PYTHON
materská škôlka pre fyzikov

Vladimír Černý

1. Prečo?

Mojím snom pri vyučovaní fyziky nie je ani tak naučiť fyziku ako naučiť poslucháčov

„rozumieť tomu, čo to znamená rozumieť“.

Znamená to (aj) napríklad, že študent prežije zákernú otázku: „Tá malá dvojka, čo ste vo vzorci napísali, to je index alebo exponent?“ Poslucháči si sami nekladú také zákerné otázky. Preto ľahko získajú mylný dojem, že učivu rozumejú.

Zažil som, ako študent napísal všeobecný vzorec pre výpočet štatistickej sumy:

$$\sum_i \exp\left(\frac{E_i}{kT}\right)$$

Neprežil kontrolnú otázku: „Ako vyzerá tá suma pre systém, ktorý sa môže nachádzať iba v dvoch rozličných stavoch?“ Problém bol v tom, že nevedel, čo je v tej sume označené indexom i. V bežnom živote často vystačíme s intuitívnym chápaním slov. Počujem „stolička“ a nekladiem si hneď kontrolnú otázku „čo presne je stolička“. Povedzte Marťanovi „prisuň mi tamtú stoličku“ a on povie „žiadna tu nie je“. Poviete: „No a pri tom klavíri, to je čo?“

Naučil som sa na seminári od nositeľa Nobelovej ceny M. Veltmana ako sa presvedčiť, že rozumiem tomu, čo znamenajú hieroglyfy, ktoré píšem. Naprogramovať to.

Počítač je totiž ideálny hlupák.

Nemá intuíciu, nič si (ani dobre ani zle) nedomyslí a hodí modrú obrazovku ak „nerozumie“. Veltman tvrdil, že kľúčom k získaniu Nobelovej ceny bolo, že sa im spolu s ‘t Hooftom podarilo naprogramovať čosi ťažké z kvantovej teórie poľa. Až tak im presne došlo, o čo v celej veci ide.

Čaká nás úvodná prednáška z fyziky. Mám takú ideu, že budete lepšie rozumieť, ak viaceré príklady nebudeme „riešiť“, ale budeme ich programovať. Ak sa vám podarí v počítači „spustiť virtuálny svet“, budete lepšie rozumieť tomu, ako funguje ten reálny. Mám takú ideu. Uvidíme.

Problém je v tom, že mnohí neviete programovať.

Nevadí, veď na to máte celý najbližší víkend.

Myslím to vážne. Tento text píšem preto, aby ste sa cez víkend naučili programovať. Paradoxne mám na mysli úplný opak toho, čo som vyššie písal o fyzike. Nenaučíte sa rozumieť tomu, ako sa programuje. Vystačíte s intuitívnym napodobovaním nejakých vzorov. Nenaučíte sa *poriadne* programovať. Ale bude to dosť na to, aby ste stvorili niekoľko primitívnych fyzikálnych virtuálnych svetov.

Programovanie v Pythone môže byť aj riadne ťažká vec. Ale Python má jednu výhodu. Jednoduché veci sa dajú urobiť veľmi jednoducho. Zložité veci robiť nebudeme. Nechcem z vás urobiť programátorov. Budeme iba používať jednoducho programovateľnú kalkulačku.

Tak preto Python.

2. Infraštruktúra.

Ak chcete programovať v Pythone, potrebujete Python. V počítači. Ak máte notebook s Windowsami, Python tam spravidla nie je. Treba si pripraviť infraštruktúru, čo znamená dve veci:

- nainštalovať vhodnú distribúciu Pythonu
- nainštalovať (ak ešte nemáte) programátorsky priateľský textový editor

2.1. Vhodná distribúcia Pythonu

Nebudem tu dlho diskutovať: **nainštalujte si distribúciu Anaconda Scientific Python Distribution**

http://docs.continuum.io/anaconda/install.html

Je to kompletný balík obsahujúci Python 2.7, jednoduchú interaktívny Python terminál ipython, pokročilejšie vývojové prostredie spyder. A najmä pre nás potrebné matematické prostredie Scipy s balíkmi ako numpy a matplotlib. Nájdete tam inštalačky pre Windows aj pre Linux.

Poznamenajme, že celá distribúcia zaberie na disku takmer 2 GB. Pri inštalácii si môžete zvoliť priečinok, do ktorého sa má nainštalovať (ako default c:\Anaconda). Inštalácie je prenositeľná (portable), t. j ak si to necháte nainštalovať na USB kľúč, potom môžete ten kľúč zastrčiť do iného počítača a všetko bude fungovať. Presnejšie všetko až na to, že na tom druhom počítači sa nenastavia prístupové cesty (PATH) v rámci vašich environment premenných (nech vás netrápi, ak toto celkom nerozumiete). Buď si ich nastavte sami, alebo budete vyvolávať Python zadaním plne kvalifikovaného mena (pozri kapitolu o spúšťaní Pythonu), alebo nepoužívajte distribúciu ako portable a nechajte si ju nainštalovať do každého počítača.

Možno vás niekto upozorní, že už existujú verzie Python 3.x. Nedajte sa nalákať, Scipy zatiaľ najlepšie funguje na verzii 2.7.

To, čo som tu napísal platí v podstate v rámci drobných modifikácií pre inštaláciu pod Windowsami aj pod Linuxom. Inštalácia je bezproblémová, otázky, ktoré užívateľ počas nej musí zodpovedať sú triviálne. Ak ste si doteraz na svoj počítač nič sami neinštalovali, potom si ale na inštaláciu zavolajte skúsenejšieho kolegu, nech sa vám pozerá na ruky.

2.2. Programátorsky priateľský textový editor

Hlavné upozornenie je toto: potrebujete textový editor a nie word procesor (ako je Microsoft Office Word). Rozdiel je v tom, že textový editor vyrába textové súbory, kde sú len bezprostredne zakódované písmená (v ASCII kódovaní), kým word procesor vyrába krajšie formátované texty, ale súbor, ktorý vyrobí, obsahuje okrem zakódovaných písmen aj rozličné formátovacie príkazy typu kde začína nový paragraf, ako má byť odsadený, či je písmo normálne, tučné alebo kurzíva a mnoho iných vecí.

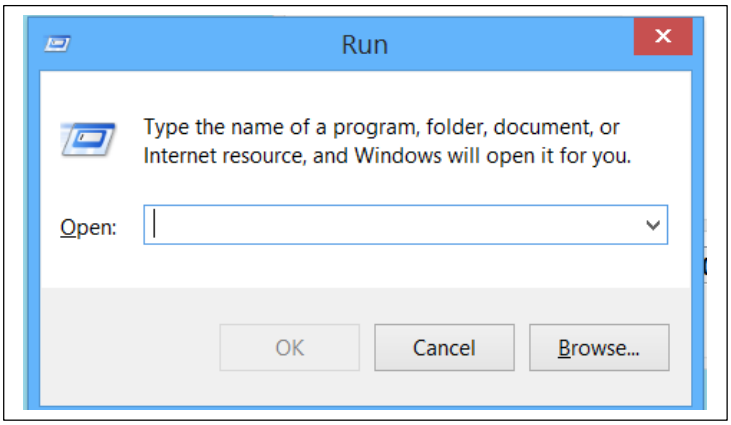
Pre výber textového editora nemám striktnú požiadavku, možno ste si už niečo obľúbili. Vo Windows vystačíte s predinštalovaným editorom Notepad ale odporúčam priateľskejší Notepad++

http://notepad-plus-plus.org/

2.3. Terminál/shell

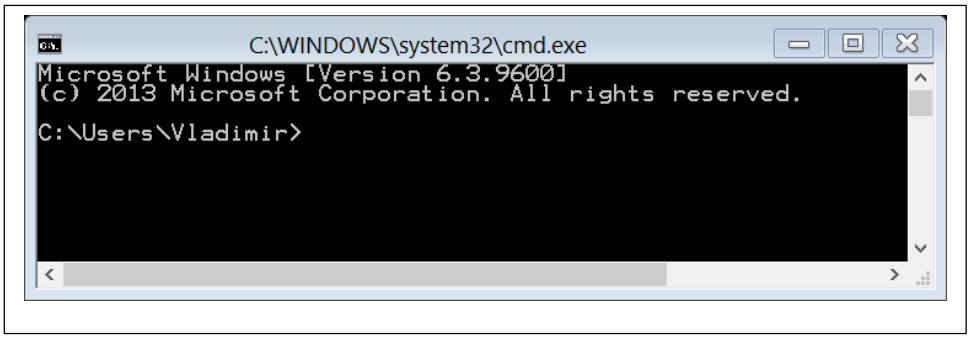
Dnešné počítače sa spravidla ovládajú klikaním myšou na ikony alebo buttony (dnes už aj v Linuxe). Ak máte programovať, musíte akceptovať ovládanie písaním príkazov v textovom móde. Možno ste také ešte nevideli.

Prvá vec je, že si musíte otvoriť **terminálové okno operačného systému**. Vo Windowsoch najjednoduchšie tak, že stlačíte súčasne dva klávesy Win+R. (Win je kláves na ktorom je vyobrazené logo Windows, nachádza sa medzi klávesmi Ctrl a Alt. Objaví sa okno „Run“



Do textového poľa zapíšete cmd. Kliknete na OK a spustí sa terminálové okno operačného systému.

Niekto hovorí, že sa spustí shell. Je to termín ktorý označuje program, ktorým sa ovláda samotný operačný systém tak, že sa zadávajú „príkazy shellu“. Ak chcete byt Windows guru, používajte namiesto cmd modernejší a bohatší powershell.

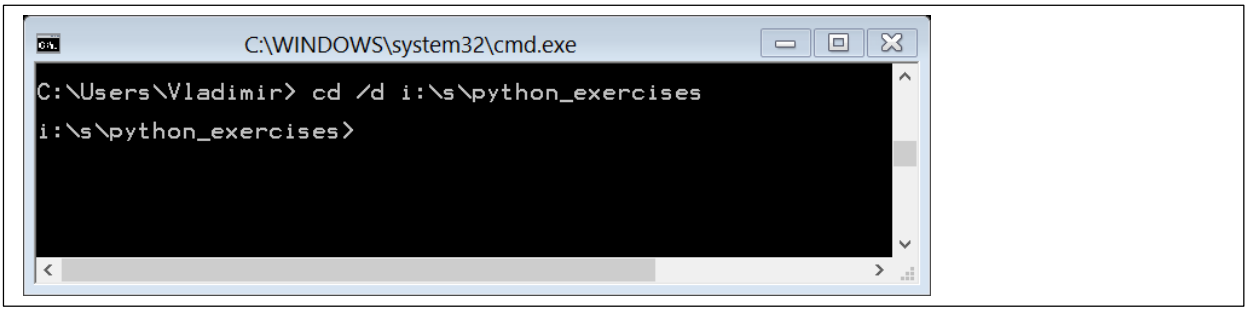


Reťazcu písmen C:\Users\Vladimir> sa hovorí command prompt. Tým reťazcom nás powershell vyzýva, aby sme zadali v tom riadku nejaký príkaz shellu. Súčasne prompt ukazuje, ktorý priečinok je momentálne mojím pracovným priečinkom (working directory). Sem sa budú napríklad zapisovať súbory, ak pri príkaze typu „save“ neuvediem plne kvalifikované meno.

Preto je užitočné zmeniť pracovný priečinok za taký, kde si chcem ukladať súbory, ktoré počas ďalšej aktivity (napríklad počas programovania v Pythone) budem vytvárať. Môžem tak urobiť príkazom

```
cd /d
```

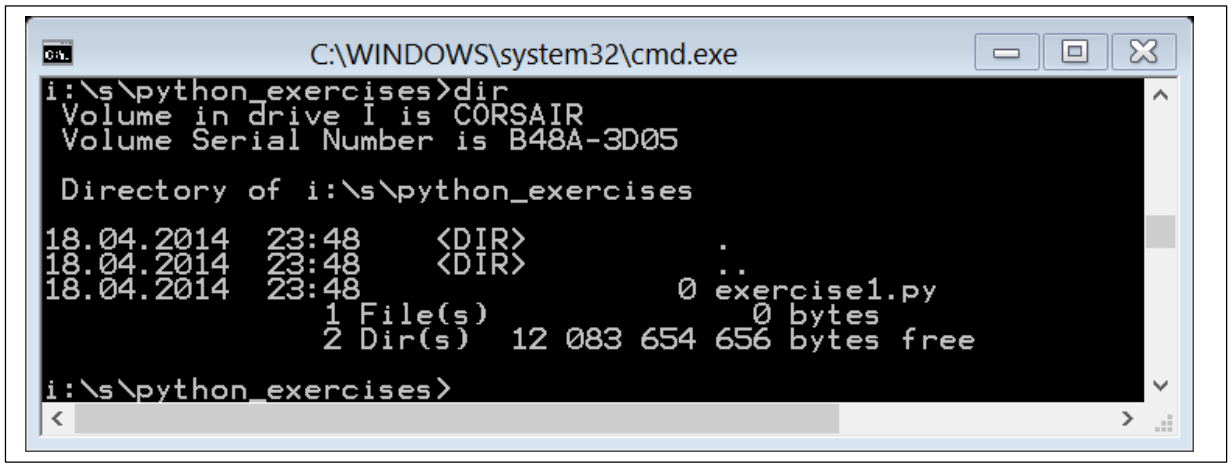
napríklad



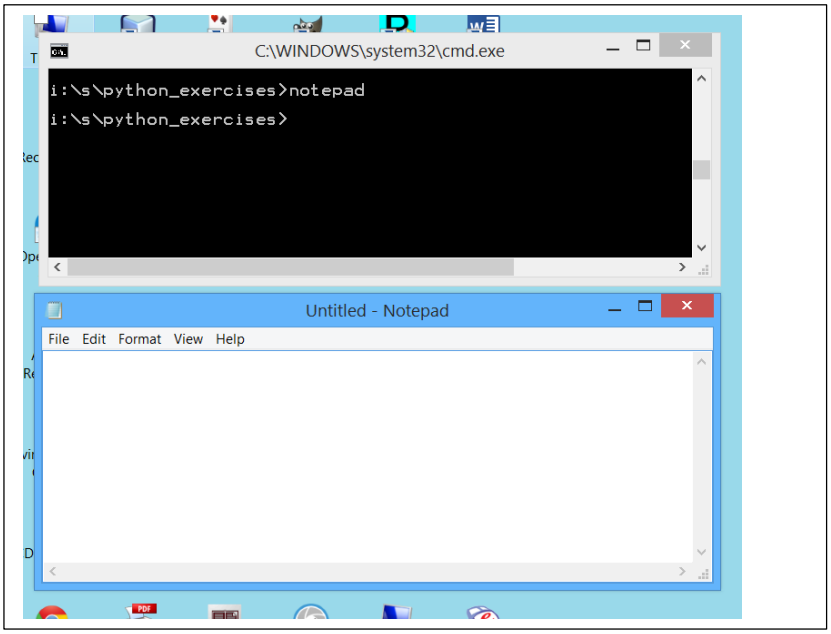
Všimnite si, že som prešiel do priečinku I:\s\python_exercises>. Ten priečinok som si založil na to, aby som si tam skúšal rôzne programy, ktoré sa učím robiť v Pythone. Je dôležité vymyslieť si nejaký svoj systém na organizáciu disku a potom ten systém dodržiavať.

Obsah priečinku môžem vypísať príkazom

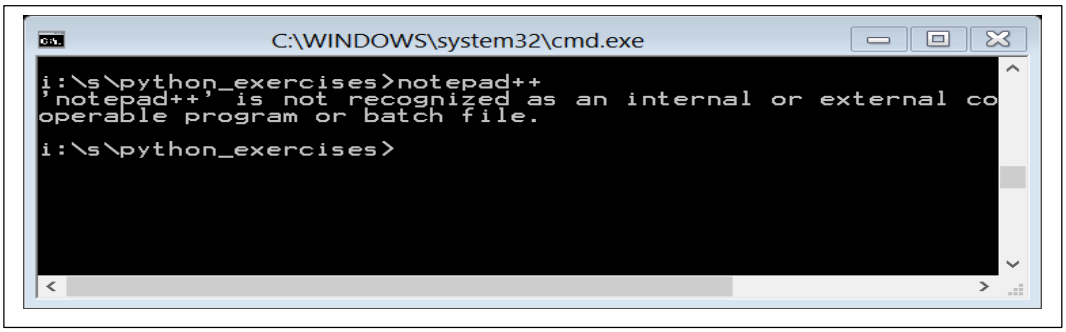
dir



Dôležitým príkazom je vyvolanie iného programu (niekedy hovoríme aplikácie). Stačí napísať meno programu, napríklad notepad a uvidíme že sa otvorí okno textového editora notepad:



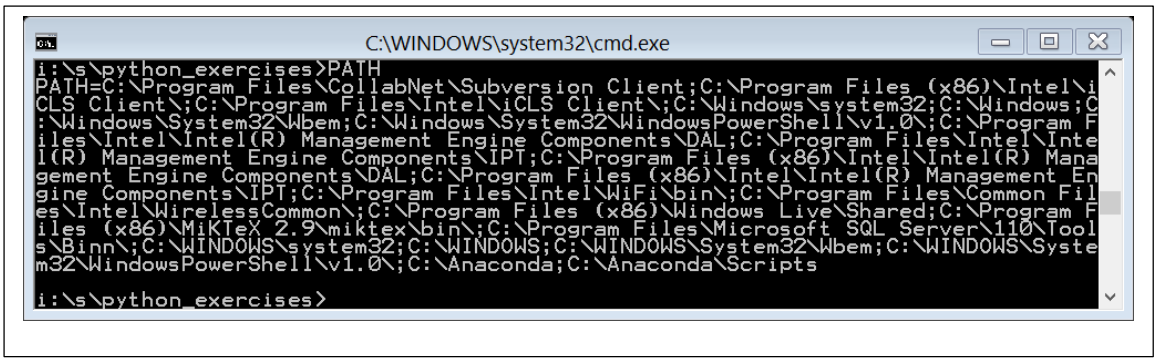
Povedali sme si ale, že budeme radšej používať textový editor notepad++. Skúsme ho vyvolať, ale namiesto okna editora sa objaví na termináli chybová hláška



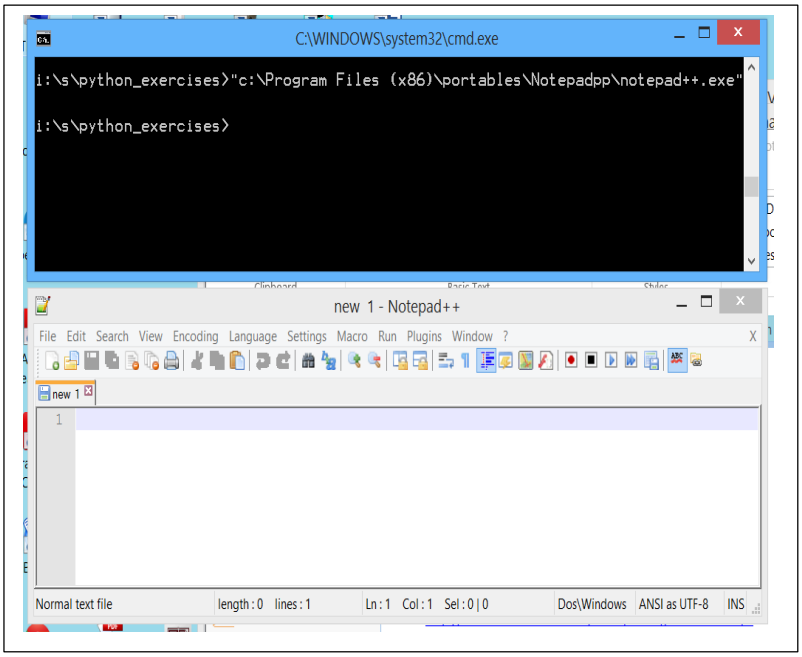
cmd shell hovorí, že nepozná program notepad++. Ale ja viem, že som si ho nainštaloval, je v priečinku

“c:\Program Files (x86)\portables\Notepadpp”

Problém je v tom, že tento priečinok nie je zaradený v zozname priečinkov, ktorý shell prehľadáva, keď hľadá program, ktorý chce spustiť. Ten zoznam je uložený v tzv. environmentálnej premennej PATH a dá sa vypísať príkazom PATH



Jednotlivé priečinky sú v tom zozname oddelené bodkočiarkou. Inšpekcia ukáže, že priečinok obsahujúci notepad++ tam uvedený nie je. Ak chcem ten program vyvolať, musím uviesť plne kvalifikované meno



Vidno, že teraz sa okno s editorom otvorilo. Všimnime si tiež, že meno sme dali do úvodzoviek. Je to preto, že meno obsahuje medzery (v názve priečinka Program Files (x86)). Úvodzovky sú v tomto prípadne potrebné, aby shell vedel, že meno programu ešte pokračuje aj po medzere.

