Eesti Teadusagentuur
Estonian Research Council
TeaMe
Euroopa Liit
Euroopa Sotsiaalfond
Eesti tuleviku heaks
TTÜ
informaatikainstituut

Tutvumine Pythoniga

Python on lihtne kuid võimas programmeerimiskeel, mis leiab üha laiemat kasutamist väga erineva iseloomuga rakenduste loomiseks. Tegemist on vabavaralise tarkvaraga.

Pythonit kasutatakse sageli ka programmeerimise õpetamiseks koolides ja ülikoolides.

Sisu

Sissejuhatus Pythonisse

Python on üldotstarbeline, objektorienteeritud, väike, võimas, lihtne ja lõbus – nagu väidavad keele loojad ja arendajad – vabavaraline programmeerimiskeel, mis on loodud 1991. aastal. Autor on Guido van Rossum Hollandist. Nimi on võetud inglise koomikute grupi Monty Python nime järgi.

Python leiab laialdast kasutamist erinevat liiki tarkvara loomisel, muuhulgas ka veebirakenduste juures ning dokumendipõhistes rakendustes. Kasutamise ulatuselt on ta võrreldav PHP ja Visual Basicuga. Neist kõrgemal on vaid sellised keeled nagu Java ja C-pere keeled (C, C++, C#), mis on eeskätt süsteem-programmeerimise keeled.

Üheks oluliseks Pythoni omaduseks on lihtsus. Selles osas on ta võrreldav Scratchi, Visual Basicu ja VBAga. Keelel on väga lihtne süntaks, milles puuduvad igasugused spetsiifilised eraldajad lausete struktuuri määramiseks võtmesõnade või looksulgudega, nagu näiteks Pascalis, C-s, Javas või mitmetes teistes keeltes. Lausete struktuur on selge ja kompaktne, selles ei ole väärtuste ja muutujate deklareerimist ega struktuurandmete jäiga ja fikseeritud struktuuri kirjeldamist, mis on iseloomulik enamikule programmeerimiskeeltele. Andmetüüpide ja andmestruktuuride käsitlemine on lihtne ja dünaamiline. See on põhjuseks ka Pythoni üha ulatuslikumas kasutamises programmeerimise algkursuste esimese keelena.

Lausete ja võtmesõnade hulk keeles on üsna väike. Suur osa tegevusi määratakse funktsioonide ja objektide meetodite abil. Süsteemi tuuma on sisse ehitatud rida **standardmooduleid**, mis sisaldavad suurt hulka väga erineva otstarbega funktsioone ja protseduure, klasside määranguid ning meetodeid. Väga lihtsalt saab kasutada ka lisamooduleid, mis on koostatud erinevate autorite ja firmade poolt. Kasutaja saab ka ise koostada oma funktsioone ja klasse ning moodustada neist oma mooduleid.

Pythoni rakendustes saab kasutada erinevat liiki andmeid: märkandmed (arvud, tekstid, ajaväärtused ja tõeväärtused), graafikaandmed (pildid, skeemid, joonised), heliandmeid ja multimeedia vahendeid, lugeda andmeid andmebaasidest ja kirjutada neid andmebaasi. Lihtsalt saab kasutada erineva organisatsiooniga andmeid: loendid, massiivid, maatriksid, sõnastikud, failid jms. Võimalik on luua ja töödelda erineva struktuuriga dokumente.

Töö programmide sisestamise, redigeerimise ja silumisega toimub lihtsas ja hõlpsasti käsitlevaks kasutaja-liideses. Programme saab koostada ja redigeerida ka suvalise tekstiredaktoriga.

On mitmeid Pythoni realisatsioone. Peamine ja enamkasutatav on CPython, mis on realiseeritud programmeerimiskeeles C ja töötab enamikes operatsioonisüsteemides: Windows, Linux, Unix, Mac Os, ... Keele arendamisega tegeleb ühing Python Software Foundation (PSF). Veebilehelt http://python.org/ saab Pythoni alla laadida ning seal on ka suurel hulgal viiteid abi- ja õppematerjalidele. CPythoni jaoks arendatakse kahte haru: **Python 2** (momendil viimane versioon 2.7) ja **Python 3** (viimane versioon 3.3), mis on küll teatud määral erinevad, kuid erinevused ei ole eriti suured.

Antud materjal on orienteeritud keelele Python 3. See on osa kursusest „Rakenduste loomise ja programmeerimise alused“ ja ei ole mõeldud päris algajatele. Eeldatakse, et õppur oskab juba teatud määral programmeerida, näiteks Scratchi või BYOB (SNAP) abil. Pythoni kursust võib vaadelda kui Scratchi kursuse järge. Üldiselt ei ole Scratchi tundmine siiski tingimata vajalik, kuid see on kasulik. Arvestame, et lugeja tunneb selliseid programmeerimise põhimõisteid nagu muutuja, avaldis ja omistamine, objekt ning selle omadused ja meetodid, protsesside juhtimine: jada, kordus ja valik. Pythoni olemust ja põhielemente püüame selgitada näidete varal. Soovitame lugejal neid Pythonis proovida, katsetada, uurida ja täiendada. Materjalis ei käsitleta kaugeltki kõiki Pythoni võimalusi, tegemist on sissejuhatusega sellesse imelisse maailma.

Pythoni kohta on üsna palju õpikuid paberkandjal ja loomulikult veebimaterjale. Arvestades keele kiiret arengut ja internetis üha laiemalt levivaid interaktiivse õppimise tehnoloogiaid, olgu järgnevalt nimetatud mõned õppevahendid.

Põhjalik ja mahukas, kuid sujuvalt algav ja pidevalt uuendatav, õppematerjalide komplekt. Peter Wentworth, Jeffrey Elkner, Allen B. Downey, and Chris Meyers. How to Think Like a Computer Scientist. Materjalid on HTML- ja PDF-vormingus, lisaks näidete komplektid.

Eelnevaga on otseselt seotud interaktiivne kursus: Brad Miller and David Ranum, How to Think Like a Computer Scientist: Interactive Edition (Using Python 3.x). Tekst põhiliselt eelmisest materjalist. Ineraktiivne Pythoni interpretaator harjutuste koostamiseks ja täitmiseks otse veebidokumendis, ekraanivisioonid, enesetestid jm.

Standfordi Ülikooliga seotud avatud veebikursuste süsteem Udacity. David Evans. Intro to Computer Science (CS101). Building a Search Engine. Pythonil põhinev intensiivne interaktiivne kursus. Ekraanivisioonid, Interaktiivne Pythoni interpretaator, enesetestid jms.

Khan Academy. Computer Science. Introduction to programming and computer science. Ekraanivisioonidel põhinev Pythoni lühikursus.

Kiirtutvus Pythoniga

See jaotis on programmi struktuuri ja komponentide tutvustamiseks.

Programmi struktuur ja komponendid

Näide: Tutvus

Programm **Tutvus** küsib kasutajalt nime ning seda, kas ta tahab lasta leida oma keha indeksi: mass/pikkus^2, mass (kaal) – kg, pikkus – m. Kui kasutaja sisestab vastuseks numbri "**0**", väljastab programm kahetsuse ja lõpetab töö. Kui kasutaja sisestab väärtuse "**1**", küsib programm kasutaja pikkuse ja massi ning leiab ja kuvab kehamassi indeksi ja hinnangu selle kohta. Paremal on toodud algoritmid Scratchis.

```
def saledus(L, m): # funktsioon
  """ Annab massiindeksi alusel hinnangu
   toitumusaste kohta. Parameetrid
   L - pikkus, m - mass """
  indeks = m / (L / 100) ** 2
  if indeks < 18:
     hinnang = "kõhn"
  else:
     if indeks <= 25:
        hinnang =  "normis"
     else:
        if indeks <= 30:
           hinnang =   "ülekaaluline"
        else:
           hinnang =  "suur ülekaal"
  return hinnang
# Tutvus. Peaprotseduur
print("Tere! Olen Python!")   # teate kuvamine
nimi = input('Sinu nimi => ')  # nime lugemine
v = input("Leian kehamassindeksi (1-jah / 0-ei )?  ")
if  v == "0" :  # valiku algus
   print ("Kahju! See on jube kasulik info!")
   print (nimi + "! Ehk mõtled veel? See on tasuta!")
else:          # alternatiiv
   pikkus = int(input(nimi + "! Sinu pikkus (cm) => "))
   mass = float(input("aga mass/kaal (kg) => "))
   indeks = round(mass / (0.01 * pikkus) ** 2, 1)
   print (35 * "=")  # prindib "joone" 35-st märgist "="
   print (nimi + "! Sinu massiindeks on:" + str(indeks) )
   print (nimi,  "! Oled ", saledus(pikkus, mass) )
print ('Kohtumiseni! Igavesti Sinu, Python!')
```

Soovitame lugejal kopeerida toodud programm Pythoni kasutajaliidesese ja katsetada seda ise. Kindlasti tuleks üle vaadata, kas liitlausetes (plokkides) on kõigil käskudel ühesugune taane.

Allpool on toodud mõned selgitused.

Programm ehk moodul koosneb üldjuhul mitmest üksusest: skriptist, protseduurist, funktsioonist. Näites koosneb programm kahest protseduurist ehk skriptist: **peaprotseduur** ja **alamprotseduur** ehk **funktsioon**. Protseduurid koosnevad **lausetest** (käskudest) ja **kommentaaridest**.

Kommentaarid algavad sümboliga **#**, võivad asuda eraldi real või rea lõpus ja ei avalda mingit mõju programmi täitmisele. Kommentaariks võib pidada ka kolmekordsete jutumärkide (*"""*) või ülakomade (*'''*) vahel asuvaid stringe (tekste), nagu praegu on funktsiooni **saledus** alguses.

Laused on korraldused/käsud, mis määravad vajalikke tegevusi. Lausetel on keele reeglitega määratud kindel otstarve ja struktuur. Need võivad sisaldada

- **võtmesõnu**: kindla asukoha, kuju ja tähendusega sõnad (if, else, return, def),
- **funktsiooniviitasid**: saledus(...), round(...), print(...), input(...),
- **string**- ja **arvkonstante**: *"hinnang"*, *"Sinu nimi => "*, 100, 0.01 jms,
- funktsioonide, muutujate ja parameetrite **nimesid**: saledus, round, v, pikkus, L, m,
- **avaldisi**: m / (L / 100) ** 2, mass / (0.01 * pikkus) ** 2, 35 * "=", indeks < 18,
- **operaatoreid** (tehtemärke): +, *, =, ==, <=,
- **piirajaid** ja **eraldajaid**: ( ) , ” : .

NB! Pythonis eristatakse suur- ja väiketähti.

Programmi (mooduli) täitmine algab peaprotseduurist. Funktsioon alustab tööd (käivitub), kui toimub pöördumine selle poole. Peaprotseduur asub lõpus, selle algust ja lõppu kuidagi ei tähistata.

Funktsioon algab alati päisega:

def *nimi* (parameetrid):

millele järgnevad järgmistel ridadel taandega protseduuri keha laused.

NB! päise lõpus peab tingimata olema koolon (:). Parameetritega esitatakse funktsiooni sisendandmed.

Praegu on funktsiooni päis järgmine:

def saledus(*L*, *m*)**:**

Funktsiooni nimeks on **saledus**, parameetriteks: L – pikkus (cm) ja m - mass (kg).

Väga sageli leiab ja tagastab funktsioon ühe või mitu väärtust. Väärtuste tagastamiseks kasutatakse **return**-lauset, mis lõpetab funktsiooni töö ning tagastab tulemuse(d) ja täitmisjärje kohta, kust toimus pöördumine funktsiooniviida abil. Funktsiooniviit esitatakse kujul: ***nimi*** (*argumendid*), kus ***nimi*** on funktsiooni nimi ning ***argumendid*** annavad parameetritele väärtused. Funktsioon **saledus** tagastab muutuja **hinnang** väärtuse, mille leiab (valib) **if**-lause muutuja **indeks** väärtuse alusel. Viimane omakorda sõltub parameetrite **L** ja **m** väärtustest. Pöördumine funktsiooni poole toimub peaprotseduuri lausest:

print (nimi, *"Oled "*, **saledus**(pikkus, mass))

Parameetritele **L** ja **m** vastavad argumendid on esitatud muutujate pikkus ja mass abil.

Funktsioonid **print**(), **input**() ja **round**() on Pythoni sisefunktsioonid.

Üheks Pythoni süntaksi oluliseks eripäraks on taanete kohustuslik ja reeglipärane kasutamine liitlausetes. Viimased sisaldavad teisi liht ja/või liitlauseid. Näites on kasutusel liitlaused **if** ja **else** nii peaprotseduuris kui ka funktsioonis **saledus**, mis on ka ise liitlause (**def**-lause).

Liitlause sisemised laused peavad algama taandega lause päise suhtes, milleks on soovituslikult 3–4 positsiooni, et struktuur oleks selgesti hoomatav. Ühesugusega tasemega laused peavad algama samast positsioonist.

Allpool on toodud näiteks peaprotseduuri **if**-lause.

NB! **else** ei ole omaette lause vaid **if**-lause osa, mis ei ole kohustuslik. Omaette **else** esineda ei saa, talle peab alati vastama üks ja ainult üks **if**-lause. Sellepärast nimetatakse seda sageli ka **else**-osalauseks.

```
if int(vastus) == 0 :  # valiku algus
    print ("Kahju! See on jube kasulik info!")
    print (nimi + "! Ehk mõtled veel? See on tasuta!")
else :
    pikkus = int(input(nimi+"! Sinu pikkus (cm) => "))
    mass = float(input("aga mass/kaal (kg) => "))
    …
    print ("Oled " + saledus(pikkus, mass) + "!")
print ('Kohtumiseni! Igavesti Sinu, Python!')
```

if lause

else lause

if-lausesse kuulub kaks lihtlauset, mis paiknevad päise suhtes taandega. Sellega seotud **else**-osalause peab algama täpselt samast positsioonist.

else-lausesse kuulub kuus lihtlauset, millel kõigil on täpselt ühesugune taane.

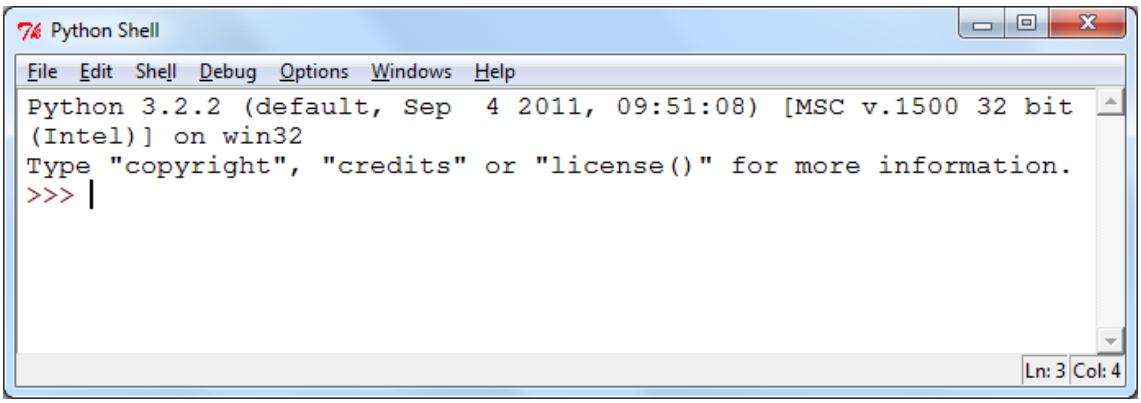
Viimasel print-lausel ei ole taanet, sest see ei kuulu **if**-lausesse.

Funktsioonis **saledus** on mitu erineva tasemega lauset. Funktsioon **def** ise kujutab liitlauset, mille sees on kaks liitlauset: **if**- ja **else**-lause. Esimeses on üks lihtlause, teises kaks liitlauset, mis on samuti **if**- ja **else**-laused. Viimases on omakorda **if**- ja **else**-lause. Taanded ja ainult taanded määravad liitlausete sisalduvuse tasemeid. Täpsemalt on liitlausete struktuurist juttu jaotises **Liitlaused**.

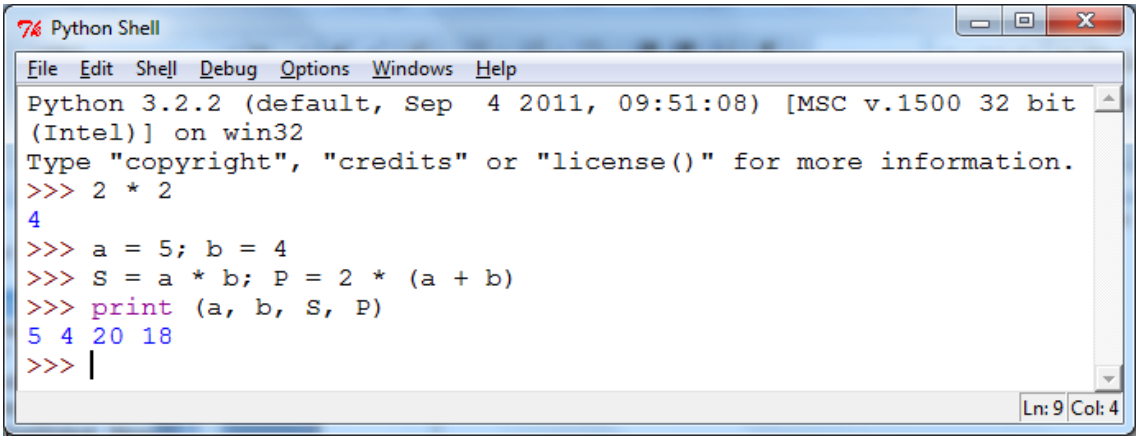
Programmide sisestamine ja täitmine

Programmide sisestamiseks, redigeerimiseks ja täitmiseks kasutatakse tavaliselt interaktiivset kasutajaliidest **IDLE** (***I**nteractive **D**eve**L**opment **E**nvironment*), mille haldur kuulub Pythoni põhikomplekti, kuid on ka mitmeid teisi Pythoni kasutajaliideseid nagu näiteks PyScripter, mille peab eraldi alla laadima. IDLE käivitamiseks võib kasutada Windowsi Start-menüüst järgmist korraldust:
Start > Programs > Python 3.x > IDLE (Python GUI).

Kuvatakse interaktiivse kasutajaliidese aken **Python Shell**.



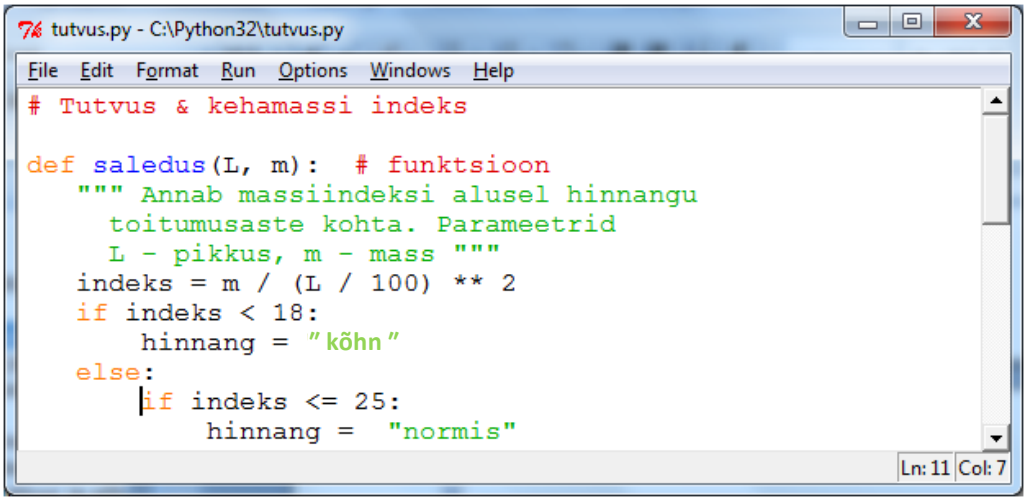
Shelli aknasse (käsuaknasse) väljastatakse tulemused ja teated. Akent võib kasutada interaktiivseteks arvutusteks nn kalkulaatori režiimis. Käsureale, mille alguses on sümbolid **>>>**, saab sisestada väärtusi, avaldisi, lauseid ja programmi fragmente.



Selles programmis on kõigepealt leitud avaldise 2*2 väärtus, mis, nagu selgub, on ka Pythoni jaoks ikka veel 4. Edasi omistatakse muutujale **a** väärtus 5 ja muutujale **b** väärtus 4. Tegemist võiks olla siin näiteks ristküliku külgede pikkustega. Järgnevate omistamislausetega leitakse ristküliku pindala (**S**) ja ümbermõõt (**P**). Lausega **print** väljastatakse ekraanile **a**, **b**, **S** ja **P** väärtused.

Selline töörežiim võimaldab mitmesuguste operatiivsete arvutuste tegemist näiteks programmide silumiseks, aga ka keelevahenditega tutvumiseks ja nende uurimiseks, kuid seda kasutatakse ka päris algajatele mõningate põhimõistete (andmete tüübid, avaldised, muutujad, …) selgitamiseks. Meie sellel pikemalt ei peatu, soovitame õppuril endal katsetada.

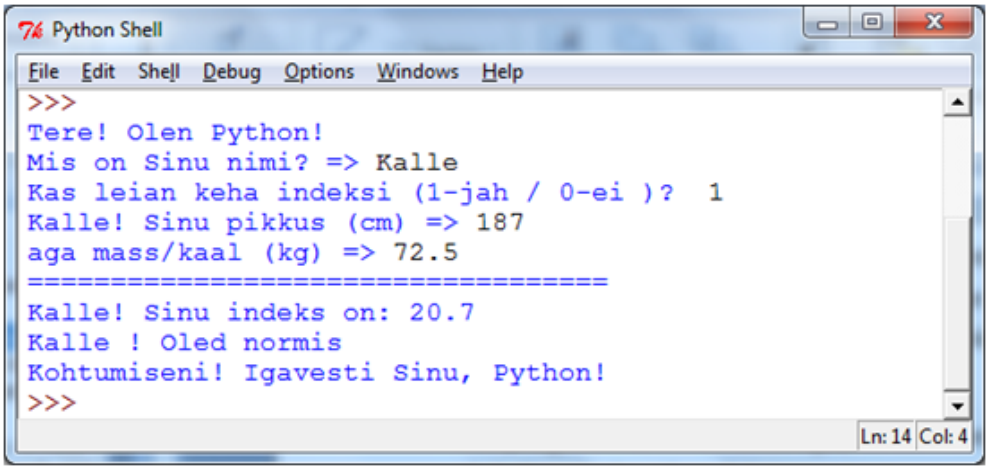
Programmi sisestamiseks tuleb File-menüüst anda käsk New Window. Ilmub **redaktori aken**, kuhu saab sisestada või kopeerida programmi. Tegemist on spetsiaalse tekstiredaktoriga, mis toetab Pythoni programmide sisestamist – taanete tegemist, programmi elementide ilmestamist, abiinfo ja spikrite pakkumist jms. Vaadeldava programmi algus on allpool toodud redaktori aknas. Redaktor kuvab erinevad programmi elemendid – kommentaarid, võtmesõnad, stringid jms – erinevate värvidega, kui programmifail on salvestatud laiendiga **.py**.



Programmi ehk mooduli saab käivitada redaktorist käsuga **Run** > **Run Module** või klahviga **F5**. Programmi täitmist korraldab spetsiaalne süsteemiprogramm – Pythoni interpretaator. Programm (fail) peab olema enne täitmist salvestatud.

Kui programmis on süntaksivigu, kuvab interpretaator veateate ja näitab ka vea eeldatava koha. Kui vigu ei ole, alustab interpretaator programmi täitmist, muutes aktiivseks käsuakna (Shelli akna), kus kuvatakse programmi teated ja tulemused. Vead võivad tekkida ka täitmisel, kui näiteks sisestatakse ebasobivad andmed vms.

Vaadeldava programmi jaoks on allpool näidatud Shelli aken variandi jaoks, kui kasutaja nimega Kalle soovis lasta arvutada kehamassiindeksi.

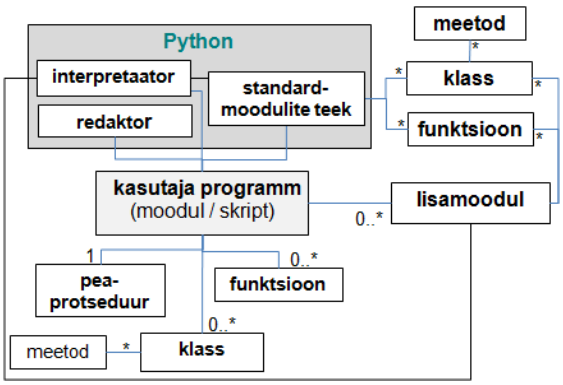


Programmide töötlemisel on kasulik võimalus pöörduda Help-menüüst käsuga Python Docs või klahviga **F1** Pythoni dokumentatsioonileheküljele (Python v3.x.x. documentation), mis vastab kasutatavale versioonile ning sisaldab hulgaliselt informatsiooni Pythoni kohta.

Moodulid, funktsioonid, klassid ja meetodid

Pythoni programm (öeldakse ka moodul või skript) võib üldjuhul koosneda mitmest üksusest. Alati on olemas **peaprotseduur**, milles võib olla suvaline hulk **funktsioone** ehk **protseduure** ja kasutaja poolt määratletud **klasse**. Funktsiooni näide (saledus) oli programmis **Tutvus**. Funktsioonide loomise põhimõtteid vaadeldakse jaotises **Funktsioonid** ning kasutatakse läbi kogu käesoleva materjali. **Klasside** loomist ei käsitleta.

Reeglina kasutatakse programmis lisaks oma protseduuridele ja klassidele ka vahendeid Pythoni moodulitest. Tavaliselt on tegemist **standardmoodulite teeki** (*The Python Standard Library*) kuuluvate moodulitega, mis laetakse alla koos süsteemiga. Lisaks standardmoodulitele saab kasutada lisamooduleid, mis ei kuulu nimetatud teeki. On suur hulk mooduleid, mis on loodud erinevate autorite ja firmade poolt ning reeglina tasuta allalaaditavad Internetist. Kasutaja saab lihtsalt moodustada ka oma mooduleid ja kasutada neid analoogselt olemasolevate moodulitega.



Mooduli nagu ka kasutaja loodud programmi komponentideks võivad olla funktsioonide ja klasside definitsioonid ehk lühemalt öelduna **funktsioonid** ja **klassid**. Lihtsamal juhul koosneb moodul ainult funktsioonidest. **Klass** määratleb ühetüübiliste objektide **omadused** ja **tegevused**. Võimalikud tegevused määratakse nn **meetodite** abil, mis oma olemuselt kujutavad funktsioone ja erinevad tavafunktsioonidest selle poolest, et neid saab kasutada ainult antud klassi kuuluvate objektidega.

Pythoni rakenduste loomine on praktiliselt alati seotud standardmoodulite teegis asuvate vahendite kasutamisega. Osa selles olevatest funktsioonidest ja klassidest kuuluvad **sisseehitatud** (*built-in*) funktsioonide ja klasside hulka. **Sisefunktsioonide** kasutamiseks ei ole vaja mingeid spetsiaalseid meetmeid. Soovitud kohta, näiteks avaldisse, kirjutatakse funktsiooniviit: funktsiooni nimi, millele

tüüpiliselt järgnevad sulgudes argumendid. Sellised on näites **Tutvus** esinevad funktsioonid: **int**, **float**, **str**, **input**, **print** ja **round**. Järgneb veidi modifitseeritud lause programmist **Tutvus**:

print (nimi + *"!"*, *"Sinu massiindeks on:"* + **str (round (**indeks, 2)))

Funktsiooni **print** argumentideks on siin kaks stringavaldist.

NB! Python 3-s on **print** funktsioon, Python 2-s on see lause.

Näite teises argumendis on kaks sisefunktsiooni: **round**(r_arv) – ümardab reaalarvu etteantud täpsuseni ja **str**(arv) – teisendab arvu tekstivormingusse.

Põhjalikku informatsiooni standardteegi moodulite ja neisse kuuluvatest vahenditest, sh ka sise-funktsioonidest, saab Pythoni dokumentatsiooni (IDLE Help, Python Docs) materjalist Library Reference. Suur osa standardmoodulites olevaid funktsioone ning klasse ja nende meetodeid ei kuulu aga sisseehitatud vahendite hulka. Nendeks on ka üldlevinud matemaatikafunktsioonid sqrt(), sin(), cos(), log() jt, mille kasutamiseks peab **importima** oma projekti vastavad moodulid või nende komponendid.

import *moodul1*, *moodul2*, …

moodul esitatakse mooduli nime abil. Näiteks: **import** math, turtle, time, random.

Käsu alusel imporditakse projekti moodulid. Formaalselt on imporditud moodulid objektid, mille nimedeks on praegu moodulite nimed, seal kirjeldatud konstandid ja funktsioonid on objekti atribuudid ja meetodid.

Viitamiseks objekti (mooduli) meetoditele kasutatakse konstruktsiooni:

objekt.***meetod***(*argumendid*),

kus ***objekt*** esitatakse nime abil, ***meetod*** on meetodi (funktsiooni) nimi, ***argumendid*** esindavad meetodi täitmisel kasutatavaid andmeid. Argumentide tähendus, arv ja järjestus sõltuvad meetodist (funktsioonist).

Järgnevas on toodud mõned viitamise näited:

math.sqrt(x ** 2 + y ** 2), **math.**sin(**math.**pi * x), **turtle.**pendown(), **turtle.**setpos(-50, 100), **turtle**.left(45), **random.**randint(1, 100), algaeg = **time**.clock(), **time**.sleep(0.1)

Importimiseks võib kasutada ka:

from *moodul* **import** *

Sel juhul tehakse kättesaadavaks *mooduli* kõikide funktsioonide, meetodite ja klasside nimed ja neile viitamisel mooduli (objekti) nime ei kasutata.

from math **import** * ; **from** turtle **import** *

Kasutamise näited:

```
y = sqrt(log10(5 * cos(x + 3));

pendown();

pencolor('red');

forward(120)
```

Näeme, et kasutamine toimub samamoodi nagu sisefunktsioonide puhul. Kuigi teine variant on kompaktsem, kasutatakse rohkem esimest. Siin on ühelt poolt tegemist traditsioonidega ja „hea tooni“ reeglitega. Teiselt poolt võib aga suurtes programmides, kus on kasutusel palju funktsioone paljudest moodulitest, teise variandi korral tekkida nimede kokkulangemise tõttu segadusi. Uurimaks standardmoodulite olemust ja nende kasutamist, võiks teha mõned harjutused Shelli aknas.

```
>>> sqrt(169)
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in
<module>
    sqrt(169)
NameError: name 'sqrt' is not defined
>>>
```

Kõigepealt ruutjuur 169-st. Python kuvab veateate väites, et nimi '**sqrt**' ei ole defineeritud. Sama teade ilmub ka siis, kui taolist asja kasutatakse programmis ilma **import** käsuta; programmi töö katkestatakse.

```
>>> import math
>>> math.sqrt(169)
13.0
>>> a = 2; b = 3; c = -9
>>> math.sqrt(b ** 2 - 4 * a * c)
9.0
```

Imporditakse moodul **math**, mis sisaldab üldlevinud matemaatikafunktsioone nagu sqrt(), sin(), cos(), log() jmt. Nimi **math** (koos punktiga) peab eelnema igale funktsiooniviidale.

```
>>> c = 9
>>> math.sqrt(b ** 2 - 4 * a * c)
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in
<module>
    math.sqrt(b ** 2 - 4 * a * c)
ValueError: math domain error
```

Näide süsteemi reaktsioonist andmetele, mille puhul mingit operatsiooni teha ei saa. Praegu on tegemist katsega leida ruutjuur negatiivsest arvust.

```
>>> 3 * math.sin(math.pi *
b/7)/math.pi ** 2
0.29634255008464655
```

Ka konstant π on olemas moodulis **math**: math.pi

```
>>> from math import *
>>> 3 * sin(pi * b / 7) / pi ** 2
0.29634255008464655
```

Kui importimiseks kasutatakse käsku **from**, ei ole funktsiooniviitades mooduli nime.

Näide: Korrutustabel

Programm võimaldab harjutada korrutamist. Kõrval on toodud algoritm.

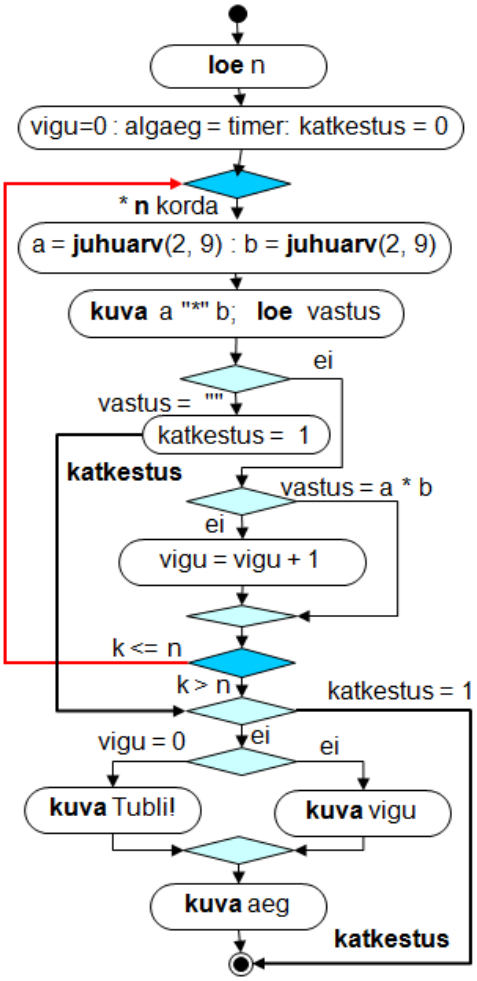
Näide illustreerib skalaarandmete ja avaldiste ning valikute kasutamist, tutvustab moodulite importimist ja korduste kasutamise põhimõtteid jms.

