České vysoké učení technické v Praze FIT

# Programování v Pythonu

Jiří Znamenáček

*Příprava studijního programu Informatika je podporována projektem financovaným z Evropského sociálního fondu a rozpočtu hlavního města Prahy.*

*Praha & EU: Investujeme do vaší budoucnosti*

# Python – Zpracování vstupu

## Úvod

Uživatelský vstup do programu můžeme v zásadě rozdělit na dva druhy:

- přímý vstup z klávesnice do běžícího programu
- vstup do programu v podobě parametrů

Python nabízí několik různých způsobů, jak zařídit obě možnosti vstupu. V dalším se seznámíme se základními z nich.

## Funkce input()

**I.** Nejzákladnějším způsobem, jak z běžícího programu vyvolat interakci s uživatelem, je pomocí metody `input()`:

```
>>> answer = input()
Ahoj, světe!
>>> answer
'Ahoj, světe!'
```

Uvedený kód čeká na zadání vstupního *řetězce* ukončeného klávesou ↵. Zadaný *řetězec* (bez odřádkování, to jen potvrdilo vstup) je předán jako návratová hodnota funkce (v našem případě tedy do proměnné *answer*).

Má-li vstupem být třeba celé číslo, je třeba vstup z vráceného řetězce na celé číslo, tj. např. `answer = int(answer)`.

> → V Python'u 2.x se tímto způsobem chovala globální funkce `raw_input()`.

**II.** Metoda `input([PROMPT])` má jeden velmi užitečný nepovinný řetězcový parametr, tzv. *prompt*:

```
>>> answer = input('Zadej text: ')
Zadej text: Ahoj, světe!
>>> answer
'Ahoj, světe!'
```

PS: Chcete-li chování této uživatelské výzvy vylepšit, můžete před jejím

použitím naimportovat modul `readline` (dostupný pouze na platformě Unix!). Vstupní řádka pak „získá“ mimo jiné historii a větší možnosti editace (některé klávesové zkratky apod.).

# Zpracování parametrů skriptu

Co se vstupních parametrů skriptu týká, máme v Python'u k dispozici kromě jednoho zcela základního modulu:

0. `sys.argv` – přístup k parametrům skriptu na nejnižší úrovni

...také několik „generací“ modulů vyšší úrovně:

1. `getopt` – nejstarší (a co do schopností nejméně vybavená) nadstavba pro práci s argumenty příkazové řádky
2. `optparse` – vylepšení předchozího modulu; dostupné od verze Python'u 2.3
3. `argparse` – poslední generace modulů pro zpracování argumentů příkazové řádky; dostupné od Python'u 3.2

Zatímco modul `sys.argv` je zcela základní a veškerou práci s parsováním vstupních agumentů na potřebné hodnoty si musíte obstarat sami, další moduly poskytují nejrůznější vylepšení, jak téhož dosáhnout snadněji. Z důvodu aktuální rošířenosti se podíváme na pár příkladech pouze na modul `optparse`.

# Modul sys.argv

**I.** Nejjednodušším způsobem, jak předat právě spouštěnému programu (skriptu) nějaké údaje, je využít *vstupních parametrů skriptu*:

**Příkazová řádka:**

```
python3 argvs.py
```

**Program** *argvs.py* **:**

```
import sys

print( sys.argv )
print( sys.argv[0] )
print( sys.argv[1:] )
```

**Výstup:**

```
['argvs.py']
argvs.py
[]
```

**Příkazová řádka:**

```
python3 argvs.py arg1 arg2 arg3
```

**Program** *argvs.py* **:**

```
import sys

print( sys.argv )
print( sys.argv[0] )
print( sys.argv[1:] )
```

**Výstup:**

```
['argvs.py', 'arg1', 'arg2', 'a
argvs.py
['arg1', 'arg2', 'arg3']
```

Vidíme, že jméno spouštěného programu je předáno jako první prvek v seznamu argumentů, zatímco vše ostatní je „rozsekáno“ podle mezer a předáno jako další prvky seznamu.