Poznamenajme, že je to veľmi zlý nápad, používať v názvoch priečinkov alebo súborov medzery alebo diakritiku. Microsoft to od istého času povoľuje, ale ak to budete používať, skôr alebo neskôr si narobíte problémy. Pri prenosoch súborov do iných operačných systémov. Aj s úvodzovkami to nie je jednoduché, dakedy ich musíte použiť, dakedy nesmiete. Všimnite si, že v zozname PATH (na predchádzajúcom obrázku) úvodzovky pri menách priečinkov s medzerami používané nie sú (a nesmú byť, ale ako to máte vedieť?), lebo tam je meno priečinku jasne ukončené bodkočiarkou.

3. Python ako kalkulačka s pomenovateľnými bunkami pamäte

3.1. Spustenie Pythonu

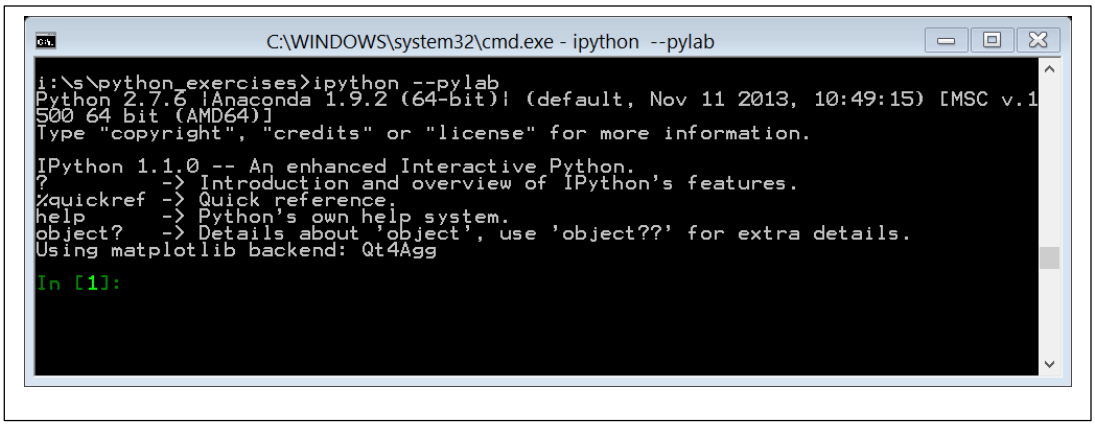
Python budeme štartovať takto:

- otvoríme terminál operačného systému (cmd)
- zadáme príkaz

```
ipython --pylab
```

slovo ipython je meno programu, jeho spustenie spôsobí, že v okno cmd shellu sa „za jazdy“ premení **na interaktívne terminálové okno Pythonu**

predchádzajúca história príkazov cmd shellu sa nevymaže, ale po vyvolaní ipythonu už nedostanem prompty cmd shellu ale prompty interaktívneho Pythonu. Vyzerá to takto



Skúsenejší programátori si môžu najprv všimnúť riadky nad promptom In [1], kde je popísané, ako sa dá vojsť do rôznych úrovní on-line helpov. Ak chcete poexperimentujte.

Reťazec In[1]: je promtom interaktívneho Pythonu a znamená, že interpreter Pythonu odo mňa očakáva zadanie prvého príkazu v jazyku Python.

Kým začneme zadávať príkazy, malá poznámka o tom, prečo sme príkaz ipython vyvolali s doplnkovým parametrom `--pylab`.

Bez toto parametra by sa totiž spustil len základný Python, ktorý by nepoznal mnohé príkazy a funkcie potrebné pre „scientific computing“ (nevedel by napríklad, čo je to `cos()`).

Parameter `--pylab` spôsobí, že do Pythonu sa „natiahnu“ aj viaceré podporné „knižnice“. Pre naše potreby je dôležitá najmä knižnica-modul numpy (pre podporu numerických výpočtov) a knižnica matplotlib (prostriedky pre kreslenie grafov). Meno „pylab“ je parafrázou na populárny komerčný softvér pre vedecko-technické výpočty Matlab (prefix Mat bol nahradený prefixom py, čo evokuje slovo Python).

Ešte dôležité: ako interaktívny Python ukončiť. Stlačením Ctrl D. Python sa opýta, že či naozaj.

3.2. Python je interpreter

Ak ste už programovali v nejakom vyššom jazyku, používali ste spojenia ako: mám nainštalovaný kompilátor c++, skompiloval som program a potom som ho spustil. Základný (mierne zjednodušený) postup pri použití c++ je

edit → compile → run

Python je interpreter, odpadá pri ňom fáza kompilácie. Ak interpreteru Pythonu predložíme na spracovanie súbor obsahujúce riadky programu v jazyku Python, potom interpreter číta (interpretuje) program riadok po riadku a ak je prečítaný riadok vykonateľný, hneď ho aj vykoná. Ak prečítaný riadok nie je priamo vykonateľný, načítava postupne ďalšie riadky kým nezistí, že prečítaná skupina riadkov je už (ako celok) vykonateľná a hneď ju aj vykoná.

Okrem spracovania programu sa dá interpreter Pythonu používať i v interaktívnom režime. V tom režime mu nepredložíme nejaký súbor riadkov pripravený v textovom editore, ale na výzvu (riadkový prompt zobrazený na obrazovke) cez klávesnicu zadáme jeden programový riadok a interpreter ho hneď vykoná. Po vykonaní zobrazí ďalší riadkový prompt a očakáva zadanie ďalšieho riadku. Podobne ako v režime spracovania programového súboru platí, že ak zadaný riadok nie je samostatne vykonateľný, zobrazí „pokračovací prompt“, teda vypýta si doplňujúce riadky. Až skupina zadaných riadkov je ako celok vykonateľná, vykoná ju a zobrazí ďalší riadkový prompt.

Niekoľko slov k terminológii.

Slovo Python sa (aj v tomto texte) používa vo viacerých významoch

- primárne je to názov programovacieho jazyka Python
- je to i názov programu, interpretera jazyka Python
- je to i názov interaktívneho terminálu interpretera Python, pomocou ktorého interpreter komunikuje s programátorom

V predchádzajúcich riadkoch sme súbor riadkov v jazyku Python nazývali program. Častejšie sa používa názov **skript**, ktorým chceme odlíšiť, že skript je čítaný interpreterom a nie kompilátorom.

Výhoda interpretera voči kompilátoru je takmer úplná prenositeľnosť. To znamená, že skript je spravidla interpretovateľný na ľubovoľnom počítači, na ktorom je nainštalovaný interpreter Pythonu. Samotný interpreter prenositeľný (binárne) nie je, na každom počítači musíme nainštalovať interpreter špecifický pre daný hardvér a daný operačný systém. Ale potom už skripty prenositeľné sú.

Oproti tomu skompilovaný program spravidla prenositeľný nie je, na každom počítači musíme najprv zdrojový (textový) program najprv skompilovať, až potom ho možno spustiť.

Nevýhodou interpretovaného jazyka voči kompilovanému je spravidla nižšia rýchlosť jeho vykonávania. Kompilácia trvá niekedy dosť dlho, ale samotný beh je potom už rýchly, lebo produkt kompilácie je optimalizovaný pre daný počítač.

Interpretovaný program „sa zdržuje“ interpretovaním každého riadka a to pri každom „behu“ znovu a znovu. Ako keby sme pred každým použitím kompilovaného programu ho museli napred vždy odznovu skompilovať.

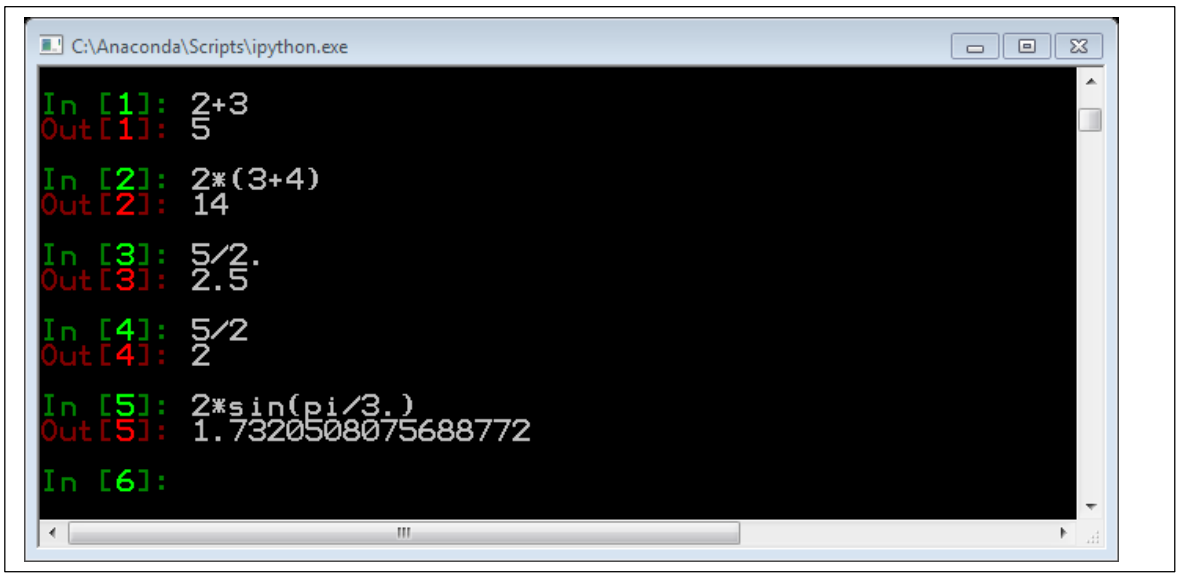
3.3. Python ako kalkulačka

Tu konečne začíname s programovaním v jazyku Python. Postupne budeme prechádzať niekoľkými praktickými cvičeniami**. Naozaj si tie cvičenia urobte**! Neobmedzte sa len na čítanie tohto textu, „lebo je to všetko jednoduché“. Pri fyzickom vykonávaní cvičenia občasne narazíte na čosi, čo nie je presne rovnaké, ako je popísané v texte. Môžete mať inú verziu programu než som používal pri písaní textu ja, iný operačný systém, inak konfigurovaný interpreter, inú verziu knižníc. Nakoniec by sa vám malo podariť všetko prebojovať, buď používaním len vlastnej hlavy alebo s pomocou skúsenejšieho kolegu. Ale určite nepodceňujte vlastnú hlavu. Ak si ju ponamáhate, často zistíte, že vaša hlava je múdrejšia ako si myslíte.

Ak použijete „priateľa na telefóne“ neposlúchajte ho iba mechanicky, keď povie čosi typu „Stlač tabelátor“. Dajte si námahu pochopiť, prečo máte urobiť práve to, čo vám radí, čo sa tým stane a teda nakoniec „ako to funguje“.

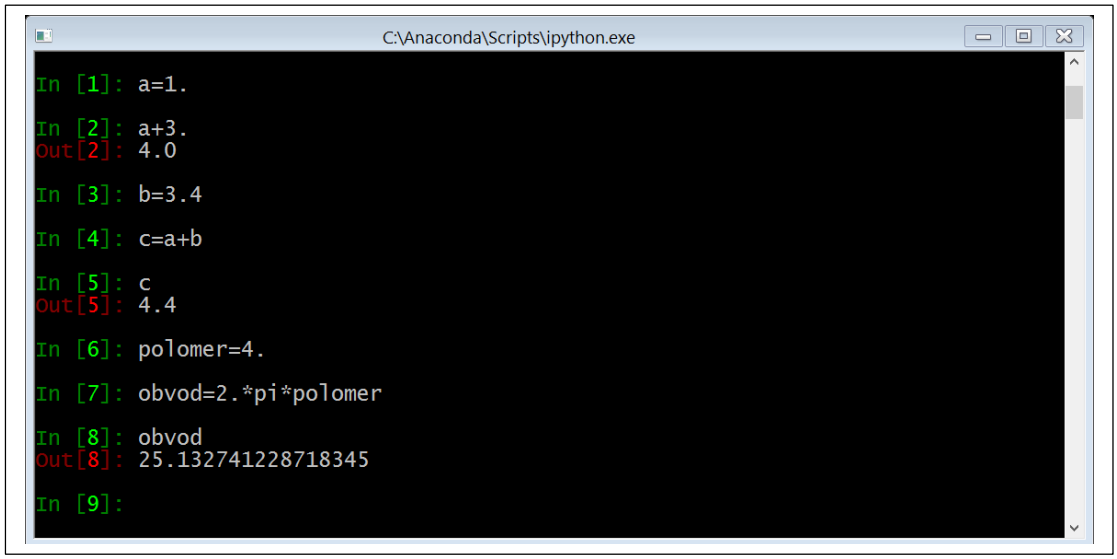
Môžete mať malý zošitok, zvaný blbníček. Ale ak si ku každému ČO nepoznačíte (alebo aspoň nepredumáte) PREČO, blbníček môže zlyhať pri prvom stretnutí s novou verziou softvéru.

Spustite interaktívny Python príkazom ipython –pylab. Potom zadávajte jednoduché programové riadky ako vidíte na obrázku



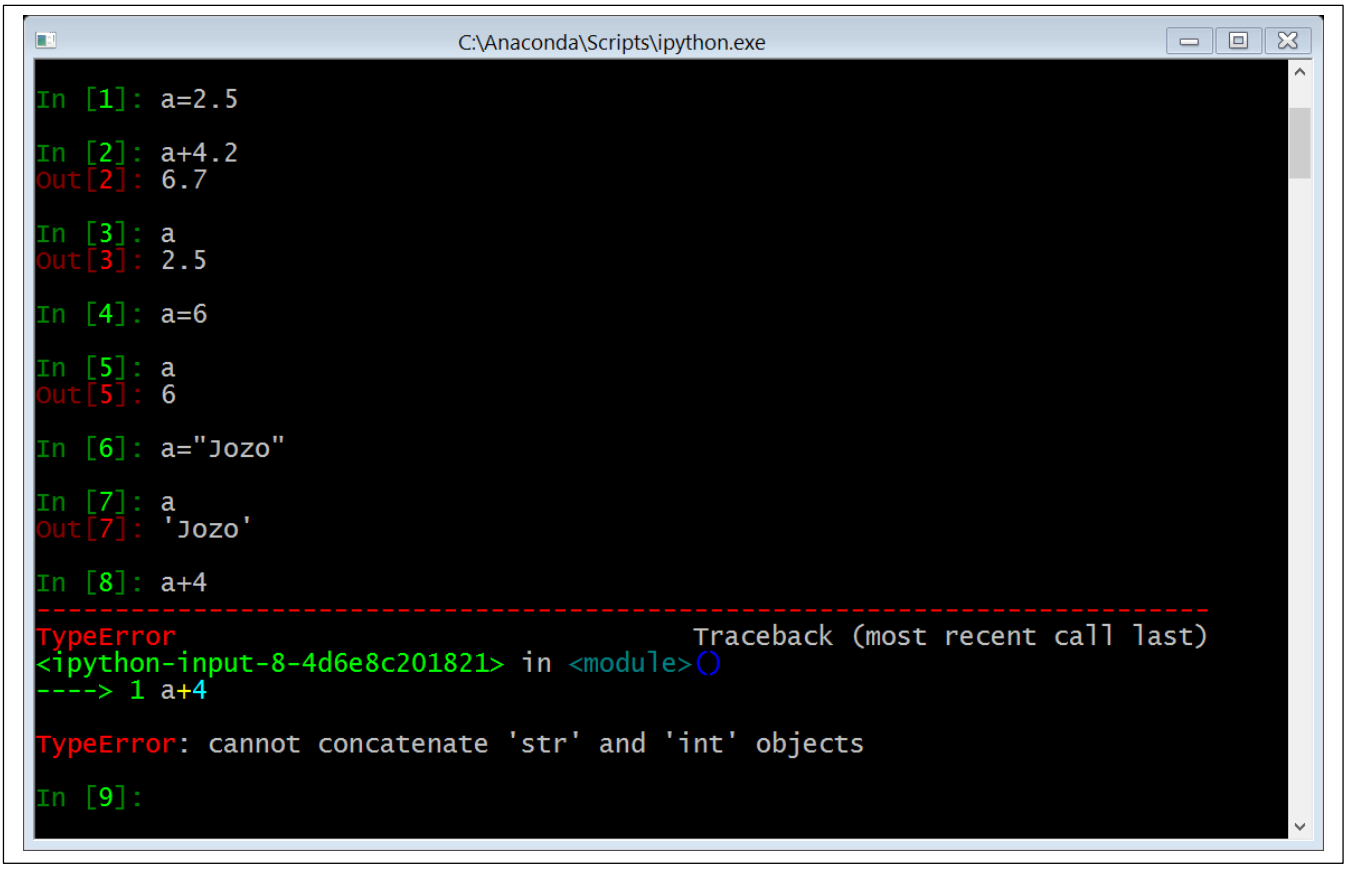
- všimnite si rozdiel vo výsledkoch v príkazoch č. 3 a 4. V príkaze č.3 delíme číslom s desatinnou bodkou, čím hovoríme Pythonu, že má deliť reálnym (float) číslom. V príkaze č. 4 delíme celým číslom 2. Ide o delenie celého čísla celým číslom, čo pre Python znamená, že uvedie výsledok celočíselného delenia (delenie so zvyškom). Keď chcem zistiť zvyšok takého delenia, použijem príkaz 5%2, dostanem výsledok 1
- dajú sa používať zátvorky v obvyklom matematickom význame
- dajú sa používať funkcie typu sin() ako v bežnej „vedeckej“ kalkulačke

Ak chcem celý výpočet rozdeliť do postupnosti niekoľkých riadkoch, potom je vhodné vždy si uložiť výsledok (medzivýsledok) do pamäte. Na rozdiel od bežnej vreckovej kalkulačky, Python má k dispozícii neobmedzený (takmer) počet pamäťových miest. Používajú sa tak, že si pre každé pamäťové miesto, ktoré chcem použiť, musím zvoliť nejaké (dosť ľubovoľné) meno, napríklad jedno písmeno ako a alebo r, alebo aj viacpísmenové označenie, ktoré mi pripomenie, čo tam mám uložené. A ukladám tam čísla pomocou príkazu =, ako je to uvedené na ďalšom obrázku.



