Python

Stručné info pro zájemce o programování ve skriptovacím jazyce Python.

Proč Python?

Protože je intuitivní a překvapivě jednoduchý k psaní kódu.

Protože je vstřícný pro porozumění a přitom dostatečně bohatý pro řešení mnoha úkolů.

Protože může pracovat jako interpret i compiler.

Protože je to už svou podstatou objektový jazyk.

Protože umí pracovat s mnoha vysokoúrovňovými typy (řetězce, seznamy, asociativní pole, množiny).

Protože má základnu k vyhledání řešení problémů na českém internetu.

A nakonec je nutno k jeho kladům přičíst přenositelnost mezi Lin, Win, MAC, OS2, WinCE platformami.

Instalace jazyka Python

Python je implicitně dodáván s distribucí Ubuntu takže není třeba jej instalovat.

Spuštění

V terminálu zadejte jméno jazyka příkazem `python`. Obdržíte kupříkladu takovouto odpověď:

```
Python 2.7.1+ (r271:86832, Apr 11 2011, 18:05:24)
[GCC 4.5.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Tři znaky »> oznamují, že je interpret Python připraven vykonat zadávané příkazy. I když okno

terminálu vypadá stejně jako předtím, vkládané příkazy bude zpracovávat Python, takže kupříkladu `ls` či `cd` nyní fungovat nebude. K ukončení Pythonu a návratu řízení zpět terminálu poslouží `Ctrl+D` či příkaz

```
>>> quit()
```

První krůčky

Světoznámé úvodní zvolání v Pythonu může mít takovýto tvar:

```
>>> print  "Hello, World!"
```

Po stisku klávesy `Enter` se příkaz provede a program odpoví vypsáním **Hello, World!**. Není-li specifikováno jinak, slouží jako standardní výstup okénko terminálu.

Program aneb posloupnost více příkazů lze zapsat libovolným textovým editorem ve formě prostého textu a uložit do souboru. Textové editory mívají podporu Pythonovské syntaxe, což činí psaní zdrojového kódu výrazně pohodlnější. I když to není povinné, bývá zvykem značit si jej příponou **.py**. Spuštění takového programu lze docílit prostým napsáním jména programu do příkazové řádky, kupříkladu

```
>>> MujPrvniPokus.py
```

Program je možno spustit i z terminálu zavoláním interpretu Python s parametrem jména programu, např. příkazem

```
Python MujPrvniPokus.py
```

Pokud se nedaří, možná chybí specifikace cesty k programu. V takovém případě lze buď přejít do složky s programem pomocí příkazu `cd` nebo připsat cestu před jméno programu. Má-li uživatel jménem *uzivatel* program umístěn v adresáři *programy*, příkaz bude `python /home/uzivatel/programy/MujPrvniPokus.py`.

Něco se nepovedlo

Může se stát že program přestane odpovídat a neustále pracuje. K tomu může dojít kupříkladu díky chybě v programu jež způsobí běh v nekonečné smyčce, takzvané zacyklení.

```
>>> while true: true
```

Tento ukázkově chybný příkaz bude mít za následek vypsání **…** a program se dostane do nekonečné smyčky. Obvykle by mělo fungovat `Ctrl+c`, ale může se stát že program nebude reagovat. Jak z toho ven? Stisk `Ctrl+z` program pozastaví, řízení se předá zpět příkazové konzoli a objeví se hlášení:

