Bool'i loogika ja informatsiooniteooria

George Boole (1815-1864) - inglise matemaatik

1847 – Loogika matemaatiline analüüs
1854 – Mõtlemise reeglid
Boole'i tegevuse eesmärgiks oli loogikatehete väljaarendamine ja „mõtlemise aritmeetika“ ehitamine. Ta andis matemaatilise kuju lausearvutusele. Tähistused 1 ja 0 pärinevad just Boole'ilt.
Boole'i loogika ehk **Boole'i algebra** põhisisu:

- Loogikaväited koosnevad lihtsamatest lausetest, mis on omavahel seotud loogikaseoste ehk tehetega (ja; või; ei; kui ..., siis ..., jne).
- Lausearvutuse seoste korral määravad osalausete tõeväärtused kogu lause tõeväärtuse, osalausete konkreetne sisu ei ole aga tähtis.
- Lausearvutuse tehteks nimetatakse niisugust lausetes kasutatavat seost, mille tõeväärtus on tema osalausete tõeväärtuste funktsioon (st sõltub osalausete tõeväärtustest) (e. Boole'i funktsioon).
- Lausearvutuse tehe on defineeritud tõeväärtustabeliga (nn Boole'i funktsiooni graafik).
- Tõeväärtused: Tõene – T, 1 (true) ja väär – F,0 (false)

Claude Elwood Shannon (1916 – 2001)

Elektroonilise infoajastu isa
1936 kaitses MIT-is magistritöö *("A Symbolic Analysis of Relay and Switching Circuits")*, milles käsitles Booli loogika rakendamist elektriskeemidele / lülitustele (loogikavõrkudele). Selleks ajaks oli Boole'i tööd loogika vallas unustatud ja Shannon kaevas need välja ning seostas elektriskeemidega. Shannon näitas ka, et põhimõtteliselt on võimalik ehitada loogikamasinat.
Sõja ajal tegeles muuhulgas krüptograafiaga - *"A Mathematical Theory of Cryptography"* (huvi kommunikatsiooniprobleemidele)

1948 – **informatsiooniteooria** alus publikatsiooniga *"A Mathematical Theory of Communication"*.
1949 – sama nimega raamat kahasse Warren Weaveriga.

Kuidas kvantitatiivselt mõõta informatsiooni?
Mõõtmise alus – "jah/ei" situatsioon, mis sisaldab **1 bitt** *(binary digit)* informatsiooni. Shannoni „loogikamasinas“ tähednaks 1, ei vooluring on suletud ja vool on sees, ning 0, et vooluring on avatud ja vool väljas.
Keerulisema informatsiooni (kus on rohkem infoühikuid) saab ülesehitada mitut bitti kasutades.
Entroopia - informatsiooni puudujääk teate sisus. Kui korrastamata süsteemi tuleb juurde informatsiooni, väheneb entroopia ja süsteem muutub korrastatumaks.

Kommuniaktsioonisüsteemi põhielemendid:
- **infoallikas** (*source*) koos edastusseadmega, mis muundab info või teate vormi sobivaks, et edastada ta kanalis
- **kanal** (*channel*) või vahend mida mööda toimub info ülekanne *(transmission)*
- **vastuvõtja** (*receiver*) dekodeerib teate originaalilähedasele kujule
- **siht** (*destination*), kes või mis peab teate vastuvõtma

- **müra allikas** (*source of noise*), mis muudab teadet ülekande jooksul etteaimamatult

Info mõõtmisel pole informatsiooniteoorias mingit seost teate sisuga.
Mõõdetavad suurused on järgmised:
- **sagedus**, millega allikas infot tekitab
- kanali **mahtuvus**/suutlikus informatsiooni edastada
- vastavat tüüpi teates sisalduva **informatsiooni hulk**

