Programski jezik Python

Python in Unix - Zgradba programa - Imena in objekti - Števila - Znakovni nizi - Seznami - Slovarji - Logični izrazi in krmilni stavki - Standardni izhod in vhod - Znakovne datoteke - Funkcije in procedure - Podatkovni in funkcijski objekti - Program v več modulih - Sistemske funkcije - Standardni moduli - Modul za polja - Ukaznovrstični parametri - Robustnost izvajanja - Spremembe verzij

Python in Unix

Python je programski jezik in njegov interpreter, to je računalniški program, ki ta jezik razume in ga izvaja.

Python zaženemo iz Unixove ukazne vrstice z ukazom python. Javi se s pozornikom >>. Odtipkavamo mu ukaze, ki jih sproti izvršuje. Končamo s tipko Ctrl+D (hkratni pritisk tipke Ctrl in tipke D).

Namesto sprotnega odtipkavanja ukazov lahko te vnaprej, s primernim urejevalnikom besedila, zapišemo v datoteko, na primer test.py. Iz ukazne vrstice jo potem izvršimo z ukazom python test.py. Lahko pa dodatno v prvo vrstico datoteke zapišemo #!/usr/bin/python in jo nato naredimo izvršno, na primer z vrstičnim ukazom chmode 755. Potem jo poženemo iz ukazne vrstice kar s tipkanjem njenega imena. Ime je sedaj lahko tudi brez končnice .py. Pod drugimi operacijskimi sistemi, ki ne prepoznavajo prve vrstice #!, je končnica obvezna za zagon.

Pri zagonu pogleda Python v tri okoljske spremenljivke. PATH kaže na imenik, kjer je interpreter python, PYTHONPATH pa na imenike, kjer so moduli (obstoječe ukazne datoteke), ki jih lahko program kliče. Spremenljivka PYTHONSTARTUP vsebuje polno ime datoteke, ki se izvede, če Python zaženemo interaktivno.

Zgradba programa

Program je sestavljen iz zaporedja ukazov. Navadno pišemo en ukaz v eno vrstico. Ukaz se mora praviloma začeti v prvi koloni. Izjeme so opisane v nadaljevanju. Konec vrstice kaže, da je ukaza konec.

V eno vrstico lahko pišemo več ukazov, če jih ločimo s podpičjem ;.

Dolg ukaz lahko pišemo v več zaporednih vrstic, če na koncu vsake vrstice pišemo poševnico \.

Nekateri ukazi so sestavljeni iz glave in telesa. Glava je ukaz, ki se konča z dvopičjem :. Telo je zaporedje ukazov, ki so vsi zamaknjeni v desno za isto število presledkov, najbolje za štiri. Sestavljeni ukazi lahko vsebujejo druge sestavljene ukaze. Ti so zamaknjeni še globlje.

Vse, kar v vrstici sledi znaku #, Python ignorira. To je komentar za vzdrževalce programa.

Imena in objekti

Osnovni ukaz je prireditveni stavek. Zgled je naslednji:

```
x = 1
```

Python nekje v pomnilnik zapiše celo število 1. To je celoštevilčni objekt. V svojo tabelo imen pa zapiše ime x in ga poveže z objektom. Tehnično je x kazalec na 1:

```
x ---> 1
```

Ime x si lahko nazorno predstavljamo kot listič z napisom x, ki je privezan na objekt 1.

Listič, ki je privezan na nek objekt, lahko od njega odvežemo in ga privežemo na drug objekt:

```
x = 1          x --->  1

x = 3          x -\    1
                  \-> 3

y = x          x --->  3
               y ------^
```

En in isti objekt ima lahko več imen, ki so nanj privezana. Eno in isto ime lahko prevežemo na različne objekte.

Kadarkoli imenu priredimo objekt, Python pogleda, ali ime že obstaja v tabeli imen. Če ne, ga ustvari. Če da, ga uporabi.

Če ime povežemo na drugo ime, ki ni privezano nikamor, se Python upre.

Python sproti briše objekte, na katere ne kaže nobeno ime več.

Imena so lahko sestavljena iz črk, številk in podčrtaja. Ne smejo se začeti s številko. Male in velike črke se razlikujejo.

Nekatera imena niso dovoljena, to je tista, ki jih Python razume kot ukaze: if, while, for in podobno, kot bomo še videli.