```
>>> while true: true
...
```

```
[1]+  Pozastaven       python
```

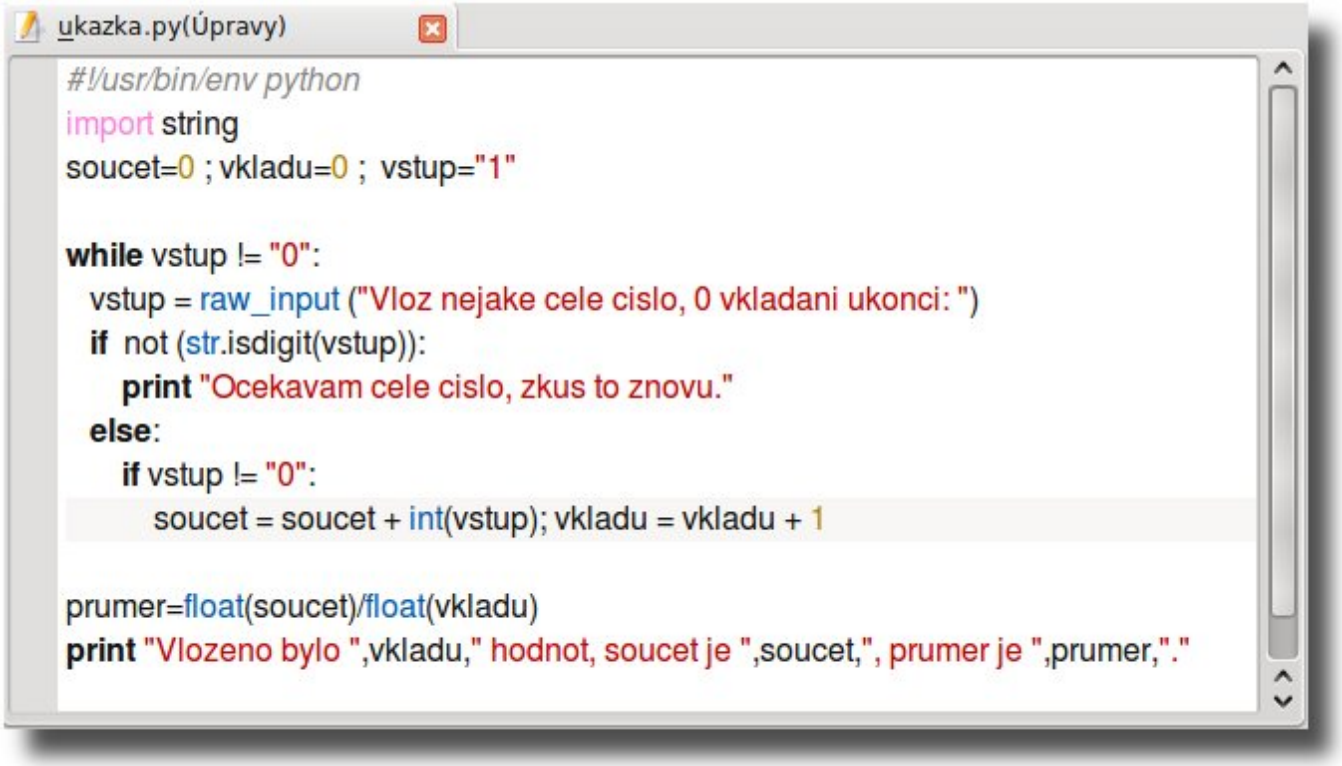
Číslo v hranaté závorce je číslem procesu. Obvyklým příkazem `kill -9 %N` lze daný program násilně ukončit. Za N nutno dosadit číslo ze závorky, v daném příkladu to bylo číslo 1.

Malý ukázkový program

```
#ukazkovy program - prumer cisel
#jak vidno, radky zacinajici znakem "#" jsou programatorske poznamky.
#Pro samotny interpreter Pythonu nemaji zadny vyznam,
#presto zvlaste v rozsahlejsich programech maji sve dulezite misto
#nebot zvysuji pochopitelnost kodu. V tomto pripade je poznamek opravdu
hodne,
#nebot jde o pokus osvetlit cely programek krok za krokem.

import string #pripoji k programu preddefinovanou knihovnu string,
              #ktera se hodi pro urceni typu obsahu promenne vstup

#inicializace promennych, kupr. promenna soucet bude celociselny typ s
vychozi hodnotou nula
soucet=0
vkladu=0
vstup="1" #promenna vstup bude typu znaku - pro pripad ze by uzivatel vlozil
chybne zadani

""" (aneb viceradkovy komentar)
Strukturovani Pythonu je excelentne jednoduche - zadne zavorkove
silenstvi...
ridi se prostym odsazovanim mezerami nebo tabulatory zleva
coz samosebou znamena, ze kod jazyku Python musi byt psan zodpovedne od
zacatku.
Je jedno zda se zvoli mezery ci tabelatory, ale v prubehu programu uz
nelze metodu odsazeni menit.
Zde vypada kod programu sedive a nehezky, ale textove editory prostredi
Ubuntu
dokazi zobrazovat kod hezky barevne odliseny a tudiz mnohem prehlednejsi.
"""