**II.** Pokud bychom chtěli předat jako jeden parametr text, který obsahuje mezery, muzíme ho buď uzávorkovat (což funguje všude):

**Příkazová řádka:**

```
python3 argvs.py "arg1a arg1b" arg2
```

**Program** *argvs.py* **:**

```
import sys

print( sys.argv )
print( sys.argv[0] )
print( sys.argv[1:] )
```

**Výstup:**

```
['argvs.py', 'arg1a arg1b', 'ar
argvs.py
['arg1a arg1b', 'arg2']
```

..nebo mezery odiskejpovat (což nezabere pod Windows a navíc se to hůř čte):

**Příkazová řádka:**

```
python3 argvs.py arg1a\ arg1b arg2
```

**Program** *argvs.py* **:**

```
import sys

print( sys.argv )
print( sys.argv[0] )
print( sys.argv[1:] )
```

**Výstup:**

```
['argvs.py', 'arg1a arg1b', 'ar
argvs.py
['arg1a arg1b', 'arg2']
```

Typické použití v programu bude vypadat asi následovně:

```
import sys

if len( sys.argv ) <= 1:
    print( "Usage: {} ARGUMENT".format( sys.argv[0] ) )
    sys.exit()
```

→ Tedy: Očekáváme-li na vstupu argument(y) a nedostaneme je, vypíšeme uživateli správný způsob volání skriptu a pomocí `sys.exit()` ukončíme jeho vykonávání, aby mohl být znovu zavolán (a tentokrát už snad správně).

Samozřejmě si musíte pohlídat, že uživatel zadal potřebný počet parametrů a ve správném tvaru. Pokud ne, dejte mu o tom vědět a ukončete provádění skriptu. (Respektive použijte nějaké výchozí hodnoty, je-li to možné, ale pak o tom též nějakým způsobem informujte. Alespoň v README :)

PS úplně na závěr: Nezkoušejte míchat parametry s *wildcard*-znaky, výsledky by vás nejspíš poněkud překvapily...

# Modul optparse – příklad

Ukažme si nejdříve chování kódu vybaveného modulem *optparse* při různých voláních skriptu z příkazové řádky. Daný skript pouze (přehledně) vypisuje, jaké argumenty na příkazové řádce obdržel:

**I.** Nevíme, jak program vůbec zavolat, tak to zkusíme bez parametrů:

**Příkazová řádka:**

```
python3 optparse.mod.py
```

**Výstup:**

```
Usage: optparse.mod.py [options] filename

optparse.mod.py: error: wrong number of arguments
```

**II.** Aha, takže přepínače nejsou povinné, ale jméno souboru ano. Tak ho zkusíme přidat (*test.html*):

**Příkazová řádka:**

```
python.exe optparse.mod.py test.html
```

**Výstup:**

```
{'xhtml_flag': False, 'cssfile': 'style.css'}
['test.html']
```

**III.** Vida, přepínače mají nějaké výchozí hodnoty. Nebyla by k nim nějaká nápověda?

**Příkazová řádka:**

```
python3 optparse.mod.py --help
```

**Výstup:**

```
Usage: optparse.mod.py [options] filename

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -x, --xhtml           create a XHTML template instead of HTML
  -c CSSFILE, --cssfile=CSSFILE
                        CSS file to link
```

→ A jak je dobrým zvykem, téhož výsledku dosáhneme zavoláním: `python3 optparse.mod.py -h`

**IV.** Tak to zkusme všechno najednou:

**Příkazová řádka:**

```
python.exe optparse.mod.py -x -c stylopis.css test.html
```

**Výstup:**

```
{'xhtml_flag': True, 'cssfile': 'stylopis.css'}
['test.html']
```

→ Přepínačem `-c` jsme změnili stylopisový soubor (klíč *cssfile*) z výchozího *style.css* na *stylopis.css* a přepínačem `-x` jsme změnili výchozí hodnotu klíče *xhtml_flag* z `False` na `True`. *test.html* je jméno souboru (určeného ke zpracování).

