Vývoj geografických aplikácií v GIS

Hana Stanková

Lekcia 2

Základy jazyka Python

História jazyka Python

- zakladateľ Guido van Rossum (Holandsko)
- hlavnému autorovi dala komunita titul Benevolent Dictator for Life (BDFL)
- jazyk je pomenovaný podľa slávneho seriálu Monty Python’s Flying Circus

História jazyka Python

- implementácia začala v decembri 1989
- Python 2.0 vyšiel 16. októbra 2000
- Python 3.0 (nazývaný tiež Python 3000 alebo py3k) vyšiel 3. decembra 2008 a je spätne nekompatibilný
- hlavné funkcie boli implementované v spätne kompatibilných verziách 2.6 a 2.7
- najnovšie stabilné verzie: 3.5.0 (13.september 2015) a 2.7.10 (23.máj 2015)

Charakteristika jazyka Python

- objektovo-orientované programovanie
- štruktúrované programovanie
- filozofia jazyka je zhrnutá v dokumente PEP 20 (The Zen of Python), ktorý obsahuje perličky ako:
- *krásne je lepšie ako škaredé*
- *explicitné je lepšie ako implicitné*
- *jednoduché je lepšie ako komplexné*
- *komplexné je lepšie ako komplikované*
- *čitateľnosť sa ráta*

Objektovo-orientované programovanie

- v jazyku Python je všetko objekt
- každý objekt má:

1. unikátne ID alebo miesto v pamäti počítača
2. množinu vlastností, ktorá ho opisuje
3. množinu metód alebo vecí, ktoré objekt môže robiť

-. procedurálne jazyky (napr. Fortran) riešia nejakú operáciu ako sadu po sebe nasledujúcich krokov

Objektovo-orientované programovanie

- napr. chceme napísať program, ktorý urobí na raňajky sandwich (chlieb s maslom a džemom)
- procedurálny jazyk:

1. choď do chladničky a vyber maslo a džem
2. choď do poličky a vyber chlieb
3. otvor zásuvku a vyber nôž
4. odkroj dva krajce chleba
5. naber na nôž maslo ..... atď.

Objektovo-orientované programovanie

- objektovo-orientovaný jazyk:

1. mySandwich = Sandwich.make
2. mySandwich.Bread = Wheat
3. mySandwich.Add(Butter)
4. mySandwich.Add(Jam)

- počet krokov programu je zredukovaný
- objekt Sandwich „vie“, ako sa má vybudovať – *zapuzdrenie (encapsulation)*

Objektovo-orientované programovanie

- definujeme vlastnosti sandwicha (napr. typ chleba) a vykonávame na ňom metódy (napr. pridávame maslo a džem)
- aby sme mohli takto jednoducho urobiť sandwich, nejaký programátor pred nami musel definovať, čo je to sandwich a čo môže robiť

Triedy

- **trieda (class) –** definuje, akým spôsobom sa vytvorí objekt, jeho vlastnosti a metódy, akým spôsobom sa vlastnosti nastavujú a používajú a čo metódy robia
- môžeme to nazvať aj návrh na tvorbu objektov (napr. v továrni sa podľa návrhu vyrábajú tisícky rovnakých výrobkov)
- triedy sú v Python-e zoskupené do modulov

Dedičnosť

- **dedičnosť (inheritance)** – triedy sú navrhnuté hierarchicky, takže každá trieda dedí vlastnosti a metódy z triedy o úroveň vyššie (nadtrieda alebo rodičovská trieda), a zároveň každá trieda odovzdáva svoje vlastnosti a metódy triedam o úroveň nižšie (podtriedam alebo triedam potomkov)
- diagram modelu objektov pre skriptovanie v Pythone v ArcGIS 9.3: http://webhelp.esri.com/arcgisdesktop/9.3/pdf/Geoprocessor_93.pdf

Funkcie a moduly

- funkcia – definovaná časť funkcionality, ktorá vykonáva špecifickú úlohu
- vyžaduje argumenty ()
- modul – súbor Python-u, kde funkcia žije
- importuje sa
- napr. math.sqrt(100)
- premenné, funkcie atď. sú v Python-e case sensitive, t.z. musíme rozlišovať medzi malými a veľkými písmenami