#pocatek smycky while (dělej dokud); prikazy se budou opakovat tak dlouho
dokud v promenne vstup
#nebude znak "0". Vyraz != zamena neni rovno.
while vstup != "0":
   #vlozeni hodnoty z klavesnice do promenne vstup
   vstup = raw_input ("Vloz nejake cele cislo, 0 vkladani ukonci: ")
   #nasleduje test zda uzivatel vlozil skutecne cislo
   #zde se pouziva preddefinovana funkce isdigit z importovane knihovny
string
   if  not (str.isdigit(vstup)):
       #pokud v promenne vstup neni korektni numericky obsah, vypise se
```

```
upozorneni
       print "Ocekavam cele cislo, zkus to znovu."
   else: #a pokud je v promenne vstup vlozeno korektni cislo, tak program
pokracuje zde
       if vstup != "0": #jen v pripade ze nebyla vlozena ukoncujici nula
           soucet = soucet + int(vstup) #program spocita to co chceme
           vkladu = vkladu + 1#a v teto promenne se uchovava pocet spravne
vlozenych cisel

#mezera nad touto radkou rika interpreteru, ze zde konci smycka while
#pro spocteni desetinneho vysledku z integer hodnot, lze pouzit funkci
pretypovani
#integer na float, tedy celociselne promenne na promennou s pohyblivou
desetinnou carkou
prumer=soucet/float(vkladu)
#zbyva vypsat vysledky
print "Vlozeno bylo ",vkladu," hodnot, soucet je ",soucet,", prumer je
",prumer,"."
```

Samotný kód programu (v tomto případě v editoru KrViewer) vypadá mnohem střízlivěji a přehledněji:



Jde to i lépe

Python samozřejmě umožňuje a podporuje tvorbu mnohem efektivnějšího a elegantnějšího kódu. Různé věci se v něm dají dělat zábavně prostě. Např. vypsání souboru.

```
print(open("priklad_souboru.txt").read())
```

Je-li třeba vypsat soubor po řádcích, stačí malá úprava.

```
for radek in open("priklad_souboru.txt"):
   print radek
```

anebo

```
print(open("priklad_souboru.txt").readlines())
```

Výše uvedený zdlouhavě rozepsaný prográmek lze kupříkladu zjednodušit do pouhých dvou řádků:

```
pole=map(float, raw_input("Vloz cisla oddelena mezerou <Enter>").split())
print "Vlozeno:",len(pole),"hodnot, soucet je",sum(pole), ", prumer
je",sum(pole)/len(pole),"."
```

O efektivnosti tvorby v Pythonu je možno se přesvědčit např. v ukázce která díky využití různých užitečných schopností Pythonu najde všechna prvočísla v rozmezí od 2 do daného čísla (ono populární Eratosthenovo síto).

```
nums = range(2, 2000)
for i in range(2, 8):
    nums = filter(lambda x: x == i or x % i, nums)
print nums
```

Python je dobře vybaven na boj s vysokou aritmetikou - typ `integer` v něm má přesnost která je omezena pouze pamětí počítače. Příkaz

```
from math import factorial;print(factorial(70)*113)
```

vrátí výsledek spočítaný na plný počet platných řádů, v tomto případě 103 míst. Pro práci v plovoucí desetinné čárce s libovolnou přeností slouží modul Decimal.

Ačkoliv při tvorbě kódu není programátor tlačen k používání tříd, Python plně podporuje objektově orientované programování. Dalo by se říci, že je svou podstatou dovedeno k nejvyšší jednoduchosti.