Poznámky:

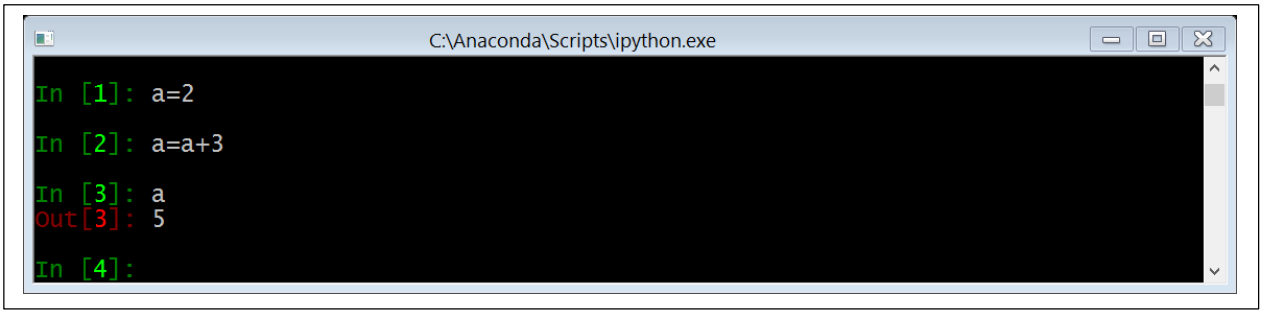
- Príkaz = sa interpretuje takto: vypočítaj výraz na pravej strane a ulož to do bunky pamäte pomenovanej na ľavej strane
- ak na ľavej strane vo výraze vystupuje pomenovaná bunka pamäte, interpretuje sa to takto: keď vyčísľuješ výraz, vyber číslo, ktoré je obsahom pomenovanej bunky a použi ho na výpočet výrazu na pravej strane
- Všimnime si, že za príkazovým riadkami č.2, 5, 8 nasleduje hneď výpis výsledku, kým za riadkami 1, 3, 4, 6, 7 výpis nenasleduje. Logika je približne takáto. V riadku, v ktorom nie je príkaz = sa výsledok výpočtu pravej strany nikam neuloží, interpreter ho preto aspoň vypíše, aby sa úplne nestratil. V riadkoch, v ktorých sa vyskytuje príkaz = sa výsledok výpočtu pravej strany uloží do bunky pamäte pomenovanej na ľavej strane, nestratí sa a propgramátor si ho môže nechať zobraziť tak, ako je ukázané v riadkoch 5 a 8.
- doteraz používaný amatérsky výraz „pomenovaná bunka pamäte“ už v ďalšom nahradíme oficiálnym výrazom **premenná**
- Ak v programe prvýkrát uvedieme meno premennej, vyhradí sa pre ňu bunka pamäte, do ktorej sa potom ukladajú hodnoty, ak je premenná uvedená na ľavej strane príkazu =.
- poznamenajme, že výraz premenná má v jazyku Python širší význam, než ako sme tu vnímali. Do príslušného miesta pamäte je možné uložiť nielen číslo ale napríklad aj reťazec písmen (tzv. string) alebo nejakú bohatšiu dátovú štruktúru.
- Python na rozdiel od iných jazykov nevyžaduje explicitnú deklaráciu typu dát, ktoré sa majú ukladať do nejakej premennej. Pozrite na obrázku



Poznámky:

- Do premennej a sme najprv v riadku 1 uložili reálne (float) číslo a tak sme s ním aj počítali. Potom sme v riadku 4 uložili do tej istej premennej celé (integer) číslo. Porovnanie výstupov 3 a 5 ukazuje, že sa zmenila nielen hodnota čísla uloženého v premennej ale aj typ obsahu, najprv to bolo float číslo, potom integer číslo.
- Potom sme do premennej a v riadku 6 uložili reťazec písmen (string. String a jeho zápis pomocou úvodzoviek tu vidíme prvýkrát.)
- Ak zabudneme, že sme do premennej uložili string a chceme s premennou a počítať ako keby jej obsahom bolo číslo, ako sme to urobili v riadku 8, počítač odpovie chybovou hláškou, ktorá hovorí, že počítač nevie ako sa má zreťaziť údaj typu string s údajom typu integer

Všimnime si ešte jeden zdanlivý paradox, ktorý niekedy robí problémy začiatočníkom, sledujte ďalší obrázok



Problém môže robiť riadok 2, ktorý vyzerá ako chybná rovnica. Laik by povedal, že hodnota a nemôže byť rovná hodnote a+3

Riadok 2 nezapisuje žiadnu rovnosť. Po logickej stránke ide o príkaz = ktorý sa interpretuje tak, ako sme to vyššie popísali. Najprv sa vyčísli pravá strana a to tak, že sa vyberie z premennej a jej obsah, pripočíta sa k nemu hodnota 3 a výsledok sa vloží do premennej a. Teda nová a stará hodnota nikdy nie sú prítomné v premenne a súčasne, k sporu nedochádza. Zápisom v riadku 3 nechám vypísať už nový obsah premennej a.

Už to, čo sme sa naučili doteraz, umožňuje nám vypočítať čokoľvek, čo sa dá vypočítať na „vedeckej" kalkulačke, teda prakticky čokoľvek. Ibaže výpočet robený takými primitívnymi prostriedkami vyžaduje priveľa ľudskej manipulácie a priveľa času. Naozajstné programy umožňujú väčšinu manipulácií automatizovať (naprogramovať), takže výpočtový program potom už pracuje autonómne bez zásahu operátora, môže vykonať tisíce príkazových riadkov a nakoniec oznámiť výsledok.

Uvedomme si ale, že prví vedeckí astronómovia nemali ani takúto primitívnu kalkulačku. Všetko rátali písomne na papieri a i tak boli schopní zrátať napríklad kedy bude najbližšie zatmenie Slnka.

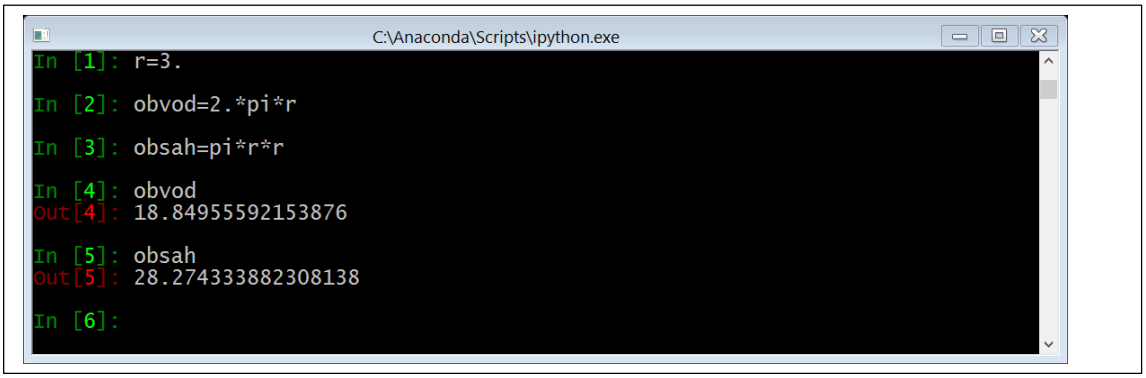
Používať interpreter Python ako kalkulačku teda už vieme. V ďalšom sa naučíme niekoľko trikov ako výpočty zautomatizovať.

Nakoniec poznámku pre pokročilejších programátorov. **Celé, čo som tu rozprával o ukladaní do buniek pamäte bola lož použitá ako dosť surová pedagogická licencia. Skutočnosť je oveľa zložitejšia.** Kto chce poznať lepšiu pravdu, nech si prečíta pekne napísanú stránku

http://python.net/~goodger/projects/pycon/2007/idiomatic/handout.html#other-languages-have-variables

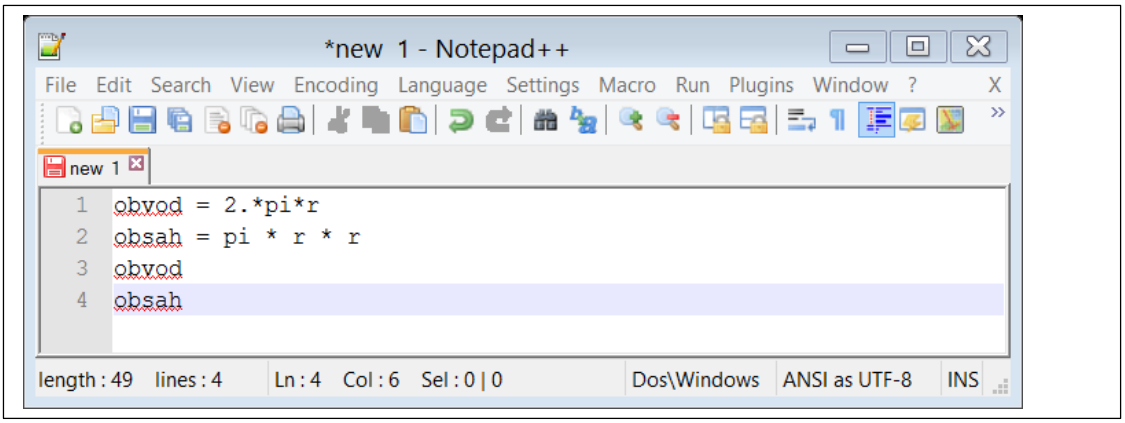
4. Skript ako niekoľko riadkov pre viacnásobné použitie

Predstavme si výpočet obvodu a obsahu kruhu zo zadaného polomeru. V Pythone to môže vyzerať takto



Predstavme si, že by sme potrebovali opakovane počítať obvody a obsahy kruhov pre rôzne hodnoty polomerov. Podľa toho, čo zatiaľ vieme, by sme museli opakovane zadávať riadky 2, 3, 4, 5. To je nepraktické. Jedno možné riešenie je zapísať tie riadky iba raz do nejakého textového súboru a ten potom opakovane „vyvolávať" v Pythone. Skúsme to.

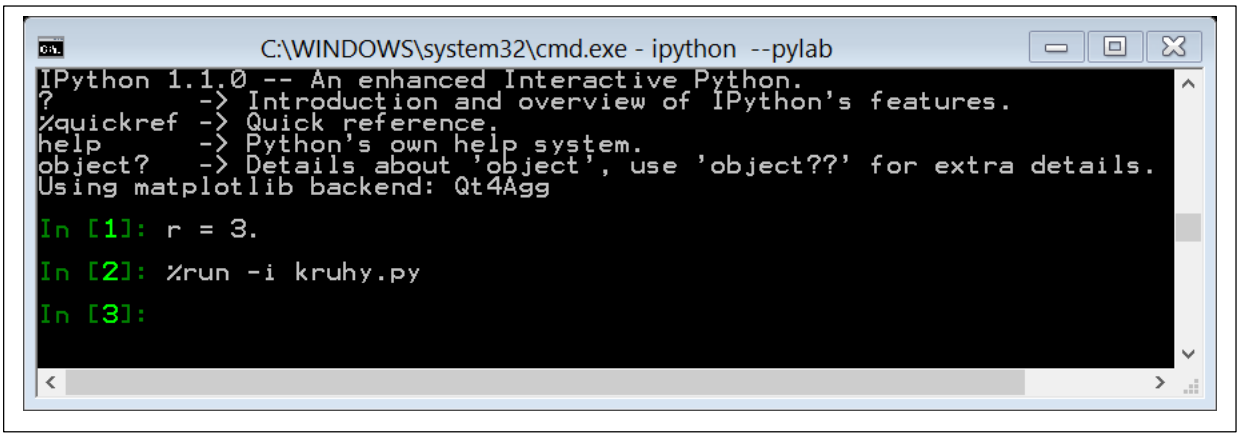
Presvedčte sa, že v terminálovom okne operačného systému sme v pracovnom priečinku, kam si chceme ukladať súbory z našich cvičení s Pythonom. Otvorte notepad++. Zapíšme do pripravovaného súboru príkazy, ktoré sme interaktívne použili v riadkoch 2, 3, 4, 5 (v notepade nemáme žiadne prompty!). Bude to vyzerať takto (poznamenajme dopredu, že tu najprv zámerne urobíme akúsi chybu, aby sme si správny postup lepšie zapamätali a najmä porozumeli)



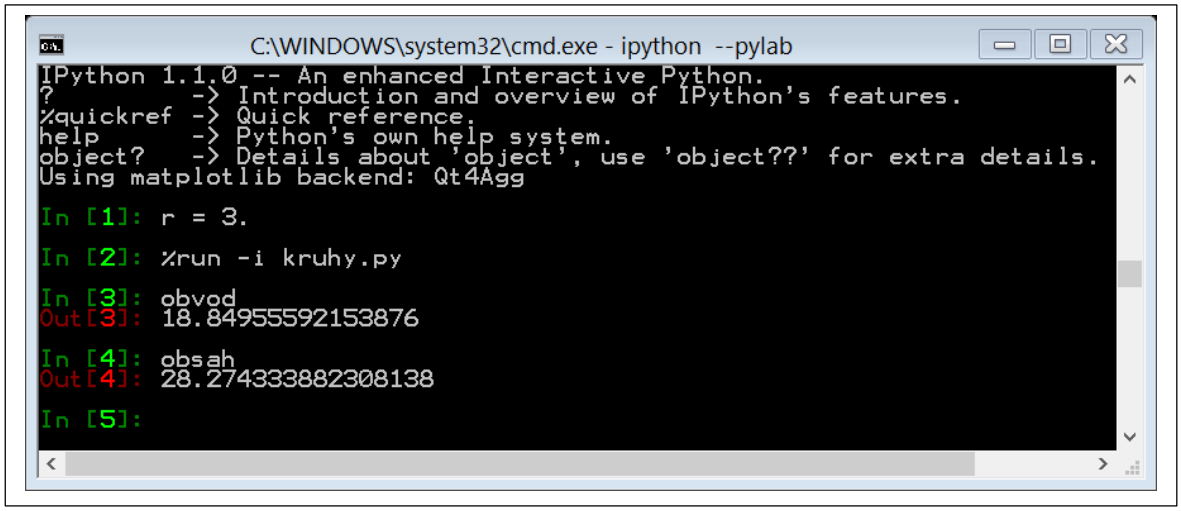
Mimochodom, všimnite si, že notepad++ sa vám snaží pomáhať tak, že ponúka predpokladané slovo, ktoré chcete napísať. (Asi to poznáte z prediktívnych editorov SMS v mobiloch.) **Poznamenajme, že aj samotný ipython ponúka v istých prípadoch prediktívne editovanie. ak neviete ako ďalej, stlačte kláves Tab: je možné, že ipython vypíše niekoľko možností, čo by mohlo ďalej nasledovať.** (Alebo aj nevypíše, keď vôbec netuší, čo tak chcete robiť.)

Teraz uložme vytvorený súbor pod menom kruhy.py do pracovného priečinku. Zvolili sme extension py. Nie je to povinné, ale je to všeobecný zvyk. V skriptoch Pythonu používať extension py.

