TeaMe

sissejuhatus

Rakenduste loomise ja programmeerimise alused

Python

- üldotstarbeline interpreteeritav programmeerimiskeel, algselt arendatud skriptimiskeeleks
- vabavaraline ja avatud lähtekoodiga
- kasutatav mitmetes operatsioonisüsteemides
- leiab laialdast kasutamist, muuhulgas ka veebirakenduste ja dokumendipõhiste rakenduste loomisel
- kasutamise ulatuselt on võrreldav PHP ja Visual Basicuga.

Pythoni oluliseks omaduseks on lihtsus:

- väga lihtne süntaks
- puuduvad igasugused spetsiifilised eraldajad
- lausete struktuur on lihtne, selge ja kompaktne

Python

Loodud **1991**, autor **Guido van Rossum** (Holland), nimi inglise koomikute grupi Monty **Python** järgi

Cpython (realiseeritud pogrammeerimiskeeles C) arendab mittetulundusühing Python Software Foundation (loodud 2001)

- CPythoni jaoks arendatakse kahte haru
  - Python 2
  - Python 3
- Pythoni interpretaatorit ja teeke levitatakse tasuta
- Meie õppematerjalid on orienteeritud **Python 3-**le, vt ka python.org

Muid Pythoni realisatsioone :

- IronPython, realiseeritud C# Microsoftis
- Jython, realiseeritud Javas
- PyPy - realiseeritud Pythonis
- VPython - visuaalne Python

Interaktiivne kasutajaliides ID(L)E

I (integrated / interactive)

D (development / design / debugging)

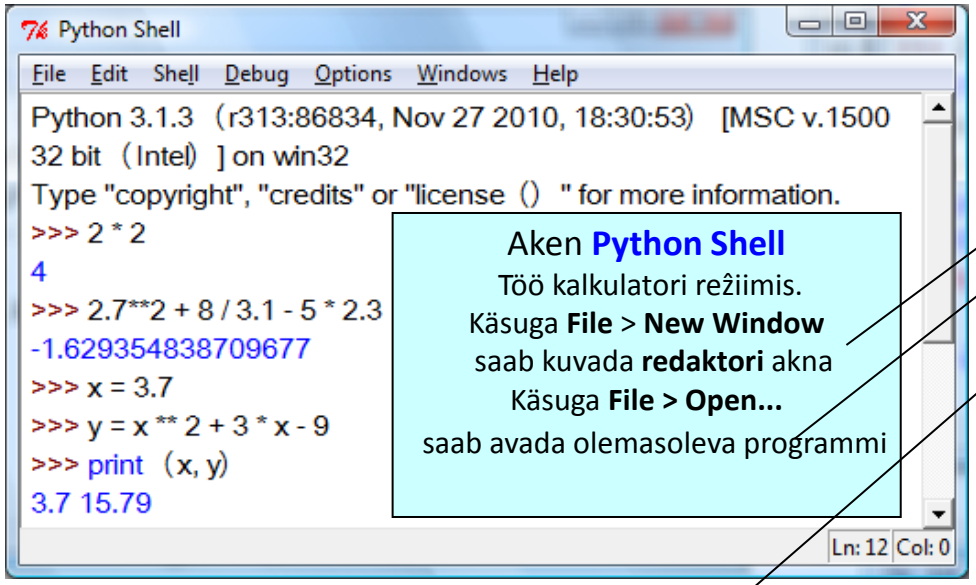
E (environment)

- Interaktiivne kasutajaliides:
  - interpretaator
  - tekstiredaktor
  - silumisvahendid
- Lisaks:
  - juhendid
  - teegid (lisamoodulid)

Interaktiivne kasutajaliides ID(L)E

- *Shelli* aknasse väljastatakse tulemused ja teated
- Akent võib kasutada interaktiivseteks arvutusteks: avaldised, laused, programmi fragmendid
- Programmi sisestamiseks **File-New Window**
- Spetsialiseeritud tekstiredaktor toetab Pythoni programmide sisestamist:
  - taanete tegemine
  - programmi elementide ilmestamine erinevate värvidega

