

Návod na programování v jazyce **Python** (Základy astronomie)

Martin Piecka

4. října 2020

1 Úvod

Základními úlohami, které musí astrofyzik typicky řešit jsou modelování a zpracování dat. V dnešní době se většinu doby nezaobejdete bez ovládnání alespoň jednoho programovacího jazyka, ve kterém by jste měli být schopni řešit zadané problémy. Je velice důležité, aby se student astrofyziky setkal s programováním už v prvním semestru. Avšak vzhledem k tomu, že programování není snadné a vyžaduje mnoho času na učení, rozhodli jsme se studentům poskytnout návod, ve kterém jsou uvedené příklady řešení vybraných problémů (studenti by si je měli vyskoušet vyřešit i sami!). Jednotlivé kódy by pak měli sloužit jako ukázka mnoha příkazů, které studentovi mohou značně usnadnit život počas studia.

Cílem dokumentu není detailní rozprava o programování, ale první setkání s automatickým řešením problémů – tohle je kurz astrofyziky, ne informatiky. Po vyřešení uvedených příkladů by student měl být schopen rychle vyřešit mnohé části praktických úloh v kurzu Základy astronomie, kde je použití programovacího jazyka velikou výhodou.

2 Instalace

2.1 Základní instalace

Pro začátek si budete muset nainstalovat Python. Pro stáhnutí základního balíčku poslouží uvedené odkazy (podle operačního systému):

- Windows ... <https://www.python.org/downloads/windows/>
- Linux ... <https://www.python.org/downloads/source/> (pokud není již nainstalován)

Je doporučeno stáhnout nejnovější verzi (v době psaní tohoto dokumentu je to verze 3.8.5). Při instalaci je důležité povolit i instalaci modulu pip (pod Customize installation), aby jste to nemuseli pak dělat sami (!!). Nezapomeňte taky povolit propojení cesty PATH (dole na uvítací obrazovce instalace), usnadní to práci při instalaci modulů. Zapište si taky cestu instalace nebo ji změňte na něco jednoduchého (např. ve Windows C:/Python38/). Po skončení instalace si musíme otevřít příkazový řádek:

- Windows: WIN + S, do vyhledávání napište *cmd* a stlačte **Enter**
- Linux: otevřte terminál (Ubuntu 20.04, CTRL + ALT + T)

Jestli jste nezapoměli nainstalovat i pip, bude instalace dalších knižnic funkcí velice jednoduchá. Nejprve si nainstalujte numpy, která umožňuje práci s poli proměnných (vektory, matice, atd.):

- Windows: do příkazového řádku napište `pip install numpy`
- Linux: do terminálu napište `pip3 install numpy`

Další důležitá knižnice, kterou budete potřebovat je matplotlib, která vám umožní vykreslit data do grafů. Táhle knižnice však vyžaduje další knižnice, které ještě nemáme nainstalovány. Tady nám usnadní práci, když použijeme příkaz:

- Windows: `pip install --pre matplotlib`
- Linux: `pip3 install --pre matplotlib`

který nainstaluje i prerekvizice. Tímto by jste měli mít Python připraven k používání. Spustit ho můžete přes IDLE poskytnuto instalačním balíčkem nebo spustit přímo přes příkazový řádek (pro Windows uživatele `cesta/Python38/python.exe`; pro Linux typicky stačí použít příkaz `python3`). Instalaci si ověřte příkazem v Python-u

```
print('Hello world!')
```

a příkaz potvrďte stlačením **Enter**. Jestli všechno běží správně, na dalším řádku by se měl objevit text “Hello world!”.

2.2 Jupyter

Alternativou k instalaci Python-u je využití prostředí **Jupyter**, které je zadarmo k dispozici online na odkazu <https://jupyter.org/try>. Tam si otevřete [Try Classic Notebook] a počkejte na načtení (může chvíli trvat). Když se vám ukáže obrazovka s textem “Welcome to Jupyter!”, klikněte na **File > New Notebook > Python 3**, mělo by vás to odkázat na konzoly Python-u, ve které již můžete pracovat (knihovny `numpy` a `matplotlib` jsou hned k dispozici).