Poďme do terminálového okna, otvorme interaktívny Python. Zadajme veľkosť polomeru v prvom riadku a potom vyvolajme skript kruhy.py príkazom %run -i. Takto



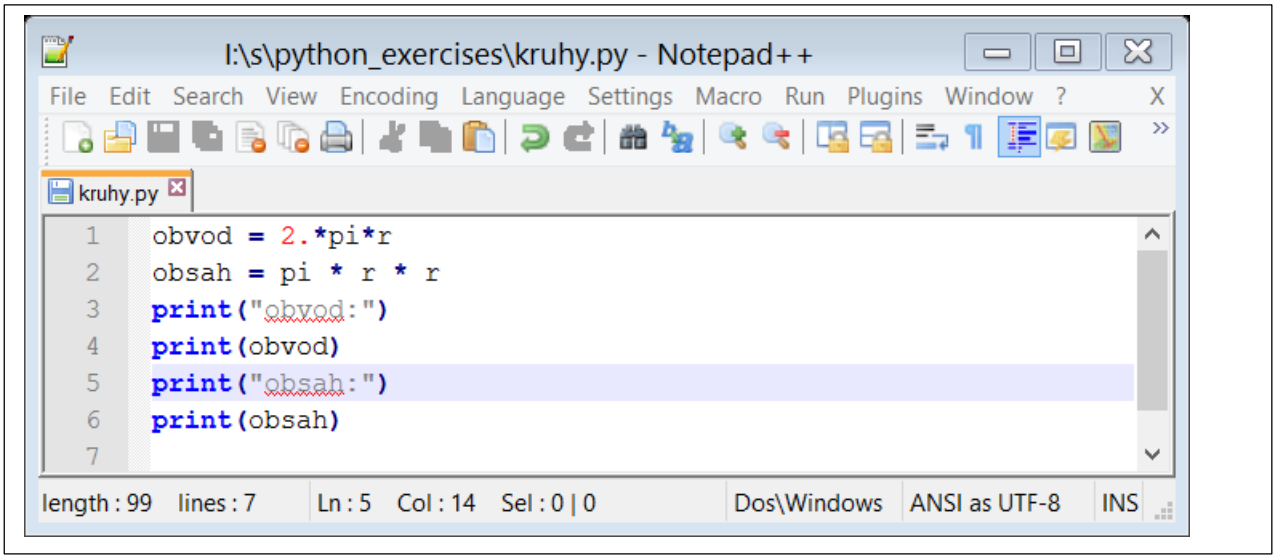
Vidno, že to nefungovalo ako sme dúfali, lebo nevidíme výsledky. Výsledky sú tam, možno ich vyvolať „ručne“ doplnením ďalších interaktívnych príkazov, ako vidno v doplnenom okne na ďalšom obrázku



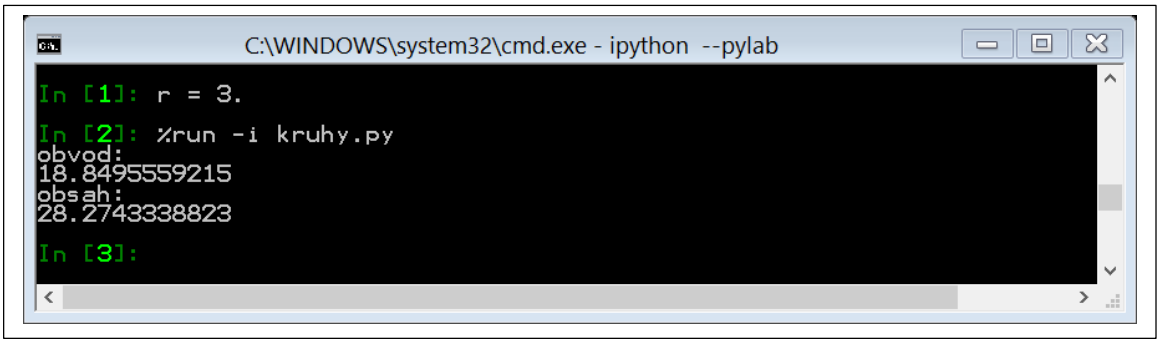
Čo sa vlastne stalo, prečo sa výsledky nevypísali, keď príkazy, ktoré sme museli znovu zadať ako riadky 3 a 4 boli v skripte kruhy.py uvedené?

Vysvetlenie je prosté. Ako sme si to už vysvetľovali, interaktívny Python riadok, ktorý neobsahuje príkaz = spracuje tak, že vypíše výsledok výpočtu na obrazovku, lebo vie, že ten výsledok sa nikam osobitne neuloží. Ibaže spustením príkazu %run –i sa dočasne opustí interaktívny mód. Príkazy skriptu sa vykonajú, ale nevypisujú sa na obrazovku, takže nevidíme, čo sa vlastne vykonalo. A teda sa ani nevypisuje výsledok výpočtu pravých strán riadkov neobsahujúcich príkaz =.

Ak chceme, aby sa niečo vypísalo na obrazovku aj počas neinteraktívneho behu skriptu, musíme to zariadiť osobitným príkazom print(). Doplníme preto skript kruhy.py o explicitné príkazy tlače, ako je to urobené na nasledujúcom obrázku.



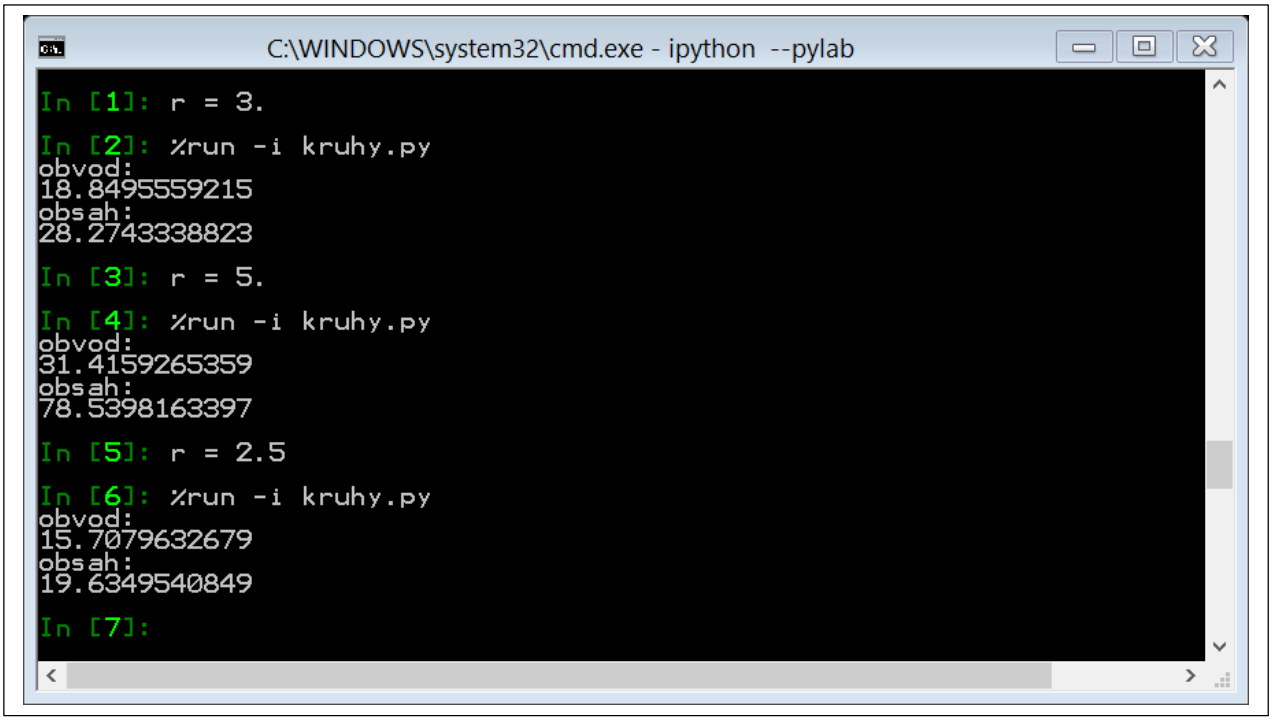
V tomto skripte príkaz z riadku 3 vypíše na obrazovku reťazec písmen (lebo sme použili úvodzovky, teda slovo obvod nie je meno premennej ale reťazec písmen). Príkaz v riadku 4 vypíše obsah premennej obvod, teda číslo, výsledok výpočtu obvodu. Reťazce sme si nechali vypísať na obrazovku, aby sme rozoznali, aké čísla sa vlastne na obrazovke vypisujú. Tu je výsledné použitie skriptu:



Poznamenajme ešte, že hodnotu premennej r, teda polomeru nezadávame v skripte, ale ju zadávame interaktívne pred vyvolaním skriptu. To je dôvodom, prečo príkaz %run používame s dodatočným parametrom –i, teda ako %run –i. **Parameter –i znamená, že skript má z interaktívneho mód prevziať všetky premenné, ktoré boli zavedené pred jeho vyvolaním**.

„Poznamenajme ešte, že prefix % v príkaze %run znamená že ide o tzv. magický príkaz. Magické príkazy sú písané do riadku po prompte tak, ako keby to boli príkazy jazyka Python. Ale nie sú. Sú to vlastne akési „metapríkazy“, ktoré riadia činnosť samotného interpretera. V tomto prípade run nariaďuje interpreteru opustiť interaktívny mód a zbehnúť príkazy skriptu.

Na záver tejto kapitolky ukážka opakovaného použitia skriptu pre výpočet viacerých kruhov:



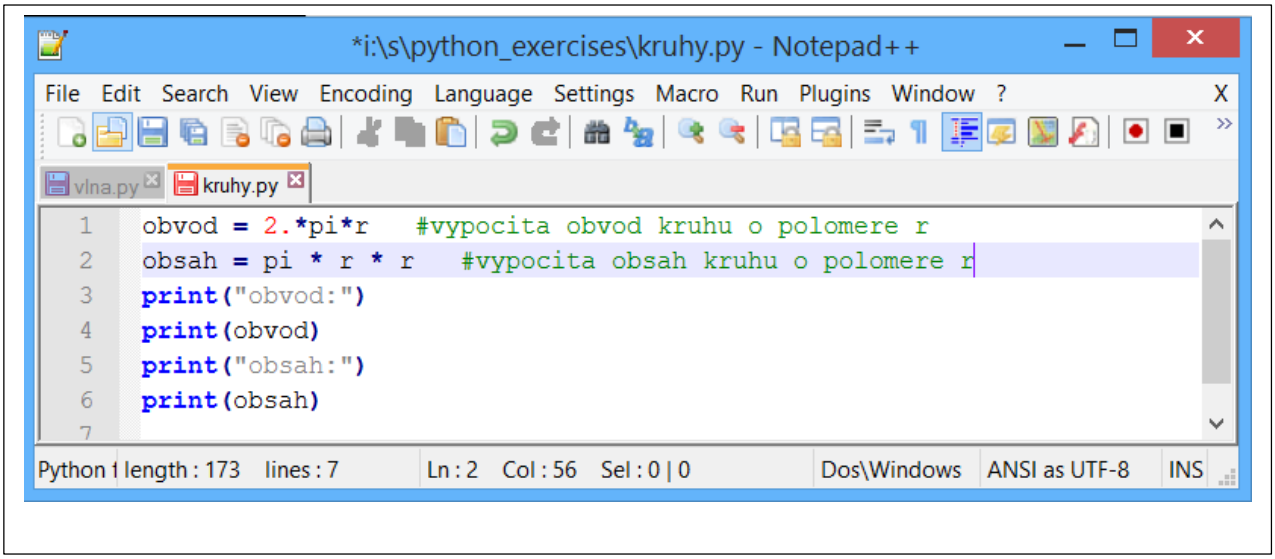
Poznamenajme ešte, že podobná funkčnosť (opakované použitie skupiny príkazov) sa profesionálnejšie robí definovaním funkcie: o tom trochu neskôr.

Skripty sa častejšie používajú vtedy, ak celý výpočet chceme vykonať neinteraktívnym spôsobom. Potom celý program pripravíme ako skript (napríklad pokus.py) a zbehneme ho príkazom

```
python pokus.py
```

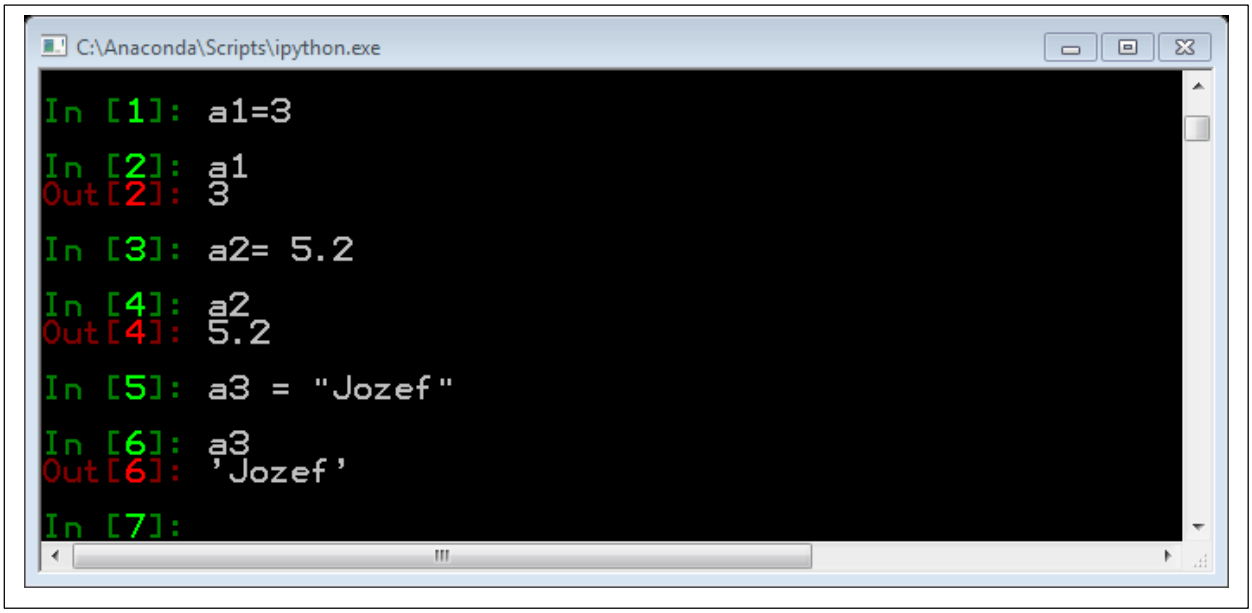
upozornime však, že v takom prípade musíme na začiatku importovať všetky potrebné moduly, lebo príkaz python nemôžeme spustiť s option stringom `--pylab` (ako sme spúšťali ipython). Podrobnejšie pozri v kapitole 10 Varia

Skript je spravidla určený pre viacnásobné použitie, niekedy si vyrobím skript a potom ho po rokoch znovu použijem. Ibaže spravidla zabudnem, ako má fungovať. Preto je užetočné do skriptu pridávať komentáre**. Ak v príkazovom riadku napíšem znak #, potom už všetko až do konca riadku Python ignoruje, považuje to za komentár.** Ako je to na ďalšom obrázku. Ešte poznámka: **v komentároch nepoužívajte diakritiku!!!** Nikdy neviete, kedy budete chcieť použiť skript pod iným operačným systémom, ktorý kóduje diakritiku inak ako Windows a spadnete do problémov.

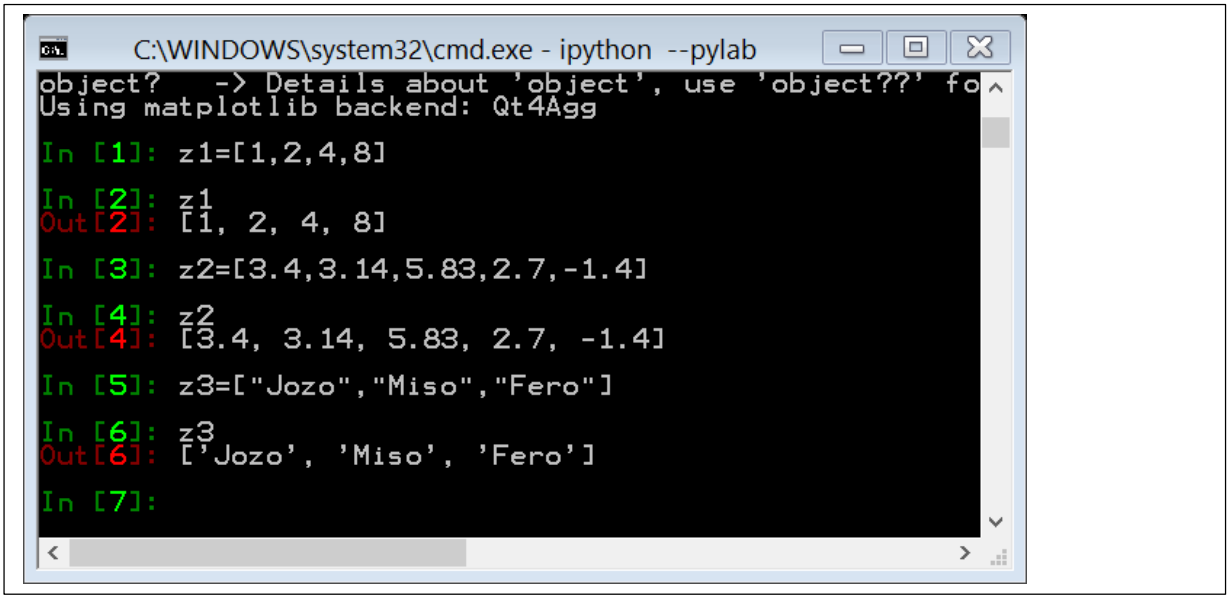


5. List, range

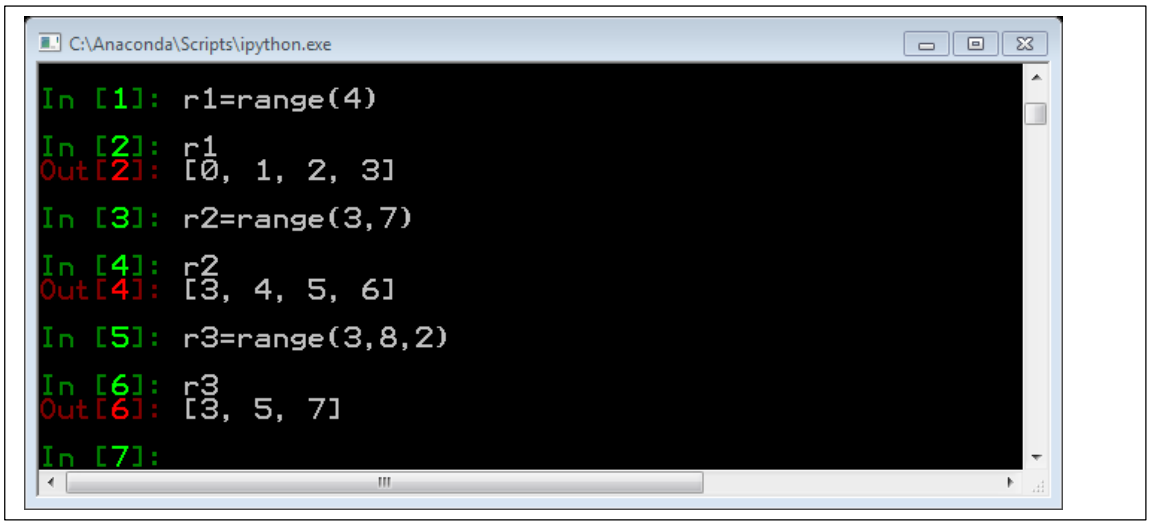
Videli sme doteraz, že v premennej (teda pomenovanej pamäťovej bunke) môže byť uložené nejaké číslo (integer alebo float) alebo reťazec písmen (string). Príklady:



Zoznámime sa teraz s novou dátovou štruktúrou, ktorá sa volá list (zoznam). Vo všeobecnosti to môže byť zoznam rôznych vecí. Na našej škôlkarskej úrovni budeme používať len zoznamy čísel alebo stringov. Zoznam sa zadáva tak, že medzi hranaté zátvorky sa vypíše skupina čísel alebo stringov oddelených čiarkami. Príklad