1949 a raamat *„A Mathematical Theory of Communication“* (koos Warren Weaver'iga).
Shannoni jaoks oli informatsiooniteooria ainult tehniline probleem. Weaver seostab informatsiooniteooria inimeste-vahelise kommunikatsiooniga, kus on kolm tasandit:
- **A tasand** - kui adekvaatselt edastatakse kommunikatsioonisümboleid (tehniline probleem)
- **B tasand** - kui täpselt edastatavad sümbolid kannavad ihaldatud tähendust (semantiline probleem)
- **C tasand** - kui efektiivselt kätte saadud teade mõjutab saajat soovitud suunas (efektiivsuse või käitumuslikkuse probleem)

Positsioonilised arvusüsteemid

Kahendsüsteem *(Binary)*

Enamus kaasaegseid arvuteid kasutavad kahendloogikat, seepärast on vajalik mõista ka kahendsüsteemi.
Kahe seisundi märkimiseks: 0 ja 1.
Arvusüsteemi alus on 2 ja igal positsioonil arvus on oma kaal (2 aste).
Näide: `1100 1010`=$1*2^7 + 1*2^6 + 0*2^5 + 0*2^4 + 1*2^3 + 0*2^2 + 1*2^1 + 0*2^0$=
`1*128 + 1*64 + 0*32 + 0*16 + 1*8 + 0*4 + 1*2 + 0*1=`**`202`**

Teisendamine
10ndsüsteem -> 2ndsüsteem
Kasutatavad on kaks võtet – lahutamine ja jagamine
Jagamisel kasutatakse kahega jagamist ja jäägi leidmist:

```
202 | 0
101 | 1
 50 | 0
 25 | 1
 12 | 0
  6 | 0
  3 | 1
  1 | 1
  0
11001010
```

Kaheksandsüsteem *(octal)*

Arvusüsteemi alus on 8 ja igal positsioonil on oma kaal – 8 vastav aste. Kasutatavad sümbolid on `0,1,2,3,4,5,6,7`.

10ndsüsteem -> 8ndsüsteem
Jagamine (vt kahendsüsteemi, kuid jagatakse ja leitakse jääk 8-ga)

2ndsüsteem -> 8ndsüsteem
Kahendarv jagatakse paremalt alates kolmikuteks ja igale kolmikule seatakse vastavusse 1 8ndsüsteemi arv.

```
011 001 010 -> 312
```

Kuueteistkümnendsüsteem *(hexadecimal)*

Arvusüsteemi alus on 16, iga positsiooni kaal on 16 vastav aste. Kasutatavad sümbolid on:
`0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F`

16ndsüsteemi eelised:
1. on kompaktne (võrreldes kahendsüsteemiga)
2. on kergesti teisendatav kahendsüsteemi (võrreldes kümnendsüsteemiga)

10ndsüsteem -> 16ndsüsteem
Jagamine (vt kahendsüsteemi, kuid jagatakse ja leitakse jääk 16-ga). See arvutamine on peast üsna tülikas.

2ndsüsteem-> 16ndsüsteem
Kahendarv jagatakse paremalt alates nelikuteks ja igale nelikule seatakse vastavusse 1 16ndsüsteemi arv.
`1100 1010 -> CA`

Kahendarvudest moodustatavad struktuurid
Mõisteid ja reeglid:

1. 0-de lisamine struktuuri algusesse ei muuda tema väärtust
2. kõige parempoolsem bitt kannab numbrit 0 ja nime **LSB** *(least significant bit)* - **madalaim bitt**, vähima kaaluga bitikoht positsioonilises süsteemis
3. iga järgnev bitt kannab ühe võrra suuremat järjenumbrit
4. kõige vasakpoolsem bitt kannab nime **MSB** *(most ignificant bit)* - **kõrgeim bitt**, suurima kaaluga bitikoht positsioonilises süsteemis

Bitt (*binary digit*, lühend b) on vähim infoühik, millega saab näidata kahte (2^1) võimalust/olekut.

Nibble - 4-bitine kombinatsioon bittidest. Saab kujutada 2^4 kombinatsiooni ehk ühe 16ndarvu.

