TÜRKMENISTANYŇ BILIM MINISTRLIGI

Halkara nebit we gaz uniwersiteti

O.Nurgeldiýew, D.Baýmyradowa, A.Ýazmuhammedowa

LOGIKI WE FUNKSIONAL PROGRAMMIRLEME

Aşgabat - 2014

Giriş

Häzirki wagtda kompýuterler önümçiligiň islendik pudagynda giňden ýaýrandyr. Şonuň üçin hem hasaplaýyş tehnikasy bilen tanyşlyk talyplaryň haýsy hünär boýunça bilim alýanlygyna garamazdan öwrenilýär.

Şu dersiň özi ýöriteleşdirilen bolup, kompýuter tehnikasy bilen işlemekde has ussat bolmaly hünärmenler üçin niýetlenen. Kompýuteriň içki düzümini, onuň gurluşlaryny, olaryň işleýiş düzgünlerini öwrenmegi öz içine almak bilen çäklenmän, ders kompýuteriň apparat taýdan sazlaşykly işlemekligini üpjün etmek, näsazlyklary aýyrmak we düzetmek boýunça görkezmeleri içinde jemleýär.

Dersiň düzüminde operaion ulgamyň işleýişi bilen içgin tanyşmaklyk, bitin sanlar bilen işlemeklik mümkinçilikleri, kompýuter torlary üçin programmalaşdyrmak, INTERNET ulgamy üçin programma düzmek giňden öwrenilýär.

Dersi okatmagyň maksady – mikroprosessor her bir kompýuteriň esasy bölegi bolup durýar, sebäbi hemme arifmetiki we logiki hasaplamalar prosessorda ýerine ýetirilýär. PYTHON dili aşak ýokary dilleriň toparyna girip, dürli gurluşlary dolandyrmak üçin niýetlenen draýwer programmalaryny ýazmaklyga mümkinçilik berýär. Şeýlelikde önümçilikde giňden ulanylýan dürli gurluşlary kompýuteriň kömegi bilen dolandyrmak üçin PYTHON dilini oňat bilmeklik talap edilýär.

Häzirki wagtda Döwletimizde yglan edilen beýik galkynyşlar döwründe halk hojalygynyň hemme pudaklarynda dürli döwrebap, çylşyrymly gurluşlar satyn alynýar we giňden ornaşdyrylýar. Ol gurluşlar bolsa mikroprosessorlaryň kömegi bilen öňden taýýarlanan programmalaryň esasynda dolandyrylýar. Ol gurluşlaryň işleýşine önat düşünmek üçin, şeýle hem olary bejermek üçin ýa-da täzeçe dolandyrmak üçin PYTHON dilini bilmeklik orän zerurdyr.

1.PYTHON DILINDE PROGRAMMIRLEMÄ GIRIŞ

1.1 Python näme?

Python barada(“piton” diýilse gowy ýöne“paýton” hem diýilýär)predmet hökmünde bu programmirleme dilini dörediji golland Guido wan Rossum şeýle diýýär:

“Python – bu interpretirlenýän, obýekte gözükdirilen dinamiki manyly ýokary derejeli programmirleme dili. maglumatlaryň otyrdylan ýokary derejeli gurluşlary dinamiki tipleşdirme we birleşdirme bilen birlikde priloženiýeleri (RAD, Rapid Application Development) çalt işläp taýýarlamakda programmalaryň böleklerini birleşdirmek üçin esasy dil hökmünde hem ulanyp bolar.Python diliniň düzülişini öwrenmek örän ýeňil, onda kodlary okatmak üçin örän uly ähmiýet berilýär, bu bolsa goşmaça programmalary düzmek üçin harçlanýan wagty azaldýar.Paýton programmirleme dili modullary we paketleri goldaýar.Python interpetatory we uly standart kitaphanasy başlangyç we ýerine ýetirilýän kodlar görnüşinde ähli esasy platformalar üçin elýeterli we erkin ýaýradylyp bilner”.

Okuwyň dowamyndaýokarda aýdylanlaryň manysy has-da aýdyň bolar,häzirlikçe bolsa Python–programmirlemäniň uniwersal dildigini bilmek ýeterlik. Onuň öz artykmaçlyklary we kemçilikleri bar,şeýle-de ulanylyş çägi bar.Python programma üpjünçiligine köp dürli meseleleri işlemek üçin ugurlar boýunça ýokary hilli kitaphanalar elýeterli: Internetiň tehnologiýalaryny we tekstlerini gaýtadan işlemek üçin serişdeler, şekilleri gaýtadan işlemek,goşundylary döretmek üçin gurallar, maglumatlar binýadynagirmegiň mehanizmleri, ylmy hasaplary geçirmek üçin paketler, grafiki interfeýsigurmagyň kitaphanasy we.ş.m.Mundan başgada Python dilinde C,C++(we Java) dilleri bilen integrirlemek üçin ýeterlik derejede sada serişdeleri bar.Bu serişdeleri integrirlemek üçin iki usul ulanylýar:bu dillerdäki programmalara interpretatory içinden bina etmegiň üsti bilen we tersine bu dillerde ýazylan kitaphanalary Python programmalarda ulanmak arkaly. Python dili programmirlemäniň birnäçe

nusgalaryny özünde saklaýar:imperatiw (gaýtalanýan,gurluşly,modullaýyn çemeleşmeler);obýekt-gözükdirme we funksional programmirleme.

Pythonprogrammalary (we olaryň protiplerini) döretmek üçin bir giden tehnologiýa diýip hasaplap bolar.Ol C kompilýatorly häzirki zaman platformalaryň ählisinde (32-bitli hem-de 64-bitli) diýen ýaly we Java platformada elýeterli.

Python dilini nädip beýan etmeli?

Bu sapakda Python dilini ulgamlaýyn beýan etmek maksat edilmeýär:munuň üçin ýörite gollanmalar bar.Bu ýerde Python dilini bir wagtda birnäçe nukdaýnazarlarda seretmek hödürlenýär.Bu gözegçilik bolsa mysallaryň üsti bilen ýerine ýetirilýär.Programma döretmek–bu özboluşly özara gepleşik bolup,bu ýerde pogrammist kompýutere maglumaty geçirýär,ol bolsa geçirilen maglumatyň üstünde amallary ýerine ýetirýär.Programmistiň bu ýerine ýetirilýän amallara düşinşine (ýagny "manysyna") semantika diýip bolar.Bu manyny geçirmegiň serişdesine bolsa programmirleme diliniň sintaksisi diýilýär.Geçirilen maglumatlaryň esasynda interpretatoryň edýän işine adatça pragmatika diýilýär.Programma ýazylanda ýokarda agzalan zynjyrda bökdençilikler bolmaly däl.

Sintaksis–dolylygyna resmileşdirilen bölüm,ony sintaksiki diagrammalaryň resmi dilinde hem beýan edip bolýar.Pragmatikanyň aňlatmasy diliň interpretatory bolup durýar.Hut interpretator sintaksisa laýyklykda ýazylan maglumaty okaýar we özünde goýlan algoritm boýunça bu maglumatlary herekete öwürýär.Resmi däl bölegi bolup semantika galýar.Programmirlemäniň iň uly kynçylygy şol semantikanyň,ýagny manynyň resmi beýan etmä geçirmekden ybarat.Python diliniň sintaksisi örän uly serişdelere eýe,olar bolsa programmist bilen interpretatoryň arasyndaky düşünişmekligi aýdyňlaşdyrmaga ýardam edýär. Python diliniň içki gurluşy barada dersiň ahyrynda bellenip geçilýär.

Python diliniň taryhy

Python diliniň döredilişi 1991-nji ýylda Gwido wan Rossum (Guido van Rossum) tarapyndan başlady.Şol döwür ol ýerleşdirilen Amýoba operasion ulgamynyň üstünde işleýärdi.Oňa ulgam çagyryşlary goldamaklygy üpjün edýän giňeltme dili gerekdi.Bu diliň esasyna ABC we Modula-3 alyndy.Onuň adyna bolsa ýylanyň ady däl-de harby-howa-güýçleriňkomediýa filmi bolan "Monti-Pitonyň uçýan sirki" atly filmiň hormatyna at goýdy.Şondan bäri Python dili Gwidonyň işlän guramalarynyň goldawy bilen ösdürildi.Bu diliň has-da kämilleşýän döwri häzirki wagtdyr,çünki bu programmanyň üstünde diňe bir bu dili döredijiler däl,eýsem bütin dünýäniň programmistler bileleşigi hem işleýär.Muňa garamazdan bu diliň ösüş ugry baradaky soňky söz Gwido wan Rossumyň yzynda galar.

Python diliniň programmasy

Python dilinde programma bir ýa-da birnäçe modullardan ybarat bolup biler.Her modul öz gezeginde kodirlenen tekst faýlyny göz-öňine getirýär we 7-bitli ASCII kody bilen ylalaşýar.Has uly bitleri ulanýan kodlaryň atlaryny görkezmek gerek.Mysal üçin,düşündirişleri we setir harplary KOI8-R kodda ýazylan modullar birinji ýa-da ikinji setirde şeýle görnüşde bolmaly:

```
#- * -coding:koi8 – r – * –
```

Ýokardaky setir arkaly Python interpretatory Unicode-setiriň simwollaryny Unicode nädip dogry geçirmelidigini biler.Bu setir bolmasa Pythonyň täze wersiýalary sekiz bitli kodlar duş gelýän her bir modulda duýduruş berer.

Programmany nädip has modully etmelidigini indiki sapaklarda belli bolar.Aşakdaky mysallarda faýllara ýazylan modullaryň bölekleri hem-de Python interpretatory bilen gepleşigiň parçalary ulanylýar.Bu gepleşik häsiýetli çakylyk >>> bilen tapawutlanýar. (#) simwoly düşündirişi soňky setire çenli belleýär.

Python dilinde programma interpretatoryň nukdaý nazaryndan logiki setirlerden ybarat.Bir logiki setir,adatça bir fiziki setirde ýerleşýär,ýöne uzyn logiki

setirler aýdyň (ters gytak çyzygyň kömegi bilen) we aýdyň däl (ýaýlaryň içinde) görnüşde birnäçe fiziki setirlerde ýerleşdirip bolýar:

```
print a, "-örän uzyn setir",\
80,"belgi ýerlerinde ýerleşmeýär".
```

Bellik:

Ähli mysallarda "Python Style Guido" domentine laýyklykda Python dilindäki kody ýerleşdirmegiň "resmi"stili ulanylýar,ony http://python.org saýtynda tapyp bolar.

Esasy algoritmik gurluşlar

Talyplar programmirlemäniň başlangyçlaryny özleşdirendir diýip,algoritmik gurluşlar bilen Python diliniň sintaksisiniň arasyndaky deňeşdirmäni geçirmek ýeterlik diýip hasaplaýarys we Python diliniň sintaksisini sintaksiki diagrammalaryň ýa-da Behus-Nauryň formasynyň kömegi bilen däl-de mysallaryň üsti bilen öwrenmegi makul bildik.

Operatorlaryň yzygiderliligi

Yzygider hereketler programmanyň yzygider setirleri bilen beýan edilýär.Ýene bir zady bellemek gerek,programmada operatorlaryň öňünde boş ýer goýmak möhüm,şonuň üçin hereketleriň yzygiderliligine girýän operatorlaryň öňünde birmeňzeş boş ýer goýulmaly:

```
a=1
b=2
a=a+b
b=a-b
a=a-b
print(a,b)
```

Python programmanyň interaktiw režiminde işlenende programma girizilýär,ol bolsa yzygider hereketlerden ybarat.Ýokarda getirilen mysalda print we baha bermek operatorlary uanyldy.

Şert we saýlaw operatory

Elbetde,programmirlemede ýeke yzygider hereketler bilen oňup bolmaz,şonuň üçin algoritm düzülende şahalanýan algoritm hem ulanylýar:

```
ifa>b:
c=a
     else:
c=b
```

Bu ýerde if iňlis dilinden “eger”, else–“ýogsa” diýmegi aňladýar.Şu ýagdaýda şahalanma operatory iki bölekden ybarat,ondaky operatorlaryň her biri şahalanma operatorlara görä saga süýşirilip ýazylýar.Saýlaw operatoryň mysalyny aşakdaky görnüşde ýazyp bolýar (sanyň alamatyny hasaplamagyň mysaly):

```
if a<0:
s=-1
elif a==0:
s=0
    else:
s=1
```

Bu ýerde elif–bu gysgaldylan else if. Eger gysgaltmadyk bolsa,onda goşmaça şahalanma operatoryny ulanmaly bolardy:

```
if a<0:
s=-1
     else:
if a==0:
s=0
else:
s=1
```

print operatordan tapawutlylykda if-else operatory-düzüm operatory.

Gaýtalanýan operatorlar

Algoritmik gurluşyň üçünji möhüm bölegi sikl bolup durýar.Sikliň kömegi bilen gaýtalanýan hereketleri beýan edip bolýar.Python dilinde sikliň iki görnüşi bar: şertli sikl (käbir hereket ýerine ýetirilende) we parametrli sikl (yzygiderligiň ähli bahasy üçin).Şertli sikli Python dilinde aşakdaky mysal üsti bilen suratlandyrylýar:

```
s=”abcdefghijklmop”
while s!=””:
    print s
    s=s[1:-1]
```

Bu ýerde while operatory Python interpretatoryna şeyle diýýär: “sikliň şerti dogry bolýança sikliň göwresini ýerine ýetirmeli”.Python dilinde sikliň göwresi saga biraz süýşmek bilen tapawutlanýar.Her bir ýerine ýetirlen sikliň göwresine iterasiya diýilýär.Ýokarda getirilen mysalda setiriň başyndan we ahyryndan simwollar aýrylýar we boş setir galýança bu hereket dowam edýär.

Has gowy çeýeligi üçin sikller düzülende break (bes etmek) we continue (dowam etmek) operatorlary ulanylýar.Birinjisi sikli bes etmäge ýardam berýär, ikinjisi bolsa–indiki iterasiýa geçip,sikli dowam etmäge (eger sikliň şerti ýerine ýetýän bolsa).

Indiki mysalda programma faýldaky setirleri okaýar we uzynlygy 5 simwoldan köp bolanlary çykarýar:

```
f=open (“file.txt”,”r”)
while l:
            l=f.read line ()
            if not l:
    break
    if  len (l)>s:
      print l,
   f.close ()
```

Bu mysalda tükeniksiz sikl düzülen, faýldan boş setir (l) alnan badyna sikl bes edýär we faýlyň soňuny aňladýar.

Python dilinde her bir obýekt logiki aňlatma eýe:

Nollar, boş setirler we yzygiderlikler, ýörite obýekt None we logiki aňlatma “ýalan” baha eýe, galan obýektler “çyn” baha eýe.Adatça “çyn” bahany belgilemek üçin 1 ýa-da TRUE ulanylýar.

Bellik:

TRUE we FALSE aňlatmalar logiki aňlatma hökmünde belgilemek üçin Python 2.3-de döredildi.

Parametrli sikl sikliň göwresini yzygiderligiň her bir element üçin ýerine ýetirýär.Indiki mysalda köpeltmek tablisasy çykarylýar:

```
for i in range (1,10):
for y in range (1,10):
     print (“% 2:” % (i * j))
print ()
```

Bu ýerde for siklleri bir-biriniň içinde ýerleşen.ranger funksiýasy [1,10) ýarym açyk diapazondan bitin sanlaryň sanawyny döredýär.Her iterasiýadan öň sikl hasaplananda bu sanawdan indiki baha alynýar.Ýarym açyk diapazonlar Python dilinde kabul edilen.Olary ulanmaklyk has amatly hasaplanýar we programma ýazylanda ýalňyşlary az döredýär.Mysal üçin, range (len (s)) s sanaw üçin indeksleriň sanawyny döredýär (Python dilinde yzygiderlikleriň birinjisiniň elementi 0 indekse eýe).Köpeltmek tablisasy has owadan çykarylmagy üçin formatirleme oprasiýasy % ulanylan (bitin sanlarda şol simwol bölünende galyndyny almak operasiýasyny aňlatmak üçin ulanylýar.Formatirleme setiri (çepden berilýär) C dilindäki print f üçin formatirleme setiri ýaly gurulýar.

Funksiýalar

Programmist öz döredýän funksiýalaryny iki usul bilen kesgitläp biler: defoperatoryň kömegi bilen ýa-da lambda-nyň üsti bilen göni aňlatmanyň içinde.Funksiýalar bilen işlemek Python dilinde funksional programmirleme

boýunça sapakda jikme-jik seredilip geçiler,bu ýerde bolsa funksiýany kesgitlemegiň we çagyrmagyň mysalyny getirsek ýerlikli bolar:

```
def baha  (men, teňňe=0): manat teňňe
return "%i $man.%i teňňe"%(man, teňňe)
print(baha(8,50))
print(baha(7))
print(baha(man=23, teňňe=70))
```

Bu mysalda iki argumentleriň funksiýalary getirilen(olaryň ikinjisiniň bahasy 0). Bu funksiýanyň takyk parametrleri bilen çagyrmagyň birnäçe görnüşleri bar. Bir zady bellemek gerek, funksiýa çagyrylanda ilki pozision parametrler, soňra atlandyrylan parametrler bolmaly. Bahaly argumentler düzgün boýunça adaty argumentlerden soň gelmeli. Return operatory funksiýanyň bahasyny gaýtaryp berýär. Funksiýadan diňe bir obýekti gaýtaryp bolýar, ýöne bu obýektiň özi hembirnäçe obýektlerden durup bilýär. Def operatordan soň gelýän baha adyfunksional obýekt bilen bagly bolup durýar.

Kadadan çykmalar

Häzirki zaman programalarda dolandyrylyşy geçirmek ýokarda getirilen gurluşlar ýaly aýdyň bolanok. Käbir aýratyn ýagdaýlary gaýtadan işlemek üçin (mysal üçin nola bölmek ýa-da ýok faýyldan okatmaga synanyşmak) kadadan çykmalar mehanizmi ulanylýar. Iň gowusy try-except operatoryň manysyny aşakdaky mysal bilen düşüneliň:

```
   try:
   res=
int(open('a.txt').read())/int(open('c.txt').read())
   print (res)
   except IOError:
         print  ("ýalnysh girish-chykysh")
   except ZeroDivisionError:
         print ("0-a bölmek")
   except KeyboardInterrupt:
         print ("Klawiaturadan bes edildi")
   except:
         print ("Yalnysh")
```

Bu mysalda iki faýldan sanlar alynýar we bir-birine bölünýär.Bular ýaly sada hereketleriň netijesinde birnäçe kadadan çykan ýagdaýlar döräp bilýär,olaryň käbirleri except bölümlerde bellenen (bu ýerde Python diliniň standart otyrdylan kadadan çykmalar ulanyldy).Bu mysaldaky soňky except bölegi ýokarda getirilmedik kadadan çykmalary öz içine alýar.Mysal üçin,faýllaryň haýsy hem bolsa birinde simwol baha bar bolsa, int() funksiýasy Value Error kadadan çykmany işe goýberer.Hut şony hem soňky except bölegi bellige alar.Eger ýalňyş çyksa we except böleklerіniň biri ýerine ýetirilse try böleginiň ýerine ýetirilmegi bes edýär.

Programmirlemäniň köp dillerinden tapawutlylykda Python dilinde kadadan çykmalar algoritmleri ýeňilleşdirmek üçin ulanylýar.try-except operatoryny ulanyp, programmist şeýle pikir edip biler: “synanyşyp göreýin, eger ýalňyşsam– except-de kod ýerine ýetiriler”.Bu örän köp aňlatmalarda ulanylýar.Mysal üçin

```
if dict.has_key(key):
value=dict [key]
else:
value=default_value
          Onuň ýerine
try:
value =dict[key]
expect:
  value=default_value
ulanylsa amatly bolýar.
```

Bellik:

Ýokardaky mysal häzirki zaman Python dilinde şeýle ýazylsa gowy:value=dict.get(key,default_value).

Kadadan çykmalary programmalardan hem döredip bolýar.Munuň üçin raise operatory hyzmat edýär.Bu operatoryň ulanlyşyny aşakdaky mysaldan görüp bolar:

```
Class MyError (Exception):
pass
try:
```

```
…
raise MyError,^ my error 1”
…
expect MyError,x:
print  (“Yalnysh:”,x)
```

Ähli kadadan çykmalar köp basgançakly synplara düzülen, şonuň üçin Zero Division Error ýalňyşy ArithmeticError hökmünde bellenilip bilner,eger degişli expect bölegi öňden gelýän bolsa.

Tassyklama ýörite assert operatory ulanylýar.Eger ondaky berlen şert ýalňyş bolsa, ol Assertion Error döredýär.Bu operator programmany gaýtadan barlamak üçin ulanylýar.Optimizirlenen kodda ol ýerine ýetmeýär, şonuň üçin onda algoritmiň logikasyny gurmak bolanok.

Mysal üçin:

```
c= a+b
assert c==a+b
```

Ýokarda getirilen operatordan başga-da try-finally görnüşi hem bar.Ol eýelenen resurslary boşatmak üçin ulanylyp biliner, bu bolsa bar bolan ýalňyşlara garamazdan hökmany ýerine ýetirilmegi talap edýär:

```
try:
...
finally:
 print (“Gaýtadan işlemeklik ygtybarly jemlenen”)
```

try- expect we try-finally görnüşleri garyşdyrmak bolanda.

Maglumatlaryň gurnalan görnüşleri

Öň belläp geçişimiz ýaly Python dilinde ähli maglumatlar obýektler hökmünde göz-öňine getirilen.Olaryň atlary bu obýektlerediňe salgylanma bolup duryarlar we özünde hiç hili görnüş boýunça işi ýerine ýetirmeýärler.Gurnalan görnüşleriň bahalary diliň sintaksisinde ýörite goldawa eýe: setriň, sanyň, hatyň, toplumyň, sözlügiň (we olaryň görnüşleriniň) bahasyny ýazyp boýar. Obýektler öz gezeginde

üýtgeýän bolup bilýärler.Mysal üçin,Python dilinde setirler üýtgemeýän görnüşde,şonuň üçin setirleriň üstündäki amallar täze setirleri döredýärler.

Aşakda gurnalan görnüşleri getirilen:

- ýörite görnüşler:None,Notimplemented we Ellipsis;
- sanlar;
  - bitin
    - ✓ adaty bitin int
    - ✓ bitin uly sanlar long
    - ✓ logiki bool
  - ýüzýän bahaly san float
  - kompleks san complex
- yzygiderlikler;
  - üýtgemeýänler:
    - ✓ setir str;
    - ✓ Unicode-setir unicode;
    - ✓ toplum tuple;
  - üýtgeýänler:
    - ✓ list sanawy;
- görkezmeler:
  - sözlük dict
- çagyryp bolýan obýektler:
  - funksiýalar (ulanyjy we gurnalan);
  - generator-funksiýalar;
  - usullar (ulanyjy we gurnalan);
  - synplar (täze we “klassiki”);
  - synplaryň nusgalary (eger_call usula eýe bolsalar);
- modullar;
- synplar (ýokarda seret);
- faýllar file;
- kömekçi görnüşler buffer,slice.

Islendik obýektiň görnüşine type() gurnalan funksiýanyň kömegi bilen bilip bolar.

int we long görnüşler

Iki görnüş bolan:int (bitin san)we long (bitin uly san) bitin sanlary göz-öňüne getirmek üçin nusga bolup hyzmat edýärler.Olaryň birinjisi ulanylýan arhitektura üçin C kompilýatordaky long görnüşine gabat gelýär.San bahalary hasaplama ulgamynda 8,10 ýa-da 16 esas bilen ýazyp bolar.

```
# Bu bahalarda ýazylan san 10
print (10,012,0xA,10 L)
```

Sanlaryň üstündäki amallaryň toplumy –manysy boýunça hem-de belgilenşi boýunça standart bolup durýar:

```
>>> print (1 + 1, 3 - 2, 2*2, 7/4, 5%3)
2 1 4 1.75 2
>>> print (92** 1000)
613518051623162272354408108837601281587845838413484732307851544005023276843040637734371254656907315755646608199808716371122525831018218649221686578707397724939388829758212200345297438424955818969855109180336888145726560315643001894285406768314234427693390043772637310760263932092015222741942483466117214624862660557761865833204378286129932725223064215060905286318171896234398580771135574285813246936909248058111458502076996307957767433119268496191892178384379822205530726181058675778240109866754413947987203068378609162631331132160620718110028089959569119381053416788584786945966233021158436214944477287775835313772788376878349172262668798745985225940108817329558941278243230008913146437828133582188108748128791067055261667188160477375542886176100509856058108471309857049793387186551193111984827016326568617875467038325370630687210267707621367352914196316640783382536286802719210679098798443539262834582113239600193379207946379796365801492683992745795071039877832769192625635944412684675613056262896670135477628100856590739402057547158894043240643174636507601835374377713284084359624382774839958045712876048644417909460218030187092316127364527397523857041007659428054856794957890098943222724326835548114179945126589407 4
```

```
129471193367604999238348054269040491995447263971468946791153686878159866276786470438905540327386064546904277853867352064252719542948887675569780769496599308661892663287503332512922734951172910267122179676550477066986322067354842636806442698856389923140129642928338513247509711157054434283764718674921206677145360045622917608137447311482085964120453035845065144829438316745903245568993360345806205787384890608815932611599039471762745126201603646160279539198792650441734977383217245053107953081876630871234989763265529566129955669854092114132498790642438050754933565619757728644195094851141175793222341352471134552414928835663089726766674307916789031765480888463201014461348492471183913183940502980433495654427266108555801711222116411662944278937 6
>>> print(3<4<6,3>=5,4==4,4!=4) # bu ýerde deňeşdirme çynmy ýa-da ýalanmy görkezýär.
TRUE FALSE TRUE FALSE
>>>print(1<<8,4>>2,~4) # bitleýn süýşme we inwersiýa
256 1-5
>>>for i,j in (0,0),(0,1),(1,0),(1,1):
      print (i,j,":",ij,i^j) # bitleýn amallar.
0 0 : 0 0
0 1 : 0 1
1 0 : 0 1
1 1 : 1 0
```

Bitin sanyň int görnüşiniň bahasy – 2147483648-den 2147483647- çenli diapazonyň çäginde bolmaly,bitin uly sanyň takyklygy bolsa elýeter ýadyň göwrümine bagly.

Eger meseläniň dowamynda int görnüşiniň çäginden uly baha alynýan bolsa, ol long görnüşine öwrülýär:

```
>>>type (-2147483648)
<class' int'>
>>>???
```

Şeýle-de hemişelikler ýazylanda seresap bolmaly.Sanyň başyndaky nullar-hasaplamanyň sekizlik ulgamy bu ýer-de 8 sifr ýok:

```
>>>008
syntax Error:invalid token
```

float görnüşi

Ulanylýan arhetektura üçin C dilindäki double görnüşine gabat gelýär.Iki görnüşde ýazylýar ya bitin sany nokat bilen bölmek arkaly ýa-da eksponenta bilen;

```
>>>pi=3.1415926535897931
>>>pi**40
7.6912142205157e+19
```

Arifemtiki amallardan başga-da math modulynyň kömegi bilen matematiki amallary hem ýerine ýetirip bolýar.

Bellik:
Finans hasaplamalar üçin degişli görnüşi hasaplamak ýerlikli bolar.
Peýdaly gurnalan funksiýalardan round ( ), abs ( ) ýatlap bolar.

complex görnüşi

Hyýaly bölegiň bahasy j-ni goşmak bilen berilýär(hyýaly birlikler köpeldilýär):

```
>>>-1j*-1j
(-1+oj)
```

Arifmetiki amallardan başga cmath modulyň amallaryny hem ulanyp bolar.

bool görnüşi

Logiki ululyklaryň tassyklanan bahalary üçin ulanylýar.Bu görnüşe diňe iki baha degişli:True (çyn) we False (ýalan).Öň belleýşimiz ýaly,Python diliniň islendik obýekti çyn baha eýe, logiki amallara aşakdaky ýaly mysal getirip bolar:

```
>>>for i in ( False,True):
  for j in (False,True):
  print (i,j,’’:’’,i and j, i or j,not i)
False False: False False True

False True: False True True

True False: False True False

True True: True True False
```

Python dilinde eger and we or amallaryň birinjisine görä netije aýdyň bolsa ikinjisi hasaplanmaýar.Şeýlelikde birinji amal çyn bolsa,ol or amalyň netijesi bolup galýar, ters ýagdaýda ikinji amal ýerine yetirilýär.and amal hem edil şu düzgünde ýerine ýetirilýär.

String we unicode görnüşleri

Python dilinde setirler iki görnüşde bolýar. Adaty we Unicode setirler.Aslynda setir–bu simwollaryň yzygiderligidir.

Setir hemişelikleri programmada setir literallaryň kömegi bilen berip bolar.Literallar üçin hem apostotlar hem-de goşa dyrnaklar ulanylýar.Köp setirli literallar üçin üçeldilen apostotlar ya-da üç sany goşa dyrnaklar ulanylyp biliner.Setir literallaryň içinde dolandyryjy yzygiderlikde ters gytak çyzyk (\ ) bilen berilýär.Setir literallary ýazmagyň mysaly:

```
s1= “setir 1”

s2= ‘setir 2\n setiriň içindäki terjime bilen’

s3=”””setir 3

Setiriň içindäki terjime bilen """

u1 =u ‘\ u043f \ u0440 \ u 0438 \ u0432 \ u0442’

u2 = u ‘ýene mysal’
```

Setirler üçin ýene bir görnüş bar:Işlenilmedik setir literallar.Bu literallarda ters gytak çyzyk we yzyndan gelýän simwollar ýörite simwollar ýaly aňlanylmaýar we bolşy ýaly çykarylýar.

my_re = r “ (\ d) = \ 1 ”

Setirleriň üstündäki amallar birleşdirmäni “+” gaýtalamany “ * ” we formatirlemäni “ % ” öz içine alýar.Setirleriň üstünde geçirilýän amallaryň mysaly aşakda getirilen:

```
>>> “ A ” + “ B “
‘A B’
>>> “A” * 10
‘AAAAAAAAAA’
>>> ” % s  %i” %  ( “abc” , 12 )
‘abc 12’
```

tuple görnüşi

Obýektleriň üytgemeýän yzygiderliklerini göz-önüne getirmek üçin tuple görnüşi ulanylýar.Onuň bahasy adatça tegelek ýaýlarda ýazylýar. Onuň mysaly aşakda getirilen:

```
p = (1.2, 3.4, 0.9)
for s in "one", "two", "three":
    print (s)
    one_item = (1,)
    empty = ()
    p1 = 1, 3, 9
    p2 = 3,8,5
```

list görnüşi

Python dilinde erkin görnüşli elementleriň massiwi ýok, onuň ýerine sanowlar ulanylýar. Olary kwadrat ýaýlarda ýazylýan literallaryň kömegi bilen berip bolýar. Sanow ulanylan mysal aşakda getirilen

```
lst1 = [1,2,3]
lst2=[x** 2 for x in range (w) if x 62 ==1]
lst3=list (“ abcde ”)
```

Yzygiderlikler

Aşakda yzygiderlileriň esasy usullar jemlenen.Bir zady bellemek gerek , yzygiderlikler üytgeýän we üýtgemeýän bolýarlar .Üýtgeýän yzygiderlikleriň , usullar birneme köpräk.