Programme saab koostada ja redigeerida ka suvalise teise tekstiredaktoriga



Aken Python Shell

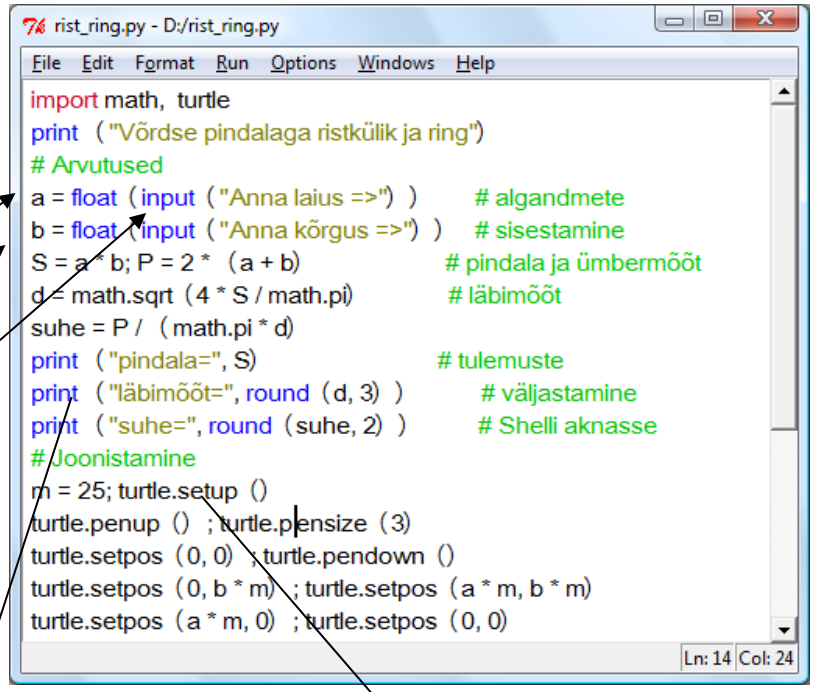
Töö kalkulatori reŽiimis.

Käsuga **File > New Window**

saab kuvada **redaktori** akna

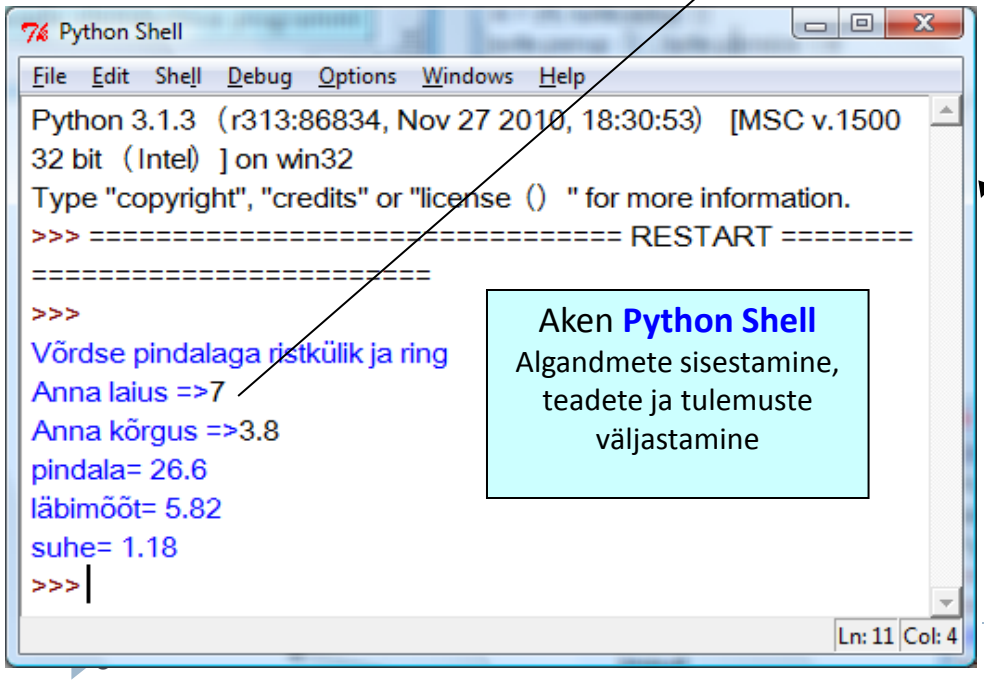
Käsuga **File > Open...**

saab avada olemasoleva programmi



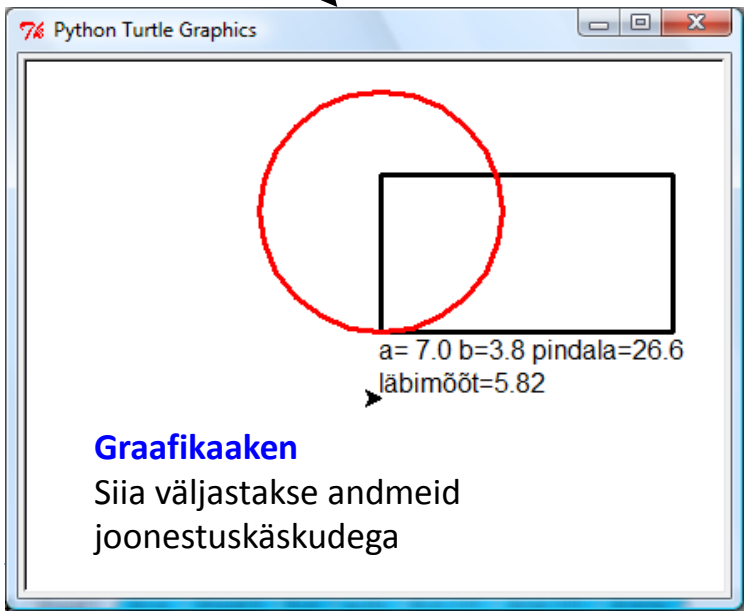
input

print



Aken Python Shell

Algandmete sisestamine, teadete ja tulemuste väljastamine



Graafikaaken

Siia väljastakse andmeid joonestuskäskudega

Programmi (skripti) käivitamine

Programmi saab käivitada otse redaktori aknast käsuga

Run - Run Module (F5)