Bait (*byte*, lühend B)- olulisim struktuur praeguste mikroprotsessorite juures (vähim adresseeritav üksus). Ristiisa 1956 a. *dr. Werner Buchholz* (töötas IBM-is).
`1 bait = 2 nibble = 8 bitti`
1 baidiga saab esitada $2^8=256$ erinevat kombinatsiooni.
Kasutamine:
- märgita täisarv: 0 .. 255
- märgiga täisarv -128 .. +127
- ASCII kooditabeli sümbol (kaaluta tabel) - põhitabel + laiendatud tabel

Sõna (*Word*) – Fikseeritud pikkusega bittide grupp, mida arvutis korraga töödeldakse. Bittide arv sõnas on arvuti arhitektuuris oluline näitaja. Tavaarvutites on praegu sõna suurus 32 või 64 bitti. Kuid sardsüsteemides kasutatavates arvutites ka väiksem (alates 8 bitti)
16-bitise sõnaga saab kujutada $2^{16}=65\ 536$ erinevat väärtust:
- märgita täisarvu `0 .. 65 536`
- märgiga täisarvu `-32 768 .. +32 767`

32-bitine sõna on 4 baiti ja seega 2^{32} erinevat väärtust

Mahtude jms mõõtmisel on aluseks tihti bait. Kasutatakse kilobait-i, megabaiti jne..

Mis on aga kibibait, mebibait, gibibait ja tebibait?
Standardi kohaselt on seosed järgmised:

Binaarühik	Väärtus	SI ühik	Väärtus
kibibait (KiB)	2^{10}	kilobait (kB)	10^3
mebibait (MiB)	2^{20}	megabait (MB)	10^6
gibibait (GiB)	2^{30}	gigabait (GB)	10^9
tebibait (TiB)	2^{40}	terabait (TB)	10^{12}
pebibait (PiB)	2^{50}	petabait (PB)	10^{15}
eksbibait (EiB)	2^{60}	eksabait (EB)	10^{18}
zebibait (ZiB)	2^{70}	zettabait (ZB)	10^{21}
yobibait (YiB)	2^{80}	yottabait (YB)	10^{24}

Ehk siis erinevalt varasemat ajastust tähistatakse eesliitega kilo 10^3 kordset suurust (nagu SI-süsteem seda ettenäeb). Eesliitega kibi- aga 2^{10} kordset suurust, mis vastab nn binaarühikutele (kibibyte - *kilo binary byte*)

Erinevad andmetüübid arvuti mälus

Kuidas andmeid arvuti mälus hoitakse?
Kui suur saab olla arv, mida arvuti suudab õigesti salvestata?
Millise täpsusega arvuti arvutab?
Kuidas hoida arve nii, et arvutusi oleks lihtsam realiseerida?

Lihtandmetüüpide kujutamine sõltub nii arvutist kui ka keelest ja kompilaatorist.

Täisarv - *Integer*

Täisarvude esitamise võimalused:

- **Märgita esitus** *(Unsigned integer representatsion)* - mittenegatiivse täisarvu esitamine positsioonilises kahendsüsteemis.
- **Märk suurusjärguga** *(Sign magnitude)* - 1 bitt (MSB) määrab märgi (0 +, 1 -), ülejäänud bitid annavad suurusjärgu.
- **Täiendesitus** *(Complement representation)* - positiivsed täisarvud on kahenkujul. **Esimese täiendväärtuse** *(One's complement)* puhul on negatiivsetel arvudel 1 MSB-s ja teised saadakse bitt haaval inverteerides. **Teine täiendväärtus** *(Two's complement)* saadakse 1. täiendväärtusele 1 juurde liitmisel.
- **Esitus nihkega** *(Biased representation)* - arvud kujutatakse kui positiivsed märgita arvud, arvu M tegelik väärtus leitakse M-nihe *(bias)*. Nihe on N-bitise arvu korral 2^{N-1} või 2^{N-1}-1.