Števila

Osnovni objekti za računanje so števila: cela, realna in kompleksna. Ustvarimo jih kot x = število, pri čemer je število, na primer:

```
12, -100            Cela števila
3.14, 1.0e-3        Realna števila
3j, 1 + 2.5j        Kompleksna števila
```

Iz števil (imen in/ali objektov) sestavljamo aritmetične izraze:

```
x + y               Vsota
x - y               Razlika
x * y               Produkt
x / y               Kvocient
x % y               Modul
x ** y              Potenca
```

in na njihove vrednosti vežemo imena, na primer:

```
z = x + y
```

Python izračuna vrednost izraza x + y, ustvari ustrezen številski objekt in nanj priveže ime z. Seveda lahko zapišemo tudi

```
x = x + y
```

Python spet izračuna vrednost izraza x + y, ustvari ustrezen objekt in nanj priveže ime x. Ime x je sedaj privezano na drug objekt kot pred tem ukazom.

V izrazu z mešanimi tipi se vsi operandi pretvorijo v najbolj "kompleksnega" izmed njih in tak je potem tudi rezultat.

Vrstni red operatorjev določajo standardna precedenčna pravila: najprej potence, potem množenje in deljenje in na koncu seštevanje in odštevanje. Eksplicitno je mogoče upravljati z oklepaji (in zaklepaji).

Znakovni nizi

Poleg treh tipov števil so objekti tudi znakovni nizi (character strings). Takšen niz ustvarimo kot x = niz, pri čemer je niz, na primer:

```
""                  Prazen niz
"abc"               Enovrstični niz
"""                 Večvrstični niz
abc                 gre lahko preko več vrstic
def                 in znaki LineFeed niso del niza
"""
```

Poleg navadnih tiskarskih znakov (številk, črk in simbolov razen poševnice \) lahko v niz zapišemo tudi preostale posebne znake in sicer s pomočjo poševnice \ takole:

```
\\                  \
\"                  "
\n                  NewLine
\r                  CarriageReturn
\t                  Tab
\f                  FormFeed
\xHH                Heksadecimalni HH
```

Posamezni znaki so v nizu x indeksirani začenši z 0. Referenciramo jih takole:

```
x[1]                Znak 1
x[1:3]              Znaki od 1 do 3-1
x[1:]               Znaki od 1 do konca
x[:3]               Znaki od 0 do 3-1
x[:]                Vsi znaki
```

Referencirane znake lahko iz niza kopiramo, ne moremo pa jih v nizu spreminjati:

```
y = x[1]            Dovoljeno
x[1] = "a"          Prepovedano
```

Znakov tudi ni mogoče nizu pripenjati, jih vanj vrivati ali iz njega črtati. Rečemo, da so znakovni nizi, prav kakor števila, _nespremenljivi_ objekti.

Z nizi tudi "računamo":

```
x + y               Spojitev (konkatenacija):
                    iz "abc" + "d" nastane "abcd"
x * 3               Razmnožitev:
                    iz "abc"*2 nastane "abcabc"
```

Seznami

Seznam (list) je oštevilčeno (začenši z 0) zaporedje števil, znakovnih nizov in drugih objektov, ki jih bomo še spoznali. V pomnilniku si ga mislimo kot vlak, to je lokomotivo s pripetimi in oštevilčenimi vagoni. Prvi vagon nosi številko 0. Ime seznama je privezano na lokomotivo. Vsak vagon vsebuje kazalec na svoj objekt. Takšen seznam ustvarimo kot x = seznam, pri čemer je seznam, na primer:

```
[]                  Prazen seznam
[10, 20, 30]        Homogen seznam
[10, 3.14, "abc"]   Heterogen seznam
```

Vlaku lahko vagone - elemente - dodajamo in odvzemamo, pa tudi spreminjamo njihovo vsebino. Rečemo, da je seznam spremenljiv objekt:

```
x.append(5)         Dodaj na koncu nov element,
                    ki kaže na število 5
x.insert(1,5)       Vrini nov element 1,
                    ki kaže na število 5
x.remove(1)         Črtaj element 1
x.pop()             Črtaj zadnji element

x[1] = 5            Element 1 seznama l naj
                    kaže na na število 5
```

Sintaksa x.append(5), kot primer, je posebnost, v katero se ne bomo spuščali. Mislimo si, da pomeni append(x), kar je

proceduralni ukaz, kakršne bomo še spoznali.

