3-1-Liste-S

3.1 Liste i drugi kontejneri

Liste su važan dio Sage-a i pythona. Srodne su poljima (array) iz standardnih programskih jezika (C, Fortran), ali imaju bitno više svojstava. Elementi liste mogu biti praktički bilo koji objekti.

```
t1 = [1, 3, 5, 7, 9, 11]; print t1
t2 = [sin, cos, log]; print t2
t3 = [[], ["jedan"], t1]; t3
```

```
[1, 3, 5, 7, 9, 11]
[sin, cos, <function log at 0x509d050>]
[[], ['jedan'], [1, 3, 5, 7, 9, 11]]
```

Već smo ranije upoznali pristupanje elementima liste pomoću indeksiranja i mogućnost da na taj način promijenimo pojedine elemente liste:

```
print t1[1]
t1[1] = "tri"
print t1[1]
```

```
3
tri
```

Ukoliko želimo pristupiti većem broju elemenata od koristi su tzv. *rezovi* liste (engl. *slice*). Općenita sintaksa je `lista[početak:kraj:korak]`, gdje je "početak" uključen u rez, a "kraj" nije:

```
print t1[2:6]
t1[2:6:2]
```

```
[5, 7, 9, 11]
[5, 9]
```

Ukoliko rez ide od početka ili do kraja liste, odgovarajuće indekse možemo izostaviti:

```
t1[2:]
```

```
t1[2:]
```

```
[5, 7, 9, 11]
```

Negativni indeksi nam omogućuju da brojimo od kraja liste ...

```
t1[-2:]   # zadnja dva elementa
```

```
[9, 11]
```

... a negativan korak da rez ide unatrag:

```
t1[::-1]  # lista natraške
```

```
[11, 9, 7, 5, 'tri', 1]
```

Za traženje indeksa nekog elementa liste, ubacivanje elemenata u liste i slične operacije stoje na raspolaganju metode lista:

```
dir(t1)[-9:]
```

```
['append', 'count', 'extend', 'index', 'insert', 'pop',
'remove',
'reverse', 'sort']
```

Ovdje smo koristili funkciju dir() koja daje listu atributa nekog objekta (uključujući njegove metode). Samo zadnjih 9 je trenutno zanimljivo. (Inaće, dir() bez argumenta daje popis svih trenutno definiranih imena.)

```
print len(dir())
dir()[:12]
```

```
1892
['AA', 'AbelianGroup', 'AbelianGroupElement', 'AbelianGroupMap',
'AbelianGroupMorphism', 'AbelianGroupMorphism_id',
'AbelianGroup_class', 'AbelianGroup_subgroup', 'AbelianStrata',
'AbelianStratum', 'AbelianVariety', 'AbstractCategory']
```

Osim count() i index(), sve ostale metode mijenjaju listu.

```
t2 = t2[:-1]; t2   # micanje zadnjeg elementa
```

```
[sin, cos]
```

```
t1
```

```
[1, 'tri', 5, 7, 9, 11]
```

Zadatak 3.1.1:Izbrišite u gornjoj listi t1 treći element (ne element s indeksom=3!) i dodajte na kraj još dva elementa, brojeve 13 i 15. Ispišite novonastalu listu, a zatim element 'tri' zamijenite s brojem 3, koristeći metodu index().

Ukoliko treba transformirati sve elemente liste, najelegantnije je koristiti *obuhvaćanje liste* (engl. *list comprehension*):

```
[n+1 for n in t1]
```

```
Traceback (click to the left of this block for traceback)
...
TypeError: unsupported operand parent(s) for '+': '<type
'str'>' and 'Integer Ring'
```

Često se javlja potreba za izborom elemenata liste koji zadovoljavaju neki kriterij. To se može izvesti obuhvaćanjem liste uz dodatni uvijet:

```
t5 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[n for n in t5 if is_even(n)]    # izbor parnih elemenata
```

```
[2, 4, 6, 8, 10]
```