Esimesena sisestatakse ülesannete arv **n**. Programmi põhiosa moodustab kordus, mis kuvab järjest **n** ülesannet, loeb kasutaja vastused ja hindab neid.

Lõpus kuvatakse üldhinnang ja kulunud aeg. Kasutaja võib katkestada töö tühja väärtuse sisestamisega.

Tegevuste sisuline pool, mis eriti ei sõltu kasutavast keelest, on järgneval toodud UML-tegevusskeemil.

UML-tegevusskeem




```
import random, time  # moodulite import
def korrutustabel():
  ''' Korrutamistabeli harjutamine.
     Küsib ülesannete arvu ja esitab
     juhuarvude (2...9) abil ülesandeid.
     Fikseerib vigade arvu ja aja. '''
  n = int(input("Mitu ülesannet? "))
  vigu = 0; katkestus = False
  algaeg = time.clock()  # jooksev arvutiaeg
  for k in range(n) :   # korduse algus
     a = random.randint(2, 9)
     b = random.randint(2, 9)
     abi = str(k+1) + '. ' + str(a) + ' * ' + str(b)
     vastus  = input(abi + "=> " )
     if vastus == "":  # kui tühi väärtus
        katkestus = True
        break   # katkestab korduse
     if int(vastus) != a * b :
        print ("Vale!")
        vigu += 1       # vigu = vigu + 1
        time.sleep(2)  # karistuseks paus 2 sek
        # korratava ploki lõpp
  if katkestus == True:
     print ("Kahju, et loobusid!")
     return      # lõpetab programmi töö
  if vigu == 0 :
    print("Tubli! Kõik oli õige!")
  else :
     print ("Vigu oli ", vigu)
  aeg =  round(time.clock() - algaeg, 2)
  print ("Aeg: " + str(aeg) + " sek")

korrutustabel()
```

Kasutusel on 9 muutujat: ***n*** – ülesannete arv, ***a*** ja ***b*** – tegurid (juhuarvud vahemikus 2...9), ***vastus*** – kasutaja poolt pakutav vastus, ***vigu*** – valede vastuste arv, ***abi*** – abimuutuja teate kuvamiseks (algoritmis ei ole kajastatud), ***algaeg*** – alguse aeg, ***aeg*** – kulunud aeg, ***katkestus*** – katkestamise tunnus: *False* – ei katkestatud (skeemil 0), *True* – katkestati, sisestati tühi väärtus (skeemil 1).

Praktiliselt kogu programm kujutab endast ühte protseduuri, millel ei ole parameetreid ja mis ei tagasta ka mingeid väärtusi. Väljaspool seda on ainult käsk **import** ja pöördumislause **korrutustabel**().

Käsuga (lausega) **import** imporditakse Pythoni standardteegist moodulid **random** ja **time**. Mooduli **random** meetodit **randint**(min, max) kasutatakse juhuslike täisarvude genereerimiseks etteantavas vahemikus. Mooduli **time** meetod **clock** annab aja, mis on möödunud programmi töö algusest, meetod **sleep**(pp) võimaldab tekitada etteantud pikkusega (sek) pausi.

Praegu kasutatava importimise variandi korral peab viitamiseks vajalikule funktsioonile (tekkinud objekti meetodile) kasutama konstruktsiooni: ***objekt.meetod***(*argumendid*).

Programmi keskseks osaks on **kordus**, mille kirjeldamiseks kasutatakse Pythoni **for**-lause ühte levinumat varianti etteantud kordamise arvuga korduste määratlemiseks (täpsemalt vt jaotist „For-lause“, lk 62), mille üldkuju on järgmine:

for *muutuja* **in range (n):**

laused

n on alati täisarvuline muutuja, mida nimetatakse sageli ka **juhtmuutujaks**. Funktsioon **range**(n) moodustab täisarvude jada 0 kuni n-1. Seega omandab **for**-lauses juhtmuutuja järjest väärtused alates 0-st kuni n-1ni sammuga 1, kusjuures muutuja iga väärtuse korral täidetakse ***laused***. Kokku täidetakse kordusesse kuuluvaid lauseid **n** korda.

Programmis on ette nähtud võimalus korduse katkestamiseks kasutaja poolt. Kui lause

vastus = **input**(abi + "=> ")

täitmisel ei sisestata väärtust, vaid vajutatakse kohe klahvile Enter, võetakse muutuja ***vastus*** väärtuseks tühi väärtus. Kui järgnev **if**-lause fikseerib taolise olukorra, võetakse muutuja ***katkestus*** väärtuseks **True** ja **break**-lause katkestab korduse, andes täitmisjärje **for**-lausele järgnevale lausele:

```
if katkestus == True:     # ==True võib ära jätta
    print ("Kahju, et loobusid!")
    return      # lõpetab protseduuri töö
```

Paneme tähele loogikamuutuja ***katkestus*** kasutamist ja katkestuse töötlemist. Enne korduse algust võetakse muutuja väärtuseks **False**. Kui toimub katkestus, võetakse muutuja uueks väärtuseks **True**. Siin arvestatakse, et korduse ja kogu programmi töö võib lõppeda kahel erineval viisil – normaalne lõpp, kui täidetakse kõik ***n*** ülesannet (***katkestus*** = **False**), või programmi (korduse) töö katkestatakse (***katkestus*** = **True**). Viimasel juhul ei ole vaja teha ka kokkuvõtet. Sellekohase otsuse teeb ülaltoodud **if**-lause. Kui tingimus ***katkestus*** == **True,** lõpetab **return**-lause antud funktsiooni (protseduuri) ja sellega ka kogu programmi töö.

Andmete sisestamiseks ja väljastamiseks kasutatakse funktsioone **input** ja **print**. Funktsiooni **input** poolt tagastatava väärtuse teisendamiseks tekstivormingust täisarvuvormingusse on funktsiooni **int**.

Väärtuste teisendamisega on tegemist ka lauses abi = **str**(k+1) + '. ' + **str**(a) + ' * ' + **str**(b)

Funktsiooni **str** abil teisendatakse avaldise **k+1** ning muutujate **a** ja **b** väärtused tekstideks. See on vajalik tekstavaldises, kus arvud ühendatakse stringidega. Muutuja **abi** on kasutusel lihtsalt järgmise lause lühendamiseks.

Andmetest: väärtused ja tüübid, konstandid ja muutujad, omistamine ja avaldised

Märkandmete väärtused ja tüübid. Klassid

Pythonis programmides on kõik **väärtused** (tekstid, arvud, tõeväärtused) **objektid** ja kuuluvad kindlasse **klassi**. Igale klassile vastab kindel nimi. Märkandmete põhitüüpideks ehk -klassideks on:

- **stringid** – klass **str**
- **täisarvud** – klass **int**
- **ujupunktarvud** (reaalarvud) – klass **float**
- **tõeväärtused** – klass **bool**.

Igale väärtusele eraldatakse programmi täitmisel ajal koht (väli ehk pesa) arvuti mälus. Väärtuse esitusviis ja välja pikkus (baitides) sõltub väärtuse tüübist (klassist).

```
>>> type("Tere")
<class 'str'>
>>> type(13)
<class 'int'>
>>> type("13")
<class 'str'>
>>> type(21.53)
<class 'float'>
>>> type('21.73')
<class 'str'>
>>> type(2 * 2)
<class 'int'>
>>> type(2 * 2 == 4)
<class 'bool'>
>>>
```

Tutvumiseks väärtuste klassidega ehk tüüpidega sisestage prooviks Shellis mõned funktsioonid **type**. Mõisted „**tüüp**“ ja „**klass**“ on samaväärsed.

Funktsioon **type**(*väärtus*) tagastab väärtuse tüübi ehk klassi nagu näidatud kõrval. Kui argument on esitatud avaldisega, leitakse selle väärtus ja tagastatakse saadud tulemuse klass (tüüp). Paneme tähele, et **jutumärkide** või **ülakomade** vahele paigutatud arve käsitletakse stringidena.

Erinevat tüüpi väärtuste salvestamiseks arvuti mälus kasutatakse erinevaid **vorminguid**. Stringid (klass **str**) esitatakse **tekstivormingus**: igale märgile (sümbolile) vastab kindel kahendkood. Arvud võivad olla salvestatud nii tekstivormingus (klass **str**) kui ka spetsiaalsetes täisarvude ja realarvude jaoks ettenähtud püsipunkt- ja ujupunkt-vormingutes (klassid **int** ja **float**).

Tekstivormingus arvudega arvuti matemaatikatehteid teha ei saa! Kui seda on vaja, peab väärtuse teisendama funktsiooniga **int** või **float** arvuvormingusse.

Lihtsamal juhul, milleks on ka toodud näide, esitatakse märkandmed programmides üksikväärtustena (skalaaridena) – **konstantide** ja **muutujatena**.

Konstandid

Konstantide väärtused esitatakse otse programmis. Nende esitusviis sõltub väärtuse tüübist (klassist). Väärtuse muutmiseks peab tegema muudatuse programmis.

Stringkonstandid paigutatakse tavaliselt **jutumärkide** või **ülakomade** (apostroofide) – nn piirajate – vahele:

"kõhn", *"Tere, olen Python!"*, *'Mis on Sinu nimi?'* , *"e"*, *"0"*, *"12"*, *'-63.5'*

Piirajad konstandi väärtuse hulka ei kuulu. Jutumärgid ja ülakomad on piirajatena samaväärsed. Ühe konstandi jaoks peavad piirajad olema samad. Ühte tüüpi piirajate vahel oleva stringi sees võivad olla teist tüüpi piirajad, nagu näiteks:

```
nimi = input ( 'romaani "Kevade" autor => ' )
```

Kasutatakse ka kolmekordsete piirajate """ või ''' vahel asuvaid stringe. Sellel variandil on mõningad täiendavaid võimalusi ja eesmärke. Väärtus võib paikneda mitmel real ja täita näiteks ka pikema kommentaari rolli. Eriline tähendus on aga taolisel stringil funktsiooni alguses, mida mitmed Pythoni redaktorid võimaldavad kuvada Help-käsu abil (info funktsiooni kohta). Sellist stringi nimetakse **dokumendistringiks** (*docstring*).

NB! Stringkonstandina esitatud arv (näiteks: "0", "12", '-63.5') salvestatakse tekstivormingus – igale sümbolile vastab kindel kood. Sellises vormingus salvestatud arvudega aritmeetikaoperatsioone täita ei saa. Olgu märgitud, et funktsiooni **input** poolt tagastatav väärtus on alati tekstivormingus. Kui sisestatud arve kasutatakse arvutustes, tuleb need teisendada funktsioonide **int** või **float** abil vastavasse arvuvormingusse. Paneme tähele võrdluse esitust varasema näite peaprotseduuri **if**-lauses:

```
…
v= input("Leian keha indeksi(1-jah /0- ei) ")
if  v ==  "0" :
…
```

Siin arvestatakse, et **input**-lausega sisestatud muutuja ***v*** väärtus on tekstivormingus. Sellepärast on võrdlemisel ka väärtus 0 (null) esitatud tekstivormingus.

Arvkonstandid esitatakse programmides enamasti tavaliste kümnendarvudena või eksponentvormingus. Reaalarvudes kasutatakse murdosa eraldamiseks **punkti:**

13, 74600, -21, 73**.**0, 73.5902, 2**.**1e6 = 2.1×10^6 , 1e-20 =10^{-20}

Loogikakonstant ehk tõeväärtus esitatakse võtmesõnade **True** või **False** abil. Kasutamist võis näha programmis **korrutustabel** lk 14.

Muutujad. Omistamine, omistamislause, avaldised

Pythonis mõistetakse muutuja all **nime**, mis **viitab väärtusele** (objektile). Vaadeldava programmi peaprotseduuris on viis muutujat: **nimi**, **v**, **pikkus**, **mass**, **indeks**, funktsioonis **saledus** on kaks muutujat: **indeks** ja **hinnang**. Funktsioonis on kasutusel ka kaks **parameetrit**: **L** ja **m**. Parameetrid on eri liiki muutujad, mis erinevad nö tavamuutujatest otstarbe ja väärtuse saamise viisi poolest. Need esitatakse programmis samuti nimede abil.

Nimi võib koosneda ühest tähest või tähtede, numbrite ja allkriipsude (_) jadast, mis peab algama tähega või allkriipsuga. Teisi sümboleid (kaasaarvatud tühikud) nimes olla ei tohi. Pythonis eristatakse nimedes suur- ja väiketahti. Need reeglid kehtivad kõikide nimede kohta (funktsioonide, loendite, klasside nimed):

nimi, v, pikkus, L, x_1, Ab_t3_st_8, _algus, täht

Erinevalt paljudest teistest programmeerimiskeeltest (Pascal, C, Java) ei pea Pythonis deklareerima muutujaid ja määratlema kirjeldustes nende tüüpe. Seda isegi ei saa teha. Pythoni muutuja luuakse programmi töö ajal **omistamislause** täitmisel, kui muutuja nimi esineb esimest korda omistamislause vasakus pooles. Sellega luuakse ja salvestatakse ka muutuja esimene (võimalik, et ainuke) väärtus. Omistamislause põhivariant esitatakse järgmiselt :

muutuja = ***avaldis***

Selles lauses on ***muutuja*** muutuja nimi, ***avaldis*** annab eeskirja väärtuse leidmiseks. Avaldise erijuhuks on konstant, muutuja ja funktsiooniviit.

hinnang = *"kõhn"*, *indeks* = *mass* / (0.01 * *pikkus*) ** 2 ; *nimi* = **input**(*'Sinu nimi => '*)

pealik = *"Juku"*; *palk* = 0; *n* = 13; *x* = 2.73; *y* = -5; *dist* = (x**2 + y**2)**0.5

Proovige Shelli aknas muutujate ja avaldiste kasutamist, sisestades näiteks allolevad korraldused.

```
>>> x = 2.73; y = -5
>>> dist = (x**2 + y**2)**0.5
>>> dist
5.696744684466735
>>> s = (x**2 - 3*y)**0.5 - 4*(x + 3) / 2
>>> s
-6.721550886629678
>>>
```

Luuakse kaks muutujat: ***x*** ja ***y*** ning omistatakse neile väärtused: 2.73 ja -5.

Luuakse muutuja ***dist***, arvutatakse avaldise väärtus ja omistatakse see muutujale.

Luuakse muutuja ***s***, arvutatakse avaldise väärtus ja omistatakse see muutujale.

Aritmeetika põhitehted ja nende prioriteedid:

****** astendamine

*****, **/** korrutamine ja jagamine

+ , **-** liitmine ja lahutamine

Toodud näidetes on mitmeid **arvavaldisi**. Nende operandide väärtusteks on arvud ja operatsioonid määratakse aritmeetikatehetega. Programmis on ka neli **stringavaldist**.

Lauses

print (nimi + "! Sinu massiindeks on:" + **str(**indeks))

olevas avaldises toimub kolme stringi ühendamine (liitmine): muutuja ***nimi*** väärtus (näiteks Kalle), konstandi *"! Sinu massiindeks on:"* ja muutuja ***indeks*** väärtus (näiteks 20.7).

Avaldisega

print(35 * "=")

korratakse stringi "=" 35 korda ning kuvatakse 35-st märgist koosnev „joon". Täpsemalt avaldistest vt jaotises „Avaldiste struktuur ja liigid".

Tutvumine loenditega

Loend ehk ühemõõtmeline massiiv kujutab endast järjestatud väärtuste (objektide) kogumit. Loend tähistatakse ühe nimega, viitamiseks elementidele kasutatakse nime koos nurksulgudes asuva indeksiga (järjenumbriga) – ***nimi***[i*ndeks*]: V[0], V[i], V[2*k+1]. Indeks võib olla konstant, muutuja või avaldis. Elementide nummerdamine (indeks) algab alati nullist!

Funktsioon **len**(loendi_nimi) võimaldab leida loendi pikkuse – elementide arvu loendis.

Näide: Meeskond

On antud mingi meeskonna liikmete nimed ja pikkused. Rakendus teeb järgmist:

- kuvab (prindib) kõikide mängijate nimed ja pikkused
- leiab ja kuvab mängijate keskmise pikkuse
- leiab nende liikmete keskmise pikkuse, kellel see on suurem üldisest keskmisest
- leiab ja kuvab kõige pikema mängija pikkuse ja nime

Luuakse kaks loendit: nimed ja P. Esimene koosneb stringidest, teine arvudest.

Esimese **for**-lause abil prinditakse meeskonna kõikide liikmete nimed ja pikkused ning leitakse pikkuste summa. Jaganud leitud summa n-ga, saadakse keskmine pikkus.

```
# Loendite kasutamise demo, loendite loomine
nimed = [ 'Haab', 'Kask', 'Mänd', 'Paju', 'Saar', 'Tamm' ]
P = [ 193, 181, 178, 203, 197, 177]
n = len(nimed) # elementide arv loendis (pikkus)
# loendite kuvamine ja summa leidmine
print ("Meeskonnas on ", n, "liiget")
print (" ", "Nimi", "Pikkus")  # tabeli päis
summa = 0
for i in range(n) :
    print (i + 1, nimed [i], P[i])
    summa = summa + P[i]
kesk  = summa / n  # keskmine pikkus
print ("Keskmine pikkus:", round(kesk, 2))
# Üle keskmiste keskmine
print ("\nÜle keskmise pikkuse")
summa = k = 0  # üks väärtus kahele muutujale
print (" ", "Nimi", "Pikkus")
for i in range(n):
   if P[i] > kesk:
      print (nimed [i], P[i])
      summa = summa + P[i]; k += 1
kesk = summa / k
print ("Üle keskmise keskmine:", round(kesk, 2))
# Kõige pikem
maks = P[0]; nr = 0
for i in range(n):
   if P[i] > maks: maks = P[i]; nr = i
print ('\nPikim on', nimed[nr], '-', maks, 'cm')
```

Programmi esimese osa väljundil on kuju:

```
Meeskonnas on  6 liiget
  Nimi Pikkus
1 Haab 193
2 Kask 181
… …   …
6 Tamm 177
Keskmine pikkus: 188.17
```

Teine **for**-lause prindib keskmisest pikemate meeskonna liikmete nimed ja pikkused.

Leitakse ka nende pikkuste summa ja arv (k), mis on vajalikud vastava keskmise leidmiseks.

```
Üle keskmise pikkuse
Nimi  Pikkus
Haab  193
Paju  203
Saar  197
Üle keskmiste keskmine: 197.67
```

Kolmas **fo**r-lause leiab maksimaalse elemendi loendis P ja selle järjenumbri.

```
Pikim on Paju – 203 cm
```

Programmi lausete struktuur ja põhielemendid

Kokkulepped süntaksi esitamiseks

Keelekonstruktsioonide kirjeldamisel kasutatakse teatud kokkuleppeid, mis võimaldavad näidata lausete ja nende elementide esitusviise kompaktselt ja ühemõtteliselt. Taolisi leppeid kasutatakse laialdaselt formaalsete keelte süntaksi kirjeldamiseks.

Võtmesõnad, **tehtesümbolid**, **piirajad** ja **eraldajad** moodustavad tavaliselt keelekonstruktsiooni püsiva osa, need peavad olema esitatud programmis täpselt nendes kohtades ja sellisel kujul, nagu on näidatud kirjelduses.

Keelekonstruktsioonide muutuvad komponendid võib valida programmi koostaja, arvestades nende esitusreegleid. Kirjeldustes on need toodud kaldkirjas: *nimi, avaldis, lause* jne. Muutuv osa võib olla esitatud üldkujul, hiljem täpsustatakse seda täiendavate kirjeldustega.

Nurksulgudes [*a*] olev element võib esineda keelekonstruktsiooni antud kohas, kuid ei ole kohustuslik. Kui nurksulgudes asuvale elemendile järgneb punktiir ([*a*]…), tähendab see, et antud element võib puududa või korduda suvaline arv kordi.

Püstkriipsu | kasutatakse tähenduses "või", selle abil esitatakse võimalikud variandid.

Näiteks funktsiooni päise kirjelduse nö esimesel tasemel võib formaalselt esitada järgmiselt:

def *nimi* ([*parameetrid*]) :

Sellest võib välja lugeda, et **päis** peab algama võtmesõnaga **def**, sellele järgneb **funktsiooni *nimi***, edasi tulevad **sulud**, mille sees võivad olla ***parameetrid*** ning päise lõpus peab olema **koolon**. Võib ka järeldada, et isegi kui parameetrid puuduvad, peavad tühjad sulud ikkagi olema.

Täpsustame parameetrite esitamise reeglit:

parameetrid ::= *parameeter* [, *parameeter*] …

Mõnikord kasutatakse kirjelduses märke ::= tähenduses „on“. Kirjeldusest järeldub, et parameetreid võib olla üks või mitu (suvaline hulk), kusjuures mitme parameetri korral eraldatakse need üksteisest komadega.

parameeter ::= *nimi* [= *väärtus*]

väärtus ::= *string* | *täisarv* | *reaalarv* | *tõeväärtus*

*nim*i ::= *täht* | _ [*täht* | *number* | _] …

Nimi võib koosneda ühest tähest või allkriipsust või tähtede, numbrite või allkriipsude jadast, mis algab tähe või allkriipsuga.

Toome näiteks veel programmi rea määratluse:

[*lause* [**;** *lause*]…] [*# kommentaar*]

Ühel real võib olla mitu lauset, mis eraldatakse semikoolonitega. Rea lõpus võib olla kommentaar, mis peab algama märgiga **#**. Real võib olla ainult üks või mitu lauset või ainult kommentaar. Rida võib olla ka tühi.

Lausete põhielemendid

Laused on korraldused (käsud), mis määravad vajalikke tegevusi. Lausetel on keelereeglitega määratud kindel otstarve ja struktuur. Lausetes võivad esineda:

- **võtmesõnad**: kindla asukoha, kuju ja tähendusega sõnad ja lühendid: if**,** else**,** elif**,** return**,** for**,** in**,** while**,** break**,** continue**,** True**,** False**,** and**,** or**,** not**,** def**,** class jm
- **viited funktsioonidele** ja **meetoditele**, [moodul**.**] nimi([argumendid]): saledus(L, m), korruta(), print(*“Summa=“*, S), input(*'a='*), random.randint(2, 9), time.clock(), math.sin(x)**,** penup()
- **avaldised**: m / (L / 100) ** 2, str(a) + ' * ' + str(b), indeks < 18
- **konstandid**: *"kõhn"*, *'Sinu nimi => '*, 100, 18, 0.01, 9, True
- muutujate, parameetrite ja objektide **nimed**: *a*, *b*, *v*, *pikkus*, *indeks*, *ind*, *L*, *m* , *radom, time*
- **tehtemärgid**: +, -, /, *, **, =, ==, <=, +=
- **piirajaid** ja **eraldajaid**: ( ) ” “ : . ;
- **…**

Struktuuri järgi jagunevad laused kahte gruppi: **lihtlaused** ja **liitlaused**.

Lihtlaused

Lihtlause, nagu ütleb nimi, ei sisalda teisi lauseid. Mõned lihtlausete näited:

- **omistamislause**, *muutuja* = *avaldis* : x = 0; k = 1; y = 2*math.sin(x)+3*math.cos(x-2); nimi = *“Juku“*; mid = ideaal(L, vanus, sugu); k = k + 1; x = x + h; L = float(input(*'pikkus='*));
- **pöördumislause**, [*moodul***.**] *nimi*([*argumendid*]): saledus(L, m); korrutamistabel() print(*“Summa=“*, S, *“keskmine=“*, k); random.randint(2, 9); time.clock(); math.sqrt(x); penup()
- **return**-lause, return [avaldis [, avaldis]…]: return; return pindala; return x1, x2, tun
- **break**-lause, **break**
- …

Liitlaused

Liitlaused sisaldavad teisi liht- ja/või liitlauseid. Liitlausete korral on Pythonis ja ka teistes programmeerimiskeeltes probleemiks lausete struktuuri kindlakstegemine, seda nii inimese kui ka interpretaatori jaoks – millised laused kuuluvad antud lause sisse, kus lõpeb üks lause ja algab teine jms.

Programmeerimiskeeltes kasutatakse liitlausete struktuuri täpseks fikseerimiseks erinevaid lahendusi. Vaatleme neid **if**-lause näitel, mille olemus ja tööpõhimõte on sarnane kõikides keeltes. Scratchis on asi lahendatud väga lihtsalt ja selgelt: sisemised plokid paigutatakse **kui** ja **muidu** harudesse. Visual Basicus ja mõnes teises keeles (Ada), määratakse **Else-**haru ja **If**-lause lõpp spetsiaalse **End If**-osalause abil. Mitmes keeles (C-pere, Java, Pascal) kasutatakse liitlausete struktuuri fikseerimiseks **looksulge** või võtmesõnade paare: **begin** … **end**. Analoogsed probleemid on seotud ka teiste liitlausetega.

Scratch, BYOB

Visual Basic, Ada

If *tingimus* **Then**
  laused 1
Else
  laused 2
End If
järgmine lause

C-pere**, Java**

If *tingimus* {
  laused_1
}
else {
  laused_2
}
järgmine lause

Python

if *tingimus* **:**
  laused_1
else :
  laused_2
järgmine lause

Pythonis on struktuuri ja lause lõpu määramisel oluline roll kohustuslikel **taanetel**. Tulemus on põhimõtteliselt lihtne, ülevaatlik ja kompaktne, kuid nõuab veidi harjumist ja tähelepanu. Ülaloleval skeemil peab *järgmise lause* taane olema samasugune kui **if**-lausel ja **else**-osalausel. Kui aga taane on võrdne *laused_2* taandega, kuulub *järgmine lause* **else**-osalausesse ja programm ei tööta päris õigesti. Teistsugust taanet olla ei saagi, sest Python kontrollib taandeid ja kui need ei sobi, kuvab veateate.

Taolise skeemiga puutusime me kokku juba esimeses näites **Tutvus**, milles oli valiku kirjeldamine **if**- ja **else**-lause abil. Kui viimasel lausel **print** (*'Kohtumiseni! Igavesti Sinu, Python!'*) oleks taane, mis võib olla võrdne ainult eelmise lause taandega, kuuluks see **else**-osalausesse. Sellisel juhul kuvatakse teade ainult siis, kui tingimus **if**-lauses on väär, kuid seda peab kuvama alati.

Liitlause esitamise põhivariant Pythonis on järgmine:

liitlause päis :
  liitlause keha – plokk

Päise esitus sõltub lause tüübist: **if**-, **else**-, **for**-, **while**-, **def**-lause jm.; lõpus peab tingimata olema **koolon :**. **Liitlause keha** on plokk, mis võib koosneda liht- ja liitlausete jadast. Kõikidel sama tasemega lausetel peab olema ühesugune taane.

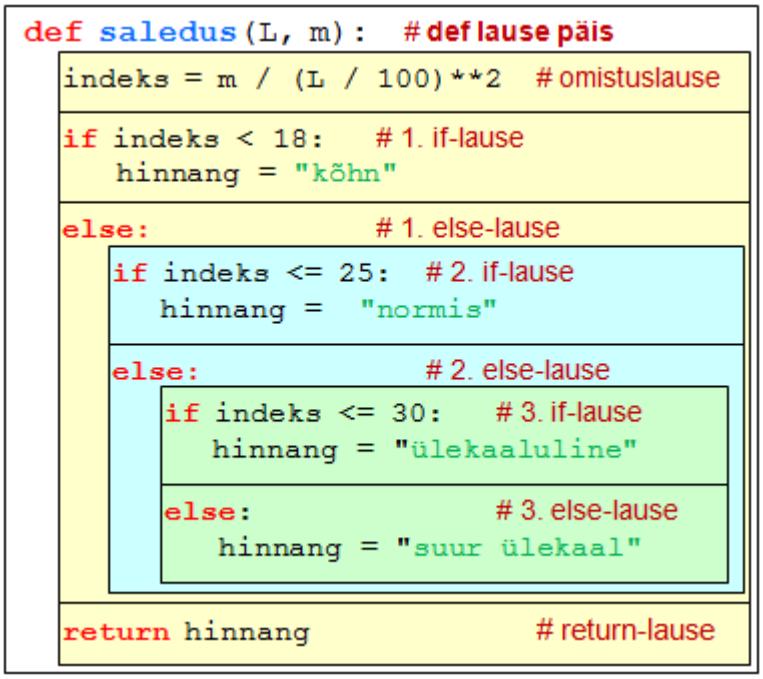
On lubatud, kuid mitte eriti soovitatav, liitlause esitamine ühel real järgmisel kujul:

liitlause päis : lihtlause [**;** lihtlause] **…**

Sellist esitusviisi võiks erandkorras kasutada juhul, kui liitlausesse kuulub 1 kuni 2 lühikest lihtlauset, näiteks

if a > 0 **:** S += a **;** n +=1

Järgnevalt käsitleme üldisemat juhtu – funktsiooni **saledus**, mis kujutab formaalselt ühte **def**-liitlauset.



def-lausesse kuulub antud juhul neli lauset: omistamislause, **if**-lause ja **else**-osalause ning **return**-lause.

Esimene ja viimane on lihtlaused, teine ja kolmas liitlaused. Kõigil neljal lausel peab olema kindlasti ühesugune taane, **if**- ja **else**-lause puhul on tegemist nende päistega, sest siselausetel on juba järgmise tasemega taanded. **else**-osalausesse kuuluvad omakorda **if**- ja **else**-laused ja viimasesse veel järgmised.

Kui **return**-lause taane oleks võrdne teise **else**-osalause taandega, kuuluks see viimase sisse. Sellisel juhul kui tingimus indeks<18 on tõene, jääb **return**-lause täitmata ja hinnangut ei tagastata.

Taanetel on oluline koht ka liitlause ilmekuse ja loetavuse seisukohalt ja seda kõikides programmeerimiskeeltes. Keeltes, kus taanded ei ole kohustuslikud, soovitatakse neid loetavuse parandamise mõttes siiski kasutada. Näites on toodud sama funktsioon Visual Basicus ilma taaneteta (formaalselt lubatud) ja taanetega. Võrdluseks on kõrval veel kord toodud see funktsioon Pythonis.

```
Function Saledus(L, m)
indeks = m / (L / 100) ^ 2
If indeks < 18 Then
hinnang = "kõhn"
Else
If indeks <= 25 Then
hinnang = "normis"
Else
If indeks <= 30 Then
hinnang = "ülekaaluline"
Else
hinnang = "suur ülekaal"
End If
End If
End If
saledus = hinnang
End Function
```

```
Function Saledus(L, m):
  indeks = m / (L / 100) ^ 2
  If indeks < 18 Then
    hinnang = "kõhn"
  Else
    If indeks <= 25 Then
      hinnang = "normis"
    Else
      If indeks <= 30 Then
        hinnang = "ülekaaluline"
      Else
        hinnang = " suur ülekaal "
      End If
    End If
  End If
  saledus = hinnang
End Function
```

```
def saledus(L, m):
  indeks = m / (L / 100) ** 2
  if indeks < 18:
    hinnang = "kõhn"
  else:
    if indeks <= 25:
      hinnang = "normis"
    else:
      if indeks <= 30:
        hinnang = "ülekaaluline"
      else:
        hinnang = "suur ülekaal"
  return hinnang
```

Sügav sisaldavuse aste liitlauses vähendab programmi teksti ilmekust ja raskendab loetavust. Võimaluse korral peaks püüdlema sisaldavuse taseme vähendamisele.

```
def saledus(L, m):
  indeks = m / (L / 100) ** 2
  if indeks < 18:
     hinnang = "kõhn"
  elif: indeks <= 25:
     hinnang = "normis"
  elif: indeks <= 30:
     hinnang = "ülekaaluline"
  else:
     hinnang = "suur ülekaal"
  return hinnang
```

Valiku kirjeldamisel **if**-lausega on Pythonis võimalus kasutada **elif**-osalauseid. Lause täitmisel kontrollitakse kõigepealt tingimust **If**-osalauses, kui see on tõene, täidetakse if_osalause tegevused, kõik ülejäänu jääb vahele. Vastupidisel juhul kontrollitakse järjest tingimusi **elif**-osalausetes (kui neid on olemas) ning kui leitakse **esimene tõene**, täidetakse järgnevad tegevused, kõik ülejäänu jääb vahele. Kui ükski tingimus ei ole tõene, täidetakse else_osalause tegevused (kui need on).

Kõrvaltoodud funktsiooni **saledus** variandil ehk liitlausel **def** on ainult kaks sisaldavuse astet. Selles on omistamislause, **if**-lause, kaks **elif**-osaluset, **else**-osalause ja **return**-lause. Kõikides sisalduvates liitlausetes on ainult üks lihtlause.