- Enne täitmist peab programm olema salvestatud (laiend **.py)**
- süntaksivea puhul kuvab interpretaator veateate ja näitab koha
- Korraga näidatakse ainult üks viga
- Vead võivad tekkida ka täitmisel, näiteks sisestatakse ebasobivad andmed vm.
- Kui vigu ei ole, alustab interpretaator programmi täitmist, muutes aktiivseks ümbrise akna (Python Shell), kus kuvatakse programmi teated ja tulemused

Pythoni programmi põhielemendid

- Programmifailid salvestatakse laiendiga **.py**
- Lihtsamal juhul koosneb programm (skript) lausete ja kommentaaride jadast.
- Iga lihtlause on eraldi real
- Kahte lihtlauset samal real eraldab ; (semikoolon)
- Kommentaarid algavad sümboliga # ja võivad asuda eraldi real või rea lõpus.
- Taanded ja koolon on kasutusel liitlausete struktuuri määramiseks

Andmed Pythonis

- Pythoni rakendustes saab kasutada:
  - märkandmed (arvud, tekstid, ajaväärtused ja tõeväärtused)
  - graafikaandmed: pildid, skeemid, joonised jm
  - erinevad heliandmed ja multimeedia vahendid
- Lihtne on kasutada erineva organisatsiooniga andmeid:
  - loendid, massiivid, maatriksid, sõnastikud, failid jm.
- Saab lugeda andmeid andmebaasidest ja kirjutada andmebaasi
- Saab luua ja töödelda erineva struktuuriga dokumente (faile).
- Andmetüüpide ja andmestruktuuride käsitlemine on lihtne ja dünaamiline
- Ei ole vaja väärtuste ja muutujate deklareerimist ning struktuurandmete jäiga ja fikseeritud struktuuri kirjeldamist

Märkandmed

- Pythoni programmides eristatakse **märkandmete** nelja põhitüüpi ehk klassi:
  - stringid – klass **str**
  - täisarvud – klass **int**
  - reaalarvud – klass **float**
  - tõeväärtused – klass **bool**
- Lihtsamal juhul esitatakse need skalaarandmetena.

Konstandid ehk literaalid

Konstandi väärtus ehk literal näidatakse programmis ja ta ei muutu programmi täitmise ajal.

Iga andmeliigi jaoks on ette nähtud kindlad konstantide esitamise reeglid

- **Arvkonstant:** esitatakse kümnendarvudena või kümne astmega. Reaalarvudes on murdosa eraldamiseks punkt

  13 -345 647.234 -35.67 2.1e6 = 2.1×10^{6} 1e-20 = 10^{-20}

- **Stringkonstant:** suvaliste märkide jada, konstandi väärtus paigutatakse piirajate (jutumärgid või ülakomad) vahele, mis ei kuulu konstandi väärtuse hulka.

  "a" "Pindala" "x1=" 'Mis on Sinu nimi?' "Ei" 'pindala='

- **Tõeväärtus: True** (tõene) ja **False** (väär)

Muutujad I

- **Muutujad** esitatakse **nimede** abil.
  NB! Python on tõstutundlik - eristatakse suur- ja väiketahti
- Pythonis ei pea deklareerima muutujaid (tegelikult ei saa)
- Muutuja tüüp määratakse dünaamiliselt väärtuse omistamisel, nii võivad muutujas erinevatel ajahetkedel olla salvestatud erinevat tüüpi väärtused
- Enne muutuja kasutamist avaldises peab sellele olema omistatud väärtus
- Pythonis on võimalik täiendatud omistamine (+=, -=, *=, /=, %=, **= jm) ja mitmene omistamine (s = n = k = 0)
- Mälu eraldamine muutujatele toimub programmi täitmise ajal: interpretaator teeb programmi teksti analüüsides kindlaks kasutatavad muutujad ja eraldab neile kohad mälus