Python a ArcGIS

- Python – skriptovací jazyk pre ArcGIS
- voľne šíriteľný, multi-platformový, ľahko sa učí, zavedená komunita
- predstavuje iný spôsob spúšťania nástrojov
- umožňuje vyvíjať, spúšťať a zdieľať geoprocesné postupy
- zvyšuje produktivitu

ArcPy

- site package na skriptovanie v ArcGIS-e
- **site package** (termín Python-u) – knižnica, ktorá pridáva do Python-u dodatočné funkcie
- umožňuje prístup ku geoprocesným nástrojom, ako aj k ďalším funkciám, triedam a modulom na tvorbu jednoduchých aj komplexných postupov (workflows)
- ArcPy je organizovaný do nástrojov, funkcií, tried a modulov (všetky majú vzťah k skriptovaniu v ArcGIS)

ArcPy

- funkcie zlepšujú geoprocesné postupy
- triedy sa dajú použiť na tvorbu komplexných objektov
- moduly poskytujú pridanú funkcionalitu
- ArcPy je postavený na module **arcgiscripting** (pred verziou ArcGIS 10.0)
- ponúka dopĺňanie kódu (code completion)
- v ArcGIS 10.0 je postavený na Python 2.6
- v ArcGIS 10.1 je postavený na Python 2.7

Geoprocesné nástroje

- funkcie prístupné cez arcpy, teda pristupuje sa k nim ako k ostatným funkcám Python-u
- je však rozdiel medzi nástrojmi a nenástrojovými (nontool) funkciami :
- sú odlišne dokumentované (nástroje majú *Tool Reference Page* v ArcGIS Desktop Help systéme, funkcie sú dokumentované v ArcPy dokumentácii)
- nástroje vracajú objekt *Result*, funkcie nie

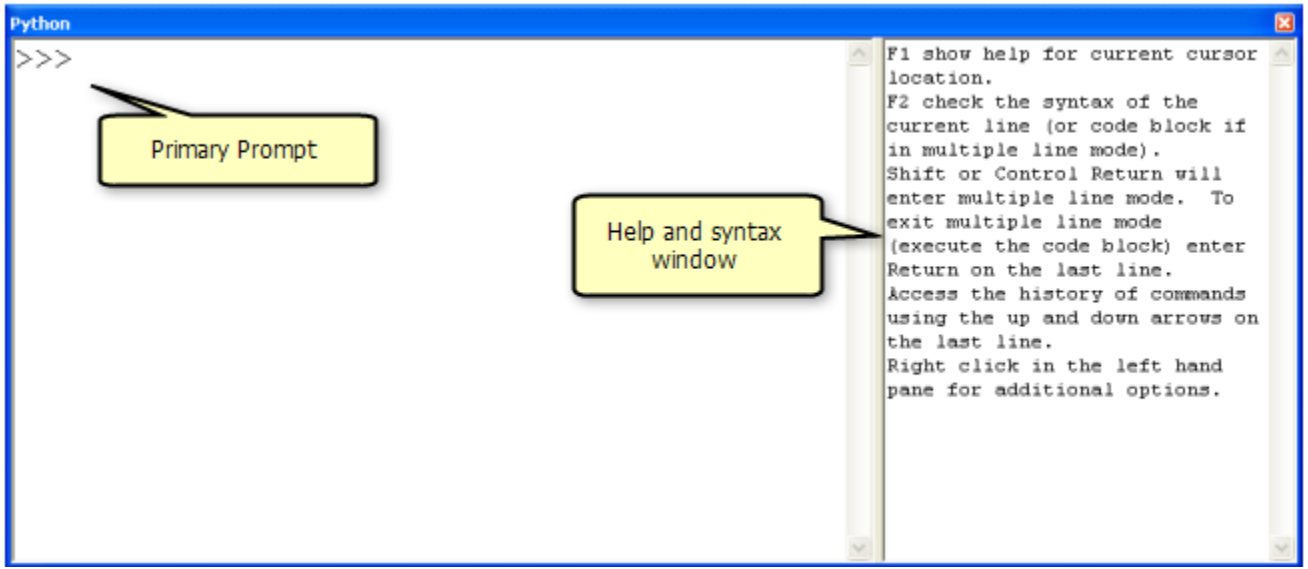
Geoprocesné nástroje

- nástroje na rozdiel od funkcií produkujú *messages* (správy), ku ktorým sa pristupuje pomocou funkcií ako GetMessages()
- nástroje sú licencované úrovňou produktu (ArcGIS Desktop Basic, Standard, Advanced) a extenziami (ArcGIS Network Analyst, Spatial Analyst atď.), kým funkcie nie sú licencované, sú nainštalované spolu s ArcPy