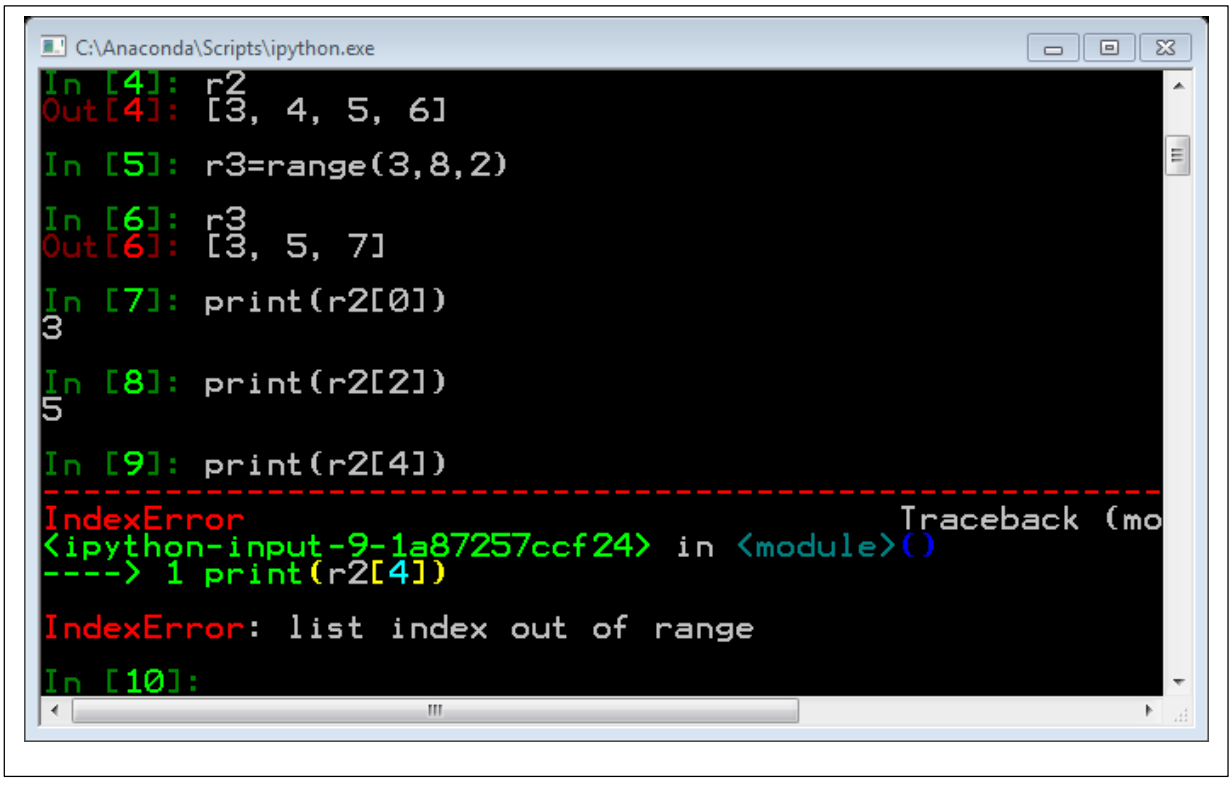


Zoznam obsahujúci ariotmetickú postupnosť celých čísel môžeme vytvoriť i užitočným príkazom range(). Príklady:



Príkaz range() môže mať až 3 celočíselné parametre s významom „od“, „až po“ „s krokom“, pričom hodnota „až po“ sa už do zoznamu nedostane, takže možno presnejší význam toho parametra by bol „ale menšie než“. Ak sa neuvedie krok, hodnota kroku bude defaultovo. Ak sa uvedie len jeden parameter, interpretuje sa ako „až po“.

V programe sa môžem odvolávať na jednotlivé členy zoznamu použitím indexu (poradového čísla v zozname), pričom indexácia začína nulou. Príklad

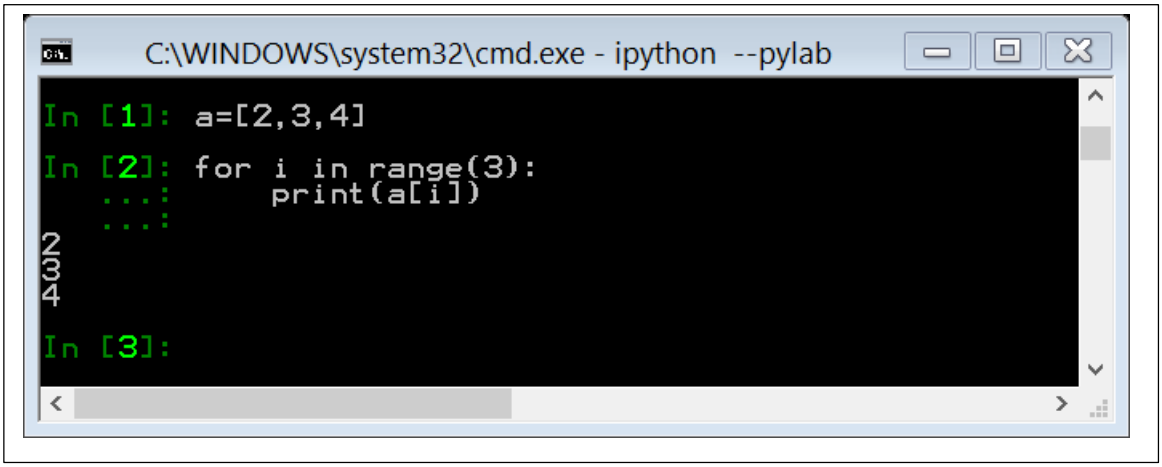


Príklady indexového prístupu sú v riadkoch 7 a 8, pričom v riadku 9 sme náročky urobili chybu, že sme sa odvolávali na neexistujúci index a dostali sme typickú chybovú hlášku. Slangovo sa tomu hovorí prešvihnúť rozsah zoznamu a je to častá programátorská chyba.

6. Cyklus for

Pozorne si preštuduje (ešte lepšie sami na klávesnici vyťukajte) všetky príklady, potom už vám dôjde aj bez detailného výkladu.

Veľmi častá je programátorská úloha zopakovať nejaký príkaz (alebo skupinu príkazov) viackrát, možno iba s obmenou nejakého parametra. Napríklad chceme vypísať po jednom všetky prvky nejakého poľa. Tu je program



Najprv pár slov o tom, ako sa to z klávesnice vyťukáva. Na prompt In[2] vyťukám

```
for i in range(3):
```

a po stlačení Enter sa neobjaví normálny riadkový prompt ale

```
pokračovací prompt ...:
```

znamená to, že príkaz for zatiaľ nie je úplný, nedá sa vykonať, čaká sa na dodatočné údaje, v tomto prípade sa čaká na príkazy, ktoré treba opakovane vykonať.

Všimnime si, že interaktívny Python aj odrazí pokračovací riadok doprava, aby sa nezačal písať hneď od začiatku riadku. Hovorí sa tomu indent (riadok sa indentuje). V jazyku Python indentovanie riadku má interpretačný význam, indentované riadky sa chápu ako jeden blok programu, v našom prípade všetky indentované príkazy sa vykonajú opakovane ako „telo cyklu for“.

V interaktívnom Pythone je indentácia do značnej miery zautomatizovaná. Pri písaní skriptu musí programátor indentáciu zabezpečiť sám. **Ale pozor, dobrá rada je nepoužívať na indentovanie kláves Tab. A najmä nie miešať pri indentácii znak medzery a Tab. Hrozí totiž, že interpreter zle vyhodnotí úroveň indentácie!**

My sme použili len jeden príkaz print(). Po jeho zadaní sa objaví nový pokračovací prompt, na ktorý keď odpoviem stlačením Enter (vlastne zadám prázdny riadok, to je dôležité v skripte, kde nemám prompty) je to pre interpreter signál že príkaz for sa tým ukončil a možno všetky zadané riadky interpretovať (vykonať).

Teraz čo sa deje v cykle for.

Riadok

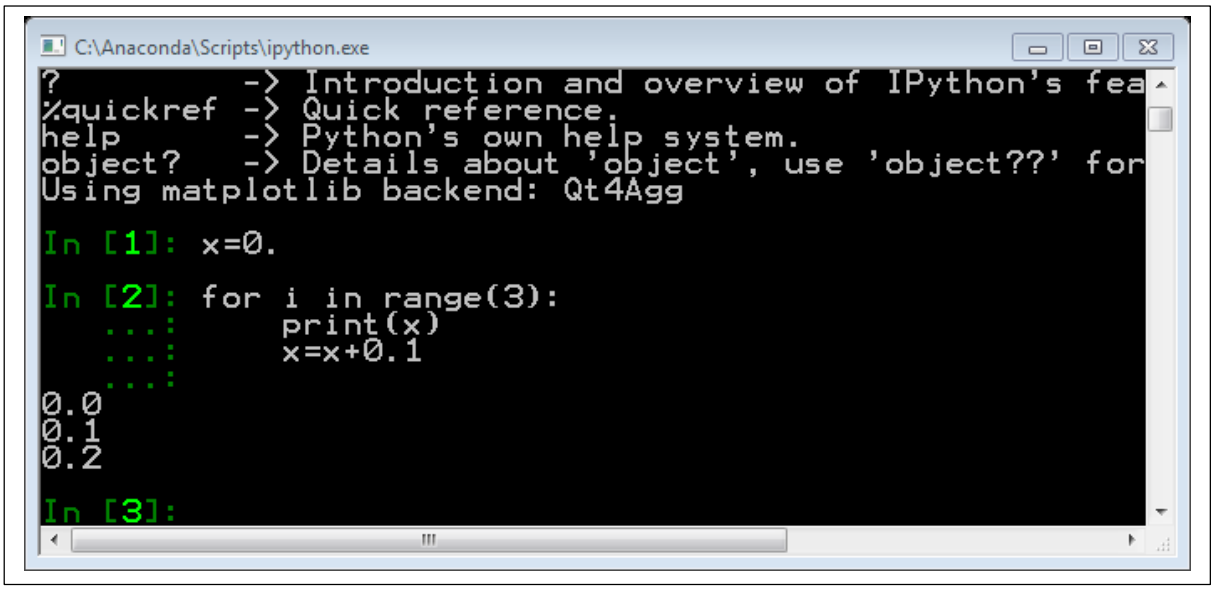
```
for i in range(3)
```

vlastne znamená

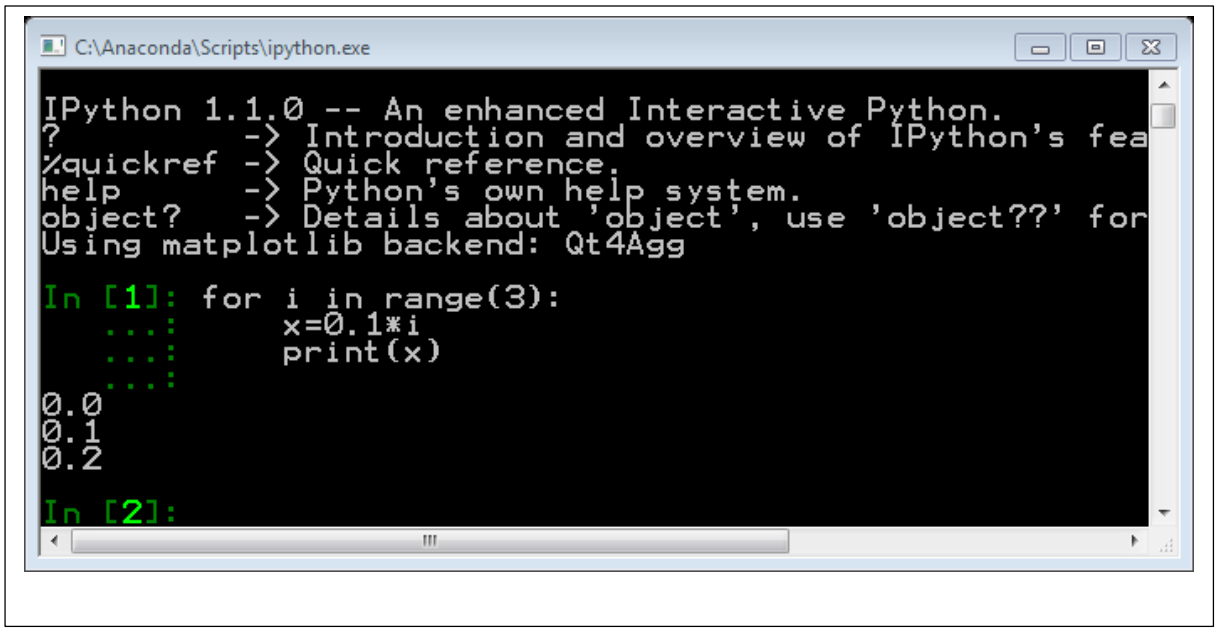
```
for i in [0, 1, 2]
```

a znamená to, že celý blok nasledujúcich indentovaných riadkov sa bude opakovane vykonávať tak, že sa postupne použijú ako i všetky hodnoty z uvedeného zoznamu. Čo aj vidíme, že sa stalo.

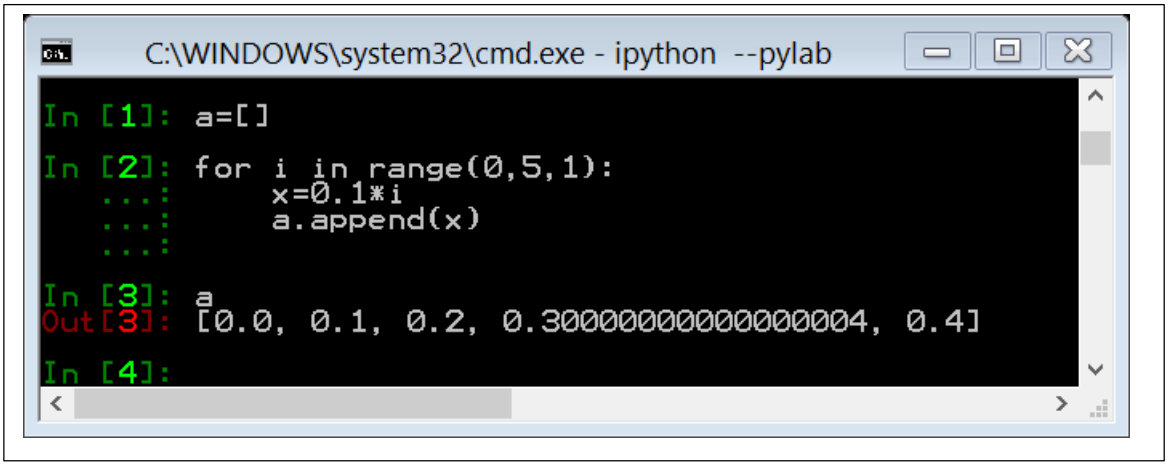
Podstatné na cykle for je, že dopredu vieme, koľkokrát sa má „jeho telo“ vykonať. Premenná i nemusí byť prípadne použitá vôbec, namiesto nej si vnútri tela cyklu môžem vyrábať inú postupne sa meniacu premennú. Napríklad takto:



Alebo napríklad takto:



Ako príklad použitia cyklu for uveďme, ako môžeme naplniť list potrebnými hodnotami

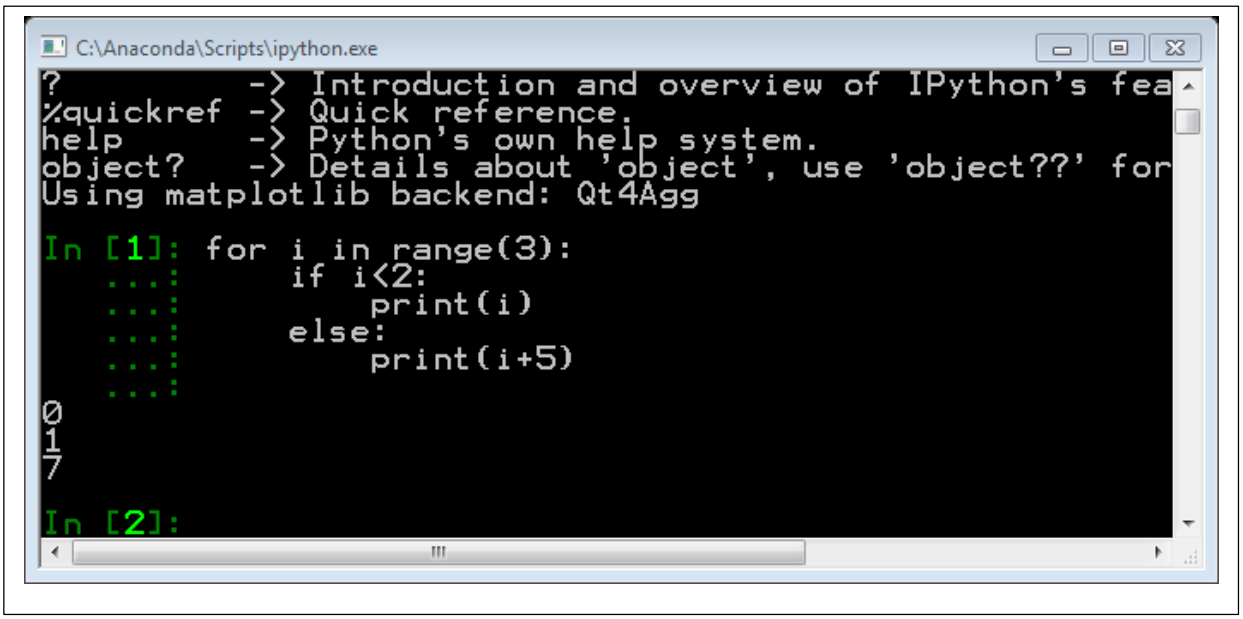


Poznámky:

- Niekedy je naozaj nepraktické, keby sme mali naplniť zoznam ručne vypísaním všetkých jeho prvkov, pričom existuje jednoduchý algoritmus ak ich postupne počítať.
- V riadku 1 inicializujeme zoznam uložený do premennej a ako prázdny zoznam, teda neobsahujúci žiadny prvok. Je to preto, aby Python „vedel“ že v premennej a je uložený zoznam a vedel, že sa môže použiť príkaz a.append(). Keby a bola napríklad premenná, v ktorej je uložené číslo typu integer, príkaz a.append() by sa nedal použiť (presnejšie neexistoval by)
- Riadok Out[3] je poučný, hodnotu a[4] sme naplnili ako 0.3 ale vypísala sa hodnota 0.30000000000000004. To neznamená žiadnu chybu. Je to dôsledok toho, že počítač používa na uschovanie čísel do pamäte dvojkovú číselnú sústavu a nie desiatkovú. A číslo 0.3 sa nedá presne reprezentovať dvojklovým číslom s konečným počtom cifier.