Toodud funktsiooni võiks esitada ka järgmiselt:

```
def saledus(L, m) :
  indeks = m / (L / 100) ** 2
  if    indeks < 18 :   return  "kõhn"
  elif  indeks <= 25 :  return  "normis"
  elif  indeks <= 30 :  return  "ülekaaluline"
  else :                return  "suur ülekaal"
```

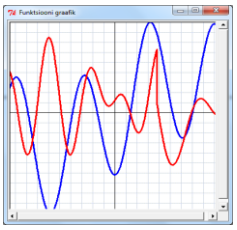
Siin on loobutud muutujast ***hinnang*** ning tagastatav väärtus esitatakse kõikidel juhtudel otse **return**-lauses. Viimane aga on esitatud vastavas üherealises liitlauses.

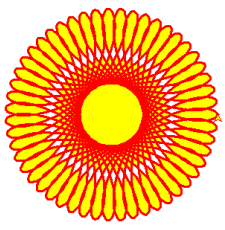
Protseduuride kasutamine on üheks sageli kasutatavaks võimaluseks sisaldavuse taseme vähendamiseks ning programmide struktuuri ja loetavuse parandamiseks. Kui näiteks loobuda programmis **Tutvus** funktsiooni **saledus** kasutamisest ja paigutada vastavad tegevused peaprotseduuri, suureneks selle sisaldavuse tase ning jälgitavus halveneks.

Veidi kilpkonnagraafikast

Pythoni kooseisu kuulub moodul **turtle**, mis sisaldab nn „kilpkonnagraafika“ vahendeid. Joonestamise operatsioone täidab spetsiaalne graafikaobjekt – ***turtle*** (kilpkonn) – Scratchi spraidi analoog. Kilpkonni (objekte) võib olla mitu ja neil võivad olla erinevad kujud. Vaikimisi on kilpkonni üks ja sellel on nooletaoline kuju ➤. Kilpkonnaga on seotud **pliiats** ja objektil on suur hulk meetodeid ehk käske, mille abil saab seda liigutada, pöörata, muuta pliiatsi suurust, värvust jms.

Kui programmis on joonistamiskäske, kuvatakse automaatselt graafikaaken ***Python Turtle Graphics***. Programmis saab määrata akna suuruse. Kui seda ei tehta, valib süsteem mõõtmed ise, arvestades arvuti ekraani suurust. Mõõtühikuks on piksel. Koordinaatsüsteemi nullpunkt on akna keskel.





Iseenesest on kilpkonna kasutamine üsna selge ja lihtne ning selle mõistmiseks ja kasutamiseks eriti suuri eelteadmisi vaja ei ole. Tegemist on joonistamise robotiga. Vastavate meetodite abil nagu **forward**(d), **setpos**(x, y), saab muuta selle asukohta, pöörata – **left**(nurk), **right**(nurk), muuta pliiatsi suurust ja värvi – **pensize**(w) ja **pencolor**(v) jm. Üldine juhtimine ja vajalikud arvutused tehakse tavaliste Pythoni lausete abil nagu kordused, valikud jms.

Järgnevalt on toodud paar lihtsat näidet. Lugeja võiks proovida neid Pythonis käivitada ning täiendada ja muuta.

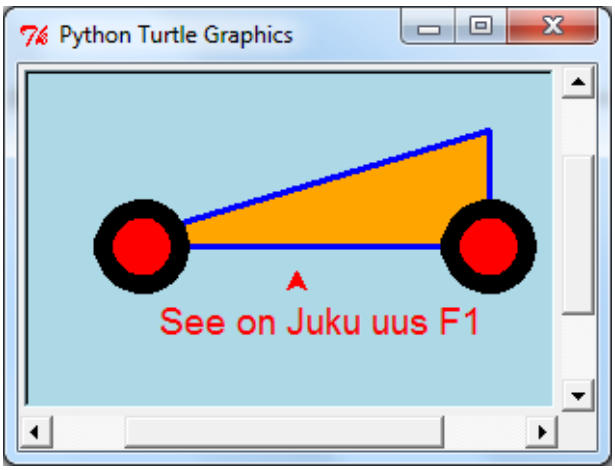
Lisainfot saab hankida Pythoni dokumentatsioonist ja jaotisest „Graafikaandmed ja graafikavahendid Pythonis“, lk 47.

Näide: Superauto ehk Juku F1

```
from turtle import *
setup(300, 200)       # akna mõõtmed
bgcolor("lightblue")  # akna põhja värv
# auto kere (kolmnurga) joonistamine
penup(); setpos(-100,0); pensize(3)
pencolor("blue"); fillcolor("orange")
begin_fill()  # alusta täitmist
pendown(); forward(200);   left(90)
forward(60) ;  setpos(-100, 0)
end_fill()    # lõpeta täitmine
# ratta (täidetud ringi) joonistamine
setpos(-60, 0)  # pliiats ringi algpunkti
color('black', 'red'); pensize(10)
begin_fill(); circle(20); end_fill()
# teise ratta joonistamine
penup(); setpos(120,0); pendown()
begin_fill(); circle(20); end_fill()
# teksti kirjutamine graafikaaknasse
penup(); setpos(-70, -50) # teksti algus
pencolor('red')
write ("See on Juku uus F1", font = ('Arial', 14) )
home()  # koju – akna keskpunkti  (0, 0)
left(90); backward(13)  #  vasakule, alla
mainloop()  # graafikaaken ooteseisu
```

Programm joonistab mingi autotaolise kujundi: täidetud kolmnurk – kere, ringid – rattad.

Kilpkonna meetodite importimiseks moodulist **turtle** kasutatakse käsku **from**, tänu millele saab kasutada meetodeid ilma viiteta objektile (kilpkonnale).



Pliiatsi värvi ja täitevärvid saab määrata eraldi meetoditega **pencolor**(jv) ja **fillcolor**(tv) või meetodiga **color**(jv, tv) koos. Värvid esitatakse nimetuste (tekstikonstatide) abil: "red", "blue" , ...

Käsud **forward**(d) ja **backward**(d) viivad kilpkonna praegusest asukohast edasi või tagasi, arvestades suunda, mis on alguses paremale (0°).

Suunda muuta saab käskudega **left**() ja **right**().

Käsk **setpos**(x, y) viib punkti (x, y), suunda muutmata.

Ringi keskpunkt on pliiatsist (kilpkonnast) vasakul, **r** piksli kaugusel, näites r=20.

Näiteks kui pliiats on punktis (-60, 0) ja suund on 90°, on ringi keskpunkt (-80, 0).

Näide: Võrdse pindalaga ristkülik ja ring

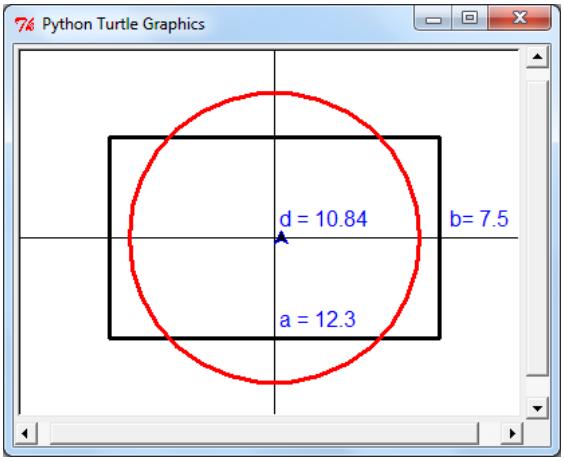
Rakendus leiab ristküliku külgede **a** ja **b** alusel ringi läbimõõdu **d**, mille pindala **S** on võrdne ristküliku pindalaga. Leitakse ka ristküliku ümbermõõdu ja ringjoone pikkuse **suhe** ning tehakse skeem.

Kasutatakse järgmisi seoseid: S = a * b, P = 2 (a + b), $d = \sqrt{4 \cdot s / \pi}$, suhe = P/(π * d)

```
from math import *
from turtle import *

# Algandmete lugemine
a = float(input("laius => "))
b = float(input("kõrgus => "))
m = float(input("mastaap => "))
#  Arvutamine
S = a * b;   P = 2 * (a + b)
d = sqrt(4 * S / pi);  suhe = P / (pi * d)

# väljund Shelli
print ( "laius =", a, " kõrgus =", b)
print ( "pindala =", round(S, 2))
print ( "läbimõõt =", round(d, 2))
print ( "suhe =", round(suhe, 2))

# Joonistamine
speed(0)  # joonistamise kiirus max !
# teljed. W - akna laius, H - kõrgus
W = window_width();  H = window_height()
penup(); setpos(-W/2, 0); pendown(); setpos(W/2, 0)
penup(); setpos(0, -H/2); pendown(); setpos(0, H/2)
# ristkülik
penup(); pensize(3)
am = a * m; bm = b * m   # korrutamine mastaabiga
setpos(-am / 2, -bm / 2)  # alumisse vasakusse nurka
pendown()
# üles, paremale, alla, algusesse
setpos(-am / 2, bm/2);  setpos(am / 2, bm/2)
setpos(am / 2, -bm /2);  setpos(-am / 2, -bm / 2)
penup()
# ring, keskpunkt vasakul kilpkonnast kaugusel  r(d/2)
pencolor( "red"); setpos(m * d / 2, 0)  # r = m * d / 2
setheading(90);  pendown(); circle(m * d/2)

# teksti kirjutamine graafikaaknasse
pencolor ( "blue")
penup(); setpos(am/2+ 5, 5)
write(" b= " + str(b), font=( "Arial", 12))
penup(); setpos(5, -bm/2 + 5)
write("a = " + str(a), font=( "Arial", 12))
penup(); setpos(5, 5)
write( "d = " + str(round(d, 2)), font=("Arial", 12))
mainloop()  # graafikaaken ooteseisu
```

Kuna joonestamise ühik (piksel) on üsna väike, kasutatakse siin mastaabitegurit ***m***: pikslite arv (ca 20–30) pikkusühikule. Sellega saab reguleerida joonise mõõtmeid graafikaaknas.

Peale algandmete sisestamist arvutatakse muutujate **S**, **P**, **d** ja **suhe** väärtused ning väljastatakse algandmed ja tulemused käsuaknasse. Graafikaakna mõõtmed (W – laius, H – kõrgus) valib Python, programm kasutab neid telgede joonestamisel.



Ristküliku joonestamisel viiakse pliiats (kilpkonn) kõigepealt ristküliku alumisse vasakpoolsesse nurka. Siit liigutakse üles, paremale, alla ja algpunkti tagasi.

Enne ringi joonestamist viiakse kilpkonn punkti (m*d/2) ja pööratakse üles (90 kraadi). Käsk **circle**(**r**) joonistab ringi raadiusega **r**, mille **keskpunkt** on **r** ühikut kilpkonnast **vasemal**.

NB! Kilpkonna suund on üles  .

Käsk **write**(tekst, …) võimaldab kirjutada teksti alates kilpkonna asukohast. Arvud peab enne väljastamist teisendama tekstivormingusse funktsiooniga **str**(). Saab anda kirja tüübi ja suuruse.

Kasutajafunktsioonid ja moodulid

Kasutajafunktsioonide ehk **protseduuridega** tutvusime juba esimeses näites. Järgnevas püüame laiendada selle valdkonna teadmisi ja oskusi.

Funktsiooni olemus, struktuur ja põhielemendid

Kaasajal mängivad protseduurid ja funktsioonid rakenduste loomisel erakordselt olulist rolli. Vähegi tõsisemates programmides kasutatakse selle jaotamist üksusteks ehk komponentideks, sest nii saab luua paindlikumaid, töökindlamaid ning paremini hallatavaid ja häälestuvaid rakendusi ja süsteeme. Kord loodud protseduure saab aga korduvalt kasutada erinevates rakendustes.

Mõni sõna terminitest. **Funktsioonid** olid algselt (ja mitmetes süsteemides nagu Fortran, Pascal, ADA, Basic, Scratch ka praegu) mõeldud ainult ühe väärtuse (arv, tekst, kuupäev, tõeväärtus, ...) leidmiseks ja tagastamiseks. Programmiüksusi, mida kasutatakse suvaliste tegevuste kirjeldamiseks, nimetatakse **protseduurideks** või **alamprogrammideks**. Pythonis nagu ka mitmetes teistes programmeerimissüsteemides nimetatakse kõiki protseduure funktsioonideks. Funktsioonid võivad määrata suvalisi tegevusi (joonestada, sisestada ja kuvada väärtusi, sisaldada andmevahetust failide ja andmebaasidega, ...) ning leida ja tagastada väärtusi – ühe või mitu. Mitmed autorid nimetavad Pythoni programmiüksusi vahel funktsioonideks, vahel protseduurideks ning nii ka siin.

Funktsiooniga/protseduuriga on seotud kaks aspekti: funktsiooni definitsioon ehk määratlus ja pöördumine funktsiooni poole. Formaalselt – süntaksi seisukohalt – kujutab funktsiooni definitsioon endast liitlauset, mille üldkuju on järgmine:

def nimi ([parameetrid]):
    [*docstring*]
    lause
    [lause]
    …

Funktsiooni päis algab võtmesõnaga **def**, millele järgneb nimi ja sulgudes parameetrid. Tühjad sulud peavad olema ka siis, kui parameetreid ei kasutata. Päise lõpus peab tingimata olema **koolon**.

Funktsiooni sisu ehk keha koosneb lausetest, mis peavad paiknema päise suhtes **taandega**. Otstarbekas on panna algusesse kolmekordsete piirajate vahele nn dokumendistring (vt lk 16).

Parameetrid esindavad funktsiooni sisendandmeid ja esitatakse kujul: *parameeter* [**,** *parameeter*] …, kus *parameeter* ::= *nimi* [**=** *väärtus*]. Parameetrid võivad esindada skalaarseid suurusi, loendeid, objekte ja funktsioone. Väärtus näitab vaikimisi võetavat väärtust – kui vastav argument pöördumises puudub, kasutatakse parameetrile omistatud väärtust. Parameetrite loetelus peavad parameetrid, mille jaoks pole esitatud vaikeväärtust, asuma eespool, vaikeväärtustega parameetrid on loetelu lõpus.

Funktsiooni poole pöördumises peab argumentide esitamisel arvestama parameetrite järjekorda, kui ei kasutata parameetrite nimesid. Kohustuslik on esitada vaikeväärtuseta parameetritele vastavad argumendid.

Järgnev näiteprotseduur joonistab ruudu küljepikkusega **a**, alguspunktiga (**x**, **y**), milleks on alumine vasakpoolne nurk. Vaikimisi võrduvad **x** ja **y** nulliga. Parameeter **w** määrab joone paksuse. Funktsioon leiab ja tagastab ruudu ümbermõõdu ja pindala.

```
from turtle import *
def ruut (a, w, x = 0, y = 0) :
  ''' ruut küljega a; x, y – koh
    w – joone paksus'''
  penup(); setpos(x, y)
  pensize(w); pendown()
  for i in range(4):
    forward(a)
    left (90)
  P = 4 * a; S = a * a
  return  P, S
```

Mõned võimalikud pöördumised:

```
ruut(100, 2)  # ainult nimedeta kohustulikud argumendid
ruut(150, y = -80, w = 5) # 1. nimeta, teised nimedega
ruut(x = -130, a = 200, w = 3) # kõik nimedega, osa puuduvad
# NB! nimedega argumendid võivad suvalises järjekorras
P, pind = ruut(30, 3)    #  P ja pind = tagastatavad väärtused
print ("pindala", pind)  # kasutatakse (kuvatakse) ainult ühte
```

Parameetrite nimed (näites: a, w, x, y), aga samuti funktsiooni sees kasutatavad muutujad (P ja S) omavad tähendust ja on kättesaadavad ainult antud protseduuris. Öeldakse, et **protseduur lokaliseerib** oma **parameetrid** ja **sisemuutujad**.

Pöördumine funktsiooni poole esitatakse kujul:

[*muutuja* [, *muutuja*] … =] ***nimi*** ([*argument* [, *argument*] …])

Siin on ***nimi*** funktsiooni nimi. Argumentide abil antakse vastavatele parameetritele tegelikud väärtused. Väärtuste esitamiseks võib kasutada konstante, muutujaid ja avaldisi. Argumentide esitamisel võib kasutada parameetrite nimesid, mida võib teha ka positsioonilistele parameetritele vastavate argumentide jaoks. Selliste argumentide esitusviisi korral ei ole nende järjekord oluline.

Vasak pool (*muutujad*) võib pöördumislauses olla, kui funktsioon tagastab **return**-lausega väärtusi. Muutujate arv peab võrduma tagastatavate väärtuste arvuga. Paneme tähele, et pöördumisel ei pea funktsiooni poolt tagastatavaid väärtusi tingimata vastu võtma.

Näite esimeses kolmes pöördumises ei võeta vastu tagastatavaid väärtusi. Seda tehakse ainult viimases pöördumises: P, pind = ruut(30, 3). Kuna **return**-lause tagastab kaks väärtust, on vasakus pooles samuti kaks muutujat. Kuvatakse ainult pindala.

return-lause üldkuju on järgmine:

return *avaldis* [, *avaldis*] …]

Lause **return** lõpetab funktsiooni töö ja tagastab avaldis(t)e väärtuse(d), andes täitmisjärje punkti, kust toimus pöördumine. Kas tagastatavad väärtused ka vastu võetakse, sõltub pöördumise viisist. Funktsioonis võib olla suvaline arv **return**-lauseid.

Mitmeprotseduurilistes programmides on andmevahetusel protseduuride vahel oluline roll, sest üks protseduur ei pääse ligi – sageli ei tohigi – teise protseduuri andmetele. Andmevahetuseks kasutatakse erinevaid viise. Peamisteks on just siin vaadeldud võimalused: parameetrite ja argumentide mehhanism ning väärtuste tagastamine **return**-tüüpi lausetega. Erinevates programmeerimiskeeltes kasutatakse mõnevõrra erinevaid viise.

Paljudes keeltes (Fortran, Pascal, Basic, C-pere, Java jt) jagunevad parameetrid kolme rühma: **sisendparameetrid**, **väljundparameetrid** ja **sisend-väljund parameetrid**.

- **sisendparameetrid** saavad väärtused argumentidelt pöördumisel, neid kasutatakse protseduuri täitmisel, kuid neid ei muudeta (tavaliselt ei saagi muuta),
- **väljundparameetritele** omistatakse väärtused protseduuri täitmise ajal ja need väärtused omistatakse vastavatele argumentidele ,
- **sisend-väljund parameetrid** omavad väärtusi enne pöördumist, neid kasutatakse ja muudetakse protseduuri täitmisel.

Pythonis on ainult sisendparameetrid, mida protseduuri täitmisel ei muudeta. Väljundparameetrite ja sisend-väljund parameetrite puudumist kompenseerib võimalus tagastada **return**-lausega mitu väärtust. Teistes keeltes saab taolisel viisil tagastada ainult ühe väärtuse.

Esitatu kehtib üksikväärtuste ehk skalaarandmete kohta. Praktiliselt kõikides keeltes, sh ka Pythonis, saab kasutada omamoodi sisend-väljund parameetritena massiive ja loendeid. Protseduur saab lugeda ja muuta parameetriks oleva loendi elementide väärtusi ja pöördunud protseduur saab muudetud väärtused kätte.

Üheks andmevahetuse võimaluseks enamikus keeltes on nn globaalsete andmete kasutamine, millele on juurdepääs mitmel protseduuril. Ka Pythonis on olemas lause **global**, millega saab kuulutada andmed globaalseteks. Arvestades mitmesuguseid võimalikke kõrvalmõjusid, ei ole taoline andmevahetuse viis eriti levinud ega mõistlik.

Andmevahetuseks kasutatakse ka faile (andmebaase ja dokumente), eriti siis, kui tegemist on suuremate andmehulkadega. Nii võib näiteks üks protseduur kirjutada andmed faili, teine protseduur neid aga sealt lugeda.

Ning lõpuks lausetest, mis moodustavad funktsiooni keha, milles võivad olla suvalised liht- ja liitlaused, sh ka **def**-laused. See tähendab, et Pythonis võib üks protseduur sisaldada teisi protseduure. Viimast võimalust kasutatakse suhteliselt harva.

Näide: Kolmnurga pindala
Funktsioon leiab kolmnurga pindala külgede pikkuste alusel. Arvestatakse võimalusega, et kolmnurka moodustada ei saa. Pindala leidmiseks kasutatakse Heroni valemit:

$$S = \sqrt{p(p-a)(p-b)(p-c)}$$, kus p = (a+b+c)/2.

Kui tingimus a+b<=c või b+c<=a või a+c<=b on tõene, siis kolmnurka moodustada ei saa.

```
def K_pind(a, b, c):
    ''' Kolmnurga pindala. a, b, c – küljed.
        Tagastab pindala või -1 '''
    if a + b <= c or b + c <= a or a + c <= b :
        return -1 # kolmnurka ei ole
    else:
        p = (a + b + c) / 2
        S = (p * (p -a) * (p - b) * (p - c)) ** (0.5)
        return S
```

```
# Funktsiooni katsetamine
a, b, c = eval (input("külgede pikkused => "))
  # korraga võib lugeda mitme muutuja väärtused
  # eval teisendab väärtused arvuvormingusse
pind = K_pind(a, b, c)
if pind == -1:
    print("Kolmnurka ei ole!", a, b, c)
else:
    print("küljed:", a, b, c, " pind=", round(pind, 3))
```

Funktsiooni töö testimisel demonstreeritakse võimalust korraga mitme muutuja väärtuse sisestamiseks. Sisestuslause võib arvude korral olla esitatud kujul:

muutuja [, *muutuja*]… = **eval**(**input**([teade]))

Mitme väärtuse sisestamisel eraldatakse need üksteisest komadega.

Funktsioon **eval**() on üldiselt ette nähtud avaldise väärtuse leidmiseks. Tegemist võib olla ka tekstavaldisega. Kui tegemist on tekstivormingus arvudega, teisendatakse need arvuvormingusse, analoogselt funktsioonidega **int**() ja **float**().

Näide: Ruutvõrrand

Järgnevalt on toodud funktsioon ruutvõrrandi $ax^2 + bx + c = 0$ lahendite **x1** ja **x2** leidmiseks. Arvestatakse võimalusega, et need võivad puududa. Kasutatakse spetsiaalset tunnust: kui lahendid puuduvad, võetakse tunnuse väärtuseks 0, muidu 1.

```
import math
def ruutvrd(a, b, c):
  ''' Ruutvõrrand. a, b, c  - kordajad.
     Tagastab tun = 1 | 0 – on ei ole
     x1, x2 – lahendid (kui on)'''
  D = b * b - 4 * a * c
  if D < 0:
    tun = 0; x1=""; x2=""
  else:
     D = math.sqrt(D)
     tun = 1
     x1 = (-b - D) / (2 * a)
     x2 = (-b + D) / (2 * a)
  return tun, x1, x2
```

```
# Peaprotseduur
print ("Lahendan ruutvõrrandeid!")
print ("Sisesta kordajad; a, b, c")
a = float(input( "Sisesta a => "))
while (a == 0):
  print ( " a ei tohi olla 0!!!")
  a = float(input("anna uus a => "))
b = float(input( "anna b => "))
c = float(input( "ja nüüd c => "))
tunnus, x1, x2 = ruutvrd(a, b, c)
if tunnus != 0:
  print("Siin need on:", x1, x2)
else:
  print("Lahendid puuduvad!!!")
```

Funktsiooni parameetriteks on ruutvõrrandi kordajad **a**, **b** ja **c**. Protseduur leiab diskriminandi ***D*** väärtuse. Kui see on negatiivne, omistatakse muutujale ***tun*** väärtus 0 ja muutujatele ***x1*** ja ***x2*** tühjad väärtused.

Peaprotseduur kontrollib kordajate sisestamisel ***a*** väärtust ja seni, kui see on null, küsib uut väärtust.

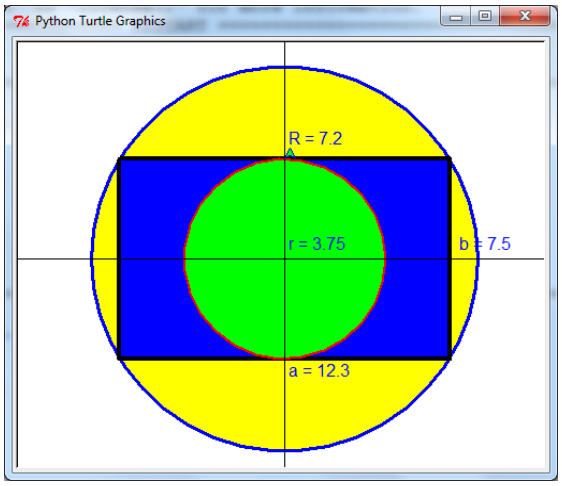
Pöördumisel funktsiooni poole lausega **tunnus, x1, x2 = ruutvrd(a, b, c)** arvestatakse, et see tagastab kolm väärtust.

Näide: Ristküliku karakteristikud

Rakendus leiab ristküliku külgede (**a** ja **b**) alusel selle pindala (**S**), ümbermõõdu (**P**), diagonaali (**d**), siseringi ja ümberringi raadiused (**r** ja **R**) ning ringide pindalad (**S1** ja **S2**). Programm joonistab ka ristküliku ja ringid. Programmi väljundid on näidatud allpool.

Python Shell

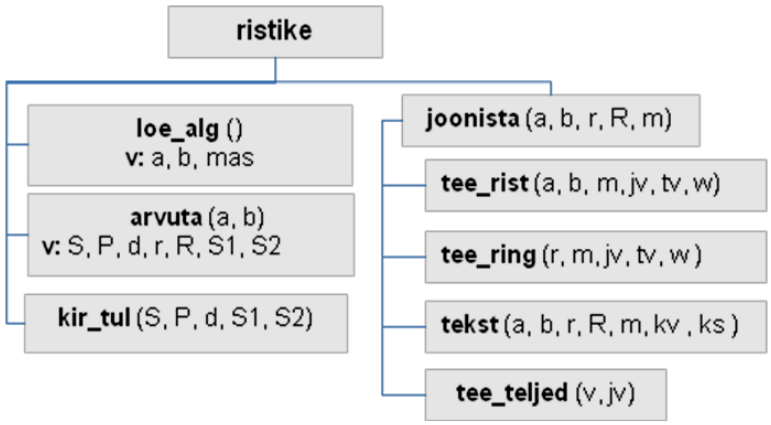
```
Python 3.2.2 (default, Sep  4 2011, 09:51:08)
>>>
Ristküliku omadused: pind, ümbermõõt jm
laius 12.3
kõrgus 7.5
mastaap (vaikimisi 25)
pindala = 92.25
ümbermõõt = 39.6
diagonaal = 14.41
siseringi pind = 44.18
ümberringi pind = 163.0
```



Programm on jagatud protseduurideks (funktsioonideks), arvestades täidetavate tegevuste iseloomu nagu algandmete sisestamine, arvutused, tulemuste kuvamine ja joonistamine. Antud juhul on tegemist väikese ülesandega, kus protseduuridest silmnähtavat kasu ei ole, kuid ettekujutus põhimõttest siiski tekib. Üheks eesmärgiks protseduuride kasutamisel on nendevaheline koostöö ja andmevahetuse korraldamise kinnistamine.

NB! Programmi jaotamine protseduurideks on rakenduse disaini juures üks olulisemaid tegevusi.

Peaprotseduuril **ristike** on neli alamprotseduuri: **loe_alg**, **arvuta**, **kir_tul** ja **joonista**. Viimasel on omakorda neli alamprotseduuri. Andmevahetuseks protseduuride vahel kasutatakse parametreid ja argumente.




```
from math import *
from turtle import *
def loe_alg ():
    ''' algandmete lugemine '''
    laius = float ( input ("laius ") )
    korgus = float ( input ("korgus "))
    mas = input("mastaap (vaikimisi 25) ")
    if mas == "":
        mas = 25
    else:  mas = int (mas)
    return laius, korgus, mas

def arvuta (a, b):
    ''' ristküliku põhiomadused '''
    S = a * b; P = 2 * (a + b)
    d = sqrt(a**2  +  b**2);  R = d / 2
    r = b / 2
    if a < b: r = a / 2    # r – pool lühemast küljest
    S1 = pi * r**2;  S2 = pi * R**2;
    return  S, P, d, r, R, S1, S2

def kir_tul (S, P, d, S1, S2):
    ''' tulemuste kirjutamine Shelli aknasse '''
    print ('pindala =', round(S, 2))
    print ('ümbermõõt =', round(P, 2))
    print ('diagonaal =', round(d, 2))
    print ('siseringi pind =', round(S1, 2))
    print ('ümberringi pind =', round(S2, 2))

def joonista (a, b, r, R, mas = 1):
    ''' joonistamise pealik '''
    tee_ring (R, mas, w = 3, jv="blue", tv="yellow")
    tee_rist (a, b, mas, w = 4, tv = "blue")
    tee_ring (r, mas, w = 2, jv="red", tv = "green")
    tee_teljed ()
    tekst (a, b, r, R, mas)
    mainloop()
```

Importimiseks kasutatakse käsku **from**, mis võimaldab kasutada funktsoonidele ja meetoditele viitamiseks kompaktsemat varianti.

Protseduur **loe_alg** loeb algandmed, milleks on ristküliku külgede pikkused. Parameetreid protseduuril ei ole. Tagastatavaid väärtusi on kolm: **laius**, **korgus** ja **mas** (mastaap).

Mastaabi jaoks on ette nähtud vaikimisi võetav väärtus: **25**. Kui kasutaja sisestab tühja väärtuse, omistatakse muutujale **mas** väärtus 25.

Funktsioon **arvuta**, leiab ja tagastab **return**-lause abil seitse väärtust. Funktsiooni sisendparameetriteks on külgede pikkused **a** ja **b**.

Tagastatavad väärtused: **S** – pindala, **P** – ümbermõõt, **d** – diagonaal, **r** ja **R** – siseringi ja ümberringi raadiused, **S1** ja **S2** – siseringi ja ümberringi pindalad.

Protseduur **kir_tul** kirjutab tulemused Shelli aknasse, kasutades Pythoni sisefunktsiooni **print**.

Väärtuste ümardamiseks kasutatakse Pythoni sisefunktsiooni **round**.

Protseduur **joonista** korraldab andmete väljastamist graafikaaknasse. Mastaabi jaoks on ette nähtud vaikimisi võetav väärtus, milleks on **1**.

Alamprotseduurides: **tee_ring**, **tee_rist**, **tee_teljed** ja **tekst** on mõnedel parameetritel vaikeväärtused. Pöördumisel on vastavad argumendid esitatud kasutades parameetrite nimesid. Erinevatel pöördumistel kasutatakse joone värvi (**jv**) ja paksuse (**w**) ning täitevärvi (**tv**) erinevaid väärtusi, mis asendavad parameetrite esialgsed väärtused.

```
def tee_teljed (w = 1, jv = "black"):
    ''' X ja Y teljed, w-joone paksus, jv -värv '''
    W = window_width();  H = window_height()
    pensize(w); pencolor(jv)
    penup(); setpos(-W/2, 0); pendown();
    setpos(W/2, 0);   penup();
    setpos(0, - H/2); pendown(); setpos(0, H/2)
```

Protseduur **tee_teljed** joonestab koordinaat-teljed, mis läbivad akna keskpunkti (0, 0). Kuna pöördumisel ei ole joone paksust (**w**) ja joone värvi (**jv**) esitatud, jäävad kehtima parameetrite vaikeväärtused. Vastavate meetodite abil teeb protseduur kindlaks akna laiuse ja kõrguse ning omistab nende väärtused muutujatele **W** ja **H**, mida kasutatakse telgede joonestamisel.

```
def tee_rist (a, b, m=1, jv = 'black', tv = "", w=1):
    ''' ristkülik a*b, m - mastaap, jv – joone värv,
      tv - täitevärv, w - joone paksus'''
    am = a * m; bm = b * m
    penup(); setpos(-am / 2, -bm / 2)
    color(jv, tv); pensize (w); pendown()
    begin_fill()
    setpos(-am /2, bm/2); setpos(am/2, bm/2)
    setpos(am/2, -bm/2); setpos(-am/2, -bm/2)
    end_fill()
```

Protseduur **tee_rist** joonistab ristküliku külgede pikkustega **a** ja **b** ning keskpunktiga (0, 0). Protseduur pakub mõnevõrra rohkem võimalusi võtmevormingus esitatud parameetritega, kui kasutatakse antud programmis.

Mastaabi (**m**) vaikimisi võetavaks väärtuseks on 1. Kui vastav argument pöördumisel puudub, kasutatakse joonestamisel ühikuna pikselit. Joone jaoks saab kasutada erinevat paksust (**w**) ja värvi (**jv**). Kujund võib olla täidetud etteantud värviga või mitte (tv = "" – tühi).

```
def tee_ring (r, m = 1, jv = 'black', tv = "", w = 1):
    '''ring, r - raadius, m - mastaap, jv – joone värv
      tv - täitevärv, w - joone paksus'''
    penup(); setpos(m * r, 0); color(jv, tv)
    pensize(w); pendown()
    setheading(90); begin_fill()
    circle(m * r); end_fill()
```

Protseduur **tee_ring** joonistab ringi raadiusega **r** ja keskpunktiga (0, 0). Arvestatakse, et joonestamisel on ringi keskpunkt vasakul pliiatsist (kilpkonnast) kaugusel **r**. Pliiats viiakse käsuga **setpos** punkti (m*r, 0) ja suunaks määratakse käsuga **setheading** 90 kraadi. Parameetrite (**m**, **jv**, **tv** ja **w**) jaoks kasutatakse sama põhimõtet, mis oli ristküliku jaoks.

```
def tekst (a, b, r, R, m = 1, kv = "blue", ks = 12):
    ''' testi kirjutamine graafikaaknasse '''
    pencolor (kv); penup(); setpos(a * m/2 + 10, 5)
    write("b = " + str(b), font=( "Arial", ks))
    penup(); setpos(5, -b * m / 2-20)
    write("a = " + str(a), font=( "Arial", ks))
    penup(); setpos(5, 5)
    write( "r = "+str(round(r, 2)), font=( "Arial", ks))
    penup(); setpos(5, m * b/2 + 10)
    write( "R="+str(round(R, 2)), font=("Arial", ks))
```

Protseduur **tekst** kirjutab külgede pikkused (**a**, **b**) ja raadiused (**r**, **R**) graafikaaknasse.