Python window

- novinka v ArcGIS 10 (nahradilo dovtedajšie Command Line window)
- dá sa z neho spúšťať akýkoľvek Python kód, nielen geoprocesné nástroje
- okno sa dá spustiť z ktorejkoľvek Desktop aplikácie cez ikonku 

Python window



Python window

- viacriadkové príkazy – na nový riadok sa dostanem CTRL + ENTER alebo SHIFT + ENTER
- spustenie viacriadkového príkazu – 2x ENTER
- uloženie skriptu – kontextové menu – Save As...

Cesty k súborom

- programovacie jazyky ako Python používajú spätné lomítko (****) ako *escape* znak (napr. **\n** je posun na nový riadok, atď.), t.j. znak, ktorý vyvoláva alternatívnu interpretáciu nasledujúcich znakov v reťazci
- preto sa cesty k súborom zapisujú pomocou obyčajného lomítka (**/**) namiesto spätného
- ďalší spôsob je použiť dve spätné lomítka (****) alebo písmeno **r** pred textovým reťazcom

Komentáre

- na komentovanie slúži znak **#**
- ak je kód na viac než jeden riadok, používa sa znak **** na prechod do ďalšieho riadku (nie je nutné, ak sme uprostred zátvoriek)

```
>>> # Som komentár.
>>> Som komentár.
  File "<input>", line 1
    Som komentár.
             ^
SyntaxError: invalid syntax
>>>
```

Premenné

- **premenné (variables)** – mená, ktoré obsahujú hodnoty
- slúžia ako náhrada za riadkové hodnoty
- hodnoty sa priraďujú pomocou =
- premenné nie je potrebné deklarovať

```
>>> path = 'C:/Data/cesty.shp'
>>> distance = 30
>>> print (path, distance)
('C:/Data/cesty.shp', 30)
>>>
```

Premenné

- sú *case-sensitive*
- nesmú obsahovať medzery
- nesmú sa začínať číslom
- mali by sa začínať malým písmenom, dvojslovné názvy by mali pokračovať prvým veľkým písmenom (myVariable)
- nesmú to byť vyhradené slová jazyka Python ako napr. import, print atď.

Zoznamy

- **zoznam (list)** – zoznam hodnôt (položiek) oddelených čiarkou, v hranatých zátvorkách **[ ]**
- zoznamy majú metódy:

  append(x) – vloží položku **x** na koniec zoznamu
  insert(i,x) – vloží položku **x** na určité miesto v zozname (**i** je poradové číslo položky, pred ktorú sa vloží nová, t.j. ak **i = 0**, vloží sa na začiatok zoznamu)
  remove(x) – odstráni položku **x** zo zoznamu (prvú, ktorá sa vyskytne)

Zoznamy

- zoznamy majú metódy:
  index(x) – vráti poradové číslo položky **x** v zozname (prvej, ktorá sa vyskytne)
  count(x) – vráti početnosť výskytu položky **x** v zozname
  reverse() – zmení poradie položiek na opačné
  sort() – usporiada položky v zozname
- dĺžka zoznamu – funkcia **len()** vracia počet položiek
- do zoznamov môžu byť vnorené ďalšie zoznamy (viacúrovňové zoznamy)

Zoznamy

```
>>> zoznam = [0,1,2,3]
>>> zoznam.append(4)
>>> print zoznam
[0, 1, 2, 3, 4]
>>> zoznam.insert(0,-1)
>>> print zoznam
[-1, 0, 1, 2, 3, 4]
>>> zoznam.index(4)
5
>>> zoznam.remove(1)
>>> print zoznam
[-1, 0, 2, 3, 4]
>>> zoznam.reverse()
>>> print zoznam
[4, 3, 2, 0, -1]
>>> zoznam.sort()
>>> print zoznam
[-1, 0, 2, 3, 4]
>>> zoznam.count(0)
1
>>> len(zoznam)
5
>>>
```

Zoznamy

- ak meníme hodnoty v zozname, je dobre si urobiť predtým kópiu zoznamu
- pohodlné je použiť výrez zo zoznamu (*list slice*) pomocou **[:]**
- pri zápise v tomto tvare sa použijú defaultné hodnoty **[začiatok : koniec]** zoznamu, resp. **[0 : n]**, takže sa uloží celý zoznam

Zoznamy

- funkcie **arcpy.List**:
  ListDatasets
  ListFeatureClasses
  ListFields
  ListFiles
  ListRasters
  ListTables
  ... atď.