Pro načítavání datových souborů budete muset nejprv uploadovat váš soubor. To uděláte přes **File > Open**, tam si zvolte napravo [Upload] (hned vedle [New]). Vyberte soubor, který chcete uploadovat, pak načtení potvrďte stlačením [Upload] v řádku s názvem souboru, který jste si vybrali.

Nevýhodou je, že se vám může stát, že se nebudete moci připojit a stránka vám vyhodí chybovou hlášku již před “Welcome to Jupyter!”. V tom případě budete muset počkat a pokusit se připojit pozděj (protože byl dosažen limit pro počet dostupných uživatelů, kteří se mohou zadarmo připojit).

2.3 Vývojové prostředí

Pokud jste si stáhli Python jak bylo uvedeno výše, měli by jste mít k dispozici i jedno vývojové prostředí, IDLE, ve kterém můžete snadno psat své programy. Existují však i jiné prostředí, kterých instalace může být o něco snadnější (např. `Enthought Canopy` si nainstaluje i vlastní Python, v době psaní dokumentu však už není oficiálně k dispozici). Zde jsou uvedeny příklady dalších volně dostupných vývojových prostředí (IDE):

- Visual Studio Code (umožňuje psaní i v jiných jazycích)
- PyCharm (community verze je zdarma)
- Spyder
- a další ...

2.4 Počítačova učebna

V nejhorším případě, že nebudete schopni provést instalaci sami a nebude vám fungovat Jupyter, na přízemí Ústavu teoretické fyziky a astrofyziky naleznete počítačovou učebnu (může byt nutno si nejprve vybavit přístup na ISIC kartu). Jestli je učebna volná, počítače v učebni mají Python nainstalován, takže můžete pracovat tam.

3 Příklady

Ted' si uvedeme několik příkladů, které by měli stačit pro pochopení toho, jak se staví kódy programů, se kterými se v Základech astronomie setkáme. Každý příklad uvede nějaký příkaz (funkci), který si vyzkoušíme a tak zjistíme, jak funguje a k čemu slouží.

3.1 Datové typy, načítání a výpis hodnot

Začneme jednoduchými příkazy, kterými můžeme načíst hodnoty do proměnných a pak je vypsát. Samozřejmě, závisí na tom, jestli chceme načíst data od uživatele nebo ze souboru (načtení ze souborů je uvedeno jako poslední příklad).

V první úloze chceme zjistit, kolik studentů je v současném semestru na oboru Astrofyzika. Program bude vypadat následovně (řádky s `>>` představují kód programu, zbytek jsou řádky výpisu v konzole):

```
01: >> input('Pocet studentu: ' )
02: Pocet studentu: 10
03: '10'
```

Příkaz na řádce 01 (`input()`, pro Python verzi 2.7 je nutno však nahradit za `raw_input()` v závorkách načte text, který se vypíše na dalším řádku (02) a hned za ním bude počítač čekat na zadání hodnoty. Momentálně zde můžeme načíst cokoli (i text), ale později si budeme muset dávat větší pozor. Pro zkoušku jsme napsali hodnotu 10 a potvrdili stisknutím **Enter**. Funkce `input()` načtenou hodnotu rovnou vypíše na řádku 03 – apostrofy kolem hodnoty znamenají, že hodnota je uložena jako řetězec znaků, tedy text.

Jestli chceme, aby výsledná hodnota funkce `input()` byla číslo, budeme muset upravit program. Nejprve si však uvedeme některé datové typy, se kterými se setkáte prakticky vždy. Rozdíl mezi datovými typy spočívá také v tom, kolik paměti počítače zabírají.