Näide 3 bitiga:

bitid:	100	101	110	111	000	001	010	011
1. täiendv.:	-3	-2	-1	0	0	1	2	3
2. täiendv.:	-4	-3	-2	-1	0	1	2	3

negatiivsed arvud nihkuvad ühe koha võrra allapoole, et vältida kahte 0-i.

Täisarvu jaoks kasutatakse enamasti **teise täiendväärtusena** esitamist.

Ujukomaarv - float

Sellist arvu saab tegelikus elus esitada ja kasutada vaid teatud täpsusega ε (epsilon) `(arv-epsilon<=arv<=arv+epsilon)`
Kokkulepitud komakohtade arvu puhul on tegemist **arvu püsikomaesitusega** (*fixed-point representation*).

Kasutusel on **arvu ujukomaesitus** (*floating point representation*).
Tegelik arv koosneb:

- **märk** (0 – positiivne, 1 – negatiivne)
- **mantissi** absoluutväärtus (annab täpsuse, koma paikneb vasakult 1. ja 2. positsiooni vahel)
- astme suurus e **eksponent** (nihkega esitus, et kujutada nii positiivset kui ka negatiivset astet, aste=tegelik_aste + nihe)

IEEE standardi 754 järgi:

lihttäpsusega ujukomaarv (*single precision*):
suurus 32 bitti (1 bitt märgile, 8 bitti eksponendile nihe 127, 23 bitti mantissile)
piirid: 1.5E-45 .. 3.4E38 täpsus: 7-8 kohta
Koma on tüvenumbites vasakpoolse arvu järel, eksponendi esitamiseks kasutatakse täisarvu nihkega esitust.

topelttäpsusega ujukomaarv (*double precision*):
suurus 64 bitti (1 bitt märgile, 11 bitti eksponendile nihe 1023, 52 bitti mantissile)
piirid: 5.0E-324 .. 1.7E308 täpsus: 15-16 kohta
Koma on tüvenumbites vasakpoolse arvu järel, eksponendi esitamiseks kasutatakse täisarvu nihkega esitust.

Tõeväärtus – *Boolean*

2 väärtust 0 ja 1, salvestatakse 1 baidile.

Andmete teisendamisel ühest tüübist teise võib toimuda andmekadu. Tuleb olla tähelepanelik!!

Tekstilise info kodeerimine ja kooditabelid

Kooditabelite ülesanne on luua seos arvutikoodide (bitijadade) ja erinevate tähemärkide jt märkide vahel, mida kasutatakse teksti kirjutamisel. Kodeerimine lubab digitaalsetel seadmetel tekstandmeid salvestada, töödelda ja vahetada.

ASCII

ASCII *(American Standard Code for Information Interchange)* on vanimaid kasutusel olevaid kooditabeleid. Tabel tugineb inglise alfabeedile.

Esimene standardi versioon avaldati 1963. ASCII tabel on 7-bitine kood, ta kasutab iga sümboli kodeerimiseks mustrit 7-st bitist. Kaheksas bitt oli andmete ülekandmise kontrolliks paarsusbitt *(parity bit)*. Seega ASCII tabel kirjeldab/kodeerib 128 märki (miks?), millest 32 esimest ei ole trükitavad.

Laiendatud ASCII

Rahvuslike tähtede lisamiseks on võetud kasutusele 8. bitt ja räägitakse **laiendatud ASCII tabelist** *(extended ASCII)*. Kui tabeli esimesed 128 sümbolit on alati ühesugused, siis tagumised 128 varieeruvad, sest vajadused erinevate tähtede järele on kultuuriliselt erinevad.

Neid erinevaid variante kutsutakse **koodilehekülgedeks** *(code pages)*, nimi on IBM-lt, kes IBM PC jaoks rahvuslik-kultuurilised laiendused algatas.