Muutujad II

- Muutujad võib jagada kolme rühma:
  - lihtmuutujad ehk tavamuutujad
  - objektmuutujad ehk viitmuutujad
  - struktuurmuutujad
- **Lihtmuutuja** on nimega varustatud koht arvuti mälus – mäluväli ehk -pesa, kuhu programm saab täitmisel ajal salvestada väärtuse (arvu, teksti jm)
  - igal ajahetkel saab muutujal olla ainult üks väärtus
  - kuni muutujale pole omistatud väärtust, on see määramatu
- **Objektimuutujale** eraldatakse tööpiirkonnas mäluväli ehk pesa.
  - salvestatakse **viit** objektile: objekti omaduste vektori aadress mälus
  - muutuja nime abil saab viidata objektile, selle omadustele ja meetoditele
- **Struktuurmuutujale** (loend jm) vastab mitu omavahel seotud ja teatud organisatsiooni omavat mälupesa (elementi)

Võtmesõnad

- Pythonis on teatud hulk nn **võtmesõnu** reserveeritud ja neid muuks otstarbeks kasutada ei saa:

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

Avaldised. Struktuur ja liigid

Avaldis määrab, mis tehteid (operatsioone) ja millises järjekorras on vaja väärtuse leidmiseks täita.

- Üldjuhul koosneb avaldis:
  - operandidest
  - operaatoritest ehk tehtesümbolitest
  - (ümar)sulgudest
- Erijuhul võib avaldis koosneda ainult ühest operandist.
  Operandideks võivad olla:
  - konstandid
  - lihtmuutujad
  - struktuurmuutujate elemendid
  - funktsiooniviidad ja meetodid
- Lihtmuutujad esitatakse nimede abil, struktuurmuutujate elementide esitus sõltub andmekogumi liigist.

Avaldised

- Sõltuvalt andmete liigist ning kasutatavatest tehetest ja leitava väärtuse liigist võib avaldised jagada järgmistesse rühmadesse:
  - arvavaldised
  - stringavaldised
  - loogikaavaldised

Tehted ehk operaatorid

- **Tehted** ehk **operaatorid** jagunevad nelja rühma:
  - aritmeetikatehted ** , * , / , // , % , + , -
  - stringitehted +, *
  - võrdlustehted == , != , < , <= , > , >=
  - loogikatehted **not**, **and**, **or**
- Avaldise väärtuse leidmisel arvestatakse tehete prioriteete liikide vahel ning aritmeetika- ja loogikatehete puhul liigi sees
- Aritmeetika- ja loogikatehted on siin toodud prioriteetide kahanemise järjekorras
- Väärtuse leidmisel täidetakse kõigepealt aritmeetika-, siis võrdlus- ning lõpuks loogikatehted
- Tehete järjekorra muutmiseks kasutatakse ümarsulge, sulgudes asuva avaldise väärtus leitakse kõigepealt
- Ümarsulgudes esitatakse ka funktsioonide ja meetodite argumendid

Stringavaldised

- Stringavaldiste operandide väärtuseks on stringid, neis võib kasutada stringitehteid ja stringifunktsioone.
- Stringitehet **+** nimetataks **sidurdamiseks**, võimaldab ühendada stringe
  NB! arvud teisendatakse stringideks funktsiooniga str()
  Näide: ‘Tulemus on ‘ + str(arv)
- * võimaldab stringi korrata, teine operand peab olema täisarv. 3 * ‘ai‘ → ‘aiaiai ‘
- stringist eraldatakse sümboleid indeksi abil (indeks algab alati 0-st): tekst[0], tekst[0:3], tekst[5:]
- Standardfunktsioon len() annab stringi pikkuse

Loogikaavaldised

- **Võrdlustehted**: ***avaldis1*** **tehtesümbol** ***avaldis2***
- Tehtesümbolid: **==** , **!=** , **<** , **<=** , **>** , **>=**
- Ühes võrdluses esinevate operandide andmetüübid peavad kokku sobima
- Võrdluse tulemiks on tõeväärtus **True** (tõene) või **False** (väär)

- Loogikatehted on **or**, **and** ja **not**.
- **or** - ***või***; tulemus on tõene, kui vähemalt ühe operandi väärtus on tõene
- **and** – ***ja***; tulemus on tõene ainult siis, kui mõlema operandi väärtused on tõesed
- **not** – ***mitte***, loogiline eitus. Tehte **not** a tulem on tõene, kui a väärtus on väär (ja vastupidi).

Omistamine ja omistamislause

- Omistamine seisneb väärtuse salvestamises arvuti sisemälu etteantud väljas, mille eelmine väärtus (kui oli) kaob
- Tüüpiliselt eelneb väärtuse salvestamisele selle leidmine (tuletamine) etteantud avaldise abil
- Üldjuht: leitakse lause paremas pooles oleva avaldise väärtus ja tulemus omistatakse vasakus pooles olevale muutujale: ***muutuja* = *avaldis***
- Avaldise väärtuse leidmisele kaasneb kasutatavate muutujate varem salvestatud väärtuste lugemine (***muutuja*** esitatakse nime abil)
- Avaldiste operandideks võivad olla konstandid, muutujad, massiivi elemendid, funktsiooniviidad
- Pythonis saab omistada väärtuse korraga mitmele muutujale

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

- a, b = b, a vahetab muutujate a ja b väärtused.

