Programátorská Encyklopedie

KORESPONDENČNÍHO SEMINÁŘE Z PROGRAMOVÁNÍ

Parsování vstupu v Pythonu 3

Začínající programátoři se velmi často potýkají s tím, jak by měli do svého programu dostat vstup a jak ho zpracovat. V tomto článku a v sesterském článku o jazyce C[1] se vám s tím pokusíme pomoci. Úvodní kapitolu mají oba články stejnou, ale poté se již každý zabývá svým jazykem.

Klávesnice versus soubory

Nejjednodušší je ve většině jazyků načítat vstup od uživatele. To je dobré pro rychlé testování nebo pro programy přijímající jenom jednu vstupní hodnotu. Ale co když program přijímá na vstupu více hodnot?

Jedna možnost je změnit jeho vnitřnosti tak, aby místo toho načítal vstup ze souboru. Ale nejrychlejší většinou bývá zachovat program tak, jak je, a jen mu místo vstupu z klávesnice předhodit soubor.

Programu totiž většinou neví o klávesnici nic, má jen věc, které se říká *standardní vstup*. Ten je většinou představovaný klávesnicí (obdobně je pak *standardní výstup* většinou představovaný vypisováním na terminál), ale dá se jednoduše přesměrovat.

Jak na to? Představme si, že máme nějaký svůj program (spustitelný zkompilovaný program v C nebo skript v Pythonu) a soubor `1.in`, který mu chceme předhodit na vstupu. V příkazové řádce všech hlavních operačních systémů (Linux a obecně všechny UNIXy, Windows) to uděláme pomocí operátoru přesměrování ve tvaru menšítka (<):

```
./program < 1.in           -- UNIXový svět
program.exe < 1.in         -- Windows
python3 program.py < 1.in  -- oboje
```

(Pozor: V UNIXovém světě se při spuštění programu z aktuálního adresáře musí explicitně uvést adresář tečka jako odkaz na aktuální umístění.)

Stejným způsobem pak lze přesměrovat i výstup, jen s použitím šipky na druhou stranu (většítka). Pokdu tedy budete řešit třeba opendatové úlohy v KSP a budete mít program v Pythonu 3, může celý příkaz vypadat jako:

```
python3 program.py < 1.in > 1.out
```

Zpracování standardního vstupu

Úvodem varujeme, že se při načítání vstupu liší chování Pythonu 2 a Pythonu 3. Hlavně v tom, že ve starší verzi se funkce `input()` pokoušela vyhodnotit vstup jako Pythoní výraz, kdežto v nové verzi se vstup načte jednoduše jen jako string (to ve starší verzi šlo pomocí `raw_input()`) a zpracování je pak plně v naší režii.

Již zmíněná funkce `input()` načte celý řádek jako string. Pokud například vstup obsahuje dvě čísla, každé na novém řádku, můžeme je načíst jako stringy a převést na číselnou hodnotu takto:

```
a = int(input())
b = int(input())
```

O trochu složitější situace nastává, když jsou čísla na stejném řádku oddělená jen mezerou. V tuto chvíli se nám velmi hodí funkce `.split()`, která rozdělí řetězec podle bílých znaků (nebo podle jiného oddělovače, pokud jí ho předáme jako parametr) a vrátí ho jako seznam stringů. Příklad výše by se tedy dal převést na:

```
pole = input().split()
a = int(pole[0])
b = int(pole[1])
```

Všimněte si, že tady musíme dělat trochu neohrabanou konstrukci kvůli tomu, že `.split()` nám vrátí seznam stringů. U dvou čísel to ještě nevadí, ale představte si to třeba pro vstup o délce deset nebo třeba tisíc prvků. Dalo by se to vyřešit nějakým cyklem, ale existuje jedna velmi pěkná syntaktická zkratka.

Tou je velmi mocná funkce `map()`, která v jednoduchosti dovoluje spustit zvolenou funkci na všechny prvky seznamu. Nevrací ale bohužel čistý seznam, což ale jednoduše můžeme vyřešit ještě zabalením do převodní funkce `list()`. Celá konstrukce pak může vypadat takto:

```
pole = list(map(int, input().split()))
```

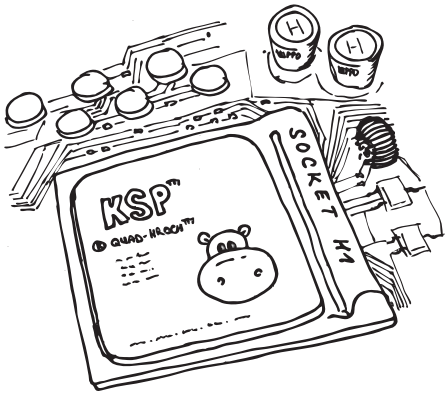
Hodí se pro zpracování rozumně velkých řádků (řekněme v jednotkách až desítkách megabajtů), pro větší řádky raději zvolte přístup přes skutečnou práci se soubory. Stejně tak pokud předem nevíte, kolik chcete načíst řádek a chcete skončit až ve chvíli, kdy dojdete na konec vstupu, je lepší použít souborové funkce.

