type(a)**; **type(c)**; **type(s)**.

Přiřazení se v Pythonu děje přednostně odkazem, tj. proměnné se přiřazuje adresa, ukazatel. Např. po **a=1**; **b=a** obě proměnné ukazují na tentýž cíl, jejich **id** jsou stejná, viz **id(a)**; **id(b)**. Rozdíl proti předávání hodnoty není příliš patrný pro neměnné (immutable) cíle, podstatné to však je v případě cílů měnitelných (mutable), mj. seznamů, slovníků a polí. Pro ně totiž po jejich sdružení, **a=[1]**; **b=a**; ovlivňují změny cíle cestou jedné proměnné i pohled cestou druhé proměnné, po **a[0]=2** vypíše **b** hodnotu **[2]**. Jinak to ovšem dopadne po příkazu **a=[3]**, tedy po přiřazení nového cíle k ukazateli **a**, což nijak neovlivní cíl ukazatele **b**. Přiřazení měnitelných cílů hodnotou lze vynutit, pro jednoduché seznamy (bez vnořených seznamů) např. **a=[1]**; **b=a[:]**; **b=list(a)**; pro složené seznamy funkcí **deepcopy()**, **from copy import deepcopy**; **b=deepcopy(a)**; pro NumPy pole metodou **copy()**, př. **import numpy**; **a=numpy.array([1])**; **b=a.copy()** vytvoří dva různé cíle.

Přiřazením lze snadno sbalovat n-tice a rozbalovat n-tice a seznamy, př. **a=1,2,3**; **x,y,z=a**; **x,y,z=list(a)**.

Proměnné lze rušit příkazem **del**.

Výrazy

Výrazy kombinují operandy pomocí operátorů. K **aritmetickým operátorům** patří běžné **+-*/** (operátor **/** se v Pythonu 3 stal výhradně reálným dělením), operátor umocnění ****** (nikoliv **^**), operátor celočíselného dělení **//** a zbytku **%** 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. **Bitové a množinové operátory** zahrnují **~&|^**, bitové navíc i posuny. Bool hodnoty jsou vráceny operátory **is**, **is not** pro porovnání ukazatelů, př. **p=None**; **p is None**; a operátory příslušnosti **in**, **not in**, př. **'b' in 'abc'**; **2 in {1,2,3}**.

Vybrané interní funkce: př. **abs(3+4j)**; **pow(2.,3)**; **round(123.456,2)**; **round(123.456,-2)** vrací 5.0, 8.0, 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**, **trunc**, **floor**, **ceil** 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 | indexovaný cyklus | ošetření výjimek |
| | if boolExpr: | for variable in collection: | try: |
| | suite | suite | suite |
| | elif boolExpr: | else: | except exception: |
| | suite | suite | suite |
| | ... | cyklus s podmínkou | ... |
| | else: | while boolExpr: | else: |
| | suite | suite | suite |
| | podmíněný výraz | else: | finally: |
| | exprT if boolExpr else exprF | suite | suite |

Kolekcí v indexovaném cyklu často bývá seznam (nově samostatný objekt) generovaný funkcí **range** s variantami rozhraní (stop), (start,stop) a (start,stop,step) nebo paměťově úspornější varianta **xrange** (v Pythonu 3 zrušená) nebo pole generované funkcemi z balíčku NumPy, např. **arange**(start,stop,step) a **linspace**(start,stop,num).

```
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 ... 10
for x in np.linspace(0,1,n+1): print(x) # 0.0 0.1 ... 1.0
```

V indexovaných cyklech je užitečná možnost vytvářet kolekce párů index-hodnota pomocí funkce **enumerate** pro n-tice a seznamy a metodou **items()** pro slovníky.

```
Př. a=[1.,2.,3.]; for i,x in enumerate(a): print(i,x);
a={1:1.,2:2.,3:3.}; for i,x in a.items(): print(i,x);
```

Seznamy umožňují kompaktní zápis nahrazující konstrukce s (1+) cykly a (0+) podmínkami (**list comprehension**):
[expression for x in sequence if condition]

```
Př. a=[x for x in range(1,n+1)]; a=[x for x in np.arange(0,2,.1) if x<=1]; a=[x for x in np.linspace(0,1,n+1)].
```

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

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).

Nezměrné je množství standardních pythonovských funkcí. Čtyři ukázky: **filter**(function,iterable), **map**(function,iterable), **reduce**(function,iterable), **sorted**(iterable,cmp,key,reverse).