- **integer** (celé číslo), používáme často v cyklech a jako indexy vektorů a matic; reálná čísla a řetězce textu obsahující pouze jedno číslo můžeme změnit na celé číslo pomocí funkce `int()`
- **float** (reálné číslo), představuje základní typ hodnot, který budeme používat pro fyzikální veličiny; celá čísla a řetězce textu obsahující pouze jedno číslo můžeme konvertovat na reálné číslo funkcí `float()`
- **string** (textový řetězec), slouží nejen k výpisu textu, ale může představovat základní typ hodnoty, kterou načítáme ze souborů; celá a reálná čísla můžeme změnit na text kdykoliv pomocí `str()`
- **boolean** (logická hodnota), nabývá hodnoty `True` nebo `False` (počáteční písmeno musí být velké), představuje základ pro podmínky (v podstatě základ programování); funkce `bool()` dokáže sebere do vstupu prakticky jakoukoliv hodnotu s tím, že vrací `True` vždy kromě nulové hodnoty daného typu proměnné (text `''`, celé a reálné číslo `0`, prázdné pole hodnot `[]`)

Náš předchozí kód upravíme tak, abychom jsme řetězec na výstupu funkce `input()` zkonvertovali na celé číslo

```
01: >> int(input('Pocet studentu: ' ))
02: Pocet studentu: 10
03: 10
```

Stojí za povšimnutí, že počítač vykonává příkazy v jistém pořadí – respektuje závorky (nejprve proběhne co je uvnitř, pak co je venku) a postupuje řádek za řádkem.

Jestli si načteme hodnotu do proměnné (více bude v dalším příkladu) `x`, můžeme pak tyto hodnoty vypsat pomocí příkazu `print()`

```
01: >> x = int(input('Pocet studentu: ' ))
02: Pocet studentu: 10
03: >> print(x)
04: 10
```

Při načítávání do proměnné se nám hodnota automaticky nevypíše!

3.2 Proměnné a operace

Většinu doby si budeme chtít ukládat hodnoty, se kterými pracujeme. Pravděpodobně s nimi budeme také chtít vykonávat nějaké operace (násobení čísel, nebo spojování textů).

Rekněme, že na začátku semestru studovalo astrofyziku na fakultě `n`-studentů, ale během semestru `m`-studentů svoje studium přerušilo. Budeme chtít vědět, kolik studentů bylo na konci semestru.

```
01: >> n = int(input('Pocet studentu na zacatku: ' ))
02: Pocet studentu na zacatku: 10
03: >> m = int(input('Pocet studentu, kteri prerusili: ' ))
04: Pocet studentu, kteri prerusili: 2
05: >> print('Pocet studentu na konci: ' + str(n-m))
06: 8
```

Využili jsme zde několik operací s proměnnými. Na řádcích 01, 02 jsme si načetli počáteční počet studentů, na řádcích 03, 04 jsme načetli úbytek studentů. Příkaz `print()` na 05 vypíše něco na další řádek. Podívejme se blíže, co jsme tam napsali:

- `'Pocet studentu na konci: ' + ...` první část je očividně textový řetězec. Co však představuje znaménko součtu? Musíme si uvědomit, že nikdy nemůžeme provést součet (nebo téměř jakoukoliv jinou operaci) dvou odlišných typů proměnných. Pro dva řetězce představuje operace (+) spájení dvou částí textu.
- `+ str(n-m)` ... jak jsme zmínili výše, text můžeme sečíst pouze s dalším textem, proto zde vystupuje funkce `str()`. Do ní vkládáme rozdíl mezi počátečním počtem a úbytkem, abychom jsme dostali počet studentů na konci semestru, operace (-) tady představuje odečítání dvou celých čísel.

Samozřejmě, hodnoty si do proměnných můžeme načíst taky přímo v kódu:

```

01: >> a1 = 5
02: >> a2 = 2
03: >> b1 = 5.0
04: >> b2 = 2.0
05: >> s = 'hello world'
06: >> print(a1/a2)
07: 1.5
08: >> print(b1/b2)
09: 1.5
10: >> print(s)
11: 'hello world'

```

Pozor na to, že výpis na řádce 07 se může lišit v závislosti na prostředí, ve kterém programujete. Python 3.8 v příkazovém řádku Windows tohle interpretuje jako reálné číslo, ale může se taky stát, že tenhle příkaz vrátí celé číslo (výsledek po dělení bez desetinné části, což je `int()` hodnoty, kterou zde vidíme).