7. If – else: logické vetvenie programu

Preštudujte si najprv nasledujúci príklad

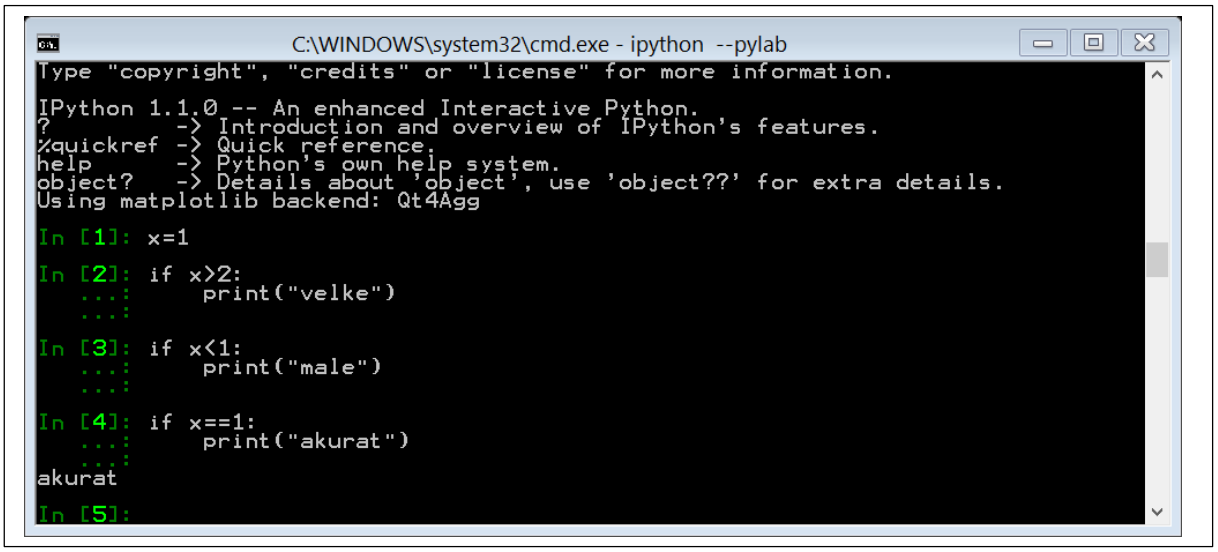


Práve ste videli príklad logického vetvenia, ktorý hovorí, že ak je splnená nejaká podmienka má sa vykonať čosi, v opačnom prípade sa má vykonať niečo iné.

Poznamenajme ešte k vyťukávaniu tohto príkladu v interaktívnom Pythone. Po napísaní príkazu if i<2: sa automaticky nastaví dvojitá indentácia nasledujúceho riadku print(i), lebo sa očakávajú riadky bloku, ktorý sa má celý vykonať v prípade, že i<2. Takže dvojitá indentácia pokračuje aj do nasledujúceho riadku, lebo interpreter netuší, že už chcem písať else. Preto musím opakovaným stláčaním klávesu Backspace zmeniť dvojitú indentáciu na jednoduchú, napísať else: a po stlačení Enter dostanem opäť dvojitú indentáciu bloku príkazov pre prípad „else“. Tú ukončím zadaním prázdneho riadku, ktorá ukončí aj zadávanie cyklu.

Prázdny riadok ako reakcia na pokračovací prompt proste znamená, že ďalšie pokračovanie už nebude a riadky ako boli zadané už majú byť interpretovateľné a vykonané.

Blok if príkazov nemusí byť sprevádzaný príslušným else blokom (čo znamená že ak testovacia podmienka je pravdivá vykoná sa if blok, ak je nepravdivá, nevykoná sa nič a program bude pokračovať ďalším (už neindentovaným) riadkom. Príklad



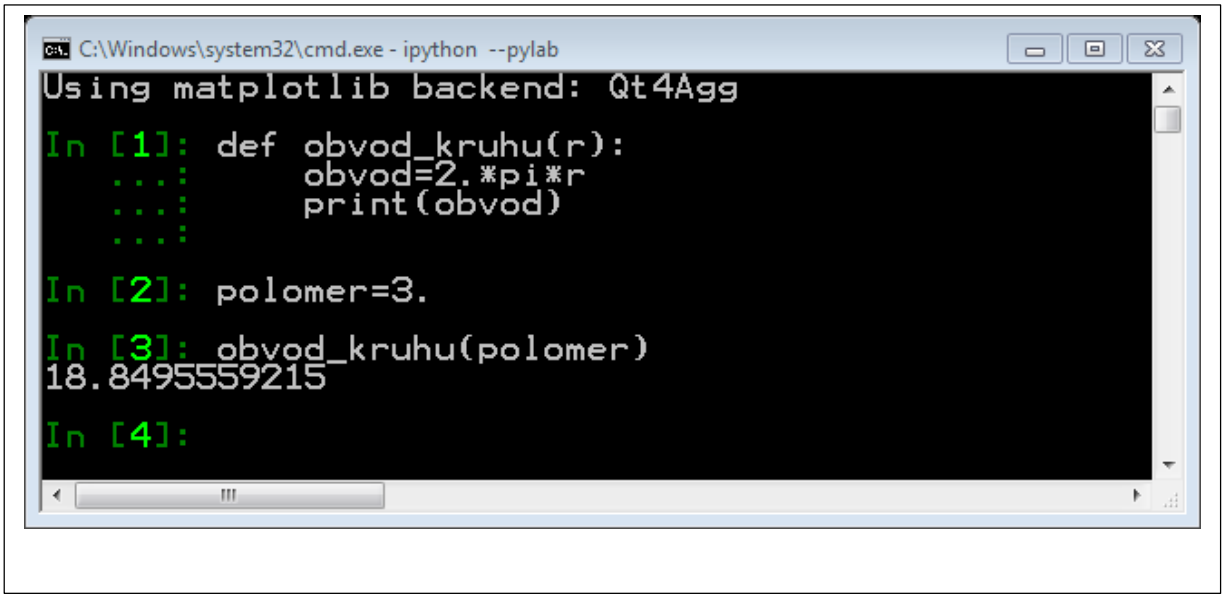
Všimnite si že test na rovnosť sa vykonáva zdvojeným znakom ==. Jednoduché = je totiž vyhradené pre príkaz uloženia výsledku z pravej strany do premennej na ľavej strane.

8. Funkcie

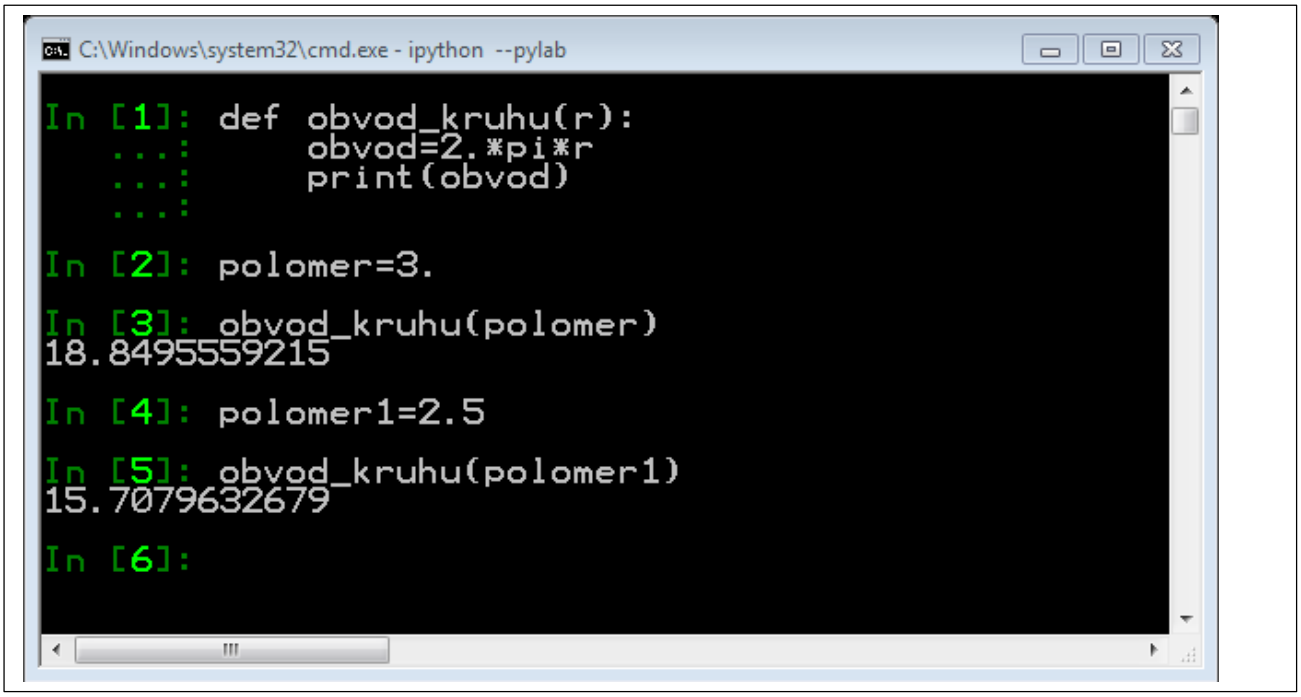
8.1. Používateľom definované funkcie

Už sme si ukázali, že ak potrebujeme viackrát opakovať nejaký výpočet, môžeme to zabezpečiť pomocou skriptu. Skript však prerušuje bežiaci proces interpretácie a začne nový beh interpretácie, a to interpretácie riadkov zapísaných v skripte. Elegantnejšie riešenie je pomocou definovania funkcií podobných ako sú matematické funkcie, napríklad sin().

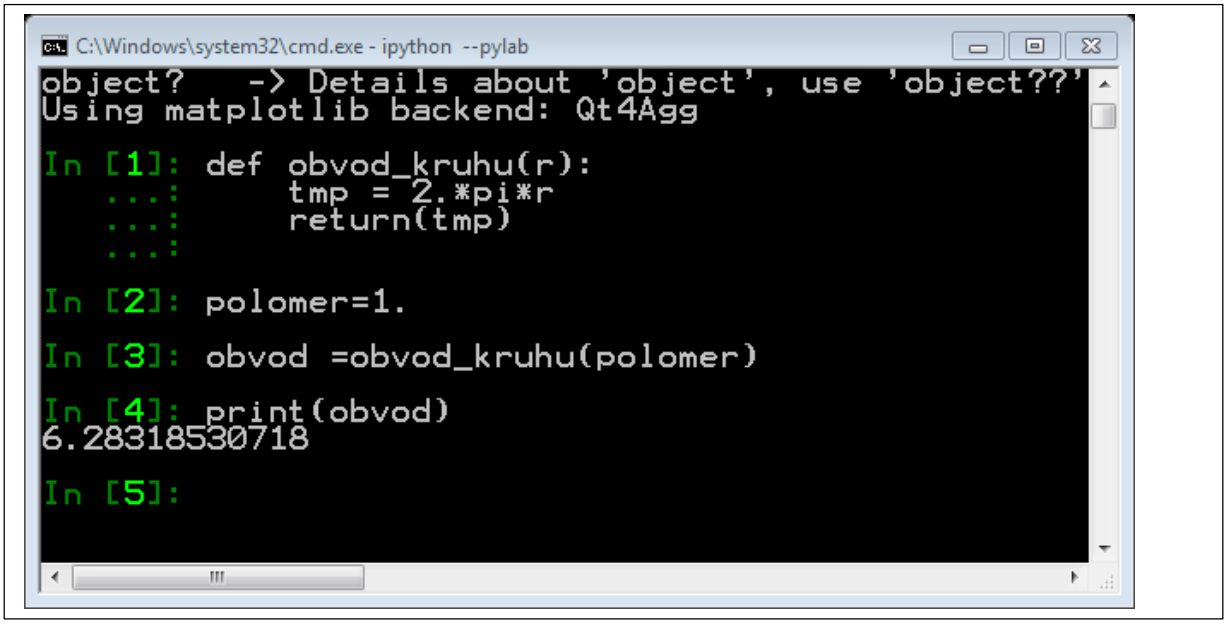
Python dovoľuje definovať si vlastné funkcie a nazvať si ich ľubovoľne. Len si treba dať pozor, aby nevznikol konflikt mien. Nie je dobre nazvať si súkromnú funkciu ako sin(). Python by už v tom behu nevedel vypočítať „normálny sínus“. Príklad definície a použitia funkcie:



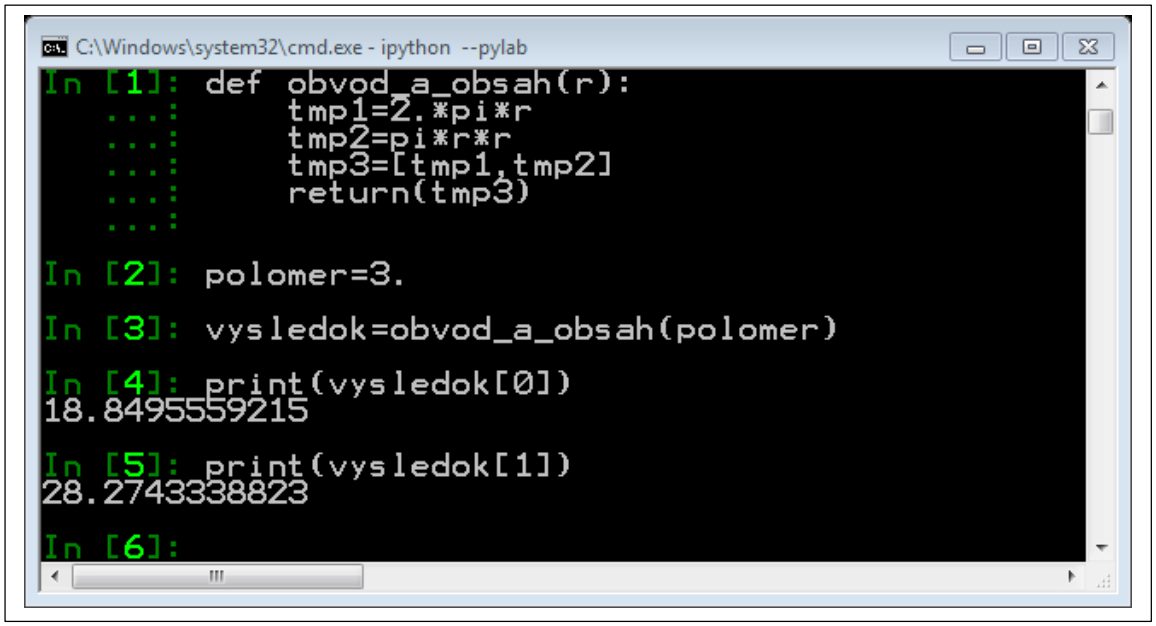
Podumajte nad tým jednoduchým príkladom a sami si urobte nejakú hypotézu, ako to celé funguje. V bloku riadkov 1 je definovaná funkcia, čosi na spôsob: zober premennú r a urob s ňou to, čo hovoria riadky „v tele funkcie“. Kľúčom pre pochopenie je to, že premenná r v definícii funkcie (hovorí sa tomu parameter r) má len zástupnú úlohu, lebo v čase, keď definujem funkciu ešte neviem ako sa bude volať premenná, v ktorej budem mať uloženú hodnotu polomeru. V riadku 3 je príkaz, ktorému sa hovorí „volanie funkcie“, kde už mám v zátvorke ako parameter uloženú premennú „so skutočným polomerom“. Samozrejme, tú istú funkcie môžem volať viackrát s rozličnými premennými ako parametrom. Príklad:



V uvedenom príklade som výsledok výpočtu „dostal von z tela funkcie tak, že som ho nechal vypísať príkazom print(). To je nepraktické, keď by som chcel s tým výsledkom ďalej niečo robiť. Funkcia môže aj „vrátiť nejakú hodnotu“ ak v jej tele použijem príkaz return(). V takom prípade môže volanie funkcie stáť na pravej strane príkazu = a vrátenú hodnotu môžem uložiť do premennej, ktorá stojí na ľavej strane =. Príklad:



Môžem dokonca vrátiť aj viac hodnôt, ak vrátim šikovne vyrobený zoznam (list). Príklad:



Ešte poznámka pre „naozajstných programátorov“. Môžete sa spýtať, či v Pythone sa parametre volajú referenciou alebo hodnotou. Odpoveď je, že ani tak ani tak. Prečítajte si o tom napríklad tu:

https://www.jeffknupp.com/blog/2012/11/13/is-python-callbyvalue-or-callbyreference-neither/

8.2. Preddefinované funkcie

Python v režime pylab má mnoho preddefinovaných funkcií. V nasledujúcich tabuľkách je (neúplný) zoznam

add(x1, x2[, out])	Add arguments element-wise.
subtract(x1, x2[, out])	Subtract arguments, element-wise.
multiply(x1, x2[, out])	Multiply arguments element-wise.
divide(x1, x2[, out])	Divide arguments element-wise.
logaddexp(x1, x2[, out])	Logarithm of the sum of exponentiations of the inputs.
logaddexp2(x1, x2[, out])	Logarithm of the sum of exponentiations of the inputs in base-2.
true_divide(x1, x2[, out])	Returns a true division of the inputs, element-wise.
floor_divide(x1, x2[, out])	Return the largest integer smaller or equal to the division of the inputs.
negative(x[, out])	Numerical negative, element-wise.
power(x1, x2[, out])	First array elements raised to powers from second array, element-wise.
remainder(x1, x2[, out])	Return element-wise remainder of division.
mod(x1, x2[, out])	Return element-wise remainder of division.
fmod(x1, x2[, out])	Return the element-wise remainder of division.
absolute(x[, out])	Calculate the absolute value element-wise.
rint(x[, out])	Round elements of the array to the nearest integer.
sign(x[, out])	Returns an element-wise indication of the sign of a number.
conj(x[, out])	Return the complex conjugate, element-wise.
exp(x[, out])	Calculate the exponential of all elements in the input array.
exp2(x[, out])	Calculate *2**p* for all *p* in the input array.
log(x[, out])	Natural logarithm, element-wise.
log2(x[, out])	Base-2 logarithm of *x*.
log10(x[, out])	Return the base 10 logarithm of the input array, element-wise.
expm1(x[, out])	Calculate `exp(x) - 1` for all elements in the array.
log1p(x[, out])	Return the natural logarithm of one plus the input array, element-wise.
sqrt(x[, out])	Return the positive square-root of an array, element-wise.
square(x[, out])	Return the element-wise square of the input.
reciprocal(x[, out])	Return the reciprocal of the argument, element-wise.
ones_like(a[, dtype, order, subok])	Return an array of ones with the same shape and type as a given array.