Př. `a=[1,3,5,2,4]; filter(lambda x: x%2!=0,a); map(lambda x: x**2,a); reduce(lambda x,y: x+y,a); sorted(a).`

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`

NumPy: pole a lineární algebra

NumPy zavádí do Pythonu datový typ **ndarray** pro pole, jedno- i vícerozměrná. Pole jsou v paměti uložena kompaktně, s „C“ řazením prvků (2D pole po řádcích). Vytvářejí se např. z kolekcí funkcí **array**, `a=array((1,2))`, `b=array([[1,2],[3,4]])`, nebo voláním funkcí **arange**, **linspace** aj. Prvky pole mají shodný básový typ a indexují se od 0, po `a=array((1,2))` se výpisem řezu `a[0:]` získá `array([1., 2.])`. Básový typ může být numerický nebo řetězcový pythonovský (int, float, complex, str, ...; int je ovšem nejvýše 8bytový) a také int8, int16, **int32**, **int64**, ..., float32, **float64**, complex64, **complex128** ad. Př.: v poli `a=array((1,))`; `type(a[0])`; jsou prvky typu int32/int64 (v 32/64bitovém operačním systému), po `a=array((1,))`; jsou float64, po `a=array((1,),'int32')` je to explicitně 4bytový znaménkový integer.

Pole se (do funkcí i v přiřazovacím příkazu) předávají odkazem, např. po `b=a` se přiřazením `b[0]=3`. mění i `a[0]`. Pro předání hodnoty slouží metoda **copy**, po `b=a.copy()` jsou pole a, b v dalším nezávislá. Přiřazení do řezu se také děje hodnotou, nikoliv odkazem.

Přetížené operace se provádějí po prvcích: př. násobení `a*b`, též **multiply(a,b)**. Pro maticové násobení jsou určeny funkce **dot(a,b)** a metoda `a.dot(b)`. NumPy přetěžuje mnoho standardních pythonovských funkcí, např. all, any, sum, prod/product, min, max a doplňuje mnoho dalších, např. empty, count_nonzero.

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`

`a=array((1,2,'3'),dtype=float)` s explicitním určením básového typu

`a=reshape((1,2,3,4),(2,2),order='F')` pro fortranské řazení po sloupcích

`arange(start,stop,step)`, `linspace(start,stop,num)`

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 (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: pythonovská vizualizace, volitelně à la MATLAB

Oblíbený balíček pro vizualizaci numerických výsledků. Ke své objektové syntaxi nabízí obal připomínající syntaxi MATLABu, a to pomocí submodulu `pyplot`, `matplotlib.pyplot`. Vstupními daty jsou NumPy pole (ne však typu `matrix`). Zpřístupnění `pyplotu` s vtažením modulů do globálního prostoru jmen:

```
from numpy import *; from matplotlib.pyplot import *
```

`ipython --pylab` (`ipython` s profilem `pylab` importuje `numpy` a `pyplot` shodným způsobem)

alternativa s konzervativnějším držením jmen v oddělených jmenných prostorech:

```
import numpy as np; import matplotlib.pyplot as plt
```

Uživatel vytvoří objekt `Figure`, v jeho rámci několik objektů `Axis`. Ty mají každý svůj `Title`, `Axis` a `Line`, vše objekty souhrnně nazývané `Artist`.

Ukázka skriptu v procedurálním MATLABovském stylu v `pyplotu`:

```
# matplotlib: pyplot (MATLAB) style
```

```
# coding: utf-8
```

```
from numpy import *; from matplotlib.pyplot import *
```

```
x=linspace(-2,2,100); y2=x**2; y3=x**3
```

```
plot(x,y2,x,y3); axis([-1,2,-2,5]);
```

```
title(u'Několik polynomů'); xlabel(r'$x$'); ylabel(r'$y$'); legend([r'$y=x^2$',r'$y=x^3$'],loc='upper left')
```

```
savefig('a1.png'); show()
```

a pro více panelů:

```
# matplotlib: pyplot (MATLAB) style
```

```
# coding: utf-8
```

```
from numpy import *; from matplotlib.pyplot import *
```

```
x=linspace(-2,2,100); y2=x**2; y3=x**3
```

```
figure(); suptitle(u'Několik polynomů')
```

```
subplot(211); plot(x,y2,color='blue',lw=2,label=r'$y=x^2$') # nebo: plot(x,y2,'b-')
```

```
axis([-1,2,-2,5]); ylabel(r'$y$'); axhline(0,color='black',lw=2); legend(loc='upper left')
```

```
subplot(212); plot(x,y3,color='green',lw=2,label=r'$y=x^3$')
```

```
axis([-1,2,-2,5]); xlabel(r'$x$'); ylabel(r'$y$'); grid(True); legend(loc='upper left')
```

```
savefig('a2.png'); show()
```

Při kódování skriptu v UTF-8 a uvedení fráze `coding: utf-8` v prvním nebo druhém řádku lze užívat UTF-8 řetězce u'KŮŇ', kdekoli lze použít i raw řetězce `r'$\log x$'` s matematickými zápisy v LaTeXovské syntaxi. (Překladač LaTeXu je součástí matplotlibu, lze volat i externí překladač, lze nastavit zpracování všech textů obrázku LaTeXem.) Formát obrázku se nastaví volbou přípony souboru (`png, jpg, pdf, eps, svg` ad.). Příkaz `show()` se píše ve skriptech, nikoliv při interaktivní práci.