Ýazylyşy	Düşündirilişi
len (s)	s yzygiderligiň uzynlygy
x in s	Yzygiderligiň elementiniň degişliligini barlamak .
x not in s	= not x ins s
s + s1	Yzygiderlikleriň birildirilişi
s*n ýa-da	n gezek gaýtalanan s yzygiderlik.Eger n<0 bolsa,boş yzygiderlige gaýtarýar.
s[i]	s-iň i-nji elementini ya-da len(s)+i-n gaýtarýar,eger-de i<0
s[i:j:d]	s yzygiderlikden i-den j-e çenlid ädim bilen kesim
min [s]	s-ň in kiçi elementi.
max (s)	s-ň iň uly elementi.

Üýtgeýän yzygiderlikler üçin.

s [i] = x	3 sanowyn i-nji elementi x-e çalşyrylýar.
s[i;j;d]=t	d ädim bilen i-den j-e çenli t sanow bilen çalşyrylýar.
del s[i;j;d]	Kesimdäki elementleri ýok etmeli.

Yzygiderlikler bilen işlemek üçin käbir usullar tablisada üýtgeýän yzygiderlikleriň (mysal üçin sanowlar) usullary getirilen.

Usullar	Düşünjeler
append (x)	Yzygiderligiň soňunda element goşýar.
Count (x)	x-e den elementlerin mukdaryny sanaýar.
Extend (x)	Yzygiderligiň soňunda s yygiderligi goşýar.
Index (x)	Iň kiçi i-ni gaýtarýar,s[i]= =x,Eger s-de x tapylmasa ValueError herekete gelýär.
Insert (i,x)	X elementi i-nji aralyga goýýar.
Pop ([i])	i-nji elementi yzygiderlikden ýok edip ony gaýtarýar.

Reverse ( )	S-iň elementleriniň tertibini ters ýagdaýa üýtgedýär.
Sort ([cmpfunc])	s-iň elementlerini saýlaýar.cmpfunc deňeşdirmäniň öz funksiýasy hem görkezilip biliner.

Indeks boýunça elementi almak we kesmek

Yzygiderligiň aýratyn elementini almak üçin kwadrat ýaýlar ulanylýar,bu ýaýlarda bolsa indeksi görkezýän aňlatma bolýar.Python dilinde yzygiderlikleriň indeksleri nuldan başlanýar.Otnisitel indeksler elementleri soňundan sanamak üçin hyzmat edýär.(-1- soňky element).Muny mysalyň üsti bilen göreliň:

```
>>> s = [0,1,2,3,4]
>>> print (s[0],s[-1],s[3])
0 4 3
>>> s[2]=-2
>>> print (s)
[0, 1, -2, 3, 4]
>>> del s[2]
>>> print(s)
[0, 1, 3, 4]
```

Bellik:

Elementleri diňe üytgeýän elementlerden ýok edip bolar we yzygiderlikleriň aýlaw sikliniň içinde etmek bolanok.

Python dilinde yzygiderlikden kesim alnanda elementleri nomerlemeli däl-de olaryň boş aralyklaryny belgilemeli.Yzygiderligiň nolunjy elementiniň (indeks boýunça) öňündäki boş aralygy 0 sana -1 we ş.m bolýar.Otrisatel bahalar baş aralyklary setiriň soňundan sanaýar.Kesimler şeýle görnüşde ýazylýar:

Yzygiderlik [başy,soňy,ädim]

Bu ýerde başy–kesimiň başy,soňy– kesimiň soňy,ädim– näçe ädimden kesimiň soňuna ýetmeli.

Kesimler bilen işlemegiň mysaly aşakdaky görnüşde bolýar.

```
>>> s=[0,1,2,3,4,5,6,7,8,9]
>>> s[0:3]
[0, 1, 2]
>>> s[-1]
9
>>> s[::3]
[0, 3, 6, 9]
>>> s[0:0]=[-1,-1,-1]
>>> s
[-1, -1, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> s
[-1, -1, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Bu mysaldan görnüşi ýaly,kesimleriň kömegi bilen islendik kiçi setirli elementleri ýok etmek üçin ýa-da elementi belli bir ýere goýmak üçin bermek amatly.

dict görnüşi

Sözlük (heş baglanyşykly massiw)-açar-baha jübütleri saklamak üçin niýetlenen maglumatlaryň üýtgeýän gurluşy bu ýerde baha açar bilen kesgitlenýär.Açar hökmünde maglumatlaryň üýtgeýän görnüşi (san,setir,toplum we başg.) çykyş edip biler.Açar-baha jübütleriniň tertibi erkin alynýar.Aşakda sözlük üçin literal we sözlük bilen işlemegiň mysaly getirilen:

```
d = {1:'one',2:'two',3:'three',4:'four'}
d0={0:'zero'}
print (d[1]) #açar boýunça alynýar
d[0] = 0 #açar boýunça baha alynýar
del d[0]    # berlen açar bilen açar-baha jübüti ýok
edilýär
print (d)
for key, val in d.items():  #tutuş sözlük boýunça
aýlaw
      print(key, val)
for key in d.keys():    # sözlügiň açary boýunça
aýlaw
      print (key, d[key])
for val in d.values():  #sözlügiň bahasy boýunça
aýlaw
      print (val)
d.update(d0)        #beýleki sözlükden doldurylýar
```

```
print (len(d))  #sözlükdäki jübütleriň sany
one
{1: 'one', 2: 'two', 3: 'three', 4: 'four'}
1 one
2 two
3 three
4 four
1 one
2 two
3 three
4 four
one
two
three
four
5
```

file görnüşi

Bu görnüşiň obýektleri daşky maglumatlar bilen işlemek üçin niýetlenen. Adaty görnüşde–diske ýazylan faýl.Faýlly obýektler esasy usullary goldamaly:read ( ),write ( ), readline ( ), readlines ( ),seek ( ), tell ( ),close ( ) we başg.

Aşakda faýly göçürmegiň mysaly getirilen:

```
f1=open ( "file 1.txt", "r" )
f=2 open ("file 2.txt", "w" )
for line in f1.readlines ( ):
      f2.write ( line )
      f2.close ( )
      f1.close ( )
```

Ýene bir zady bellemek gerek, Python dilinde faýllaryň özünden başga faýllara meňzeş obýektler hem ulanylýar.Mysal üçin, (URL) salgylanma boýunça maglumatlary aşakdaky setirleri ulanyp filer.txt faýla göçürüp bolar:

```
import  urllib

f1=urllib.urlopen ( "http;//python.onego.ru" )
```

Modullar,synplar,obýektler we funksiýalar barada indiki Sapaklarda gürrüň ediler.

Aňlatmalar

Programmirlemäniň häzirki zaman dillerinde maglumatlaryň köp bölegi aňlatmalarda amala aşyrylýar.Aňlatmalaryň ýazylyş düzgünleri köp programmirleme dillerinde,şol sanda Python dilinde hem birmeňzeş.

Aňlatmalaryň ýazylyş düzgüni aşakdaky tablisada getirilen:

Python dilinde ýazylyşy	**Ady**
lombda	lýambda-aňlatma
or	ýa-da –logiki aňlatma
and	we-logiki aňlatma
not x	däl-logiki aňlatma
in, not in	degişliligini barlamak
is,is not	meňzeşligini barlamak
<, <=, >,>=,!=, ==	deňeşdirmeler
|	bit boýunça ýa-da
^	bit boýunça ýa-da aýyrmak
&	bit boýunça we
<<,>>	bit boýunça süýşmeler
+,-	goşmak, aýyrmak
*, /, %	köpeltmek,bölmek,galyndy
+x,-x	unar goşmak we alamaty çalyşmak
~ x	bit boýunça däl
**	derejä götermek
x.nyşan	nyşana salgylanma
x [indeks]	elementi indeksi boýunça almak
x [baş:soňy]	kesimi bellemek (başy we soňy)

f (argument,…)	funksiýasy çagyrmak
(…)	ýaýlar ya-da toplum
[…]	sanaw ýa-da sanawa girizmek
{açar:baha,...}	açar-baha jübütleriň sözlügi
‘aňlatma’	setire özgertmek (repr)

Şeýlelikde,amallaryň ýerine ýetiriliş tertibi şeýle düzgünler bilen kesgitlenýär:

1.Derejä götermekden galan ähli ammallar çepden saga ýerine ýetirilýär.

2.a<b>c…y<z deňeşdirme zynjyry aşakdaky bilen deň: (a<b) and (b<c) and … (y < z).

3.Funksiýalaryň argumentleri,sanawlar,sözlükler we ş.m. üçin aňlatmalar yzygiderlikde çepden-saga hasaplanýar.

Amallaryň haýsy biri birinji ýerine ýetirilmelidigi aýdyň bolmasa ýaýlar ulanylýar.Şol bir simwollar dürli amallar üçin ulanylýandygyna garamazdan, amallaryň yzygiderligi üýtgemeýär.Mysal üçin, %-amaly *-amaly ýaly birinji hasaplanylýar, şonuň üçin indiki mysalda ýaýlary hökmanulanmaly,sebäbi köpeltmek amaly formatirleme amalynda öz ýerine ýetirmeli:

```
print "%i" % (i*j)
```

Köp ýagdaýlarda aňlatmalar baha bermek operatoryň sag boleginde ýerleşýärler.Python dilinde (mysal üçin C-dilinden tapawutlylykda baha bermek amaly ýok,şonuň üçin belginiň öňünde diňe identifikator,kesim ya-da sanaw bolup biler.

Atlar

Atlar latyn harplar bilen ýa-da aşagyny çyzmak bilen başlanyp biliner,soňra sanlary hem ulanyp bolýar. Identifikator hökmünde diliň açar sözlerini ulanmaly däl. Açar sözleri aşakdaky ýaly bilip bolar:

```
>>> import keyword
>>> keyword.kwlist
```

```
['False','None','True','and','as','assert','break','class','continue','def','del','elif','else','except','finally','for','from','global','if','import','in','is','lambda','nonlocal','not','or','pass','raise','return','try','while','with','yield'].
```

Aşagy bir ýa-da iki çyzyk bilen başlanýan atlar aýratyn mana eýe. Bir çyzyk bilen başlaýanlar ýerli ulanylyşy aňladýar we modulyň çäginde ulanylmaly däl. Başynda we soňunda iki çyzyk bilen belgilenenler adatça ýörite baha eýe bolýarlar–bu barada obýektegönükdirilen programmirle Sapaksynda gürrüň ediler.

Programmanyň her bir nokadynda interpretator adyň 3 sany giňişligini görýär: lokal,global we gurnalan.

Atlaryň giňişligi – bu atlardan obýektlere şöhlelendirilmegidir.

Python dilinde käbir üýtgeýäniň bahasy nädip tapylýandygyny düşünmek üçin koduň blogy diýen düşünjäni girizmeli. Python dilinde koduň blogy diýip funksiýanyň,synpyň ya-da modulyň kesgitlenilşine aýdylýar.

Lokal atlar – kodun berlen blogunda baha berlen atlar.

Global atlar – moduly kesgitleýän ýa-da global opetatory bilen berilýän kodun blogy derejesinde kesgitleýän atlar, gurnalan atlar `_builtins_`ýörite sözlükden alynýan atlar.Koduň bolgynda ulanylýan ýöne koduň çägindaki baha bilen bagly bolmadyk üýtgeýänlere erkin üýtgeýänler diyilýär.Üýtgeýänleri bloguň islendik ýerindäki obýekt baglaşdyryp bolýanlygy üçin,ony ulanylmazdan öň baglaşdyrmaly ýogsa `NameError` ýalňyş çykar.Atlary bahalar bilen baglaşdyrylmak baha bermek,for,impot operatorlarynda bolup geçýär.

Baglaşdyrmak bilen bagly köp aýratynlyklar bar. Bu aýratynlyklara programmalara bagly bolmaz ýaly aşakdaky kadalary berjaý etmeli :

1. Hemişe üytgeýäni baha bilen ulanyşdan öň baglaşdyrmaly.
2. Global üýtgeýänlerden daşda durmaly we maglumatlaryparametrleri görnüşde geçirmeli.Modul derjesinde global hökmünde diňe konstant-atlar,synplaryň we funksiýalaryň atlary galmaly.
3. `From` moduly `import*` ulanmaly däl – bu beýleki modullardan atlary gysýar,funksiýanyň kesgitlemegiň çäginde bolsa gadagan.

Adyň obýekt bilen baglanyşygyny aýyrmak üçin `del` operatory ulanylýar. Bu ýagdaýda,eger obyekte başga salgylanma ýok bolsa, ol ýok edilýär.

Programmirlemäniň stili:

Programirlemäniň stili – programma kodunyň gurluşyna we görnüşine topar bolup işleýän programistler tarapyndan goýulan goşmaça çäklendirmeler. Onuň

kömegi bilen ulanmak üçin amatly, ýeňil okalýan we täsirli programmalary alyp bolýar. Programmanyň görnüşinde esasy çäklendirmeleri programmirleme diliniň sintaksisi berýär we onuň bozulmagy sintaksik ýalnyşlary döredýär.

Prorammirleme stili kody ýazmagyň ähli düzgünlerine degişli:

- Obýektleri görnüşine,maksadyna, görnüş çägine baglylykda atlandyrylyşy;
- Funksiýalary usullary,synplary, modullary taýýarlamak we olary programmanyň koduna girizmek;
- Programmalary kesgitli häsiýetli modullara düzmek;
- Işlenen maglumaty girizmegiň usuly;
- Dürli kompýuter platformalarynda işlenip düzülýän programmalarynyň ylalaşyjylyk derejesine baglylykda funksiýalary (usullary) ulanmak;
- Howpsuzlyk ýagdaýlary göz öňüne tutup ulanylýan funksiýalary çäklendirmek.

Python dil üçin Gwido wan Rossum resmi stili işläp düzdi.

“Python Style Guide” mazmuny bilen http;www.python.org/doç/essays/styleguide.html salgy boýunça tanyşyp bilersiňiz.

Bu stiliň möhüm düzgünnamalary bilen aşakda tanyşyp bilersiňiz.

- 4 probel bilen süýşirilip ulanmak maslahat berilýär.
- Fiziki setiriň uzynklygy 79 simwoldan uly bolmaly däl.
- Uzyn logiki setirleri bölünse gowy.Bölünen setiriň ikinji ýarymynyň öňünde goýulýan boş ýer ýaýlar boýunça ýa-da birinji ýarymyndaky operand boýunça deňlenmeli.Python-mode režimindäki Emacs tekst redaktory we IDE Python-programmalarda gerekli boş ýerleri awtomatiki goýýar:

```
def draw(figure, color="White", border_color="Black",
size=5):
if color == border_color or \
size == 0:
raise "Bad figure"
else:
_draw(size, size, (color,
       border_color))
```

- Açylýan ýaýyň soňunda ýa-da ýapylan ýaýyň öňunde,oturyň,nokatly oturyň öňinde,funksiýa ýa-da indeks aňlatma ýazylanda açylýan ýaýyň öňünde probel goýmak maslahat berilmeýär.
- Ilki ýerine ýetirilýän amallaryň töwereginde çepden we sagdan deň sanly probeller goýulýar.Eger amallar ilki ýerine ýetirilmeýän bolsa,deňeşdirme we baha bermek operatoryň töwereginde bir probel goýulýar.

Düşündirişleri ýazmagyň düzgünleri.

- Düşündirişler koduň wajyp ýagdaýyny şöhlelendirmeli:

Python diliniň ýazary iňlis dilinde geplemeýän programistlere düşündirişleri iňlis dilinde ýazmagy haýýş edýär.

“#” simwoldan soň bir probel goýulmaly. Abzaslary şol bir derejede “#” simwoly setir bilen oýaryp bolýar. Blokly düşündirişleri boş setirler bilen aýyryp bolýar.

- Belli bir setire degişli düşündirişleri köp ulanmak maslahat berilmeýär. “#”simwoldüşündirilýän operatordan azyndan iki probel bilen aýrylmaly.

Modulyň daşynda ulanmak üçin niýetlenen ähli modullar, synplar, funksiýalar we usullar özleriniň ulanylyşyny, giriş we çykyş parametrleri suratlandyrýan resminamalaşdyryş setirleri bolmaly.

- Aýratyn programma üçin resminamalaşdyryş setiri programmada ulanylýan açarlary, argumentleriň we üýtgeýänleriň bahalaryny we ş.m. maglumaty düşündirmeli.
- Resminamalaşdyryş setirleri üçin hemme ýerde üç sany goşa dyrnak (” ” ”) ulanylmagy maslahat berilýär.
- Modul üçin resminamalaşdyrylyşda eksportirlenýän funksiýalar, synplar, ýalňyşlar we beýleki obýektler hersine bir setirden sanalyp geçilmeli.
- Funksiýa ýa-da usul üçin resminamalaşdyrma setiri funksiýanyň hereketlerini, onuň giriş parametrlerini we ýüze çykýan ýalňyşlary beýan etmeli.
- Synp üçin resminamalaşdyrylyşda ulanylýan usullar we atributlar sanalyp geçilmeli.
- Modullaryň atlary setir harplar bilen ýazylsa gowy, mysal üçin, shelve, string, ýa-da birinji harpy baş harp edip bolýar, String IO, UserDict.
- Açar sözleri at hökmünde ulanmaly däl, ýöne, eger at hökmünde ulanmaly bolsa, onda adyň soňunda bir gezek aşagyny çyzmaly:class_.
- Adatça synplary sözleriň birinji harplaryny baş harp bilen ýazyp belleýärler, mysal üçin, Tag ýa-da HTTServer.
- Kadadan çykma atlaryň düzüminde “ error “ (ýa-da “ warning “) sözi bar. Gurnalan modullar bu sözi setir harplar bilen ýazýarlar (es.error), ýöne baş harp bilen hem ýazyp bilýärler: (distutils.DistutilsModuleError).
- Global üýtgeýänleriň atlary (eger ulanylýan bolsa) aşagyny çyzyp başlamaly, çünki olar from-import operatoryň kömegi bilen moduldan çykarylyp bilner.
- Funksiýalar her hili-görnüşde ýazylyp bilner.Has möhüm funksiýalary baş harplar bilen, goşmaçalary bolsa setir harplar bilen berip bolar.

- Usullaryň atlary hem edil funksiýalaryň atlary ýaly ýazylýar.
- Hemişelikleriň atlaryny baş harplar bilen ýazmaly:RED, GREEN, BLUE.

Netije

Bu Sapakda diliň sintaksisi mysallaryň üsti bilen görkezilen.Python diliniň esasy operatorlara, aňlatmalara we maglumatlaryň gurnalan görnüşleriniň köpüsine döredilip geçildi.Python diliniň atlar bilen işleýiş düzgünleri gysgaça düşündirilen.Python diliniň resmi programmirleme stiliniň kadalary getirilen.

2. PYTHON DILINIŇ ESASY STANDART MODULLARY

Python diliniň esasy artykmaçlyklarynyň biri modullarynyň we paketleriň uly kitaphanasynyň barlygy.

Modul düşünjesi

Standart kitaphananyň modellerini öwrenmezden öň Python dilinde modulyň kesgitlenilişini öwrenmek gerek.Programmirlemä bolan modul çemeleşmä laýyklykda uly mesele birnäçe kiçi meselelere bölünýär,olaryň her birini bolsa aýratyn modul işleýär.Dürli usullarda modullaryň ölçeglerine dürli çäklenmeler berilýär,ýöne programmanyň modul gurluşy düzülende olaryň arasyndaky baglanyşyk näçe az bolsa,şonça gowy.Elementleriň köp baglanyşyklaryny öz içinde saklaýan synplaryň we funksiýalaryň toplumyny bir modulda ýerleşdirilse dogry bolardy.Ýene bir bellik:modullary täzeden ýazylandan,ulanylany aňsat bolmaly.Bu diýmek, modulyň amatly interfeýsi bolmaly.Python dilinde bir meselä degişli modullaryň toplumyny pakete ýerleşdirip bolýar.Şeýle paketleriň mysaly hökmünde xml paketi getirip bolar,onda XML gaýtadan işlemek üçin dürli modullar ýygnalan.

Python programmirleme dilinde modul obýekt-modul hökmünde göz-öňüne getirilen, olaryň alamatlary modulda kesgitlenen atlar bolup durýarlar:

```
>>>import datetime
>>>d1=datetime.date (2004,11,20)
```

Bu mysalda datetime moduly importirlenýär.import operatoryň işiniň netijesinde datetime atly obýekt döreýär.Python dilindäki programmalarda ulanylýan modullar öz gelip çykyşy boýunça ikä bölünýärler: sada (Python dilinde yazylanlar) we beýleki dillerde ýazylan giňeltme modullary (adatça C dilinde ýazylan). Olar bir-birinden diňe çaltlygy bilen tapawutlanýarlar.Bu modullar bir-birinden şeýle tapawutlanýarlar,mysal üçin,pickle we cPickle modullary.Adatça Python dilindäki modullar giňeltme modullara garanyňda has çeýe.

Python dilindäki modullar

Modul başlangyç kodly aýratyn faýl görnüşinde düzülýär.Standart modullar katalogda ýerleşýär,diliň interpretatory ol ýerden degişlisini tapyp biler.Bu kataloglary sys.path üýtgeýäniň bahasynyň üsti bilen görüp bolar:

```
>>> import sys
>>> sys.path
['C:/Python33','C:\\Python33\\Lib\\idlelib','C:\\Windows\\system32\\python33.zip','C:\\Python33\\DLLs','C:\\Python33\\lib', 'C:\\Python33', 'C:\\Python33\\lib\\site-packages']
```

Programma işe girizilende modullaryň gözlegi açylan katalogda ýerine ýetirilýär. Python dilinde modullar import operatoryň kömegi bilen işe girizilýär.Bu operatoryň iki görnüşi bar: import we from-import:

```
import os
import pre as re
from sys import argv,environ
from string import *
```

Birinji görnüş ulanylynda görünýän çäk diňe modulyň obýektine salgylanýan at bilen baglaşylýar,ikinji ulanylanda – görünýän çäk modulyň obýektleriniň atlary bilen baglaşylýar.Obýektiň baglaşýan adyny as-iň kömegi bilen import edilende çalşyryp bolýar.Birinji ýagdaýda modulyň atlarynyň giňişligi aýratyn atda saklanýar we modulyň takyk bir adyna barmak üçin nokat ulanylýar.Ikinji ýagdaýda atlar açylan modulda kesgitlenen ýaly edip ulanylýar:

```
os.system ("dir")
digits =re.compile ("Id+")
print (argv [o],environ).
```

Ulanylan moduly gaýtadan import edilende çalt ýerine ýetirilýär,sebäbi ol interpretatortarapyndan keşirlenýär.Ýüklenen moduly reload ( ) funksiýasynyň kömegi bilen işe goýberip bolar:

```
import mymodule
...
reload (mymodule)
```

Ýöne bu ýagdaýda öň ýüklenen modulyň düzümindäki obýektler öňki ýagdaýynda saklanyp galar.

Gurnalan funksiýalar

Python programmirleme dilinde importyň goşmaça amallaryndan başga ýüklerde gurnalan obýektler,esasanam,funksiýalar we kadadan çykmalar bar.Funksiýalary,amatlylyk üçin,şertli kategoriýalara bölünen:

1.Görnüşleri we synplary özgerdýän funksiýalar:
coerce,str,repr,int,list,tuple,long,float,complex,dict,super,file,bod,object.

2.Setir we san funksiýalary:
abs,divmod,ord,pow,len,chr,unichr,hex,oct,cmp,round,unicode.

3.Maglumatlary gaýtadan işlemegiň funksiýalary: apply, map, filter, reduce, rip, range, xrange, max, min, iter, enumerate, sum.

4.Häsiýetleri kesgitlemegiň funksiýalary:hash,id.callable,issubclass,isinstance,type.

5.Içki gurluşlara barmak funksiýalar:locals, globals, vars, intern, dir.

6.Kompilýasiýanyňwe ýerine ýetirmegiňfunksiýalary: eval, execfile, reload, import, compile.

7.Giriş-çykyş funksiýalary:input, raw input, open.

8. Atributlar bilen işlemegiň funksiýalary: getattr,setattr,delattr,hasattr.

9.Synplaryň usullaryna "bezeg" berýän funksiýalar: staticmethod, classmethod, property.

10.Beýleki funksiýalar: buffer,slace.

Bellik:Bellenilenfunksiýanyňargumenntlerini we netijesini Python interpretatoryň interaktiw sessiýasynda takyklap bolar:

```
>>> help(len)
Help on buil-in Gunction len in module built-ins:

Len (…)

     Len (object)→ integer

Return the number of items of a seguence or mapping.
```

Tipleri we synplary özgerdýän funksiýalar

Bu katigoriýadaky funksiýalar wesynplar maglumatlaryň görnüşlerini özgertmek üçin hyzmat edýärler. Muňa aşakdakyny mysal getirip bolar:

```
>>> int (23.5)
23
```

```
>>> float ('12.345')
12.345
>>> dict (['a',2), ('b',3)]
{'a':2,'b':3}
>>> object
<class 'object'>
>>> class  My Object  (object):
…pas
…
```

Setir we san funksiýalary

Funksiýalar setir ýa-da san argumentleri bilen işleýär.Aşakdaky tablisada bu funksiýalar getirilen.

abc (x)	x sanyň moduly.Netijede:|x|
divmod (x,y)	Paýyň bitin we galyndy bölekleri.Netijede: (bitin,galyndy).
pow (x,y[,m]	m modul boýunça x-i y derejä götermek. Netijede:x**y % m
round (n,[,z]	Sanlary nokatdan soň (ýa-da öň) berlen sifre çenli tegelemeli.
ord (s)	Funksiýa sanalýan görnüşli bahanyň tertip nomerini kesgitleýär.
chr (n)	n tertip nomeri boýunça simwoly kesgitleýär.
len (s)	Yzygiderligiň elementleriniň sanyny kesgitleýär.
oct(n), hex (n)	Sekiz belgili ýa-da on alty belgili setiriň tertip nomerini kesgitleýär.
cmp (x,y)	Iki bahany deňeşdirmek.Netijede:deňeşdirmäniň netijesine baglylykda otrisatel, nol ýa-da položitel.
unichr (n)	Unicode-setiriň simwolyny kody bilen kesgitleýär.
unicode (s, [, encoding [, errors]])	Berlen encoding kodirlemesinde s setire gabat gelýän Unicode-obýekti döredýär.Kodirlemäniň ýalňyşlyklary errors laýyklykda gaýtadan işlenýär we şeýle bahalary alyp biler: ‘ strict ‘ (berk özgerme), ‘replace’ (ýok simwollary çalyşmak) ýa-da ‘ignore’ (ýok faýllary inkär etmek).encoding=’utf-8’, errors=’strict:

Aşakdaky mysalda iňlis harplaryň Unicode kodirlemegiň mysaly getirilen:

```
print("Unicode tablisa (iňlis harplary)"center
(18*4))
i=0
for c m "ABCDEFJHIKLMNOPQRSTUVWXYZ"|
```

```
“abcdefjhiklmnopqrstuvwxyz”
u=unicode (c, ‘koi8-r’)
print (“% 3i: %1s %s” % (ord (u), c, ‘u’),
I + =1 ?
if i % 4 ==0:
print
```

Maglumatlary işläp taýýarlamagyň funksiýalary

Bu funksiýalar funksional programmirleme Sapaksynda giňişleýin gözegçilik ediler.enumerate ( ) funksiýasyna degişli mysal:

```
>>> for i, c in enumerate ("ABC"):
  print (i,c)

0 A
1 B
2 C
```

Häsiýetleri kesgitlemegiň funksiýalary

Bu funksiýalar obýektleriň käbir gurnalan atributlaryna we häsiýetlerine barmaga üpjün edýär.Aşakdaky mysal şol funksiýalaryň käbirlerini görkezýär:

```
>>> s = "abcde"
>>> s1="abcde"
>>> s2="ab"+"cde"
>>> print("hash:", hash(s), hash(s1), hash(s2))
hash: -1335406700 -1335406700 -1335406700
>>> print ("id:", id(s), id(s2))
id: 37550688 37550688
```

Bu ýerden görnüşi ýaly,şol bir “abcde” setir literal üçin şol bir obýekt alynýar,munuň tersine bolsa birmeňzeş manyly obýektler üçin dürli obýektleri alyp bolýar.

Içki gurluşlara barmak üçin funksiýalar

Python programmirleme dilinde global we lokal üýtgeýänleri sözlük görnüşinde globals ( ) we locals ( ) funksiýalaryň kömegi bilen alyp bolýar.Ýöne bu sözlüklere bir zatlary ýazmak maslahat berilmeýär.

vars ( ) funksiýa lokal atlary tablisa geçirýär (eger parametr berilmedik bolsa, ol locals ( ) funksiýanyň ýerine ýetirýän işini edýär).Adatça, formatirleme amalyny ýerine ýetirmek üçin sözlük hökmünde ulanylýar:

```
a=1
b=2
c=3
print ("%(a)s+%(b)s=%(c)s" % vars ())
```

```
>>>
1+2=3
>>>
```

Kompilýasynyň we ýerine ýetirmegiň funksiýalary

Bu kategoriýa degişli bolan reload ( ) funksiýasyny biz eýýäm seredip geçdik, indi eval ( ) funksiýasynyň üstünde durup geçeliň.Adyndan gelip çykyşy ýaly, bu funksiýa berilýän aňlatmany hasaplaýar.Muny aşakdaky mysalyň üsti bilen görüp bileris:

```
a = 2
b = 3
for op in "+-*/%":
    e = "a" + op + "b"
print (e, "->", eval(e))
>>>
a%b -> 2
>>>
```

Ýokardaky hasaplamadan daşary eval ( ) funksiýanyň ýene-de iki parametri bar-olaryň kömegi bilen atlaryň global we lokal atlary berip bolýar.Ýokarda ýazylan mysaly aşakdaky görnüşde hem ýazyp bolar:

```
for op in "+-*/%":
    e = "a" + op + "b"
print (e, "->", eval(e, {'a': 2, 'b': 3}))
```

Giriş- çykyş funksiýasy

Input ( ) we raw_input ( ) funksiýalary standart girişden girizmek üçin ulanylýar.open ( ) funksiýasy faýly okamak, ýazmak we üýtgetmek üçin açýar.Indiki mysalda faýl okamak üçin açylýar:

```
f=open (“file.txt”,”v”,1
for line in f:
…
f.close ( )
```

Funksiýa üç argumenti alýar:faýlyň ady (faýlyň ýerleşýän ýeri), açmak düzgün (“r” – okamak,”w” – ýazmak,”a” – goşmak ýa-da “wt”,”a+”, “r+” – üýtgetmek).Şeýle-de “t” goşulyp biliner,bu tekst faýly aňladýar.Üçünji argument bufer düzgünini görkezýär:0-bufer ýok,1-setir boýunça buferlemek,1-den uly – görkezilen ölçeglerdäki baytly bufer.Python diliniň soňky wersiýalarynda open ( ) file ( ) birmeňzeş funksiýalar.

Atributlar bilen işlemegiň funksiýalary

Python programmirleme dilinde obýektleriň atributlary bar (C++ dilinde funksiýa-agzalar we maglumat –agzalar).