Ovdje smo koristili funkciju is_even() koje je jedna iz velike porodice tzv. predikata. Predikat je termin iz matematičke logike koji označava funkciju koja poprima isključivo vrijednosti True ili False. Većina predikata definiranih u Sageu počinje s "is_" (jer odgovaraju na pitanje "da li je ..."). Ispišimo ih (koristimo činjenicu da su stringovi kao liste znakova pa možemo koristiti rezove na njima):

```
[p for p in dir() if p[:3]=='is_']
```

```
['is_2_adic_genus', 'is_32_bit', 'is_64_bit', 'is_AbelianGroup',
'is_AbelianGroupElement', 'is_AbelianGroupMorphism',
'is_AbsoluteNumberField', 'is_AdditiveGroupElement',
'is_AffineScheme', 'is_AffineSpace', 'is_Algebra',
'is_AlgebraElement', 'is_AlgebraicField', 'is_AlgebraicNumber',
'is_AlgebraicReal', 'is_AlgebraicRealField', 'is_
AlgebraicScheme',
'is_AmbientSpace', 'is_BinaryStringMonoidElement',
'is_CallableSymbolicExpressionRing', 'is_Category',
'is_CommutativeAlgebraElement', 'is_CommutativeRing',
'is_CommutativeRingElement', 'is_ComplexDoubleElement',
'is_ComplexField', 'is_ComplexIntervalField',
'is_ComplexIntervalFieldElement', 'is_ComplexNumber',
'is_CyclotomicField', 'is_DedekindDomain',
'is_DedekindDomainElement', 'is_DirichletCharacter',
'is_DirichletGroup', 'is_DiscreteProbabilitySpace',
'is_DiscreteRandomVariable', 'is_DualAbelianGroup',
'is_DualAbelianGroupElement', 'is_Element', 'is_EllipticCurve',
'is_Endset', 'is_EuclideanDomain', 'is_EuclideanDomainElement',
'is_ExpectElement', 'is_Field', 'is_FieldElement', 'is_
FiniteField',
'is_FiniteFieldElement', 'is_FractionField',
'is_FractionFieldElement', 'is_FreeAbelianMonoid',
'is_FreeAbelianMonoidElement', 'is_FreeAlgebra', 'is_
FreeModule',
'is_FreeModuleElement', 'is_FreeModuleHomspace',
'is_FreeModuleMorphism', 'is_FreeMonoid', 'is_
FreeMonoidElement',
'is_FreeQuadraticModule', 'is_Functor', 'is_GapElement',
'is_GlobalGenus', 'is_GpElement', 'is_Graphics',
'is_HexadecimalStringMonoidElement', 'is_Homset',
'is_HyperellipticCurve', 'is_Ideal', 'is_Infinite',
'is_InfinityElement', 'is_Integer', 'is_IntegerMod',
'is_IntegerModRing', 'is_IntegralDomain',
'is_IntegralDomainElement', 'is_KashElement',
'is_LaurentPolynomialRing', 'is_LaurentSeries',
'is_LaurentSeriesRing', 'is_MPolynomial', 'is_MPolynomialIdeal',
'is_MPolynomialRing', 'is_MagmaElement', 'is_Matrix',
'is_MatrixGroup', 'is_MatrixGroupElement', 'is_MatrixSpace',
'is_MaximaElement', 'is_ModularAbelianVariety',
'is_ModularFormElement', 'is_ModularFormsSpace',
```

```
'is_ModularSymbolsElement', 'is_ModularSymbolsSpace', 'is_
Module',
'is_ModuleElement', 'is_Monoid', 'is_MonoidElement', 'is_
Morphism',
'is_MultiplicativeGroupElement', 'is_NumberField',
'is_NumberFieldElement', 'is_NumberFieldFractionalIdeal',
'is_NumberFieldFractionalIdeal_rel', 'is_NumberFieldIdeal',
'is_NumberFieldOrder', 'is_OctalStringMonoidElement', 'is_
Parent',
'is_ParentWithAdditiveAbelianGens', 'is_ParentWithBase',
'is_ParentWithGens', 'is_ParentWithMultiplicativeAbelianGens',
'is_PermutationGroupElement', 'is_PermutationGroupMorphism',
'is_Polynomial', 'is_PolynomialQuotientRing', 'is_
PolynomialRing',
'is_PowerSeries', 'is_PowerSeriesRing', 'is_PrimeField',
'is_PrimeFiniteField', 'is_PrincipalIdealDomain',
'is_PrincipalIdealDomainElement', 'is_ProbabilitySpace',
'is_ProjectiveSpace', 'is_QuadraticField', 'is_QuadraticForm',
'is_RElement', 'is_R_algebra', 'is_Radix64StringMonoidElement',
'is_RandomVariable', 'is_RationalField', 'is_RealDoubleElement',
'is_RealField', 'is_RealIntervalField',
'is_RealIntervalFieldElement', 'is_RealNumber',
'is_RelativeNumberField', 'is_Ring', 'is_RingElement',
'is_RingHomomorphism', 'is_RingHomset', 'is_Scheme',
'is_SchemeMorphism', 'is_SchubertPolynomial', 'is_Set',
'is_SingularElement', 'is_Spec', 'is_Vector', 'is_VectorSpace',
'is_commutative', 'is_even', 'is_field',
'is_fundamental_discriminant', 'is_integrally_closed',
'is_iterator', 'is_odd', 'is_optimal_id', 'is_pAdicField',
'is_pAdicRing', 'is_package_installed', 'is_power_of_two',
'is_prime', 'is_prime_power', 'is_pseudoprime',
'is_pseudoprime_small_power', 'is_square', 'is_squarefree',
'is_triangular_number']
```