Vyzkoušíme si ještě několik zaokrouhlovacích funkcí pro reálná čísla. Patří mezi ně příkazy `round()`, `ceil()` a `floor()`. Druhá a třetí funkce jsou snadné – číslo, které jim poskytneme zaokrouhlí nahoru (`ceil()`) nebo dolů (`floor()`). První funkce funguje tak, jak jsme při zaokrouhlování zvyklí, ale umožňuje přidat ještě jeden parametr, kterým ovládáme počet desetinných míst:

```

01: from numpy import *
02: >> a = 2.0
03: >> print(a**2)
04: 4.0
05: >> print(sqrt(a))
06: 1.4142135623730951
07: >> print( ceil( sqrt(a) ) )
08: 2.0
09: >> print( floor( sqrt(a) ) )
10: 1.0
11: >> print( round( sqrt(a) ) )
12: 1
13: >> print( round( sqrt(a) , 0 ) )
14: 1.0
15: >> print( round( sqrt(a) , 3 ) )
16: 1.414

```

Poprvé využijeme volání další knihovny, která není automaticky do Python-u vbudována – na řádce 01 si načteme všechny funkce z knihovny `numpy` (poslouží na odmocňování a umocňování čísel). Na řádce 13 zaokrouhlujeme na celé číslo (ale ponecháme ho ve formě reálného čísla), na řádce 15 pak zaokrouhlíme číslo na třetí desetinné místo. Měli byste si všimnout, že různé funkce vyžadují různý počet vstupních hodnot. Příkaz `input()` v podstatě nevyžaduje žádný vstup. Zaokrouhlování pomocí `round()` vyžaduje jednu hodnotu, ale funkci můžeme ovládat přidáním dalších vstupních parametrů.

Přehled základních funkcí v Python-u (bez použití nějaké z knižnic) najdete na <https://docs.python.org/3/library/functions.html>.

3.3 Podmínky

Někdy potřebujeme zkontrolovat, jestli jsou čísla stejná nebo odlišná. Na tohle všechno (a více) nám slouží podmínky.

Máme proměnnou `x = 12.0` a chceme zjistit, jestli načtená hodnota `a` je menší než `x` – jestli ano, hodnotu vypíšeme; jestli ne, program musí vyhodit nějakou chybovou hlášku.

```
01: >> x = 12.0
02: >> a = int(input())
03: 6.0
04: >> if (x>a):
05: ..     print(a)
06: .. else:
07: ..     print('Spatna vstupni hodnota')
08: ..
09: 6.0
```

Podívejme se krok za krokem, co se v tomto programu stalo. Na 01 jsme zadali do proměnné `x` hodnotu 12.0 a na dalších dvou řádcích jsme do proměnné `a` načetli číslo 6.0. Podmínku začínáme na 04 příkazem `if`, který bude pokračovat pouze v případě, že následující hodnota je `True`. Dvojtečka uvádí sérii řádků (v tomto příkladu pouze jeden, 05), které budou tělem tohoto `if` a budeme je muset posunout mezerou (doporučeno použít fixní počet mezer, např. 2, nebo klávesu `Tab`). Jestliže chceme opustit tělo podmínky `if`, nepoužijeme stanovený počet mezer. Následně můžeme (ale v podstatě nemusíme) do programu vložit druhou část podmínky, a to je `else`, která proběhne v případě, že `if` nad ní vrátil `False`. Na 07 je uveden příklad chybové hlášky, kterou program vypíše, když není uvedena podmínka splněna. Na řádek 08 nepíšeme nic, jenom potvrdíme stiskem `Enter`, že jsme podmínku uzavřeli a program jí vyhodnotí (výpis na řádku 09).