Kasutusel on kolm võtmevormingus parameetrit:
m – mastaap,
kv – kirja värv (= joone värv) ja
ks – kirja suurus (pikselites).

Pliiats viiakse käsuga **setpos** vajalikku kohta ning kuvatav tekst esitatakse käsus **write** string-avaldise abil, kusjuures saab määrata ka kirja tüübi ja suuruse.

```
# peaprotseduur
print ('Ristküliku omadused: pind, ümbermõõt jm')
laius, korgus, mas = loe_alg()
S, P, d, r, R, S1, S2 = arvuta (laius, korgus)
kir_tul(S, P, d, S1, S2)
joonista (laius, korgus, r, R, mas)
```

Peaprotseduur käivitab järjest vastavad alamprotseduurid, edastab neile argumentide abil vajalikud väärtused ning võtab vastu protseduuride **loe_alg** ja **arvuta** tagastatavad väärtused.

Universaalne funktsioon Ristkülik

Ülalpool vaadeldud näites oli funktsioonidel **tee_rist** ja **tee_ring** oluline puudus – loodavate jooniste asukoht oli jäigalt seotud graafikaakna keskkohaga.

```
import turtle
def rist (kk,  x,  y,  a,  b,  m = 1,
          jv = 'black',  tv = "",  w = 1):
  ''' Joonistab ristküliku. Parameetrid:
    kk – kilpkonn,  m – mastaap: pikseleid  ühikule
    x, y – algus; a, b – laius ja kõrgus
    jv – joone värv, tv – täitevärv
    w – joone paksus, vaikimisi 1 piksel    '''
  x = m * x; y =  m * y; am = m * a; bm = m * b
  kk.penup(); kk.setpos(x, y)
  kk.width(w); kk.color(jv, tv)
  kk.pendown(); kk.begin_fill()
  kk.setpos(x + am, y); kk.setpos(x+ am, y +bm)
  kk.setpos(x, y + bm); kk.setpos(x, y)
  kk.end_fill()

aken = turtle.Screen()
J = turtle.Turtle()
rist (J, -150, -300,300,400, tv = "gray")
rist (J, -170, 80,340,20, tv = "darkgray")
rist (J, 0, 200, 120, 25, jv = 'gray', tv = 'white')
rist (J, 0, 225, 120, 25, jv = 'black', tv = 'black')
rist (J, 0, 250, 120, 25, jv = 'blue', tv = 'blue')
rist (J, 0 ,100, 6, 100, tv = "brown")
# aed
rist (J,-300, -260, 590, 10, jv = 'black', tv = 'green')
x = -300; y = -280; a = 10; b = 70
for i in range(30): # lipid
  rist(J, x, y, a, b, jv = 'black', tv = 'yellow')
  x = x + 20
rist(J,-300, -220, 590, 10, jv = 'black', tv = 'brown')
aken.mainloop()
```

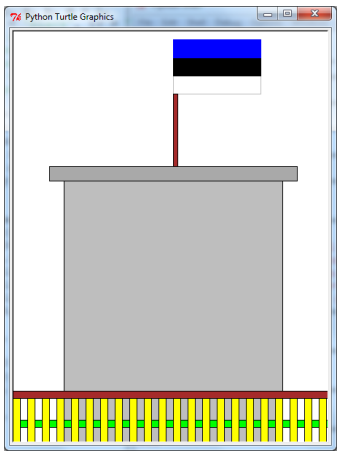
Antud juhul eeldatakse, et pöördumisel antakse **kk**-le vastav objekt (kilpkonn), vt näide.

Siin on võetud kasutusele parameetrid **x** ja **y**, mis määravad ristküliku alguspunkti: alumine vasakpoolne nurk. See võimaldab paigutada kujundi suvalisse kohta aknas.

Parameetri **m** (mastaap) vaikeväärtus on 1, st kui pöördumisel vastavale argumendile väärtust ei anta. Mõõtmed esitatakse pikslites. Enne joonistamist korrutatakse **x**, **y**, **a** ja **b** väärtused **m**-ga.

Kui pöördumisel parameetrile **tv** (täitevärv, vaikimisi tühi väärtus) vastavat argumenti ei anta, siis kujundit ei täideta

Pöördudes korduvalt protseduuri **rist** poole, saadakse selline pilt



Kasutajamoodul Inimene.py

Lihtsamal ja enamkasutatavamal juhul kujutab kasutajamoodul endast ühes failis (laiendiga **.py**) asuvat funktsioonide kogumit. Fail peab asuma seda kasutava programmiga samas kaustas või kaustas, kust Pythoni interpretator seda otsida oskab. Failis asuvad funktsioonid (protseduurid) saavad kasutatavaks käskude **import** või **from** täitmise järel.

Toome näiteks mooduli **inimene.py**, mis sisaldab funktsioone inimese omaduste leidmiseks: ideaalne mass, rasvaprotsent, ruumala jms. Vajalikud algandmed ja valemid on toodud allpool.

Algandmed:

L – pikkus (cm), **m** – mass (kg), **v** – vanus (aasta), **sugu** – mees/naine

Omadused:

$\mathbf{m_{id}}$ – ideaalne mass (kg), **r** – rasvaprotsentsus (%), **ρ** – tihedus(kg/m^3), **V** – ruumala(dm^3), **S** – pindala (m^2)

$$m_{id} = \begin{cases} (3 \cdot L - 450 + v) \cdot 0.225 + 40{,}5 \text{ naine} \\ (3 \cdot L - 450 + v) \cdot 0.250 + 45{,}0 \text{ mees} \end{cases} \quad r = \begin{cases} \frac{m - m_{id}}{m} \cdot m + 22 \text{ naine} \\ \frac{m - m_{id}}{m} \cdot m + 15 \text{ mees} \end{cases} \quad \rho = 8{,}9 \cdot r + 11 \cdot (100 - r)$$

$$V = \frac{1000 \cdot m}{\rho}, \quad S = \frac{(1000 \cdot m)^y \cdot L^{0,3}}{3118{,}2}, \text{ kus } y = (35{,}75 - \log_{10} m)/53{,}2, \; kind = m/(L/100)^2$$

Toodud funktsioonid on ühtlasi täiendavateks näideteks funktsioonide loomise ja kasutamise kohta.

Mõnel juhul on tegemist nö mitmetasemeliste funktsioonidega – üks funktsioon pöördub teise funktsiooni poole, see pöördub omakorda järgmise taseme funktsiooni poole jne.

```
import math
from datetime import *

def ideaal(L, v, sugu):
    ''' Inimese ideaalne mass (kg).
        L-pikkus (cm), v-vanus(aasta)
        sugu - n(naine), m(mees) '''
    abi = 3 * L - 450 + v
    if sugu[0].lower() == 'n' :
        mid = abi * 0.225 + 40.5
    else:
        mid = abi * 0.250 + 45.0
    return mid

def rasvaprots(L, m, v, sugu):
    ''' Rasvaprotsent(%), L-pikkus (cm),
        m - mass(kg), v-vanus(aasta),
        sugu - n(naine), m(mees) '''
    mid = ideaal(L, v, sugu)
    abi = (m - mid) / m * 100
    em = sugu[0].lower()
    if em == 'n':  return abi + 22
    else:          return abi + 15
```

Moodul **datetime** sisaldab meetodeid ja funktsioone tegevuste täitmiseks kuupäevade ja kellaegadega.

Tingimuse kontrollimisel eraldatakse parameetri sugu väärtusest esimene märk (indeksiga 0): sugu[0], teisendatakse see meetodiga lower() väiketäheks ja võrreldakse konstandiga *'n'*. See võimaldab esitada soo ühe väike- või suurtähega (n, m, N, M) või sõnaga (näit naine, Naine, mees, …) . Tegelikult omab tähtsust ainult see, kas esimene või ainukene täht on *'n'* (väike või suur) või mitte. Kui jah, eeldatakse, et tegemist on naisega, kui on midagi muud, kasutatakse mehe jaoks ette nähtud valemit.

Funktsioon **rasvaprots** pöördub eelmise funktsiooni poole ideaalse massi (**mid**) leidmiseks, kasutades argumentidena parameetrite kaudu saadud väärtusi.

Siin omistatakse parameetri **sugu** väiketäheks teisendatud esimene märk muutujale **em**, et võrdlus oleks lühem.

Tagastatav väärtus esitatakse avaldisena otse **return**-lauses.

```
def tihedus(L, m, v, sugu):
    ''' Tihedus (kg/m3). L-pikkus (cm),
        m - mass(kg); v - vanus(aasta)
        sugu - n(naine), m(mees) '''
    rasv = rasvaprots(L, m, v, sugu)
    return 8.9 * rasv + 11 * (100 - rasv)
```

Funktsioon **tihedus**, mille poole pöördutakse mõnest teisest protseduurist (näiteks funktsioonist **ruumala**), pöördub funktsiooni **rasvaprots** poole. Viimane pöördub omakorda funktsiooni **ideaal** poole.

```
def ruumala (L, m, v, sugu):
    ''' Ruumala (dm3/ltr). L-pikkus (cm),
        m - mass(kg); v-vanus(aasta)
        sugu - n(naine), m(mees) '''
    ro = ro = tihedus(L, m, v, sugu)
    return 1000 * m / ro
```

Siin toimub juba neli järjestikust pöördumist, kui arvestada ka pöördumist antud funktsiooni poole.

Selline suur pöördumiste ahel ei pruugi alati sobida, sest teeb funktsioonid üksteisest sõltuvaks. Vajadusel võib ühendada tegevused ühte protseduuri (vt **ruumala2**)

```
def ruumala2(L, m, v, sugu):
    ''' Ruumala (dm3/ltr). L-pikkus (cm),
        m - mass(kg); v-vanus(aasta)
        sugu - n(naine), m(mees) '''
    abi = 3 * L - 450 + v
    if sugu[0].lower() == 'n':
        mid = abi * 0.225 + 40.5
        rasv = (m - mid) / m * 100 + 22
    else:
        mid = abi * 0.250 + 45.0
        rasv = (m - mid) / m * 100 + 15
    ro = 8.9 * rasv + 11 * (100 - rasv)
    return 1000 * m / ro
```

Selles näites toodud funktsioon ühendab endas tegevused, mis olid eespool realiseeritud eraldi funktsioonidena. Nüüd on kõik tegevused ühes funktsioonis ning teisi funktsioone ei lähe vaja.

Soovi korral võib lisada tagastatavate väärtuste hulka ka **mid**, **rasv** ja **ro** väärtused.

```
def pindala(L, m):
    ''' Pindala (m2). L-pikkus (cm),
        m - mass(kg) '''
    y = (35.75 - math.log10(m)) / 53.2
    S = (1000 * m)**y * L ** 0.3 / 3118.2
    return S
```

Siin toimub lihtsalt väärtuste leidmine omistamislausete abil. Kasutataks funktsiooni **log10** moodulist **math**.

```
def saledus(L, m):
    ''' Kehamassi indeks ja hinnang.
        L-pikkus(cm),m - mass(kg) '''
    kind = m / (L/100)**2
    if   kind < 18:     hind = "kõhnake"
    elif kind <= 25:    hind = "normis"
    elif kind <= 30:    hind = "ülekaal"
    else:               hind = "suur ülekaal"
    return kind, hind
```

Selle funktsiooni analoog oli esimeses näiteprogrammis **Tutvus**. Praegu on kehamassiindeksi ja selle põhjal hinnangu andmine pandud ühte funktsiooni, mis tagastab kaks väärtust.

```
def sugu(kood):
    ''' isikukoodi alusel
    tagastab sugu'''
    em = kood[0]
    if int(em) % 2 == 0:
        return 'naine'
    else:
        return 'mees'
```

Siin ning järgnevates funktsioonides kasutatakse viitamist stringide elementidele indeksite abil, vt jaotist „Stringid (sõned) ja stringavaldised“, lk 40.

Isikukoodi esimene number näitab isiku sugu ja ka sünniaja sajandit. Paaritud numbrid (1, 3, 5) näitavad sugu „mees“, paarisnumbrid (2, 4, 6) „naine“.

```
def saeg(kood):
  ''' sünnikuupüev
     isikukoodi alusel'''
  en = kood[0]
  if en == '1' or en == '2' :
      ab = 1800
  elif en == '3' or en == '4' :
      ab = 1900
  elif en == '5' or en == '6' :
      ab = 2000
  else:
      return 0  # olematu sajand
  aasta2 = int(kood[1:3])
  aasta = ab + aasta2
  kuu = int(kood[3:5])
  paev = int(kood[5:7])
  skp = date(aasta, kuu, paev)
  return skp
```

Funktsioon tagastab isiku sünnikuupäeva kujul:

aasta – kuu – päev näiteks: 1989 – 03 – 25

Isiku sünniaasta kaks viimast numbrit on koodi 2. ja 3. positsioonis. Esimesed kaks numbrit (seotud sajandiga) on kodeeritud esimeses numbris: 1 ja 2 – 1800, 3 ja 4 – 1900, 5 ja 6 – 2000. Funktsioon eraldab kõigepealt koodi esimese numbri (indeks 0) ja omistab selle muutujale **en**. Edasi omistatakse sõltuvalt **en** väärtusest muutujale **ab** väärtus 1800, 1900 või 2000. Sellele liidetakse arv, mis saadakse koodi 2. ja 3. numbri alusel.

NB! Viitamist stringi lõikudele vt jaotisest „Stringid (sõned) ja stringavaldised“.

Mooduli **datetime** funktsioon **date** moodustab kuupäeva kolmest osast koosneva liitväärtusena. Väärtus tagastatakse **return**-lausega.

```
def vanusK(kood):
    ''' vanus aastates ja püevades
        isikukoodi alusel '''
    sa = saeg(kood)
    tana = date.today()
    vp = (tana - sa).days  # vanus päevades
    va = vp/365.25         # vanus aastades
    return va, vp
```

Funktsioonid **vanusK** ja **vanusA** tagastavad inimese vanuse aastates ja päevades kasutades vastavalt isikukoodi või sünnikuupäeva.

Funktsioonides kasutatakse mooduli **datetime** vahendeid.

```
def vanusA(paev, kuu, aasta):
    ''' vanus aastates ja püevades
        sünnikuupäeva alusel '''
    sa = date(aasta, kuu, paev)
    tana = date.today()
    vp = (tana - sa).days  # vanus päevades
    va = vp/365.25         # vanus aastades
    return va, vp
```

Siin antakse parameetritena isiku sünni kuupäev, kuu ja aasta. Funktsiooniga **date** moodustatakse kuupäev standardvormingus, mida saab kasutada tehetes kuupäevadega.

Järgnevalt on toodud näide mooduli kasutamise kohta.

```
from inimene import *
def isik():
    ik = input("Anna oma isikukood ")
    while len(ik) != 11:
        ik = input ("Peab olema 11 märki! Anna uus ")
    L = float(input("pikkus "))
    m = float(input("mass "))
    s = sugu(ik)
    va, vp = vanusK(ik)
    imas = ideaal(L, va, s)
    ruum = ruumala (L, m, va, s)
    print ("Oled sündinud", saeg(ik))
    print ("Sinu vanus on", round(va), "aastat")
    print ("Ideaalne mass", round(imas, 2))
    print ("ruumala", round(ruum, 2))
    kind, hind = saledus(L, m)
    print ("massiindeks", round(kind), "oled", hind)

isik()
```

Tuletame meelde, et moodul **inimene.py**, mille funktsioonid on toodud ülalpool, peab asuma samas kaustas, kus on programm. Importimiseks võib kasutada käsku **import** või **from**. Siin kasutatakse käsku **from**, mille puhul pole vaja viitamist moodulile.

Programm loeb algandmed:
isikukood (**ik**), pikkus (**L**), mass (**m**), sugu (**s**)

Ja teeb mõned pöördumised mooduli **inimene** funktsioonide poole ning kuvab tulemused.

Kasutajamoodul funktsioon.py

Moodul sisaldab funktsioone, mis võimaldavad leida kasutaja poolt antud ühemuutuja funktsiooni F(x) mitmesuguseid omadusi (maksimaalne ja minimaalne väärtus ning nende asukohad, integraal ja pindala, nullkohad) etteantaval lõigul ja joonistada funktsiooni graafiku.

```
def F_max(F, a, b, n):
    ''' Funktsiooni F maks
        ja selle asukoht vx '''
    h = (b - a) / n
    maxi = F(a); vx = a
    for i in range(n + 1):
        x = a + i * h
        y = F(x)
        if  y > maxi:
            maxi = y
            vx = x
    return maxi, vx
```

```
def F_min(F, a, b, n):
    ''' Funktsiooni F min
        ja selle asukoht vx '''
    h = (b - a) / n
    mini = F(a); vx = a
    for i in range(n + 1):
        x = a + i * h
        y = F(x)
        if y < mini:
            mini = y
            vx = x
    return mini, vx
```

Parameeter **F** on nii selles kui ka järgmistes protseduurides antud punktis funktsiooni väärtuse leidmise protseduuri nimi.

Protseduurid on peaaegu identsed, ainuke erinevus on **if**-lause võrdlusmärgis.

Mõlemad funktsioonid tagastavad kaks väärtust: **maxi** või **mini** ja **vx** (vastav x).

Väärtusi **maxi** ja **mini** kasutatakse koordinaatsüsteemi määramisel graafikaakna jaoks.

```
def integraal(F, a, b, n):
    ''' Määratud integraal
        trapetsivalemiga'''
    h = (b - a) / n
    if h <= 0: return
    S = (F(a) + F(b)) / 2
    for i in range(1, n):
        S = S + F(a + i * h)
    return h * S
```

```
def pindala(F, a, b, n):
    ''' Pindala
        trapetsivalemiga'''
    h = (b - a) / n
    if h <= 0: return
    S = (abs(F(a))+abs(F(b))) / 2
    for i in range(1, n):
        S = S + abs(F(a + i * h))
    return h * S
```

Näites on tegemist sarnaste funktsioonidega, milles kasutatakse trapetsivalemit:

$S = h(y_0/2 + y_1 + y_2 + \ldots + y_{n-1} + y_n/2)$

Määratud integraal kujutab endast pindala – arvestatakse funktsiooni väärtuste märki.

Pindala leidmisel kasutatakse absoluutväärtusi.

```
def nullid(F, x0, xn, n,
           eps = 0.00001) :
    ''' Nullkohad lõigul [x0, xn],
        poolitusmeetodiga'''
    h = (xn – x0) / n
    y1 = F(x0)
    if y1 == 0 : print (x0)
    for i in range(1, n+1) :
        x = x0 + i * h
        y2 = F(x)
        if  y2 == 0 : print (x)
        if y1 * y2 < 0 :
            xk=F_pool(F, x-h, x, eps)
            print (round(xk, 3))
        y1 = y2
```

```
def F_pool(F, a, b, \
           eps = 0.00001) :
    ''' Nullkoht lõigul [a, b],
        poolitusmeetod '''
    y1 = F(a)
    while b - a > eps :
        c = (a + b) / 2
        y2 = F(c)
        if y2 == 0: return c
        if y1 * y2 > 0 :
            a = c
        else :
            b = c
    return c
```

Funktsioon **F_pool**() leiab (täpsustab) nullkoha lõigul [a; b] etteantava täpsusega **eps**. Eeldatakse, et nullkoht antud lõigul on olemas. Kasutatakse poolitusmeetodit.

Protseduur nullid() leiab kõik nullkohad lõigul [x0, xn]. Järjest arvutatakse ja võrreldakse funktsiooni naabervääртusi. Kui need on erineva märgiga (y1*y2 < 0), on vahemikus nullkoht. Selle väärtuse täpsustamiseks kasutatakse funktsiooni **F_pool**.

```
def fungraaf(F, x0, xn, n, w = 2):
    ''' funktsiooni F graafik lõigul [x0; xn] '''
    mini, vx =  F_min (F, x0, xn, n)
    maxi, vx =  F_max(F, x0, xn, n)
    setworldcoordinates(x0, mini, xn, maxi)
    penup(); setpos (x0, 0)    # X - telg
    pendown(); setpos(xn,0)
    penup(); setpos (0, mini)  # Y - telg
    pendown(); setpos(0,maxi)
    x = x0   # jaotised X-teljel
    while x < xn: # jaotised X-teljel
        penup();   setpos(x,-0.2)
        pendown(); setpos(x,0.2)
        x = x + 1
    # graafik
    h = (xn - x0)/n; pencolor('blue')
    penup(); setpos(x0,F(x0))
    pendown(); width(w)
    for k in range(n + 1):
        x = x0 + k * h
        y = F(x)
        setpos(x, y)
```

```
def funtab(F, x0, xn, n):
    ''' funktsiooni F tabel
        lõigul [a; b] '''
    h = (xn - x0) / n
    for k in range(n + 1):
        x = x0 + k * h
        y = F(x)
        print (round (x, 2), "\t", round(y, 4))
```

Protseduur **funtab** arvutab ja kuvab Shelli aknas antud lõigul uuritava funktsiooni väärtused.

Protseduur **fungraaf** joonistab funktsiooni graafiku antud lõigule. Oluline roll on siin mooduli **turtle** meetodil:

setworldcoordinates(x0, mini, xn, maxi),

mis määrab kasutaja koordinaatsüsteemi, näidates akna alumise vasakpoolse nurga (x0, mini) ja ülemise parempoolse nurga (xn, maxi) väärtused.

Protseduur joonistab koordinaatteljed ja jaotised X-teljele.

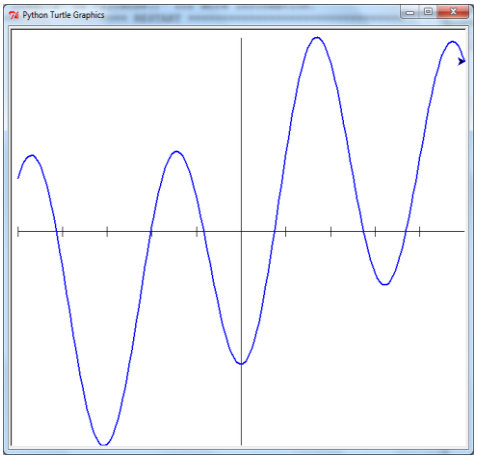
Graafiku enda joonistamine toimub **for**-lause juhtimisel, kus muutuja x väärtusi muudetakse järjest ja arvutatakse muutuja y väärtused ning joonistatakse vastavad lõigud.

Näide funktsiooni kasutamise kohta, vt „Kasutajamoodul funktsioon.py“, lk 36.

```
from math import *
from turtle import *
from funktsioon import *

def F1(x):
  return 3*sin(x /2)+5 * cos(2 * x+3)

a = -5; b = 5; n1 = 10; n2 = 200
funtab (F1, a, b, n1)
mini, x1 = F_min(F1, a, b,n2)
maxi, x2 = F_max(F1, a, b,n2)
print ("Y min:", round(mini, 3),
                round(x1, 2))
print ("Y max:", round(maxi, 3),
                 round(x2, 2))
integ = integraal(F1, a, b, n2)
pind = pindala(F1, a, b, n2)
print ("Integ:", round(integ,3))
print ("pind:", round(pind,3))
nullid(F1, a, b,n2)
fungraaf (F1, a, b, n2)
mainloop()
```

Kasutaja võib defineerida Pythoni funktsiooni matemaatilise ühemuutuja funktsiooni väärtuse leidmiseks antud punktis x. Selle funktsooni nimi (siin **F1**) antakse argumendina pöördumisel mooduli funktsioonide poole.

Tulemuseks on järgmine graafik:




```
-5.0      1.9741
-4.0      -1.3096
-
3.0       -1.5632
4.0       2.75
5.0       6.3327
Y min: -7.998 -3.07
Y max: 7.22 1.69
Integ: 2.691
pind: 35.778
-4.134
-2.019
-0.838
0.746
2.741
3.69
```

Märkandmed ja tegevused nendega

Pythoni programmides kasutatakse kolme liiki märkandmeid:

- arvud: täisarvud – klass int, reaalarvud ehk ujupunktarvud – klass float,
- tekstid (stringid ehk sõned)– klass str,
- tõeväärtused – klass bool.

Väärtuse liigist sõltuvad ka võimalikud tegevused andmetega.

Avaldised ja funktsioonid

Avaldis määrab ära tehted (operatsioonid) ja nende sooritamise järjekorra.

Avaldiste struktuur ja liigid

Üldjuhul koosneb avaldis:

- operandidest
- operaatoritest ehk tehtesümbolitest
- ümarsulgudest

Erijuhul võib avaldis koosneda ainult ühest operandist. Operandideks võivad olla:

- konstandid
- lihtmuutujad
- struktuurmuutujate elemendid
- funktsooniviidad ja meetodid

Lihtmuutujad esitatakse nimede abil, struktuurmuutujate elementide esitus sõltub andmekogumi liigist.

Funktsiooniviit ja **meetod** esitatakse kujul:

[***objekt***.] ***nimi*** ([*argumendid*]),

kus ***nimi*** on Pythoni sisefunktsiooni või kasutajafunktsiooni nimi. Sisefunktsioonide nimed ei kuulu reserveeritud võtmesõnade hulka, kuid muuks otstarbeks neid kasutada ei ole mõistlik, vältimaks segaduse tekkimist. Funktsiooniviites esinev ***argument*** näitab funktsioonile edastatavat väärtust. Argumendid võivad olla esitatud avaldiste abil. Argumentide arv, tüüp ja esitusjärjekord sõltuvad konkreetsest funktsioonist.

Tehted ehk **operaatorid** jagunevad nelja rühma:

- **aritmeetikatehted**: ** , * , / , // , % , + , -
- **stringitehted** +, *
- **võrdlustehted**: == , != , < , <= , > , >=
- **loogikatehted**: not, and, or

Üldjuhul võib ühes ja samas avaldises esineda tehteid kõikidest liikidest. Avaldise väärtuse leidmisel arvestatakse tehete prioriteete liikide vahel ning aritmeetika- ja loogikatehete puhul ka liigi sees. Tehete liigid on siin esitatud prioriteetide kahanemise järjekorras. Aritmeetika- ja loogikatehete prioriteedid kahanevad vasakult paremale. Avaldises **a + b > c** and **a + c > b** and **b + c > a** esinevad aritmeetika-, võrdlus- ja loogikatehted.

Väärtuse leidmisel täidetakse kõigepealt aritmeetika-, siis võrdlus- ning lõpuks loogikatehted.

Tehete järjekorra muutmiseks võib kasutada ümarsulge, kusjuures sulgudes oleva avaldise väärtus leitakse kõigepealt (eraldi). Ümarsulgudes esitatakse ka funktsiooniviidete ja meetodite argumendid.

Sõltuvalt andmete liigist ning kasutatavatest tehetest ja leitava väärtuse liigist võib avaldised jagada järgmistesse rühmadesse:

- arvavaldised
- stringavaldised
- loogikaavaldised

Arvavaldised ja matemaatikafunktsioonid

Arvavaldiste operandide väärtusteks on arvud ning neis kasutatakse aritmeetikatehteid ning funktsioone, mis tagastavad arvväärtusi.

Aritmeetikatehted ja nende prioriteedid on järgmised.

Prioriteet	**Tehte sümbolid**	**Selgitus**
1	**	astendamine a**n
2	-	unaarne miinus -a * -b + c
3	* **ja** /	korrutamine ja jagamine a * b, a / b
3	//	täisarvuline jagamine a // b, 13 // 5 = 2
3	%	jagatise jääk a % b, 13 % 5 = 3
4	+ **ja** -	liitmine ja lahutamine a + b, a - b

Võrdse prioriteediga tehteid täidetakse järjest vasakult paremale. Erandiks on astendamine, kus tehteid täidetakse paremalt vasakule. Tehete järjekorda saab reguleerida ümarsulgudega.

Tehete prioriteetide rakendamise näiteid:
-3**2 * 5 + 18 / 2 * 3 = -9 * 5 + 9 * 3 = -45 + 27 = -18,

(-3)**2 * 5 + 18 / (2 * 3) = 9 * 5 + 18 / 6= 48

4**2**3 = 48 = 65 536 64**1/3 = 64^1/3 = 64/3 64**(1/3) = 4

Matemaatikafunktsioonid ja konstandid

Matemaatikafunktsioonid kuuluvad moodulisse **math** ja esitatakse programmis kujul **math**.nimi(a) või nimi(a), olenevalt mooduli importimise viisist.

sqrt(a)	ruutjuur sqrt(b**2 - 4*a*c) = (b**2 - 4*a*c)**(1/2)
log(a)	naturaallogaritm (ln a) log(a) / log(10) = $\log_{10}$a.
log10(a)	kümnendlogaritm
exp(a)	e^a (e = 2,71828…) (exp(-x) + exp(2 * x)) / 2 = (e^{-x} + e^{2x}) / 2
abs(a)	absoluutväärtus abs ((a - x) / (a + x))
sin(a), cos(a), tan(a)	sin(x)+cos(2*x)+tan(x**2)-cos(2*x)**2 **NB!** argument on radiaanides
asin(a), acos(a), atan(a)	arkusfunktsioonid. Radiaanides (-π/2<x<π/2). atan(a/sqrt(1-a**2))=asin a , 4*atan(1) = π
pi, e	pi = π, e – naturaallogaritmi alus: 2.718281828459045

Teisendusfunktsioonid

str(a)	teisendus stringvormingusse
int(a)	teisendus täisarvuks
float(a)	teisendus reaalarvuks
round(a, n)	ümardamine: round (13.74615, 2) = 13.75
trunc(x)	täisosa
ceil(x)	vähim

Stringid (sõned) ja stringavaldised

Stringavaldiste operandide väärtusteks on stringid (sõned), milles võib kasutada stringitehteid ja stringifunktsioone. Stringitehet **+** nimetataks ka **sidurdamiseks**. See võimaldab ühendada stringe ja ka arve. Viimased peab teisendama funktsiooniga **str** tekstivormingusse.

Näiteid:
"Peeter" + " " + "Kask" annab Peeter Kask, str(35.7) + ″ ″ + str(2.5) annab 35.7 2.5

Kui S=5378.75, x1=2.538, x2=-1.34, siis

"Summa= " + str(S) annab Summa= 5378.75,

"x1= " + str(x1) + " x2= " + str(x2) => x1= 2.538 x2= -1.34

Stringandmete jaoks on rida funktsioone ja meetodeid. Stringe (sõnesid) käsitletakse järjestatud märkide jadana. Olgu näiteks antud stringid:

s = *'See on string'* ja isik = *"Juku Naaskel"*

Neid võib kujutada järgmiste märkide jadadena:

s

S	e	e		o	n		s	t	r	i	n	g
0	1	2	3	4	5	6	7	8	9	10	11	12

isik

J	u	k	u		N	a	a	s	k	e	l
0	1	2	3	4	5	6	7	8	9	10	11

Märkide nummerdamine algab alati nullist. Funktsioon **len**(string) annab märkide arvu stringis

len(*'Juku'*) = > 4; len(s) => 13; len(isik) => 12

Viitamiseks üksikule märgile kasutatakse nime koos indeksiga: nimi [indeks]

s[0] => S, s[2] => e, s[5] => n, s[12] => g

isik[0] => J, isik[2] => k, isik[5] => N, isik[11] => l

Saab viidata stringi osadele, kasutada stringi **lõikeid** (*slices*). Lõige esitatakse kujul:

nimi [m : n]

siin **m** on lõigu algus (kaasarvatud), **n** – lõigu lõpp (väljaarvatud) – tegelikult on lõpp (n - 1).

s[0 : 3] => *'See'*, s[4 : 6] => *'on'*,

s[7 :] => *'string'*, s[: 7] = > *'See on '*

Stringe saab omavahel liita (+) ja korrutada (*) arvuga

'Tere, ' + isik[0 : 4] + *'!'* => *'Tere, Juku!'*

isik[: 4] + *' on '* + 3 * *'väga '* + *'tubli!'* => *'Juku on väga väga väga tubli!'*

NB! Stringi väärtust saab muuta ainult tervikuna!

isik = *'Kalle'*; isik = s - on võimalikud

isik[0] = *'k'* , isik[5 :] = *'Kärmas'* - annavad vea!

Võib luua uue stringi taoliselt:

isik2=*“”*; isik2 = isik2 + isik[: 4] + *' Kärmas'* => *'Juku Kärmas'*

Operatsioon (tehe) **str1 in str2** annab tulemuseks **True**, kui str1 sisaldub str2-s.

Näiteks lause **if** 'a' **in** tekst: k = k + 1 suurendab muutuja k väärtust kui täht 'a' on antud tekstis olemas.