```
# birinji programma:
class A:
pass
a = A()
27
a.attr = 1
try:
print a.attr
except:
print None
del a.attr
# ikinji programma:
class A:
pass
a = A()
setattr(a, 'attr', 1)
if hasattr(a, 'attr'):
print getattr(a, 'attr')
else:
print None
   delattr(a, 'attr')
```

Synplaryň usullarynyň "bezeg"-funksiýalary

Bu funksiýalara obýekte-gönükdirilen programmirlemede seredilip geçiler.

Standart kitaphana syn

Standart kitaphananyň modullaryny görnüşi boýunça şertli bölüp bolar:

1.Ýerine ýetirmek döwrüniň hyzmaty.Modullar:sys, atexit, copy, traceback, math, cmath, random, time, calendar, datetime, sets, array, struct, itertools, locale,gettext.

2.Işläp taýýarlamak siklini goldamak.Modullar:pdb, hotshot, profile, unittest, pydoc.Paketler:docutils, distutils.

3.Operasion ulgamy bilen işlemek (faýllar,prosesler).Modullar:os, os.path, getopt, glob, popen 2, shutil, select, signal, stat, tempfile.

4.Tekstler bilen işlemek.Modullar:string, re, StringIO, codecs, difflib, mmap, sgmllib, htmllib,htmlentitydefs.Paket:xml.

5.Köp ugurly hasaplamalar.Modullar:threading, thread, Queue.

6.Maglumatlary saklamak.Arhiwlemek.Modullar:picle,shelve,anydbm, gdbm, gzip,zlib, zipfile,bzz,csv,tarfile.

7.Platforma bagly modullar.UNIX üçin:commands,pwd,grp,fcntl, resource, termios, readline,rlcompleter.Windows üçin: msvcrt, _winreg, winsound.

8. Tor bilen işlemek. Internet protokollary. Modullar: cgi, Cookie, urllib, urlparse, httplib, smtlib, poplib, telnetlib, socket, asyncore. Serwerleriň mysaly: SocketServer, BaseHTTPServer, xmlzpclib, asynchat.

9. Internet bilen işlemek. Maglumatlaryň format. Modullar: quopri uu, base64, binhex, binascii, rfc822, mimetools, Mimewriter, multifile, mailbox. Paket: e_mail.

10. Python dilinde işlemek. Modullar: parser, symbol, token, keyword, inspect, tokenize, pyclbr, py_compile, compileall, dis, compiler.

11. Grafiki interfeýs. Modul: Tkinter.

Bellik: Köp ýagdaýlarda modullar bir ýa-da birnäçe synplary saklaýar, olaryň kömegi bilen gerekli görnüşiň obýekti döredilýär. Şonuň üçin bu ýerde modulyň atlary bilen iş salyşman, obýektiň atributlary bilen iş salyşylýar.

Ýerine ýetirmek döwrüniň hyzmatlary

sys modul

sys modul programma ýerine ýetirilişiň gurşawy, Python interpretatory baradaky maglumaty özünde saklaýar. Aşakda bu modulyň has köp ulanylýan obýektleri getirilen.

exit([c])	Programmadan çykmak. Programmany jemlemek üçin sanly kody berip bolýar: 0 – mesele ýüze çykmasa; beýleki sanlar – programmadan çykylanda mesele ýüze çyksa.
argv	Buýruk setiriniň argumentleriniň sanawy. Adatça sys.argv[0] işe göýberilen programmanyň adyny özünde saklaýar, galan parametrler bolsa buýruk setirinden berilýär.
platform	Interpretatoryň işleýän programmasy.
stdin, stdout, stderz	Standart giriş, çykyş ýalňyşlary çykarmak. Açyk faýl obýektleri.
version	Interpretatoryň wersiýasy
setrecursionlimit(limit)	Rekursiw çagyryşlaryň maksimum işleýiş derejesini gurnamak.
exc_info()	Düzedilýän ýalňyşlar baradaky maglumatlar.

copy moduly

Bu modul obýektleri göçürmek üçin funksiýalary öz içinde saklaýar. Aşakda bir mysaly getireliň:

```
lst1=[0,0,0]
lst=[lst1]*3
print(lst)
lst[0][1]=1
print(lst)
```

Netijede şeýle alynýar:

```
>>>
[[0, 0, 0], [0, 0, 0], [0, 0, 0]]
[[0, 1, 0], [0, 1, 0], [0, 1, 0]]
>>>
```

Ýokardaky sanawy köpeltmek üçin copy moduldaky copy() funksiýany ulanmak gerek:

```
from copy import copy
lst1=[0,0,0]
lst=[copy (lst1) for i in range (3)]
print(lst)
lst[0][1]=1
print(lst)
```

Netijede şeýle bolýar:

```
>>>
[[0, 0, 0], [0, 0, 0], [0, 0, 0]]
[[0, 1, 0], [0, 0, 0], [0, 0, 0]]
>>>
```

copy modulynda ýene-de düýpli göçürmek üçin deepcopy() funksiýasy bar, bu ýerde obýektler rekursiw göçürilýär.

math we cmath modullar

Bu modullarda hakyky we kompleks argumentler üçin matematiki funksiýalar getirilen. Bular ýaly funksiýalar C dilinde hem ulanylýar. Aşakdaky tablisada math modulyň funksiýalary getirilen. z harpy bilen belgilenen argumentler bolan funksiýalar cmath modulda hem kesgitlenen.

Funksiýa ýa-da hemişelik	**Düşündirilişi**
acos(z)	arkkosinus (z)
asin(z)	arksinus(z)
atan(z)	arktanges(z)
atan2(y,x)	atan(y/x)
ceil(x)	x-e deň ýa-da uly bolan iň kiçi bitin san
cos(z)	kosinus(z)
cosh(z)	giperbolik kosinus(z)
e	e-hemişelik
exp(z)	eksponenta, ýagny e**z
fabs(x)	x-iň absolýut bahasy
floor(x)	x-e deň ýa-da kiçi bolan iň uly bitin san

fmod(x,y)	x-i y-e bölünende galýan galyndy.
frexp(x)	mantissany we (m,i) jübüt hökmünde x tertibi berýär, bu ýerde m – ýüzýän nokatly hakyky san, I – bitin san bolup, x=m*2.**i. Eger 0 bolsa, onda (0,0) bolýar, ýogsa 0,5<=abs(m)<1.0
hypot(x,y)	sqrt(x*x+y*y)
ldexp(m,i)	m*(2**i)
log(z)	natural logarifm z
log10(z)	onluk logarifm z
modf(x)	x bitin we drob böleklerden ybarat bolan (y,q) jübüt alýar
pi	pi hemişelik
pow(x,y)	x**y
sin(z)	sinus z
sinh(z)	giperbolik sinus z
sqrt(z)	z-den alynýan kwadrat kök
tan(z)	tangens z
tanh(z)	giperbolik tangens z

random moduly

Bu modul dürli paýlaşdyrmalar üçin törän sanlary emele getirýär. Olardan iň köp ulanylýan funksiýalar:

random()	ýarym açyk diapazondan [0.0, 1.0) tötän sanlary emele getirýär.
choice(s)	s yzygiderlikden tötän elementi saýlaýar.
shuffle(s)	s üýtgeýän yzygiderligiň elementlerini ýerinde garyşdyrýar.
randrange([stat,]stop [,step])	range(start, stop, step) diapazondan tötän bitin sany berýär. Şuňa meňzeşlikde choice (range (start, stop, step).
normalvariate (mu, sigma)	tötan sanlardan kadaly ýerleşdirilen yzygiderlikden sany berýär. Bu ýerde mu – orta, sigma – orta kwadrat gyşarma (sigma >0)

time moduly

Bu modul wagty görkezmek we wagtyň formatlaryny özgertmek üçin funksiýany berýär.

set moduly

Bu modul köplükler üçin maglumatlaryň görnuşini doredýär. Indiki mysalda bu modul nähili ulanylýandygy görkezilen.

```
A=set([1,2,3])
B=set([2,3,4])
print(A|B, A&B, A-B, A^B)
for i in A:
    if i in B:
```

```
        print(i)
```
Netijede şeýle alynýar:
```
>>>
{1, 2, 3, 4} {2, 3} {1} {1, 4}
2
3
>>>
```

Array we struct modullary

Bu modullar kiçiderejeli massiwleri we maglumatlaryň gurluşyny döredýärler. Olaryň esasy wezipesi – maglumatlaryň ikilik formatyny derňemek.

Itertools moduly

Bu modul iteratorlar bilen işlemek üçin funksiýalaryň toplumyny öz içinde saklaýar.Iteratorlar maglumatlar bilen siklde işlenilişi ýaly yzygider işlemegi ýardam berýär.Iteratorlara funksional programmirleme Sapaksynda giňişleýin durulyp geçiler.

locale moduly

Bu modul pul birligi, wagty, senäni we ş.m. ýazmak üçin ulanylyp biliner.Aşakdaky programma muňa mysal bolup biler:

```
import time,locale
locale.setlocale (locale.LC_ALL,None )
print(time,strftime("%d%B%Y",time.localtime(time.time())))
```
Netijede alynar:
```
03 July 2014
```

gettext moduly

Python programma dilini dürli ýurtlarda ulanylanda programmanyň menýusyny degişli dile terjime etmek zerurlygy ýüze çykýar.gettext moduly bu meseläni ýeňilleşdirmäge ýardam berýär.Programmanyň esasy bölekleri iňlis dilinde ýazylýar.Programmada ýörite bellenen setirleriň terjimeleri her dil üçin aýratyn faýllaryň görnüşinde berilýär.

Taslama siklini goldamak

Bu bölümiň modullary resminamalary goldamaga, Python dilinde programmalary kämilleşdirmäge ýardam berýärler, şeýle-de taýyn programmalary ýaýratmakda hyzmat edýärler.

Mysal hökmünde şeýle matematik hasaplamany getireliň. “Eratosfeniň elegi” atly algoritm boýunça sada sanlary hasaplamak üçin käbir modul döredeliň.Bu modul sieve.py faýlynda ýerleşer we bir funksiýadan primes (N) ybarat bolar.Bu funksiýa öz işiniň netijesinde 2-den N-e çenli ähli sada sanlary (özüne we bire bölüniji natural sany bolmadyk) berýär:

```
import math
”””2-den N-e çenli sada sany hasaplamak üçin modul“““
def primes ( N )
“““ 2-den N-e çenli ähli sada sanlary çykarmaly “““
sieve =set ( range ( 2,N ))
for i in sieve:
sieve-= set (range (2*i,N,I))
return sieve
```

pdb moduly

Modul pdb buýruk setir-interfeýs bilen sazlaýjynyň funksiýasyny göz-öňüne getirýär.Bu modul ulanylan programmany aşakda mysal getireliň:

```
>>> import pdb
>>> pdb.runcall(Sieve.primes, 100)
> /home/rnd/workup/intuit-
python/examples/Sieve.py(15)primes()
-> sieve = sets.Set(range(2, N))
(Pdb) l
10 import sets
11 import math
12 """2-denN-eçenli sada sany hasap-k üçin modul"""
13 def primes(N):
14 """2-denN-e çenlisadasanlary çykarýar"""
15 -> sieve = sets.Set(range(2, N))
16 for i in range(2, int(math.sqrt(N))):
17 if i in sieve:
32
18 sieve -= sets.Set(range(2*i, N, i))
19 return sieve
20
(Pdb) n
> /home/rnd/workup/intuit-
python/examples/Sieve.py(16)primes()
-> for i in range(2, int(math.sqrt(N))):
(Pdb) n
```

```
> /home/rnd/workup/intuit-
python/examples/Sieve.py(17)primes()
-> if i in sieve:
(Pdb) n
> /home/rnd/workup/intuit-
python/examples/Sieve.py(18)primes()
-> sieve -= sets.Set(range(2*i, N, i))
(Pdb) n
> /home/rnd/workup/intuit-
python/examples/Sieve.py(16)primes()
-> for i in range(2, int(math.sqrt(N))):
(Pdb) p sieve
Set([2, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27,
29, 31, 33, 35, 37, 39,41, 43, 45, 47, 49, 51, 53, 55,
57, 59, 61, 63, 65, 67, 69, 71, 73, 75, 77, 79,81, 83,
85, 87, 89, 91, 93, 95, 97, 99])
(Pdb) n
> /home/rnd/workup/intuit-
python/examples/Sieve.py(17)primes()
-> if i in sieve:
(Pdb) n
> /home/rnd/workup/intuit-
python/examples/Sieve.py(18)primes()
-> sieve -= sets.Set(range(2*i, N, i))
(Pdb) n
> /home/rnd/workup/intuit-
python/examples/Sieve.py(16)primes()
-> for i in range(2, int(math.sqrt(N))):
(Pdb) p sieve
Set([2, 3, 5, 7, 11, 13, 17, 19, 23, 25, 29, 31, 35,37,
41, 43, 47, 49,53, 55, 59, 61, 65, 67, 71, 73, 77, 79,
83, 85, 89, 91, 95, 97])
```

profile moduly

Bu modulyň kömegi bilen programma üpjünçiliginde dürli funksiýalaryň we usullaryň ýerine ýetirilmeginde naçe wagt gerekdigini bilip bolýar.

Ýokardaky mysaly dowam edip,primes ( ) funksiýa ulanylanda näçe wagt sarp edilýändigini görüp bolar:

```
>>> import profile
>>> profile.run("Sieve.primes(100000)")
709 function calls in 1.320 CPU seconds
Ordered by: standard name
```

```
ncalls tottime percall cumtime percall
filename:lineno(function)
   1 0.010 0.010 1.320 1.320 <string>:1(?)
   1 0.140 0.140 1.310 1.310 Sieve.py:13(primes)
   1 0.000 0.000 1.320 1.320
profile:0(Sieve.primes(100000))
   0 0.000 0.000          profile:0(profiler)
65 0.000 0.000     0.000    0.000 sets.py:119(__iter__)
314 0.000 0.000 0.000 0.000 sets.py:292(__contains__)
65 0.000 0.000     0.000    0.000
sets.py:339(_binary_sanity_check)
66 0.630 0.010 0.630 0.010 sets.py:356(_update)
660.000 0.000 0.630 0.010 sets.py:425(__init__)
65 0.010 0.000 0.540 0.008 sets.py:489(__isub__)
65 0.530 0.008 0.530 0.008
sets.py:495(difference_update)
```

Bu ýerde ncalls –funksiýany ýa-da usuly çagyrmagyň sany,tottime–funksiýanyň kodynyň ýerine ýetirilmeginiň doly wagty (çagyrylýan funksiýalardan gözlemegiň wagtyny hasaba almazdan), percall – şol bir zat, ýöne bir çagyryşdan artyk hasaba alnan, Cumtime – funksiýada hemme çagyrylýan funksiýalar bilen birlikde bar bolmagyň umumy wagty.Soňky sütünde faýlyň ady, funksiýaly ýa-da usully setiriň nomeri we ady getirilen.

Bellik:

Bu ýerde _iter_,_contains_ we _isub_-elementler boýunça inerasiýany,elemente degişliligini (in) barlamagy we -= amaly ýerine ýetirýän usullaryň atlary.

init_ usuly –obýektiň konstruktory.

unittest moduly

Programma üpjünçiligi taýýarlananda yza gaýtma (regressiýa) atly barlaglary ulanmak maslahat berilýär.Her bir modul üçin testleriň ýygyndysy düzülýär.Onuň kömegi bilen diňe bir adaty hasaplamalar barlanylan,eýsem algoritmiň her bir şahasyny barlanylmaly.Bu modul üçin düzülen test test_sieve.py faýlynda ýerleşýär:

```
# file: test_Sieve.py
import Sieve, sets
import unittest
class TestSieve(unittest.TestCase):
def setUp(self):
pass
```

```
def testone(self):
primes = Sieve.primes(1)
self.assertEqual(primes, sets.Set())
def test100(self):
primes = Sieve.primes(100)
self.assert_(primes==sets.Set([2,3,5,7,11,13,17,19,23,
29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97]))
if __name__ == '__main__':
  unittest.main()
```

Barlag moduly kesgitli synpdan ybarat we unittest.TestCase synpyna degişli.Bu ýerde synaglara bolan taýýarlyk (setUp usuly) we synaglaryň özi, test başlaýan – – usullar suratlandyrylýar.

Barlaglaryň işi unittest.main () funksiýanyň ýerine ýetirilmegi bilen başlaýar.Barlag gowy geçse, şeýle görnüşde bolýar:

```
$ python test_Sieve.py
..
----------------------------------------------------------
Ran 2 tests in 0.002s
  OK
```

Programma taýýarlananda her goýberilişden öň ähli modullary regressiw barlaglardan geçirilýär.Bu barlaglar ýalňyşlaryň öňüni almak üçin niýetlenen.Ýöne hiç bir test çylşyrymly programmanyň ýalňyşsyzlygyny kepillendirip bilmeýär.Modullaryň üsti doldurylanda test-barlaglarda hem üýtgeşmeler bolup biler.

Python programmirleme dilinde we onuň standart kitaphanasynda her modul üçin test-barlaglary bar we test katologynda ýerleşýär.

pydoc moduly

Programmanyň üstünligi diňe bir täsirli we ýokary hilli kod bilen üpjün etmeklige bagly bolman, eýsem resminamanyň hiline hem bagly.pydoc utilitasy Unix-däki man komandasyna meňzeş:sah.34.

```
$ pydoc Sieve
Help on module Sieve:
NAME
Sieve-2-den N-e çenli sada sanlary hasaplamak üçin modul
FILE
Sieve.py
FUNCTIONS
```

```
primes(N)
2-den N-e çenli ähli sanlary berýär
```

distutils paketi

Bu paket Python diliniň hususy paketlerini ýaýratmak üçin mümkinçilik berýär.distutils ulanýan setup.py kiçiräk konfigurasion faýly we python setup.py install buýrugy bilen ulanyjylar gurnar ýaly MANIFEST.in proýektiň faýllaryny sanaşdyrýan faýly ýazmak ýeterlik.

Operasion ulgam bilen özara täsir

Katalogy bölüjiler we onuň bilen bagly beýleki belgiler hemişelik görnüşinde elýeterli.

Hemişelik	Aňlatmasy
os.curdir	Işjeň katalog
os.pardir	Başlangyç katolog
os.sep	Elementleriň bölünijisi
os. altsep	Elementleriň beýleki bölünijisi
os. pathsep	Salgylaryň sanawynda salgylaryň bölünijisi
os. defpath	Salgylaryň sanawy
os.linesep	Setiriň ahyrlandygynyň alamaty

Python dilindäki programma operasion ulgamda aýratyn proses görnüşde işleýär. os modulyň funksiýalary prosese we onuň işleýän gurşawyna degişli dürli ululyklary elýeterli edýär. os modulyň içinde elýeterli möhüm obýektleriň biri üýtgeýän ululyklaryň sözlügi environ bolup durýar. Muny aşakdaky mysaldan görüp bileris:

```
import os
PATH = os.environ['PATH']
```

Funksiýalaryň köp bölegi fayllar we kataloglar bilen işlemäge niýetlenen. Aşakdaky tablisa Unix-de we Windows-da elýeterliler getirilen.

access(path, flag)	path atly faýlyň ýa-da katalogyň elýeterliligini barlamak. Soralýan salgynyň düzgüni flags bahalary bilen görkezilen. Ol bolsa şeyle

	toplumlardan ybarat: os.F_OK (faýl bar), os.R_OK (faýldanokap bolýar), os.W_OK (faýla ýazyp bolýar) we os.X_OK (faýly işledip, katalogy görüp bolýar).
chdir(path)	path işjeň katalog edyär.
getcwd()	işçi katalog
chmod (path , mode)	mode aňlatmada path faýla bolan salgyny gurnaýar. chmode funksiýa öň bar bolan düzgüniň üstüni doldurman, täzeden gurnaýar.
listdir (dir)	dir katalogdaky faýllaryň sanawyny düzyär. Bu sanawa "." we ".." ýörite belgiler girmeýär.
mkdir (path, [mode])	path katalogy döredýär. Kada laýyklykda mode düzgüni 0777 deň, ýagny: S_IRWXUIS_IRWXG| S-IRWXO.
makedirs (path [,mode])	mkdir( ) funksiýa meňzeş bolup, eger ýok bolsa, ähli gerekli kataloglary döredýär.Iň soňky katalog bar bolsa, ýalňyş çykarýar.
remove (path) ,unlink (path)	path faýly ýok edýär. Kataloglary ýok etmek üçin rmdir ( ) we removedirsl ( ) ulanylýar.
rmdir (path)	path boş katalogy ýok edýär.
removedirs (path)	path boş kataloglary ýok edýär. Eger bu prossesiň dowamynda soňky katalog boş bolmasa, OSError duýduryş çykýar.
rename (src, dst)	src atly faýlyň ýa-da katalogyň adyny dst ada çalyşýar.
renames (src, dst)	rename ( )funksiýa meňzeş bolup, dst salgy üçin ähli gerekli kataloglary döredýär we src salgydaky boş katalogdaky ýok edýär.
utime (path, times)	Faýla girizilen soňky üýtgeşmeleriň wagtyny (mtime) we onuň ýoluny (atime)görkezýär. Eger times None deň bolsa, wagt hökmünde işlenen wagt ulanylýar.

Prosesler bilen işlemek üçin os modul şeýle funksiýalary hödürleýär:

abort ( )	Bolup geçýän proses üçin SIGABRT duýduryşy çagyrýar.
system (cmd)	cmd buýruk setirini ýerine ýetirýär.
times ( )	Prosesiň işleýiş wagtyny sekuntlarda görkezýär.
getloadng ( )	Prosesiň işliliginiň soňky 1,5 we15 minudyň dowamyndaky bahasyny görkezýär.

stat moduly

Bu modulda os.stat( ) we os.chmod ( ) funksiýalarda ulanylýan hemişelikler görkezilen.

tempfile moduly

Programma ýazylanda kähalatlarda wagtlaýyn faýl döretmek zerur bolýar, bu faýl käbir hereketler ýerine yetirilenden soň gerek däl. Bu maksatlar üçin TemporaryFile funksiýa ulanylýar.

Aşakdaky mysalda wagtlaýyn faýl döredilýär, bu faýla maglumat girizilýär, soňra ondan okalýar:

```
import tempfile
f = tempfile.TemporaryFile()
f.write("0"*100) # 0 simwolyňýüzüsi ýazylýar
f.seek(0)        # faýlyň başygörkezilýär
print        len(f.read())        #faýlyň        soňuna
çenliokalýarweuzynlygykesgitlenýär
```

Maglumatlary saklamak. Arhiwirlemek.

Bu topara maglumatlaryň daşky saklanýan yerleri bilen işleýän modullar degişli.

pickle moduly

Obýekti baýtlaryň yzygiderliligi görnüşde ýazmak prosesine seriýalylyk diýilýär. Obyekti daşky huşda saklamak üçin ony ilki seriýalaşdyrmaly.

pickle moduly obýektleri seriýalaşdyrmaga we olary setirde ýa-da faýlda ýatda saklamaga ýardam berýär. Aşakdaky obýektler seriýalaşdyrylyp biliner:

- Gurnalan görnüşler: None, sanlar, setirler(adaty we Unicode)
- Diňe seriýalaşdyrylýan obýektleri özünde saklaýan sanlar we sözler.
- Modul derejesinde kesgitlenen funksiýalar.
- Gurnalan funksiýalar.
- Modul derejesinde kesgitlenen synplar.
- _dict_ ýa-da _setstate_ synplaryň obýektleri.

Bu modulyň ulanylýan mysaly aşakda getirilen:

Ýatda saklamak

```
import pickle, time
mydata = ("abc", 12, [1, 2, 3])
output_file = open("mydata.dat", "w")
p = pickle.Pickler(output_file)
p.dump(mydata)
```

```
output_file.close()
```

Dikeltmek:

```
import pickle
input_file = open("mydata.dat", "r")
mydata = pickle.load(input_file)
print mydata
input_file.close()
```

shelve moduly

Obýektleri Python formatynda saklamak üçin shelve modly ulanylýar. Özüniň gurluşy boýunça ol sözlükden kän tapawutlanmaýar. Muňa aşakdaky mysaldan görüp bolar:

```
import shelve
data = ("abc", 12)
key = "key" # - açar (setir)
filename = "polka.dat"
d = shelve.open(filename)
d[key] = data # key açar bilen maglumatlary saklamak
data = d[key] # açar boýunça bahasyny çykarmak
len(d) # obýektleriň gerekli sanyny almak
d.sync()
del d[key] # açary we bahasyny ýok etmek
flag = d.has_key(key) # açaryň barlygyny barlamak
lst = d.keys() # açarlaryň sanawy
d.close()
```

Anydbm we gdbm modullar

Maglumatlary daşynda saklamak üçin açar-baha jübütini özünde saklaýan ýönekeý maglumatlar binýadyny ulanyp bolar. Pythondilinde şeýle binýatlar bilen işlemek üçin birnäçe modullar bar: bsddb, gdbm, dbhash we bashg. anydbm bar bolan heşleriň birini saýlaýar, şonuň üçin ony dürli formatlary okamak üçin ulanyp bolar (any-islendik).

Python dilinder heşe barmak sözlüge barmakdan kän tapawutlanýar. Sözlükden tapawutlylykda faýly döretmek, okamak we ýazmak üçin heşi açmaly we soňundan ýapmaly. Mundan başga-da maglumatlar ýazylanda üýtgemezligi üçin heş blokirlenýär.

csv moduly

csv formaty (comma separatet values – otur bilen bölünen baha) elektron tablisalar bilen maglumatlar binýadynyň arasynda maglumatlary alyş-çalyş etmek üçin ulanylýar. Indiki mysalda CSV-faýla ýazmak we ondan okamak görkezilen:

```
mydata = [(1, 2, 3), (1, 3, 4)]
import csv
# Faýla ýazmak:
f = file("my.csv", "w")
writer = csv.writer(f)
for row in mydata:
writer.writerow(row)
f.close()
# Faýldan okamak:
reader = csv.reader(file("my.csv"))
for row in reader:
print row
```

Platforma bagly modullar

Bu modullar takyk operasion ulgamda işlemek üçin niýetlenen. Olar esasan üç platforma degişli: Unix, Windows we Macintosh.

Tory goldamak. Internet protokollar

Bu topara degişli ähli modullar şol bir düzgün boýunça işleýär: moduldan sewer bilen baglanyşygy baradaky maglumaty özünde saklaýan obýekti bolan synp gerek, onuň usullary bolsa degişli protokol boýunça serwer bilen özara täsiri amala aşyrýar. Şeýlelikde, protokol näçe çylşyrymly bolsa, işiň amala aşyrylmagy üçin şonçada usullar gerek. Bu barada aýratyn Sapak berlen.

Internet goldamak. Maglumatlaryň formaty

Python diliniň standart kitaphanasynda dürli formatlar bilen işlemek üçin dürli derejeli modullar bar, olar Internet torunda maglumatlary kodirlemek üçin ulanylýar.

Häzirki wagtda RFC282 formatyndaky maglumatlary gaýtadan işlemek üçin iň gowy serişde E_mail paketi bolup durýar. Onuň kömegi bilen maglumatlary amatly görnüşde gaýtadan işläp we formatirläp bolýar.

Gtafiki interfeýs

Häzirki zaman goşundylaryň ählisinde diýen ýaly grafiki interfeýs bar. Şeýle goşundylary Python dilinde hem döredip bolýar. Python programmirleme dilinde Tkinter moduly bar, ol Tcl/Tkdiline bolan interfeýs, bu dilde bolsa grafiki interfeýsi ýazyp bolýar.

Grafiki interfeýsi programmirlemek üçin başgada paketler bar: wxPython (wx Windowsa esaslanan), PyGTK we başg. Bu paketleriň içinde diňe bir platformada işleýänler hem bar.

Netije

Bu Sapakda Python diliniň gurnalan funksiýalary we onuň standart kitaphanasynyň modullary barada gürrüň edildi. Käbir ugurlara indiki Sapaklarda has giňişleýin seredip geçiler. Python programmirleme diliniň şeýle bir uly

standart kitaphanasy bar welin, bir Sapaknyň dowamynda ony kiçiräk mysallar bilen ýüzleý geçip bolar.

3. FUNKSIONAL PROGRAMMIRLEMÄNIŇ ELEMENTLERI

Funksional programmirleme näme?

Funksional programmirleme–bu diňe funksiýalary ulanýan programmirleme stil. Başga sözler bilen aýdylanda bu imperatiw buýruklardaky däl-de aňlatmalardaky programmirleme.

Dewid Mertz (David Mertz)Python dilindäki fuksional programmirleme baradaky makalasynda şeýle belleýär: "funksional programmirleme–funksional dillerdäki (LISP, ML, OCAML, Haskell,...) programmirleme", onuň esasy atributlary:

- "Yza çekiliş programmanyň esasy dolandyryjy gurluşy bolup durýar.
- Sanawlaryň (yzygiderlikleriň) üstünde işlemek.
- Funksiýalaryň zyýanly täsiri bolmaly däl.
- Operatorlar bolmaly däl, olaryň ýerine aňlatmalar bolmaly.
- Ýokary tertipli funksiýalary ulanmaly."

Funksional programma

Matematikada funksiýa obýektleri bir köplükden (funksiýany kesgitlemegiň köplügi) beýleki köplüge (funksiýanyň bahalarynyň funksiýasy) geçişini görkezýär. Matematiki funksiýalar (olara "arassa" diýýärler) berlen argumentler boýunça netijeleri hasaplaýarlar.

Funksional stilde programmalar funksiýalaryň toplumy görnüşinde guralýar. Üstesine funksiýa edil matematikada ýaly ýerine ýetirilýär: bir obýekti beýleki bir obýekte geçirýär. Programmirlemede "arassa" funksiýalar–ideal, tejribede bular ýaly funksiýalary gazanyp bolanok. Adatça funksiýalaryň goşmaça täsiri bar: çagyryşlaryň arasyndaky ýuu7agdaýlary saklaýarlar ýa-da beýleki obýektleriň ýagdaýyny üýtgedýärler. Umuman aýdylanda, programmalar ýakynlaşan hasaplamalary berýär.

Aňlatmalara ýazylýan "+", "-", "*", "/" biner amallar iki funksiýalar. Olar şeýle köp ulanylýany üçin, programmirleme diliniň sintaksinde olar üçin gysga ýazylyş düzgüni bar. Operator moduly bu amallary funksional görnüşde ýazmaga ýardam berýär:

```
    >>> form operator import add, mul
    >>> print (add (2, mul (3,4)))
14
```

Kesgitlemek we çagyrmak funksiýasy

Öň belläp geçişimiz ýaly, Python dilinde funksiýany iki usul bilen kesgitläp bolýar: def operatoryň we lambda-aňlatmanyň kömegi bilen. Birinji usul operatory ulanmaga ýardam berýär. Ikinji usulda–funksiýanyň kesgitlenmesi diňe aňlatma bolup biler.

Funksiýany kesgitlemegiň sintaksisini mysallaryň üsti bilen seretsek gowy bolar. Degişli funksiýany kesgitlemekden soň ony çagyrmagyň birnäçe görnüşleri görkezilýär.

Funksiýanyň kesgitlenilişinde resmi parametrlerin sanawy we funksiýanyň kesgitlenilişiniň göwresi bolmaly. Resmi parametrler funksiýanyň kesgitleniş göwresindäki lokal atlar bolup duryarlar. Funksiýa işe girizilende olar obýektler bilen bagly bolýarlar.

Funksiýa işe girizilende sintaksiki şeýle görnüşde bolýar: obýekt-funksiýa. Adatça obýekt-funksiýa-bu funksiýanyň ady, ýöne hasaplamalaryň netijesinde ýerine ýetirilýän obýekti berýän islendik aňlatma hem bolup biler.

Bir argumentiň funksiýasy:

```
def swapcase(s):
    return s.swapcase()
print swapcase("ABC")
```

Iki argumentiň funksiýasy:

```
def inc(n, delta=1):
        return n+delta
print inc(12)
print inc(12, 2)
```

Bir hokmany argumentli, öň bahasy bar bolan we atlandyrylan argumentleriň kesgitsiz sany bolan argumentli funksiýa:

```
def warp (text, width=70, **kwargs):
    form texwarp import Text Wrapper
    # kwargs-atlary we bahalary bolan argumentli sözlük
    w=Text Wrapper (width=width, **kwargs)
    return. w. warp (text)
print (warp ("my long text…", width=4)
```

Erkin sanly argumentleriň funksiýasy:

```
    def max_min (* args):
    # args-görkezilen tertipdäki argumentleriň sanawy
    return max (args), min (args)
print (max_min (1,2,-1,5,3)
```

Adaty we atlandyrylan argumentli funksiýa:

```
def swiss_knife(arg1, *args, **kwargs):
        print arg1
print args
```

```
print kwargs
return None

print swiss_knife(1)
print swiss_knife(1, 2, 3, 4, 5)
print swiss_knife(1, 2, 3, a='abc', b='sdf')

lst = [2, 3, 4, 5]
dct = {'a': 'abc', 'b': 'sdf'}
print swiss_knife(1, *lst, **dct)
```

lambda-aňlatmanyň kömegi bilen funksiýany kesgitlemegiň mysaly aşakda berlen:

```
    func=lambda x, y:x+y
```

lambda-aňlatmanyň netijesinde atsyz obýekt-funksiýa alynýar, soňra bolsa oňa käbir ady baglaşdyrmak üçin ulanylýar. Düzgün boýunça lambada-aňlatma bilen kesgitlenýän funksiýalar funksiýanyň parametrleri hökmünde ulanylýar.

Python dilinde funksiýa diňe bir bahany çykaryp bilýär. Indiki mysalda divmod() standart funksuýanyň iki sanyň bölünende netijesini we galyndysyny çykarýar:

```
def bin(n):
    """Natural sanlaryň sifrlerindäki görnüşi"""
digits=[]
while n>0:
    n,d=divmod(n,2)
    digits=[d]+digits
retrun digits
print (bin (69))
```

Bellik:
Funksiýanyň adynyň yzynda obýekt duryandygyny bellemek gerek. Bu obýekti başga at bilen hem bglaşdyryp bolar:

```
def add (x,y)
return x+y
addition=add #indi addition we add bir obýektiň dürli atlary.
```

Aşakda ýene bir mysala seredip geçeliň:

```
def mylist (val, lst=[]):
    lst.append (val)
    return lst
```

```
print (mylist (1))
print (mylist (2))
```

Rekursiýa

Käbir ýagdaýlarda funksiýany beýan etmek üçin şol funksiýany çagyrmaly bolýar. Funksiýa öz-özüni çagyrýan ýagdaýda rekursiýa diýilýär. Funksional programmirlemede rekursiýa iterasiýadan (aýlawlar) has köp ulanylýar.

Indiki mysalda bin() funksiýasy rekursiw görnüşde getirilen:

```
def bin(n):
    """Natural sanyň iki belgili sifrleri"""
    if n == 0:
        return []
    n, d = divmod(n, 2)
    return bin(n) + [d]
print bin(69)
```

Bu ýerden görnüşi ýaly, while aýlaw sikli ulanylmaýar, onuň ýerine rekursiýanyň soňlanandygy baradaky şert döreýär: şert ýerine ýetse, funksiýa öz-özüni çagyrmaýar.

Owadan rekursiw çözüwleri gazanjak bolup, meselaniň täsirliligini gözden düşürmeli däl. Muny Fibonaççiniň n-nji sanyny hasaplamak üçin ulanylýan funksiýa görkezýär:

```
    def Fib (n)
    if n<2:
    return n
    else:
    return Fib (n-1)+Fib(n-2)
print(Fib(n))
```

Bu ýagdaýda rekursiw çagyryşlaryň sany n sandan eksponensial artýar, bu bolsa işlenilýän meseläniň wagtlaýyn çylşyrymlylygyna gabat gelenok.

Bellik:

Rekursiw funksiýalar bilen işlenende Python rekursiýada mümkin bolan çuňlugunu gazanyp bolýar. Rekursiýanyň çuňlugyny sazlamak üçin talap edilýän N bahany gurnap, sys moduldan setrecursionlimit (N) funksiýany ulanmak gerek.

Funksiýa parametr we netiije hökmünde

Python programmirleme dilinde funksiýa edil sanlar, setirler ýa-da yzygiderlikler ýaly obýekt bolup durýar.

Diýmek, funksiýalary funksiýalaryň parametrleri hökmünde geçirip bolýar.

Argumentleri kabul edýän ýa-da netijede beýleki funksiýalary ýokary tertipli funksiýalar diýilýär.

Python dilinde programmistler ýokary tertipli funksiýalary köp ulanýarlar. Olary ulanmak bilen köp ýagdaýlarda ters çagyryşlaryň mehanizmleri (callback) gurulýar, ýöne başga görnüşler hem bar. Mysal üçin, gözleg algoritmi berlen funksiýany her tapylan obýekt üçin çagyryp biler.

apply () funksiýa

apply () funksiýa birinji argument hökmünde berlen funksiýany ikinji we üçünji argumentler tarapyndan berlen parametrlere ulanýar. Bu funksiýa Python dilinde köneldi, şonuň üçin onuň ýerine parametrleri ýyldyzjyklaryň kömegi bilen berip bolar:

```
>>> lst=[1,2,3]
>>> dct={'a':4, 'b':5}
>>> max (*lst)
3
>>> dct (**dct)
{'a':4, 'b':5}
```

Yzygiderlikleriň üstünde işlemek

Köp algoritmler massiwleriň üstünde işlemek üçin we netijede täze massiwleri almak üçin ulanylýar. Python dilinde gurnalan funksiýalaryň içinde yzygiderlikler bilen işlemek üçin niýetlenenler hem bar.

Yzygiderlik diýlip Python dilinde yzygiderligiň interfeýsini goldaýan maglumatlaryň islendik görnüşine düşünilýär.

range() funksiýasy

range() funksiýasy birden üçe çenli argumenti alyp bilýär. Eger bir argument bolsa, ol 0-dan berlen sana çenli sanawy işe girizýär. Eger iki argument bolsa, onda sanaw birinji argumentiň görkezen sanyndan başlaýar. Eger üç sany argument bolsa – üçünji argument ädimi görkezýär.

```
>>>print (range(10))
range(10)
>>>print (range(1,10))
range(1,10)
>>>print (range(1,10,3))
range(1,10,3)
```

map() funksiýa

Käbir funksiýany yzygiderligiň ähli elementlerine ulanmak üçin map(f,*args) funksiýa ulanylýar. Bu funksiýanyň birinji parametri – yzygiderligiň ähli elementlerine ulanyljak funksiýa. Her n+1-nji parametr yzygiderlik bolmaly, sebäbi onuň her bir elementi f( ) funksiýa çagyrylanda n-nji parametr hökmünde ulanylar. Munuň netijesi bu funksiýanyň ýerine ýetirilmeginde düzülen sanaw bolýar.

Indiki mysalda iki sanawlaryň bahalary goşulýar:

```
l1=[2,7,5,3]
l2=[-2,1,0,4]
for i in map(lambda x,y: x+y, l1,l2):
    print(i)
>>>
0
8
5
7
>>>
```

Bu mysalda l1 we l2 yzygiderlikleriň ähli elementleriniň jemini almak üçin atsyz funksiýa ulanylýar. Eger yzygiderlikleriň biri beýlekisinden gysga bolsa, degişlik operandyň ýerine None bolar, bu bolsa goşmak amalyny bozar. Işlenilýän meselä baglylykda ýa funksiýany üýtgetmeli, ýa-da uzynlyklary boýunça dürli yzygiderlikleri ýalňyş hasaplap, algoritmiň şahasy hökmünde onuň üstünde işlemeli.

map( ) funksiýanyň ýene bir mysaly – birinji argument hökmünde None ulanylýar.

```
>>> l1 = [2, 7, 5, 3]
>>> l2 = [-2, 1, 0, 4]
>>> print map(None, l1, l2)
[(2, -2), (7, 1), (5, 0), (3, 4)]
```

filter( ) funksiýa.

Ýene bir köp duş gelýän amallaryň biri başlangyç yzygiderligi käbir şerte laýyklykda süzmek bolup durýar. filter(f,seq) funksiýa iki argumenti alýar: şertli funksiýa we yzygiderlik. Netijeleýji yzygiderlige diňe f( ) funksiýanyň çykaran bahalary düşer.

Eger-de f-iň ýerine None berilse, netijeleýji yzygiderlik başlangyç yzygiderligiň True bahalaryny alan elementlerden durar.

Mysal üçin, aşakda harplardan ybarat bolmadyk simwollaryň ýok edilişi görkesilen:

```
>>>filter(lambdax: x.isalpha(),'Salam!Meniň
                              ýagdaýlarym gowy!')
'SalamMeniňýagdaýlarymgowy'
```

Sanawa girizmeler

Bu goşulan `for` aýlaw sikller we `if` şertler üçin ýörite gysgaldylan sintaksis, onuň kömegi kesgitli aňlatma sanawa goşulýar, mysal üçin:

```
all_pairs=[ ]
for i in range(5):
```

```
        for j in range(5):
            if i<=j:
                all_pairs.append((i,j))
                print(all_pairs)
>>>
[(0, 0)]
[(0, 0), (0, 1)]
[(0, 0), (0, 1), (0, 2)]
[(0, 0), (0, 1), (0, 2), (0, 3)]
[(0, 0), (0, 1), (0, 2), (0, 3), (0, 4)]
[(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 1)]
[(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 1), (1, 2)]
[(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 1), (1, 2), (1, 3)]
[(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 1), (1, 2), (1, 3), (1, 4)]
[(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 1), (1, 2), (1, 3), (1, 4), (2, 2)]
[(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 1), (1,2), (1,3), (1,4), (2,2), (2,3)]
[(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 1), (1, 2), (1, 3), (1, 4), (2, 2), (2,
3), (2, 4)]
[(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 1), (1, 2), (1, 3), (1, 4), (2, 2), (2,
3), (2, 4), (3, 3)]
[(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 1), (1, 2), (1, 3), (1, 4), (2, 2), (2,
3), (2, 4), (3, 3), (3, 4)]
[(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 1), (1, 2), (1, 3), (1, 4), (2, 2), (2,
3), (2, 4), (3, 3), (3, 4), (4, 4)]
>>>
```

Bu mysaly sanawa girizmeler görnüşinde şeýle ýazyp bolar:

```
all_pairs=[(i,j) for i in range(5) for j in range(5)if
i<=j]
print(all_pairs)
>>>
[(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 1), (1,
2), (1, 3), (1, 4), (2, 2), (2, 3), (2, 4), (3, 3), (3,
4), (4, 4)]
>>>
```

Görnüşi ýaly, sanawa girizmeler map( ) we filter( ) funksiýalaryny okamak üçin has ýeňil gurluşa çalyşmaga ýardam berýär.
Indiki tablisada birmeňzeş aňlatmalaryň dürli görnüşleri getirilen:

Funksiýa görnüşde	**Sanawa girizme görnüşde**
filter (f,lst)	[x for x in lst if f(x)]
filter (None, lst)	[x for x in lst if x]
map (f, lst)	[f(x) for x in lst]

sum ( ) funksiýa

Elementleriň jemini sum( ) funksiýanyň kömegi bilen alyp bolar:

```
>>> sum(range(10))
45
```

Bu funksiýa diňe san görnüşler üçin işleýär, ony setirler üçin ulanyp bolanok. Setirler üçin join( ) funksiýany ulanmaly.

reduce( ) funksiýa

Zynjyrly hasaplamalary gurmak üçin reduce( ) funksiýa ulanylýar. Ol üç argumenti alýar: iki argumentleriň funksiýasy, yzygiderlik we başlangyç baha. Bu funksiýanyň kömegi bilen sum( ) funksiýany ýerine ýetirip bolýar:

```
def sum (lst, start):
   return reduce (lambda x,y: x+y, lst, start)
```

Indiki mysalda jemiň aralyk netijeleri toplanýar:

```
lst = range (10)
f = lambda x,y: (x[0]+y, x[1] + [x[0]+y])
print (reduce (f, lst, (0, [ ])))
```

Netijede alynar:

```
(45, [0,1,3,6,10,15,21,28,36,45])
```

zip ( ) funksiýa

Bu funksiýa toplumlaryň sanawyny çykarýar, onda i-nji toplum argumentler – yzygiderlikleriň i-nji elementini saklaýar. Netijeleýji yzygiderligiň uzynlygy yzygiderlikler argumentleriň iň gysgasyna deň:

```
>>> print (zip(range(5), "abcde")
[(0, 'a'), (1, 'b'), (2, 'c'), (3, 'd'), (4, 'e')]
```

Iteratorlar.

Maglumatlaryň üstünde işlemek üçin yzygiderlikleri ulanmak hemişe täsirli bolanok, sebäbi wagtlaýyn maglumatlary saklamak üçin köp operatiw ýat gerek. Şonuň üçin has täsirli çözüw diýip iteratorlary hasaplap bolar. Iteratorlar – ýörite obýektler bolup, maglumatlary yzygider hasaplamaga mümkinçilik bolýar. Eger aňlatmada iteratorly amallar bar bolsa, aralyk hasaplamalar köp ýer tutmaz, sebäbi hasaplamak üçin zerurlyk ýüze çykanda maglumatlar alynýar. Iteratorlar ulanylyp, maglumatlar işlenende ýat diňe başlangyç maglumatlar we netijeler üçin gerek bolýar, sebäbi maglumatlar faýla ýazylanda we okalanda diskden ýerine ýetirilýär.

Iteratorlary `for` operatorynda yzygiderligiň ýerine ulanylyp biliner. Faýl görnüşli obýektler hem iteratorlar, bu bolsa huşda kän ýer tutman uly faýllaryň üstünde işlemäge ýardam berýär.

Iterator talap edilýän ýerde yzygiderligi hem ulanyp bolar.

Iteratory bir-birden hasaplap hem bolýar. Iteratoryň interfeýsini goldaýan islendik obýekt next( ) usula eýe. Bu usul ulanylanda her çagyryşda iteratoryň indiki bahasy çykarylýar. Eger başga baha galmasa StopIteration duýduryş çykýar.

Käbir obýekt boýunça iteratory almak üçin bu obýekte iter( ) funksiýany ulanmak gerek (for sikli muny awtomatiki ýerine ýetirýär).

iter( ) funksiýa

Bu funksiýany iki usul bilen ulanyp bolýar. Birinji ýagdaýda bir argumenti bar, ol bolsa öz iteratoryny bermeli.

Ikinji ýagdaýda argumentleriň biri – argumentsiz funksiýa, ikinji – ahyrky baha. Iterator görkezilen funksiýany soňuna çenli çagyrýar. Ikinji usul birinjä garanyňda seýrek duş gelýär, sebäbi "boş ýerden" bahany çykarmak kyn:

```
it1=iter([1,2,3,4,5])
def forit (mystate=[]):
    if len(mystate)<3:
        mystate.append("")
        return ""
    it2=iter(forit, None)
    print([x for x in it1])
    print([x for x in it2])
```

Bellik:

Eger funksiýa bahany çykarmaýan bolsa, onda ol None bahany çykaryar. Bu ýagdaý ýokardaky meselede duş gelýär.

enumerate( ) funksiýa.

Bu funksiýa beýleki iteratoryň elementlerini nomerleýän iteratory döredýär. Netijeleýji iterator toplumy berýär, onda birinji element – nomer (noldan başlap), ikinji – başlangyç yzygiderligiň elementi:

```
>>> print([x for x in enumerate("abcd")])
[(0, 'a'), (1, 'b'), (2, 'c'), (3, 'd')]
```

sorted( ) funksiýa

Python 2.4 görnüşde dörän bu funksiýa sortlamany ýerine ýetirýän iteratory döretmäge ýardam berýär:

```
>>> sorted('bolmak')
['a', 'b', 'k', 'l', 'm', 'o']
```

Indi itertools modulyň funksiýalaryna seredeliň.

itertools.chain( ) funksiýa

Bu funksiýa birnäçe yzygider birleşdirilen iteratorlardan ybarat iteratory düzmäge ýardam berýär. Iteratorlar aýratyn argument hökmünde berilýär. Mysal üçin:

```
from itertools import chain
it1=iter([1,2,3])
it2=iter([8,9,0])
for i in chain(it1,it2):
    print(i)
```

Netije şeýle bolar:

```
>>>
1
2
3
8
9
0
```

itertools.repeat( ) funksiýa

repeat( ) funksiýa käbir obýekti berlen n gezek gaýtaladýan iteratory düzýär:

```
from itertools import repeat
for i in repeat(1,4):
   print(i)
```

Netijede:

```
>>>
1
1
1
1
```

itertools.count( ) funksiýa

Berlen sandan başlap, bitin sanlary berýän tükeniksiz iterator:

```
for i in itertools.count(1):
print i,
if i > 100:
break
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57
58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75
76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93
94 95 96 97 98 99 100 101
```

itertools.cycle( ) funksiýa

Käbir yzygiderligi (ýa-da beýleki iteratoryň bahasyny) tükeniksiz gaýtaladyp bolýar:

```
from itertools import cycle
yzygiderlik =[1,2,3]
for i in cycle(yzygiderlik):
    print(i)
```

Netijede
1231231231231231231231231231231231231231…

itertools. starmap( ) we itertools.filter( ) funksiýalar

starmap( ) funksiýa edil map( ) funksiýa meňzeş, ýöne bary-ýogy iki argumenti bar. Ikinji argument – yzygiderlik bolup, onuň her elementi funksiýanyň birinji argumenti üçin parametrleriň toplumyny berýär:

```
from itertools import starmap
for i in starmap(lambda x,y:str(x)+y,[(1,'a'), (2,'b')]):
    print(i)
```

Netijede:

```
>>>
1a
2b
```

Filterfalse( ) funksiýa funksiýanyň bahasyna otrisatel alamaty berýär:

```
from itertools import filterfalse
for i in filterfalse(lambda x:x>0, [1,-2,3,-3]):
    print(i)
```

Netijede:

```
>>>
-2
-3
```

itertools.takewhile( ) we itertools.dropwhile( ) funksiýalary

Süzgüjiň ýene-de görnüşleri: takewhile( ) we onuň tersi bolan dropwhile( ). Indiki mysalda olaryň işleýiş düzgüni getirilen:

```
from itertools import takewhile
from itertools import dropwhile
for i in takewhile(lambda x:x>0, [1,-2,3,-3]):
    print(i)
print
for i in dropwhile(lambda x:x>0, [1,-2,3,-3]):
```

```
    print(i)
```

Netijede:

```
>>>
1
-2
3
-3
```

Şeýlelikde, takewhile( ) funksiýa şert ýerine ýetýänçä bahalary berýär, iteratoryň galan bahalaryny bolsa bermeýär. Onuň tersine, dropwhile( ) funksiýa şert ýerine ýetýänçä hiç zat berenok, soňra bolsa ähli bahalary berýär.

Itertools.groupby( ) funksiýa.

Bu funksiýanyň iki argumenti bar: iterator (zerur) we zerur däl – açaryň bahasyny berýän funksiýa: groupby (iterable [,func]). Indiki mysalda položitel we otrisatel elementleriň yzygiderligi toplanýar:

```
import itertools, math
from itertools import groupby
lst=map(lambda x: math.sin(x*.4), range(30))
for k, i in groupby(lst, lambda x: x>0):
    print(k,list(i))
```

Netijede:

```
>>>
False [0.0]
True [0.3894183423086505, 0.7173560908995228,
0.9320390859672264, 0.9995736030415051,
0.9092974268256817, 0.6754631805511506,
0.33498815015590466]
False [-0.058374143427580086, -0.44252044329485246, -
0.7568024953079282, -0.951602073889516, -
0.9961646088358406, -0.8834546557201531, -
0.6312666378723208, -0.27941549819892586]
True [0.11654920485049364, 0.4941133511386089 4,
0.7936678638491531, 0.9679196720314865,
0.9893582466233818, 0.8545989080882804,
0.5849171928917617, 0.22288991410024592]
False [-0.1743267812229814, -0.5440211108893698, -
0.8278264690856537, -0.9809362300664916, -
0.979177729151317, -0.8228285949687079]
```

itertools.tee( ) funksiýa.

Bu funksiýa iteratorlary klonirlemäge ýardam berýär. Onuň birinji argumenti – klonirleniljek iterator. Ikinji argument – (N) – – gerekli nusgalaryň sany. Funksiýa N iteratorlaryň toplumyny çykarýar. N sanyň iň kiçi bahasy 2-ä deň.

Hususy iterator.

Bu ýerde ulanyjy tarapyndan kesgitlenen iteratoryň mysalyna seredeliň:

```
class Fibonacci:
" " "N-e çenli Fibonacci yzygiderligiň iteratory" " "
def _init_(self,N):
        self.n, self.a, self.b, self.max=0,0,1,N
def _iter_(self):
   #öz-özüne iterator: synpda next( ) usuly bar
   return self
def next(self):
        if self.n<self.max:
   a,self.n, self.a, self.b = self.a, self.n+1,
self.b, self.a+self.b
   return a
else:
   raise StopIteration
for i in Fibonacci(100):
   print(i)
```

Sada generatorlar.

Bu dili işläp düzýän programmistler iteratorlarda durmadylar. Python dilinde sada generatorlary işletmek örän aňsat eken. Sada generator diýip, ýörite obýekte diýilýär, onda hasaplamalar baahny doly çykarmak üçin dowam edýär we indiki bahany çykarmagyň zerurlygy ýüze çykýança bes edilýär. Sada generator gerator-funksiýa tarapyndan emele gelýär, ol bolsa sintaksiki manyda adaty funksiýa meňzeş, ýöne indiki bahany çykarmak üçin ýörite yield operatoryny ulanýar. Çagyrylanda şeýle funksiýa hiç zady hasaplamaýar, ol hasaplamalary almak üçin iteratoryň interfeýsi bilen obýekti döredýär. Başga sözler bilen aýdylanda, eger-de funksiýa yzygiderligi berilmeli bolsa, ondan sada generatory etmek aňsat, ol bolsa öz-özi jogabyny çykarýar. Bular ýaly hasaplamalar iň soňky pursata çenli goýbolsun edilýär we netijede alynýan baha indiki hasaplamalar üçin ulanylýar.

Fibonaççiniň yzygiderliginiň mysalynda şeýle generatory gurup bolýar:

```
def Fin(N):
   a,b = 0,1
for i in range (N):
   yield a
a,b = b, a+b
```

Muny ulanmak beýleki iteratorlardan kyn däl:

```
for i in Fib(100):
   print(i)
```

Generatorly aňlatma

Sintaksiki many boýunça ol sanawa meňzeş, ýöne kwadrat ýaýlaryň ýerine adaty ýaýlar ulanylýar. Sanaw girizmeler sanawlary döredýär, diýmek, huşda köp ýer eýelemek mümkin. Generatorly aňlatma ulanylanda bolsa, sanaw döredilenok, onuň ýerine her indiki baha zerur bolsa hasaplanylýar (next( ) usul çagyrylanda).

Indiki mysalda faýldaky setirlerden käbir çalyşmalar ýerine ýetirlýär:

```
for line in (l.replace("-","-") for l in open("input.dat")):
   print(line)
```

Iteratorlary faýla ýazmak üçin hem ulanyp bolýar:

```
open("output.dat", "w").writelines(
       l.replace("-", "-") for l in open("input.dat"))
```

Bu ýerde generatorly aňlatma üçin goşmaça ýaýlar hökman däl, sebäbi ol çagyrylan funksiýanyň ýaýlarynyň içinde ýerleşen.

Karring

Xoltar toolkit kitaphanasynda (ýazary Bryn Keller) functional moduly bar, ol bolsa funksional programmirlemäniň mümkinçiliklerini ulanmaga ýardam berýär.

Funksiýanyň karringinde (bölekleýin ulanylanda) başlangyç funksiýanyň käbir argumentlerini berýän täze funksiýa döredilýär. Indiki mysalda tapawudyň bölekleýin ulanylyşy suratlandyrylýar.

```
from functional import curry
def subtract(x, y):
    return (x + y)
    print (subtract(3, 2))
    subtract_from_3 = curry(subtract, 3)
print (subtract_from_3(2))
print (curry(subtract, 3)(2))
```

Ähli üç ýagdaýda-da 1 bolar. Indiki mysalda ikinji argument ulanylyp, täze funksiýa alynýar. Beýleki argumentiň ýerine ýörite Blank aňatma goýulýar.