Funktsioonid

- Pöördumine funktsiooni poole toimub **funktsiooniviida** abil kujul:

  nimi (argument {, argument, ... })

- Sisefunktsioonide nimed ei kuulu reserveeritud võtmesõnade hulka
- Funktsiooniviites esinev argument näitab funktsioonile edastatavat väärtust
- Argumentide arv, tüüp ja esitusjärjekord sõltuvad konkreetsest funktsioonist
- Argumendid võivad olla esitatud avaldiste abil

Moodulid

- Mõned funktsioonid, nn. sisefunktsioonid, on Pythonis alati kasutatavad
- Suurem osa funktsioone paikneb moodulites (eraldi failides), mis tuleb põhimooduliga siduda.
- **import mooduli_nimi** *import math, turtle*
  moodulis olevaid funktsioone ja konstante saab kasutada kujul **mooduli_nimi.funktsioon()**

  *t = math.sin(x*math.pi); turtle.forward(10)*
- **from mooduli_nimi import *** *from math import **
  võimaldab kasutada kõiki mooduli funktsioone nagu sisefunktsioone (ilma mooduli nime lisamata)

  *t = sin(x*pi)*
- **from mooduli_nimi import fn_nimi**
  võimaldab moodulist kasutada ainult funktsiooni fn_nimi

Andmete väljastamine ekraanile

- Andmete väljastamiseks *shelli* aknasse saab kasutada funktsiooni **print**:

  print ([*argument* [, *argument*]…])

- Argumentideks võib olla mitu väärtust (konstandid, muutujad ja avaldised läbisegi)

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

- Kuvamisel lisab print väärtuste vahele vaikimisi ühe tühiku ja lõppu reavahetuse (‘\n)
- **print**() väljastab tühja rea
- end = ““ argumentide loetelu lõpus blokeerib ülemineku uuele reale; sep = ... määrab uue eraldaja

Andmete sisestamine klaviatuurilt

- Andmete sisestamiseks vt funktsioon **input()**

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

- Teade ei ole kohustuslik; see võib olla esitatud tekstavaldisena, arvud teisendatud stringvormingusse:

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

- Funktsiooni **input()** väljund on tekstivormingus (string), Kasutatakse teisendusfunktsioone **int** ja **float**, näiteks:

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

- Korraga saab sisestada ka mitme muutuja väärtused eraldades need komadega. Väärtused eraldatakse funktsiooni eval abil.
  Lause võib olla esitatud kujul:

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

Lihtlausete näiteid

Lihtlause ei sisalda teisi lauseid. Peamised lihtlaused on:

- **omistamislause** muutuja [, muutuja] ... = avaldis [, avaldis]...

  y = x**3 + 2 * x - 7.2; k = k +1 ; k += 1

  S = k = t = 0 ; v, w = 2 * x, 3+x; x1, x2 = ruutvrd(a, b, c)

- pöördumine funktsiooni poole (tagastatav(ad) väärtus(ed)

  nimi = **input** ("Mis on sinu nimi?"); L = **float** (**input** (“sisesta arv"));

  a, b, c = **eval** (**input** ("sisesta kordajad"));

  print("Summa = ", S) **;** **joonista**()

- **return**

  return 2 * x

- **break**
- **continue**

Liit- ehk struktuurlaused

- **Liitlause** sisaldab ühe või mitu liht- või liitlauset.
  Esitatakse kujul

  lause **päis :**

  lause

  lause

  …

- Päiselause algab võtmesõnaga ja lõpeb kooloniga (:)

- Sisemised laused moodustavad ploki,
  Ploki esimese taseme lausetel peab olema ühesugune taane.
  Sisemised laused võivad olla omakorda liitlaused

Liitlauseteks on näiteks valikulaused ja korduslaused.

Valikulause

Valikulaused (If-laused) võimaldavad tegevuste täitmist sõltuvalt etteantud tingimustest