Ovakve liste su pogodne za simboličke račune, ali za numeriku su efikasnija numpy polja koja smo već dosta koristili u prošlom poglavlju. Ona su zapisana u neprekinutom slijedu memorijskih lokacija što omogućuje brži pristup no zbog toga je nužno unaprijed deklarirati koji tip objekata su elementi (integer, float, complex ...) i svi elementi moraju biti istog tipa.

```
import numpy as np   # uvođenje skraćenice za ime modula
```

```
a = np.array(t5); print a   # konverzija obične liste u numpy
polje
b = np.array([2., 2.5, 3, 3.5]); print b
```

```
[ 1  2  3  4  5  6  7  8  9 10]
[ 2.   2.5  3.   3.5]
```

Očitavanje tipa objekata u numpy polju:

```
print a.dtype
b.dtype
```

```
int64
dtype('float64')
```

Pri konstrukciji numpy polja izbor tipa objekata je automatski, ali moguće ga je i odrediti opcionalnim argumentom dtype:

```
c = np.array(t5, dtype=np.float64); print c
```

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

Numpy polja imaju dosta više metoda od običnih lista:

```
dir(a)[-66:]
```

```
['any', 'argmax', 'argmin', 'argsort', 'astype', 'base',
'byteswap',
'choose', 'clip', 'compress', 'conj', 'conjugate', 'copy',
'ctypes',
'cumprod', 'cumsum', 'data', 'diagonal', 'dot', 'dtype', 'dump',
'dumps', 'fill', 'flags', 'flat', 'flatten', 'getfield', 'imag',
'item', 'itemset', 'itemsize', 'max', 'mean', 'min', 'nbytes',
'ndim', 'newbyteorder', 'nonzero', 'prod', 'ptp', 'put',
'ravel',
'real', 'repeat', 'reshape', 'resize', 'round', 'searchsorted',
'setfield', 'setflags', 'shape', 'size', 'sort', 'squeeze',
'std',
'strides', 'sum', 'swapaxes', 'take', 'tofile', 'tolist',
'tostring', 'trace', 'transpose', 'var', 'view']
```

