

1.1 Algoritm

ALGORITM

Kui me midagi suuremat teha tahame, siis võib tulemuse saavutamisel abi olla sellest, kui me kavandatu kuidagi sammudeks jaotame. Seejuures me peaksime muidugi mõtlema, mis järjekorras need sammud tuleb või saab teha. Kas mingeid samme saab teha samaaegselt? Kas midagi tuleb teha korduvalt? Kas mõnda asja tuleb või saab teha ainult teatud tingimustel? Äkki saab hoopis keegi teine meie endi asemel midagi ära teha?

Elus paraku (kahjuks või õnneks) mõeldakse sammude peale sageli alles pärast nende toimumist. Kui aga tahta kellelegi teisele selgeks teha, kuidas teatud asi toimuma peaks, siis peaks selleks kasutama mingit mõlemale poolele arusaadavat viisi. Vast on nii mõnedki kogenud, et täpsed juhised võivad kaasa aidata tulemuse saamisele: "Pööra ringteelt välja teiselt mahasõidult!", "Keeda 20 minutit!", "Lisa kaks labidatäit kruusa!", "Istuge, palun!".

Kui soovime ümbritsevat maailma mingit moodi käituma suunata, siis hea meetod selleks on kehtestada eeskirjad. Eeskirjad võivad olla keelavad või kohustavad. Keelavad eeskirjad on näiteks sellised: "Ära mine vales kohas üle tänava!", "Ära söö (joo) nii palju!", "Ära ületa kiirust!", "Ära mängi nii palju arvutimänge!", "Ära ...".

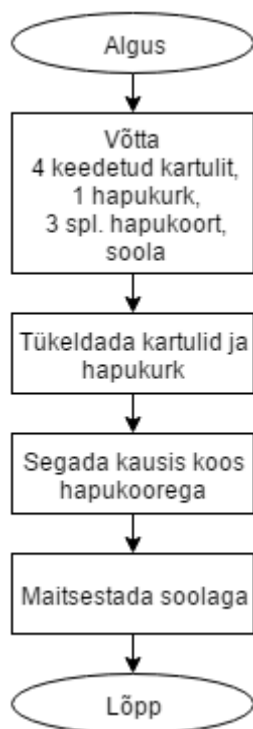
Nüüd aga keskendume kohustavatele eeskirjadele - neile, mis kirjeldavad tegevusi, mida peab tegema. Tavaliselt on elus mingi ülesande lahendamiseks vaja teha mitu sammu järjest - sel juhul saame rääkida lahenduseeskirjast ehk **algoritmist**. Algoritm ongi mingi hulk õiges järjekorras kohustavaid eeskirju (vt ka [Eesti keele seletavast sõnaraamatust](#)).

ALGORITMIDE ESITAMINE

Algoritmi saab põhimõtteliselt kirja panna erineval moel. Näiteks võib selle lihtsalt samme järjest (nummerdades) üles kirjutada. Uurige, kuidas Omniva on esitanud [algoritmi paki saatmiseks](#). Suureks abiks võivad olla illustratsioonid, näiteks [esmaabi andmise puhul](#). Mõnedel juhtudel ongi ainult illustratsioonid, näiteks lennukis olevatel ohutuskaartidel.

Üsna tavaline on, et teatud samme tuleb (või saab) sooritada ainult siis, kui teatud tingimus on täidetud. Näiteks on vingugaasimürgituse esmaabi puhul kirjas: "Kui kannatanu ei hinga, tee kunstlikku hingamist."

Üks levinud viis algoritmi kirjeldamiseks on plokkskeem. Meie plokkskeemis tähistatakse algoritmi algust ja lõppu ovaalide abil (vt järgnevat joonist). Algoritmil on alati üks algus ja üks lõpp. N-õ tavalise sammu tähistamiseks kasutatakse riskülikut. Plokkide järgnevust märgitakse nooltega. Nii kirjeldavad kartulisalati tegemist tänaseks juba emeriitprofessorid Mare Koit, Ülo Kaasik ja Jüri Kiho oma õpikus "Kuidas programmeerida" (1990). (Neil oli küll veidi keerulisem retsept.)