Tingimused esitatakse võrdluste või loogikaavaldiste abil

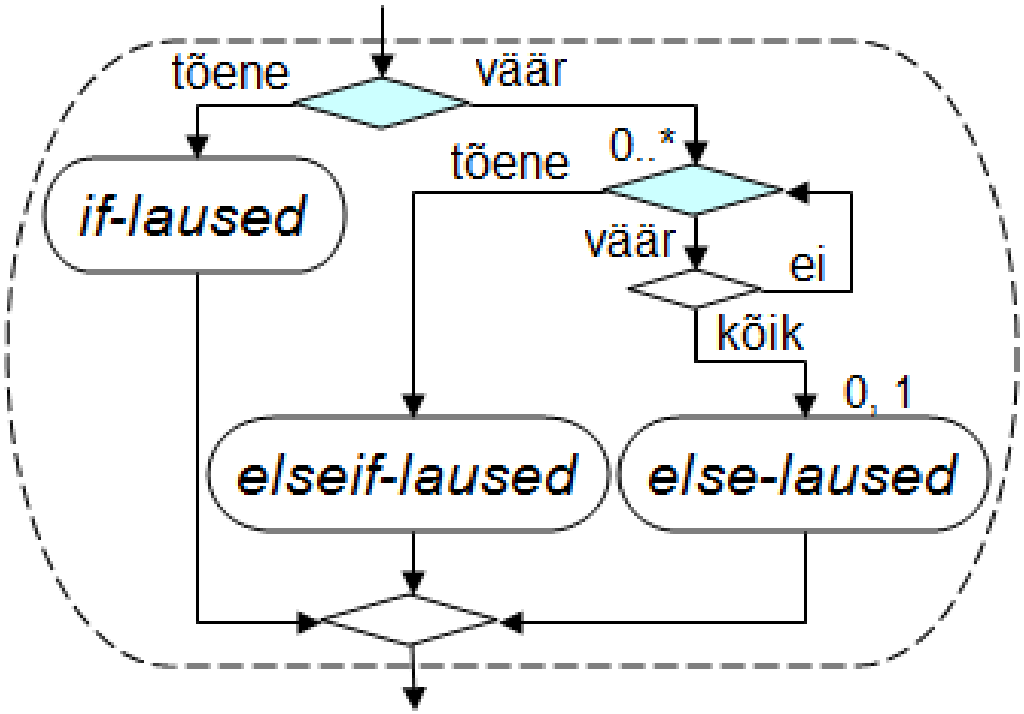
- Lause täitmisel kontrollitakse kõigepealt tingimust **if**-osalauses, kui see on tõene, täidetakse *if-ploki laused*, kõik ülejäänu kuni valikulause lõpuni jääb vahele
- Vastupidisel juhul kontrollitakse järjest tingimusi **elif**-lausetes (kui neid on) ning kui leitakse esimene tõene, täidetakse selle ploki laused ning jätkatakse if-lausele järgnevast lausest
- Kui ükski tingimus pole tõene, täidetakse *else-ploki laused* (kui need on)

Struktuuri tagamiseks peab kasutama taandeid!

Valikulause

if-lause üldine variant: mitmest võimalikust tegevuste rühmast valitakse tingimuste alusel välja üks.

```
if (tingimus):
    if_laused
[ elif (tingimus):
    elif_laused ] …
[ else :
    else_laused ]
```



Kordused ja korduslaused

Pythonis on korduste kirjeldamiseks kaks lauset:

- **while**-lause
- **for**-lause

Lausetel on mitu varianti

Korduslause sees võib olla käsk **break** korduse katkestamiseks (tavaliselt mingi valikulause sees)

break katkestab lähima (viimase) korduse

Skripti töö saab lõpetada klahvikombinatsiooniga Ctrl+c

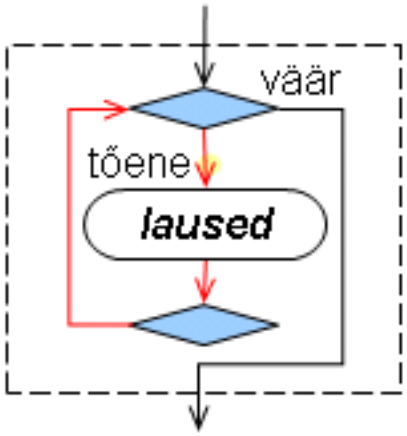
while-lause

while ***tingimus*** **:**

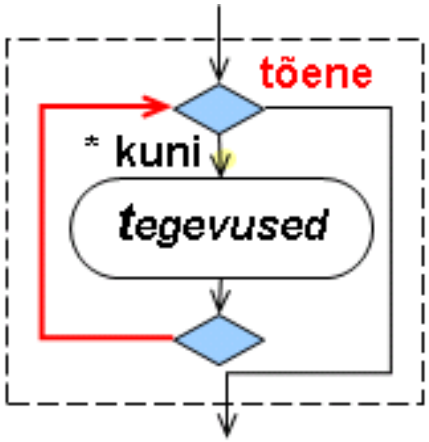
laused

[**break**]

laused



Scratchi eelkontrolliga kordus töötab seni, kuni tingimus saab tõeseks



- while-lause täitmisel korratakse tegevusi, kui tingimus on tõene.
- Järgnevate lausete kuuluvus while-lausesse määratakse taandega, mis peab olema võrdne esimese lause omaga (kui nad ei kuulu järgmise taseme lause sisse)
- Lõputu korduse saab while-lause abil määrata tingimusega, mille väärtus on alati tõene: **while True:**

