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 – Obsluha externích procesů

Úvod

Podobně jako u argumentů skriptu, kde `sys.argv` poskytovalo přístup na té nejnižší úrovni a celá armáda postupně vznikajících modulů nabízela čím dál tím větší komfort, spouštění externích programů a zpracování jejich výstupu se dá také rozdělit na podobné dvě části:

- `os.system()` – „nejbrutálnější“, platformně závislý způsob spouštění cizích programů
- `subprocess` – vyvrcholení snahy o jednotný přístup a obsluhu externích procesů

Jelikož (nejen) pro spouštění externích procesů je potřebná znalost aktuální pozice v rámci souborového systému, podíváme se v této kapitole krátce i na ni.

os.system()

Tento příkaz má následující syntaxi `os.system(PŘÍKAZ)`, kde *PŘÍKAZ* je řetězec obsahující kód pro příkazovou řádku, který chceme zavolat. Následuje několik komentovaných ukázek typického použití:

I. Nejzákladnější použití představuje vyvolání nějakého systémového programu, např. *ls* pod UNIXy. Při dané adreářové struktuře dostaneme:

Vstup *ls.test.in* **:**

```
.
|-- ls.test.py
|-- something
|   `-- example.py
|-- something_else
`-- test
    `-- test.txt
```

Program *ls.test.py* **:**

```
import os
os.system("ls")
```

Výstup:

```
ls.test.py
something
something_else
test
```

→ Výstup programu směřuje na aktuální *sys.stdout*, čili nejspíš na terminál.

II. Ve své přímočarosti `os.system()` „kašle“ na jakékoli změny v prostředí (např. *sys.stdout* a další). Musíme se k němu opravdu chovat, jako kdybychom příslušný program spouštěli přímo na příkazové řádce:

```
os.system( "ls >listing.txt" )
```

Praktičtější příklad z praxe:

```
os.system( "saxon -o example.html example.xml example.xslt" )
```

Základní operace s cestou

Spouštění globálně dostupných programů je možné pouze podle jejich jména, pro neglobální však už nikoli. Podobně vyrobí-li volaný program nějaký výstup, resp. musíme-li mu nějaký vstup ze souborového systému naopak poslat, je znalost, jak se k uvedenému souboru dostat, nezbytná. Následuje výběr nejdůležitějších operací pro práci se souborovým systémem:

I. Operace zjištění a změny aktuální cesty:

- `os.getcwd()` – vrátí řetězec reprezentující absolutní cestu k aktuálnímu adresáři
- `os.chdir(CESTA)` – změní aktuální pracovní adresář na umístění *CESTA*; to může být zadané jak relativně, tak absolutně

  → Cesty je třeba zadávat ve tvaru použitelném na všech platformách, viz následující bod.

II. Cesty je důležité získávat a zadávat ve tvaru, který bude fungovat na různých platformách. Pro to je k dispozici mnoho metod v modulu `os.path`, z nichž mezi nejdůležitější patří:

- `os.path.normpath(CESTA)` – vrátí cestu v normalizovaném tvaru vhodném pro zadávání do jiných metod; na Windows to ještě nemusí stačit a může být

třeba použít `os.path.normcase(CESTA)` (týká se zvláště velikosti písmene svazku např. při porovnávání cest)

- `os.path.abspath(CESTA)` – jako předchozí, ale vrací absolutní cestu
- `os.path.join( path1[, path2[, ...]]) )` – vyrobí výslednou cestu z uvedených částí (narozdíl od „ručního“ lepení řetězců správně pro každou platformu)
- `os.path.split(CESTA)` – podle možností rozdělí cestu na *head* a *tail*, přičemž *tail* nikdy neobsahuje / (typicky bude *tail* soubor a *head* cesta k němu); viz též `os.splitext(path)`
- `os.path.exists(CESTA)` – vrací *True*, existuje-li (nebo je přístupná) daná cesta
- `os.path.isabs(path)`, `os.path.isfile()`, `os.path.isdir(path)`, `os.path.islink(path)`, `os.path.ismount(path)` – vrací *True*, je-li uvedená cesta absolutní, resp. soubor, resp. adresář, resp. symbolický odkaz (*symlink*), resp. *mount point*

III. Adresáře a soubory je přitom samozřejmě možné i vypisovat, vyrábět, mazat apod.:

- `os.listdir(CESTA)` – vrátí seznam jmen souborů (tj. i adresářů) na uvedené cestě
- `os.mkdir(CESTA[, mode])`, `os.makedirs(CESTA[, mode])` – vyrobí adresář (v uvedeném módu, je-li aplikovatelný) na uvedené cestě; druhá varianta doplní i případné neexistující nadřazené adresáře
- `os.remove(CESTA_K_SOUBORU)`, `os.rmdir(CESTA_K_ADRESÁŘI)`, `os.removedirs(CESTA_K_ADRESÁŘI)` – pokusí se na uvedené cestě odstranit soubor, resp. adresář, resp. adresář včetně nadřazených; pro specifičtější práci viz též `shutils.rmtree()`
- `os.rename(src, dst)`, `os.renames(old, new)` – pokusí se přejmenovat uvedenou na cestu na novou