Teistsuguse standardi lõi ISO: **ISO 8859**, milles oli kirjeldatud teistsugune 8-bitine kooditabel. Alumine osa (128 märki) on samad. Populaarseim - **ISO 8895-1**, ka ISO Latin-1 (tähed Lääne-Euroopa keelte jaoks).

ISO 8895-2 sobib Ida-Euroopa keelte jaoks.

MS tegi oma standardi ja andis talle nimeks ***code page 1252***. See on suures osas vastavuses ISO 8895-1-ga.

Unicode

Standard, mis püüab likvideerida erinevatest kodeeringutest tulenevaid konflikte ja kodeerida kõikvõimalikud sümbolid, milles teksti esitatakse. Tabelisse on reserveeritud $2^{20} + 2^{16} = 1\ 114\ 112$ positsiooni. Nendest esimesed 256 on vastavuses ISO 8895-1-ga.

Unicode peaks võimaldama ühekorraga kasutada rohkem kui paari keele sümboleid.

Kogu kooditabelit ei ole korraga vaja kasutada ja seetõttu võetakse sellest välja osasid. On kaks **vastavuste tekitamise meetodit** *(mapping)*: *Unicode Transformation Format (UTF)* kodeering ja *Universal Character Set (UCS)* kodeering.

Konkreetsesse kodeeringusse võetakse fikseeritud suurusega ja fikseeritud piirkonnast kodeeritud sümbolid. Numbrid kodeeringute nimedes tähendavad vastavalt kasutatavate bittide (UTF) või baitide (UCS) arvu. Arvatavasti on **UTF-8** ja **UTF-16** ühed tuntumad skeemid. Näiteks UTF-8 on 8-bitine, kuid muutuva pikkusega kodeering maksimaalselt vastavuses ASCII-ga. UTF-8 kasutab 1 kuni 4 baiti sümboli kodeerimiseks.

Pythoni andmetüübid

Täisarv (integer)

Negatiivsed ja positiivsed. Pythonis on kolme tüüpi täisarve: *plain integers, long integers, booleans.*

Plain integer on suuruste vahemikus -2147483648 kuni 2147483647. Võivad olla suuremad, sõltudes arvuti sõna (word) suurusest. Praegu on see valdavalt 32-bitine. Viga, mis võib täisarvudega arvutamisel juhtuda, on ületäitumine (erind `OverflowError`). Tavaliselt teisendatakse tulemus *long integer'iks.*

Long integers on väidetavalt piiranguteta. Loomulikuks piiranguks vaid arvuti mälumaht. Ka seda andmetüüpi hoitakse teise täiendväärtuse.

Booleans kasutatakse loogikaväärtuste *true* ja *false* esitamiseks. Ta on *plain integer'i* alamtüüp ja väärtused on 0 ja 1. Kui neid väärtuseid on vaja näidata, siis teisendatakse stringideks 0 ja 1.

Ujukomaarv (Floating point number)

Kujutatakse masina tasemel IEEE standardi topelttäpsusega ujukomaarvudena. Dokumentatsiooni

kohaselt määravad arvu täpsuse ja piirid, samuti käitumise ületäitumiste korral masina arhitektuur.

Kompleksarv (Complex numbers)

Masina tasemel esitatakse kahe topelttäpsusega ujukoamarvu paarina. Kõik, mis kehtib ujukomaarvu kohta, kehtib ka siin. Reaal- ja imaginaarosa saab lugeda read-only atribuutide `z.real` ja `z.imag` kaudu.

Struktuursed andmetüübid

Struktuurset tüüpi muutuja saab sisaldada mitut väärtust korraga. Klassikalisteks struktuurseteks andmetüüpideks on **massiiv** *(array)* ja **kirje** *(record)*.

Massiiv võimaldab salvestada ühte tüüpi andmeid. Andmetele liigpääs tagatakse indeksite abil. Massiivid võivad olla ühemõõtmelised. Kahemõõtmelised jne ...mõõtmelised. Sõltub keelest, kuidas massiive kasutada saab arvutusteks, sisestamiseks, väljastamiseks – st kas on olemas käsud kogu massiiviga töötamiseks või tuleb elemente ükshaaval töödelda.