All trigonometric functions use radians when an angle is called for. The ratio of degrees to radians is $180^\circ/\pi$.

sin(x[, out])	Trigonometric sine, element-wise.
cos(x[, out])	Cosine element-wise.
tan(x[, out])	Compute tangent element-wise.
arcsin(x[, out])	Inverse sine, element-wise.
arccos(x[, out])	Trigonometric inverse cosine, element-wise.
arctan(x[, out])	Trigonometric inverse tangent, element-wise.
arctan2(x1, x2[, out])	Element-wise arc tangent of `x1/x2` choosing the quadrant correctly.
hypot(x1, x2[, out])	Given the "legs" of a right triangle, return its hypotenuse.
sinh(x[, out])	Hyperbolic sine, element-wise.
cosh(x[, out])	Hyperbolic cosine, element-wise.
tanh(x[, out])	Compute hyperbolic tangent element-wise.
arcsinh(x[, out])	Inverse hyperbolic sine element-wise.
arccosh(x[, out])	Inverse hyperbolic cosine, element-wise.
arctanh(x[, out])	Inverse hyperbolic tangent element-wise.
deg2rad(x[, out])	Convert angles from degrees to radians.
rad2deg(x[, out])	Convert angles from radians to degrees.

Podrobnejšiu dokumentáciu nájdete na stránke

http://docs.scipy.org/doc/numpy/reference/ufuncs.html#available-ufuncs

Poznamenajme, že matematické funkcie sa vyskytujú nielen v module numpy (ktorý je automaticky importovaný v režime pylab ale aj v module math, ktorý ale defaultovo importovaný nie je.

8.3. Náhodné čísla

Pre naše experimentovanie budeme ešte potrebovať generátor náhodných čísel. V rámci modulu numpy sa vyskytuje (pod)modul random, ktorý obsahuje viacero typov náhodných generátorov. My budeme používať generátor random.random(). Tu je príklad opakovaného volania:

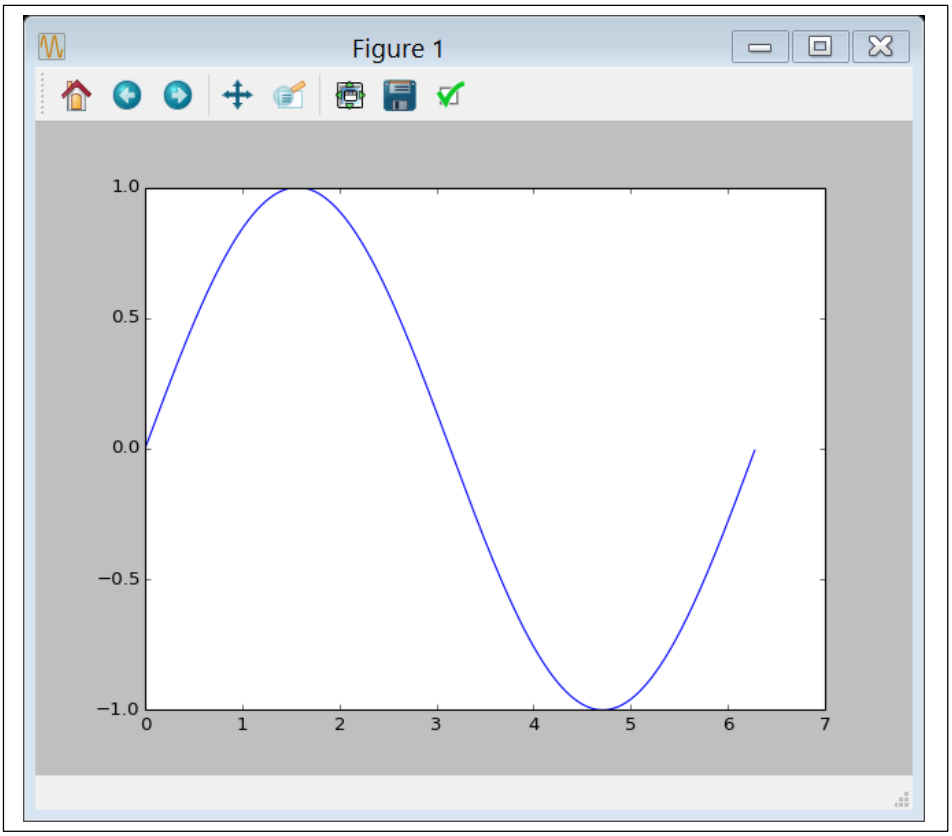
```
In [1]: random.random()
Out[1]: 0.04289660907690551

In [2]: random.random()
Out[2]: 0.11575833879847375

In [3]: random.random()
Out[3]: 0.34691256575300744

In [4]: random.random()
Out[4]: 0.1007055076375275

In [5]: random.random()
Out[5]: 0.5410739556813356

In [6]:
```

9. Kreslenie grafov

Úplne najjednoduchšie sa dá graf nakresliť takto

```
In [1]: x=[]

In [2]: for i in range(0,1000):
   ...:     x.append(0.001*i*2.*pi)
   ...:

In [3]: y=sin(x)  #toto vypocita sinusy vsetkych hodnot v liste x

In [4]: plot(x,y)
Out[4]: [<matplotlib.lines.Line2D at 0x9bde2b0>]

In [5]:
```

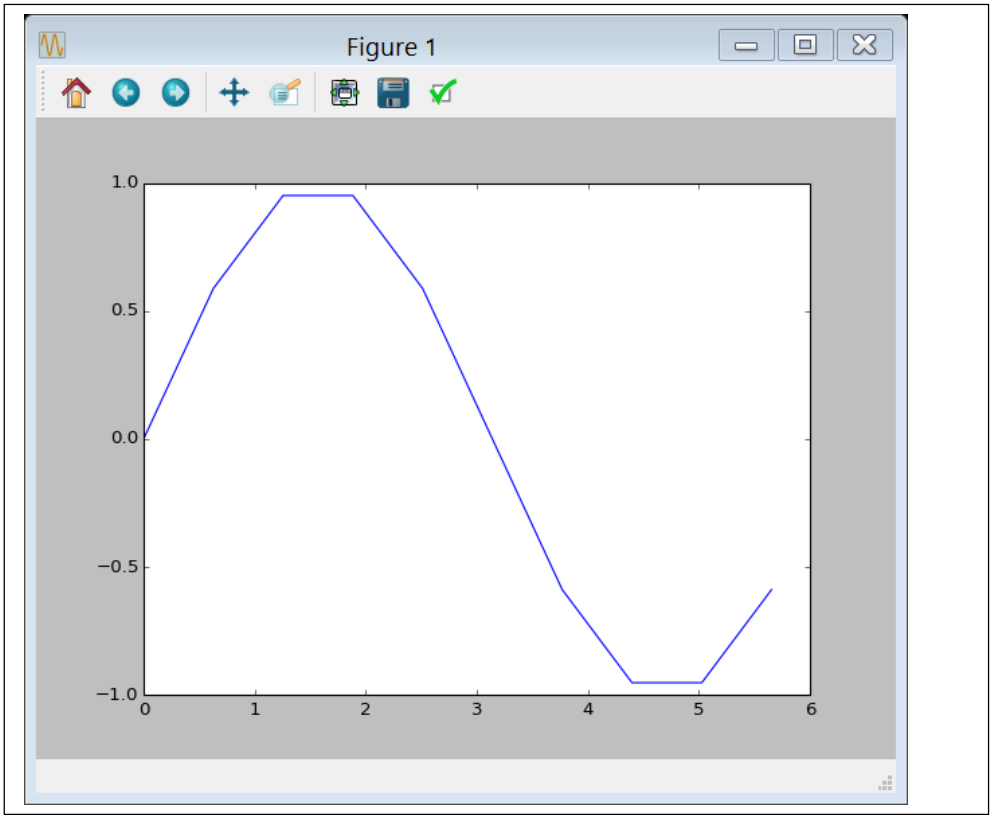
Nakreslí to takýto graf



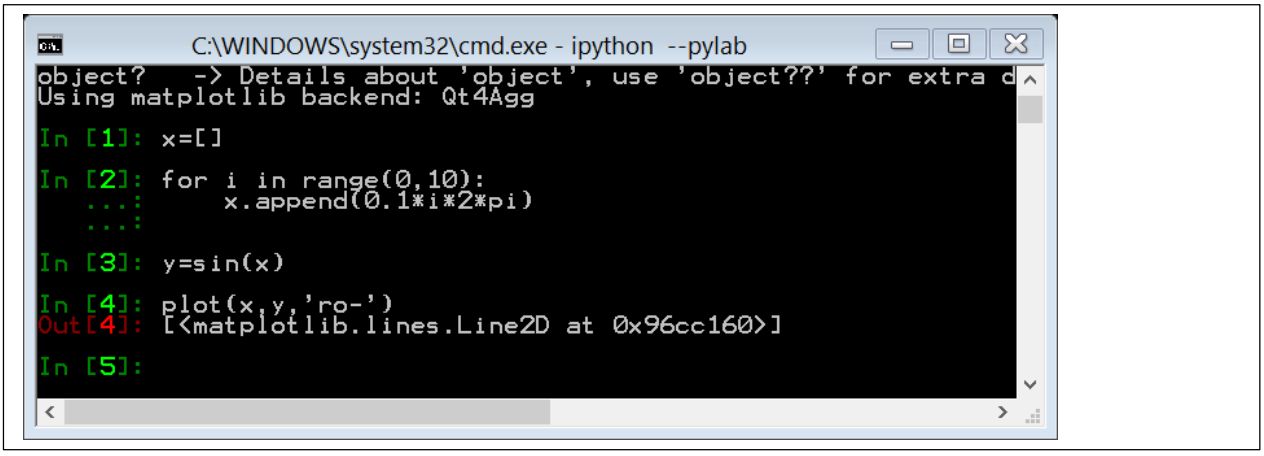
Premyslite si, čo sme to urobili. V riadkoch 1 a 2 sme naplnili array x hodnotami premennej z intervalu (0., 2*pi) rozdeleného na 1000 kúskov. V riadku 3 sme jedným príkazom vypočítali sínusy všetkých hodnôt v array x. Vznikol tak array y. Polia x a y obsahujú sebe zodpovedajúce páry hodnôt x a sin(x). Je to vlastne 1000 bodov v rovine xy. Príkaz plot nakreslí v rovine xy tie body a aj ich pospája malými úsečkami. Aby sme to lepšie videli, urobíme to isté s menej bodmi. Napríklad

```
In [1]: x=[]

In [2]: for i in range(0,10):
   ...:     x.append(0.1*i*2.*pi)
   ...:

In [3]: y=sin(x)  #toto vypocita sinusy vsetkych hodnot v liste x

In [4]: plot(x,y)
Out[4]: [<matplotlib.lines.Line2D at 0x9bc9da0>]

In [5]:
```

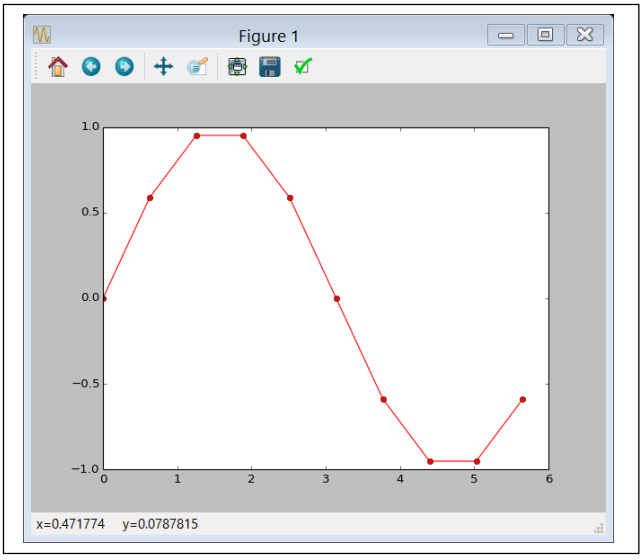
Vyrobí to takýto graf



Ukázali sme si zatiaľ najjednoduchšie kreslenie grafov, všetko rozhodovanie (rozsah hodnôt na osiach, farba grafu, spôsob kreslenia) sme ponechali na defaulty. Samozrejme, môžeme prevziať rozhodovanie na seba a zadať dopňujúce paramtere. Farba grafu a spôsob kreslenia za zadáva ako voliteľný formátovací reťazec, môže to vyzerať napríklad takto:



Nakreslí to takýto graf:



všimnime si formátovací string ‘ro-‘ v príkaze plot v riadku 4. String obsahuje tri znaky r je príkaz pre farbu (red, červená), o je príkaz pre kreslenie markerov a – je príkaz pre štýl čiary. Marker je symbol, ktorý ploter nakreslí za každý zobrazený bod, konkrétne o znamená krúžok. Ak zadáme aj príkaz pre štýl čiary, ploter nenakreslí iba body ale ich pospája úsečkami. Konkrétne symbol – znamená „použi plnú čiaru“. Ak nezadáme žiadny formátovací string (tak ako sme to videli v prvom príklade) je to to isté ako by sme zadali ‘-‘. Znamená to „použi defaultovú modrú farbu, nevykresľuj body pomocou markerov a pospájaj ich plnou čiarou.

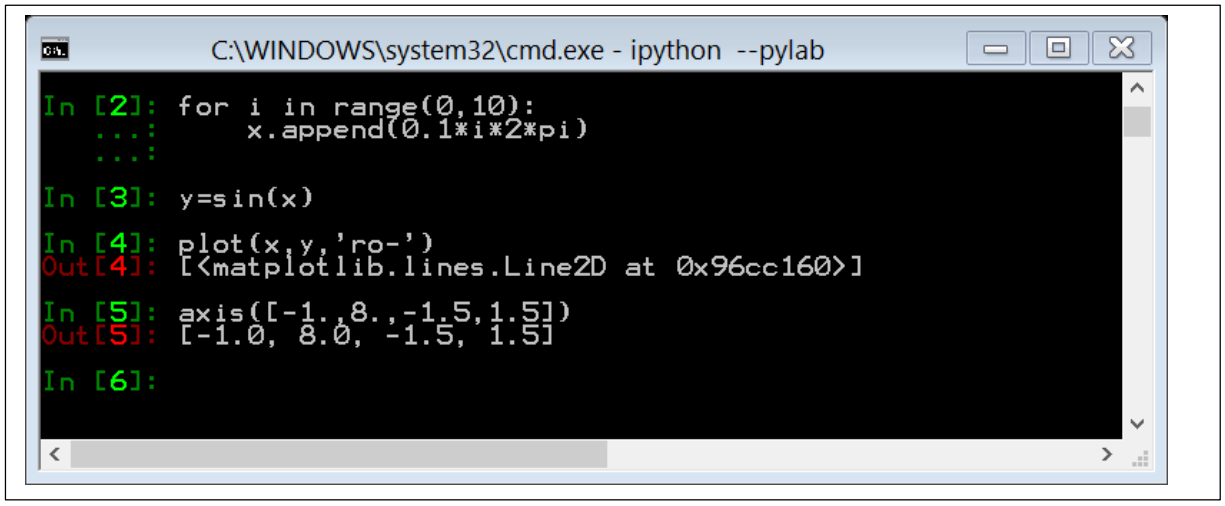
Tu sú príklady pre symboly rôznych farieb, markerov a čiarových štýlov.

character	description
'-'	solid line style
'--'	dashed line style
'-.'	dash-dot line style
':'	dotted line style

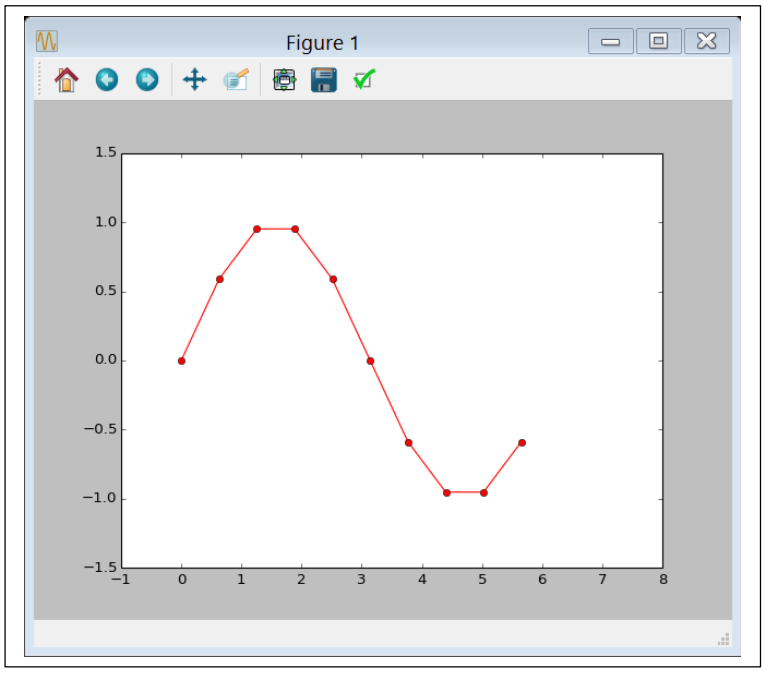
character	color
‘b’	blue
‘g’	green
‘r’	red
‘c’	cyan
‘m’	magenta
‘y’	yellow
‘k’	black
‘w’	white

character	description
'-'	solid line style
'--'	dashed line style
'-.'	dash-dot line style
':'	dotted line style
'.'	point marker
','	pixel marker
'o'	circle marker
'v'	triangle_down marker
'^'	triangle_up marker

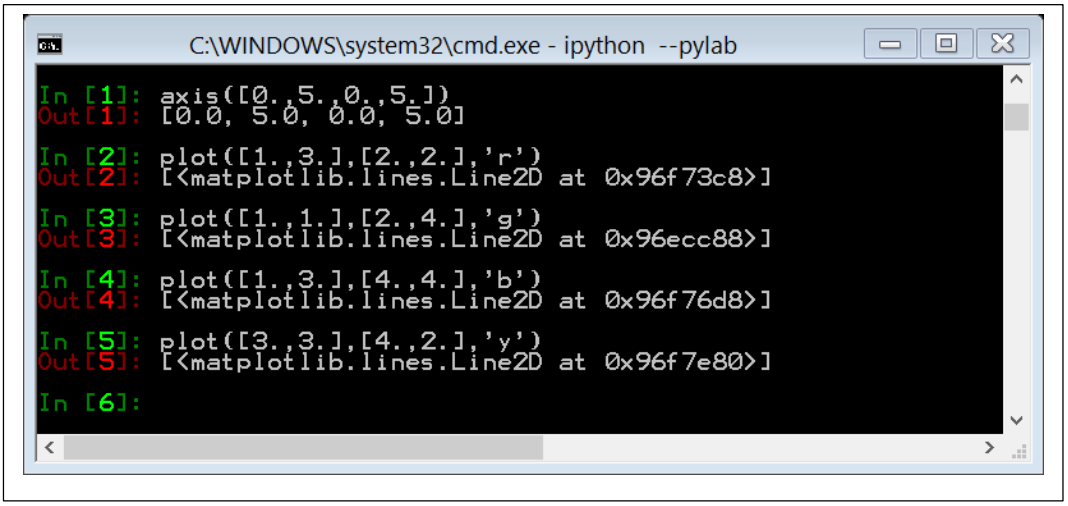
Ak chceme sami zvoliť rozsahy hodnôt na osiach, musíme zadať (najlepšie pred príkazom plot) príkaz axis. Príkaz axis môžeme zadať aj dodatočne, po príkaze plot. Tu je príklad



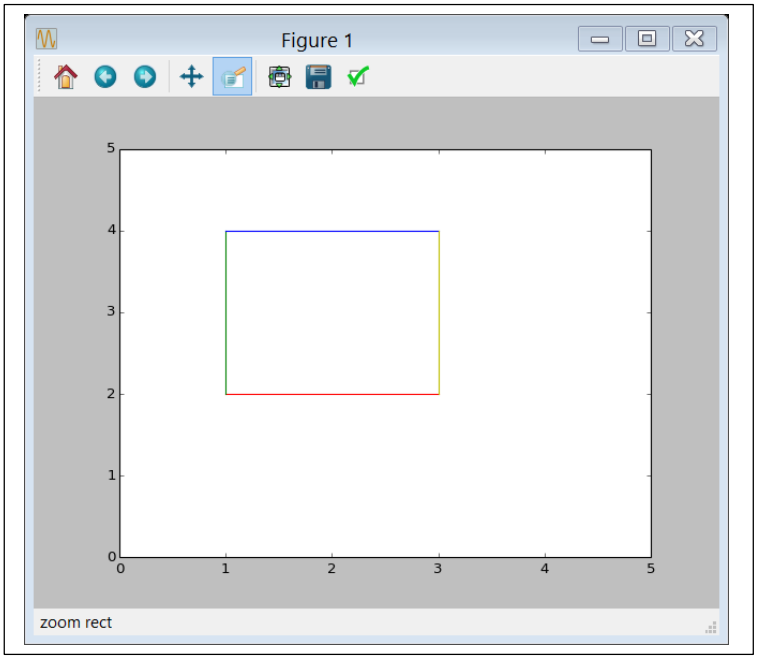
Parametrom príkazu axis je list obsahujúci 4 čísla v poradí ľavá hranica vodorovnej osi, pravá hranica vodorovnej osi, spodná hranica zvislej osi, horná hranica zvislej osi. Obrázok vyzerá potom takto:



Všetko, čo budeme potrebovať nakresliť na našej „škôlkárskej úrovni“ zvládneme pomocou príkazu plot. Príkazom plot totiž možnmo ľahko nakresliť ľubovoľnú úsečku a teda aj všetko, čo sa dá pomocou úsečiek zobraziť. Tu je príklad ako nakresliť „štvorček“.