```
from functional import curry, Blank
def subtract(x, y):
    return x + y
```

```
print (subtract(3, 2))
subtract_2 = curry(subtract, Blank, 2)
print (subtract_2(3))
print (curry(subtract, Blank, 2)(3))
```

Netije

Bu Sapakda funksional programmirlemäniň gurluş düzgüni seredilip geçildi. Mundan başga-da Python dilinde we onuň standart modullarynda funksional programmalary döretmek üçin örän gowy serişdeleriniň bardygy görkezildi.

Ýene-de bir zady bellemek gerek. Iteratorlar – bu Python dilinde funksional başlangyjyň ykjam dowamydyr. Iteratorlar öz manysy boýunça “lazy computations” atly hasaplamalary gurnamaga ýardam berýär, ýagny bahalary diňe zerur ýagdaýynda hasaplap berýärler.

4. OBÝEKTE GÖNÜKDIRILEN PROGRAMMIRLEME

Python dili programmirlemäniň obýekte gönükdirilen dil hökmünde düzülen. Bu bolsa (Smalltalk atly obýekte gönükdirilen diliň ýazary Alan Keýiň nazaryýeti boýunça) aşakdaky prinsiplerden düzülendigini aňladýar:

1. Ähli maglumatlar obýektler hökmünde göz-öňüne getirilen.
2. Programmany bir-birine habar ugradýan we özara täsirleşýän obýektleriň toplumy görnüşinde düzüp bolmaly.
3. Her obýektiň öz hususy ýat bölegi bolmaly we beýleki obýektlerden durup bilmeli.
4. Her obýektiň tipi bolmaly.
5. Bir tipli obýektleriň hemmesi şol bir habary alyp (ýa-da şol bir hereketi edip) bilmeli.

Obýekte gönükdirilen programmirleme – bu kody ýazmagyň usulydyr. Python programmirleme diliniň ýeterlik derejede güýçli obýekte gönükdirilen programmirlemäniň (OGP) goldawy bar.

Bellik:

Gynansakda OGP girişlerinin köpüsinde örän köp adalgalar bar, olar bolsa meseläniň düýp manysyna düşünmekligi kynlaşdyrýar. Bu ýerde diňe programmistleriň programma işläp düzenlerinde ulanjak ýa-da dünýägaraýşyňy giňeltjek adalgalar ulanylar. Dürli programmirleme dillerinde OGP-niň öz aýratynlyklary bolany üçin ýaýlarda şol adalgalaryň sinonimleri ýa-da meňzeşleri berler.

Esasy düşünjeler

Proseduraly programmirlemede programma algoritme laýyklykda böleklere bölünýär: her bölegi (kiçi programma, funksiýa, prosedura) algoritmiň düzüji bölegi bolup durýar.

Obýekte gönükdirilen programmirlemede programma özara täsirleşýän obýektleriň jemi ýaly gurulýar.

Obýekte gönükdirilen çemeleşmä laýyklykda obýekt – bu bahasy (mazmuny), görnüşi (häsiýeti) we aýratynlygy bolan bir zat. Obýektleriň üstünde amallary ýerine ýetirip bolýar (habar iberip bolýar). Python dilinde ähli maglumatlar obýektler görnüşinde göz-öňüne getirilen.

Obýektleriň özara täsiri bir obýektleriň usullarynyň beýleki obýektler tarapyndan çagyrylmakdan ybarat. Adatça bular ýaly ýagdaýda obýektler bir-birlerine habar ugradýarlar diýýärler. Habar – bu obýektiň käbir hereketleri ýerine ýetirmegi üçin ýüzlenme. (Habar, usul, amal sinonimlerdir).

Her obýektiň özüniň ýagdaýy bar (munuň üçin onuň atributlary bar) we kesgitli usullaryň toplumy bar. (Sinonimler: atribut, meýdan, slot, obýekt-agza). Usullar obýektiň özüni alyp barşyny kesgitleýärler. Synpyň obýektleriniň umumy özüni alyp barşy bar.

Obýektler suratlandyrylanda hersi aýratyn edilmän, synplaryň kömegi bilen ýerine ýetirilýär. Synp – obýektleriň ülňüsi bolan obýekt. Käbir synpyň esasynda döredilen obýekte synpyň nusgasy diýilýär.

Python dilinde synpy kesgitlemek üçin class operator ulanylýar:

```
class synpyň_ady (synp1, synp2, …):
   # usullaryň kesgitlenilişi
```

Synp obýektiň görnüşini kesgitleýär, ýagny onuň mümkin bolan ýagdaýlaryny we amallaryň toplumyny.

Abstraksiýa we dekompozisiýa

OGP-de abstraksiýa düzülýän informasion modeliň nukdaý nazaryndan uly bolmadyk böleklerine üns bermezden, maglumatlardan we bu maglumatlaryň gaýtadan işleýiş algoritmlerinden obýektleri düzmäge ýardam berýär. Şeýlelikde, programma dekompozisiýa sezewar bolýar.

Hatda sadaja obýekte gönükdirilen programmany ýazmazdan öň obýektleriň synplaryny ýüze çykarmak üçin derňew geçirmeli.

Obýektler belli edilende olara mahsus bolan häsiýetleriň köpüsinden ünsi çekip, meselä ähmiýetli bolan häsiýetlerde ünsi jemlemeli.

Bellenilýän obýektler fiziki obýektlere meňzemek hökman däl – çünki olar abstraksiýalar, diýmek olar prosesler, özaratäsirler, gatnaşyklar hem bolup biler.

Şowly dekompozisiýa köp zada degýär. Oňa diňe bir kodyň mukdar häsiýeti bagly bolman, eýsem soňky işiň gidişi hem bagly. Programma ýazylanda ony mümkin boldugyça anyk meselä degişli we sada etmeli.

Prosedur çemeleşmede hem dekompozisiýa ulanylýar, ýöne obýekte gönükdirilen çemeleşmede has kiçi bölekleriniň algoritminiň dekompozisiýasy bolman, obýektleriň synplarynyň dekopmozisiýasy bolýar.

Obýektler

Python dilininiň obýektleri öň duş geldi: her bir san, setir, funksiýa, modul we ş.m. – bularyň ählisi obýektlerdir. Käbir gurnalan obýektler Python dilinde sintaksiki goldawa eýe. Olara sanlar, setirler, sanawlar we beýleki görnüşler degişlidir. Indi bu obýektlere ýokarda getirilen kesgitlemelerde seredip geçeliň. Mysal üçin:

```
a = 3
b = 4.0
c = a + b
```

Bu ýerde şeýle zatlar bolup geçýär. Ilki “a” at atlaryň ýerli giňişliginde 3(bitin san) obýekt-san bilen baglaşýar. Soňra “b” at 4.0(hakyky san) obýekt-san bilen baglaşýar. Mundan soň 3 we 4.0 obýektleriň üstünde goşmak amaly ýerine ýetirilýär we “c” at alnan obýekt bilen baglaşýar. Şeýle-de amallara, esasan, usullar

diýiler we Python dilinde olar sintaksiki goldawa eýe, ýokardaky mysalda – infiks ýazgy bolup durýar. Bu mysaly şeýle görnüşde hem ýazyp bolar:

```
c = a._add_(b)
```

Bu ýerde _add_( ) – a obýektiň usuly, ol bir obýekt bilen beýleki obýektiň arasynda + amalyny ýerine ýetirýär.

Käbir obýektiň usullarynyň toplumyny dir( ) funksiýanyň kömegi bilen bilip bolar:

```
>>> dir(a)
['__abs__', '__add__', '__and__', '__bool__',
'__ceil__', '__class__', '__delattr__', '__dir__',
'__divmod__', '__doc__', '__eq__', '__float__',
'__floor__', '__floordiv__', '__format__', '__ge__',
'__getattribute__', '__getnewargs__', '__gt__',
'__hash__', '__index__', '__init__', '__int__',
'__invert__', '__le__', '__lshift__', '__lt__',
'__mod__', '__mul__', '__ne__', '__neg__', '__new__',
'__or__', '__pos__', '__pow__', '__radd__', '__rand__',
'__rdivmod__', '__reduce__', '__reduce_ex__',
'__repr__', '__rfloordiv__', '__rlshift__', '__rmod__',
'__rmul__', '__ror__', '__round__', '__rpow__',
'__rrshift__', '__rshift__', '__rsub__',
'__rtruediv__', '__rxor__', '__setattr__',
'__sizeof__', '__str__', '__sub__', '__subclasshook__',
'__truediv__', '__trunc__', '__xor__', 'bit_length',
'conjugate', 'denominator', 'from_bytes', 'imag',
'numerator', 'real', 'to_bytes']
```

Bu ýerde Python diliniň ýenede bir aýratynlygyny belläp bolar. Diňe bir infiks amallar däl, eýsem gurnalan funksiýalar hem obýektiň käbir usullarynyň toplumyna eýe. Mysal üçin, şeýle ýazyp bolar:

```
abs(c)
```

abs( ) funksiýa bolsa hakykatda oňa berlen obýektiň usulyny ulanýar:

```
c._abs_( )
```

Obýektler funksiýalar çagyrylanda döreýär we obýekte bolan soňky salgylanma ýok edilende ýok bolýarlar. del operator atlar giňişliginden ady (diýmek, obýekte bolan salgylanmany) ýok edýär.

```
a = 1
```

```
#...
del a
#a at indi ýok
```

Tipler we synplar

Tip obýektiň mümkin bolan bahalarynyň çägini we onuň üstünden edilýän amallaryň toplumyny kesgitleýär. OGP-de tip özüni alyp barmak bilen, ýagny obýektiň hereketleri bilen örän berk baglanyşykly.

Öň Python dilinde maglumatlaryň gurnalan görnüşleri synpyň nusgalaryna degişli däldi, şonuň üçin olar kesgitli görnüşin obýektleridi. Indi gurnalan görnüşleriň obýektleriniň synpy bar. Şeýlelikde, tip we synp Python dilinde sinonimler bolup duryar.

Python diliniň interpretatory obýektiň haýsy görnüşe degişlidigini aýdyp biler. Ýöne obýekt amalda ulanylanda onuň haýsy synpa degişlidigi möhüm däl, esasy zat, obýekt nähili usullary goldaýandygy.

Synplaryň nusgalary programmada diňe bir literallardan ýa-da amallaryň netijelerinde döränok. Adatça synpyň obýektini almak üçin bu synpyň käbir parametrleri bilen konstruktoryny çagyrmak ýeterlik.

```
>>> s=set([1,2,3])
>>> print(s)
{1, 2, 3}
```

Bu mysalda Set synpyň konstruktory [1,2,3] parametrleri bilen çagyrylýar. Netijede, üç elementden 1,2,3 ybarat bolan köplük baglaşýar.

Synpy kestilemek.

Goý Graf synpy kesgitlemek gerek bolsun. Graf – bu depeleriň köplügi we bu depeleri jübüt edip birleşdirýän gapyrgalaryň toplumy. Grafyň üstünde şeýle amallary ýerine ýetirip bolýar, ýagny depeleri, gapyrgalary goşmak, grafda gapyrgalaryň barlygyny barlamak we ş.m. Python dilinde synpy kesgitlemek şeýle görnüşde bolar:

```
class G:
    def __init__(self, V, E):
            self.vertices = set(V)
            self.edges = set(E)
    def add_vertex(self, v):
            self.vertices.add(v)
    def add_edge(self, v1, v2):
            self.vertices.add(v1)
            self.vertices.add(v2)
            self.edges.add((v1, v2))
    def has_edge(self, v1, v2):
            return (v1, v2) in self.edges
```

```
    def __str__(self):
            return "%s;%s"%(self.vertices,self.edges)
```

Synpy şeýle görnüşde ulanyp bolar:

```
g = G([1, 2, 3, 4], [(1, 2), (2, 3), (2, 4)])

print (g)
g.add_vertex(5)
g.add_edge((5,6))
print (g.has_edge((1,6)))
print (g)
```

Netijede şeýle alynar:

```
Set([1, 2, 3, 4]); Set([(2, 4), (1, 2), (2, 3)])
False
  Set([1,2,3,4,5,6]); Set([(2,4), (1,2), (5,6), (2,3)])
```

Ýokardaky mysaldan görnüşi ýaly, synpy kesgitlemek kyn däl. Synpyň konstruktorynyň ýörite `_init_` ady bar (destruktoryň ady `_del_`). Synpyň usullary synpyň atlarynyň giňişliginde kesgitlenýärler. Usulyň birinji argumenti hökmünde `self` ulanmak kabul edilen. Usullardan başga synpyň obýektinde iki atribut bar: vertices (depeler) we edges (gapyrgalar). G obýektiň setir görnüşinde göz-öňüne getirmek üçin ýörite _str_( ) usuly ulanylýar.

Haýsy synpa degişlidigini gurnalan funksiýanyň isinstance( ) kömegi bilen bilip bolar:

```
print (isinstance (g, G))
```

Inkapsullaşma

Adatça, OGP-ni inkapsullaşmasyz göz-öňüne getirip bolanok diýilýär, bu esasy düşünje. Programmirlemäniň ösüşi programma üpjünçiliginiň çylşyrymlylygy bilen göreşmegi dowam edýär. Örän köp programmistleriň gatnaşmagynda döredilýän örän uly programma ulgamlarynyň çylşyrymlylygy ýokary derejelerinde pes derejeleriniň amala aşyrylyş bölekleri görünmese azalýar. Hususanda, bu çykalgada proseduraly çemeleşme ilkinji ädim bolupdy. Inkapsulýasiýa diýip (incapsulation – “gabamak” diýip bolar) obýektiň içki gurluşy baradaky maglumaty gizlemek diýip düşünilýär, bu ýerde obýekt bilen işlemek diňe onuň köpçülige elýeter (public) interfeýsiniň üsti bilen bolar. Şeýlelikde, beýleki obýektler işlenilýän obýektleriň “içki işine” goşulman, diňe çagyryşlary ulanyp biler.

Häsiýetlere bolan elýeterlilik

Python dilinde käbir atributlara baryp bolýar, olar synpyň interfeýsi hökmünde suratlandyrylan. Şeýle atributlara häsiýetler (properties) diýilýär.

Porgrammirlemäniň beýleki dillerinde häsiýetlere barmak üçin ýörite usullar döredilýär. Python dilinde atributa bolan salgylanma ulanmak ýeterlik, elbetde, eger-de häsiýet obýektde hiç zada täsir etmeýän bolsa. Eger-de obýektde käbir hereketler talap edilýän bolsa, onda ony häsiýet hökmünde suratlandyryp bolar:

```
class G:
class C(object):
        def getx(self): return self.__x
  def setx(self, value): self.__x = value
 def delx(self): del self.__x
  x=property(getx,setx,delx,"I'm the 'x' property.")
```

Sintaksiki seredilende x-iň häsiýetine barmak atributyň adaty salgylanmasy ýaly bolar:

```
>>> c = C()
>>> c.x = 1
>>> print c.x
1
>>> del c.x
```

Ýene bir zady belläliň, Python dilinde synpyň nusgasynda islendik atributa barmagy gurnap bolýar, onuň üçin ýörite usullary çagyrmagyň üstünde işlemeli:

getattr(self, name)	Obýektiň bu usuly diňe atribut başga ýagdaýda tapylmasa çagyrylýar. Bu ýerde name – atributyň ady. Usul atributyň bahasyny hasaplamaly ýa-da ýalňyşy çagyrmaly AttributeError. “Täze synplaryň” atributlaryny dolylygyna dolandyrmak üçin _getattribute_( ) usuly ulanylýar.
setattr(self, name, value)	Bu usul käbir atributa baha berlende çagyrylýar. _getattr_( ) usuldan tapawutlylykda, bu usuly hemişe çagyryp bolýar, şonuň üçin bu usuly seresap ulanmaly, ýogsa rekursiýany döredip bolýar.
delattr(self, name)	Adyndan belli bolşy ýaly, bu usul atributy ýok etmek üçin ulanylýar.

Indiki mysalda ýokarda agzalan usullaryň ählisi görkezilen. Bu mysalda sözlükden obýekt döredilýär, onuň atributlarynyň atlary sözlügiň açarlary bolar, bahalary bolsa – sözlükdäki berlen açarlar boýunça bahalar.

```
class AttDict(object):
    def __init__(self, dict=None):
        object.__setattr__(self,'_selfdict',dict or {})

    def __getattr__(self, name):
        if self._selfdict.has_key(name):
            return self._selfdict[name]
        else:
            raise AttributeError

    def __setattr__(self, name, value):
        if name[0] != '_':
            self._selfdict[name] = value
        else:
            raise AttributeError
    def __delattr__(self, name):
        if name[0] != '_' and
self._selfdict.has_key(name):
            del self._selfdict[name]

    ad = AttDict({'a': 1, 'b': 10, 'c': '123'})
    print (ad.a, ad.b, ad.c)
    ad.d = 512
    print (ad.d)
```

Maglumatlary gizlemek

Atributyň adynyň başynda çyzmaklyk (“_”) elýeter interfeýse girmeýändigini görkezýär. Adatça bir çyzyk ulanylýar, ol bolsa kän uly orny eýelemeýär, ýöne programmistler şeýle diýýärler: “bu usul diňe içki ulanylyş üçin”. Iki çyzyk ulanylanda atribut hususy diýmegi aňladýar. Üstesine atribut elýeterli, ýöne başga at bilen, muny bolsa aşakda görüp bolar:

```
  >>> class x:
  …  x=0
  …  _x=0
  …  __x=0
  …
  >>> dir(x)
  ['__class__',  '__delattr__',  '__dict__',  '__dir__',
'__doc__',      '__eq__',      '__format__',      '__ge__',
'__getattribute__',  '__gt__',  '__hash__',  '__init__',
'__le__',  '__lt__',  '__module__',  '__ne__',  '__new__',
'__qualname__',       '__reduce__',       '__reduce_ex__',
'__repr__',   '__setattr__',   '__sizeof__',   '__str__',
'__subclasshook__', '__weakref__', '_x', '_x__x', 'x']
```

Polimorfizm

Grek dilinden terjime edilende polimorfizm “köp görnüşlilik” diýmegi aňladýar. Muny informatikada bir ady dürli hereketleri ýerine ýetirmek üçin ulanmagyň mümkinçiligine diýilýär.

Polimorfizme programmirlemäniň diline baglylykda köp kesgitlemelere duş gelip bolar. Adatça, polimorfizmiň ýüze çykmagy hökmünde usullara täzeden kesgitlemä aýdylýar.

Python dilinde polimorfizm häsiýetleri alyp galmak bilen bagly däl-de, synpdaky elýeter usullaryň toplumy we manysy bilen bagly. Aşakda kesgitli usullara eýe bolup, setir üçin ýa-da islendik gurnalan görnüş üçin synpy döredip boljakdygy görkeziler. Munuň üçin tipe mahsus bolan usullaryň toplumyny kesgitlemek gerek. Elbetde, usullaryň gerekli toplumyny alyp bolýar, ýöne Python dilinde bu gerek däl.

Python dilinde funksiýa ýazylanda gerekli argument haýsy tipe (synpa) degişlidigi barlanylmaýar: käbir usullar berlen obýektde ulanylýar. Şeýlelikde, funksiýalar mümkin boldugyça umumylaşdyrylan bolýar: olar obýekt-parametrlerden kesgitli atly, argumentleriň toplumy we semantikasy bolan usullaryň bolmagyndan artyk hiç zat talap etmeýärler.

Indiki mysalda Python diline mahsus bolan polimorfizm görkezilen:

```
def get_last(x):
    return x[-1]
print (get_last([1, 2, 3]))
print (get_last("abcd"))

>>>
3
d
```

Ýokarda beýan edilen funksiýada argument hökmünde -1 indeksi (soňky element) alyp boljak ähli zat gabat geler. Ýöne “soňky elementi almak” semantikasy diňe yzygiderlikler üçin ýerine ýetiriler. Bu funksiýa sözlükler üçin hem işlär, ýöne manysy üýtgär.

Tiplere öýkünmek

Polimorfizmi suratlandyrmak üçin “funksiýa” gurnalan tipe meňzeş hususy görnüşi gurup bolar. Usullara ýa-da funksiýalara meňzeş çagyrylýan synplary, obýektleri şeýle gurup bolýar:

```
class CountArgs(object):
    def __call__(self, *args, **kwargs):
        return len(args) + len(kwargs)
cc = CountArgs()
print (cc(1, 3, 4))
```

```
>>>
3
```

Bu mysaldan görnüşi ýaly, CountArgs synpyň nusgalaryny funksiýalar ýaly çagyryp bolýar. Nusga çagyrylanda `_call_( )` usul ähli argumentleri bilen çagyrylýar.

Indiki mysalda synpyň nusgalarynyň deňeşdirmelerini hem dolandyryp bolýandygyny görüp bolýar:

```
class Point:
    def __init__(self, x, y):
        self.coord = (x, y)
    def __nonzero__(self):
        return self.coord[0] !=0 or self.coord[1]!= 0
    def __cmp__(self, p):
        return cmp(self.coord, p.coord)
for x in range(-3, 4):
    for y in range(-3, 4):
        if Point(x,y) < Point(y,x):
            print ("*"),
        elif Point(x,y):
                print ("."),
        else:
            print ("o"),
print
```

Programma şeýle çykarar:

```
. * * * * * *
. . * * * * *
. . . * * * *
. . . o * * *
. . . . . * *
. . . . . . *
. . . . . . .
```

Bu programmada Point (Nokat) synpyň _nonzero_( ) usuly bar, ol bolsa synpyň obýektiniň hakyky bahasyny kesgitleýär. Çyn bahany diňe bahasy (0,0)-dan tapawutly nokatlar berer. Beýleki usul - _cmp_( ) nokady beýleki obýekt bilen deňeşdirmek zerurlygy bolanda çagyrylýar. Bir zady belläliň, ýagny _cmp_( ) usulyň ýerine deňeşdirme amaly üçin aýratyn usullary kesgitläp bolar: `_1t_`, `_1e_`, `_ne_`, `_eq_`, `_ge_`, `_gt_` (degişlilikde <, <=, !=, <>, =>, > üçin).

San görnüşleri hem hasaplatmak ýeterlik derejede aňsat. Infiksli + sintaksisiniň amatlyklaryny ulanýan synpy şeýle kesgitläp bolar:

```
class Plussable:
```

```
    def __add__(self, x):
...
    def __radd__(self, x):
...
    def __iadd__(self, x):
...
```

Bu ýerde `_add_( )` usul Plussable synpyň nusgasy goşmagyň çep tarapynda duranda çagyrylýar, `_radd_( )` – eger goşmagyň sag tarapynda bolsa we ondan çepdäki usulda `_add_( )` usuly ýok bolsa. `_iadd_( )` usul += amala aşyrmak üçin gerek.

Synplaryň arasyndaky gatnaşyk

Dowam etdirmek

Programmirlemede köp ýagdaýlarda şeýle ýagdaýlar duş gelýär, ýagny örän ýakyn, ýöne şol bir wagtda meňzeş bolmadyk synplar. Synplaryň birmeňzeşligini ulanmagyň hasabyna olary gysgaça ýazmak usullarynyň biri synplary köpbasgançaklyga düzmek. Bu köpbasgançaklygyň düýbünde esasy synp durýar, ondan bolsa galan synplar öz atributlaryny dowam etdirýärler we özlerini alyp baryşlaryny takyklaýarlar we giňeldýärler. Adatça synplaşdyrmany gurmagyň esasy "IS-A" ("BAR") gatnaşyk bolup durýar. Mysal üçin, Töwerek synpy programmada – grafiki redaktorda Geometriki Figura synpyny dowam etdirip biler. Üstesine Töwerek synpy Geometriki Figura synpynyň kiçi synpy bolar.

Python dilinde synplaryň köpbasgançaklyklarynyň başynda object synpy durýar. Bu köpbasgançaklylykda bulaşmazlyk üçin käbir gurnalan funksiýalar bar, olara aşakda seredip geçeris. Funksiýa `issubclass`(x,y) x synpyň y synpyň kiçi synpy bolup biljekdigini aýdyp biler:

```
>>> class A(object): pass

>>> class B(A): pass

>>> issubclass(A, object)
True
>>> issubclass(B, A)
True
>>> issubclass(B, object)
True
>>> issubclass(A, str)
False
>>> issubclass(A, A) #synp öz synpynyň kiçi synpy
True
```

Synplaşdyrmany gurmagyň esasynda hemişe derňelýän we modelirlenýän ulgamda esasy orny eýeleýän prinsip ýatyr.

Eger-de synp dowam etdirmek üçin niýetlenen bolsa, oňa abstrakt diýilýär. Abstrakt synpyň nusgalarynyň uly manysy ýok. Işçi nusgaly synplara takyk diýilýär.

Python dilinde abstrakt synpyň mysaly hökmünde `basestring` gurnalan tip bolup biler, onuň takyk kiçi synplary bar `str` we `unicode`.

Köp gezek dowam etdirmek

Mysal üçin, Java dilinden tapawutlylykda Python dilinde synpy birnäçe synplardan dowam etdirip bolar. Beýle ýagdaýa köp gezek dowam etdirmek diýilýär (multiple inheritance).

Birnäçe synplary dowam etdirýän synp özünde olaryň häsiýetlerini jemleýär. Bular ýaly ýagdaýy örän seresap ulanmaly, oňa bolan zerurlyk bolsa seýrek duş gelýär:

- Köp gezek dowam etdirmekligi berlen elýeter usullary bolan synpy almak üçin ulanyp bolar, bu ýerde usullary birinji synp berýär we ikinji synpyň usullarynyň üsti bilen amala aşyrylýar. Birinji synp dolylygyna abstrakt bolup biler.
- Bu ýagdaý köp dürlüligi (mixins) goşmak üçin ulanylýar. Bu diýmek – ýörite gurnalan synp bolup, käbir synpa käbir häsiýetleri (atributlary) goşýar. Bular ýaly synplar adatça abstrakt bolup durýar.
- Örän seýrek ýagdaýda özüniň esasy manysy boýunça ulanylýar, haçan-da synpyň köp gezek dowam etdirmegiň netijesinde alynýan obýektleri ähli esasy synplaryň obýektleri hökmünde ulanmak üçin niýetlenende.

Python dilinde dowam etdirmäni usullaryň toplumyny synplaryň böleklerine ýygnamak üçin ulanylýar:

```
class A:
    def a(self): return 'a'
class B:
    def b(self): return 'b'
class C:
    def c(self): return 'c'
class AB(A, B):
    pass
class BC(B, C):
    pass
class ABC(A, B, C):
    pass
```

Gerekli usullary başgaça hem ýygnap bolýar:

```
def ma(self): return 'a'
def mb(self): return 'b'
def mc(self): return 'c'
class AB:
```

```
    a = ma
    b = mb
class BC:
    b = mb
    c = mc
class ABC:
    a = ma
  b = mb
  c = mc
```

Agregasiýa

Konteýnerler

Konteýner diýlip esasy wezipesi beýleki obýektlere bolan ýoly saklamak we barmak bolan obýektlere aýdylýar. Konteýnerler obýektleriň arasyndaky “HAS-A” (“EÝE”) atly gatnaşygy amala aşyrýar. Gurnalan görnüşler, sanaw we sözlük – konteýnerleriň aýdyň mysaly. Öz hususy konteýnerleri hem gurup bolýar, ol ýerde saklanýan obýektlere baryp bolýar. Konteýnerde obýektleriň özi däl-de, olara bolan salgylanmalar saklanýar.

Adatça Python dilinde gurnalan konteýnerler ýeterlik (sözlük we sanaw), ýöne zerur bolan halatynda beýleki konteýnerleri hem döredip bolýar. Aşakda sanawyň esasynda amala aşyrylan Stek synpy getirilen:

```
class Stack:
    def __init__(self):
        """Steki işe göýbermek """
        self._stack = [ ]
    def top(self):
        """Stekiň depesinigaýtarmak"""
return self._stack[-1]
    def pop(self):
"""Stekden elementi aýyrmak"""
returnself._stack.pop()
defpush(self, x):
        """Elementi steke ýerleşdirmek"""
self._stack.append(x)
    def __len__(self):
"""Stekde elementleriň sany"""
returnlen(self._stack)
def __str__(self):
        """Setir görnüşinde görkezmek """
        return":".join(["%s"%e for e in self._stack])
```

Ulanylanda:

```
>>> s=Stack()
```

```
>>> s.push(1)
>>> s.push(2)
>>> s.push("abc")
>>> print(s.pop())
abc
>>> print(len(s))
2
>>> print(s)
1 : 2
```

Şeýlelikde, konteýnerler beýleki islendik obýektleri olaryň saklanyş gurluşyna laýyklykda we obýektleriň içki işlerine goşulmazdan dolandyrmaga ýardam berýär. `Stack` synpyň interfeýsini bilip, onuň sanawyň esasynda amala aşyrylandygyny we onuň kömegi bilen nädip bolandygyny aňman hem bolar. Ýöne stek ulanmak üçin bu möhüm däl.

Bellik:
Ýokardaky mysalda gysgaça beýan etmek üçin käbir hereketleriň netijesinde ýalňyşlar ýüze çykyp biljekdigi göz-öňüne tutulmandyr. Mysal üçin, boş stegiň depesinden elementi aýyrmaga synanyşyk edilse.

Iteratorlar

Iteratorlar –bu konteýneriň elementlerine yzygider barmak üçin mümkinçilik berýän obýektler. Iterator elementleri ýeke-ýeke saýlamaga mümkinçilik berýär. Indiki mysalda N boýunça sanawdan bahalary berýän iterator berlen:

```
class Zahlreim:
    def __init__(self, lst, n):
        self.n = n
        self.lst = lst
        self.current = 0
    def __iter__(self):
        return self
    def next(self):
if self.lst:
        self.current=(self.current+self.n-1)%len(self.lst)
            return self.lst.pop(self.current)
        else:
            raise StopIteration
    print (range(1, 11))
    for i in Zahlreim(range(1, 11), 5):
        print (i)
```

Programma şeýle berer:

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
  5 10 6 2 9 8 1 4 7 3
```

Häzirki wagtda iteratorlaryň ähmiýeti gitdigiçe artýar we olar barada funksional programmirleme Sapaksynda aýdylyp geçildi.

Assosiasiýa

Eger-de agregasiýa ýagdaýynda öräп aýdyň “HAS-A” (“EÝE”) gatnaşyk bar bolsa:

```
lst = [1,2,3]
if l in lst
…
```

onda assosiasiýa ýagdaýynda “USE-A” (“ULANÝAR”) gatnaşygy bar. Bu gatnaşyk synplaryň arasyndaky baglanyşygy görkezýär.

Python dilinde agregasiýa bilen assosiasiýanyň arasynda kän uly çäk goýulanok, sebäbi agregasiýada obýektler huşda konteýnere niýetlenen ýerde saklanmaýarlar (diňe salgylanmalar saklanýarlar).

Obýektler hem bir-birlerine salgylanyp bilýärler. Bu ýagdaýda aýlaw salgylanmalar döreýär, olar tertipli ulanylmasa bökdençlik döräp biler.

Gowşak salgylanmalar

Obýektleriň assosiasiýasy bökdençsiz işlemegi üçin gowşak salgylanmalar ulanylýar. Programmada salgylanmalar obýekti ýok etmek üçin päsgel bermeli däl, şonuň üçin gowşak salgylanmalaryň mehanizmi döredilen. Şeýle salgylanma obýekte bolan salgylanmalar sanalanda hasaba alynmaýar, diýmek, obýekt soňky “güýçli” salgylanma bilen ýok edilýär.

Gowşak salgylanmalar bilen işlemek üçin `weakref` modul ulanylýar. Onuň işleýşiniň esasy prinsipleri indiki mysaldan belli bolar:

```
>>> import weakref
>>>
>>> class MyClass(object):
   def _str_(self):
       return "MyClass"


>>> s = MyClass()#synpyň nusgasy döredilýär
>>> print(s) #proksi-obýekt başlangyç ýaly işleýär
<__main__.MyClass object at 0x02308F50>
>>> s1=weakref.proxy(s)#proksi-obýekt döredilýär
>>> print(s1)
<__main__.MyClass object at 0x02308F50>
>>> ss=weakref.ref(s) #gowşak salgylanma döredilýär
```

```
>>> print(ss())
<__main__.MyClass object at 0x02308F50>
>>> del s
>>> print(ss())
None
>>> print(s1)
Traceback (most recent call last):
  File "<pyshell#88>", line 1, in <module>
    print(s1)
ReferenceError: weakly-referenced object no longer
exists
```

Statiki usul

Kä halatlarda synpyň nusgasyna däl-de özüne degişli usullary ulanmak gerek bolýar. Bular ýaly ýagdaýlarda statiki usuly beýan edip bolar. Python diliniň öňki wersiýalarynda statiki usuly şeýle kesgitlemeli bolýardy:

```
class A(object):
def name():
return A.__name__
name = staticmethod(name)
print (A.name())
a = A()
print (a.name())
>>>
A
A
```

Statiki usula synpyň nusgasy bilen parametr berlenok. Python diliniň täze wersiýalarynda dekorator atly täze sintaksis döredildi:

```
class A(object):
    @staticmethod
    def name():
        return A.__name__
    name = staticmethod(name)
print (A.name())
a = A()
print (a.name())
```

Synpyň usuly

Eger-de statiki usula meňzeş usullar C++ we Java-da bar bolsa, onda synpyň usuly Python dilinde synplar obýektlerdigine esaslanan. Statiki usuldan tapawutlylykda synpyň usulyna birinji parametr hökmünde obýekt-synp berilýär. self –iň ýerine cls ulanmak kabul edilen.