Kjer je to smiselno, referenciramo intervale elementov v seznamu prav tako kot intervale črk v znakovnem nizu, na primer x[1:3].

Slovarji

Slovar (dictionary) je seznam, katerega elementi so indeksirani z znakovnimi nizi, ključi, ne z zaporednimi številkami. V pomnilniku si ga mislimo kot lokomotivo z vagoni, na katerih so s črkami napisani ključi. Ime slovarja je privezano na lokomotivo. Takšen slovar ustvarimo kot x = slovar, pri čemer je slovar, na primer:

```
{}
{"marylin": 2432, "audrey": 1253}
```

Vlaku lahko vagone - elemente - dodajamo in odvzemamo, pa tudi spreminjamo njihovo vsebino. Pri tem vagone referenciramo preko njihovih ključev. Rečemo, da je slovar spremenljiv objekt:

```
x["marylin"] = 16   Sprememba obstoječega elementa
x["michael"] = 65   Stvaritev novega elementa

del x["audrey"]     Uničenje elementa
del x               Uničenje slovarja

x.clear()           Uniči vse elemente
x.values()          Pokaže vse vrednosti kot seznam
x.keys()            Pokaže vse ključe kot seznam
```

V slovarju je pod ključem lahko spravljen kakršenkoli objekt, celo seznam. In obratno: v seznam so lahko nanizani kakršnikoli objekti, celo slovarji.

Logični izrazi in krmilni stavki

Zaporedno izvrševanje programa spreminjamo s krmilnimi stavki, ki vsebujejo logične izraze. To so izrazi, katerih vrednosti so True ali False.

Objekta True in False vračata vrednosti True in False.

Številske primerjave vračajo vrednosti True in False:

```
x == y
x != y
x > y
x >= y
x < y
x <= y
```

Bolj zapletene logične izraze sestavljamo z operatorji not, and in or, na primer x > 5 and x < 10. Logične izraze bomo poimenovali kar pogoji.

Razvejitev:

```
if pogoj1:
    ukazi
elif pogoj2:
    ukazi
else:
    ukazi
```

Ukaz lahko vsebuje nič, eno ali več glav elif, ter nič ali eno glavo else. Če je v kakem bloku le en sam ukaz, ga lahko pišemo v isto vrstico kot njegovo glavo. Če sintaksa zahteva ukaz, pa ni potrebno ničesar narediti, zapišemo ukaz pass.

Pogojna iteracija:

```
while pogoj:
    ukazi
```

V telo lahko vtaknemo še dva posebna ukaza: continue in break. Oba povzročita skok ven iz zanke: prvi vrne izvajanje na glavo, drugi pa na prvi ukaz za telesom.

Iteracija:

```
for x in sekvenca:
    ukazi
```

Pri tem je sekvenca lahko znakovni niz, seznam ali posebna funkcija xrange(n), ki vrača zaporedje števil 0..n-1. Funkcijo lahko kličemo tudi z dvama argumentoma (n,m); potem vrne zaporedje n..m-1. V telo lahko tudi zapišemo ukaza continue in break.

Standardni izhod in vhod

Tekstovno reprezentacijo poljubnega objekta zapišemo na standardni izhod (zaslon):

```
print x, y
```

Objekti se zapisujejo po vrsti in so med seboj ločeni s presledkom. Na koncu se zapiše znak LineFeed. Če tega nočemo, za zadnjim znakom napišemo vejico ,.

Izpis lahko formatiramo:

```
print "X = %s Y = %6d Z = %6.3f" % (x, y, z)
```

```
s       Znakovni niz
d       Celo število (na 6 mest)
f       Realno število v fiksni notaciji (na 3 dec.)
e       Realno število v eksponentni notaciji
```

Standardni izhod lahko preusmerimo v datoteko, in sicer z znakom > v ukazni vrstici.