Testovacie podmienky

- príkazy **if**, **else**
- na konci každej podmienky musí byť dvojbodka **:**
- **odsadenie (indentation)** určuje, čo sa vykoná
- v jazyku Python je povinné!
- testovanie rovnosti **==**, operátory **<**, **>**, **!=** ,...
- príkaz **elif** (skratka z **else if**) – môže ich byť 0 a viac, slúžia ako náhrada za výrazy *switch* alebo *case* používané v iných jazykoch

Testovacie podmienky

```
>>> var = "a"
>>> if var == "a":
...     print "Premenná je a."
... else:
...     print "Premenná je nie je a."
...
...
Premenná je a.
>>> x = 4
>>> if x < 0:
...     print "Záporné"
... elif x == 0:
...     print "Nula"
... elif x == 1:
...     print "Jeden"
... else:
...     print "Viac"
...
...
Viac
>>>
```

Cykly

- techniky pre opakovanie – iterácie a cykly
- *while* cykly, počítané cykly, cykly zoznamov
- na konci každej podmienky musí byť dvojbodka **:**
- odsadenie určuje, čo sa vykoná
- cykly môžu byť aj vnorené (cyklus vnútri iného cyklu)

Cyklus while

- vykonáva sa dovtedy, kým nie je splnená zadaná podmienka

```
>>> x = 1
>>> while x < 5:
...     print x
...     x = x+1
...
...
1
2
3
4
>>>
```

Cyklus for

- trochu odlišný od podobných príkazov v jazykoch ako Pascal (aritmeticky prechádza číslami) alebo C (umožňuje zadať iteračný krok aj podmienku na zastavenie)
- príkaz **for** v Python-e prechádza po položkách akejkoľvek sekvencie (zoznamu alebo reťazca) v poradí, v akom sa objavujú v sekvencii

Cyklus for

```
>>> for cislo in range(1,5):
...     print cislo
...
...
1
2
3
4
>>> # Meranie retazcov
>>> slova = ['cat','window','defenestrate']
>>> for slovo in slova:
...     print slovo, len(slovo)
...
...
cat 3
window 6
defenestrate 12
>>>
```

Funkcia range

- zabudovaná funkcia, ktorá generuje aritmetické rady
- hodí sa, ak potrebujeme iteratívne prechádzať sekvenciou čísel
- syntax: **range([start,] stop[, step])**
- ak nedefinujeme začiatočné číslo, default je 0
- funkcia **range(i,j)** vráti **[i,i+1,i+2,....,j-1]**
- prírastok (step) môže byť aj negatívny

Funkcia range

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(5,10)
[5, 6, 7, 8, 9]
>>> range(1,12,3)
[1, 4, 7, 10]
>>> range(-10,-120,-40)
[-10, -50, -90]
>>>
```

List comprehensions

- skrátený spôsob tvorby zoznamov
- bežný spôsob – vytváranie zoznamov pomocou cyklov a podmienok
- **list comprehensions** - pozostávajú z **[ ]** a kľúčového slova **for**, za ktorým môžu byť ďalšie **for** a **if** pravidlá
- môže obsahovať aj matematické funkcie a vnorené funkcie
- pri vnorených zoznamoch n-tice hodnôt musia byť v **( )**

List comprehensions

```
>>> stvorce = []
>>> for x in range(5):
...     stvorce.append(x**2)
...
...
>>> stvorce
[0, 1, 4, 9, 16]
>>> stvorce = []
>>> stvorce = [x**2 for x in range(5)]
>>> stvorce
[0, 1, 4, 9, 16]
>>> zoznam = [-4,-2,0,2,4]
>>> [x for x in zoznam if x>0]
[2, 4]
>>> ovocie = ['hruska ',' jablcko','borievka ']
>>> [plod.strip() for plod in ovocie]
['hruska', 'jablcko', 'borievka']
>>> [(x,x**2) for x in range(4)]
[(0, 0), (1, 1), (2, 4), (3, 9)]
>>> [x,x**2 for x in range(4)]
  File "<input>", line 1
    [x,x**2 for x in range(4)]
           ^
SyntaxError: invalid syntax
>>>
```