Massiivid võivad olla staatilised või dünaamilised.

Staatiline massiiv – elementide maksimaalne arv määratakse programmi töö alguses, hiljem seda suurendada ei saa.

Dünaamiline massiiv – elementide arv massiivis on muudetav (tavaliselt lisatakse elemente juurde)

Kasutatavast keelest sõltub ka indekseerimise tava. C-tüüpi keeled määravad alati massiivi indeksid 0-st. Pascal lubab kirjeldada massiivile suvalise indeksite vahemiku, tavaliseks on algus 1-st.

Tüüpiliselt kasutatakse indeksite eristamiseks massiivi nime taga kandilisi sulge `[]`

Näide:

On massiiv `nimed`, lubatud salvestada stringe.

`nimed[2]` – tähendab massiivi 2. või 3. elementi ehk 2. nime või 3. nime. Sõltub indekseerimise kokkulepetest antud keeles (kas esimene indeks on 1 või 0)

`nimed[2] = "Juuli"` - 2. indeksiga tähistatud lahtrisse kirjutatakse Juuli.

Massiivis on N elementi, all on indeksid

Maali	Juuli	Tuuli	Meeli	...	...	Aali	Raali
1	2	3	4	...	...	N-1	N

Alternatiivne indekseerimine:

Maali	Juuli	Tuuli	Meeli	...	...	Aali	Raali
0	1	2	3	...	...	N-2	N-1

Massiivi on sobiv kasutada loogilises mõttes ühte tüüpi/sorti andmete hoidmiseks, kus töötlemisel on vaja kõigi andmetega samu toiminguid teha (nimed, pikkused, hinded jms).

Kirje võimaldab salvestada ja töödelda erinevat tüüpi andmeid. Kirjesse on tavaks koondada info sama objekti kohta. Näiteks üliõpilase eesnimi, perekonnanimi, õpinguraamatu number, sünniaasta ja eriala.

Kirjes olevaid positsioone andmete salvestamiseks nimetatakse **kirje väljadeks** *(field)*. Igale väljale antakse nimi, mille kaudu väljas olevat infot käidelda saab.
Igal kirje väljal on tüüp, mis kuulub keele standardtüüpide (liht- või struktuursete) hulka. St kirje väljaks võib olla teine kirje või massiiv, täis- ja ujukomaarvudest rääkimata.

Süntaks ja mängureeglid kirje väljade käitlemiseks on keeleti erinevad.

Näiteks:

`opilane.eesnimi`

`opilane->eesnimi`

(kus `opilane` on kirje nimi ja `eesnimi` on välja nimi)

Pythoni jadad

Pythoni struktuurseteks andmetüüpideks on **jadad** *(sequence)*, mida on 3 tüüpi.

- string
- list
- ennik *(tuple)*

Jadad on organiseeritud järjestikku ja jada elementidele pääseb ligi indeksi kaudu. Mitmed tehted ja toimingud töötavad kõigi jadade puhul ühte moodi.

Jada elemente indekseeritakse kahel erineval viisil:

`0 kuni N-1` (kus N on jada elementide arv) ja

`-N kuni -1`

Maali	Juuli	Tuuli	Meeli	...	...	Aali	Raali
0	1	2	3	...	...	N-2	N-1
-N	-N+1	-N+2	-N+3	...	...	-2	-1

Lõiked

1 element: `jada[X]`
elementide järjendid:
`jada[X1:X2]` elemendid indeksite vahemikus X1 (kaasaarvatud) kuni X2 (väljaarvatud)
`jada[X1:]` - indekist X1 kuni lõpuni
`jada[:X1]` – algusest kuni X1-ni

Kordamine

`jada * täisarv` – jada elemente korratakse täisarv korda, tekib uus jada

Liitmine

`jada1 + jada2` – jada1 ja jada2 ühendatakse uueks jadaks, kus alguses on jada1 elemendid ja seejärel jada2 elemendid.