S standardnega vhoda (tipkovnice) čitamo tekstovno reprezentacijo poljubnega objekta, recimo realnega števila, kot

```
x = raw_input("Vpiši:")
```

Po potrebi ga nato pretvorimo v ustrezno število:

```
y = int(x)
y = float(x)
y = complex(x)
```

Znakovne datoteke

Znakovna datoteka je zaporedje znakov (oktetov), ki se konča z znakom EndOfFile. Vmes so lahko posejani znaki LineFeed. Obstoječo zunanjo datoteko, na primer podatki.txt, odpremo za čitanje:

```
f = open("podatki.txt","r")
```

Podobno odpremo izhodno datoteko za pisanje od začetka z "w" in za dodajanje na koncu z "a".

Čitaj naslednjih n znakov v niz s:

```
s = f.read(n)
```

Čitaj vse znake (celotno datoteko) v niz s:

```
s = f.read()
```

Čitaj naslednjo vrstico (to je vse znake vključno do znaka LineFeed v niz s; če je prazna, vrni prazni niz:

```
s = f.readline()
```

Čitaj vse vrstice v seznam nizov (list of strings) l. Vsaka vrstica gre v en element seznama:

```
l = f.readlines()
```

Zapiši niz s:

```
f.write(s)
```

Zapiši vse vrstice iz seznama vrstic l:

```
f.writelines(l)
```

Zapri datoteko

```
f.close()
```

Funkcije in procedure

Funkcija je izoliran del programa, ki ima svoje lokalne objekte in svoja lokalna imena. Prototip je naslednji:

```
def f(x):
    r = x**2
    return r

a = 3
u = f(a)
```

Imena f, a in u so zapisana v zunanji tabeli imen, imeni x in r pa v lokalni tabeli. Zunanja imena v funkciji niso poznana. Lokalna imena v okolici niso poznana.

Ko pokličemo funkcijo, se x poveže tja, kamor kaže a. Ko funkcija konča z delom, pa se u poveže tja, kamor kaže r.

Funkcija ima lahko več argumentov poljubnega tipa. Praviloma ima en sam izhod poljubnega tipa. Če je vhodov in izhodov več, pišemo takole:

```
def f(x,y):
    r =
    t =
    return (r,t)

a =
b =
(u,v) = f(a,b)
```

Funkcija je povezana z okolico samo preko imen x in r. Kaj se zgodi, če v funkciji povežemo x kam drugam, recimo x = 1? Listek x sedaj kaže na objekt 1 znotraj funkcije, zunanji objekt, kamor je bil x prej vezan, pa ostane nespremenjen.

Povedano velja splošno, torej tudi za x, ki kaže na začetek zunanjega seznama. Če povežemo x = 1, ostane zunanji seznam nespremenjen. Če pa povežemo x[i] = 1, ne umikamo kazalca x z lokomotive, ampak preusmerimo kazalec iz i-tega vagona. I-ta komponenta zunanjega seznama se s tem spremeni!

Pythonova funkcija torej lahko spreminja okolico preko argumenta, vendar le v primeru, ko ta kaže na seznam. Funkcijo, ki vpliva na okolico preko argumentov in ne preko izhodne vrednsoti, poimenujemo procedura. Prototip je:

```
def f(x,n):
    x[n] = 10
    return

n = 1
a = [1,2,3]
f(a,n)            # a je sedaj [1,10,3]
```

Kadar želimo s funkcijo spreminjati zelo dolge sezname, je bolje, da jih spreminjamo preko argumenta kot preko izhoda. V slednjem primeru mora namreč funkcija ob vsakem klicu zgraditi nov lokalni seznam, kar pomeni več prostora in počasnejše delovnaje.

Podatkovni in funkcijski objekti

Iz osnovnih podatkovnih objektov - znakovnih nizov, števil, seznamov in slovarjev - lahko zgradimo nov podatkovni objekt.

Sestavljeni objekt, record, brez notranje strukture definiramo kot

```
class record(object):
    pass
```

Njegovo inkarnacijo, na primer oseba, ustvarimo takole:

```
oseba = record()
```

Notranjo podatkovno strukturo objekta zgradimo po temle vzorcu:

```
oseba.ime = "Marilyn"
oseba.starost = 22
oseba.mere = [90, 60, 90]
```

Posamične komponente uporabljamo prav tako, kot če bi bile samostojne.

Poleg podatkovnih komponent lahko sestavljeni objekt vsebuje tudi funkcije. Te so pisane tako, da spreminjajo le podatke znotraj objekta. S takimi objekti se ne bomo ukvarjali.