Mõned stringide meetodid

Esitatakse kujul: **string**.meetod()

count (*a_str*)	loendab alamstringi esinemise arvu stringis
find (*a_str*)	leiab alamstringi (*a_str*) alguse (indeksi) stringis
lower (), upper ()	teisendab teksti koopia väike- või suurtähtedeks
lstrip(), rstrip(), strip()	eemaldab juhtsümbolid vasakult, paremalt või kõik
split()	jagab lause sõnadeks, moodustades loendi

"Loendab alamstringi esinemise arvu stringis".**count**(*"string"*) => 2

Tühiku asukoht stringis isik: isik.**find**(" ") => 4

Eesnime ja perenime eraldamine stringist isik:

eesnimi = isik[0 **:** isik**.find**(" ")]; perenimi = isik[isik**.find**(" ") + 1 **:**]

eesnimi – lõige algusest (0) kuni tühikuni, perenimi – lõige peale tühikut stringi lõpuni.

*"Juku"***.upper**() => *'JUKU'*

" *tekst* "**.strip**() = 'tekst'

>>> "Jagab lause sõnadeks, moodustades loendi".**split**()

[*'Jagab'*, *'lause'*, *'sõnadeks,'*, *'moodustades'*, *'loendi'*]

Erisümbolid

Teksti sees võivad olla mitteprinditavad juhtsümbolid, mis avaldavad teatud mõju teksti väljanägemisele. Need esitatakse teksti sees sümboli **** (langkriips) järel. Taolisi sümboleid on mitmeid, siin nimetame ainult paari:

\n - uus rida printimisel või kirje lõpp failis

\t - tabuleerimine printimisel

```
print ("Igaüks\neraldi\nreale");   # iga sõna prinditakse eraldi reale
print (x, "\t", y, "\t", z);          # jäävad suuremad vahed
```

Paar näidet. Funktsioon **mitu_tais** teeb kindlaks täishäälikute arvu antud tekstis, funktsioon **ilma_tais** tagastab antud teksti (stringi) ilma täishäälikuteta.

```
def mitu_tais(txt):
    ''' täishäälikute arv '''
    tais = 'AEIOUÕÄÖÜ'
    mitu = 0
    for t in txt:
        if t.upper() in tais:
            mitu += 1
    return mitu
```

```
def ilma_tais(txt):
    ''' ilma täishäälikuteta '''
    tais = 'AEIOUÕÄÖÜ'
    uus = ''
    for t in txt :
        if t.upper() not in tais:
            uus += t
    return uus
```

Mõlemas funktsioonis on moodustatud täishäälikuid sisaldav stringkonstant **tais**, mis sisaldab ainult suurtähti. Antud tekstis aga võivad olla nii suur- kui ka väiketähed, mida arvestatakse meetodi **upper** kasutamises.

```
tekst = input("Anna tekst ")
n = mitu_tais(tekst)
print ("Täishäälikuid oli:", n)
print (ilma_tais(tekst))
```

Funktsioonides tasub panna tähele **for**- ja **if**-lauseid. Kuna string **txt** kujutab endast jada, siis kontrollitakse järjest selle kõikide märkide sisalduvust stringis **tais**, kasutades **if**-lauses operaatorit **in**.

Vt ka näiteid moodulis „Inimene“, lk 35.

Võrdlused ja loogikaavaldised

Võrdlused on käsitletavad loogikaavaldiste erijuhtudena, nende kuju on järgmine:

avaldis1 **tehtesümbol** *avaldis2*

Tehtesümbolid (operaatorid) on järgmised: **==** , **!=** , **<** , **<=** , **>** , **>=**

Avaldised *avaldis1* ja *avaldis2* on arv- või stringavaldised. Ühes võrdluses esinevad avaldised peavad kuuluma samasse liiki. Võrdluse tulemiks on alati tõeväärtus **True** (tõene) või **False** (väär). Võrdluste näiteid

x <= 0, b * b - 4*a*c < 0, x * x + y * y > r * r, vastus.upper() == "EI"

NB! Stringide võrdlemisel eristatakse suur- ja väiketähti!

Loogikaavaldise üldkuju on järgmine:

avaldis **LTS** *avaldis* {**LTS** *avaldis*}

Siin on *avaldis* võrdlus või loogikaavaldis ja **LTS** loogikatehte operaator. Peamised loogikaoperaatorid on **or**, **and** ja **not**. Nende tähendused on:

or – või: tehte a **or** b väärtus on tõene (**True**), kui vähemalt ühe operandi väärtus on tõene, vastupidisel juhul on tulem väär (**False**).

and – ja: tehte a **and** b tulem on tõene (**True**) ainult siis, kui mõlema operandi väärtused on tõesed, vastupidisel juhul on tehte tulem väär (**False**).

not – mitte: tehte **not** a tulem on tõene (**True**) siis, kui a väärtus on väär (**False**) ja väär (**False**) vastupidisel juhul.

Loogikaavaldiste näiteid:

x >= 2 **and** x <= 13, x < 2 **or** x > 13 , a + b > c **and** a + c > b **and** b + c > a

Omistamine ja omistamislause

Omistamine on üks fundamentaalsemaid tegevusi, mida arvuti saab programmi toimel täita. See seisneb mingi väärtuse salvestamises arvuti sisemälus etteantud väljas või pesas. Välja (pesa) eelmine väärtus, kui see oli, kaob.

Üheks peamiseks vahendiks väärtuse salvestamiseks (omistamiseks) on kõikides programmeerimiskeeltes omistamislause. Tüüpiliselt eelneb väärtuse salvestamisele selle leidmine (tuletamine) etteantud avaldise abil. Kusjuures avaldise väärtuse leidmisega kaasneb enamasti muutujate ja/või omaduste varem salvestatud väärtuste lugemine. Omistamine võib kaasneda ka mõnede muude lausete ja meetodite täitmisele.

Omistamise ja vastavate omistamislausete olemus on praktiliselt sama. Ühelt poolt kujutab muutujale eraldatav mäluväli (pesa) endast objekti, mille omadust *väärtus* omistamislause täitmisel muudetakse. Objekti igale omadusele eraldatakse omaduste vektoris mäluväli, kuhu omistamisel salvestatakse vastav väärtus.

Omistamislause põhivariant on järgmine:

muutuja = *avaldis*

Ühe lausega saab omistada väärtused ka mitmele muutujale:

muutuja [, *muutuja*]… = *avaldis* [, *avaldis*]…

Igale muutujale erinev väärtus: y, z, w = x, 2 * x + 3, sin(x) + cos(x)

muutuja [= *muutuja*]… = *avaldis*

Üks väärtus mitmele muutujale: PS = pn = NS = nn = 0

Muutuja esitatakse nime abil. Tuletame meelde, et avaldiste operandideks võivad olla konstandid, muutujad, objektide omadused, massiivi elemendid, funktsiooniviidad (funktsioonid) sisefunktsioonidele või kasutaja koostatud funktsioonidele. Avaldis võib koosneda ka ainult ühest operandist.

Omistamislausete näiteid

k = 0; x = 2.5; v = w = z = 13; nimi = *"A. Kask"*

Erijuht: muutujale omistatakse konstandi väärtus.

x = y ; t = v; ymax = y

Erijuht: muutujale omistatakse teise muutuja väärtus, st paremas pooles oleva muutuja väärtus kopeeritakse vasakus pooles oleva muutuja väärtuseks.

x = a + i * h ; y = 3 * math.sin(x) – math.sqrt(x^4 + 5)

Üldjuht: leitakse paremas pooles oleva avaldise väärtus ja tulemus omistatakse vasakus pooles olevale muutujale, st salvestatakse antud muutuja mäluväljas (pesas).

k = k + 1; S = S + y;: n = n – k; F = F * k

k += 1; S += y; n -= k; F *= k *# võib kasutada lühendatud variante*

Erijuht: sama muutuja esineb omistamislause vasakus ja paremas pooles. Tegemist on uue väärtuse leidmisega ja asendamisega eelmise (jooksva) väärtuse alusel. Näiteks lause k = k + 1 täitmisel loetakse k jooksev väärtus, liidetakse sellele 1 ja saadud tulemus võetakse k uueks väärtuseks, st k väärtust suurendatakse ühe võrra.

S, P, d = a * b, P = 2 * (a + b), math.sqrt(a**2 + b**2)

Lauses leitakse kolm väärtust ja omistatakse muutujatele S, P ja d.

Lause **a, b = b, a** vahetab muutujate **a** ja **b** väärtused.

Omistamislaused **2 = x** ja **x + 5 = y** on aga täiesti mõttetud ja lubamatud! Ei saa omistada väärtust konstandile või avaldisele ehk salvestada (!) konstandis või avaldises midagi. Omistamislause vasakus pooles võib olla ainult muutuja nimi või objekti omadus!

Andmete väljastamine ekraanile

Andmete väljastamisega (öeldakse ka kuvamisega, printimisega, trükkimisega) ja sisestamisega (öeldakse ka lugemisega) tuleb vähemal või suuremal määral kokku puutuda peaaegu iga rakenduse loomise juures. Programmid peavad tüüpiliselt väljastama tulemusi ja harilikult vajavad nad ka algandmeid ülesande lahendamiseks. Tegemist on vastusuunaliste, kuid teatud mõttes sarnaste tegevustega. Pythonis on rikkalik vahendite kogum andmete sisestamiseks ja väljastamiseks erineval kujul. Järgnevalt vaadeldakse vaid kõige lihtsamaid võimalusi.

Andmete väljastamiseks Shelli aknasse saab kasutada funktsiooni **print**:

print ([*argument* [, *argument*]…])

Lihtsaimal juhul võib funktsioonil (lausel) olla kuju: **print()** ja see väljastab tühja rea.

Üsna sageli kasutatakse funktsiooni teadete väljastamiseks.

Järgnevas näites on kasutusel mitu stringkonstanti:

print(*"Tere,"* , *'mina olen Python.'* , *"Mis on Sinu nimi?"*).

Siin on tegemist kolme argumendiga (stringiga). Kuvamisel lisab Python väärtuste vahele ühe tühiku. Sama pildi saame, kui kasutame ühte argumenti:

print (*“Tere, mina olen Python. Mis on Sinu nimi?“*).

Tavaliselt on argumentideks mitu väärtust ning konstandid, muutujad ja avaldised võivad esineda läbisegi:

print (*"laius="*, a, *"kõrgus="*, b, *"pindala="*, a * b)

On olemas vahendid vormindamiseks, kuid neil me ei peatu. Üheks praktiliseks probleemiks võib olla arvu üleliia suur murdosa pikkus, sellisel juhul võib kasutada ümardamist funktsiooni **round**() abil. Näiteks:

print (*"x="*, **round**(x, 2), *"y="*, **round**(y, 4))

Kasulikuks võimaluseks on spetsiaalne element end = " " argumentide loetelu lõpus; see blokeerib ülemineku uuele reale pärast antud rea väljastamist.

Allpool toodud laused kirjutavad tulemused ühele reale järgmiselt: 0 1 4 9 16 25 36 49 64 81

for k **in range**(10):

  print (k * k, end = " ")

Andmete sisestamine klaviatuurilt

Andmete sisestamiseks saab kasutada funktsiooni **input**(), mille põhivariant on järgmine:

muutuja = **input** ([teade])

teade (ei ole kohustuslik) võib olla esitatud tekstikonstandi või -avaldise abil, näiteks:

vastus = **input**(str(a) + " + " + str(b) + " = ")

Arvud peavad taolistes avaldistes olema teisendatud funktsiooniga **str**() tekstivormingusse.

Selle täitmisel kuvatakse **teade** (kui on) ja programm jääb ooteseisu. Peale seda, kui kasutaja sisestab väärtuse ja vajutab klahvile **Enter**, omistatakse väärtus muutujale ja töö jätkub. Kui kasutaja vajutab kohe Enter-klahvile omistatakse muutujale tühi väärtus.

Oluline on silmas pidada, et funktsiooniga **input** sisestatud väärtused on esitatud **tekstivormingus**.

Proovige käsuaknas (Shelli aknas) mõningaid sisestamisega seotud tegevusi.

```
>>> x = input("x = ")
x = tere
>>> print (x, type(x))
tere <class 'str'>
>>>
```

Vastuseks funktsiooniga **input** kuvatud teatele sisestatakse tekst **tere**. Funktsioon **print** kuvab x-i väärtuse ja selle tüübi:

tere <class 'str'>

Võiks öelda, et kõik on ootuspärane.

```
>>> x = input("x = ")
x = 13
>>> print (x, type(x))
13 <class 'str'>
>>>
```

Vastuseks funktsiooniga **input** kuvatud teatele sisestatakse arv **13**. Funktsioon **print** kuvab x-i väärtuse ja selle tüübi:

13 <class 'str'>

Nagu näeme, on x väärtuse tüübiks samuti 'str'.

```
>>> x = input("x = ")
x =
>>> print (x, type(x))
 <class 'str'>
>>>
```

Vastuseks funktsiooniga **input** kuvatud teatele ei sisestata midagi, vaid vajutatakse kohe klahvile Enter. Funktsioon **print** ei kuva x kohal midagi (tegemist on tühja väärtusega), tüübiks on ikka 'str':

<class 'str'>

```
>>> x = input("x = ")
x = 13
>>> x / 3
…
  x / 3
TypeError: unsupported operand
type(s) for /: 'str' and 'int'
>>>
```

Vastuseks funktsiooniga **input** kuvatud teatele sisestatakse arv **13**. Edasi proovitakse jagada x väärtust 3-ga. Vastuseks kuvab Python veateate, et operandid tüübiga 'str' ja 'int' ei sobi tehte / jaoks.

Lausega **input** sisestatud väärtus on seega alati tekstivormingus (tüüp **str**). Selliste arvväärtustega arvutusi teha ei saa ning sellepärast kasutataks sageli teisendusfunktsioone **int** ja **float**, näiteks:

n = **int** (**input**(*"Mitu küsimust?"*)); a = **float**(**input**(*"kaugus"*))

NB! Kui teisendamiseks kasutatakse funktsiooni **int**, kuid sisestatav väärtus on aga reaalarv (arvul on murdosa), siis tekib viga. Funktsiooni **float** korral täisarvu sisestamisel viga ei teki ning sellepärast on sageli parem kasutada teisendamiseks funktsiooni **float**.

Mõnikord on otstarbekas sisestamisel kasutada teisendamiseks funktsiooni

eval(**input**(*teade*)).

Funktsioon on stringina ette antud avaldise väärtuse leidmiseks. Sisestamisel teeb see kindlaks sisestava väärtuse tüübi ja teisendab selle vastavasse vormingusse. Võiks proovida näiteks järgmisi tegevusi.

```
>>> x = eval(input("x = "))
x = 13
>>> print (x, type(x))
13 <class 'int'>
>>>
```

Vastuseks teatele sisestatakse täisarv 13 ja see teisendatakse täisarvuks 'int'.

```
>>> x = eval(input("x = "))
x = 13.7
>>> print (x, type(x))
13.7 <class 'float'>
```

Vastuseks teatele sisestatakse reaalarv 13.7 ja see teisendatakse tüüpi 'float'.

```
>>> x = eval(input("x = "))
x = tere
Traceback (most recent call last):
…
  x = eval(input("x = "))
…
NameError: name 'tere' is not defined
>>>
```

Vastuseks teatele sisestatakse tekst tere.

Kuvatakse veateade, nimi tere on määratlemata. Süsteem käsitleb sisendit nimena, sest interpreteerib seda avaldisena.

```
>>> x = eval(input("x = "))
x = 'tere'
>>> print (x, type(x))
tere <class 'str'>
>>>
```

Sisestatav väärtus on paigutatud ülakomade vahele – *'tere'*. Formaalselt on tegemist stringkonstandiga, mille tüübiks on 'str'.

```
>>> x = eval(input("x = "))
x = 3 * 13 – 21          # sisestatakse avaldis,
>>> print (x, type(x))  # mis sisaldab
18 <class 'int'>         # konstante
>>> a = 13
>>> x = eval(input("x = "))
x = a      # sisestatakse muutuja nimi
>>> print (x, type(x))
13 <class 'int'>
>>> x = eval(input("x = "))
x = 3 * a - a / 2 + 100  # sisestatakse avaldis,
>>> print (x, type(x))   # mis sisaldab
132.5 <class 'float'>    # muutujaid ja konstante
```

Funktsioon **eval** võimaldab siin kasutajal sisestada avaldise väärtuse arvutamiseks.

Avaldistes võib kasutada muutujate, objektide ja protseduuride nimesid, mis on kättesaadavad antud protseduuris või skriptis.

Funktsiooni **eval** saab kasutada ka mitme väärtuse korraga sisestamiseks (eraldatakse üksteisest komadega). Lause esitatakse järgmisel kujul: *muutuja* [, *muutuja*]… = **eval**(**input**([*teade*]))

Graafikaandmed ja graafikavahendid Pythonis

Üldised põhimõtted

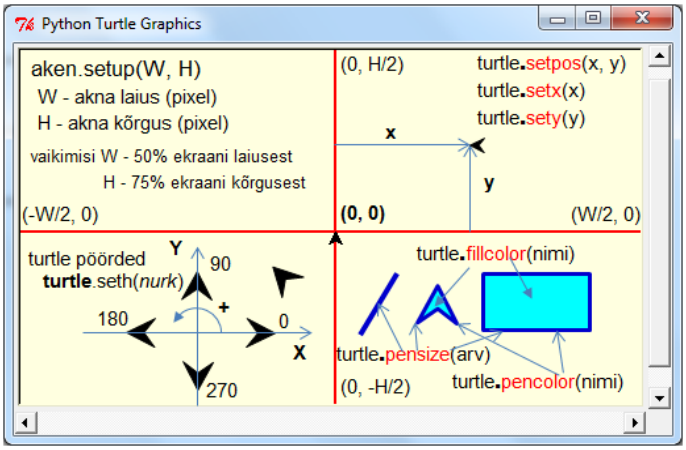
Graafikaandmete (pildid, joonised, skeemid, digrammid, graafikud jms) loomiseks ja töötlemiseks on suur hulk erinevaid tarkvarasid, alates lihtsatest joonistamisprogrammidest ja lõpetades pilditöötlemise, kujundamise joonestamis- ja projekteerimissüsteemidega. Graafikaandmed (graafikaobjektid) leiavad laialdast kasutamist ka rakendusprogrammide dokumentides ning veebidokumentides. Arvutigraafikas eristatakse kahte liiki graafikat: **rastergraafika** ja **vektorgraafika**. Rastergraafikas luuakse graafiline kujutis pikselitest – väikestest värvilistest punktidest. Veidi lihtsustades võib öelda, et vektorgraafikas kasutatakse kujutiste loomiseks sirgjoonte lõike ja neist moodustatavaid kujundeid – kolmnurgad, ristkülikud, ringid, … .

Enamikes programmeerimissüsteemides on olemas vahendid graafikaandmete kasutamiseks, kuid ka loomiseks. Pythoni standardmoodulite teeki kuulub moodul **turtle**, milles sisalduvate vahendite abil saab luua ja töödelda graafikaandmeid nn **kilpkonnagraafika** stiilis. Meetod ja nimetus võeti kasutusele aastaid tagasi programmeerimissüsteemis **Logo**, mis on päevakorral ka praegu, eriti programmeerimise õpetamisel. Kilpkonnagraafika on kasutusel mitmetes programmeerimissüsteemides nagu Scratch, BYOB, Basicu mitmed versioonid jms. Selle elemente on praktiliselt kõikides programmeerimissüsteemides, sest tegemist on arvutigraafika ja eriti vektorgraafika algelementidega. Joonistamisel lähtutakse tõsiasjast, et suurema osa kujunditest (joonised, graafikud, diagrammid, …) saab moodustada sirgjoone lõikudest. Vajaduse korral kasutatakse ka kujundite täitmist värviga.

Moodulis **turtle** sisalduvate vahendite abil saab luua joonistusi, teatud määral manipuleerida olemasolevate graafikaobjektidega (piltidega) ning luua ka animatsioone. Kuid eeskätt on see vahend siiski mõeldud joonistamiseks, võimaldades suhteliselt kiiresti ja lihtsalt omandada joonistamise ja joonestamise programmeerimise alused ja põhimõtted. Graafika programmeerimine aitab kaasa üldiste programmeerimisoskuste süvendamisele. Peab märkima, et ka robotite programmeerimisel leiavad kasutamist mitmed analoogilised põhimõtted ning isegi sarnased käsud.

Joonestamise operatsioone täidab spetsiaalne graafikaobjekt – ***turtle*** (kilpkonn) – Scratchi spraidi analoog. Kilpkonna (objekte) võib olla mitu ja neil võivad olla erinevad kujud. Vaikimisi eksisteerib üks kilpkonn, millel on nooletaoline kuju ➤. Kilpkonna rollis võib kasutada ka teisi graafilisi kujundeid, nt gif-vormingus pilte. Joonistused tekivad spetsiaalses graafikaaknas *Python Turtle Graphics*, mis luuakse automaatselt, kui programmis täidetakse esimene graafikakäsk (*turtle* meetod).

Käsuga **setup** saab määrata akna suuruse, mille mõõtmed antakse pikslites. Pikslil ei ole fikseeritud suurust – see sõltub ekraani suurusest ja eraldusvõimest. Koordinaatsüsteemi nullpunkt asub akna keskel. Oluline roll on kilpkonna suunal ehk pöördel (omadus **heading**). Mitmed liikumise ja pööramise meetodid: **forward**, **backward**, **left** jt arvestavad suuna hetkeväärtust. Alguses viiakse kilpkonn akna nullpunkti ja suunaks võetakse 0° (paremale).



Kilpkonnaga on seotud **pliiats**, mis võib liikumisel jätta jälje. Nähtamatu pliiats liigub koos kilpkonnaga ning tal on kaks võimalikku olekut: all (*pendown*) – jätab liikumisel joone, üleval (*penup*) – joont ei teki. Saab määrata pliiatsi suuruse (*pensize*) ehk joone paksuse ja pliiatsi (joone) värvuse (*pencolor*).

Kilpkonna (sisuliselt pliiatsi) asukoha ja suuna muutmiseks on rida käske (meetodeid), mis toimivad kahel erineval viisil:

- uus asukoht määratakse jooksva suuna ja ette antava distantsiga (d) – käsud forward(*d*), backward(*d*),
- uus asukoht määratakse sihtkoha koordinaatidega (x, y) – setpos(x, y) või goto(x, y), setx(x), sety(y).

Viit **turtle**-objekti meetodile ehk **kilpkonnagraafika käsk** esitatakse kujul: [***objekt*.**] ***meetod*** ([argumendid])

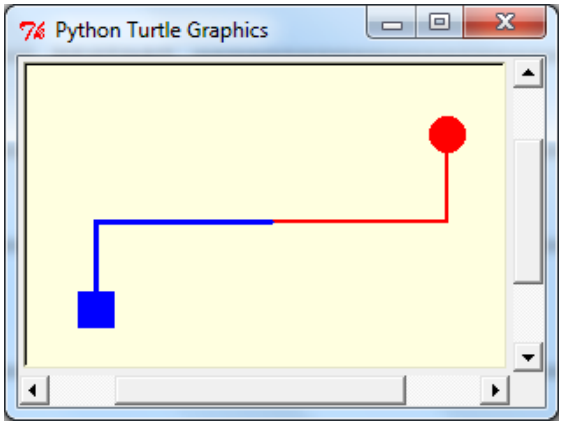
- objekt esitatakse nime turtle või muutuja nime abil, kui eelnevalt on objektiga seotud muutuja muutuja = turtle,
- ***meetod*** määrab tegevuse,
- ***argumendid*** näitavad tegevuse täitmiseks vajalikke andmeid, mis mõnedel meetoditel puuduvad.

Käsu **from** kasutamisel importimise juures ei ole viita objektile vaja. Kuigi viida kasutamise korral on programm kompaktsem, eelistatakse sageli otsest viidet objektile, sidudes selle muutujaga. See variant on üldjuhul paindlikum, võimaldades näiteks paralleelselt kasutada mitut erinevat kilpkonna. Sageli on otstarbekas siduda muutujaga ka graafikaaken. Allpool olevas näites on algseaded programmi jaoks, kus kasutatakse kahte kilpkonna: kati ja mati. Ühtlasi demonstreerib see ka mõnede meetodite kasutamist.

```
import turtle
# akna seaded
aken = turtle.Screen()
aken.setup(300, 200)   # akna mõõtmed pikslites
aken.bgcolor('lightyellow')  # tausta värv
# kilpkonn kati
kati = turtle.Turtle()  # uus turtle objekt kati
kati.shape("circle")   # kati pildiks ring
kati.color("red")
kati.pensize(2)
kati.forward(100)
kati.left(90)
kati.forward(50)
# kilpkonn mati
mati = turtle.Turtle()   # uus turtle objekt mati
mati.shape("square")  # mati pildiks ruut
mati.color("blue")
mati.pensize(3)
mati.forward(-100)
mati.left(-90)
mati.forward(50)
aken.mainloop()  # aken ooteseisu
```

Screen() on meetod, mis loob klassi **Screen** kuuluva objekti. Omistamislausega seotakse loodud objekt muutujaga **aken**.

Samamoodi loob meetod **Turtle**() klassi **Turtle** kuuluva objekti ning omistamislause seob selle muutujaga, mille nimi on näidatud lause vasakus pooles.



Valik mooduli turtle meetodeid

Järgnevalt on esitatud valik **turtle** meetodeid (käske), kuid need ei hõlma sugugi kõiki võimalusi. Täiendavat informatsiooni selle teema kohta võib hankida Pythoni dokumentatsioonist.

Akna seaded	**Selgitused**
setup(W, H)	akna laius (W) ja kõrgus (H). aken.setup(600, 400) – 600*400 pix
bgcolor(värv)	tausta värv : *'black'*, *'red'*, *'white'*, *'green'* jmt
bgpic(nimi.gif)	tausta pilt – gif-fail, peab olema programmiga samas kaustas

setworldcoordinates(xv, ya, xp, yy) – kasutaja koordinaadid: xv – x vasak, ya – y alumine,
xp – x parem, yy – y ülemine

Kipkonna põhiomaduste küsimine ja muutmine

xcor(), ycor(); heading()	x- ja y-koordinaat; jooksev suund
showturtle(); hideturtle()	näita või peida, võib lühemalt: st(); ht()
shape(nimi)	Kilpkonna kujund: *'arrow'*, *'turtle'*, *'circle'*, *'square'*, *'triangle'*, *'classic'*
addshape(nimi.gif)	pildi lisamine gif-failist, mis peab olema samas kaustas programmiga.

Kilpkonna liikumine

forward(d); backward(d)	liigu edasi või tagasi **d** pikslit jooksvas suunas, arvestades **d** märki
setpos(x, y) | goto(x, y)	mine punkti (**x**, **y**), meetodid on samaväärsed
setx(x), sety(y)	määra **x**- või **y**-koordinaat;
home()	algseis: keskpunkti (0, 0), **heading**(90) ➤
right(nurk); left(nurk)	pööra paremale või vasakule **nurk** kraadi + pöörab vastupäeva
setheading(nurk)	võtta suunaks **nurk** kraadi
circle(r, …)	ring raadiusega **r turtle**'st vasakul (arvestades suunda) 

Pliiatsi omaduste määramine

pendown(); penup()	pliiats alla või üles
pensize(w) | size(w)	pliiatsi suurus (joone paksus), meetodid on samaväärsed
pencolor(v), fillcolor(v)	pliiatsi värv, täitevärv. v - värv: *'black'*, *'red'*, *'white'*, *'green'* jmt
pencolor(pv, tv)	pliiatsi (joone) värv (pv) ja täitevärv (tv)
begin_fill(), end_fill()	täitmise algus, täitmise lõpp
clear()	kustutab antud kilpkonna joonistused

Lugemine ja kirjutamine

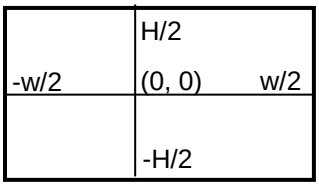
textinput(päis, teade)	teksti lugemine dialoogiboksist
numinput (päis, teade, …)	arvu lugemine dialoogiboksist
write (tekst, …font(…), …)	teksti kirjutamine graafikaaknasse, pliiats eelnevalt vajaliku kohta

Graafikamoodul kujud.py

Näitemoodulis kujud.py on funktsioonid elementaarkujundite (sirgjoone lõik, kolmnurk, ristkülik, …) joonistamiseks. Nende abil saab luua erinevaid pilte ja jooniseid. Lugeja võiks proovida toodud kollektsiooni täiendada.

```
def teljed (a, jv = "black", w = 1):
    ''' X ja Y teljed. a - aken
        jv-joone värv, w - paksus '''
    kk = turtle.Turtle()
    W = a.window_width();  H = a.window_height()
    kk.penup(); kk.setpos(-W/2, 0); kk.pendown();
    kk.pensize(w); kk.pencolor(jv)
    kk.setpos(W/2, 0);   kk.penup()
    kk.setpos(0, - H/2); kk.pendown()
    kk.setpos(0, H/2)
```

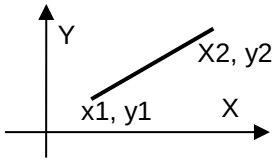
Graafikaaken **a** on määratletud pöörduvas protseduuris. Telgede joonestamiseks on eraldi kilpkonn (kk). Protseduur teeb kindlaks akna mõõtmed: W – laius, H – kõrgus.




```
def tekst (kk, tekst, x, y,
           kv = 'black', ks = 12, kt = 'normal'):
    ''' tekst graafikaaknasse. x, y – koht,
        kv – kirja värv, ks – suurus, kt – tüüp'''
    kk.penup(); kk.setpos (x, y); kk.pencolor(kv)
    kk.write(tekst, font=('Arial', ks, kt))
```

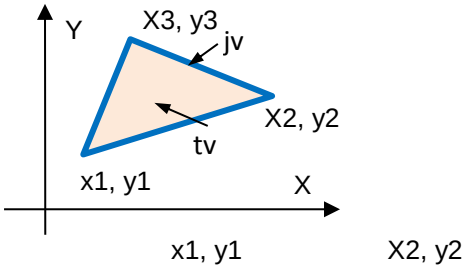
Teksti kirjutamisel graafikaaknasse saab määrata kirja värvuse, suuruse ja tüübi: *normal* | ***bold*** | *italic*.

```
def joon (kk, x1, y1, x2, y2, m=1, jv = 'black', w=1):
    ''' sirgjoone lõik (x1, y1) - (x2, y2), kk-kilpkonn
        m - mastaap,jv - joone värv,  w - paksus '''
    kk.penup(); kk.pencolor(jv); kk.pensize(w)
    kk.setpos(x1 * m, y1 * m); kk.pendown()
    kk.setpos(x2 * m, y2 * m)
```

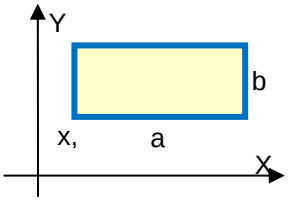


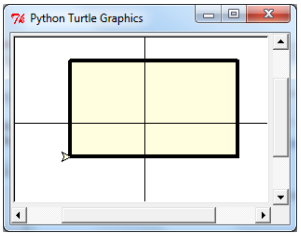

```
def kolmnurk (kk, x1, y1, x2, y2, x3, y3,
              m = 1, jv = 'black', tv ='', w = 1):
    ''' kolmnurk tippudega (x1,y1),(x2,y2),(x3,y3),
        kk-kilpkonn, m-mastaap, jv-joone värv,
        tv – täitevärv,  w- joone paksus '''
    kk.penup(); kk.color(jv, tv); kk.pensize(w)
    kk.begin_fill()
    joon(kk, x1, y1, x2, y2, m ,jv, w)
    joon(kk, x2, y2, x3, y3, m ,jv, w)
    joon(kk, x3, y3, x1, y1, m ,jv, w)
    kk.end_fill()
```

Kolmnurk moodustatakse kolmest sirglõigust, kasutades protseduuri **joon**.