Nakreslí to toto:



Aby sme lepšie identifikovali ktorý riadok nakreslí ktorú úsečku, použili sme štyri rôzne farby.

Poznamenajme, že možnosti ako použiť prostriedky poskytované v Pythone systémom matplotlib sú mimoriadne bohaté. Presahujú ale našu „škôlkársku úroveň“. Dokumentácia na webe nie je dokonalá ale programátor, ktorý si dá námahu a spolu s čítaním dokumentácie aj na počítači s grafikou experimentuje, nakoniec zvládne aj komplikované úlohy.

10. Varia

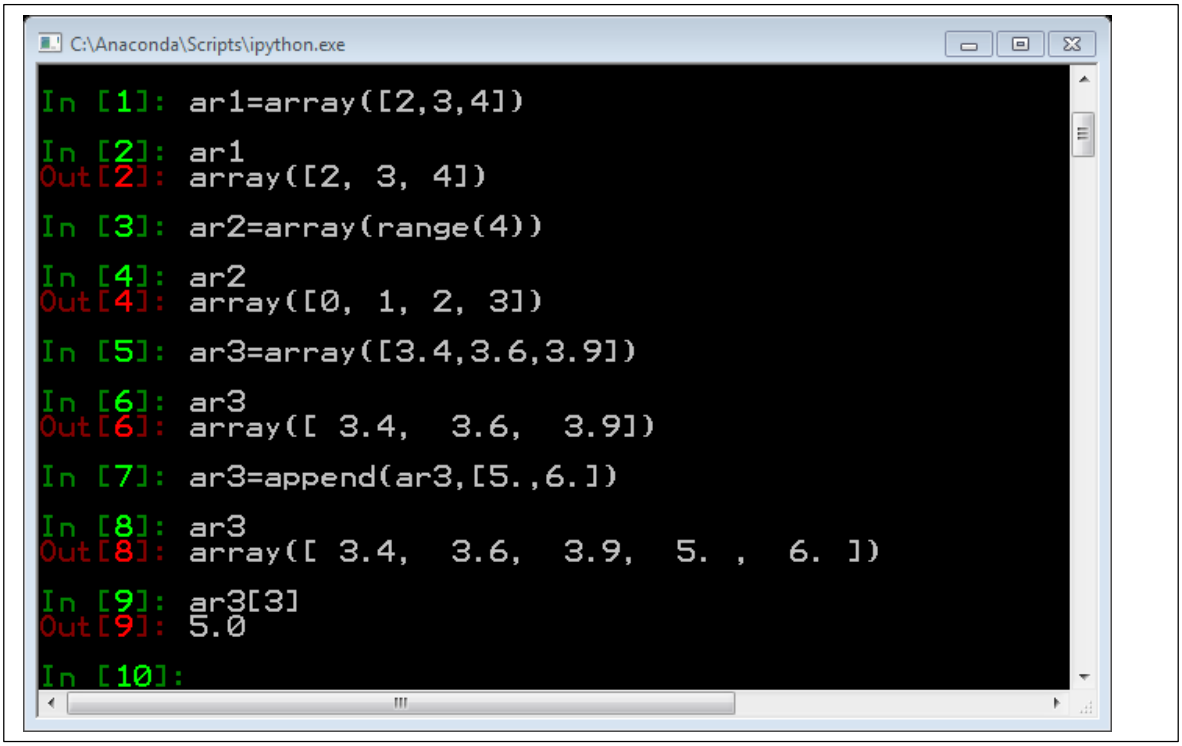
10.1. Globále a lokálne premenné

Python považuje všetky premenné, použité vnútri tela funkcie sa považujú za lokáne premenné, teda žijúce len počas práce funkcie, ale neviditeľné zvonku. A to aj vtedy ak nejaká premenná rovnakého mena už bola použitá globálne, teda mimo tela nejakej funkcie. Ak chceme použiť vnútri funkcie globálne definovanú premennú, musím to explicitne povedať príkazom global

Všimnite si rozdiel, funkcia zero90 nijako neovplyvnila globálnu premennú a, lebo nebola označená ako global vnútri tela funkcie. Oproti tomu funkcia correctzero() ovplyvnila globálnu premnnú a, lebo sme ju ako globálnu označili.

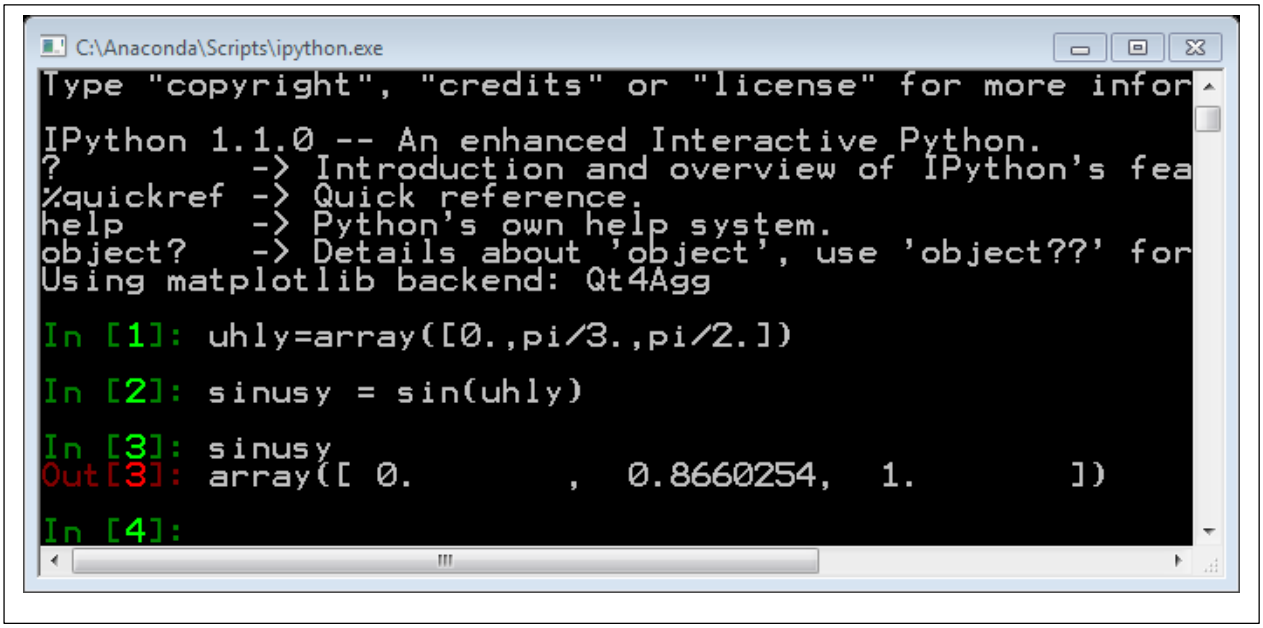
10.2. Arrays

V „matematickom Pythone (pylab) používame častejšie „zoznamy“ čísel konštruované nie ako list ale ako array (pole). Rozdiel je okrem formálne inej konštrukcie najmä vnútri Pythonu, ktorý so zoznamami typu array efektívnejšie narába pri numerických výpočtoch, takže čas výpočtu sa niekedy môže skrátiť aj 10x . Pre užívateľa je rozdiel v tom, ako sa array inicializuje a ako graficky vyzerá výpis jeho obsahu. Príklad



Vidíme, že array sa inicializuje príkazom array() a ako parameter sa použije vhodný list. Príkaz array vlastne vyrobí z obyčajného zoznamu nový zoznam typu array. V riadku 7 súčasne vidíme príklad, ako sa do existujúceho poľa pridajú (na koniec) ďalšie prvky. V riedku 9 vidíme, že prvky poľa sa dajú „oslovovať“ jednotlivo indexovaním.

Podobne ako list, aj array sa dá aj šikovne využiť, ak potrebujeme vykonať rovnakú operáciu so všetkými prvkami v poli. Napríklad vypočítať sínus každého prvku. príklad



array je konštrukcia definovaná v knižničnom balíku NumPy (importuje ho automaticky - - pylab). Záujemca o detaily si musí pozrieť dokumentáciu k tomuto balíku. V balíku je definovaných viac užitočných funkcií pre manipuláciu s poliami typu array, napríklad užitočná funkcia arange(), to je obdoba funkcie range pre listy. namiesto aby sme pole kreovali tak, že napred vykreujeme list príkazom range a potom pole, teda

```
In [5]: xlist=range(2,6,2)

In [6]: ar=array(xlist)

In [7]: ar
Out[7]: array([2, 4])
```

Môžeme array kreovať priamo príkazom arange

```
In [8]: ar=arange(2,6,2)

In [9]: ar
Out[9]: array([2, 4])
```

Môžeme kreovať array príkazom empty. Argument funkcie empty je počet prvkov poľa. Ale pozor. Pole je formálne nenaplnené, ale v skutočnosti jeho prvky obsaujú „garbage“ z pamäte počítača, ako to ukazuje príklad

```
In [27]: ar=empty(10)

In [28]: ar
Out[28]:
array([  2.49566589e-316,   2.49566866e-316,   2.49567143e-316,
         2.49567419e-316,   2.49567696e-316,   2.49567973e-316,
         2.49568249e-316,   2.49568526e-316,   2.49568803e-316,
         2.49569079e-316])
```

Po vytvorení poľa príkazom empty musím pole zmysluplne naplniť napríklad cyklom

```
In [43]: ar = empty(5)

In [44]: for i in range(5):
    ....:     ar[i]= i+2
    ....:

In [45]: ar
Out[45]: array([ 2.,  3.,  4.,  5.,  6.])
```

Poznamenajme, že príkaz plot() pracuje rovnako dobre s parametrami typu array ako s parametrami typu list. (V skutočnosti ak sa mu zadá parameter typu list, on si ho najprv typecastuje do typu array a potom s ním ďalej pracuje.

10.3. Import modulov a symbolov

V našom škôlkarskom programovaní sme sa nemuseli starať o import potrebných knižníc, lebo všetko potrebné za nás urobil parameter `--pylab` s ktorým sme vyvolávali príkaz ipython. Keď však hľadáte nejaké programové inšpirácie na webe vidíte, že ako prvé príkazy sú uvedené príkazy typu import, s ktorými sme sa nestretli. Ide o explicitný import knižníc (modulov) resp. symbolov z modulov. Explicitný import musíme zadať ak chceme spúšťať skripty v neinteraktívnom režime.

Najjednoduchší ekvivalent ako začať skript, keď som zvyknutý spúšťať ipython s option --pylab je

skript pylabtest.py

```
from pylab import *
from numpy import *
x = arange(0, 10, 0.2)
y = sin(x)
plot(x, y)
show()
```

spúšťa sa ako

`python pylabtest.py`

Nemusím importovať všetky symboly, alternatívne použitie zvyčajne uvádzané v tutoriáloch je

```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(0, 10, 0.2)
y = np.sin(x)
plt.plot(x, y)
plt.show()
```

11. Python memo program

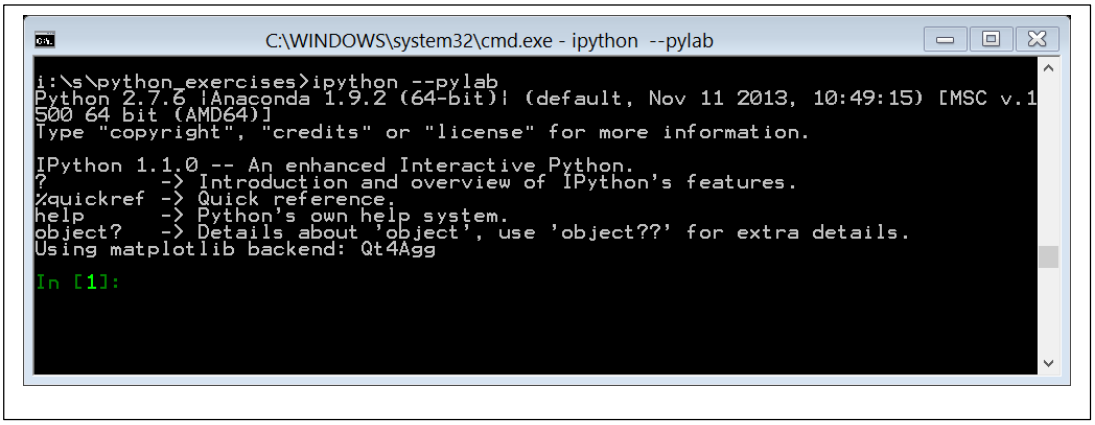
Programujem vo viacerých jazykoch, a potom si nikdy nepamätám, syntax napríklad cyklu pre daný kokrétny jazyk. Preto mám na disku niekoľko typických programov vo viacerých jazykoch, ktoré síce nerobia nič užitočné ale sú zbierkou príkladov jazykových konštrukcií. Vyrobil som súbor PythonMemo.py. Kto si ho prečíta, a tuší, čo to je „programovať“ sa nemusí učiť Python a aj tak odpozoruje ako môže niečo jednoduché v Pythone naprogramovať. Tu je

```
from pylab import *
#from numpy import *
a=1.2
b=a+2.3
print(b)
################  list examples ####################
lst=[1.2,2.3,4.] #example of list
print(lst)
#################  for examples ####################
for x in lst:    #for statement
        print(x+2.3)
lst1=["fero","jano"]   #list of strings
for s in lst1:
        print(s+" neznamy")
##############  range examples  ########################
lst2=range(2,6,1)     # from, up_to-but-less-then, step (arguments must be integers)
print(lst2)
for i in range(0,10,2):
        print(2*i)
##############  arrays examples  ################################
arr1=arange(2.1,6.1,1.9)   #array, arguments may be floats
print(arr1)
arr2=arange(0,10,0.1)
arr3=empty(10)
for i in range(0,10,1):
        arr3[i]=log(0.1*i)
print(arr3)
for i in range(0,10,1):
        if i<5:
                arr3[i]=i
        else:
                arr3[i]=2*i
print(arr3)
####################  plot example  ################
yarr2=sin(arr2)    #apply function to a list
plot(arr2,yarr2)
y2=2.*yarr2;
plot(arr2,y2)
show()   #blocking execution until plot window closed
##################  user function example  #################
def is_inside(x,y):
        # returns true if the point x,y is inside a unit circle, false if not
        if (x*x+y*y)<1:
                return (True)
        else:
                return (False)
################# random number example  ##############
# Monte Carlo calculation of circle area
area=0.
for trial in range(0,10000,1):
        x=-1.+2.*np.random.random()
        y=-1.+2.*np.random.random()
        if is_inside(x,y):
                area=area+1
print(4.*area/10000.)  #should be pi
```

12. Čo ďalej

Skončili sme náš škôlkarsky úvod do Pythonu. Treba povedať, že Python je dnes veľmi obľúbený a poskytuje veľa funkčnosti, takže schopní programátori v ňom vytvárajú aj veľmi zložité programové systémy.

Kto chce v Pythone pokročilo programovať, musí sa podrobnejšie zoznámiť s jazykom i s knižnicami. Tu uvádzam niekoľko liniek, aby sa prípadný záujemca mohol naštartovať. Poznamanávam, že samotný ipython poskytuje dosť bohaté možnosti nápovedy a istú úroveň API dokumentácie. Spôsob prístupu je naznačený prvými riadkami zobrazenými po vyvolaní programu ipython.



Vhodným menej náročným pokračovaním tohto škôlkarského kurzu je napríklad

http://learnpythonthehardway.org/book/

Python tutorial je tu: https://docs.python.org/2/tutorial/index.html

Referencia na wikibooks: http://en.wikibooks.org/wiki/Python_Programming

Caltech Python short course: http://www.wag.caltech.edu/home/rpm/python_course/

Python Reference: https://docs.python.org/2/reference/index.html#reference-index

Python standard Library: https://docs.python.org/2/library/index.html#library-index

Dokumentácia k Scientific Pythonu (Numpy, SciPy, Matplotlib, Ipython): http://scipy.org/docs.html

Matplotlib FAQs: http://matplotlib.org/faq/index.html