Program v več modulih

Tekstovno datoteko, v kateri je zapisan program, imenujemo modul. Ime modula je ime datoteke brez končnice.

Velike programe zapišemo v več datotek. V "glavno" datoteko zapišemo glavni program, v "pomožne" datoteke pa deklaracije podatkov in definicije funkcij.

V glavnem modulu nato včitamo potrebni modul, na primer test.py, in ga po potrebi še preimenujemo v okrajšavo, na primer t:

```
import test
import test as t
```

Ime x (podatkovne strukture ali funkcije) iz včitanega modula

uporabljamo takole:

```
test.x
t.x
```

Alternativno lahko iz modula uvažamo tudi z ukazoma

```
from test import x
from test import *
```

Razlika je v tem, da so sedaj uvožena imena dostopna kar kot x. Pri tem se lahko zgodi, da so v različnih modulih definirana enaka imena. Tedaj velja zadnja uvožena definicija, predhodne pa so izgubljene. Zato je uvoz z * nevaren.

Sistemske funkcije

Sistemske funkcije so na voljo zmeraj, ne da bi bilo potrebno prej uvoziti kak modul.

```
help()           Izpiši pomoč

dir()            Vrne seznam lokalnih imen
dir(modul)       Vrne imena v importiranem modulu

str(objekt)      Vzame poljubni objekt, vrne kot niz

int(niz)         Vzame niz, vrne kot integer
float(niz)       Vzame niz, vrne kot float
complex(niz)     Vzame niz, vrna kot complex

list(niz)        Vzame niz, vrne kot seznam

abs(število)     Absolutna vrednost števila
round(real,n)    Zaokroži realno število
                 na n decimalnih mest

ord(znak)        Vzame znak, vrne ASCII število 0..255
chr(število)     Vzame ASCII število 0..255, vrne kot znak

len(sekvenca)    Dolžina sekvence (niza ali seznama)

map(f,sekvenca)  Uporabi funkcijo f na sekvenci. Vsak
                 element sekvence obdela s to funkcijo in
                 vrne sekvenco obdelav.
```

```
        s.split()        Vrne seznam besed iz s
        l.splitlines()   Vrne seznam vrstic iz l
```

Standardni moduli

Modul sys:

```
        argv             Seznam ukaznovrstičnih besed
        exit(n)          Izhod iz programa s kodo n
```

Modul math:

```
        pi               3.14...
        e                2.72...
        ceil(x)          Prvo absolutno višje celo število
        floor(x)         Prvo absolutno nižje celo število
        pow(x,y)         Potenca x^y
        sqrt(x)          Kvadratni koren
        exp(x)           Eksponentna funkcija e^x
        log(x)           Naravni logaritem
        log10(x)         Desetiški logaritem
        sin(x), asin(x)  Sinus in arkus sinus
        cos(x), acos(x)  Kosinus in arkus kosinus
        tan(x), atan(x)  Tangens in arkus tangens
        atan2(x,y)       Arkus tangens koordinate
        radians(deg)     Deg v rad
        degrees(rad)     Rad v deg
```

Modul random:

```
        seed()           Zasej seme iz sistemske ure
        random()         Slučajno število 0..1
        normalvariate(mu,sigma)  Normalna porazdelitev
```

Modul za polja

Osnovni Python ne pozna podatkovnega tipa array - eno ali večdimenzionalno polje števil. Obravnamo ga sicer lahko kot poseben primer seznama z istovrstnimi elementi, vendar je to pri velikih poljih računsko (prostorsko in hitrostno) zelo neugodno.

Tej pomanjkljivosti odpomore modul numpy, ki definira in obdeluje array (in še marsikaj drugega zraven).

Polje se od seznama razlikuje takole. Število elementov v polju je stalno. Ne moremo jih dodajati ali odvzemati. Vsi elementi so istega tipa in so števila. Vsebino elementov lahko spreminjamo.

Polje kreiramo takole:

```
v = array([2, 5, 8])        Naredi vektor [2 5 8]
    zeros(3)                s tremi ničlami [0 0 0]
    ones (3)                s tremi enicami [1 1 1]
    arange(3)               s tremi komponentami [0 1 2]
    empty(3)                s tremi neznanimi komponentami

m = array([[10,20],[30,40]])Naredi matriko z elementi
    zeros((3,3))            3 x 3, ničle
    ones ((3,3))            3 x 3, enice
    identity(3)             3 x 3, enotna matrika
```

Funkcije array... lahko sprejmejo še dodatni argument, ki pove tip elementov: int, float, complex.