(No zbog efikasnosti implementacije nema upravo onih metoda koje imaju obične liste. To je zato jer je npr. umetanje elementa u listu vrlo "skupa" operacija koja uključuje brisanje i pisanje svih elemenata koji u memoriji dolaze nakon tog mjesta umetanja. Ako želimo raditi takve stvari upotreba numpy polja nam ionako neće donijeti nikakve prednosti i treba koristiti obične liste.).

```
b=a.reshape(2,5); print b
```

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

```
b.shape  # "oblik" polja - broj redaka i stupaca
```

```
(2, 5)
```

Pristupanje pojedinom elementu višedimenzionalnog polja je moguće očitim indeksiranjem a[i][j]..., ali je efikasnije koristiti skraćeno indeksiranje: a[i, j, ...]:

```
b[1,1] = 42.
```

```
print b
```

```
[[ 1  2  3  4  5]
 [ 6 42  8  9 10]]
```

```
print a
```

```
[ 1  2  3  4  5  6 42  8  9 10]
```

Pažnja: radi efikasnosti, rezovi kroz numpy polja ne kopiraju originalne elemente već samo daju pokazivače na ista mjesta u memoriji (vidi gore element '42' koji se pojavio i u listi a). Obične liste nemaju takvo ponašanje:

```
t5 = t1[2:4]; t5
```

```
[5, 7]
```

```
t5[0] = 'novi'; print t5
t1
```

```
['novi', 7]
[1, 'tri', 5, 7, 9, 11]
```

Za daljnje detalje o numpy poljima, vidi ovdje.

Konstruiranje lista

Za konstrukciju liste cijelih brojeva najkorisnija je funkcija range():

```
range(-5, 15, 3)
```
```
[-5, -2, 1, 4, 7, 10, 13]
```

range() radi samo sa cijelim brojevima ...

```
range(2., 7.5, 1.5)
```
```
[2, 3, 4, 5, 6]
```

... pa ukoliko trebamo liste realnih (float) brojeva, možemo ili dijeliti ove cijele s nekom odabranom realnom konstantom, ili koristiti numpy inačicu funkcije range() koja se zove arange() ...

```
np.arange(2., 9.9, 1.1)
```
```
array([ 2. ,  3.1,  4.2,  5.3,  6.4,  7.5,  8.6,  9.7])
```

... koju po potrebi možemo i konvertirati natrag u običnu listu funkcijom list():

```
list(_)
```
```
[2.0, 3.1000000000000001, 4.2000000000000002,
5.3000000000000007,
6.4000000000000004, 7.5, 8.6000000000000014, 9.7000000000000011]
```

Kombiniranjem funkcije range() i idioma obuhvaćanja liste možemo kreirati kompleksne stvari. Npr. lista simboličkih izraza:

```
[x^n/factorial(n)  for n in range(1,6)]
```
```
[x, 1/2*x^2, 1/6*x^3, 1/24*x^4, 1/120*x^5]
```

Zadatak 3.1.2:Konstruirajte listu svih prim-brojeva manjih od 100. Koliko ima prim-brojeva manjih od $10^4, 10^5, 10^6, \ldots$? Usporedite rezultat (i brzinu njegovog dobivanja) s ugrađenom funkcijom prime_pi(), te

slijedećom aproksimacijom (koja je sadržaj slavnog teorema o prim-brojevima)

$$\pi(x) = \int_2^x \quad dx$$

Inaće, mjerenje vremena potrebnog da se izvrši neka ćelija izvodi se stavljanjem "%time" u prvi red:

```
%time
factor(29230032746618058364074216492428094683663774517 41)
```

```
120892581961462917470618 9 * 241785163922925834941236 9
CPU time: 0.85 s,  Wall time: 0.88 s
```

(Prekid računa koji traje predugo izvodi se putem menija Action->Interrupt ili pritiskom na tipku "Escape".)