Je doporučeno si vyzkoušet naprogramovat různé verze podmínky `if`, aby se student seznámil s tím, jak funguje. Podmínky se v kódu používají velice často.

3.4 Pole proměnných

Důležitým typem proměnné v jazyce `Python` je pole proměnných (existuje několik typů, používat zde však budeme pouze `list`). Pole představuje soubor stejných nebo i různých datových typů, které jsme jmenovali v předešlé části textu (např. vektor reálních čísel). Slouží na efektivní ukládání hodnot a na práci s nimi – hodnoty pole můžeme dostat pomocí indexů, které začínají od nuly (první hodnota bude mít index 0).

Mějme zadaný pole hodnot `x = [2,0,7]`. Naším úkolem je tohle pole vypsát, vypsát počet hodnot v poli a zjistit hodnotu na posledním indexu.

```
01: >> x = [2,0,7]
02: >> print(x)
03: [2, 0, 7]
```

```

04: >> print(len(x))
05: 3
06: >> print(x[len(x)-1])
07: 7

```

Co se tady stalo? Příkaz `len()` na 04 vrací počet hodnotu v poli `x`. Hodnotu na nějakém indexu pak dostaneme jako `x[index]`, kde `index` musí být celé číslo. Na řádcích 06 a 07 pak vypíšeme hodnotu na posledním indexu – ta je daná počtem hodnot v poly, ale protože indexování začíná od nuly, musíme od `len(x)` odečíst `-1` (`x[2]` představuje hodnotu na třetím místě v poly).

Vyzkoušejme si, jak budeme postupovat při práci s maticemi (poslední příklad se týkal vektoru). Mějme pole `A` s řádky `[1,0]`, `[0,1]` a `[2,3]`. Chceme vypsat rozměry matice, druhý řádek a druhou hodnotu na prvním řádku.

```

01: >> A = []
02: >> A.append([1,0])
03: >> A.append([0,1])
04: >> A.append([2,3])
05: >> print(len(A))
06: 3
07: >> print(len(A[0]))
08: 2
09: >> print(A[1])
10: [0, 1]
11: >> print(A[0][1])
12: 0

```

Řádky 01 až 04 je možno nahradit jediným, `A = [[1,0], [0,1], [2,3]]`. Je však lepší, když si vyzkoušíte funkci `append()`, která na konec pole vsune další proměnnou. V tomhle případě používáme tři pole (představují řádky) o velikosti dva (index řádku představuje sloupec). Příkaz na 05 vypíše počet elementů, což je počet řádků. Jestli chceme dostat počet sloupců, musíme vypsat počet elementů jednoho řádku (všechny řádky jsou v tomhle případě stejné velikosti, takže je jedno, který použijeme). Příkaz `A[řádek]` (09) vypíše daný řádek, `A[sloupec]` (09) konkrétní hodnotu matice.

Nezapomeňte! Indexování v `Python`-u začíná na nule, proto (např.) druhý řádek matice vypíšeme pomocí indexu o jedno menšího jako je pořadí, tedy `A[1]`. Na závěr si vyzkoušejte pro první příklad s vektorem vypsat výsledek příkazu `x + x` – co představuje operace (+) pro pole typu `list`?

3.5 Cykly

Cykly v programovacích jazycích představují jednoduchou cestu jak projít sérii čísel nebo analyzovat všechny prvky pole. V posledním příkladu jsme si skusili vypsat jeden prvek matice, ale co když by matice měla velké dimenze? Psát explicitně řádek po řádku výpis jednotlivých elementů matice je nepraktické – tenhle postup nahrazuje cyklus `for`.