Synpyň usulyny ulanylýan mysaly nltk (Natural Language Toolkit, adaty dil üçin gurallaryň toplumy) paketiň tree modulynda görüp bolar. Aşakda Tree synpy kesgitlemegiň bölegi getirilen.

```
class Tree:
    # ...
    def convert(cls, val):
        if isinstance(val, Tree):
            children = [cls.convert(child) for child in
        val]
            return cls(val.node, children)
        else:
            return val
    convert = classmethod(convert)
```

Ulanmagyň mysaly:

```
>>> # Tree synpyň nusgasyna tree özgertmek
>>> tree = Tree.convert(tree)
>>> # " " " " " ParentedTree
>>> tree = ParentedTree.convert(tree)
>>> # " " " " " MultiParentedTree
>>> tree = MultiParentedTree.convert(tree)
```

Synpyň usuly esasan synplar bilen bagly bolan hereketleri suratlandyrmaga ýardam berýär.

Metasynplar

Synplaryň arasyndaky gatnaşyklaryň ýene biri synp-metasynp gatnaşygy bolup durýar. Metasynpy obýekte gönükdirilen programmirlemäniň iň ýokary derejesi diýip hasaplap bolar, ýöne Python dilinde hususy metasynplary hem döredip bolýar.

Mysalyň üsti bilen muňa seredip geçeliň:

```
def cls_factory_f(func):
class X(object):
pass
setattr(X, func.__name__, func)
  return X
```

Ulanylanda şeýle bolar:

```
def my_method(self):
print "self:", self
My_Class = cls_factory_f(my_method)
```

```
my_object = My_Class()
  my_object.my_method()
```

Bu mysalda `cls_factory( )` funksiýa ýeke-täk usuly bilen synpy berýär. Bu synpdan nusgalary alyp bolýar, nusgalardan bolsa `my_method` usuly çagyryp bolýar.

Indi bolsa synpy gurmak maksat edineliň, onuň nusgalary bolsa synplar bolar. Synplary döredýän synpa metasynp diýilýär.

Python dilinde type synpy bar, ol bolsa iş ýüzünde metasynp bolup durýar. Onuň kömegi bilen şeýdip synp gurup bolýar:

```
def my_method(self):
print ("self:", self)
My_Class=type('My_Class',(object,),{'my_method': my_method})
```

`type` synpyň birinji parametri hökmünde synpyň ady, ikinji parametr hökmünde berlen synp üçin esasy synplar, üçünji – atributlar berilýär.

Hususy metasynpy düzmek üçin type metasynp ulanylsa gowy bolar:

```
  >>> class My_Type(type):
     def __new__(cls, name, bases, dict):
         print("synp uchin hushda yer tutmak", name)
         return type.__new__(cls,name,bases,dict)
     def __init__(cls, name, bases, dict):
         print("synpyn inisializasiyasy", name)
         return super(My_Type,cls).__init__(cls,name,
bases,dict)


  >>> my = My_Type("X", (), {})
  synp uchin hushda yer tutmak X
  synpyn inisializasiyasy X
```

Bu mysalda synp döredilende goşulanok. Ýöne __new__( ) we __init__( ) –de ýerine ýetirilende döredilýän synp dolylygyna dolandyrlýar.

Bellik:

Bir zady bellemek gerek, metasynplarda usullaryň birinji argumenti self däl-de cls. Diýmek, programmistiň işleýän nusgasy ýöne bir obýekt däl-de synp bolup durýar.

Multiusullar

Käbir obýekte gönükdirilen usullar Python diline ýa-da onuň standart kitaphanasyna girmeýär. Aşakda multiusullara seredilip geçiler – bu usul bir

wagtda birnäçe dürli synplaryň obýektlerini birleşdirýär. Mysal üçin, dürli görnüşli iki sany goşmak üçin multiusullar talap edilýär:

```
>>> import operator
>>> operator .add(1,2)
3
>>> operator .add(1.0, 2)
3.0
>>> operator .add(1, 2.0)
3.0
>>> operator .add(1, 1+2j)
(2+2j)
>>> operator .add(1+2j, 1)
(2+2j)
```

Bu mysalda operator .add özüni multiusul ýaly alyp barýar we parametrleriň dürli kombinasiýalary üçin dürli amallary ýerine ýetirýär.

Hususy multiusullary döretmek üçin Multimethod (ýazary Neel Krishnaswami) modulyndan peýdalanyp bolar, ony bolsa Internetden tapmak aňsat. Indiki mysalda hususy multiusulyň gurluşy görkezilen:

```
from Multimethod import Method, Generic,
AmbiguousMethodError
# классы, для которых будет определен мультиметод
class A: pass
class B(A): pass
# функции мультиметода
def m1(a, b): return 'AA'
def m2(a, b): return 'AB'
def m3(a, b): return 'BA'
# определение мультиметода (без одной функции)
g = Generic()
g.add_method(Method((A, A), m1))
g.add_method(Method((A, B), m2))
g.add_method(Method((B, A), m3))
# применение мультиметода
try:
print ('Типы аргументов:', 'Результат')
print ('A, A:', g(A(), A()))
    print ('A, B:', g(A(), B()))
    print ('B, A:', g(B(), A()))
    print ('B, B:', g(B(), B()))
except AmbiguousMethodError:
    print ('Неоднозначный выбор метода')
```

Durnukly obýektler.

Obýektler döreden programmadan tapawutlylykda has köp ýaşamaklary üçin baýtlaryň yzygiderliligi görnüşde görkezilen mehanizm gerek. Ikinji Sapakda pickle modulyna seredilip geçilipdi, ol bolsa obýektleri seriýalaşdyrmaga ýardam berýär.

Bu ýerde synplaryň obýektlere uzak wagtlyk ýokary hilli bolmagyna ýardam berýändigi görkeziler. Aşakdaky usullar synpda kesgitlenende pickle modulyň işini dolandyrmaga ýardam berýär.

__getinitargs__()	Obýekt döredilende __init__( ) usulyň girişinde argumentleri çykarmaly.
__getstate__()	Obýektiň ýagdaýyny görkezýän sözlügi çykarmaly. Eger bu usul synpda kesgitlenen bolsa, onda her obýektde bar bolan __dict__ atributy ulanylýar.
__setstate__(state)	Obýektiň öň saklanan state düzgüni dikeltmeli.

Indiki mysalda CC synp pickle modulyň kömegi bilen öz nusgalaryny dolandyrýar:

```
from time import time, gmtime
import copy
class CC:
    def __init__(self, created=time()):
        self.created = created
        self.created_gmtime = gmtime(created)
        self._copied = 1
        print (id(self), "init", created)
    def __getinitargs__(self):
        print (id(self), "getinitargs", self.created)
        return (self.created,)
    def __getstate__(self):
        print (id(self), "getstate", self.created)
        return {'_copied': self._copied}
    def __setstate__(self, dict):
        print (id(self), "setstate", dict)
        self._copied = dict['_copied'] + 1
    def __repr__(self):
        return"%sobj:%s%s%s"%(id(self), self._copied,
                    self.created, self.created_gmtime)
a = CC()
print (a)
b = copy.deepcopy(a)
print (b)
```

Netijede şeýle alynar:

```
  36622096 init 1407948155.455699
  36622096 obj: 1 1407948155.455699
time.struct_time(tm_year=2014, tm_mon=8, tm_mday=13,
tm_hour=16, tm_min=42, tm_sec=35, tm_wday=2,
tm_yday=225, tm_isdst=0)
  36622096 getstate 1407948155.455699
  35864848 setstate {'_copied': 1}
```

Obýektiň ýagdaýy üç atributdan ybarat: `created`, `created_gmtime`, `copied`. Olaryň birinjisi konsturktora parametrleriň iberilmegi bilen dikeldilip biliner. Ikinjisi – konstruktorda birinjiniň esasynda hasaplanyp biliner. Üçünji atribut bolsa synpyň interfeýsine girmeýär, şonuň üçin ol diňe `getstate/setstate` mehanizmiň üsti bilen iberilip biliner.

Obýektleri saklamak üçin diňe bir pickle.dump/pickle.load ýa-da shelve ýaly saklanyş mehanizmler ulanylmaýar. Python diliniň seriýalaşdyrylan obýektleri ýörite ýerlerde (mysal üçin `ZODB`) saklap hem bolýar.

Şeýle-de bu ýagdaý torlar boýunça iberilýän maglumatlara hem degişli. Eger-de sada obýektleri (setir ýa-da san bolsa) HTTP, XML-RPC, SOAP we ş.m. üsti bilen göni ugradyp bolýan bolsa, onda erkin obýektleri ilki ugradylýan tarapda gaplamaly we alynýan tarapda açmaly.

OGP-niň kem taraplary

Obýekte gönükdirilen çemeleşmeler häzirki wagtda öňde baryjy hasaplanýar. Ýöne ol diňe obýekt çemeleşmeler golaý bolan uly taslamalarda görünýär: grafiki interfeýsiň düzülişi, ulgamlaryň modelirlenmegi we ş.m.

Obýekt programmalaryň çeýeligi ýaly mesele hem öňde goýulýar. Täze usul döredilse proseduraly çemeleşmede aýratyn prosedura ýazylýar, ondaky her algoritmiň şahasynda maglumatyň şol görnüşi işlenilýär (ýagny, kodyň diňe bir ýeri redaktirlenýär). Obýekte gönükdirilen çemeleşmede bolsa her synpa täze usul girizip üýtgetmeli bolýar (ýagny birnäçe ýerlerde üýtgeşmeleri girizmeli). Ýöne OGP-niň utuşly ýeri – maglumatyň täze görnüşiniň girizilmegi: berlen görnüş üçin ähli usullar ýazylanda, üýtgeşmeler diňe bir ýerde bolup geçýär. Goý A, B, C synplar we a, b, c usullar berlen bolsun:

```
# OGP-de
class A:
    def a(): ...
    def b(): ...
    def c(): ...

class B:
    def a(): ...
    def b(): ...
    def c(): ...
```

```
class C:
    def a(): ...
    def b(): ...
    def c(): ...

# proseduraly çemeleşme
def a(x):
    if type(x) is A: ...
    if type(x) is B: ...
    if type(x) is C: ...

def b(x):
    if type(x) is A: ...
    if type(x) is B: ...
    if type(x) is C: ...

def c(x):
    if type(x) is A: ...
    if type(x) is B: ...
    if type(x) is C: ...
```

Obýektiň täze görnüşi girizilende OG-programmada diňe bir modul üýtgeýär, proseduralyda bolsa – ähli proseduralar:

```
#OGP-de
class D:
    def a(): ...
    def b(): ...
    def c(): ...

#proseduraly çemeleşmede
def a(x):
    if type(x) is A: ...
    if type(x) is B: ...
    if type(x) is C: ...
    if type(x) is D: ...

def b(x):
    if type(x) is A: ...
    if type(x) is B: ...
    if type(x) is C: ...
    if type(x) is D: ...

def c(x):
```

```
    if type(x) is A: ...
    if type(x) is B: ...
    if type(x) is C: ...
    if type(x) is D: ...
```

Hem-de tersine, indi täze usul goşmaly. Proseduraly çemeleşmede bary-ýogy täze prosedura ýazylýar, obýekt çemeleşmede bolsa ähli synplary üýtgetmeli bolýar:

```
#proseduraly çemeleşme
def d(x):
    if type(x) is A: ...
    if type(x) is B: ...
    if type(x) is C: ...

#OGP-de
class A:
    def a(): ...
    def b(): ...
    def c(): ...
    def d(): ...
class B:
    def a(): ...
    def b(): ...
    def c(): ...
    def d(): ...
class C:
    def a(): ...
    def b(): ...
    def c(): ...
    def d(): ...
```

Ýokarda getirilen mysallar Python diliniň standart kitaphanasynda aňladylan, diýmek, ol ýerde hem funksiýalar hem-de synplar ulanylýar.

Netije.

Bu Sapakda örän köp adalgalar ulanyldy. Ähli esasy düşünjeleriň üstünde durulyp geçildi: obýekt, görnüş, synp we obýektleriň arasyndaky gatnaşyklaryň görnüşi (IS-A, HAS-A, USE-A). Mundan başga-da obýekte gönükdirilen programmirlemede inkapsullaşmanyň we polimorfizmiň nämedigine seredilip geçildi. OGP-niň hem gowy taraplary hem-de kemçilikleri görkezildi.

5. SANLY ALGORITMLER. MATRISA HASAPLAMALARY

Numeric Python – köpölçegli massiwli hasaplamalar üçin köp sanly priloženiýelere birnäçe modullar. Numeric moduly Python diline MatLap, Octave (MatLab meňzeş), APL, J, S+, IDL ýaly paketleriň we ulgamlaryň mümkinçiliklerini girizýär. Bir zady bellesek bolar, ýagny Python dilindäki käbir sintaksiki mümkinçilikler (hususanda kesimleri ulanmak bilen baglanyşykly) ýörite Numeric üçin işläp düzülen.

Numeric Python aşakdakylar üçin serişdesi bar:

- LinearAlgebra matrisa hasaplamalary;
- FFT tiz Furýe özgermesi;
- MA ýetmeýän eksperimental maglumatlar bilen işlemek;
- RNG statiki modelirleme;
- MatLab programmanyň esasy funksiýalarynyň emulýasiýasy.

Numeric moduly

Numeric moduly massiw-görnüşi kesgitleýär we özünde massiwleriň üstündäki amallar üçin köp sanly funksiýalary saklaýar. Massiw – bu indeksler boýunça elýeter bolan birjynsly elementleriň toplumy. Numeric modulyň massiwleri köpölçegli bolup biler.

Massiw döretmek

Massiw döretmek üçin massiwiň düzümini we görnüşini görkezýän `array( )` funksiýany ulanyp bolar. `array( )` funksiýa, eger onuň argumenti massiw bolsa nusga döredýär. `asarray( )` funksiýa edil `array( )` funksiýa meňzeş, ýöne onuň argumenti massiw bolsa-da nusgasyny döretmeýär:

```
>>> from Numeric import *
>>> print array([[1, 2], [3, 4], [5, 6]])
[[1 2]
 [3 4]
 [5 6]]
>>> print array([[1, 2, 3], [4, 5, 6]], Float)
[[ 1. 2. 3.]
 [ 4. 5. 6.]]
>>> print array([78, 85, 77, 69, 82, 73, 67], 'c')
[N U M E R I C]
```

Massiwiň elementleri hökmünde indiki görnuşleri ulanyp bolar: `Int8-Int32`,`UnsignedInt8-UnsignedInt32`, `Float8-Float64`, `Complex8-Complex64` we`PyObject`. Bu ýerde 8, 32 we 64 sanlar ululygy saklamak üçin gerek bolan bitleriň mukdaryny görkezýärler. Int, UnsignedInt,

Float we Complex görnüşler berlen platformadaky bahalaryň iň ulusyna gabat gelýär. Şeýle-de massiwde islendik obýekte bolan salgylanmany saklap bolýar.

Massiwiň görnüşini shape atributyň kömegi bilen üýtgedip bolýar:

```
>>> from Numeric import *
>>> a = array(range(15), Int)
>>> print a.shape
(15,)
>>> print a
[ 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14]
>>> a.shape = (3, 5)
>>> print a.shape
(3, 5)
>>> print a
[[ 0 1 2 3 4]
[ 5 6 7 8 9]
[10 11 12 13 14]]
```

Massiwleriň usullary

Massiwiň gerekli görnüşini `Numeric.reshape( )` funksiýanyň kömegi bilen berip bolýar. Bu funksiýa yzygiderlikden gerekli görnüşdäki obýekt-massiwi döredýär:

```
>>> import Numeric
>>> print Numeric.reshape("maşgalanyň", (5, -1))
[[m a]
 [ş g]
 [a l]
 [a n]
 [y ň]]
```

Bu mysalda -1 – degişli bahany hasaplap bolýandygyny görkezýär. Massiwiň elementleriniň umumy sany belli (10), şonuň üçin bir ölçeg boýunça uzynlygy bermek hökman däl.

`flat` atributyň üsti bilen massiwiň birölçegliligini göz-öňüne getirip bolýar:

```
>>> a = array([[1, 2], [3, 4]])
>>> b = a.flat
>>> b
array([1, 2, 3, 4])
>>> b[0] = 9
>>> b
array([9, 2, 3, 4])
>>> a
array([[9, 2],
```

```
  [3, 4]])
```

Bu ýerde şol bir massiwiň dürli görnüşi berlen, şonuň üçin onuň elementlerine baha bermeklik başlangyç massiwiň üýtgemegine getirýär.

`Numeric.resize( )` funksiýa `Numeric.reshape( )` funksiýa meňzeş, ýöne elementleriň sanyny düzüp bilýär:

```
>>> print Numeric.resize("NUMERIC", (3, 2))
[[N U]
 [M E]
 [R I]]
>>> print Numeric.resize("NUMERIC", (3, 4))
[[N U M E]
 [R I C N]
 [U M E R]]
```

`Numeric.zeros()` funksiýa diňe nollardan massiw döredýär, `Numeric.ones()` funksiýa bolsa diňe birliklerden. Birlik matrisany `Numeric.identity(n)` funksiýanyň kömegi bilen alyp bolar:

```
>>> print Numeric.zeros((2,3))
[[0 0 0]
 [0 0 0]]
>>> print Numeric.ones((2,3))
[[1 1 1]
 [1 1 1]]
>>> print Numeric.identity(4)
[[1 0 0 0]
 [0 1 0 0]
 [0 0 1 0]
 [0 0 0 1]]
```

Massiwleri göçürip almak üçin `copy()` usulyny ulanyp bolar:

```
>>> import Numeric
>>> a = Numeric.arrayrange(9)
>>> a.shape = (3, 3)
>>> print a
[[0 1 2]
 [3 4 5]
 [6 7 8]]
>>> a1 = a.copy()
>>> a1[0, 1] = -1 # göçürmegiň üstünde amal
>>> print a
[[0 1 2]
```

```
[3 4 5]
[6 7 8]]
```

Massiwi tersine sanawa `tolist( )` usulyň kömegi bilen öwürip bolar:

```
>>> a.tolist()
[[0, 1, 2], [3, 4, 5], [6, 7, 8]]
```

Kesimler

Numeric obýekt-massiwler kesimi bellemegiň giňeldilen sintaksisini ulanýarlar. Indiki mysalda kesimleri ýazmagyň dürli görnüşleri getirilen. `Numeric.arrayrange()` funksiýa massiwler üçin `range()` funksiýa meňzeş.

```
>>> import Numeric
>>> a = Numeric.arrayrange(24) + 1
>>> a.shape = (4, 6)
>>> print a # başlangyç massiw
[[ 1 2 3 4 5 6]
 [ 7 8 9 10 11 12]
 [13 14 15 16 17 18]
 [19 20 21 22 23 24]]
>>> print a[1,2] # 1,2 element
9
>>> print a[1,:] # 1 setir
[ 7 8 9 10 11 12]
>>> print a[1] # 1 setir
[ 7 8 9 10 11 12]
>>> print a[:,1] # 1 sütün
[ 2 8 14 20]
>>> print a[-2,:]
[13 14 15 16 17 18]
>>> print a[0:2,1:3] # 2x2 penjire
[[2 3]
[8 9]]
>>> print a[1,::3] # 1-nji setiriň her üçünji elementi
[ 7 10]
>>> print a[:,::-1] # ters tertipde setiriň elementleri
[[ 6 5 4 3 2 1]
 [12 11 10 9 8 7]
 [18 17 16 15 14 13]
 [24 23 22 21 20 19]]
```

Kesim massiwi göçürip alanok, ol diňe massiwiň belli bir bölegini alýar. Indiki mysalda 1-nji setiriň her üçünji elementi 0-a çalşyrylýar:

```
>>> a[1,::3] = Numeric.array([0,0])
>>> print a
[[ 1 2 3 4 5 6]
 [ 0 8 9 0 11 12]
 [13 14 15 16 17 18]
 [19 20 21 22 23 24]]
```

Aşakdaky mysallarda seýrek ulanylýan sintaktiki gurluş getirilen: köpnokatly kesim (Ellipsis).

```
>>> import Numeric
>>> a = Numeric.arrayrange(24) + 1
>>> a.shape = (2,2,2,3)
>>> print a
[[[[ 1 2 3]
[ 4 5 6]]
[[ 7 8 9]
[10 11 12]]]
[[[13 14 15]
[16 17 18]]
[[19 20 21]
[22 23 24]]]]
>>> print a[0,...] # 0-njy blok
[[[ 1 2 3]
[ 4 5 6]]
[[ 7 8 9]
[10 11 12]]]
>>> print a[0,:,:,0]
[[ 1 4]
[ 7 10]]
>>> print a[0,...,0]
[[ 1 4]
[ 7 10]]
```

Uniwersal funksiýalar

Modul Numeric massiwiň elementlerine ulanmak üçin funksiýalaryň toplumyny kesgitleýär. Bu funksiýalar diňe bir massiwlere däl, eýsem yzygiderliklere hem ulanylýar. Netijede massiwler alynýar.

Funksiýa	Beýany
`add(x, y)`, `subtract(x, y)`	Goşmak we aýyrmak.
`multiply(x, y)`, `divide(x, y)`	Köpeltmek we bölmek.
`remainder(x, y)`,	Bölünmekden galyndyny almak.

`fmod(x, y)`	
`power(x)`	Derejä götermek.
`sqrt(x)`	Kwadrat kökden çykarmak.
`negative(x)`, `absolute(x)`, `fabs(x)`	Alamaty üýtgetmek we absolýut baha.
`ceil(x), floor(x)`	Argumentden uly ýa-da kiçi iň kiçi (iň uly) bitin san.
`hypot(x, y)`	Gipotenuzanyň uzynlygy.
`sin(x), cos(x), tan(x)`	Trigonometrik funksiýalar.
`arcsin(x), arccos(x), arctan(x)`	Ters trigonometrik funksiýalar.
`arctan2(x, y)`	Hususy argumentden arktangens.
`sinh(x), cosh(x), tanh(x)`	Giperbolik funksiýalar.
`arcsinh(x), arccosh(x), arctanh(x)`	Ters giperbolik funksiýalar.
`exp(x)`	Eksponenta (e^x).
`log(x), log10(x)`	Natural we onluk logarifmler.
`maximum(x, y), minimum(x, y)`	Maksimum we minimum.
`conjugate(x)`	Özara baglylyk (kompleks sanlar üçin).
`equal(x, y), not_equal(x, y)`	Deň, deň däl.
`greater(x, y), greater_equal(x, y)`	Uly, uly ýa-da deň.
`less(x, y), less_equal(x, y)`	Kiçi, kiçi ýa-da deň.
`logical_and(x, y),logical_or(x, y)`	Logiki We, Ýa-da.
`logical_xor(x, y)`	Logiki kadadan çykma Ýa-da.
`logical_not(x)`	Logiki Ýok.
`bitwise_and(x, y), bitwise_or(x, y)`	Bitleýin We, Ýa-da.
`bitwise_xor(x, y)`	Bitleýin kadadan çykma Ýa-da.
`invert(x)`	Bitleýin inwersiýa.
`left_shift(x, n), right_shift(x, n)`	n bit gezek bitleýin çepe ýa-da saga süýşmeler.

Ýokarda agzalan funksiýalar `ufunc` görnüşiň obýektleri bolup durýar we massiwleriň her elementi üçin ulanylýar. Bu funksiýalaryň ýörite usullary bar:

`accumulate()`	Netijäni jemlemek.

`outer()`	Daşky “jem”.
`reduce()`	Gysgaltmak.
`reduceat()`	Berlen nokatlarda gysgaltmak.

add( ) funksiýaly mysal bu funksiýanyň we onuň usullarynyň uniwersaldygyny görkezýär:

```
>>> from Numeric import add
>>> add([[1, 2], [3, 4]], [[1, 0], [0, 1]])
array([[2, 2],
    [3, 5]])
>>> add([[1, 2], [3, 4]], [1, 0])
array([[2, 2],
    [4, 4]])
>>> add([[1, 2], [3, 4]], 1)
array([[2, 3],
    [4, 5]])
>>> add.reduce([1, 2, 3, 4]) # ýagny 1+2+3+4
10
>>> add.reduce([[1, 2], [3, 4]], 0) # ýagny [1+3 2+4]
array([4, 6])
>>> add.reduce([[1, 2], [3, 4]], 1) # ýagny [1+2 3+4]
array([3, 7])
>>> add.accumulate([1,2,3,4]) #ýagny[1 1+2 1+2+3 1+2+3+4]
array([ 1, 3, 6, 10])
>>> add.reduceat(range(10),[0,3,6]) #ýagny[0+1+2 3+4+5
6+7+8+9]
array([ 3, 12, 30])
>>> add.outer([1,2], [3,4]) # ýagny [[1+3 1+4] [2+3
2+4]]
array([[4, 5],
  [5, 6]])
```

Uniwresal funksiýalarda bir ýa-da iki gerekli parametrlerden başga-da ýene-de bir argument, ýagny funksiýanyň netijesini kabul etmek üçin berilip biliner. Üçünji argumentiň görnüşi netijäniň görnüşine gabat gelmeli. Mysal üçin, `sqrt( )` funksiýa bitin sandan bolsa-da `Float` görnüşine eýe.

```
>>> from Numeric import array, sqrt, Float
>>> a = array([0, 1, 2])
>>> r = array([0, 0, 0], Float)
>>> sqrt(a, r)
array([ 0. , 1. , 1.41421356])
>>> print r
  [ 0. 1. 1.41421356]
```

Numeric modulynyň funksiýalary

`Numeric` modulyň aşakdaky funksiýalary köp ulanylýan funksiýalaryň we usullaryň düzümleri bolup durýar:

Funksiýa	Funksiýa meňzeş
`sum(a, axis)`	`add.reduce(a, axis)`
`cumsum(a, axis)`	`add.accumulate(a, axis)`
`product(a, axis)`	`multiply.reduce(a, axis)`
`cumproduct(a, axis)`	`multiply.accumulate(a, axis)`
`alltrue(a, axis)`	`logical_and.reduce(a, axis)`
`sometrue(a, axis)`	`logical_or.reduce(a, axis)`

Bellik:
`axis` parametr ölçegi görkezýär.

Massiwler bilen işlemek üçin funksiýalar

Funksiýalar örän köp bolany üçin, olaryň diňe ikisine seredilip geçiler, galany tablisada getiriler.

Numeric.take( ) funksiýa

Bu funksiýa massiwiň berlen indeksler boýunça bölegini almaga ýardam berýär. Eger-de üçünji argument görkezilmese, ol nola deň.

```
>>> import Numeric
>>> a = Numeric.reshape(Numeric.arrayrange(25), (5, 5))
>>> print a
[[ 0 1 2 3 4]
 [ 5 6 7 8 9]
 [10 11 12 13 14]
 [15 16 17 18 19]
 [20 21 22 23 24]]
>>> print Numeric.take(a, [1], 0)
[ [5 6 7 8 9]]
>>> print Numeric.take(a, [1], 1)
[[ 1]
 [ 6]
 [11]
 [16]
 [21]]
>>> print Numeric.take(a, [[1,2],[3,4]])
[[[ 5 6 7 8 9]
```

```
  [10 11 12 13 14]]
 [[15 16 17 18 19]
  [20 21 22 23 24]]]
```

Kesimden tapawutlylykda, Numeric.take( ) funksiýa massiwiň ölçegini saklaýar, elbetde, eger-de berlen indeksleriň gurluşy birölçegli bolsa. Numeric.take(a, [[1,2], [3,4]]) netije indeksler boýunça alnan bölekler indeksleriň gurluşy bilen massiwe ýerleşdirilýär, mysal üçin, 1-iň ýerine [5 6 7 8 9] bolar, ýa-da 2-iň ýerine [10 11 12 13 14] we başg.

Numeric.diagonal( ) we Numeric.trace( ) funksiýalar.

`Numeric.diagonal( )` funksiýa matrisanyň diagonalyny berýär. Onuň aşakdaky argumentleri bar:

`a`	Başlangyç massiw.
`offset`	Esasy diagonaldan saga süýşmek.
`axis1`	Diagonalyň alýan ölçegleriniň birinjisi.
`axis2`	Diagonalyň alynýan tekizliginde döreýän ikinji ölçeg. Başlangyç baha axis2=1.

Numeric.trace( ) funksiýanyň şol bir argumenti bar, ýöne ol diagonalyň elementlerini jemleýär. Aşakdaky mysalda bu funksiýalaryň ikisem görkezilen:

```
>>> import Numeric
>>> a = Numeric.reshape(Numeric.arrayrange(16), (4, 4))
>>> print a
[[ 0 1 2 3]
 [ 4 5 6 7]
 [ 8 9 10 11]
 [12 13 14 15]]
>>> for i in range(-3, 4):
... print "Sum", Numeric.diagonal(a, i), "=",
Numeric.trace(a, i)
...
Sum [12] = 12
Sum [ 8 13] = 21
Sum [ 4 9 14] = 27
Sum [ 0 5 10 15] = 30
Sum [ 1 6 11] = 18
Sum [2 7] = 9
Sum [3] = 3
```

Numeric.choose( ) funksiýa.

Bu funksiýa 0-dan n-e çenli bitin sanly bir massiwi berlen massiwleriň birinden bahalary almak üçin ulanýar.

```
>>> a = Numeric.identity(4)
>>> b0 = Numeric.reshape(Numeric.arrayrange(16),(4, 4))
>>> b1 = -Numeric.reshape(Numeric.arrayrange(16),(4,4))
>>> print Numeric.choose(a, (b0, b1))
[[ 0 1 2 3]
 [ 4 -5 6 7]
 [ 8 9 -10 11]
 [ 12 13 14 -15]]
```

Numeric modulyň funksiýalary

Indiki tablisada `Numeric` modulyň funksiýalary getirilen.