```
class MojeTrida:
   def CoJsi(self):
      print("Já jsem MojeTrida")
```

I takto průzračná deklarace třídy Pythonu postačí, dokonce ani parametr se nemusí nezbytně nutně jmenovat `self`.

Podporu Pythonu poskytuje velké množství knihoven - knihovny grafických rozhraní, přístupu k databázím, zpracování grafů, neuronových sítí, kryptografie, ke službám operačního systému, GUI, HTTP, FTP, POP3, SMTP a jiným protokolům. Pro nebojsy je připraven např. modul regulárních výrazů či multithreadingu. Určeno je i několik modulů pro přístup k vnitřním mechanismům Pythonu (garbage collector, parser, kompiler). Mnoho knihoven je obsaženo již v implicitní instalaci jazyka, další se dají snadno instalovat přímo z repozitářů pomocí `Synapticu`, anebo získat z různých internetových zdrojů.

Ve výchozím stavu Python nepodporuje české kódování. Naštěstí se dá snadno přemluvit - na začátek programu stačí uvést druh kódování (v tomto případě UTF-8).

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

Práci s Unicode Python samozřejmě podporuje. U verzí 2.x k tomu slouží označení **u**

```
>>> type("žšč");type(u"žšč")
<type 'str'>
<type 'unicode'>
```

Python 3.x již prefix **u** nevyžaduje.

Zvláštnosti hroznýše

Python má - řekněme si otevřeně - i svá jistá specifika, jejichž znalost bude pro začátečníka nepochybně velmi užitečná. Některé postřehy:

Dělení nefunguje!

```
>>> a=5; b=2; print a/b
2
```

No toto…! Důvod je však prostý. Python2.x podporuje nativně celočíselnou aritmetiku. Pro získání správného výsledku je potřeba použít buď funkce float(), nebo jednomu z argumentů prostě připsat desetinnou čárku:

```
>>> a=5; b=2.0; print a/b
2.5
```

Python 3.x již tuto otázku má vyřešenou, běžné dělení funguje „lidsky“.

Nedopočítává to cykly!

```
>>> for i in range(1,4): print i,
1 2 3
```

Kde je čtyřka? Inu, Python bere rozsah (a,b) ve smyslu matematického zápisu <a,b), tedy pro hodnoty i: a⇐i<b. Pokud není nezbytně potřeba přesně daný rozsah hodnot, je výhodnější použít cykl ve formě např.

```
>>> for i in range(4):print i,
0 1 2 3
```

Mění to něco jiného než chci!

Mějme případ kdy seznam **sez1** obsahuje jediné písmeno. A protože chceme tento seznam zachovat,

uděláme si prostým přiřazením jeho kopii do seznamu **sez2**, se kterým pak budeme pracovat v domnění, že původní obsah **sez1** zůstane nezměněn:

```
>>> sez1=["a"];sez2=sez1;sez2.append("b");print sez1,sez2
['a', 'b'] ['a', 'b']
```

Jakto že se změnil i sez1? Je to tím že názvy proměnných ukazují na datový obsah, a ten je v případě přiřazení nezměněn; tudíž `sez2=sez1` vytvoří odkaz jménem sez2 na ten samý datový objekt. Netřeba zde bádat a rozebírat nakolik je to logické či ne, stačí vědět jak se tomu vyhnout.

```
>>> sez1=["a"];sez2=[]+sez1;sez2.append("b");print sez1,sez2
['a'] ['a', 'b']
```

Kopii objektu lze vytvořit též pomocí modulu **Copy**

```
>>>from copy import copy
>>> sez1=["a"];sez2=copy(sez1);sez2.append("b");print sez1,sez2
['a'] ['a', 'b']
```

Odkazy

- http://www.python.org domovská stránka Pythonu
- http://howto.py.cz opravdu skvělé české stránky pro každého začátečníka
- http://www.linuxsoft.cz/article_list.php?id_kategory=217 jiné hezké stránky pro začínající Pythonýry
- http://k-prog.wz.cz/python/tkinter1.php nenásilný úvod do grafického rozhraní Tkinter
- http://www.openbookproject.net/pybiblio/gasp/course/G-gasp.html stránky o grafickém rozhraní GASP
- http://programujte.com/?akce=diskuze&kam=diskuze&sekce=40-python diskuzní fórum o otázkách kolem programování v Pythonu
- http://python.wraith.cz/uvod-vykon.php velmi zajímavé stránky o modulu, který dovede v některých případech zmnohonásobit rychlost Pythonu