Posamične komponente polja referenciramo tako kot komponente seznama: v[1], m[1,2].

Z vektorji v in matrikami m računamo takole:

```
c * v                       Množenje s skalarjem
v1 + v2                     Seštevanje po komponentah
v1 * v2                     Množenje po komponentah
dot(v1,v2)                  Skalarni produkt
cross(v1,v2)                Vektorski produkt

c * m                       Množenje s skalarjem
m1 + m2                     Seštevanje
dot(m,v)                    Množenje z desne
dot(v,m)                    Množenje z leve

diagonal(m)                 Diagonala
trace(m)                    Sled
transpose(m)                Transponiranje

sort(v)                     Sortiraj po velikosti
min(v)                      Min element
max(v)                      Max element
argmin(v)                   Indeks min elementa
argmax(v)                   Indeks max elementa
size(v|m)                   Koliko elementov
```

```
shape(v|m)                  Koliko vrstic in kolon

absolute(v|m)               Absolutne vrednosti
negative(v|m)               Negativne vrednosti

sum(v|m)                    Vsota elementov
average(v|m)                Povprečje elementov

mean(v)                     Povprečje
std(v)                      Disperzija
cov(u,v)                    Kovarianca
```

Posebej priročna je uporaba funkcij na vsaki komponenti posebej, na primer:

```
exp(v|m), log(v|m), sin(v|m) ...
```

Pri tem so funkcije exp, log, sin itd. tiste iz modula numpy in ne one iz modula math, čeravno imajo enaka imena.

Podobno lahko uporabimo kakšno lastno funkcijo f:

```
array(map(f,v|m),float)
```

Vektor čitamo iz tekstovne datoteke (kjer je vsaka komponenta zapisana v novi vrstici) podatki.txt takole

```
v = loadtext("podatki.txt",float)
```

Matriko (ki je zapisana z vsako vrstico v novi vrstici) čitamo enako.

Ukaznovrstični parametri

V programu je na voljo ime, s katerim smo ga pognali v ukazni vrstici, ter morebitni seznam argumentov (besed, ločenih s presledki):

```
sys.argv         Seznam ukaznovrstičnih besed
```

Element sys.argv[0] vsebuje ime programa, preostali pa lepo po vrsti njegove argumente. Tipično je eden izmed argumentov ime datoteke, ki naj jo program čita ali vanjo piše.

Robustnost izvajanja

Pri izvajanju ukazov se lahko pojavijo napake, na primer odprtje datoteke, ki je ni. Takih napak je več vrst. Kadar se kaka napaka pojavi, Python postavi njej ustrezajočo spremenljivko (s predefiniranim imenom) na vrednost True. Taki spremenljivki rečemo past. Vsaka vrsta napak ima svojo past. S tem je omogočeno, da v programu nanjo reagiramo. Postopek je takle:

```
try:
    ukaz
except past:
    ukazi
```

Če se je med izvajanjem ukaza past postavila na True, se izvedejo ukazi, sicer pa ne. Konkretne napake in pasti so naslednje.

Napaka:	Past:
Uvoz modula	
import	ImportError
Odprtje datoteke	
open()	IOError
Pretvorba niza	
int(s)	ValueError
float(s)	
complex(s)	
Deljenje z nič:	
/	ZeroDivisionError

Spremembe verzij

Python se razvija. Predstavljena je verzija 2.7. V verzijah 3+ so nastale naslednje glavne spremembe.

Dva tipa celih števil, int in longint, sta združena v en sam tip, int.

Deljenje dveh celih števil z znakom / ne proizvaja več celega števila, pač pa ulomek. Namesto 3/2 = 1 velja torej 3/2 = 1.5. Za celoštevilčno deljenje se uporablja nov operator //,

torej 3 // 2 = 1.

Funkcija range(), ki tvori seznam, je ukinjena. Funkcija xrange(), ki tvori zaporedje števil, je preimenovana v range().

Funkcija input() ukinjena. Funkcija raw_input() je preimenovana v input().

Stavek print x je spremenjen v funkcijo print(x).