Funksiýa we onuň argumentleri	Funksiýanyň ýerine ýetirýän işi
`allclose(a, b[, eps[, A]])`	a we b bahalary berlen göräli eps we absolýut A ýalňyşlyklar bilen deňeşdirmek. Bu ýerde eps=1.0e-1, A=1,0e-8.
`alltrue(a[, axis])`	a massiwiň tutuş axis oky boýunça logiki We.
`argmax(a[, axis])`	Massiwde berlen axis ölçeg boýunça maksimal bahalaryň indeksi.
`argmin(a[, axis])`	Massiwde berlen axis ölçeg boýunça minimal bahalaryň indeksi.
`argsort(a[, axis])`	Saýlanan massiwleriň indeksleri, mysal üçin, `take(a,argsort(a, axis),axis)` saýlanan a massiwi berýär we `sort(a, axis)` gabat gelýär.
`array(a[, type])`	Berlen `type` görnüşiň a yzygiderliginiň esasynda massiw döretmek.
`arrayrange(start[, stop[, step[, type]]])`	Massiw üçin range( ) meňzeş.
`asarray(a[, type[, savespace]])`	array( ) meňzeş, ýöne eger-de a massiw bolsa, täze massiw döretmeýär.
`choose(a, (b0,...,bn))`	a massiwden indeksler boýunça alnan elementleriň esasynda massiw döredýär. Massiwleriň a, b1, …, bn görnüşleri gabat gelmeli.
`clip(a, a_min, a_max)`	a massiwiň bahalary a_min we a_max bahalaryň aralygynda ýerleşer ýaly kesýär.

`compress(cond, a[, axis])`	cond şerti kanagatlandyrýan (nol däl) a massiwiň elementlerinden massiw döredýär.
`concatenate(a[, axis])`	Berlen axis ölçeg boýunça iki massiwi birleşdirýär (konkatenasiýa)
`cross_correlate(a, b[, mode])`	Iki massiwiň özara korrelýasiýasy. mode parametr 0, 1 ýa-da 2 baha eýe bolup biler.
`cumsum(a[, axis])`	Aralyk netijeler boýunça jem.
`diagonal(a[, k[,// axis1[, axis2]]])`	axis1 we axis2 tekizliklerde a massiwiň k-nji diagonalyny almak.
`dot(a, b)`	Matrisanyň içinde massiwleriň köpeltmek hasyly. Kesgitleme boýunça: `innerproduct(a, swapaxes(b, -1, -2))`.
`dump(obj, file)`	file açylan faýl obýekte a massiwi ýazmak. Faýl binar düzgünde açyk bolmaly. Faýla birnäçe obýektleri yzygider ýazyp bolýar.
`dumps(obj)`	obj obýektiň ikilik görnüşdäki setiri.
`fromfunction(f, dims)`	`f(*tuple(indices(dims)))` üçin gysgaldylan görnüş.
`fromstring(s[, count[, type]])`	Setirde saklanýan binar maglumatlaryň esasynda massiwi döretmek.
`identity(n)`	(n,n) ölçegli iki ölçegli massiwi döredýär.
`indices(dims[, type])`	Berlen uzynlyk boýunça massiwi döredýär, mysal üçin, `indices([2, 2])` iki ölçegli massiwi `[1][[0, 1],[0, 1]]` berýär.
`innerproduct(a, b)`	Iki massiwiň köpeltmek hasyly. Amalyň dowamynda ýitirim bolýan elementler jübütme-jübüt köpeldilýär we jemlenýär.
`load(file)`	file faýldan massiwi okamak. Faýl binar düzgünde açyk bolmaly.
`loads(s)`	Setirde berlen binar ýagdaýa gabat gelýän obýekti berýär.
`nonzero(a)`	Birölçegli massiwiň nola deň bolmadyk elementleriň indekslerini berýär.
`outerproduct(a, b)`	a we b bahalaryň köpeltmek hasyly.
`product(a[, axis])`	a massiwiň axis ölçegi boýunça köpeltmek hasyly.
`put(a, indices, b)`	indices ähli indeksler üçin a[n]=b[n], massiwiň böleklerine baha bermek.
`putmask(a, mask, b)`	b-den a elementlere baha bermek.
`ravel(a)`	Massiwi birölçegli görnüşe öwürmek.
`repeat(a, n[, axis])`	axis ölçeg boýunça a massiwiň elementlerini n gezek gaýtalaýar.

`reshape(a, shape)`	Gerekli görnüşdäki massiwi döredýär. Başlangyç we täze massiwlerde elementleriň sany gabat gelmeli.
`resize(a, shape)`	Erkin shape görnüşli massiwi döredýär. Başlangyç massiwiň ölçegi möhüm däl.
`searchsorted(a, i)`	a massiwde her i element üçin ýer tapmak. a massiw birölçegli bolmaly. Netijede i massiwiň görnüşi bolmaly.
`shape(a)`	a massiwiň görnüşini berýär.
`sometrue(a[, axis])`	Tutuş axis ölçeg boýunça a massiwiň logiki Ýa-da bahasy
`sort(a[, axis])`	saýlamak.
`sum(a[, axis])`	Berlen ölçeg boýunça massiwiň elementlerini jemlemek.
`swapaxes(a, axis1, axis1)`	Ölçegleri çalyşmak.
`take(a, indices[, axis])`	axis ölçeg boýunça indices indeksleriň esasynda a massiwiň böleklerini saýlamak.
`trace(a[, k[, axis1[, axis2]]])`	Diagonalyň ugry boýunça elementleriň jemi, ýagny `add.reduce(diagonal(a, k, axis1, axis2))`
`where(cond, a1, a2)`	condşertiň esasyndaa1 wea2 massiwlerdenelementlerisaýlamak.
`zeros(shape[, type])`	Berlen type we shape görnüşlerde nollardan ybarat bolan massiwi döretmek.

LeanerAlgebra moduly

Bu modul çyzykly algebranyň algoritmlerini özünde saklaýar, hususanda, matrisanyň kesgitleýjisini tapmak, çyzykly deňlemeleriň ulgamyny çözmek, ters matrisa öwürmek, matrisanyň hususy bahalaryny we hususy wektorlaryny tapmak, matrisany köpeltmek agzalaryna dagytmak: Holeski, singulýar, iň kiçi kwadratlaryň usuly.

`LinearAlgebra.determinant( )` funksiýa matrisanyň kesgitleýjisini tapýar:

```
>>> import Numeric, LinearAlgebra
>>> print LinearAlgebra.determinant(
... Numeric.array([[1, -2],
... [1, 5]]))
  7
```

`LinearAlgebra.solve_linear_equations()` funksiýa ax=b görnüşli çyzykly deňlemäni berlen a we b argumentler boýunça çözýär:

```
>>> import Numeric, LinearAlgebra
>>> a = Numeric.array([[1.0, 2.0], [0.0, 1.0]])
>>> b = Numeric.array([1.2, 1.5])
>>> x = LinearAlgebra.solve_linear_equations(a, b)
>>> print "x =", x
x = [-1.8 1.5]
>>> print "Barlag:", Numeric.dot(a, x) - b
  Barlag: [ 0. 0.]
```

Haçan-da a matrisanyň nolunjy kesgitleýjisi bolsa, ulgamda ýeke-täk çözüw bolmaýar we LinearAlgebraError ýalňyşy çykýar:

```
>>> a = Numeric.array([[1.0, 2.0], [0.5, 1.0]])
>>> x = LinearAlgebra.solve_linear_equations(a, b)
Traceback (most recent call last):
File "<stdin>", line 1, in ?
File "/usr/local/lib/python2.3/site-
packages/Numeric/LinearAlgebra.py", line
98,
in solve_linear_equations raise LinAlgError, 'Singular
matrix'
  LinearAlgebra.LinAlgError: Singular matrix
```

`LinearAlgebra.inverse()` funksiýa ters matrisany çykýar. Ýöne bu funksiýanyň kömegi bilen çyzykly deňlemeleri ters matrisa köpeltmek bilen işlemeli däl, sebäbi ol `LinearAlgebra.solve_linear_equations()` üsti bilen kesgitlenýär:

```
def inverse(a):
        return solve_linear_equations(a,
                Numeric.identity(a.shape[0]))
```

`LinearAlgebra.eigenvalues()` funksiýa matrisanyň hususy bahalaryny tapýar, `LinearAlgebra.eigenvectors()` – hem hususy bahalaryny hem-de hususy wektorlary:

```
>>> from Numeric import array, dot
>>> from LinearAlgebra import eigenvalues, eigenvectors
>>> a = array([[-5, 2], [2, -7]])
>>> lmd = eigenvalues(a)
>>> print "Hususy bahalar:", lmd
Hususy bahalar: [-3.76393202 -8.23606798]
```

```
>>> (lmd, v) = eigenvectors(a)
>>> print "Hususy wektorlar:"
Hususy wektorlar:
>>> print v
[[ 0.85065081 0.52573111]
[-0.52573111 0.85065081]]
>>> print "Barlag:", dot(a, v[0]) - v[0] * lmd[0]
  Barlag: [ -4.44089210e-16 2.22044605e-16]
```

Bu ýerden görnüşi ýaly, tožděstwo ýokary takyklyk bilen ýerine ýetirilýär: hususy bahalar we hususy wektorlar dogry tapyldy.

RandomArray moduly

Bu modulda tötän sanlaryň massiwleriniň generasiýasy üçin funksiýalar toplanan. Olary matematiki modelirleme üçin ulanyp bolar.

RandomArray.random() funksiýa (0,1) aralykda deňölçegli ýerleşdirilen tötän sanlardan massiwleri döredýär:

```
>>> import RandomArray
>>> print RandomArray.random(10)
[ 0.28374212 0.19260929 0.07045474 0.30547682
0.10842083 0.14049676
    0.01347435 0.37043894 0.47362471 0.37673479]
>>> print RandomArray.random([3,3])
[[ 0.53493741 0.44636754 0.20466961]
[ 0.8911635 0.03570878 0.00965272]
[ 0.78490953 0.20674807 0.23657821]]
```

RandomArray.randint() funksiýa berlen aralykdaky we görnüşdäki deňölçegli ýerleşdirilen sanlardan massiwi almak üçin ulanylýar:

```
>>> print RandomArray.randint(1, 10, [10])
[8 1 9 9 7 5 2 5 3 2]
>>> print RandomArray.randint(1, 10, [10])
[2 2 5 5 7 7 3 4 3 7]
```

RandomArray.permutation() kömegi bilen tötän ýerleşdirmeleri hem alyp bolar:

```
>>> print RandomArray.permutation(6)
[4 0 1 3 2 5]
>>> print RandomArray.permutation(6)
[1 2 0 3 5 4]
```

Berlen orta we standart gyşarmaly kadaly ýerleşdirilen ululyklaryň massiwini almak üçin beýleki ýerleşdirmeler hem elýeter:

```
>>> print RandomArray.normal(0, 1, 30)
[-1.0944078 1.24862444 0.20415567 -0.74283403
0.72461408 -0.57834256
0.30957144 0.8682853 1.10942173 -0.39661118 1.33383882
1.54818618
0.18814971 0.89728773 -0.86146659 0.0184834 -1.46222591
-0.78427434
1.09295738 -1.09731364 1.34913492 -0.75001568 -
0.11239344 2.73692131
-0.19881676    -0.49245331    1.54091263    -1.81212211
0.46522358 -0.08338884]
```

Aşakdaky tablisada beýleki ýerleşdirmeler üçin funksiýalar getirilen:

Funksiýa we onuň argumenti	Beýany
`F(dfn, dfd, shape=[])`	F-ýerleşdirme
`beta(a, b, shape=[])`	Beta-ýerleşdirme
`binomial(trials, p, shape=[])`	Binomial ýerleşdirme
`chi_square(df, shape=[])`	hi-kwadrat ýerleşdirme
`exponential(mean, shape=[])`	Eksponensial ýerleşdirme
`gamma(a, r, shape=[])`	Gamma-ýerleşdirme
`multivariate_normal(mean, cov, shape=[])`	Köpölçegli kadaly ýerleşdirme
`negative_binomial(trials, p, shape=[])`	Negatiw binomial ýerleşdirme
`noncentral_F(dfn, dfd, nconc, shape=[])`	Merkezi bolmadyk F-ýerleşdirme
`noncentral_chi_square(df, nconc, shape=[])`	Merkezi bolmadyk hi-kwadrat ýerleşdirme
`normal(mean, std, shape=[])`	Kadaly ýerleşdirme
`permutation(n)`	Tötän ýerleşdirme

`poisson(mean, shape=[])`	Puasson ýerleşdirme
`randint(min, max=None, shape=[])`	Tötän bitin ýerleşdirme
`random(shape=[])`	(0,1) aralykda deňölçeli ýerleşdirme
`random_integers(max, min=1, shape=[])`	Tötän bitin
`standard_normal(shape=[])`	Standart kadaly ýerleşdirme
`uniform(min, max, shape=[])`	Deňölçegli ýerleşdirme

Netije.

Bu Sapakda sanly hasaplamalar üçin modullaryň toplumyna seredilip geçildi. Numeric moduly köpölçegli massiwiň görnüşini we massiwler bilen işlemek üçin funksiýalaryň köplügini kesgitleýär. Şeýle-de çyzykly algebra üçin we dürli ýerleşdirmelerde tötän sanlaryň yzygiderliklerini gurnamagyň modullary getirilen.

6. TEKSTLERIŇ ÜSTÜNDEIŞLEMEK. UNICODE.

Tekstleriň üstündeişlemekdiýmektekst maglumatlary döretmek, gözlemek, derňemek, özgertmek diýmegi aňladýar.

Setirler.

Python dilinde setirler ýörite tekst maglumatlary işlemek üçin ýörite niýetlenen maglumatlaryň görnüşidir. Setir islendik uzynlygy alyp biler (ol diňe ýadyň ölçegi bilen çäklenip biler).

Python dilinde setiriň iki görnüşi bar: adaty setirler (b/aýtlaryň yzygiderligi) we Unicode-setirler (simwollaryň yzygiderligi). Unicode-setirlerde her simwol kompilýasiýa döwrüniň sazlanylşyna baglylykda ýatda 2 ýa-da 4 baýt ýeri alyp biler. Dört baýtly simwollar esasan gündogar diller üçin ulanylýar.

Bellik:

Python dilinde we onuň standart kitaphanasynda käbir kadadan çykmalardan başga ýagdaýlarda setirler we Unicode-setirler özara çalyşylýar. Hususy priloženiýelerde sygdyrmak üçin setirleriň iki görnüşiniň hem tipini barlamakdan gaça durmaly. Eger bu zerur bolsa, onda esasy tipe degişliligini `isinstance(s, basestring)` kömegi bilen barlap bolar.

Python programmalarda kodlamak

Python programmalarda Unicode-setirler interpretator tarapyndan dogry kabul edilmegi üçin programmanyň başynda kodlamany görkezmeli we birinji ýa-da ikinji setirde şeýle ýazmaly (Unix/Linux üçin):

```
# -*- coding: koi8-r -*-
```

ýa-da (Windows üçin):

```
# -*- coding: cp1251 -*-
```

Başga görnüşler hem bolup biler:

```
# -*- coding: latin-1 -*-
# -*- coding: utf-8 -*-
# -*- coding: mac-cyrillic -*-
# -*- coding: iso8859-5 -*-
```

Kodlamanyň doly sanawy:

```
>>> import encodings.aliases
>>> print (encodings.aliases.aliases)
{'iso_8859_6_1987': 'iso8859_6', 'iso_8859_8_1988':
'iso8859_8', 'iso_8859_3': 'iso8859_3', 'iso_8859_2':
'iso8859_2', 'iso_8859_5': 'iso8859_5', 'windows_1258':
'cp1258', 'iso_8859_7': 'iso8859_7', 'iso_8859_6':
```

```
'iso8859_6', 'iso_8859_9': 'iso8859_9', 'asmo_708':
'iso8859_6', 'hzgb': 'hz', 'iso_ir_127': 'iso8859_6',
'ksc5601': 'euc_kr', 'ms1361': 'johab', '1250':
'cp1250', '1251': 'cp1251', '1252': 'cp1252', '1253':
'cp1253', '1254': 'cp1254', '1255': 'cp1255', '1256':
'cp1256', '1257': 'cp1257', '1140': 'cp1140',
'maclatin2': 'mac_latin2', 'ebcdic_cp_nl': 'cp037',
'iso_646.irv_1991': 'ascii', 'iso_8859_4': 'iso8859_4',
'tis_620_2529_1': 'tis_620', 'arabic': 'iso8859_6',
'chinese': 'gb2312', 'ibm367': 'ascii', 'iso8859':
'latin_1', 'u8': 'utf_8', 'ks_x_1001': 'euc_kr',
'utf16': 'utf_16', '424': 'cp424', 'utf7': 'utf_7',
'ks_c_5601': 'euc_kr', 'ms936': 'gbk', 'iso_8859_11':
'iso8859_11', 'iso_8859_15': 'iso8859_15', 'ms932':
'cp932', 'sjis': 'shift_jis', 'iso_2022_jp_2':
'iso2022_jp_2', 'csiso2022jp': 'iso2022_jp',
'csibm858': 'cp858', 'iso_8859_14': 'iso8859_14',
'cskoi8r': 'koi8_r', 'eucjp': 'euc_jp',
'csisolatincyrillic': 'iso8859_5', 'ansi_x3_4_1968':
'ascii', 'csptcp154': 'ptcp154', 's_jisx0213':
'shift_jisx0213', 'uhc': 'cp949', 'iso_8859_9_1989':
'iso8859_9', 'sjisx0213': 'shift_jisx0213', 'ibm775':
'cp775', 'csibm1026': 'cp1026', 'iso_ir_157':
'iso8859_10', 'ibm855': 'cp855', 'iso_2022_jp_2004':
'iso2022_jp_2004', 'ibm857': 'cp857', 'iso_8859_10':
'iso8859_10', 'iso_2022_jp_1': 'iso2022_jp_1',
'ibm850': 'cp850', 'iso_2022_jp_3': 'iso2022_jp_3',
'ibm852': 'cp852', '869': 'cp869',
'cspc850multilingual': 'cp850', 'gb18030_2000':
'gb18030', 'iso_8859_1': 'latin_1', '861': 'cp861',
'860': 'cp860', 'ebcdic_cp_wt': 'cp037', '862':
'cp862', '865': 'cp865', '864': 'cp864', '866':
'cp866', 'thai': 'iso8859_11', 'l10': 'iso8859_16',
'csiso58gb231280': 'gb2312', 'iso_8859_4_1988':
'iso8859_4', 'utf_16le': 'utf_16_le', 'latin4':
'iso8859_4', 'csisolatinarabic': 'iso8859_6',
'iso2022jp_2004': 'iso2022_jp_2004', 'iso_2022_jp':
'iso2022_jp', 'l3': 'iso8859_3', 'csascii': 'ascii',
'863': 'cp863', 's_jis_2004': 'shift_jis_2004',
'roman8': 'hp_roman8', 'iso_ir_101': 'iso8859_2',
'ibm1140': 'cp1140', 'sjis_2004': 'shift_jis_2004',
'shiftjis2004': 'shift_jis_2004', 'iso_2022_kr':
'iso2022_kr', 'iso_ir_109': 'iso8859_3',
'ks_c_5601_1987': 'euc_kr', 's_jis': 'shift_jis',
'iso_8859_16': 'iso8859_16', 'hz_gb': 'hz', 'ibm864':
```

```
'cp864',   '1258':   'cp1258',   'ibm866':   'cp866',
'csiso2022kr':   'iso2022_kr',   'ibm860':   'cp860',
'ibm861':   'cp861',   'ibm862':   'cp862',   'ibm863':
'cp863', '437': 'cp437', 'ibm869': 'cp869', 'latin8':
'iso8859_14',   'latin9':   'iso8859_15',   'csibm500':
'cp500', 'iso_8859_14_1998': 'iso8859_14', 'utf_32be':
'utf_32_be',   'ebcdic_cp_he':   'cp424',   'ksx1001':
'euc_kr',   'greek':   'iso8859_7',   'eucgb2312_cn':
'gb2312', 'macturkish': 'mac_turkish', 'big5_hkscs':
'big5hkscs', 'csibm855': 'cp855', 'csibm857': 'cp857',
'maciceland':   'mac_iceland',   'ms_kanji':   'cp932',
'u_jis':   'euc_jp',   'maccyrillic':   'mac_cyrillic',
'utf8_ucs2':         'utf_8',         'iso2022jp_ext':
'iso2022_jp_ext',   'iso_8859_3_1988':   'iso8859_3',
'iso_ir_138':   'iso8859_8',   'latin':   'latin_1',
'iso_ir_100': 'latin_1', 'iso_8859_1_1987': 'latin_1',
'macgreek':   'mac_greek',   'cp936':   'gbk',   '949':
'cp949',   'utf_32le':   'utf_32_le',   'macintosh':
'mac_roman', 'eucjisx0213': 'euc_jisx0213', 'tis260':
'tactis', 'tis_620_2529_0': 'tis_620', 'iso2022jp':
'iso2022_jp', 'cp1361': 'johab', 'ujis': 'euc_jp',
'x_mac_trad_chinese':   'big5',   'csibm860':   'cp860',
'csibm861': 'cp861', 'l2': 'iso8859_2', 'csibm863':
'cp863', 'csibm864': 'cp864', 'hkscs': 'big5hkscs',
'csibm866':   'cp866',   'eucjis2004':   'euc_jis_2004',
'l8': 'iso8859_14', 'csibm869': 'cp869', 'cspcp852':
'cp852', 'euc_cn': 'gb2312', 'unicodelittleunmarked':
'utf_16_le',   'macroman':   'mac_roman',   'csibm037':
'cp037',   'csisolatin1':   'latin_1',   'csisolatin2':
'iso8859_2', 'csisolatin3': 'iso8859_3', 'csisolatin4':
'iso8859_4', 'csisolatin5': 'iso8859_9', 'csisolatin6':
'iso8859_10', 'iso_8859_11_2001': 'iso8859_11', 'utf':
'utf_8', 'iso_ir_166': 'tis_620', '8859': 'latin_1',
'euc_jis2004':   'euc_jis_2004',   'x_mac_simp_chinese':
'gb2312',   'hebrew':   'iso8859_8',   'ebcdic_cp_be':
'cp500', 'cyrillic': 'iso8859_5', 'pt154': 'ptcp154',
'1026': 'cp1026', 'mskanji': 'cp932', 'shiftjisx0213':
'shift_jisx0213',   'latin5':   'iso8859_9',   'utf8':
'utf_8', 'iso_ir_199': 'iso8859_14', 'csbig5': 'big5',
'x_mac_japanese':   'shift_jis',   'ibm437':   'cp437',
'iso_8859_5_1988':   'iso8859_5',   'cspc8codepage437':
'cp437', 'iso_8859_7_1987': 'iso8859_7', 'iso_8859_8':
'iso8859_8',     'unicodebigunmarked':     'utf_16_be',
'cyrillic_asian': 'ptcp154', 'ebcdic_cp_ca': 'cp037',
'037': 'cp037', 'cp_gr': 'cp869', 'ibm819': 'latin_1',
```

'ebcdic_cp_ch': 'cp500', 'latin7': 'iso8859_13', 'u32':
'utf_32', '850': 'cp850', '852': 'cp852',
'gb2312_1980': 'gb2312', 'euckr': 'euc_kr', '855':
'cp855', 'ibm039': 'cp037', '857': 'cp857', 'ibm037':
'cp037', 'windows_1257': 'cp1257', 'windows_1254':
'cp1254', 'windows_1255': 'cp1255', 'windows_1252':
'cp1252', 'windows_1253': 'cp1253', 'windows_1250':
'cp1250', 'windows_1251': 'cp1251', 'ansi_x3.4_1968':
'ascii', 'latin6': 'iso8859_10', 'korean': 'euc_kr',
'ibm865': 'cp865', 'iso_ir_58': 'gb2312',
'iso_8859_10_1992': 'iso8859_10', 'ibm500': 'cp500',
'950': 'cp950', 'iso_ir_126': 'iso8859_7', 'ecma_114':
'iso8859_6', 'tis620': 'tis_620', 'latin2':
'iso8859_2', 'utf_16be': 'utf_16_be', 'ms949': 'cp949',
'latin3': 'iso8859_3', 'latin10': 'iso8859_16', 'l4':
'iso8859_4', 'ibm1026': 'cp1026', 'us_ascii': 'ascii',
'latin1': 'latin_1', 'elot_928': 'iso8859_7',
'iso_ir_144': 'iso8859_5', 'jisx0213': 'euc_jis_2004',
'unicode_1_1_utf_7': 'utf_7', '646': 'ascii', 'cp154':
'ptcp154', 'iso_ir_148': 'iso8859_9', 'cp367': 'ascii',
'csisolatingreek': 'iso8859_7', 'l5': 'iso8859_9',
'csibm424': 'cp424', 'iso_2022_jp_ext':
'iso2022_jp_ext', 'r8': 'hp_roman8', 'l1': 'latin_1',
'cp819': 'latin_1', 'iso_8859_2_1987': 'iso8859_2',
'big5_tw': 'big5', 'hz_gb_2312': 'hz', 'ms950':
'cp950', 'cspc862latinhebrew': 'cp862',
'iso_8859_16_2001': 'iso8859_16', 'ecma_118':
'iso8859_7', 'us': 'ascii', 'x_mac_korean': 'euc_kr',
'csibm865': 'cp865', 'ansi_x3.4_1986': 'ascii',
'csshiftjis': 'shift_jis', 'u16': 'utf_16',
'maccentraleurope': 'mac_latin2', 'l6': 'iso8859_10',
'iso_ir_110': 'iso8859_4', '500': 'cp500',
'windows_1256': 'cp1256', 'utf32': 'utf_32',
'utf8_ucs4': 'utf_8', 'greek8': 'iso8859_7',
'iso_ir_6': 'ascii', 'l9': 'iso8859_15', '775':
'cp775', 'iso646_us': 'ascii', 'csHPRoman8':
'hp_roman8', 'iso_celtic': 'iso8859_14', 'l7':
'iso8859_13', 'iso2022kr': 'iso2022_kr', 'u7': 'utf_7',
'ebcdic_cp_us': 'cp037', 'iso8859_1': 'latin_1', '932':
'cp932', 'dbcs': 'mbcs', 'euccn': 'gb2312', '936':
'gbk', '858': 'cp858', 'ibm424': 'cp424',
'cspc775baltic': 'cp775', 'tis_620_0': 'tis_620',
'iso2022jp_1': 'iso2022_jp_1', 'cp_is': 'cp861',
'iso2022jp_3': 'iso2022_jp_3', 'iso2022jp_2':
'iso2022_jp_2', 'iso_8859_13': 'iso8859_13',

```
'iso_ir_226':     'iso8859_16',     'csisolatinhebrew':
'iso8859_8',    'gb2312_80':    'gb2312',    'shiftjis':
'shift_jis', 'ibm858': 'cp858'}
```

Eger kodlanma görkezilmedik bolsa, onda us-ascii ulanylýar diýip hasaplanýar. Üstesine, Python diliniň interpretatory modul işe göýberilende şeýle duýduryş berer:

```
sys:1: DeprecationWarning: Non-ASCII character '\xf0'
in file example.py
on line 2, but no encoding declared;
see http://www.python.org/peps/pep-0263.html for
details
```

Setir literallar

Programmada setirleri setir literallaryň kömegi bilen berip bolar. Literallar ‘ ýa-da “ , ýa-da bu simwollary üç gezek alynyp ýazylýar. Literalyň içinde ters gytak çyzygyň öz ähmiýeti bar. Ol ýörite simwollary girizmek üçin we simwollary kodlaryň üsti bilen aňlatmak üçin hyzmat edýär. Eger-de setir literallaryň öňünde r goýlan bolsa, ters gytak çyzygyň ähmiýeti ýok (r iňlisçe raw sözünden bolup, “bar” diýmegi aňladýar). Unicode-literallaryň öňünde u goýulýar. Ynha birnäçe mysal:

```
s1 = "setir 1"
s2 = r'\1\2'
s3 = """apple\ntree""" # \n – setiriň terjimesiniň
simwoly
s4 = """apple
tree"""
s5 = '\x73\65'
u1 = u"Unicode literal"
u2 = u'\u0410\u0434\u0440\u0435\u0441'
```

Bellik:
Ters gytak çyzyk literalda soňky simwol bolmaly däl, ýagny “str\” sintaksiki ýalňyşy çykarar.

Kodlamany görkezmeklik Unicode-literallarda haýsy kody ulanmalydygyny ýardam berýär. Eger kodlama görkezilmedik bolsa, onda ters gytak çyzygyň üsti bilen simwollaryň kodlaryndan peýdalanyp bolýar.

Setirleriň üstündäki amallar

Python dilinde ýörite sintaksiki goldawa eýe bolan setirleriň üstündäki amallara, hususanda, konkatenasiýa (ýelmemek), setiri gaýtalamak, formatirlemek degişli:

```
>>> print ("A" + "B", "A"*5, "%s" % "A")
AB AAAAA A
```

Formatirleme amalynda çep operand formatyň setiri bolup durýar, sag operandy bolsa başga görnüşiň islendik bahasy:

```
>>> print ("%i" % 234)
234
>>> print ("%i %s %3.2f" % (5, "ABC", 23.45678))
5 ABC 23.46
>>> a = 123
>>> b = [1, 2, 3]
>>> print ("%(a)i: %(b)s" % vars())
123: [1, 2, 3]
```

Formatirleme amaly

Formatyň setirinde tekstden başga ýörite belgiler ulanylyp biliner, olar çykarylýan bahanyň formatyny kesgitleýärler. Bu belgileriň şeýle sintaksisi bar:

```
"%"
[açar][baýdak*][giňligi][.takyklygy][görnüşiň_uzynlygy]
ýörite belgi
açar: "(" ýaýlardan galan simwollar * ")"
baýdak: "+" | "-" | boş ýer| "#" | "0"
giňligi: ("1" ... "9")("0" ... "9")* | "*"
takyklygy: ("1" ... "9")* | "*"
görnüşiň_uzynlygy: "a" ... "z" | "A" ... "Z"
ýörite belgi: "a" ... "z" | "A" ... "Z" | "%"
```

Bu ýerde simwollar aşakdakylary aňladýar:

`açar` – sözlükdäki açar.