Pro práci s dočasnými cestami je k dispozici celý vlastní modul `tempfile`.

subprocess I

Modul *subprocess* je dnes preferovaný (a podstatně bezpečnější) způsob práce s externími programy. Definuje jednu základní třídu..

```
subprocess.Popen(args, bufsize=0, executable=None, stdin=None, stdout=None, stderr=None,
                 preexec_fn=None, close_fds=False, shell=False, cwd=None, env=None,
                 universal_newlines=False, startupinfo=None, creationflags=0)
```

..plus několik užitečných konstant (`subprocess.PIPE`, `subprocess.STDOUT`) a přehršel pomocných funkcí.

→ Dostupnost různých metod se liší platformu od platformy.
→ Některé z metod mohou vyvolat *deadlock*.

I. Nejjednodušší způsob použití představuje přímé zavolání externího programu bez jakýchkoli parametrů..

```
import subprocess

# zavolej příkaz 'ls'
subprocess.Popen("ls")
```

..případně s parametry:

```
import subprocess

# zavolej příkaz 'ls -al'
args = ['ls', '-al']
subprocess.Popen(args)
```

II. Jelikož výstupy předchozích programů jdou ve výchozím nastavení přímo na standardní výstup, nejsou jednoduše dále použitelné v rámci našeho programu. Můžeme si je ale snadno přesměrovat, například do souboru:

```
import subprocess

with open('process.B.out1', 'w') as f:
    p = subprocess.Popen("ls", stdout=f)

args = ['ls', '-al']
with open('process.B.out2', 'w') as f:
    p = subprocess.Popen(args, stdout=f)
```

subprocess II

Volání externích procesů je ale ještě flexibilnější. S výhodou můžeme použít zvláště následující:

- Parametry `stdin`, `stdout` a `stderr`, které slouží k přesměrování vstupu, výstupu a chybového výstupu volaného procesu.
- Parametr `cwd` slouží ke změně adresáře, v němž dojde k zavolání externího procesu.

  Cesta k procesu nemůže být zadána relativně vůči tomuto adresáři.

- Parametr `env` slouží k modifikaci proměnných prostředí pro potřeby spouštěného externího procesu.
- Obsluhu *standardního proudu* `subprocess.PIPE` zajišťuje metoda `communicate([input=None])`. Jejím úkolem je posílat data do *stdin* (v tom případě vyžaduje nastavení `stdin=subprocess.PIPE`) a číst je ze *stdout* a *stderr* (čeká přitom na ukončení volaného procesu).

  > Její nepovinný parametr je ve výchozím nastavení `None` (tj. žádná vstupní data), jinak to musí být bajtový řetězec. (Což je v ostrém kontrastu s Pythonem 2.x – tam to byl řetězec. Změna je to ale pochopitelná, protože Python 3.x narozdíl od Python'u 2.x striktně rozlišuje mezi binárními 8-bitovými daty a řetězci.)

Několik příkladů:

I. Vyvolání externího programu se vstupem ze standardního proudu a výstupem do souboru:

Program *process.1.py* **:**

```
import subprocess

outfile = open("process.1.out", "w", encoding="ascii")
p = subprocess.Popen("sort", stdin=subprocess.PIPE, stdout=outfil
p.communicate( b"hello\nhow\nare\nyou" )
outfile.close()
```

Výstup:

```
are
hello
how
you
```

→ Voláme tedy externí program *sort*.

→ Vstup je pomocí parametru *stdin=subprocess.PIPE* přesměrován ze standardního proudu, je jím tedy text *hello\nhow\nare\nyou*.

→ Výstup je směřován do souboru *out.txt*.

II. Složitější varianta předchozího – standardní i chybový výstup přesměrovány také na standardní proud:

Program *process.2.py* **:**

```
import subprocess

p = subprocess.Popen("sort", stdin=subprocess.PIPE,
                             stdout=subprocess.PIPE,
                             stderr=subprocess.STDOUT)
print( "handle:", p )
output, error = p.communicate( b"hello\nhow\nare\nyou" )
print( "output:", output )
print( "error:", error )
```

Výstup:

```
handle: <subprocess.Popen object at 0x00C06030>
output: b'are\nhello\nhow\nyou\n'
error: None
```

→ Při použití speciální hodnoty *stderr=subprocess.STDOUT* bude chybový výstup přesměrován na stejné „zařízení“ jako *stdout*.