For-lause

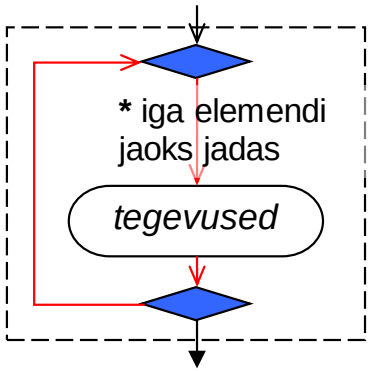
For-lause üldkuju:

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 määratlemise üheks võimaluseks on funktsioon range:

range ([algus,] lõpp [, samm])

- kui puudub algus, võetakse see võrdseks nulliga
- kui puudub samm, võetakse see võrdseks ühega
- jada viimane liige on vähemalt ühe sammu võrra väiksem lõppväärtusest

Loendid

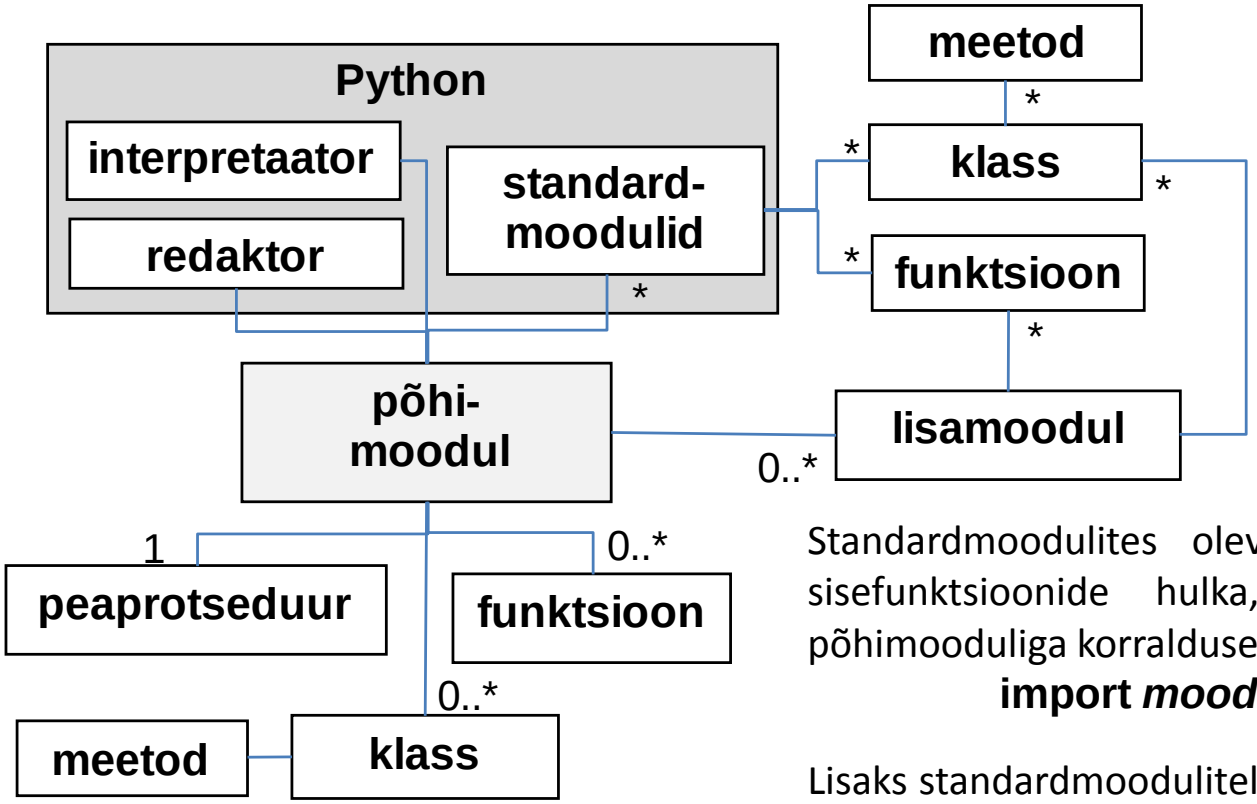
- **Loend** (*List*) ehk ühemõõtmeline massiiv on järjestatud mäluväljade (muutujate) kogum, mis tähistatakse ühe nimega
- Viitamiseks elementidele toimub indeksnime (nimi ja järjenumber) abil:

  loendi_nimi[***indeks***]
- Indeks paigutatakse nurksulgudesse [ ] ja võib olla esitatud avaldise abil
- Pythoni loendi elementide järjenumbrid algavad alati nullist
- Loendi pikkus (elementide arv loendis) vt funktsioon **len**(**loendi_nimi**)
- Loendite loomiseks saab andmed:
  - sisestada klaviatuurilt
  - luua avaldiste abil (näiteks ka juhuarvudest)
  - lugeda väärtused failist
  - omistada väärtuste loetelu: nimi = ['Kask', 'Kuusk', 'Mänd', 'Paju', 'Saar', 'Tamm']