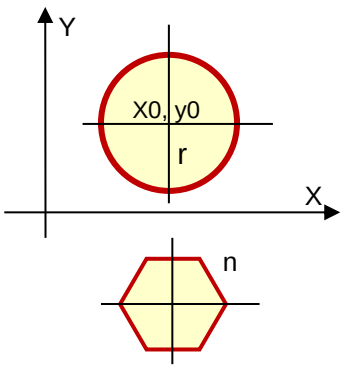
```
def rist (kk, x, y, a, b, m = 1,
        jv = 'black', tv = "", w = 1):
    ''' Ristkülik.  kk - kilpkonn,  m – mastaap,
      x, y – algus; a, b - laius ja kõrgus
      jv - joone värv, tv – täitevärv,  w - paksus '''
    x = m*x; y =  m*y; am = m * a; bm = m * b
    kk.penup(); kk.setpos(x, y)
    kk.width(w); kk.color(jv, tv)
    kk.pendown(); kk.begin_fill()
    kk.setpos(x + am, y)
    kk.setpos(x + am, y + bm)
    kk.setpos(x, y + bm); kk.setpos(x, y)
    kk.end_fill()
```





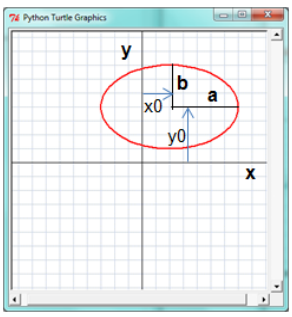

```
def  ring (kk,  x0, y0, r, m=1,
        jv = "black", tv = "", w=1, n=""):
    """ ring või hulknurk.
        kk - joonistaja, m - mastaap,
        x0, y0 - ringi keskpunkt,   r - raadius,
        jv - joone värv, tv – täitevärv, w - paksus
        n - hulknurga külgede arv, kui tühi - ring """
    x0 = m * x0; y0 = m * y0; rm = m * r
    kk.penup(); kk.goto(x0+rm, y0)
    kk.seth (90) ;  kk.pendown(); kk.pensize(w);
    kk.pencolor(jv); kk.fillcolor(tv)
    kk.begin_fill()
    if  n == "":    kk.circle(rm)
    else:           kk.circle(rm, steps = n)
    kk.end_fill()
```

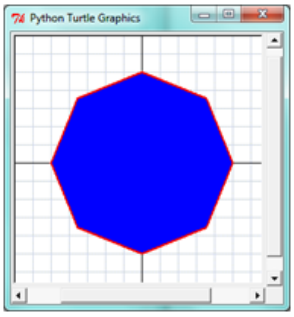
Protseduuri peamiseks elemendiks on mooduli **turtle** meetod **circle**, mis võimaldab joonistada ka korrapäraseid hulknurki, kui parameetrile **steps** anda mitte tühi väärtus.




```
def ellips (kk, x0, y0, a, b,
            jv="black", tv = "", w = 1, n = 64):
    """ elips, ring või hulknurk.
        kk - joonistaja, m - mastaap,
        x0, y0 - keskpunkt,   a, b - raadiused,
        jv - joone värv, tv – täitevärv, w - paksus
        n - hulknurga külgede arv """
    kk.color(jv, tv); h = 360/n
    kk.penup(); kk.setpos(x0+a, y0)
    kk.pendown(); kk.width(w);
    kk.begin_fill()
    for k in range(n):
        fi = k * h * pi / 180
        x = x0 + a * cos(fi)
        y = y0 + b * sin(fi)
        kk.setpos(x, y)
    kk.end_fill()
```

Protseduur võimaldab joonestada ellipsi, ringi või hulknurga. Selle aluseks on ellipsi võrrand parameetrilisel kujul. Joonistamisel joonestatakse järjest n kaart. Kui n on suhteliselt väike (<20) tekib hulknurk. Kujundi kuju saab määrata a ja b väärtusega. Kui a = b on tegemist ringi või võrdkülgse hulknurgaga.





```
def lill(kk, x0, y0, a, b,
         jv = "black", tv = "", w = 1):
  ''' x0, y0 – keskpunkt, a - kroonlehe pikkus
    b – kroonlehtede arv '''
  kk.color(jv, tv)
  kk.penup(); kk.setpos(x0,y0)
  kk.pendown(); kk.width(w)
  kk.begin_fill()
  for fi in range(181):
     fir = fi * pi / 180
     ro = a * sin (fir * b)
     x = x0 + ro *  sin(fir)
     y = y0 + ro * cos(fir)
     kk.setpos(x, y)
  kk.end_fill()
```

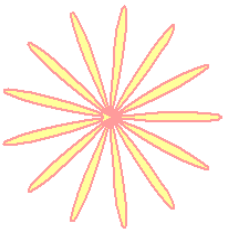
See protseduur joonistab lilletaolise kujundi.

Protseduuris kasutatakse võrrandit polaar-koordinaatides, joone koordinaadid arvutatakse järgmiste võrrandite abil:

$\rho = a*\sin(b*\varphi)$

$x = \rho*\cos(\varphi)$

$y = \rho*\sin(\varphi)$.



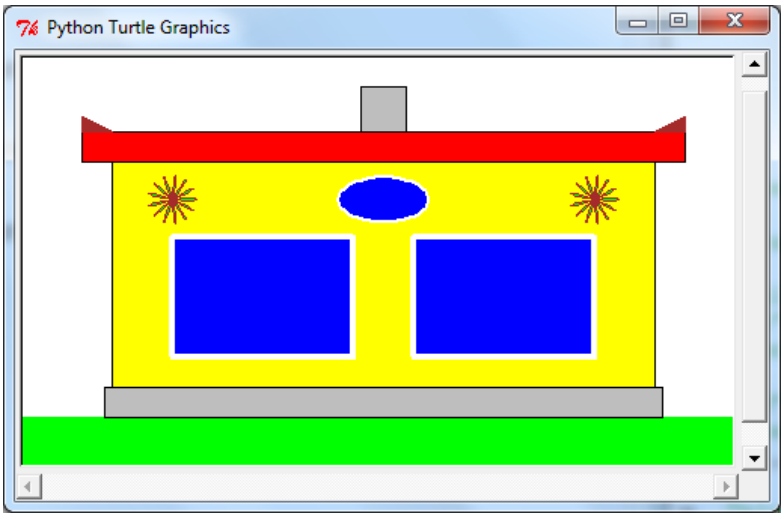
Muutuja ***a*** väärtuseks on kroonlehe pikkus ja ***b*** väärtuseks kroonlehtede arv.

lill (J, 0, 0, 100, 13, jv = "red", tv = "yellow", w=2)

Näide: Ehitame maja

Näide demonstreerib mooduli **kujundid** kasutamist. Moodulis olevatest „detailidest“ pannakse kokku (vist küll mõningate liialdustega) maja pilt.

```
from kujud import *
aken = turtle.Screen()
aken.setup(500, 300)
J = turtle.Turtle()
# maa
rist (J,-240, -140, 480, 40, jv = "green", tv = "green")
# vundment
rist (J,-185, -100, 370, 20, jv = "black", tv = "gray")
# sein ja kaunistused
rist (J,-180, -80, 360, 150, jv = "black", tv = "yellow")
lill (J, 140, 45, 16, 13, jv = "brown", tv = "green")
lill (J, -140, 45, 16, 13, jv = "brown", tv = "green")
# katus ja servad
rist (J, -200, 70, 400, 20, jv = "black", tv = "red")
kolmnurk(J, -200, 90, -180, 90, -200, 100, jv = 'brown', tv = 'brown')
kolmnurk(J, 180, 90, 200, 90, 200, 100, jv = 'brown', tv = 'brown')
# aknad
rist (J, -140, -60, 120, 80, jv = "white", tv =  "blue", w=4)
rist ( J, 20, -60, 120, 80, jv = "white", tv =  "blue", w=4)
ellips (J, 0, 45, 30, 15,jv = "white", tv =  "blue", w=2)
# korsten
rist (J,-15, 90, 30, 30, jv = "black", tv = "gray")
aken.exitonclick()  # lõpp, kui klõpsatakse akent
```



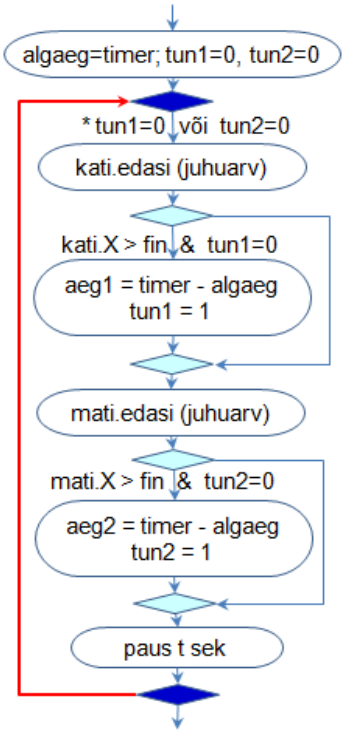
Animatsioonide näited

Näide: Suur võidujooks

Programm demonstreerib lihtsamaid võimalusi animatsiooni realiseerimiseks kilpkonnagraafika vahenditega, aga lisaks ka muud nagu taustapildi lisamine, objekti sidumine muutujaga, objektide kloonimine (paljundamine), mitme **turtle**-objekti, **while**-korduse ja ajaarvestuse kasutamine.

Kaks **turtle**-objekti – **kati** ja **mati** –, pannakse liikuma akna vasakust servast juhuslike sammudega etteantud kauguseni **fin**. Fikseeritakse kohalejõudmise ajad ja nende alusel tehakse kindlaks võitja.

```
import turtle, time
from random import *
aken = turtle.Screen(); W = 600; H = 300
aken.setup(W, H); aken.bgcolor("green")
aken.bgpic("aafrika.gif")  # taustapilt
kati = turtle.Turtle()  # luuakse kilpkonn kati
kati.shape("turtle")  # kati pildiks kilpkonn
kati.fillcolor("red"); kati.penup(); kati.ht()
kati.setpos(-W / 2 + 40, -H/2+70); kati.st()
mati = kati.clone()  # kloonitakse mati!!!
mati.fillcolor("blue"); mati.sety(-H/2 + 35)
time.sleep(1)  # paus 1 sek
# algab suur võidujooks
fin = W/2 - 90 # finishjoone x-koordinaat
algaeg = time.clock()  # aja algväärtus
h = 20; tun1 = tun2 = 0
while tun1 == 0 or tun2 == 0: # kordus
   kati.forward(randint(h/2, h))  # samm
   if kati.xcor() > fin and tun1 == 0 :
      aeg1 = time.clock() -  algaeg; tun1 = 1
      # fikseeritakse kati aeg
   mati.forward(randint(h/2, h))  # samm
   if mati.xcor() > fin and tun2 == 0:
      aeg2 = time.clock() -  algaeg; tun2 = 1
      # fikseeritakse mati aeg
   time.sleep(0.01)  # paus 0.01 sek
# kes võtis?
if aeg1 < aeg2:  v = "kati";  yt = kati.ycor()
else:            v = "mati";  yt = mati.ycor()
# teksti kirjutamine graafikaaknasse
kohtunik = turtle.Turtle(); kohtunik.ht()
kohtunik.penup()
kohtunik.goto(100, yt - 10)
kohtunik.pencolor("red")
kohtunik.write("Võitis " + str(v))
print(aeg1, aeg2)
aken.exitonclick()
```



Kasutatakse kolme **turtle**-objekti, mis on seotud järgmiste muutujatega: **kati**, **mati** ja **kohtunik.**

Lausega: kati = turtle.Turtle(), luuakse uus **turtle**-objekt ja seotakse see muutujaga **kati**.

Käsuga kati.shape("turtle") määratakse, et **kati** graafiliseks kujutiseks on kilpkonnataoline pilt.

Pildi värvuseks võetakse punane ja **kati** viiakse ekraani vasakusse serva.

Käsuga **clone**() kloonitakse **katist mati** – kõik **kati** omadused lähevad **matile**. Edasi muudetakse **mati** värvi ja y-koordinaati.

Programmi keskseks osaks on tingimuslik kordus, millega juhitakse objektide liikumist. Tegevusi korratakse seni, kuni tingimus on tõene. Oluline koht on siin muutujatel **tun1** ja **tun2**, mille väärtuseks võetakse alguses 0. Kui võistleja jõuab lõppu (x>fin), fikseeritakse tema aeg ja vastava tunnuse väärtuseks võetakse 1, millega tagatakse, et võistleja aeg fikseeritakse ainult üks kord, kui vastav tunnus võrdub nulliga. Kordust ja ka mõlemaid **if**-lauseid täidetakse kuni jooksu on lõpetanud mõlemad võistlejad.

Näide: Suur võidusõit

Näide on sarnane eelmisega, kuid imiteeritakse kahe auto võidusõitu.

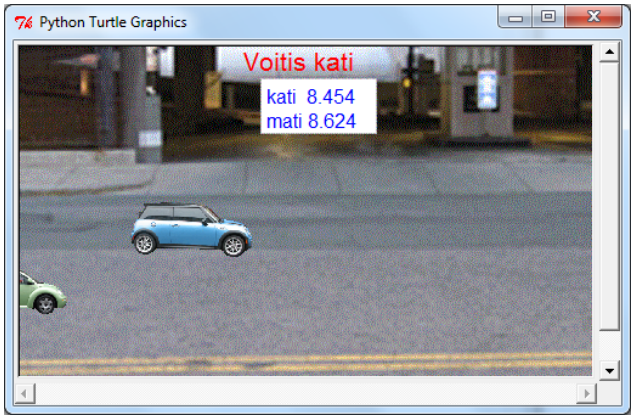
```
import turtle, time
from random import *
aken = turtle.Screen(); W = 1000; H = 600
aken.setup(W, H); dist = W
aken.bgpic("rada.gif")          # tausta pilt
aken.addshape("car_1.gif")  # lisa auto 1
aken.addshape("car_2.gif")  # lisa auto 2
kati = turtle.Turtle()       # luua objekt kati
kati.shape("car_1.gif")   # auto 1 kati pildiks
kati.penup(); kati.hideturtle(); kati.setx(-W/2)
mati= kati.clone()          # kloonida mati
mati.shape("car_2.gif")  # auto 2 mati pildiks
mati.sety(-50)
kati.st(); mati.st()    # peida kati ja mati
kati.speed(0); mati.speed(0)
ringe = aken.numinput("Suursoit", "Mitu ringi?", 3)
dist = 1200; h = 30; r1 = r2 = tun1 = tun2 = 0
algaeg = time.clock()  # aja algväärtus
while tun1 == 0 or tun2 == 0:
  kati.forward(randint(h / 2, h))  # kati samm
  if kati.xcor()  > dist :          # kas ring täis?
    kati.setx(-W / 2); r1 +=1   # algus, lisa ring
    if r1 == ringe and tun1 == 0: # kas kõik?
      aeg1 = time.clock()- algaeg; tun1 = 1
  mati.forward(randint(h / 2, h))  # mati samm
  if mati.xcor() > dist:              # kas ring täis?
    mati.setx(-W / 2); r2 +=1   # algus, lisa ring
    if r2 == ringe and tun2 == 0: # kas kõik?
      aeg2 = time.clock()- algaeg; tun2 = 1
  time.sleep(0.001)   # paus 0.01 sek
if aeg1 < aeg2:     v = "kati"    # kes võitis?
else:                 v = "mati"
# tekst graafikaaknasse
kohtunik=turtle.Turtle(); kohtunik.ht()
kohtunik.pu(); kohtunik.setx(-60);
kohtunik.sety(125)
kohtunik.pd(); kohtunik.pencolor("red")
kohtunik.write(" Voitis " + str(v), font=("Arial", 16))
kohtunik.pencolor("blue")
kohtunik.pu(); kohtunik.goto(-35,100);
kohtunik.write("kati  " + str(round(aeg1,3)))
kohtunik.pu(); kohtunik.goto(-35,80);
kohtunik.write("mati " + str(round(aeg2,3)))
aken.mainloop()
```

Kilpkonna võib esitada suvalise gif-vormingus pildiga. Pildi fail peab olema samas kaustas, kus asub programm.

Käsuga **aken.addshape**(fail) lisatakse objekt aknasse ja käsuga **nimi.shape**(fail) seotakse see konkreetse kilpkonnaga.

Erinevalt eelmisest näitest, sõidavad siin autod etteantud arvu ringe: iga kord, kui auto jõuab kaugusele **dist,** viiakse see akna vasakusse serva.

Allpool on toodud sõidu juhtimise algoritm pseudo-koodis.




```
loe ringe
dist = 1200; h = 30; r1 = r2 = tun1 = tun2 = 0
algaeg = timer
kordus seni kui tun1 = 0 või tun2 = 0
  kati.edasi(juhuarv)
  kui kati.X > dist siis
    kati.X = algus; r1 = r1 +1
    kui r1 = ringe siis
      aeg1 = timer - algaeg; tun1=1
  lõpp kui
  mati.edasi(juhuarv)
  kui mati.X > dist siis
    mati.X = algus; r2 = r2 +1
    kui r2= ringe siis
      aeg2 = timer - algaeg; tun2=1
  lõpp kui
  paus pp
lõpp kordus
```

Juhtimine

Juhtimiseks kasutatakse valikuid ja korduseid.

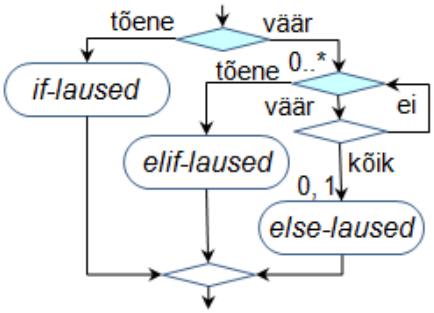
Valikud ja valikulaused

Valikulaused võimaldavad määrata tegevuste (lausete) valikulist täitmist sõltuvalt etteantud tingimustest ja kriteeriumitest. Need kujutavad endast liit- ehk struktuurlauseid, mis võivad sisaldada teisi liht- ja/või liitlauseid.

Lausete struktuur ja täitmise põhimõtted

if-lause üldkuju ja täitmise põhimõte on järgmised:

if *tingimus*:
    if-laused
[**elIf** *tingimus*:
    elif-laused]
…
[**else** :
    else-laused]



Lause koosneb ühest **if**-osast ning võib sisaldada suvalise arvu **elif** (else if) osalauseid ja ühe else-osalause.

Tingimused esitatakse võrdluste või loogikaavaldiste abil. **Laused** võivad olla suvalised liht- ja liitlaused, sh ka **If**-laused. Struktuuri esitamiseks peab kasutama taandeid!

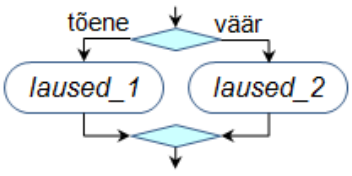
Lause täitmisel kontrollitakse kõigepealt tingimust **if**-lauses, kui see on tõene, täidetakse *if_laused*, kõik ülejäänu jääb vahele. Vastupidisel juhul kontrollitakse järjest tingimusi **elif**-osalausetes (kui neid on olemas) leidnud esimene tõese, täidetakse järgnevad laused ning kõik ülejäänu jääb vahele. Kui ükski tingimus ei ole tõene, täidetakse *else_laused* (kui need on olemas).

Ülaltoodud **if**-lause üldist varianti nimetatakse sageli ka mitmeseks valikuks – mitmest võimalikust tegevuste rühmast valitakse tingimuste alusel välja üks.

Lause üldkujust tulenevad ka kaks sageli kasutatavat varianti, mis võimaldavad määratleda valiku kahest ja valiku ühest.

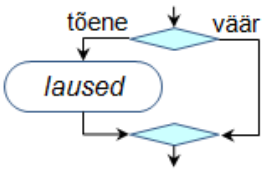
Kahendvalik

if *tingimus* **:**
    laused_1
else :
    laused_2



Valik ühest

if *tingimus* **:**
    laused



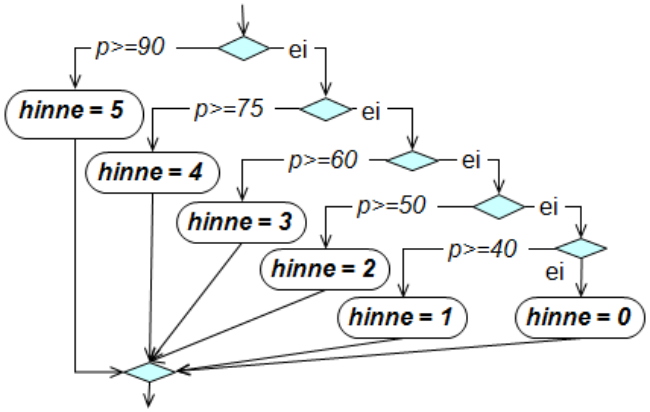
Näide: Punktid ja hinded

Mingi eksami, testi, … hinne määratakse saadud punktide alusel nagu siin esitatud.

Tuleb koostada funktsioon, mis tagastaks vastavalt saadud punktidele hinde.

p = 90–100, hinne = 5
p = 75– 89, hinne = 4
p = 60– 74, hinne = 3
p = 50– 59, hinne = 2
p = 40– 49, hinne = 1
p < 40 hinne = 0

Järgnevalt on toodud selleks kaks varianti: tavaline esitusviis ja kompaktne.

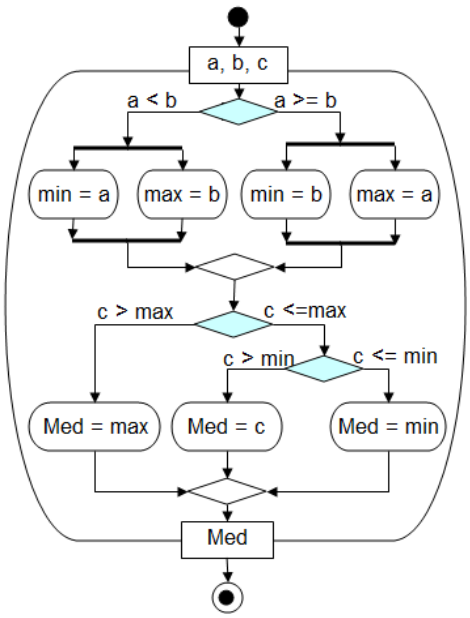



```
def hinne1 (p):
  if p >= 90:
     hinne = 5
  elif  p >= 75:
     hinne = 4
  elif p >= 60:
     hinne = 3
  elif  p >= 50:
     hinne = 2
  elif  p >= 40:
     hinne = 1
  else:
     hinne = 0
  return hinne
```

```
def hinne2 (p):
  if p >= 90:   return 5
  elif p >= 75: return 4
  elif p >= 60: return 3
  elif p >= 50: return 2
  elif p >= 40: return 1
  else:         return 0
```

Näide: Kolme arvu mediaan

Leida kolmest arvu mediaan (suuruse mõttes keskmine väärtus). Allpool on toodud ülesande lahendamise tegevusskeem (algoritm) ja Pythoni funktsioon **med**. Funktsioonil on kolm parameetrit: **a**, **b** ja **c**. Kasutusel on kaks muutujat: **min** ja **max**. Kõigepealt omistatakse muutujale **min** väiksem väärtus esimesest kahest arvust (**a** ja **b**) ja muutujale max suurem väärtus. Edaspidise jaoks ei ole tähtis, kumb arvudest **a** või **b** on suurem või väiksem.




```
def med (a, b, c):
  if a < b :
     min = a; max = b
  else:
     min = b; max = a
  if c > max:
     return max
  elif c > min:
     return c
  else:
     return min
```

```
def med(a, b, c):
  if a < b: min = a; max = b
  else:     min = b; max = a
  if c > max:   return max
  elif c > min: return c
  else:         return min
```

Edasi võrreldakse **min** ja **max** väärtusi kolmanda arvuga. Kui **c** on suurem kui **max**, on tegemist olukorraga, kus **min ≤ max ≤ c** ja mediaaniks on **max**. Kui aga **c ≤ max**, on kaks võimalust: **c > min**, siis **min ≤ c ≤ max** ja mediaaniks on **c**, kui ei ole, siis mediaaniks on **min.**

Kordused

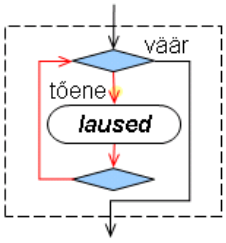
Pythonis on korduste kirjeldamiseks kaks lauset: **while**-lause, **for**-lause. Nendel lausetel on mitu varianti.

Eelkontrolliga while-lause

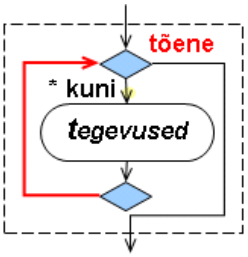
Põhivariant

while *tingimus* :
    laused
    [**break**]
    laused

Kordus töötab, kuni tingimus on tõene.



Scratchi eelkontrolliga kordus töötab seni, kuni tingimus saab tõeseks.



Sisemiste lausete kuuluvus **while**-lausesse määratakse taandega. Kõikide sisemiste lausete taane peab olema võrdne esimese lause omaga, kui need ei kuulu järgmise taseme lause sisse. Lause sees võivad olla **break**-laused, mis võivad katkestada kordamise ning on tavaliselt mingi tingimuslause sees.

NB! Sarnane eelkontrolliga tingimuslik lause töötab Scratchis mõnevõrra teisiti. Pythoni **while**-korduse täitmisel korratakse tegevusi siis, kui tingimus on tõene. Sellise reegliga kordusi (korratakse kuni tingimus on tõene) nimetatakse sageli **while**-tüüpi korduseks. Scratchi **korda kuni** ploki täitmine kestab aga seni kuni tingimus **saab tõeseks**. Sellist kordust (korratakse kuni tingimus saab tõeseks) nimetatakse **until**-tüüpi korduseks.

Näide 1: Start

```
from time import *
def start (k):
  print ("Valmis olla!")
  while k > 0:
    print (k, end = "  ")
    k = k - 1
    sleep(1)
  print ("Start!")
start (10)
```

Antud funktsioon kuvab pöördumise **start** (10) järel järgmised teated.

Valmis olla!
10 9 8 7 6 5 4 3 2 1 Start!

print-lause väljastab teate „Valmis olla!“. Edasi algab kordus.

while-lause igal täitmisel kontrollitakse tingimust **k** > 0. Kui see on tõene, kuvatakse **k** väärtus, lahutatakse **k** väärtusest 1, tehakse paus 1 sek ja kõik kordub kuni **k** saab võrdseks nulliga, st tingimus pole enam täidetud. Peale seda kuvatakse tekst „Start!“.

Näide 2: Arvu arvamine

```
import random
y = random.randint(1, 100)
x = int(input("Arva ära arv 1...100. : "))
k = 1
while x != y:
   k += 1
   if x < y:
      x = int(input('Vähe! Proovi veel: '))
   else:
      x = int(input('Palju! Proovi veel: '))
print ('Korras! Katseid oli: ', k)
```

Väljund võiks olla järgmine:

```
>>>
Arva ära arv 1...100. : 50
Palju! Proovi veel: 30
Palju! Proovi veel: 10
Vähe! Proovi veel: 20
Palju! Proovi veel: 15
Vähe! Proovi veel: 17
Korras! Katseid oli: 5
>>>
```

Muutujale ***y*** omistatakse juhuarv vahemikust 1–100. Kasutaja peab selle arvu ära arvama võimalikult väikese katsete arvuga. Programm toetab veidi kasutajat, teatades, kas pakutud arv ***x*** on väiksem või suurem arvuti poolt mõeldud arvust ***y***. Programm loendab ka katsete arvu, kasutades selleks muutujat ***k***.

Tingimus **while**-lauses on määratud võrdlusega x != y. See tähendab, et kordus kestab seni, kuni kasutaja ei ole sisestanud õiget vastust. Igal kordamisel kontrollitakse kõigepealt tingimust. Kui kasutaja poolt pakutud arv ***x*** ei võrdu arvuti arvuga ***y*** (tingimuse väärtus on **tõene)**, siis täidetakse sisemised käsud. Kui tingimuse väärtus on väär (x = y), katkestatakse kordamised ning täidetaks **while**-lausele järgnev lause, mis väljastab teate ja katsete arvu ***k***. Kuna tingimuse kontroll toimub kohe korduse alguses, peab x esimene väärtus olema sisestatud enne korduse algust.

Näide: Funktsiooni tabuleerimine

Rakendus võimaldab arvutada ja kuvada etteantud funktsiooni väärtused etteantaval lõigul, mis on jagatud **n** võrdseks osaks

```
from math import *
def Fy(x) :   return 3 * sin(2*x)-5 * cos(x/3)
def tabuleerimine (a, b, n) :
   h = (b - a) / n
   x = a
   while x <= b + h/2:
      print (x, round (Fy(x), 3))
      x = x + h
print ("Funktsiooni tabuleerimine")
x0 = float (input ("algus => "))
xn = float (input ("lõpp => "))
n = int (input ("jaotisi => "))
tabuleerimine (x0, xn, n)
```

```
Funktsiooni tabuleerimine
algus => 0
lõpp => 5
jaotisi => 10
0.0–5.0
0.5–2.406
1.0–1.997
1.5–3.965
2.0–6.2
2.5–6.239
3.0–3.54
3.5 0.005
4.0 1.792
4.5 0.883
5.0 -1.153
```

Peaprotseduur loeb algandmed (x0, y0 ja n) ning käivitab funktsiooni **tabuleerimine** (a, b, n), edastades sellele vastavad argumendid. Funktsioon leiab sammu **h** ning võtab **x** algväärtuseks **a**.

Järgnevas **while**-lauses arvutatakse järjest ja kuvatakse funktsiooni väärtus ning muudetakse **x** väärtust sammu **h** võrra. Peale igat muutust kontrollitakse tingimust **x <= b + h/2**. Teoreetiliselt on **x** lõppväärtuseks **b**, kuid **x** muutmisel reaalarvulise sammuga võib tekkida olukord, kus viimane (**b**-le vastav) väärtus võib olla veidi suurem **b**-st ning viimane **y** väärtus jääb leidmata. Selle tõttu on tingimuses võetud lõppväärtus poole sammu võrra suuremaks, tagamaks, et viimane väärtus leitaks alati.

```
print ("Funktsiooni tabuleerimine")
x0 = float (input ("algus => "))
xn = x0
while x0 == xn :
  xn = float(input ("lõpp => "))
  if x0 == xn: print ("x0 ja xn ei tohi olla võrdsed!")
n = 0
while n == 0:
  n = int (input ("jaotisi => "))
  if n == 0: print ("jaotiste arv ei tohi olla null!")

tabuleerimine(x0, xn, n)
```

Antud ülesande lahendamisel on oht sattuda lõputusse kordusse. Kui kasutaja sisestab x0 ja xn jaoks ühesugused väärtused, saadakse sammu h väärtuseks null, mille tõttu **while**-korduse täitmisel x väärtus ei muutu ja tööd ei lõpetata. Öeldakse, et programm jäi „lõputusse tsüklisse“, millest saab väljuda, kasutades klahvikombinatsiooni Ctrl+C.

Vältimaks sellise olukorra tekkimist, tuleks juba algandmete sisestamisel kontrollida selle variandi võimalikkust ning, kui kasutaja sisestab xn jaoks x0-ga võrdse väärtuse, paluda sisestada uus väärtus.

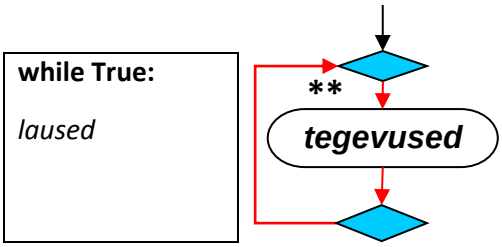
Tegemist on üsna tüüpilise ülesandega algandmete kontrollimiseks sisestamise käigus ja väärtuse võimaliku korduva sisestamisega.

Antud juhul eksisteerib veel teise võimaliku vea oht. Kui **n** väärtuseks sisestatakse 0, tekib sammu arvutamisel jagamine nulliga ja programmi töö katkestatakse. Kuna see ei ole soovitatav, kasutatakse selliste tegevuse täitmiseks enamasti **while**-lauset ja sellist skeemi nagu ülalpool esitatud.

Algselt omistatakse meelega muutujatele **xn** ja **x0** sellised väärtused, et tingimus oleks tõene ja toimuks sisestamine. Kui sisestatatakse muutujatele võrdsed väärtused, väljastatakse teade. Kuna tingimus on siis tõene, pakutakse võimalust uue väärtuse sisestamiseks. Seda tehakse seni, kuni sisestatakse muutujatele erinevad väärtused. Sarnaselt toimub kontroll ja sisestamine ka muutuja **n** jaoks.

Mõnikord on teatud põhjustel vaja meelega tekitada antud kohas lõputu kordusga sarnane tsükkel, mis katkestataks näiteks teise paralleelselt toimiva protsessi toimel. Sel juhul tuleks samuti kasutada **while**-lauset.

Lõputu kordus: while-lause



kordus

tegevused

lõpp kordus

Lõputu korduse saab **while**-lause abil määrata tingimusega, mille väärtus on alati **tõene**. Kõige lihtsam on kirjutada lausesse lihtsalt tõeväärtus **True**. Kasutatakse ka taolisi võrdlusi nagu 1 == 1 või muud sellist. Korduses olevate lausetega määratud tegevusi täidetakse sellisel juhul põhimõtteliselt lõputult. Korduse saab lõpetada vajutusega klahvidele Ctrl+C. **NB!** Selle klahvikombinatsiooni kasutamine võib osutada vajalikuks mõnikord ka siis, kui programm mingi vea tõttu tööd ei lõpeta. Sel juhul öeldaksegi, et programm jäi tsüklisse.

Lõputu kordus leiab kasutust erilise iseloomuga rakendustes, sest reeglina peab programmi töö lõppema loomulikul teel, st programmis määratud tingimuste alusel.

Lõputu kordus katkestusega: break-lause

```
import random
def arva():
  print ("Arva ära arv 1...100.")
  y = random.randint(1, 100)
  k = 0
  while True:
    k = k + 1
    if k > 13:
      print ("Katsete arv on täis!")
      tun = 0; break
    x = input("Paku arv => ")
    if x == "" :
       tun = 0; print("Kahju! "); break
    x = int(x)
    if x == y : tun = 1; break
    if x < y : print ("Vähe!")
    else:  print ("Palju!")
  if tun == 1:
    print("Korras! Katseid oli:", k)

arva()
```

Ette võib tulla olukordi, kus põhjuseid korduse katkestamiseks on mitu ja need asuvad erinevates kohtades. Sellistel juhtudel on kasulik kordamiste aluseks võtta lõputu kordus ning näha ette korduse katkestamine **break-l**ause abil, kui tekib vastav sündmus, mida kontrollitakse valikulauses. Muuhulgas saab selliselt modelleerida ka nn **järelkontrolliga kordust**, mille jaoks on mitmetes keeltes spetsiaalsed laused.