Vypisování

Náplní článku je sice hlavně načítání vstupů, ale zmíníme se krátce i o vypisování. Základem všeho je v Pythonu 3 funkce `print()` (jako varování dodáme, že v Pythonu 2 to nebyla funkce, ale příkaz, a ještě k tomu s odlišnou syntaxí).

Této funkci je možné předat skoro jakýkoliv objekt Pythonu a ona ho vypíše a odřádkuje. Pokud ji předáme více parametrů, vypíše je všechny oddělené jednou mezerou a teprve poté odřádkuje:

```
print(1)    # vypíše: 1
a = 7.6
print(a, 1) # vypíše: 7.6 1
```



[1] `http://ksp.mff.cuni.cz/encyklopedie/parsovani-vstupu-c.html`

Ale počkat, co když chci vypisovat bez odřádkování, třeba když mám nějaké pole a chci ho v cyklu vypsat? Na to existuje řešení, funkci `print()` se totiž dá předefinovat koncový znak (defaultně odřádkování) i oddělovač (defaultně mezera) a to pomocí pojmenovaných parametrů.

Následující konstrukce vypíše každý prvek z pole a jeho dvojnásobek oddělené dvojtečkou a tyto dvojice oddělené tabulátorem:

```
for prvek in seznam:
        print(prvek, prvek*2, sep=":", end="\t")
```

Příkaz výše má jediný problém a to, že i po poslední dvojici se vypíše tabulátor. To se sice dá obejít nějakými podmínkami, ale existuje elegantnější varianta pomocí funkce `.join()`. Ta dostane jako parametr seznam stringů, které má spojit, a spojí je pomocí stringu, na kterém je volána. Nejlépe to ukáže příklad:

```
seznam = ["1", "2", "3"]
s = ":".join(seznam)
# Vrátí "1:2:3"
```

Ale co když budeme chtít `.join()` předat seznam čísel? Pak opět přichází na scénu naše známá funkce `map()`, kterou všechna čísla převedeme na stringy:

```
seznam = [1, 2, 3]
s = ":".join(map(str, seznam))
# Vrátí "1:2:3"
```

Čtení ze souborů

Python má pro přístup k souborům odlišnou syntaxi, než pro standardní vstup, ale spousta principů zmíněných výše se dá aplikovat i zde. Nejdříve je nutné soubor otevřít voláním funkce `open()`, která nám vrátí odkaz na soubor. Po skončení práce se souborem ho zavřeme voláním `.close()` na něm samotném.

Té předáme jako první parametr cestu k souboru a jako druhý parametr volitelně mód otevření (bez uvedení otevře soubor ke čtení). Parametr `r` otevírá jen pro čtení, parametr `w` pak pro zápis a parametr `a` pro přidávání na konec.

```
vstup = open("1.in", "r")
# nějaká práce se souborem
vstup.close()
```

Další příkazy se ale volají již přímo na objektu `vstup` (zde se pěkně projevuje objektový přístup Pythonu). Dá se voláním `.readline()` načíst celá jedna řádka ze souboru (a dále s ní pracovat jako při použití `input()`), nebo se dá pomocí `.read()` načítat vstup po znacích:

```
vstup = open("1.in", "r")
slova = vstup.readline().split()
pismeno = vstup.read()
vstup.close()
```

Při volání `.read(10)` si ještě číslem můžeme říct, že chceme načíst více bytů najednou. Při vypisování pak můžete použít funkci `.write()`. Ta ale narozdíl od `print()` nepřidává na konec automaticky znak nového řádku a je potřeba ho přidat ručně, pokud ho chceme:

```
vystup = open("1.out", "w")
vystup.write("bagr\n")
vystup.close()
```

Dodejme ještě, že příkazy pro práci se soubory lze použít i pro práci se standardním vstupem a výstupem. Stačí importovat modul `sys` a pak je možné místo objektu reprezentujícího vstup a výstup použít `sys.stdin` a `sys.stdout`:

```
import sys;
radek = sys.stdin.readline()
sys.stdout.write("bagr\n")
```

Pěkná syntaktická perlička, kterou vám závěrem ukážeme, je jak jednoduše zpracovat v Pythonu celý soubor, dokud nedojdete na jeho konec:

```
vstup = open("1.in", "r")
for radek in vstup:
    # dělej něco
vstup.close()
```

Doufáme, že pro vás návod byl alespoň trochu užitečný a že i díky němu porozumíte Pythonu o trochu více.

Článek pro vás sepsal
Jirka Setnička