Loendid

- len(loend) – loendi pikkus (elementide arv)
- del(loend[nr]) – väärtus(t)e kustutamine
  (nr – järjenumber või numbrite vahemik)
- loend.append(x) – väärtuse lisamine loendi lõppu
- loend.extend(*L*) – loendi lisamine loendi lõppu
- loend.insert(*i*, *x*) – väärtuse x lisamine kohale i
- loend.remove(*x*) - väärtuse x eemaldamine loendist
- loend.pop([*i*]) – väärtuse lugemine ja eemaldamine
  (väljavõtmine); kui i puudub, võetakse viimane

Rakenduse struktuur ja põhielemendid



Standardmoodulites olevad funktsioonid ei kuulu sisefunktsioonide hulka, mooduli peab siduma põhimooduliga korraldusega

import *mooduli_nimi*

Lisaks standardmoodulitele on Pythoni jaoks suur hulk lisamooduleid (seotakse samuti import käsuga).

Kasutaja saab ise luua funktsioone ja klasse ning paigutada need oma rakenduse põhimoodulisse või moodustada neist eraldi lisamooduli(d).

Funktsiooni ja meetodi poole pöördumine

- Sisefunktsiooni poole pöördutakse **funktsiooniviida** abil:

  funktsiooni_nimi(*argumendid*)

- Argumentide arv, tähendus, tüüp ja järjekord sõltub funktsioonist
- Moodulid lisatakse rakendusele:
  - **import mooduli_nimi**

    nii lisatud mooduli funktsioonide (meetodite) ja konstantide (omaduste) poole pöördutakse

    mooduli_nimi.funktsiooni_nimi(*argumendid*)
  - **from mooduli_nimi import *** (või funktsioonide loetelu)

    nii lisatud mooduli funktsioonide (meetodite) ja konstantide (omaduste) poole pöördutakse nagu sisefunktsioonide poole:

    funktsiooni_nimi(*argumendid*)
- Objekti meetodite poole pöördutakse:

  objekti_nimi.meetodi_nimi(*argumendid*)

Funktsioonide loomine

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

laused

[**return** *avaldis*]

…

- Funktsiooni päis algab võtmesõnaga **def**, millele järgneb nimi ja sulgudes parameetrid; tühjad sulud peavad olema ka siis, kui parameetrid puuduvad.
- Päise lõpus peab olema **koolon :**
- Funktsiooni sisu ehk keha laused peavad paiknema päise suhtes taandega
- Lause **return *avaldis(ed)*** lõpetab funktsiooni töö ja tagastab väärtuse(d)
- Pöördumine funktsiooni poole:

  [***muutuja*** =] **nimi** ([***argumendid***])

  muutuja on vajalik, kui funktsioon tagastab väärtuse
- Argumentide arv, tüüp ja järjekord peab vastama parameetritele

Tekstifailid

- Fail koosneb **kirjetest** ehk ridadest. Kirje koosneb ühest või mitmest **väljast**.
- Väljad eraldatakse üksteisest mingi eraldajaga, mida ei tohi esineda välja väärtuse sees.
  Kirje lõpus on (mittenähtav) reavahetuse sümbol **\n**.
- Enne lugemist või kirjutamist avatakse fail korraldusega:

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

  failiobjekt - objektimuutuja

  failinimi - faili täisnimi: [**tee**]nimi.txt

  "**w**" - kirjutamine, "**r**" - lugemine (võetakse ka vaikimisi)
- Peale töötlemist tuleb fail sulgeda meetodiga **close**()

  failiobjekt.**close**()

Andmete lugemine failist

- Terve faili lugemine korraga:

  loend = *failiobjekt*.**readlines**()

- Andmete lugemine kirje kaupa:

  kirje = *failiobjekt*.**readline**()

- Kirjete lugemine korduses:

  for *kirje* in *failiobjekt*:

  tegevused ***kirjega***

- Stringi (objekti) meetodid abiks:

  stringavaldis.strip() – tühikute ja juhtsümbolite (\n \t) eemaldamine

  stringavaldis.split() – moodustub loend

Andmete kirjutamine faili

- Fail peab olema avatud kirjutamiseks (töötlusviis “w”, “a”)
- Kirjutamise põhivariant

  *failiobjekt.***write**(*string*)

- String tuleb koostada ühe kirje andmetest
  - arvud teisendatakse stringideks (fn. str())
  - väljade vahele tuleb lisada eraldaja (tühik)
  - kirje lõppu lisada \n
- Kirje lisamiseks faili sobib ka

  *failiobjekt.***print**(*v1, v2, … , file=failiobjekt*)

  v1, v2, … - väärtused (ei pruugi olla tekstid)
  failiobjekt - kirjutamiseks avatud fail

  print lisab vaikimisi väärtuste vahele tühikud ja lõppu \n