Esitatud näites on toodud arvu arvamise mängu uus variant. Siin on lisatud tingimus, et katsete arv ei tohi olla suurem ette antud arvust (programmis on selleks võetud 13). On arvestatud võimalusega, et mängija katkestab mängu, vajutades arvu sisestamise asemel lihtsal klahvi **Enter**, st tagastatakse tühi väärtus.

Antud juhul on korduse lõpetamiseks kolm võimalikku varianti:

- arv on ära arvatud, st x = = y
- katsete arv on ületatud, st k > 13
- kasutaja loobus, st x = = ''

For-lause

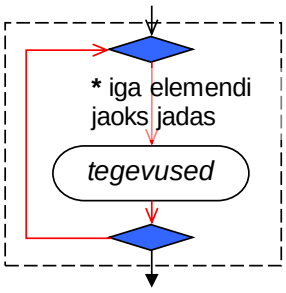
For-lause abil saab kirjeldada erineva täitmise põhimõtetega kordusi. Lause üldkuju on järgmine:

for *element* **in** *jada*

laused

[**if** *tingimus* : **break**]

laused



kordus Iga elemendi jaoks jadas

tegevused_1

kui *tingimus* **välju**

tegevused_2

lõpp kordus

Tegevusi täidetakse teatud jada või kogumi iga elemendi jaoks. Jada või kogumi määratlemiseks võib kasutada erinevaid viise ja vahendeid. Üheks kõige lihtsamaks on funktsioon **range**, mille üldkuju on järgmine:

range ([algus,] lõpp [, samm])

Näiteks määrab range(3, 12, 2] loetelu, mille elementideks on väärtused 3, 5, 7, 9, 11. Kui puudub algus, võetakse see võrdseks nulliga ja kui samm ei ole näidatud, võetakse see võrdseks ühega. Jada viimane liige võetakse ühe võrra väiksemaks kui lõpp. Näiteks, kui n = 5, siis tekitatakse järgmised jadad

range(n) => 0, 1, 2, 3, 4; **range**(1, n) => 1, 2, 3, 4; **range**(n, 8) => 5, 6, 7

Funktsiooni range abil saab kirjeldada etteantud kordamiste arvuga kordusi. Näiteks lause

```
for k in range(7):
    print ('Tere!')
```

väljastab seitse korda teksti „Tere!“.

Loendite (massiivide) töötlemisel on väga levinud **for**-lause kasutamine. Kui määratletud on järgmine loend

X = [3, 5, 8, 10],

siis lause

for elem **in** X: **print** (elem**2, end = ″ ″)

kuvab jada 9 25 64 100.

Näide: Naturaalarvude ruutude summa

```
def nats2(n1, n2):
    S = 0
    for k in range(n1, n2+1):
        S = S + k * k
    return S

print (nats2(5,13))
```

Funktsioon **nats2** leiab naturaalarvude ruutude summa alates algväärtusest **n1** kuni arvuni **n2**. Algus ja lõpp antakse pöördumisel parameetritele vastavate argumentidega.

Põhiosa funktsioonist moodustab **for**-lause. Kõigepealt võetakse muutuja ***S*** algväärtuseks 0, kusjuures **for**-lauses muudetakse juhtmuutuja **k** väärtust alates n1 kuni n2+1 ning liidetakse summale järgmise väärtuse ruut. Tulemuseks tagastatakse **S**.

Näide: Funktsiooni tabuleerimine ja maksimaalne väärtus

```
from math import *

def Fy(x):    return 3 * sin(2 * x + 3)

def Fz(x):
    return 3 *sin(2 * x + 3)-5 * cos(x / 3)

def tabuleerimine (F, a, b, n):
    h = (b - a) / n
    print ('Funktsioon:', F)
    for i in range(n + 1):
        x = a + i * h
        print ("%3d  %5.2f %10.4f" %(i, x, F(x)))

def Fmax(F, a, b, n = 1000):
    h = (b - a) / n
    ymax = F(a)
    for i in range(n + 1):
        x = a + i * h  # x punktis i
        y = F(x)
        if y > ymax: ymax = y
    return ymax

print ("Funktsiooni tabuleerimine")
x0 = float (input ("algus => "))
xn = float(input ("lõpp => "))
n = int(input ("jaotisi => "))
tabuleerimine (Fy, x0, xn, n)
print ("suurim Fy:", Fmax(Fy, x0, xn))
tabuleerimine (Fz, x0, xn, n)
print ("suurim Fz.", Fmax(Fz, x0, xn))
```

See ülesanne oli varem lahendatud ühe funktsiooni **Fy** jaoks **while**-lause baasil (vt lk 60). Siin kasutatakse korduse juhtimiseks **for**-lauset ning töödeldakse veel teist funktsooni **Fz**. Lisatud on ka üks lisategevus – maksimaalse väärtuse leidmine.

Peaprotseduur loeb algandmed **x0**, **xn**, **n** ning käivitab järjest protseduurid **tabuleerimine** ja **Fmax** funktsioonide **Fy** ja **Fz** jaoks.

tabuleerimine arvutab ja kuvab argumendi ja parameetrina antud funktsiooni väärtused määratud lõigul.

Tulemuste kuvamisel kasutatakse vormindamisstringi:
%*w*d – täisarv w positsiooni,
%*w*.*m*f – reaalarv w positsiooni, murdosa pikkus m.

Fmax leiab määratud lõigul funktsiooni maksimaalse väärtuse. Selles protseduris on parameetri **n** vaikeväärtus 1000. Kui vastavat argumenti pöördumisel ei anta (ja siin seda ei tehta), kasutatakse vaikeväärtust. Praegu tähendab see seda, et maksimumi arvutamisel kasutatakse jaotiste arvu 1000, mis tagab piisavalt suure täpsuse.

Programm demonstreerib võimalust funktsiooni (protseduuri) kasutamiseks parameetrina ja argumendina. Protseduuride **tabuleerimine** ja **Fmax** poole pöördutakse kaks korda. Esimesel korral on argumendiks funktsioon **Fy**, teisel **Fz**. Protseduurid **tabuleerimine** ja **Fmax** on selles mõttes universaalsed, et need ei ole seotud konkreetse kasutatava funktsiooniga – see antakse pöördumisel.

Loendid

Loend (*List*) ehk **ühemõõtmeline massiiv** on järjestatud väärtuste kogum. Sellist kogumit tähistatakse ühe nimega, tema elementidele viidatakse nime ja indeksi (järjenumbri) abil. Loendite käsitlemises on üsna palju ühist stringidega, kuid loendi igale elemendile eraldatakse mälus eraldi väli (pesa) ja loendi elemente saab muuta neid lisades, eemaldades, asendades, ...

Loendi olemus ja põhiomadused

Loendi elementide järjenumbrid algavad alati nullist. Viitamine elementidele toimub indeksnime abil järgmiselt:
loendi_nimi [*indeks*], indeks = 0, 1, 2, ...

Indeks paigutatakse nurksulgudesse [ ] ja see võib olla esitatud kui konstant, muutuja või avaldis.

Loendi pikkuse – elementide arvu loendis – saab teha kindlaks funktsiooniga len(loendi_nimi).

T = [*'Kask'*, *'Kuusk'*, *'Mänd'*, *'Paju'*, *'Saar'*, *'Tamm'*, *'Vaher'*]

P = [920, 670, 1120, 990, 1040, 1230, 848]

0, 1, 2, 3, 4, 5, 6

Loendis **T** on töötajate nimed, loendis **P** palgad; len(T) = len(P) = 7

T[0] = Kask T[1] = Kuusk; T[4] = Saar; P[0] = 920; P[4] = 1040; P[6] = 848

Saab viidata ka loendi elementide vahemikele: nimi [m : n]

T[1 : 4] => [*'Kuusk'*, *'Mänd'*, *'Paju'*]; P[4:] => [1040, 1230, 848]

Loendeid saab liita (ühendada): [*'kass'*, *'koer'*] + [*'lõvi'*, *'hunt'*] => [*'kass'*, *'koer'*, *'lõvi'*, *'hunt'*] ja korrutada: 3 * [*'kass'*, *'koer'*] => [*'kass'*, *'koer'*, *'kass'*, *'koer'*, *'kass'*, *'koer'*]

100 * [0] tekitab 100-st nullist koosneva massiivi.

Tühi loend luuakse lausega **nimi** = [ ], mis oleks vajalik näiteks uue loendi loomisel:

```
V = [ ]
for k in range (10):
    V.append(k**2 + 13)
```

Siin luuakse esiteks tühi loend **V** ja seejärel lisatakse sellesse 10 liiget.

Valik loendite meetodeid

append(x)	Lisab x-i (väärtus või objekt) loendi lõppu. T.append(*'Mets'*). Loendi T lõppu lisatakse väärtus 'Mets'
extend(loend)	Lisab loendile teise loendi. P.extend([2100, 1750, 2300]). P lõppu lisandub 3 väärtust
pop()	Tagastab viimase elemendi ja eemaldab selle loendist. maha = T.pop(). maha= *'Paju'*, T-st eemaldatakse
insert(nr, x)	Paneb väärtuse x numbriga määratud elemendi ette. T.insert(2, *'Lepp'*). Nimi *'Lepp'* elemendi nr. 2 ette
index(x)	Tagastab x-i asukoha (indeksi) loendis. k = T.index(*'Kuusk'*). k-le omistatakse väärtus 1
sort()	Sorteerib loendi elemendid kasvavas järjekorras

Näiteid loenditega

Eesti–inglise tõlketest

Loendis nimega **eesti** on sõnad eesti keeles, loendis **inglise** – inglise keeles. Programm palub järjest tõlkida eestikeelsed sõnad inglise keelde, teeb kindlaks õigete vastuste arvu ja protsendi.

```
def test_1 ():
  eesti  =  ["kass", "koer", "hiir", "lehm", "karu"]
  inglise = ["cat",  "dog",  "mouse", "cow", "bear"]
  print ("Tõlgi inglise keelde")
  n = len (eesti)
  oige = 0
  for k in range(n):
    vastus = input (eesti [k] + "=> ")
    if (vastus == inglise [k]):
      oige = oige + 1
    else:
      print ("Vale!")
  return oige / n * 100
print ("protsent:", test_1())
```

Programm leiab funktsiooni **len** abil elementide (sõnade) arvu loendites ning omistab selle muutujale **n**. Seejärel omistatakse muutujale **oige** algväärtus 0. Korduses kuvatakse järjest sõnu loendist **eesti**, kasutades elementidele viitamiseks indeksit (muutujat) **k**. Kasutaja antud vastust võrreldakse sama järjenumbriga sõnaga loendis **inglise**. Kui vastus on õige, suurendatakse muutuja **oige** väärtust, vastupidisel juhul kuvatakse teade „Vale!“. Kui kõik sõnad on läbitud, leitakse õigete vastuste protsent.

Eesti–inglise tõlketest 2

```
def test_2():
  eesti = ["kass", "koer", "hiir", "lehm", "karu"]
  inglise = ["cat",  "dog",  "mouse", "cow", "bear"]
  print ("Tõlgi inglise keelde")
  oige = 0; k = 0
  for sona in eesti:
    vastus = input ( sona + "=")
    if (vastus == inglise [k]): oige += 1
    else: print ("Vale!")
    k += 1
  return oige / len (eesti) * 100
```

Selles näites on eelmise testi ülesanne lahendatud teisiti.

for-lauses kasutatakse jada määramisel loendit **eesti**.

Vastavalt sellise **for**-lause käsitlemise reeglitele täidetakse sisemised laused loendi iga elemendi korral ning viitamiseks elementidele ei kasutata indekseid.

Selliseid korduseid nimetatakse mõnedes programmeerimiskeeltes **for each**-kordusteks ehk korduseks kollektsiooni või hulgaga. Viitamiseks teise loendi (inglise) elementidele kasutatakse indeksit (k).

Näide: Tõlge inglise–eesti

```
def trans_IE() :
   eesti = ["kass", "koer", "hiir", "lehm", "karu"]
   inglise = ["cat",  "dog",  "mouse", "cow", "bear"]
   sona = input ("anna sõna => ")
   nr = -1; k = 0
   for elem in inglise :
      if sona == elem :
         nr = k; break
      k += 1
   if nr == -1:
      print ("Sellist sõna minul ei ole!")
   else :
      print ("Siin see on: ", eesti [nr])
```

Kui kasutaja annab sõna inglise keeles, siis funktsioon peab tõlkima selle eesti keelde.

Kasutades **for**-lauset, tehakse otsimine loendis **inglise**. Muutujale **nr** omistatakse väärtus -1 ning muutujale **k** algväärtus 0. Sisestatud sõna võrreldakse järjest loendi **inglise** jooksva elemendiga. Kui need on samad (võrdsed), omistatakse muutujale **nr** muutuja **k** väärtus ja kordus katkestatakse. Kui sõnad ei võrdu, suurendatakse **k** väärtust ühe võrra. Kui korduse lõppemisel jäi muutuja **nr** väärtuseks -1, tähendab see, et otsitavat sõna loendis pole.

Loendi sisestamine klaviatuurilt

Funktsioon **loe_vek** võimaldab luua loendi, sisestades selle elemendid klaviatuurilt.

```
def loe_vek(V, n):
   for k in range(n):
      elem = input("anna element => ")
      V.append(elem)
```

Funktsiooni parameetriteks on loend (vektor) ja elementide arv selles. **For**-lauses küsitatakse järjest väärtusi ja lisatakse meetodi **append** abil sisestatud väärtused loendi lõppu. Elementide tüüp võib olla suvaline – **string**, **int** või **float**.

```
X = [ ]
n = int(input("mitu => "))
loe_vek(X, n)
for k in range(n):
   print (X[k])
```

Kasutamise näide:
Korraldusega X = [] luuakse tühi loend **X**. Sisestatakse elementide arv **n** ja käivitatakse funktsioon.

Kontrolliks väljastatakse loodud vektor ekraanile.

Loendi loomine juhuarvudest

```
import random
def tee_vek(V, n, mini, maxi):
  for i in range(n):
    elem = random.randint(mini, maxi)
    V.append(elem)
X = [ ]
n = int(input("mitu => "))
tee_vek(X, n, -100, 100)
for k in range(n):
  print (X[k], end = " ")
```

Programmide katsetamiseks ja testimiseks kasutatakse sageli juhuslikke arve, mille hulk võib olla suvaline.

Selles näites antakse funktsioonile **tee_vek** parameetritena ette loend **V**, kuhu salvestatakse genereeritavad juhuarvud, nende hulk **n** ja arvude piirid **mini** ja **maxi**.

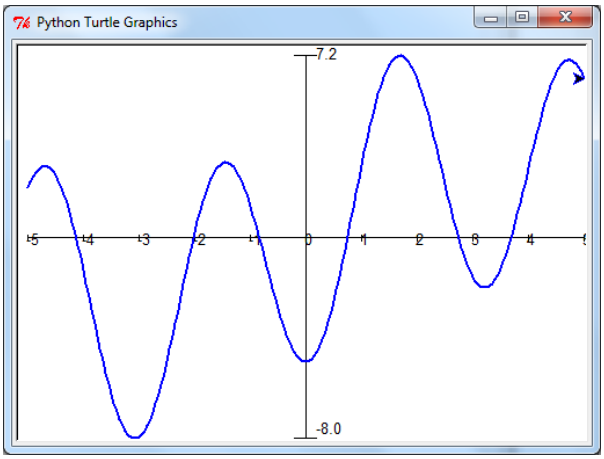
Antud juhul genereeritakse täisarve.

Konkreetne vektor (**X**), elementide arv **n** ja piirid (-100, 100) antakse peaprotseduurist pöördumisel argumentidena.

Funktsiooni uurimine

Rakendus võimaldab teha suvalise ühemuutuja funktsiooni graafiku etteantaval lõigul, arvutada ja kuvada argumendi ja funktsiooni väärtused jaotiste arvuga määratud punktides, leiada samal lõigul mõned funktsiooni karakteristikud: minimaalse ja maksimaalse väärtuse ning nendele vastavad x väärtused, määratud integraali ja pindala ning nullkohad. Argumendi ja funktsiooni väärtused salvestatakse loenditesse (massiividesse), millest viimaseid kasutatakse karakteristikute leidmisel ja graafiku tegemisel.

Allpool on toodud tulemuste näited funktsiooni F(x) = 3 * sin(x / 2) + 5 * cos(2 * x + 3) jaoks lõigul [-5; 5], toodud graafikaakna koopia ning Shelli aknas kuvatavad x, y ja karakteristikute väärtused. Allpool on toodud programmi protseduurid ja kommentaarid nende kohta. Vt ka sama ülesande realisatsiooni lihtmuutujatega mooduli **funktsioon** abil.



x	y	
-5.0	1.974	y min: -7.993 x: -3.05
-4.0	-1.31	y max: 7.22 x: 1.7
-3.0	-7.942	integraal: 2.6906
-2.0	0.177	pindala: 35.7778
-1.0	1.263	nullkohad
0.0	-4.95	-4.13438
1.0	2.857	-2.01871
2.0	6.294	-0.838
3.0	-1.563	0.74594
4.0	2.75	2.74154
5.0	6.333	3.6896

```
from math import *
from turtle import *

def Fy(x):
    return 3 * sin(x / 2) +  5 * cos(2 * x + 3)

def Fz(x,  p = 2):
    if (x <= p):
        return 5 * cos(3 * x) * sin(x/2) + 1
    else:
        return 7 * sin(2 * x - 1) * cos(x  / 3)
```

Kasutaja võib uurida suvalise ühemuutuja funktsiooni käitumist. Selleks peab ta koostama vastava protseduuri funktsiooni väärtuse leidmiseks sellisel kujul nagu on näidatud vasakul ning peale programmi käivitamist sisestama programmi teadete vastuseks vajalikud algandmed, sh ka funktsiooni nime.

Vt allpool protseduure **loe_alg** ja **pealik** (lk 68, 70).

```
def pealik():
    '''peaprotseduur'''
    F, a, b, n, n2 = loe_alg()
    tee_tabel(F, a, b, n)
    X = tee_mas_X(a, b, n2)
    Y = tee_mas_Y(F, X)
    mini, maxi = tee_kar(X, Y)
    AL = 800; AK = 600
    aken(AL, AK, a, b, n, mini, maxi)
    graafik (X, Y, "blue")
    mainloop()
```

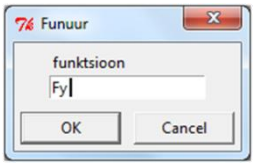
Programm on jagatud järgmisteks protseduurideks:

loe_alg () – algandmete sisestamine
tee_tabel (F, a, b, n) – x ja y väärtused käsuaknasse
tee_mas_X (a, b, n2) – loendi (massiivi) X loomine
tee_mas_Y (F, X) – loendi (massiivi) Y loomine
tee_kar (X, Y) – karakteristikute leidmine ja kirjutamine Shelli
min_nr (V) – min vektoris ja selle järjenumber
max_nr (V) – max vektoris ja selle järjenumber
integraal (X, Y) – määratud integraal antud lõigul
pindala (X, Y) – pindala antud lõigul
aken(AL, AK, a, b, n, mini, maxi) – graafikaakna seadistamine
graafik (X, Y, *"blue"*) – graafiku joonistamine

```
def loe_alg():
    ''' Algandmete lugemine. F – funktsiooni nimi,
    a, b – lõigu otsapunktid, n, n2 – jaotiste arvud '''
    F = eval(textinput("Funuur", "funktsioon"))
    a = numinput("Funuur", "lõigu algus", -5)
    b = numinput("Funuur", "lõigu lõpp", 5)
    n = int(numinput("Funuur", "jaotisi tabelis", 10))
    n2 = int(numinput("Funuur", "jaotisi graafikul", 200))
    return  F, a, b, n, n2
```

Algandmete sisestamiseks kasutatakse graafikamooduli **turtle** sisendbokse.

Lause täitmisel kuvatakse boks, kuhu kasutaja saab sisestada väärtuse.



Esimese lausega sisestatakse uuritava funktsiooni nimi.

```
def tee_tabel(F, a, b, n):
    ''' argumendi ja funktsiooni väärtused Shelli aknasse.
      F – funktsioon, a, b – lõigu otsapunktid, n - jaotisi  '''
    h = (b -a) / n            # tabuleerimise samm
    print ("  x \t", "  y")    # tabeli päis Shelli
    for i in range(n+1):   # jaotise number ( i ) 0…n
        x = a + i * h          # x- i jooksev väärtus
        y = F(x)               # y- i jooksev väärtus
        print (round(x, 2), "\t", round(y, 3))  # x, y Shelli
```

Protseduur arvutab argumendi ja funktsiooni väärtused ning kirjutab need ka käsuaknasse.

Parameetri F väärtus – kasutaja poolt protseduuri **loe_alg** täitmisel sisestatud funktsiooni nimi, tuleb peaprotseduurist. Jaotiste arv **n** valitakse lõigu pikkust arvestades. Näiteks **n** on 10...20 ja lõigu pikkus 10...20 ühikut.

```
def tee_kar(X, Y):
    ''' karakteristikute leidmine ja kirjutamine Shelli
      aknasse. X – argumendi, Y- funktsiooni vekorid'''
    mini, nr = min_nr(Y)
    print ("mini Y", round(mini, 3), " x:", round(X[nr], 2))
    maxi, nr = max_nr(Y)
    print ("maks Y:", round(maxi, 3), " x:", round(X[nr], 2))
    print ("integral:", round(integral(X, Y), 4))
    print ("pindala:", round(pindala(X, Y), 4))
    NV = nullid(X, Y)
    print ("nullkohad")
    for x in NV:
        print (round(x, 5))
    return mini, maxi
```

Protseduur käivitab alamprotseduurid: **min_nr**(Y), **max_nr**(Y), **integral**(X, Y), **pindala**(X, Y) ja **nullid**(X, Y) ja kuvab viimaste poolt tagastatud väärtused käsuaknas.

Karakteristikute leidmiseks kasutatakse protseduurides **tee_mas_X**(x0, xn, n) ja **tee_mas_Y**(F, X) arvutatud ja loendites (vektorites) X ja Y salvestatud väärtusi.

Karakteristikute piisava täpsusega väärtuste saamiseks peab punktide arv (elementide arv vektorites X ja Y) olema piisavalt suur – lõigul pikkusega 10–20 ühikut vähemalt 100...200 või enam.

```
def tee_mas_X(x0, xn, n):
    ''' argumendi väärtuste vektori loomine.
      x0, xn – otsapunktid,   n – jaotiste arv '''
    h = (xn - x0) / n       # x muutmise samm
    X = [ ]                 # tühi loend
    for i in range(n+1):  # jaotise number ( i ) 0…n
        x = x0 + i * h          # x- i jooksev väärtus
        X.append(x)             # lisatakse x loendisse
    return X
```

Protseduur arvutab argumendi (x) väärtused ja salvestab tagastatavas loendis (massiivis) X.

for-lauses muudetaks jaotamispunkti **i** väärtust 0 kuni n.

NB! Funktsioon **range**(n+1) tekitab väärtuste jada: 0, 1, 2, ... n.

```
def tee_mas_Y(F, X):
    ''' argumendi  massiivi Y loomine
      loendis X olevate väärtuste alusel'''
    Y = [ ]
    for x in X:  # x väärtusteks võetakse X-i elemente
        Y.append(F(x))
    return Y
```

Protseduur arvutab ja salvestab tagastatavas loendis **Y** funktsiooni väärtused, kasutades protseduuri, mille nimi tuleb anda parameetri **F** väärtuseks. Argumendi (x) väärtused on varem salvestatud loendis **X**. **for**-korduse täitmisel võetakse muutuja **x** väärtusteks järjest elemente loendist **X** ning leitakse vastav funktsiooni väärtus.

```
def max_nr(V):
    ''' maks vektoris ja
      selle indeks '''
    n = len(V)
    maxi = V[0]; nr = 0
    for i in range (n) :
        if  V[i] > maxi :
            maxi = V[i]; nr = i
    return maxi, nr
```

```
def min_nr(V):
    ''' min vektoris ja
      selle indeks '''
    n = len(V)
    mini = V[0]; nr = 0
    for i in range (n) :
        if  V[i] < mini :
            mini = V[i]; nr = i
    return mini, nr
```

Need protseduurid on peaaegu identsed, peamine erinevus on **if**-lauses olevas võrdlusmärgis.

Mõlemad funktsioonid tagastavad kaks väärtust: **maxi** või **mini** ja **nr**, mis on maxi või mini asukoht (järjenumber) loendis.

maxi ja **mini** väärtusi kasutatakse kasutaja koordinaatsüsteemi määramisel graafikaakna jaoks.

```
def integral(X, Y):
    '''määratud integraal
      trapetsvalemiga'''
    n = len(X)
    h = X[1] - X[0]
    S = (Y[0] + Y[n-1]) / 2
    for i in range(1, n - 1):
        S = S + Y[i]
    return h * S
```

```
def pindala(X, Y):
    ''' pindala joone ja
      X-telje vahel'''
    n = len(X)
    h = X[1] - X[0]
    S=(abs(Y[0])+abs(Y[n-1]))/2
    for i in range(1, n - 1):
        S = S + abs(Y[i])
    return h * S
```

Tegemist on funktsiooniga, milles kasutatakse trapetsvalemit:

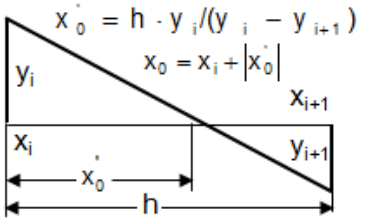
$S = h * (y_0/2 + y_1 + y_2 + \ldots + y_{n-1} + y_n/2)$

Määratud integraal kujutab algebraliselt pindala, arvestatakse funktsiooni väärtuste märki.

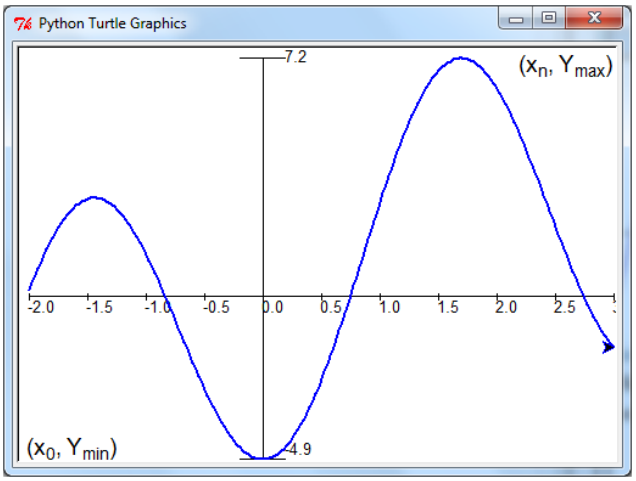
Pindala arvutamiseks kasutatakse absoluutväärtusi.

```
def nullid(X, Y):
  ''' nullkohtade leidmine'''
  NV = [ ]; n = len(X)
  h = X[1] - X [0]  # samm
  for i in range (n - 1):
    if Y[i] == 0: NV.append(X[i])
    if Y[i] * Y[i + 1] < 0:
      x0 = h * Y[i] / (Y[i] - Y[i+ 1])
      x0 = X[i] + abs(x0)
      NV.append (x0)
  return NV
```

Vaadeldaval lõigul (a, b) võib olla suvaline hulk nullkohti. Eeldades, et jaotiste arv on piisavalt suur ehk lõigu alamjaotiste pikkus (h) on väike (näiteks 0,1 … 0,01 ühikut), võib toimida järgmiselt. Vaadata järjest läbi funktsioonide väärtused (loend Y) ning võrrelda naaberpunktide väärtusi. Kui väärtused on erineva märgiga (yi·yi+1<0), siis kahe punkti vahel on nullkoht, mis leitakse esitatud valemitega, arvestades skeemi.




```
def aken(W, H, x0, xn, n, Ymin, Ymax):
  ''' Aken ja teljed. W akna laius, H - kõrgus
     x0 - lõigu algu, xn - lõpu lõpp;
     Ymin, Ymax - funktsioonide min, max '''
  setup(W, H);  speed(0)
  setworldcoordinates(x0, Ymin, xn,  Ymax)
  penup(); setpos (x0, 0);
  pendown(); setpos(xn, 0)
  penup(); setpos (0, Ymin)
  pendown(); setpos(0, Ymax)
  x = x0; hx=(xn-x0)/n  # kriipsude samm
  hk =(Ymax - Ymin)/100  # kriipsu pikkus
  while x <= xn:
    penup();   setpos(x, -hk)
    pendown(); setpos(x,  hk)
    penup(); setpos(x, -3 * hk)
    write (str(round(x)), font=('Arial', 10))
    x = x + hx
  # markerid Y-teljele
  penup(); setpos(-0.2 , Ymax); pendown()
  setpos(0.2 , Ymax); penup()
  setpos(0.2, Ymax-2*hk); pendown()
  write (str(round(Ymax, 1)),font=('Arial', 10))
  penup(); setpos(-0.2 , Ymin); pendown()
  setpos(0.2 , Ymin)
  write (str(round(Ymin, 1)), font=('Arial', 10))
```



Protseduur määrab graafikaakna mõõtmed pikselites, kasutaja koordinaadid, joonistab teljed, teeb jaotised X-teljele ning **Ymax** ja **Ymin** markerid Y-teljele. Väga oluline on turtle meetod setworldcoordinates(x0, Ymin, xn, Ymax), millega määratakse nn maailma ehk kasutaja koordinaadid, nii et graafik mahuks parasjagu aknasse. Süsteem ise valib kasutatavad ühikud ning programmi koostaja ei pea väga palju sellega tegelema.

```
def graafik(X, Y, jv = "red", w = 2 ):
    n = len(X); speed(0)
    penup(); setpos(X[0],Y[0]); pendown()
    pencolor(jv); width(w)
    for i in range(n):
      setpos(X[i], Y[i])

pealik ()
```

Graafiku (joone) joonistamine, kui aken on fikseeritud ning argumendi ja funktsiooni väärtused asuvad loendites, on juba võrdlemisi lihtne. Pliiats viiakse punkti (X[0],Y[0]) ning edasi liigutakse järjest joone ühest punktist teise. Selleks, et graafiku joon oleks sile, peab jaotiste arv olema suhteliselt suur: 100–200 või enam.

Näide: Loto

```
import random
def loto():
  ''' Rakendus imiteerib loto mängu '''
  nmax = int (input("Maks number ? "))
  np = int (input("Numbreid piletil? "))
  nl = int (input("Numbreid loosimisel? "))
  # teeb pileti numbrid
  P = tee_num(np, nmax)
  sort_jada (P)
  print (P)
  # teeb loosimise numbrid
  L = tee_num(nl, nmax)
  L.sort()
  print (L)
  # leiab ühesugused väärtused
  T = yhised(P, L)
  print ("Tabas: ", len(T))
  print(T)
```

Rakendus võimaldab luua kaks komplekti juhuslikke arve etteantavas vahemikus. Arvud salvestatakse vastavates loendites: üks sisaldab pileti numbreid, teine loosimise numbreid. Numbrid ühes loendi piires ei kordu. Selleks, et kasutajal oleks lihtsam võrrelda ja kontrollida pileti ja loosimise numbreid, kuvab programm need sorteerituna kasvavas järjestuses. Programm teeb kindlaks kokkulangevate numbrite arvu ja kuvab need eraldi loendis.

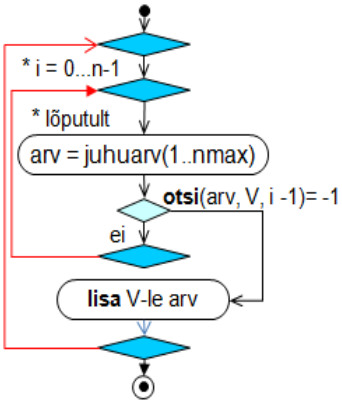
Siin on kasutusel mitu tüüpalgoritmi: erinevate juhuarvude genereerimine, väärtuste otsimine massiivides, massiivi elementide sorteerimine, kokkulangevate väärtuste leidmine massiivides. Toodud on erinevad variandid.

Maks number ? 36
Mitu numbrit piletil? 5
Mitu numbrit loosimisel? 10
[7, 21, 24, 25, 29]
[1, 3, 7, 15, 20, 24, 27, 29, 33, 35]
Tabas: 3
[7, 24, 29]

```
def tee_num(n, nmax):
  ''' Loob vektori V n erinevast
     juhuarvust [1..nmax].
     Kasutab funktsiooni otsi()'''
  V = [ ]
  for i in range(n):
    while True:
      arv = random.randint(1, nmax)
      if otsi(arv, V, i - 1) == -1:
        V.append (arv); break
  return V
```

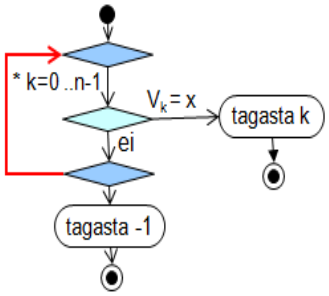
Protseduur loob **n** üksteisest erinevat arvu vahemikus 1…nmax ja salvestab need loendis V.

for-lause muudab elemendi järjenumbrit 0..n-1. Luuakse juhuarv vahemikus 1..nmax ning kontrollitakse funktsiooniga **otsi**, kas selline arv on eespool olemas. Kui on, genereeritakse uus arv, kui ei – salvestatakse **arv** loendis **V** ja genereeritakse järgmine arv




```
def otsi(x, V, n):
  ''' otsib x asukohta V-s 0..n
     kui ei ole, tagastab -1 '''
  for k in range(n + 1):
    if x == V[k]: return k
  return -1
```

Funktsioon otsib väärtusega **x** võrdset väärtust loendist **V**, alates algusest kuni elemendini numbriga **n**. Võimalik, et ka kuni lõpuni. Kui vastav väärtus on olemas, tagastab selle järjenumbri, kui ei, siis tagastab väärtuse -1.



```
def tee_num_1(n, nmax):
    ''' Loob vektori n erinevast
        juhuarvust [1..nmax]. Kasutab
        random-meetodit shuffle() '''
    arvud = list(range(1, nmax + 1))
    random.shuffle(arvud)
    return arvud[: n]
```

Antud protseduuris kasutatakse juhuslike üksteisest erinevate väärtuste genereerimiseks Pythoni vastavaid vahendeid.