→ V „dlouhém“ provedení by volání vypadalo: `python.exe optparse.mod.py --xhtml --cssfile=stylopis.css test.html` (znak `=` tam sice není povinný, ale lépe se to s ním čte)

**V.** Aby se neřeklo, tak ještě obligátní dotaz na verzi programu :)

**Příkazová řádka:**

```
python.exe optparse.mod.py --version
```

**Výstup:**

```
optparse.mod.py 1.0
```

# Modul optparse – použití

Příklad na předchozím slajdu je upraven podle ukázky na http://www.saltycrane.com/blog/2009/09/python-optparse-example/ (originální dokumentace k modulu `optparse` je velmi „výživná“):

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


# import modulu pro práci s argumenty příkazové řádky
from optparse import OptionParser

# zavedení parseru vstupu
# ~ usage: nápověda pro volání skriptu
# ~ version: verze skriptu
parser = OptionParser(usage="usage: %prog [options] filename",
                      version="%prog 1.0")
# přidání přepínačů
# ~ jméno přepínače krátké a dlouhé
# ~ action: co provést v přítomnosti přepínače
# ~ dest: jméno klíče pro uložení hodnoty přepínače
# ~ default: výchozí hodnota pro přepínač
# ~ help: nápověda k přepínači
parser.add_option("-x", "--xhtml",
                  action="store_true",   # tj. nastav "dest" na "T
                  dest="xhtml_flag",
                  default=False,
                  help="create a XHTML template instead of HTML")
parser.add_option("-c", "--cssfile",
                  action="store",   # nepovinné, protože uložení z
                  dest="cssfile",
                  default="style.css",
                  help="CSS file to link",)
# zparsování vstupu
# ~ options: bude obsahovat „slovník“ přepínačů a jejich hodnot {
# ~ args: bude obsahovat seznam argumentů [ arg1, ... ]
(options, args) = parser.parse_args()


# na vstupu čekáme právě jeden argument (a to filename)
if len(args) != 1:
    parser.error("wrong number of arguments")


# tisk získaných vstupů
print( options )
print( args )
```

# Modul optparse – poznámky

Už na první pohled je modul `optparse` poměrně mocný. Pro podrobnosti se podívejte přímo do dokumentace, tady už jen několik poznámek:

- Přestože se návratová hodnota *options* z kódu `options, args = parser.parse_args()` tváří jako slovník, slovník to není :-( Naštěstí to spraví

jedno volání funkce `vars()`, která převádí svůj argument na slovník (alespoň pokud je to objekt vybavený atributem *__dict__*):

```
options, args = parser.parse_args()
options_dict = vars(options)
```

→ Použití `vars()` není úplně bez problémů, ale pokud pouze „vytáhnete“ hodnoty z parseru, nemělo by se nic zvláštního stát.

- Parametr *action* má k dispozici následující standardní operace: `store` (výchozí), `store_const`, `store_true`, `store_false`, `append`, `append_const`, `count`, `callback`, `help`.
- S výhodou můžete použít též parametr *type* – určuje typ argumentu, který příslušný přepínač očekává. Jeho výchozí hodnotou je (nepřekvapivě) "*string*" (uvozovky jsou důležité), dále můžete použít "*int*", "*float*", "*complex*" a "*choice*" (viz další bod). Nestačí-li vám to, můžete *optparse* celkem snadno rozšířit o další typy.
- Pokud má mít nějaký přepínač pouze jistou množinu povolených hodnot, použijte kombinaci parametrů *type*="*choice*" *choices*=*VÝBĚR*:

```
parser.add_option('-e', '--env',
                  action='store',
                  dest='environment',
                  type='choice',   # přepínač je výběrem z definov
                  choices=['production', 'staging', 'testing',],
                  default='production',
                  help='Environment to run on',)
```

→ Upraveno opět podle http://www.saltycrane.com/blog/2009/09/python-optparse-example/ .

- Přepínače můžete seskupit do více skupin pomocí `optparse.OptionGroup`:

```
# import potřebných konstruktorů
from optparse import OptionParser, OptionGroup

# zavedení vlastního parseru a (vizuálních) skupin přepínačů
parser = OptionParser(..)
group1 = OptionGroup(parser, 'NÁZEV', 'POPISEK')
..

# A) ke každé skupině se chováme stejně, jako dříve k samotnému pa
group1.add_option(..)
..
# B) na konci pak skupinu přepínačů přidáme jako celek do parseru
parser.add_option_group(group1)
```