`baýdak` – özgermäniň goşmaça häsiýetleri.

`giňligi` – meýdanyň iň kiçi ini

`takyklyk` – hakyky san üçin

`görnüşiň_uzynlygy` – tipiň modifikatory.

`ýörite belgi` – çykarylan obýektiň görkeziliş görnüşi.

Indiki tablisada formatirlemäni ýöriteleşdirmek üçin iň köp ulanylýan bahalaryň käbirleri getirilen:

Simwol	Ulanylýan ýeri	Görkezýän zady
0	baýdak	Çepden nollar bilen doldurmak
-	baýdak	Çep tarapdan deňlemek
+	baýdak	Sanyň alamatyny hökman çykarmak
boş ýer	baýdak	Sanyň alamatynyň ýerine boş ýer ulanmak

d,i	ýörite belgi	Alamatly bitin san
u	ýörite belgi	Alamatsyz bitin san
o	ýörite belgi	Sekiz belgili alamatsyz bitin san
x, X	ýörite belgi	On alty belgili alamatsyz bitin san (uly we kiçi latyn harplary bilen)
e, E	ýörite belgi	Eksponentaly formatdaky ýüzýän nokatly san
f, F	ýörite belgi	Ýüzýän nokatly san
g, G	ýörite belgi	Gysgaldylan görnüşde ýazylan ýüzýän nokatly san
c	ýörite belgi	Birbelgili simwol (bitin san ýa-da bir simwolly setir)
r	ýörite belgi	repr( ) funksiýa tarapynda setire geçirilen islendik obýekt
s	ýörite belgi	str( ) funksiýa tarapynda setire geçirilen islendik obýekt
%	ýörite belgi	Göterimiň belgisi. Ýekeleýin göterimi bermek üçin %% görnüşde ýazmaly

Indeksler we kesimler

Setirler üýtgemeýän yzygiderlikler, şonuň üçin olara indeks boýunça elementi almak ýa-da kesim ýaly amallary ulanyp bolýar:

```
>>> s="maglumat"
>>> print(s[0], s[-1])
m t
>>> print(s[-4])
u
>>> s="maglumat"
>>> print(s[0], s[-1])
m t
>>> print(s[-4:])
umat
>>> print(s[:5])
maglu
>>> print(s[4:7])
uma
```

Bellik:
Kesimler bellenilende setiriň simwollary däl-de, olaryň aralyklary bellenilýär.

string moduly

Setirler üçin usullar peýda bolýança, setirleriň üstündäki amallar üçin string moduly ulanylýardy. Aşakdaky mysalda formatirleme amalynyň başga görnüşi bolan Template synpy görkezilen:

```
>>> import string
>>> tpl=string.Template("$a+$b=${c}")
>>> a=2
>>> b=3
>>> c=a+b
>>> print(tpl.substitute(vars()))
2+3=5
>>> del c # c ady ýok edýär
>>> print(tpl.safe_substitute(vars()))
2+3=${c}
>>> print(tpl.substitute(vars(), c=a+b))
2+3=5
>>> print(tpl.substitute(vars()))
Traceback (most recent call last):
  File "<pyshell#168>", line 1, in <module>
    print(tpl.substitute(vars()))
  File"C:\Python33\lib\string.py",line121,in substitute
    return self.pattern.sub(convert, self.template)
  Fil"C:\Python33\lib\string.py",line 111, in convert
    val = mapping[named]
KeyError: 'c'
```

Bu ýerde iki usul bar: substitute( ) we safe_substitute( ).

Setirleriň usullary

Aşakdaky tablisada obýekt-setirleriň we unicode-obýektleriň iň köp ulanylýan usullary getirilen.

Usul	Beýany
`center(w)`	w uzynlykly meýdanda setiri merkeze ýerleşdirýär
`count(sub)`	sub setiriň beýleki setire girmeginiň sany
`encode([enc[,errors]])`	enc kodlamadaky setiri çykarýar. errors parametr “strict”, “ignore”, “replace” ýa-da “xmlcharrefreplace” bahany alyp bilýär
`endswith(suffix)`	Setir suffix bilen gutarýarmy
`expandtabs([tabsize])`	Tabulýasiýa simwollary boş ýer bilen çalşyrýar.

	Ölçegi berilmese tabsize=8.
`find(sub [, start [, end]])`	sub kiçi setiriň setire girip başlaýan indeksiniň iň kiçisi çykarýar. start we end parametrler start:end gözleg bilen çäklenýärler. Eger kiçi setir tapylmasa -1 baha çykarylýar
`index(sub [, start [, end]])`	find( ) meňzeş. Ýöne jogap çykmasa ValueError ýalňyşy çykarýar.
`alnum()`	Eger-de setir 0-dan uzyn bolsa we diňe harplary we sifrleri özünde saklaýan bolsa True baha eýe, ýogsa False.
`isalpha()`	Eger-de setir 0-dan uzyn bolsa we diňe harplary özünde saklaýan bolsa True baha eýe.
`isdecimal()`	Eger-de setir 0-dan uzyn bolsa we diňe onluk sanlary (diňe Unicode setirler üçin) özünde saklaýan bolsa, onda True baha eýe
`isdigin()`	Eger-de setir 0-dan uzyn bolsa we diňe sifrleri özünde saklaýan bolsa, onda True baha eýe
`islower()`	Eger-de setir özünde diňe setir harplary (1-den köp) saklaýan bolsa, onda True baha eýe, ýogsa False.
`isnumeric()`	Eger-de setir özünde diňe sanlary (diňe Unicode üçin) saklaýan bolsa çykarýar.
`isspace()`	Eger setir diňe boş simwollardan durýan bolsa True baha eýe, ýogsa False (bu ýerde boş setir üçin hem False baha).
`join(seq)`	Berlen setiriň bölünijisiniň üsti bilen seq yzygiderlikden setiri birikdirmek
`lower()`	Setiri harplaryň iň aşaky registrine getirýär
`lstrip()`	Çepden boş simwollary ýok edýär.
`replace(old, new[, n])`	old kiçi setiri new bilen çalşyrylan setiriň nusgasyny çykarýar.
`rstrip()`	Sag tarapdan boş simwollary ýok edýär.
`startswith(prefix)`	Setir prefix kiçi setirden başlaýar
`strip()`	Setiriň başyndan we soňundan boş simwollary ýok edýär
`translate(table)`	Unicode-setirler üçin bolup, kodlarda koda table tablisanyň kömegi bilen özgerdilme ýerine ýetirilýär.
`translate(table[, dc])`	Şol bir zat, ýöne adaty setirler üçin.
`upper()`	Setiriň harplaryny ýokarky registre terjime edýär.

Indiki mysalda setiri sanawa (bölünijiler boýunça) we tersine sanawdan setirlere birleşdirmek üçin split( ) we join( ) usullar ulanylýar:

```
>>> s="Bu hem bir mysal."
>>> lst=s.split(" ")
>>> print(lst)
['Bu', 'hem', 'bir', 'mysal.']
>>> s2="\n".join(lst)
>>> print(s2)
Bu
hem
bir
mysal.
```

Setir kesgitli harplaryň düzümine gutarýandygyny barlamak üçin `endswith()` usulyny ulanyp bolar:

```
>>> filenames = ["file.txt", "image.jpg", "str.txt"]
>>> for fn in filenames:
   if fn.lower().endswith(".txt"):
       print(fn)


file.txt
str.txt
```

Setirdäki gözlegi ýerine ýetirmek üçin `find()` usuly ulanylýar. Indiki programma `def` operatory tarapynda modulda kesgitlenen ähli funksiýalary çykarýar:

```
import string
text = open(string.__file__[:-1]).read()
start = 0
while 1:
    found = text.find("def ", start)
    if found == -1:
        break
    print (text[found:found + 60].split("(")[0])
    start = found + 1
```

Tekst maglumaty özgertmek üçin esasy zat `replace()` usuly bolup durýar, oňa aşakda seredeliň:

```
>>> a="bu setirde , oturlar , dogry goyulmady."
```

```
>>> b=a.replace(" ,", ",")
>>> print(b)
bu setirde, oturlar, dogry goyulmady.
```

Täsirlilik boýunça maslahatlar

Örän uzyn setirler bilen ýa-da örän köp setirler bilen işlenende ulanylýan amallar programmanyň tizligine dürlüçe täsir edip biler.

Mysal üçin, köp sanly setirleri bir setire birleşdirmek üçin konkatenasiýa amalyny köp ulanmak maslahat berilmeýär. Iň gowusy setirleri sanawda toplap, soňra `join()` kömegi bilen bir setirde jemlemeli.

```
>>> a=""
>>> for i in range(1000):
            a+=str(i)  # bu täsirli däl


>>> a="".join([str(i) for i in range(1000)]) #has täsirli
```

Eger-de soňra setirleriň üstünde işlenilse, onda iteratorlary ulanyp bolar, olar huşda az ýer tutmaga ýardam berýärler.

StringIO moduly.

Käbir ýagdaýlarda setir bilen faýl ýaly işlemeli. StringIO moduly şu mümkinçiligi berýär.

“Faýlyň” açylyşy `StringIO()` çagyrylyp amala aşyrylýar. Argumentsiz çagyrylsa – täze “faýl” açylýar, argument hökmünde setir berilse – “faýl” okalmak üçin açylýar:

```
import StringIO
my_string = "1234567890"
f1 = StringIO.StringIO()
f2 = StringIO.StringIO(my_string)
```

Soňra f1 we f2 faýllar bilen adaty faýl obýektleri ýaly işläp bolýar. Şeýle faýlyň düzümini setir görnüşinde almak üçin `getvalue()` usuly ulanylýar:

```
f1.getvalue()
```

difflib moduly

Iki setiri takmyn deňeşdirmek üçin standart kitaphanada difflib moduly göz-öňüne tutulan.

Berlen setire ýakyn n setirleri bellemek üçin `difflib.get_close_matches()` funksiýa ulanylýar.

```
get_close_matches(word, possibilities, n=3, cutoff=0.6)
```

Bu ýerde: `word` – ýakyn setirler gözlenýän setirler, `possibilities` – mümkin wariantlaryň sanawy, `n` – ýakyn setirleriň talap edilýän sany, `cutoff` – gabat gelýän setirleriň gerekli derejesiniň koeffisiýenti ([0,1] çäkden) we `word` bilen deňeşdirilende kiçi baha eýe setirler alynmaýar.

Indiki mysal `difflib.get_close_matches()` funksiýanyň işleýşini görkezýär:

```
>>> import unicodedata
>>> names = [unicodedata.name(unicode(chr(i))) for i in
range(40, 127)]
>>> print difflib.get_close_matches("LEFT BRACKET", names)
['LEFT CURLY BRACKET', 'LEFT SQUARE BRACKET']
```

`names` sanawda – Unicode-simwollaryň atlary ASCII-kodlary bilen 40-dan 127-ä çenli.

Yzygiderli aňlatmalar

Tekstler bilen işlemek üçin seredilip geçilen standart mümkinçilikler hemişe bolup duranok. Mysal üçin, `find()` we `replace()` usullarda bary-ýogy bir setir berilýär. Hakyky meselelerde şeýle ýagdaý örän seýrek duş gelýär, köplenç käbir nusga gabat gelýän setirleri tapyp çalyşmak talap edilýär.

Yzygiderli aňlatmalar (regular expressions) ýörite dil ulanyp, setirleriň köplügini ulanýar. Yzygiderli aňlatma berlen setire nusga diýilýär.

Python dilinde yzygiderli aňlatmalar bilen işlemek üçin re moduly ulanylýar. Indiki mysalda yzygiderli aňlatma tekstden ähli sanlary bellemäge ýardam berýär.

```
>>> import re
>>> pattern = r"[0-9]+"
>>> number_re = re.compile(pattern)
>>> number_re.findall("122 234 65435")
['122', '234', '65435']
```

Bu mysalda pattern nusga setirleriň köplügini suratlandyrýar, olar bolsa “0”, “1”, …, “9” toplumdan bir ýa-da birnäçe simwollardan ybarat. `re.compile( )` funksiýa nusgany ýörite Regex-obýekte kompilirleýär, ol bolsa berlen setire nusga gabat gelýän ähli sanawlary almak üçin ulanylýan findall( ) funksiýasyny, şeýle-de birnäçe usullary özünde saklaýar.

Bu mysaly başgaça şeýle görnüşde hem ýazyp bolýar:

```
>>> import re
>>> re.findall(r"[0-9]+", "122 234 65435")
['122', '234', '65435']
```

Nusgany öňünden kompilýasiýasy sikliň içinde ýerine ýetirilse gowy.

Bellik:
Nusgany ulanmak üçin işlenilmedik setir ulanylan. Bu mysalda zerur bolmasada, umumy ýagdaýlarda setir literallary ters gytak çyzygyň üsti bilen ýazylýan ýörite yzygiderlikleriň täsiriniň öňüni almak üçin şeýle ýazmaly.

Yzygiderli aňlatmalaryň sintaksisi

Python dilinde yzygiderli aňlatmalaryň sintaksisi edil Perl, Grep we beýleki gurallardaky ýaly. Simwollaryň bir bölegi (esasan harplar we sifrler) öz-özüni aňladýar.

Ýene-de bir zady bellemek gerek, ýagny dürli amallar nusgany dürlüçe ulanýarlar. Mysal üçin, search( ) berlen setirde nusga gabat gelýän setiriň birinji elementinden gözleýär, match( ) bolsa setiriň nusga başdan gabat gelmegini talap edýär.

Yzygiderli aňlatmalardaky ýazgylarda ýörite baha eýe bolan simwollar:

Simwol	Aňlatmasy
"."	Islendik simwol
"^"	Setiriň başy
"$"	Setiriň soňy
"*"	Parçany gaýtalamak nol we ondan köp gezek
"+"	Parçany gaýtalamak bir we ondan köp gezek
"?"	Öňden gelen parça ýa bar ýa-da ýok
"{m,n}"	Parçany m-den n-e çenli gaýtalamak
"[...]"	Ýaýlardaky toplumdan islendik simwol. Simwollaryň çägini yzygider gidýän kodlary bilen berip bolar: a-z
"[^...]"	Ýaýlaryň içindäkilere degişli bolmadyk islendik simwol
"\"	Ters gytak çyzyk yzyndan gelýän simwolyň ýörite bahasyny çykarmaýar.
"|"	Sagdan ýa-da çepden parça
"*?"	Parçany nol ýa-da ondan köp gezek gaýtalamak
"+?"	Parçany birden köp gezek gaýtalamak
"{m,n}?"	Öňünden gelen parçany m-den n-e çenli gaýtalamak.

Eger A we B – yzygiderli aňlatmalar bolsa, onda olaryň AB konkatenasiýasy täze tertipli aňlatmalar bolar, mundan başga-da a we b setirleriň konkatenasiýasy AB-ni kanagatlandyrar, eger a setir A-ny we b setir B-ni kanagatlandyrýan bolsa. Konkatenasiýa – yzygiderli aňlatmalary düzmegiň esasy usuly diýip hasaplap bolar.

Aşakdaky tablisada ters gytak çyzygy ulanýan ýörite yzygiderlikler berlen:

Yzygiderlik	**Nämä gabat gelýär**
"\1" - "\9"	Görkezilen nomerli topar. Toparlar 1-den başlap nomerlenýär.
"\A"	Ähli setiriň öňündäki boş ýer (“^” meňzeş)
"\Z"	Ähli setiriň soňundaky boş ýer (“$” meňzeş)

"\b"	Sözüň öňünden ýa-da soňundan simwollaryň arasyndaky boş ýer
"\B"	Tersine, sözüň öňünden ýa-da soňundan simwollaryň arasyndaky boş ýere gabat gelenok
"\d"	Sifr. “[0-9]” meňzeş
"\s"	Islendik probel simwoly. "[\t\n\r\f\v]" meňzeş
"\S"	Islendik probel däl simwol. [^\t\n\r\f\v]" meňzeş
"\w"	Islendik sifr ýa-da harp (LOCALE baýdaga bagly)
"\W"	Sifr ýa-da harp bolmadyk islendik simwol (LOCALE baýdaga bagly).

Yzygiderli aňlatmalar bilen ulanylýan baýdaklar:

`"(?i)", re.I, re.IGNORECASE` - deňeşdirme harplaryň registri hasaba alynmazdan geçirilýär.

`"(?L)", re.L, re.LOCALE` - gurşawa (locale) baglylykda kesgitli harplara `"\w", "\W", "\b", "\B"` täsir edýär.

`"(?m)", re.M, re.MULTILINE` – eger bu baýdak berlen bolsa, onda “^” we “$” begliler setiriň başyna we soňuna gabat gelip biler.

`"(?s)", re.S, re.DOTALL` – eger berlen bolsa, onda “.” belgi “\n” setiriň ahyrky simwolyna gabat geler.

`"(?u)", re.U, re.UNICODE` – setirde we nusgada Unicode ulanylypdyr.

Obýekt-nusganyň usullary

Nusganyň re.compile( ) funksiýa tarapyndan şowly kompilýasiýasyndan soň obýekt-nusga alynýar, onuň bolsa birnäçe usullary bar.

`match(s)` funksiýa – s setiri nusga bilen deňeşdirýär, eger deňeşdirme şowly bolsa deňeşdirme netijesi bilen obýekti çykarýar (obýekt `SRE_Match`). Eger şowuna bolmasa None çykarýar. Deňeşdirme setiriň başyndan başlaýar.

`search(s)` funksiýa – `match(s)` funksiýa meňzeş, ýöne s setiriň tutuş özünden laýygy gözleýär.

`split(s [, maxsplit=0])` funksiýa – setirleri kiçi setirlere bölýär.

`findall(s)` funksiýa – nusgany kanagatlandyrýan ähli s kiçi setirleri gözleýär.

`finditer(s)` funksiýa – nusgany kanagatlandyrýan ähli setirler üçin deňeşdirme netijesi bilen obýektler boýunça iteratorlary çykarýar.

Aşakdaky mysalda berlen nusga boýunça setir kiçi setirlere bölünýär:

```
>>> import re
>>> delim_re = re.compile(r"[:,;]")
>>> text = "This,is;example"
>>> print (delim_re.split(text))
['This', 'is', 'example']
```

Indi bolsa setir näme bilen bölünendigine seredeliň:

```
>>> delim_re = re.compile(r"([:,;])")
>>> print (delim_re.split(text))
['This', ',', 'is', ';', 'example']
```

Nusgalaryň mysallary

Yzygiderli aňlatmalara eýe bolup, maglumatlaryň üstünde işlemek üçin algoritmleri gurmagy tizleşdirip bolýar.Iň gowusy nusgalar bilen takyk mysallarda tanyşalyň:

`r"\b\w+\b"` – harplardan we aşagy çyzylan belgilerden ybarat bolan söze gabat gelýär.

`r"[+-]?\d+"` – bitin sana gabat gelýär.

`r"\([+-]?\d+\)"` – ýaýlaryň içindäki san.

`r"[a-cA-C]{2}"` – “a”, “b” ýa-da “c” iki harplardan durýan setire gabat gelýär. Mysal üçin, “Ac”, “CC”, “bc”.

`r"aa|bb|cc|AA|BB|CC"`–iki sany birmeňzeş harplardan durýan setir.

`r"([a-cA-C])\1"` – iki sany birmeňzeş harplardan durýan setir, ýöne setir topary ulanmak bilen berdi.

`r"aa|bb".` – “aa” ýa-da “bb” gabat gelýär.

`r"a(a|b)b"` – “aab” ýa-da “abb” gabat gelýär.

`r"^(?:\d{8}|\d{4}):\s*(.*)$"` – sekiz ýa-da dört sifrlerden we iki nokatdan ybarat bolan toplumdan durýan setire gabat gelýär.

`r"(\w+)=.*\b\1\b"` – deňdiriň çepinden we sagyndan gelýän sözler bar. `\1\` begli bilen belgilenen operand ýaý bilen bellenen 1 nomerli topara gabat gelýär.

`r"(?P<var>\w+)=.*\b(?P=var)\b"` - şol bir zat, ýöne atlandyrylan var topary ulanylýar.

`r"\bregular(?=\s+expression)".` – probelden soň “`expression`” sözi gelýän ýagdaýynda “`regular`” sözüne gabat gelýär.

`r"(?<=regular )expression"` – “`expression`” sözüniň öňünden “`regular`” sözi we bir probel duran söze gabat gelýär.

Yzygiderli aňlatmalaryň ulanylyş mysallary

Logy gaýtadan işlemek

Bu mysalda yzygiderliaňlatma logdaky ýazgylary kesgitlemäge we gysgaldylan görnüşde bermäge ýardam berer:

```
import re
log_re=re.compile(r"""(?P<date>[A-Za-
z]{3}\s+\d+\s+\d\d:\d\d:\d\d) \S+
kernel:
PAY:.+DST=(?P<dst>\S+).*LEN=(?P<len>\d+).*DPT=(?P<dpt>\d+)"
""")

for line in open("message.log"):
m = log_re.match(line)
```

```
if m:
print("%(date)s%(dst)s:%(dpt)ssize=%(len)s"% m.groupdict())
```

Netijede şeýle alynýar:

```
Nov 27 15:57:59 192.168.1.115:1039 size=1500
Nov 27 15:57:59 192.168.1.200:8080 size=40
Nov 27 15:57:59 192.168.1.115:1039 size=515
Nov 27 15:57:59 192.168.1.200:8080 size=40
Nov 27 15:57:59 192.168.1.115:1039 size=40
Nov 27 15:57:59 192.168.1.200:8080 size=40
Nov 27 15:57:59 192.168.1.115:1039 size=40
```

Sany ýazmagyň derňewi

Yzygiderli aňlatmanyň gowy mysalyny fpformat modulynda tapyp bolýar. Bu yzygiderliaňlatma sany ýazmagyň üstünde işlemäge ýardam berýär:

```
decoder = re.compile(r'^([-+]?)0*(\d*)((?:\.\d*)?)(([eE][-+]?\d+)?)$')
# Sanly literalyň indiki bölekleri toparlaryň kömegi
bilen bellenýärler:
# \0 – tutuş literal
# \1 – başlangyç alamat ýa-da boş
# \2 – nokatdan çepdäki sifrler
# \3 – drob bölegi(boş ýa-da nokatdan başlaýar)
# \4 - görkeziji(boş ýa-da ‘e’ ýa-da ‘E’-den başlaýar)
```

Mysal üçin:

```
import re
decoder=re.compile(r'^([-
+]?)0*(\d*)((?:\.\d*)?)((?:[eE][-+]?\d+)?)$')

print (decoder.match("12.234").groups())
print (decoder.match("-0.23e-7").groups())
print (decoder.match("1e10").groups())
```

Netijede alarys:

```
>>>
('', '12', '.234', '')
('-', '', '.23', 'e-7')
('', '1', '', 'e10')
```

Köp gezek çalyşmak

Käbir priloženiýelerde tekstiň içinde bir wagtda birnäçe çalyşmalary etmeli bolýar. Bu meseläni çözmek üçin ýörite sub( ) usuly ulanylýar.

```
import re
def multisub(subs_dict, text):
    def _multisub(match_obj):
```

```
        return str(subs_dict[match_obj.group()])
    multisub_re=re.compile("|".join(subs_dict.keys()))
    return multisub_re.sub(_multisub, text)

repl_dict = {'one': 1, 'two': 2, 'three': 3}
print (multisub(repl_dict, "One, two, three"))
```

Şeýle jogap alynar:

```
>>>
One, 2, 3
```

Birnäçe faýllar bilen işlemek

Birnäçe faýllar bilen işlemegi ýeňilleşdirmek üçin `fileinput` moduly ulanylýar. Ol bir aýlawyň içinde buýruk setirinde görkezilen faýllaryň ähli setirlerini gaýtadan işlemäge ýardam berýär.

```
import fileinput
for line in fileinput.input():
  process(line)
```

Eger-de faýllar ýok bolsa, onda standart giriş ulanylýar.

Unicode bilen işlemek

Unicode döremänkä kompýuterde simwollar bir baýt bilen (ýa-da ýedi bit bilen) kodirlenýärdiler. Bir baýt 0-dan 255aralygyndaky kodlaryň çägini öz içine alýar, bu bolsa bir baýtda iki elipbiýden, sifrlerden, belgilerden we käbir simwollaryň toplumyndan artyk sygmaýar. Her öndüriji şol bir elipbiý üçin öz kodirlemesini ulanýardy. Mysal üçin, häzirki güne çenli kirilisanyň tutuş bäş sany kodirlemesi saklanyp galdy we her ulanyjy öz brauzerinde ýa-da elektron hatynda kodlaryň gabat gelmeýän mysalyna duş gelýär.

Unicode standart – bu dünýäniň ähli dilleriniň simwollary üçin bitewi kodlama. Munuň gowy tarapy bir Unicode-setirde dürli dilleriň simwollary ýerleşip bilýär. Erbet tarapy bolsa, ulanyjylar eýýäm bir baýtly kodlamalary ulanmagy endik etdiler, üstesine-de köp priloženiýeler bir baýtly kodlama niýetlenen, köp ulgamlarda bolsa Unicode dolylygyna amala aşyrylmaýar, çünki şriftleri gaýtadan işlemek üçin köp iş talap edilýär.

Ýene bir zady bellemek gerek, ýagny faýllar baýtlaryň yzygiderligi diýip hasaplanýar, şonuň üçin teksti Unicode görnüşde saklamak üçin ýörite Unicode kodlamalary (utf-7, utf-8, utf-16,...) ulanmak talap edilýär. Bu kodlamalar berlen platformadaky baýtlaryň tertibine eýe. Muny bilmek üçin sys modulyň atributyny okamaly. Intel platformada bu şeýle bolar:

```
>>> import sys
>>> sys.byteorder
```

Setiri Unicode özgertmek üçin tekst haýsy koda kodlanandygyny bilem gerek. Mysal üçin ol cp1251 bolsun. Onda teksti Unicode özgertmek üçin şeýle alarys:

```
>>> s = "cp1251"
>>> s.decode("cp1251")
u'\u0421\u0442\u0440\u043e\u043a\u0430 \u0432 cp1251'
```

Muny `unicode( )` gurnalan funksiýanyň kömegi bilen hem alyp bolar:

```
>>> unicode(s, 'cp1251')
u'\u0421\u0442\u0440\u043e\u043a\u0430 \u0432 cp1251'
```

Bu modulyň peýdaly funksiýalarynyň biri `codecs.open( )` funksiýasy, ol faýly başga kodda açmaga ýardam berýär:

```
vcodecs.open(, mode[, enc[, errors[, buffer]]])
```

Bu ýerde:

`filename` – faýlyň ady, `mode` – faýly açmagyň režimi, `enc` – kody, `errors` – kodyň ýalnyş bolan ýagdaýynda täsir režimi (`'strict'` – ýalňyşy çykarmak, `'replace'` – ýok simwollary çalşyrmak, `'ignore'` – ýalňyşlary ret etmek), `buffer` – bufer režimi (0 – bufer ýok, 1 – setir boýunça, n – buferiň baýty).

Netije

Bu sapakda teksti dolandyrmagyň esasy görnüşleri seredilip geçildi: setirler we Unicode-setirler. Yzygiderli aňlatmalar örän aýdyň beýan edildi – teksti derňemegiň örän täsirli mehanizmleriň biri. Sapagyňyň ahyrynda Unicode bilen işlemek üçin käbir funksiýalar getirilen.

EDEBIÝATLAR

1. Türkmenistanyň Konstitusiýasy. Aşgabat, 2008.
2. Gurbanguly Berdimuhamedow. Ösüşiň täze belentliklerine tarap. Saýlanan eserler. I tom. Aşgabat, 2008.
3. Gurbanguly Berdimuhamedow. Ösüşiň täze belentliklerine tarap. Saýlanan eserler. II tom. Aşgabat, 2009.
4. Gurbanguly Berdimuhamedow. Garaşsyzlyga guwanmak, Watany, halky söýmek bagtdyr. Aşgabat, 2007.
5. Gurbanguly Berdimuhamedow. Türkmenistan – sagdynlygyň we ruhubelentligiň ýurdy. Aşgabat, 2007.
6. Türkmenistanyň Prezidenti Gurbanguly Berdimuhamedowyň Ministrler Kabinetiniň göçme mejlisinde sözlän sözi. (2009-njy ýylyň 12-nji iýuny). Aşgabat, 2009.
7. Türkmenistanyň Prezidentiniň "Obalaryň, şäherleriň, etrapdaky şäherçeleriň we etrap merkezleriniň ilatynyň durmuş-ýaşaýyş şertlerini özgertmek boýunça 2020-nji ýyla çenli döwür üçin" Milli maksatnamasy, Aşgabat, 2007.
8. "Türkmenistany ykdysady, syýasy we medeni taýdan ösdürmegiň 2020-nji ýyla çenli döwür üçin Baş ugry" Milli Maksatnamasy, "Türkmenistan" gazeti, 2003-nji ýylyň 27-nji awgusty.
9. "Türkmenistanyň nebitgaz senagatyny ösdürmegiň 2030-njy ýyla çenli döwür üçin Maksatnamasyt". Aşgabat, 2006.
10. Марк Саммерфилд. Программирование на Python 3. Москва - 2009
11. Сузи Роман Авриевич. Язык программирования Python
12. Г.Шилдт. C# 4.0. Полное руководство, Москва-Санкт Петербург, 2011.
13. М.Е.Фленов. Библия C#. 2-ое издание. СанктПетербург, 2011.
14. The C Programming language, Second Edition, Brain W.Kernigan, Deniss M. Ritchie. Colorado Uniwersiteti, 1982.
15. C at Any Speed, Richard S. Wiener, Uniwersity of Colorado at Colorado Spring Colorado Uniwersiteti, 1988.
16. Б. Керниган, Д.Ритчи Язык программирования C++, Москва "Финансы и статистика" – 1992.
17. Р.Уинер. Язык ТУРБО С, Москва, 1991.
18. Дэвид Дж.Круглинский, Скотт Уингоу, Джордж Шеферд (Для профессионалов) Программирование на Microsoft VISUAL C++ 6.0, Санкт-Петербург, Москва, Харьков, Минск, 2002.
19. Н.Культин C/C++ в задачах и примерах, Санкт-Петербург "БХВ-Петербург", 2008.

Mazmuny