Ukázka skriptu v objektové syntaxi s pyplotovskou inicializací (výstupní obrázek stejný jako výše):

```
# matplotlib: OOP style with pyplot init
# coding: utf-8
from numpy import *; import matplotlib.pyplot as plt
x=linspace(-2,2,100); y2=x**2; y3=x**3
# -- one way
# fig=plt.figure(); ax=fig.add_subplot(111)
# -- other way
fig,ax=plt.subplots()
ax.plot(x,y2,x,y3); ax.axis([-1,2,-2,5])
ax.set_title(u'Několik polynomů'); ax.set_xlabel(r'$x$'); ax.set_ylabel(r'$y$');
ax.legend(['r'$y=x^2$',r'$y=x^3$'],loc='upper left')
fig.savefig('a3.png'); plt.show()
```

Více panelů:

```
# matplotlib: OOP style with pyplot init
# coding: utf-8
from numpy import *; import matplotlib.pyplot as plt
x=linspace(-2,2,100); y2=x**2; y3=x**3
fig,(ax1,ax2)=plt.subplots(2,1); fig.suptitle(u'Několik polynomů')
ax1.plot(x,y2,color='blue',lw=2,label=r'$y=x^2$'); ax1.axis([-1,2,-2,5])
ax1.set_ylabel(r'$y$'); ax1.axhline(0,color='black',lw=2); ax1.legend(loc='upper left')
ax2.plot(x,y3,color='green',lw=2,label=r'$y=x^3$'); ax2.axis([-1,2,-2,5])
ax2.set_xlabel(r'$x$'); ax2.set_ylabel(r'$y$'); ax2.grid(True); ax2.legend(loc='upper left')
fig.savefig('a4.png'); plt.show()
```

Výběr dalších funkcí a metod: zjištění backendu `print(plt.get_backend())`, otevření a zavření oken `figure(n)`, `close(n)`, `close('all')`, `clear figure clf()`, `clear axes cla()`, popis os `tick_params(axis, direction, ...)`, pro texty `text(xx,yy,string)`, `figtext(xx,yy,string)`, metody textových objektů `tx=xlabel(...)`; `tx.set_color('b')`, `tx.set_weight('bold')` aj.

Možnosti zpracování obrázků balíčkem PIL (Pillow) jsou zde rozšířeny o práci s NumPy poli. Př. vizualizace matice:

```
from numpy import *; import matplotlib.pyplot as plt
n=10; x=arange(1,n+2); [mi,mj]=meshgrid(x,x); hilb=array(1./(mi+mj-1),float32)
plt.imshow(hilb); plt.title('Hilbert matrix'); plt.colorbar(); plt.savefig('hilbert.png'); plt.show()
```

Pro 3D obrázky je určen toolkit (nadstavba) `mplot3d`, mapové projekce obsahují externí toolkity `basemap` a `cartopy`. Dokumentační PDF matplotlibu obsahuje 900 stránek příkladů.

Python Imaging Library (PIL) a následník Pillow: 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
infile='A.jpg'; outfile='A.png'; Image.open(infile).save(outfile)
outfile='Arot.jpg'; im=Image.open(infile); im.size; imrot=im.rotate(30); imrot.save(outfile); imrot.show()
```

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

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

Atributy: `format`, `size`, ...

Přehled instalovaných verzí

| | Ubuntu 12.04 | Ubuntu 14.04 | Win/EPD 7.3-2 | W/Canopy 1.5.2 | poslední verze 12/2015 |
|------------|--------------|--------------|---------------|----------------|------------------------------|
| python2 | 2.7.3 | 2.7.5 | 2.7.3 | 2.7.6 | 2.7.10 |
| python3 | 3.2.3 | 3.4.0 | – | – | 3.5.1 |
| numpy | 1.6.1 | 1.8.2 | 1.6.1 | 1.8.1 | 1.10.1 (s Python 3 od 1.5.0) |
| scipy | 0.9.0 | 0.13.3 | 0.10.1 | 0.14.1 | 0.16.1 (s Python 3 od 0.9.0) |
| matplotlib | 1.1.1 | 1.3.1 | 1.1.0 | 1.4.2 | 1.4.3 (s Python 3 od 1.2) |
| sympy | 0.7.1 | 0.7.4 | – | 0.7.6 | 0.7.6 (s Python 3 od 0.7.2) |

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 Canopy, 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>, <http://mamut.spseol.cz/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. 12. 2015