Vytvořte pole hodnot, které bude na prvním sloupci obsahovat čísla `[0 .. 20]` a na druhém sloupci hodnoty jejich druhých mocnin. Řádky tohoto pole na výstupu vypíšte.

```

01: >> from numpy import *
02: >> x = []
03: >> for i in range(21):
04: ..     x.append([i, i**2])
05: ..
06: >> for i in x:
07: ..     print(i)
08: ..
09: [0, 0]
    [1, 1]
    [2, 4]
    ...
    [18, 324]
    [19, 361]
    [20, 400]

```

Tady již musíme použít funkci `append()`, proto si nejprve vytvoříme prázdné pole (02). Cyklus `for` použijeme tak, že nejprve určíme název proměnné, kterou budeme iterovat (tady je to `i`) a za “in” vložíme pole hodnot, kterými bude proměnná `i` procházet. V tomhle případě jsme zvolili `range(21)`, co představuje pole 21 celých čísel, počínajíc od nula (tedy `[0..20]`). Proměnná `i` bude postupně nabývat daných hodnot a budou vykonávány příkazy v těle cyklu (podobně jak u podmínek, tělo cyklu vyžaduje řádky posunuté o mezeru). V podstatě by jsme v tomhle cyklu mohli rovno vypisovat řádky matice, ale vyzkoušíme si to v dalším cyklu, aby jsme získali cit pro to, jak funguje cyklus `for`. Na 06 budeme opět iterovat proměnnou `i`, tentokrát však přes pole `x` – iterující proměnná bude mít podobu řádků matice a proto ji můžeme rovno vypisovat (07).

3.6 Statistické zpracování

Počas vašho studia budete potřebovat často zpracovávat data. K tomu potřebujeme být schopni vypočítat základní statistické veličiny. Ty můžeme dostat z knihovny `numpy` a jsou to například:

- `mean()` ... vypočítá střední hodnotu pole
- `median()` ... vypočítá medián hodnot pole
- `std()` ... vypočítá standartní odchýlku hodnot pole

Bylo by vhodné si tyto příkazy vyzkoušet. Určete průměrnou hodnotu a standartní odchýlku následujících vektorů (nebo si vytvořte i vlastní!)

- `x = [0, 1, 2, 3, 4]`
- `y = [8.06, 4.12, 4.17, -2.42, 2.48, 5.66, 7.40, 4.96, 6.65, 8.25]`
- `z = [0, 1, 1, 1, 2, 3, 5]`

Promyslete a ověřte – je pro určení nejtypičtější hodnoty v `z` lepší použít `mean()` nebo `median()`?

3.7 Vykreslení dat

Součástí analýzy dat je taky vykreslení grafu závislosti mezi měřenými veličinami. Zde zjistíme, jak postupovat v jazyce Python, když si chceme vykreslit graf a nějak ho upravit (přidat popisky, změnit měřítko osy, ...).

Vykreslete si postupně (ne do jednoho grafu) vzhled funkcí $\sin(x)$, $\cos(x)$, $\log_{10}(x)$ a $\exp(-(x-2)**2)$. Pro první dvě funkce použijte rozsah $x = [-\pi, \pi]$, pro zvyšší funkce použijte $x = [0, 10]$. V tomhle příkladu si jenom ukážeme obecný postup.

```
01: >> from numpy import *
02: >> import matplotlib.pyplot as plt
03: >> x = arange(-1, 1, 0.01).tolist()
04: >> y = []
05: >> for i in x:
06: ..     y.append(i**2)
07: ..
08: plt.plot(x, y, 'k-')
09: plt.xlim(-1,1)
10: plt.ylim(0,1)
11: plt.xlabel('x [jednotky]')
12: plt.ylabel('y [jednotky]')
13: plt.show()
```