Zadatak 3.1.3: Izračunajte funkciju $\pi(x)$ (broj prim-brojeva manjih od x) za $x = 10^{10}$ statističkom metodom: Izaberite uzorak od n slučajnih cijelih brojeva između 1 i x i testirajte koliko ima prim-brojeva među njima. Iz tog udjela odredite $\pi(x)$. Kolika veličina uzorka vam treba da bi relativna greška prema prime_pi(10^10) razumno često pala ispod 1%?

Ostali kontejneri

Osim lista postoje i drugi "kontejneri" za objekte. Kao prvo tu su već u prošlom poglavlju spominjani tuple-ovi. Glavna razlika prema listama (osim očite razlike da liste idu u uglate, a tuplovi u okrugle zagrade) je da su tuplovi nepromjenjivi - nije moguće promijeniti neki njihov element ili im dodati nove. Ukoliko je to iz nekog razloga nužno treba konstruirati novi tupl:

```
tpl = (1, 3, 5, 7)
tpl[3] = 2
```

```
Traceback (click to the left of this block for traceback)
...
TypeError: 'tuple' object does not support item assignment
```

```
tpl[:2] + (2,) + tpl[3:]
```

```
(1, 3, 2, 7)
```

Ali to je rijetko potrebno. Naime, osim ove razlike u promjenjivosti glavna razlika između tuplova i lista je da su tuplovi obično heterogeni strukturirani skupovi u kojima pojedina mjesta u tuplu nose različita značenja, dok su liste obično homogeni skupovi istovrsnih objekata. Npr, koordinate neke točke je prirodno staviti u tupl (x, y), a ne u listu [x,y], ali niz točaka je prirodno staviti u listu takvih tuplova [(x1, y1), (x2, y2), ...].

Daljnji kontejneri koji nam stoje na raspolaganju su *skupovi* (engl. *set*), koji su skupovi različitih(!) objekata i s kojima možemo raditi standardne stvari poput unije, presjeka, komplementa ...

```
s1 = set([10, 9, 10, 8, 7, 7, 7, 4]); s1
```

```
set([8, 9, 10, 4, 7])
```

(Uočite da se duplikati automatski izbacuju.)

```
print s1.union(t1)
print s1.intersection(t1)
s1.difference(t1)
```

```
set(['tri', 1, 4, 5, 7, 8, 9, 10, 11])
set([9, 7])
set([4, 8, 10])
```

Zadnji, vrlo važan, kontejner kojeg ćemo spomenuti je *riječnik* (engl. *dictionary*). Riječ je o preslikavanju key->value, gdje je value bilo koji objekt, a key može biti tipično broj ili string (premda su i druge stvari dopustive kao key, npr tuplovi).

```
d1 = {'a':1, 'c':3, 'b':2}; print d1
d2 = {1: sin, 2: cos, 3: log}; print d2
d3 = {'ime': 'pero', 'prezime': 'peric', 'spol':'M'}; d3
```

```
{'a': 1, 'c': 3, 'b': 2}
{1: sin, 2: cos, 3: <function log at 0x509d050>}
{'spol': 'M', 'ime': 'pero', 'prezime': 'peric'}
```

Redosljed elemenata u riječniku nema značenja. Pristup pojedinim elementima riječnika je moguć "indeksiranjem" pomoću ključa:

```
d1['c']
```

```
3
```

Na isti način je moguće dodavati nove elemente u riječnik:

```
d2[5] = tan
```

```
d2.keys()
```

```
[1, 2, 3, 5]
```

Iteriranje po riječniku ne daje elemente već samo ključeve:

```
[it for it in d2]
```

```
[1, 2, 3, 5]
```

```
[d2[n](1.) for n in d2.keys()]
```

```
[0.841470984807897, 0.540302305868140, 0.000000000000000,
1.55740772465490]
```

Zadatak 3.1.4: (Nema veze s riječnicima.) Pronađite najveći dvoznamenkasti broj k za koji je $m = 2^k - 1$ prost broj. Ispišite broj m u dekadskom i binarnom sustavu:

```

```

```

```