Lausega arvud = list(range(1, nmax + 1)) luuakse loend järjestikustest väärtustest 1…nmax.

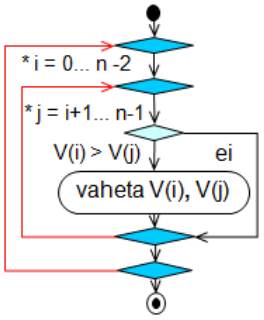
Mooduli **random** meetodiga **shuffle** pannakse need juhuslikku järjekorda. Käsk **arvud[: n]** eraldab loendist **arvud** esimesed **n** arvu.

```
def sort_jada(V):
    '''sordib V jadameetodiga'''
    n = len(V)
    for i in range (n-1):
        for j in range(i+1, n):
            if V[i] > V[j]:
                V[i], V[j] = V[j], V[i]
```

Kõige lihtsam sorteerimise meetod. Võrreldakse elemente Vi (i=0..n-2) järgnevate elementidega Vj (j=i+1..n-1).

Kui Vi > Vj, vahetakse elemendid.

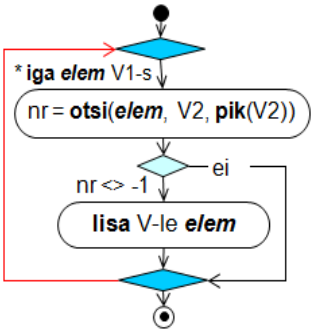
Järgmises jaotises on näiteks toodud veel sorteerimisalgoritme.




```
def yhised(V1, V2):
    '''leiab yhesugused väärtused
       vektorites V1 ja V2 ning
       salvestab need vektoris V '''
    V = [ ]
    for elem in V1:
        nr = otsi(elem, V2, len(V2) - 1)
        if nr != -1: V.append(elem)
    return V

loto()
```

Võrreldakse väärtusi kahes vektoris **V1** ja **V2**. Võetakse järjest elemente vektorist **V1** ja otsitakse võrdset väärtust vektoris **V2**. Kui leitakse kokkulangev väärtus, lisatakse element vektorisse **V**.

Väärtuste otsimisel kasutatakse funktsiooni **otsi**.



Mõned sorteerimisalgoritmid

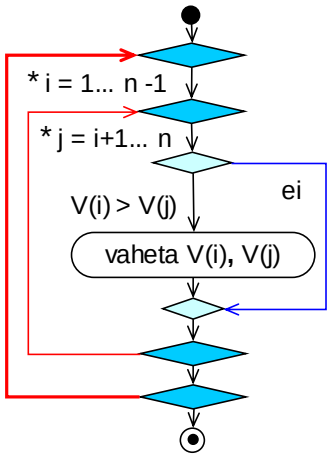
Sorteerimisalgoritmide demodega saab tutvuda järgmiselt lingilt Demod.

Jadasortimine

Jadasortimine on kõige lihtsam sortimismeetod, mille aeg on enam-vähem võrreldav **mullsortimisega** (täpsemalt allpool). Jadasortimises võrreldakse elemente V_i (i = 1..n-1) järgnevate elementidega V_j (j = i+1..n-1). Kui $V_i > V_j$, siis vahetakse need elemendid omavahel. Elementide arv vektoris on **n**.

Sellise meetodi korral tõstetakse väiksemad väärtused järjest ülespoole.

Täitmise aeg on proportsionaalne $\mathbf{n^2}$.



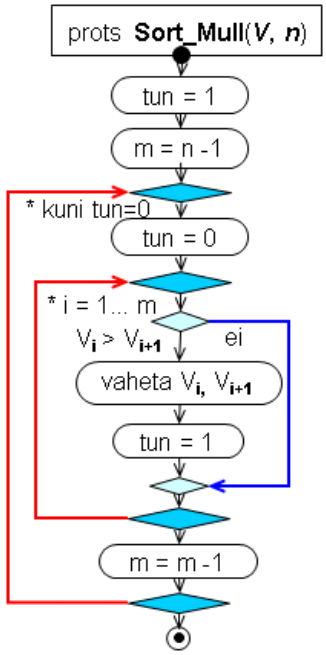

```
protseduur Sort_jada(V, n)
kordus i = 1, n - 1
   kordus j = i + 1, n
      kui V(i) > V(j) siis
         abi = V(i);  V(i) = V(j)
         V(j) = abi
      lõpp kui
   lõpp kordus
lõpp kordus
```

```
def sort_jada(V):
  n = len(V)
  for i in range (n-1):
    for j in range(i+1, n):
     if (V[i] > V[j]):
       V[i], V[j] = V[j], V[i]
```

Mullsortimine

Mullsortimine on üks vanemaid sortimisalgoritme, mille kasutamisel vaadatakse massiiv korduvalt läbi. Iga kord võrreldakse naabreid ning kui V(i) > V(i+1), vahetakse need omavahel ümber. Väiksemad väärtused tõusevad ülespoole, suuremad laskuvad allapoole. Läbivaatamine kestab seni, kuni ei tehta enam ühtegi vahetust.

See meetod on 2–3 korda aeglasem kui valiksortimine ning üldjuhul ka veidi aeglasem kui jadasortimine. Kiire on see meetod siis, kui väärtused on peaaegu järjestatud.



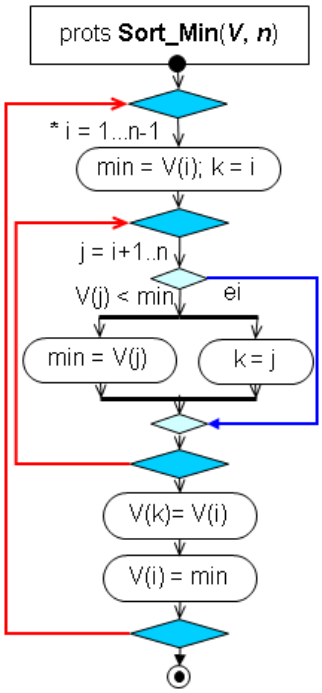

```
protseduur sort_mull(V, n)
 tun = 1
 m = n - 1
 kordus kuni tun = 0
   tun = 0
  kordus i = 1... m
    kui V(i) > V(i + 1) siis
      abi = V(i)
      V(i) = V(i + 1)
      V(i + 1) = abi
      tun = 1
    lõpp kui
    m = m - 1
 lõpp kordus
```

```
def sort_mull(V):
   n = len(V)
   m = n - 1
   tun = 1
   while (tun != 0):
     tun = 0
     for i in range (m):
       j = i + 1
       if (V[i] > V[j]):
         V[i], V[j] = V[j], V[i]
         tun = 1
     m = m – 1
```

Valiksortimine

Selle meetodi korral muudetakse indeksi **i** väärtust vahemikus 1 kuni **n**-1 ning iga **i** väärtuse korral leitakse minimaalne element vektori osas järjenumbritega **i** kuni **n** ja vahetatakse see elemendiga number **i**.

Valiksortimine on 2–3 korda kiirem kui mullsortimine.



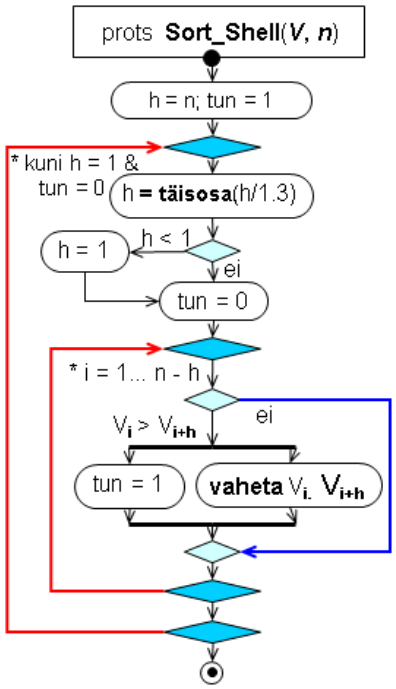

```
protseduur sort_min(V, n)
 kordus i = 1 kuni n - 1
   min = V(i)
   k = i
   kordus j = i + 1 kuni n
     kui V(j) < min siis
       min = V(j)
       k = j
     lõpp kui
   lõpp kordus
   V(k) = V(i)
   V(i) = min
lõpp kordus
```

```
def sort_min(V):
  n = len(V)
  for i in range(n-1):
     min = V[i]
     k = i
     for j in range(i+1, n):
        if  V[j] < min:
           min = V[j]
           k = j
     V[k]= V[i]
     V[i] = min
```

Shelli meetod

See meetod on sarnane mullsortimisele, kuid siin kasutatakse **muutuvat sammu**, mis on alguses suur, edaspidi järjest väheneb. Keerukus sõltub oluliselt sammu valikust, kuid üks paremaid sammu muutmise reegleid on $n_i = n_{i-1} / 1{,}3$.

Shelli meetodi keerukus on umbes $n*\log_2 n$, mis on oluliselt kiirem eelmistest, eriti kui n väärtus on suur.
n = 1000 – umbes 10 korda, n = 100 000 – umbes 500 korda, n = 1 000 000 – umbes 5000 korda!




```
prots Sort_Shell(V, n)
   h = n
   tun = 1
   kordus kuni h=1 & tun=0
     h = täisosa(h/1,3)
     kui h < 1 siis h = 1
     tun = 0
     kordus i = 1... n - h
       kui V(i) > V(i + h) siis
         abi = V(i)
         V(i) = V(i + h)
         V(i + h) = abi
         tun = 1
       lõpp kui
     lõpp kordus
   lõpp kordus
```

```
def sort_shell(V):
   n = len (V)
   h = n
   while True:
     h = int(h // 1.3)
     if h < 1: h = 1
     tun = 0
     for i in range (0, n-h):
       if  V[i] > V[i + h]:
         V[i], V[i + h] = V[i + h], V[i]
         tun = 1
     if (h == 1 and tun == 0 ): break
```

Tabelid ja maatriksid

Nii tavaelus kui ka paljudes programmeerimiskeeltes kasutatakse tabelit kui ülevaatlikku andmekogumit, mis koosneb järjestatud ridadest ja veergudest. Tabeli lahtrites võivad olla arvud, stringid, … väärtused. Lahter võib olla ka tühi. Programmis **Meeskond** kasutatavad andmed meeskonna liikmete kohta võiks esitada tabelina nagu näidatud allpool. Selliselt esitatud andmestruktuure nimetatakse ka kahemõõtmelisteks massiivideks. Viitamiseks massiividele ja tabelitele kasutatakse nime koos kahe indeksiga, milleks on reanumber ja veerunumber: **nimi**[*rida*][*veerg*].

Tabel **T**

nimi	**pikkus**	
Haab	193	0
Kask	181	1
Mänd	178	2
Paju	203	3
Saar	197	4
Tamm	177	5
0	1	

Maatriks **A**

5	7	-9	4	8	0
-3	-8	1	-9	-4	1
6	.3	2	3	4	2
1	-6	3	4	-8	3
0	1	2	3	4	

T[rida][veerg] **A**[rida][veerg]

T[0, 0], T[3, 1], T[5][1] A[0, 0], A[0, 2], A[2,1], A[4, 4]

Pythonis esitatakse tabelid ja maatriksid loendite loenditena, kus loendi elemendid võivad olla ka loendid. Näites **Meeskond** olid andmed meeskonna liikmete kohta esitatud kahe lihtloendiga: **nimed** ja **P** (pikkused).

```
nimed =  [ 'Haab', 'Kask', 'Mänd', 'Paju', 'Saar', 'Tamm' ]
P =  [ 193,  181,  178,  203,  197,  177 ]
```

Tabelina ehk loendite loendina võib sama info esitada järgmiselt:

```
T = [ ['Haab', 193], ['Kask', 181], ['Mänd', 178], ['Paju', 203], ['Saar', 197], ['Tamm', 177 ] ].
```

Tabelis olevad loendid on kõik võrdse pikkusega: elementide arv kõikides tabeli ridades on sama.

Järgnevas näites on toodud mõned protseduurid vaadeldava tabeliga.

```
def fkesk(T, v):
  '''veeru keskmine'''
  m = len(T)
  S = 0
  for i in range(m):
     S = S + T[i][v]
  return S / m
```

```
def fmaks(T, v):
   ''' max veerus v'''
   m = len(T)
   maks = T[0][v]
   nr = 0
   for i in range(1, m):
     if T[i][v] > maks:
        maks = T[i][v]
        nr = i
   return maks, nr
```

```
T = [['Haab', 193], ['Kask', 181], ['Mänd', 178], \
 ['Paju', 203],    ['Saar', 197], ['Tamm', 177 ] ]

print("Meeskonna liikmed")
kuva_tab(T)
kesk = fkesk(T,1)
print ("\nkeskmine pikkus:", round(kesk, 1))
maxi, nr = fmaks(T,1)
print ("pikim:", T[nr])
```

```
def kuva_tab(T):
  ''' tabeli kuvamine '''
  m = len(T); n = len(T[0])
  for i in range(m):
    for j in range(n):
      print (T[i][j], end = ' ')
    print()
```

```
Meeskonna liikmed
Haab 193
Kask 181
Mänd 178
Paju 203
Saar 197
Tamm 177
keskmine pikkus: 188.2
pikim: ['Paju', 203]
```

Funktsioon **fkesk**(T, v) leiab antud numbriga veeru (v) elementide aritmeetilise keskmise, **fmaks**(T, v) – maksimaalse elemendi ja rea numbri, kus see element asub. Mõlemad funktsioonid on mõnevõrra üldisemad kui vaja antud ülesande jaoks, ning vastava karakteristiku leidmine on kirjeldatud mitte konkreetse veeru (1), vaid suvalise veeru jaoks.

Ridade arvu tabelis – alamloendite arvu loendis – teeb kindlaks funktsioon lausega m = len(T).

Protseduuris **kuva_tab**(T), mis kuvab parameetrina **T** esitatud loendi tabelina, kasutatakse ka tabeli veergude arvu. Protseduur on antud ülesande vajadustest universaalsem, set ridade ja veergude arv võib olla suvaline. Paneme tähele elementide paigutamise määramist tabelina **print**-funktsioonides. Funktsioonis **print** (T[i][j], end = ' ') kasutatakse argumenti end = '', mis blokeerib automaatse ülemineku peale funktsiooni täitmist. Tabeli ühe rea elemendid paigutatakse järjest ekraani ühele reale. Iga kord peale sisemise korduse lõppu täidetakse funktsioon **print**(), mis viib uuele reale.

Näide: Operatsioonid maatriksitega

Järgnevalt on toodud mõned näited protseduuridest, mis täidavad erinevaid operatsioone maatriksiga nagu näiteks maatriksi loomine juhuslikest arvudest, maatriksi kuvamine, maatriksi ridade ja veergude leidmine jms.

```
def tee_mat(m, n, a = -10, b = 10):
    ''' maatriksi loomine juhuarvudest (a…b) '''
    A = [ ]   # tühi loend
    for i in range(m):
        rida = [ ]   # tühi loend uue rea jaoks
        for j in range(n):     # arvude loomine
            rida.append (randint(a, b))  # lisamine
        A.append(rida)    # rea lisamine loendisse
    return A    # tagastab loodud maatriksi
```

```
def kuva_mat_R(A):
    ''' maatriksi kuvamine ridade kaupa '''
    m = len(A)  # ridade arv maatriksis
    for i in range(m):
        print (A[i])   # rea i kuvamine
def kuva_vek(V):
    ''' vektori kuvamine vertikaalselt '''
    m = len(V)   # elementide arv vektoris
    for i in range(m):
        print (V[i])    # elemendi i kuvamine
```

```
def ri_sum(A): # maatriksi ridade summad
    V=[ ]   # tühi vektor summade jaoks
    m = len(A); n = len(A[0])  # read ja veerud
    for i in range(m):  # iga rea jaoks 0…m-1
        S = 0  # rea i summa nulli
        for j in range(n): # iga veeru jaoks 0…n-1
            S = S + A[i][j]  # liita element summale
        V.append(S)      # summa vektorisse
    return V   # tagastab loodud maatriksi
```

```
def ve_sum(A): # maatriksi veergude summad
    V=[ ]   # tühi vektor summade jaoks
    m = len(A); n = len(A[0])  # read ja veerud
    for j in range(n):  # iga veeru jaoks 0…n-1
        S = 0   # veeru j  summa nulli
        for i in range(m):  # iga rea jaoks 0…m-1
            S = S + A[i][j]    # liita element summale
        V.append(S)       # summa vektorisse
    return V     # tagastab loodud maatriksi
```

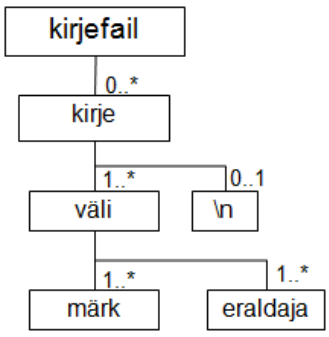
```
# peaprotseduur
M = tee_mat(4, 5)
# maatriks kuvatakse kolmel erineval kujul
print (M); kuva_tab(M); kuva_mat_R(M)
# veergude summade leidmine ja kuvamine
print("veergude summad")
VS = ve_sum(M); print(VS)
# max ja number antud veerus
vrg =int(input("Anna veerg, mille max "))
maxi, nr = fmaks(M, vrg)
print ("maksi veerus", vrg, maxi, "number:", nr)
# ridade summade leidmine ja kuvamine
print("ridade summad")
RS = ri_sum(M); print(RS)
```

```
[ [6, -4, -9, 5, -3], [-4, -1, 6, 2, -5],
 [7, -3, 2, 6, 4], [-1, 5, 10, -6, 9] ]
6    -4    -9    5    -3
-4   -1    6     2    -5
7    -3    2     6    4
-1   5     10    -6   9
[6, -4, -9, 5, -3]
[-4, -1, 6, 2, -5]
[7, -3, 2, 6, 4]
[-1, 5, 10, -6, 9]
veergude summad
[8, -3, 9, 7, 5]
Anna veerg, mille max 3
maksi veerus 3 6 number: 2
ridade summad
[-5, -2, 16, 17]
-5
-2
16
17
```

Failid

Failide tüübid ja struktuur

Faile kasutatakse erinevat liiki informatsiooni säilitamiseks arvutis. Käesolevas jaotises vaadeldakse **tekstifaile**, mida kasutatakse laialdaselt andmete salvestamiseks. Tekstifailidel võib olla erinev organisatsioon, kuid lihtsamal juhul kujutab tekstifail märkide jadast koosnevat jadafaili. Väga sageli kasutatakse ka **kirjefaile**. Selline tekstifail koosneb **kirjetest** ehk **ridadest**. Kirje koosneb ühest või mitmest **väljast**, mis on üksteisest eraldatud ühe või mitme tühiku või tabuleerimissümboliga (\t). Kirje lõpus on (mittenähtav) reavahetuse sümbol (\n), millega peab arvestama failide ja nende kirjete töötlemisel. Sellise struktuuriga faile kasutatakse tabelite salvestamiseks. Samuti saab seda kasutada ka tabelitöötlus-programmide andmebaasides. Tekstifailide töötlemine (lugemine ja kirjutamine) toimub tavaliselt kirjete kaupa.



Failide avamine ja sulgemine

Kaheks põhitegevuseks failidega on andmete **lugemine** ja **kirjutamine**. Enne lugemist või kirjutamist peab faili avama. Pythonis kasutatakse selleks lauset:

failiobjekt = **open**(*failinimi*, *töötlusviis*)

failiobjekt – objektimuutuja, *failinimi* – faili täisnimi: [tee]nimi.txt

Näiteks: palgad.txt või C:/raamatupidamine/palgaarvestus/palgad.txt

Kui fail on programmiga samas kaustas, ei ole teed vaja näidata. Tekstifailide üldlevinud failitüübiks on **.txt**, selle vaatamiseks ja ka koostamiseks on mugav kasutada üldlevinud tekstiredaktoreid, kuid põhimõtteliselt võib kasutada suvalist tüübitunnust.

Töötlusviisi põhivariantideks on:
"**w**" – kirjutamine (*write*) , "**r**" – lugemine (*read* – võetakse ka vaikimisi)

Faili avamisel luuakse nn **failiobjekt** (öeldakse ka failimuutuja), millel on rida meetodeid operatsioonide määramiseks: write(), read(), readline().

fail = **open**(*"puud.txt"*, *"w"*) – fail puud.txt avatakse kirjutamiseks ning failiobjekti nimeks on ***fail***.
fob = **open**(*"puud.txt"*) – fail puud.txt avatakse lugemiseks, failiobjekti nimeks on ***fob***.

Peale töötlemist peab faili sulgema meetodiga close(): failiobjekt.close().

Andmete kirjutamine faili

Andmete tekstifaili kirjutamise põhivariantideks on:

failiobjekt.**write**(*string*)
print([*avaldis*, ...], *sep* = ' ', *end* = *'\n'*, *file = failiobjekt*)

Esimesel juhul on tegemist failiobjekti meetodiga. Teisel juhul **print**-funktsiooniga, mida oleme siiani kasutanud andmete väljastamisel käsuaknasse. Ainukeseks erinevuseks on parameeter **file**, millele pöördumisel vastab failiobjekt. Väljastamisel ekraanile seda ei kasutada või esitatakse kujul file = sys.stdout.

```
fail_1 = open("ankeet.txt", "w")
tee_fail(fail_1,"Juku", 163, 54, 13)
fail_1.close()
```

Esimene lause avab faili nimega **ankeet.txt**. Kui sellise nimega fail on anud kaustas olemas, selle sisu kustutatakse. Allpool on toodud faili loomise protseduuri **tee_fail** kolm varianti.

```
def tee_fail(fm, nimi, L, m, v):
    fm.write("See on tähtis fail")
    fm.write("nimi: " + nimi)
    fm.write("pikkus: " + str(L))
    fm.write("mass: " + str(m))
```

Kuna meetodi **write** argumendiks peab olema üks string, siis peab selle moodustama stringavaldise abil elementidest.

Protseduuri täitmisel tekib ühest reast (kirjest) koosnev fail:
See on tähtis failnimi: Jukupikkus: 163mass: 54

```
def tee_fail(fm, nimi, L, m, v):
    fm.write("See on tähtis fail" + "\n")
    fm.write("nimi: " + nimi + "\n")
    fm.write("pikkus: " + str(L)+ "\n")
    fm.write("mass: " + str(m)+ "\n")
```

Selleks et iga kirje läheks eraldi reale, peab stringi lõppu panema reavahetuse sümboli "\n".

Faili tekib neli eraldi ridadel asuvat kirjet.

```
See on tähtis
fail
nimi: Juku
pikkus: 163
mass: 54
```

```
def tee_fail(fm, nimi, L, m, v):
    print("See on tähtis fail",  file = fm)
    print("nimi: ",  nimi,  file = fm)
    print("pikkus: ", L, file = fm)
    print("mass: ", m,  file = fm)
```

Selles variandis kasutatakse **print**-funktsiooni. Saadud faili struktuur on sama nagu eelmises variandis, kuid siin ei pea koostama stringavaldist, sest funktsioon **print** lubab kasutada mitut argumenti, mis ei pruugi olla stringid.

Ka reavahetus (kirje lõpp) lisandub automaatselt.

Üldiselt on **print**-funktsiooni kasutamine mõnevõrra mugavam ja lihtsam.

Järgnevalt on toodud näide funktsiooni **print** kasutamisest andmete kirjutamisel nii ekraanile kui ka faili. Kasutatakse ühte protseduuri, milles argumendi **file** väärtusega määratakse, kuhu andmed väljastatakse.

```
from math import *
import sys

def Fy(x):
    return 3 * sin(x / 2)+ 5 * cos(2 * x + 3)

def tee_tab(F, a, b, n, fm):
    ''' funktsiooni tabuleerimine ja
        väljastamine ekraanile või faili '''
    if fm == "":  fm = sys.stdout
    h = (b -a) / n
    for i in range(n+1):
        x = a + i * h
        y = F(x)
        print (round(x, 2), "\t",
               round(y, 4), file = fm)
''' funktsiooni tabuleerimise peaprotseduur '''
 x0 = float(input("lõigu algus "))
xn = float(input("lõigu lõpp "))
n = int(input("jaotisi "))
kuhu = int(input("kuhu? 1-ekraan, 2-fail "))
if kuhu == 1:
    fm = ""
else:
    fm = open ("funtab.txt", "w")
tee_tab(Fy, x0, xn, n, fm)
if kuhu == 2: fm.close()
```

Moodul **sys** sisaldab vahendeid juurdepääsuks mõnedele süsteemisisestele funktsioonidele ja objektidele, mida kasutab peamiselt interpretaator (sh standardsisend ja -väljund).

Oluline roll on siin parameetril **fm**. Kui väljund toimub ekraanile, on vastava argumendi väärtuseks tühi tekst. Parameetrile **fm** vastab omakorda argument **file** funktsioonis **print**. Protseduuris kontrollitakse **fm** väärtust. Kui see on pöördumisel tühi, võetakse tema väärtuseks **stdout** (standardväljund) – väljund läheb ekraanile. Väljastamisel faili on vastavaks väärtuseks failiobjekt, kusjuures fail peab olema eelnevalt avatud lugemiseks.

Peaprotseduur loeb algandmete (x0, xn ja n) väärtused ja küsib, kuhu tahetakse suunata väljund – ekraanile või faili. Vastavalt sellele võetakse **fm** jaoks kas tühi väärtus või funktsiooniga **open** määratav failiobjekt.

Kui väljastamine toimus faili, sulgeb selle peaprotseduuri viimane lause.

Sageli toimub andmete kirjutamine faili loenditest või tabelitest. Järgnev näide demonstreerib andmete kirjutamist loenditest **nimed** ja **pikkused**, mis olid kasutusel näites **Meeskond**.

```
def faili(nimed, pikkused):
    n = len(nimed)
    fm = open("meeskond.txt", "w")
    for i in range(n):
        print(nimed[i], pikkused[i], file = fm)
    fm.close()

nimed = [ 'Haab', 'Kask', 'Mänd', 'Paju', 'Saar', 'Tamm' ]
pikkused  =  [ 193, 181, 178, 203, 197, 177 ]
faili (nimed, pikkused)
```

Loendid **nimed** ja **pikkused** on määratletud peaprotseduuris.

Funktsiooniga **open** avatakse meeskond.txt. Seda faili kasutatakse ka järgnevates näidetes andmete lugemisel.

Kirjutamisel kasutatakse funktsiooni **print**.

```
Haab 193
Kask 181
Mänd 178
Paju 203
Saar 197
Tamm 177
```

Andmete lugemine failist

Andmete lugemiseks failist saab kasutada mitmeid failiobjekti meetodeid ja ka muid vahendeid. Järgnevalt vaatleme mõnda varianti neist:

- faili lugemine korraga stringmuutujasse: string = failiobjekt.**read**()
- faili lugemine korraga loendisse: loend = failiobjekt.**readlines**()
- kirjete kaupa korduslauses: **for** kirje **in** failiobjekt: [tegevused kirjega]

Kogu faili lugemist kasutatakse tavaliselt väikeste ja lihtsa struktuuriga failide korral.

```
fob = open("ankeet.txt", "r")
string = fob.read()
fob.close()
print (string, type(string))
```

Loetakse terve fail (näites ankeet.txt) ja omistatakse see ühele stringmuutujale. Stringi kuuluvad ka mittenähtavad erisümbolid *'\n'*, mille tõttu kuvatakse kirjed eraldi ridadel.

```
See on tähtis fail
nimi: Juku
pikkus: 163
mass: 54
 <class 'str'>
```

```
fob = open("ankeet.txt", "r")
loend = fob.readlines()
fob.close()
print (loend, type(loend))
```

Tervest loetud faili sisust moodustatakse loend, mis koosneb faili kirjete elementidest:

```
['See', 'on', 'tähtis', 'fail', 'nimi:', 'Juku', 'pikkus:', '163', 'mass:', '54']
<class 'list'>
```

Vaadeldava faili sisu saab näiteks kuvada. Enamasti toimub failide lugemine ja sellega kaasnev töötlemine kirjete kaupa. Siin võib eristada kolme põhivarianti:

- loetakse järjest kirjeid ja neis sisalduvate andmete põhjal leitakse kohe vajalikud suurused,
- loetakse järjest kirjeid ja neist moodustatakse üks või mitu lihtloendit ehk vektorit,
- loetakse järjest kirjeid ja neist moodustatakse liitloend ehk tabel.

```
def loe_fail1():
  fm=open("meeskond.txt", "r")
  S = 0; n = 0
  for kirje in fm:
     rida = kirje.split()
     S += eval(rida[1])
     n += 1
     print (rida)
  print ("kesk", S / n, 1)
  fm.close()

loe_fail1()
```

Esitatud protseduur loeb failist **meeskond.txt** kirjeid ja leiab nende alusel keskmise pikkuse. **for**-lauses omistatakse muutujale **kirje** järjest faili kirjeid. Meetod **split** moodustab sõnadest loendi **rida**, milles on antud juhul kaks elementi: **nimi** ja **pikkus**. Iga järgnev kirje asendab loendi **rida** eelmised väärtused.

Pikkuse väärtus (loendi teine element järjenumbriga 1) lisatakse summale **S** ja suurendatakse muutuja **n** väärtust. Väärtused loendist **rida** kuvatakse ekraanil.

```
['Haab', '193']
['Kask', '181']
['Mänd', '178']
['Paju', '203']
['Saar', '197']
['Tamm', '177']
kesk 188.2
```

Faili töötlemisel näidatud viisil jääb peale protseduuri täitmist mällu ainult viimane kirje.

```
def loe_fail2():
   ''' moodustatakse kaks loendit '''
   nimed = [ ]       # tühjad
   pikkused = [ ]    # loendid
   fm= open("meeskond.txt", "r")
   for kirje in fm:
     rida = kirje.split()
     nimed.append(rida[0])
     pikkused.append(int(rida[1]))
   print (nimed,pikkused )
   fm.close()
   nimed, pikkused = \
        lisa_kirjed(nimed, pikkused)
   print (nimed, pikkused)
   faili (nimed, pikkused)  # tagasi faili

def lisa_kirjed(nimed, pikkused):
   while True:
     kas = input("kas lisada j-jah, e-ei ")
     if kas == 'e': break
     nimi = input("anna nimi ")
     pikkus = eval(input("pikkus "))
     nimed.append(nimi)
     pikkused.append(pikkus)
   return nimed, pikkused

loe_fail2()
```

Antud protseduur loeb faili kirjete kaupa ja moodustab andmetest kaks loendit (vektorit): **nimed** ja **pikkused**. Peale seda võib kasutaja lisada mõned kirjed ja andmed kirjutatakse faili tagasi.

Igast kirjest moodustatakse kaheelemendiline loend **rida**, mille üks element lisatakse loendisse **nimed** ja teine loendisse **pikkused**. Loendid on sellised:

['Haab', 'Kask', 'Mänd', 'Paju', 'Saar', 'Tamm']
[193, 181, 178, 203, 197, 177]
Pöördumine protseduuri **lisa_kirjed** poole.

Loendid on peale lisamist ja enne kirjutamist tagasi faili järgmised:

['Haab', 'Kask', 'Mänd', 'Paju', 'Saar', 'Tamm', 'Kuusk', 'Mets']
[193, 181, 178, 203, 197, 177, 187, 205]
Protseduur pakub võimalust kirjete lisamiseks faili. Kui kasutaja vastab küsimusele "j" , küsitakse nime ja pikkust.

```
kas tahad lisada j-jah, e-ei j
anna nimi Kuusk
pikkus 187
kas tahad lisada j-jah, e-ei j
anna nimi Mets
pikkus 205
kas tahad lisada j-jah, e-ei e
```

```
def loe_fail3():
   ''' kirjetest moodustatakse tabel '''
   T = [ ]  # tühi loend(tabel)
   fm = open("meeskond.txt", "r")
   for kirje in fm:
     rida = kirje.split()   # kirje loendiks
     T.append(rida)       # tabelisse
   fm.close()
   print (T)
   kuva_tab(T)
```

Igast kirjest luuakse loend **rida** ja see lisatakse loendisse **T**. Tulemusena tekib tabelit esindav loendite loend.

```
[['Haab', 193], ['Kask', 181], ['Mänd', 178],
['Paju', 203], ['Saar', 197], ['Tamm', 177]]
Haab 193
Kask 181
Mänd 178
Paju 203
Saar 197
Tamm 177
keskmine .  188.17
```

```
def kuva_tab(T):
   ''' tabeli kuvamine '''
  m = len(T); n = len(T[0])
  for i in range(m):
    for j in range(n):
      print (T[i][j], end = ' ')
    print()

loe_fail3()
```

Protseduur kuvab loendi tabelina.