Na prvních dvou řádcích si načteme všechny potřebné knihovny (`numpy` obsahuje funkce jako `sin` či `log`, a `matplotlib.pyplot` poskytne funkce pro vykreslování grafů). Na 03 si do proměnné `x` vložíme pole hodnot s rovnoměrným rozdělením – první číslo ve funkci `arange()` představuje spodní hodnotu rozsahu, druhé číslo horní hodnotu rozsahu a třetí číslo vzdálenost mezi sousedními hodnotami. Dále spočítáme hodnoty `y`. Když máme připraveny obě pole (`x` i `y` musí mít stejný počet hodnot!), můžeme si vykreslit graf pomocí funkce `plt.plot(x,y)`. Třetí parametr funkce určuje způsob, jakým se má závislost `y(x)` vykreslit – zde jsme zvolili `'k-'`, co znamená, že barva grafu bude černá (`'k'`) a funkce bude vyznačena čarami spojujícími sousední body (`'-'`). Pro další možnosti se podívejte na https://matplotlib.org/3.3.1/api/_as_gen/matplotlib.pyplot.plot.html. Další parametry grafu přidáváme za čárkou (např. `plt.plot(x, y, 'bo', ms=2.0, alpha=0.5)`).

Na řádcích 09 a 10 jsme specifikovali měřítko osí grafu, jak jej chceme vidět ve výsledném obrázku. Na 11 a 12 přidáváme popisky k osám. Nejdůležitější je pak příkaz `plt.show()`, který graf vykreslí v systémovém okně (doteď graf existoval pouze jako data v paměti), ale pozor – po vykreslení se data `plot` z paměti vymažou (ne však hodnoty `x` a `y`). Pro opětovné vykreslení pomocí této funkce budete muset znovu začít od 08.

3.8 Čtení z datových souborů

Jako poslední si vyzkoušíme čtení hodnot ze souborů na disku. V téhle podobě budete většinu doby dostávat data, proto je důležité aby jste se pořádně naučili soubory načítávat.

Vytvořte si pole, které bude mít 10 řádků a 2 sloupce. Na prvním sloupci budou hodnoty `x = [1..10]` a na druhém sloupci spočtete jejich odmocniny. Pole hodnot si pak uložte někde na disk do textového souboru. Z něho si následně do nové proměnné tyto hodnoty načtete.

```

01: >> from numpy import *
02: >> x = arange(1, 11, 1).tolist()
03: >> a = []
04: >> for i in x:
05: ..     a.append([i, sqrt(i)])
06: ..
07: >> soubor = open('D:/datatest.txt' , 'w')
08: >> for i in a:
09: ..     soubor.write( str(i[0])+ ' '+str(i[1])+ '\n' )
10: ..
11: >> soubor.close()
12: >> with open('D:/datatest.txt' , 'r') as f_data:
13: ..     all_data=[x.split() for x in f_data.readlines()]
14: ..     f=array([list(map(str,x)) for x in all_data[0:]])
15: ..
16: >> print(f)

```

Po řádek 06 by mělo být všechno srozumitelné z předešlých příkladů. Příkaz na 07 vytvoří nový soubor na dané cestě (cestu k souboru nebo i jeho názov můžete změnit), parametr 'w' pak řekne programu, že chceme do souboru psát od začátku souboru ('w+' by nám dovolilo psát i číst, 'r' iba číst, 'a' začne psát od konce souboru). V následujícím cyklu začneme do souboru psát pomocí příkazu `soubor.write()`, s tím, že na konec každého řádku přidáme `\n` (tohle je standartní znak pro zalamování řádků). Příkaz na 11 je velice důležitý, protože správně uzavře soubor (doporučeno psát vždy po ukončení práce se souborem).

Je trochu složitější popsat, co série řádků 12 až 14 dělá. Jednoduše – tohle slouží na otevírání souborů a všechny obsah souboru zkonvertuje na textové řetězce a uloží do proměnné `f`. Důležité pro vás je, že zde musíte zadat cestu k souboru, a že může byt nutno odfiltrovat některé řádky (`all_data[0:]` sebere všechno od prvního řádku souboru až na konec, `all_data[1:4]` by načítalo pouze řádky 2 až 4, tedy indexy 1 az 3). Na konci programu je výpis proměnné `f`. Kdybychom chtěli s polem `f` pracovat dále, museli bychom hodnoty změnit na reálná čísla (skuste si to).