Sisaldumise kontroll

```
element in jada
element not in jada
```

Tehte tulemuseks on tõeväärtus *true* või *false* vastavalt sellele, kas küsitud element kuulus jadasse või mitte.

Võrdlemine

kehtivad erinevad võrdlustehted. Tekstide võrdlemine toimub ASCII tabeli kohaselt, a < b jne. Arvude võrdlemine on nö tavaline.

Sisefunktsioonid

Jadadele saab rakendada funktsioone (näited):

`len(jada)` – jada elementide arv
`max(jada)` – jada suurima elemendi väärtus
`min(jada)` – jada vähima elemenid väärtus
`str(asi)` – teisendus stringiks
`list(asi)` – teisendus listiks
`tuple(asi)` – teisendus ennikuks
jne

String

String on suvaliste sümbolite jada. Stringi elementideks on **märgid** *(character)*. Pikkus ei ole piiratud. Stringi eraldajateks on jutumärgid `("")` või ülakomad `('')`.

Moodul `string` sisaldab sisefunktsioone stringide töötlemiseks.
Stringid on **muudetamatud** *(immutable)*, st stringi muutmiseks tuleb moodustada uus string ning stringis ühte sümbolit teisega lihtsa omistamise abil asendada ei saa.

String on **objekt** ning Pythonis on mitmed **meetodid**, mida stringi töötlemiseks kasutada saab:

```
    nimi =  "maali maasikas"
    nimi = nimi.capitalize() # Esimene täht suureks
```

Stringi elementide eraldamine sümbol haaval:

```
# Kasutatakse indeksit, järjest kõik tähed eraldada,
# indeksi piirid on 0 kuni stringi pikkus
s6na = 'abcde'
for i in range(0, len(s6na)):
     print s6na[i]

# Järgnev kood on "pythoonilisem", indeksita
s6na = 'abcde'
for t2ht in s6na:
    print t2ht
```

List

Listi elementideks on suvalised andmed, ka teised listid. Elemendid võivad kõik olla erinevat tüüpi. Listi eraldajateks on kandilised sulud `[]`. Samuti on listi indeksi eraldajateks kandilised sulud.

Näide:

```
asjad=['hobune',12,3.45,'pann',['list','listis']]
```

List on **muudetav** *(mutable)*. Teda saab muuta elemente asendades, lisades ja kustutades. Asendamisel võib muutuda ka elemendi tüüp.

Sarnaselt stringile saab listi vaadelda ka kui objekti ja kasutada meetodeid tema töötlemiseks.

Otstarbekas on listi kasutada kui massiivi (ehkki Pythonil on ka erimoodul massiivi tegemiseks), st hoida ühes listis loogilises mõttes sama tüüpi andmeid.

Ennik

Ennik *(tuple)* on sarnane listile, kuid ta on **muudetamatu** *(immutable)*. Seega sobib teda kasutada konstandina, st hoida seal väärtuseid, mida programmis korduvalt vaja kasutada on, näiteks kuude nimesid.

NB! 1 kilobait ei ole 1000 baiti vaid 1024 baiti.

2nd, 8nd, 10nd ja 16nd süsteemide vastavus

Binary	Octal	Decimal	Hexdecimal
0000B	00Q	00	00H
0001B	01Q	01	01H
0010B	02Q	02	02H
0011B	03Q	03	03H
0100B	04Q	04	04H
0101B	05Q	05	05H
0110B	06Q	06	06H
0111B	07Q	07	07H
1000B	10Q	08	08H
1001B	11Q	09	09H
1010B	12Q	10	0AH
1011B	13Q	11	0BH
1100B	14Q	12	0CH
1101B	15Q	13	0DH
1110B	16Q	14	0EH
1111B	17Q	15	0FH
10000B	20Q	16	10H