Samale tulemusele võib jõuda ka mõnevõrra teistsuguse algoritmiga. Mõned sammud võivad olla teises järjekorras, aga mõnede sammude puhul on omavaheline järjekord kindel. See, millised sammud ühte plokki lugeda, võib olla ka üsna vabalt valitav. Kui tahame midagi paralleelselt teha, et tulemust kiiremini saada, siis võiks salati tegemisel kartulite ja hapukurgi tükeldamist eraldi võtta ja neid lasta erinevatel inimestel samal ajal teha. Siinkohal kerkib loomulik küsimus: kas meil on olemas selleks vajalikke ressursse - inimesi, nuge, lõikelaudasid? Paralleelsete protsesside kasutamine programmides on praegusel ajal tegelikult äärmiselt oluline, aga ka keeruline ja see jääb meie kursusest välja.

Ülesanne

Tegemist on enesekontrolliülesandega, mille lahendamise tulemusi ei salvestata. Võite julgesti ka valesti vastata, aga püüdke ikka õigesti.

Järjestage lennukis hapnikumaski kasutamise tegevused

Aidata teisi hapnikumaski paigaldamisel	▼
Tõmmata lähim mask endale	▼
Hingata normaalselt	▼
Hapnikumaskid ilmuvad automaatselt teie ette	▼
Kinnitada mask endale üle nina ja suu	▼
Kasutada maski kuni meeskonnaliige lubab selle ära võtta	▼

Ülesanne

Järjestage kontserdil käimise tegevused

Väljun saalist ▼

Saan kontserdielamuse ▼

Ostan pileti ▼

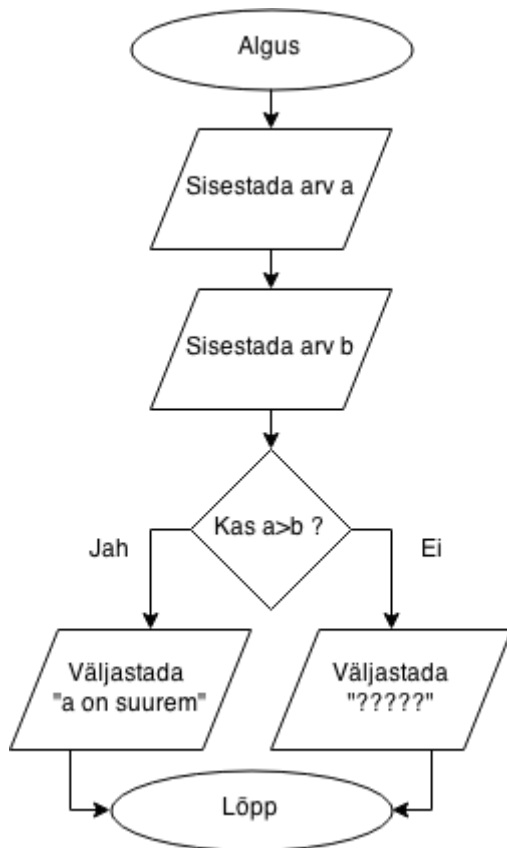
Loen kuulutust ▼

Lähen saali ▼

Mitme tegevuse paralleelselt tegemist me siin ei käsitle. Küll aga on meil olulisel kohal olukorrad, kus tuleb kahest võimalikust jätkust valida üks. Plokkskeemis kasutame selliste hargnemiste - kontrollplokide - kirjeldamiseks rombi. Kontrollplokis on olulisel kohal tingimus, mille täidetuse põhjal otsustatakse, kumba teed edasi minna. Kontrollplokist väljub alati täpselt kaks noolt, sissetulevate noolte hulk ei ole piiratud.

Põhimõtteliselt saab kasutada ka skeeme, kus kontrollplokist väljub rohkem nooli. Selliseid (näiteks [see skeem](#)) on näiteks Euroopa parlamendi ja nõukogu [määruses nr 1272/2008](#), mis käsitleb ainete ja segude klassifitseerimist, märgistamist ja pakendamist. Seal on Jah/Ei asemel näiteks Jah, kiiresti / Jah, aeglaselt / Ei. Meie aga jätkame siiski nii, et on täpselt kaks valikut - kas tingimus kehtib või mitte.

Kui nüüd tulla rohkem arvutite juurde, siis üsna sageli on vaja programmi kasutajalt mingit sisendit ja tulemuseks on kasutajale millegi väljastamine. Sellist andmevahetust märgitakse rööpkülikutega. Järgmises algoritmis sisestab kasutaja arvud a ja b ning pärast kontrolli väljastatakse talle vastav teade.



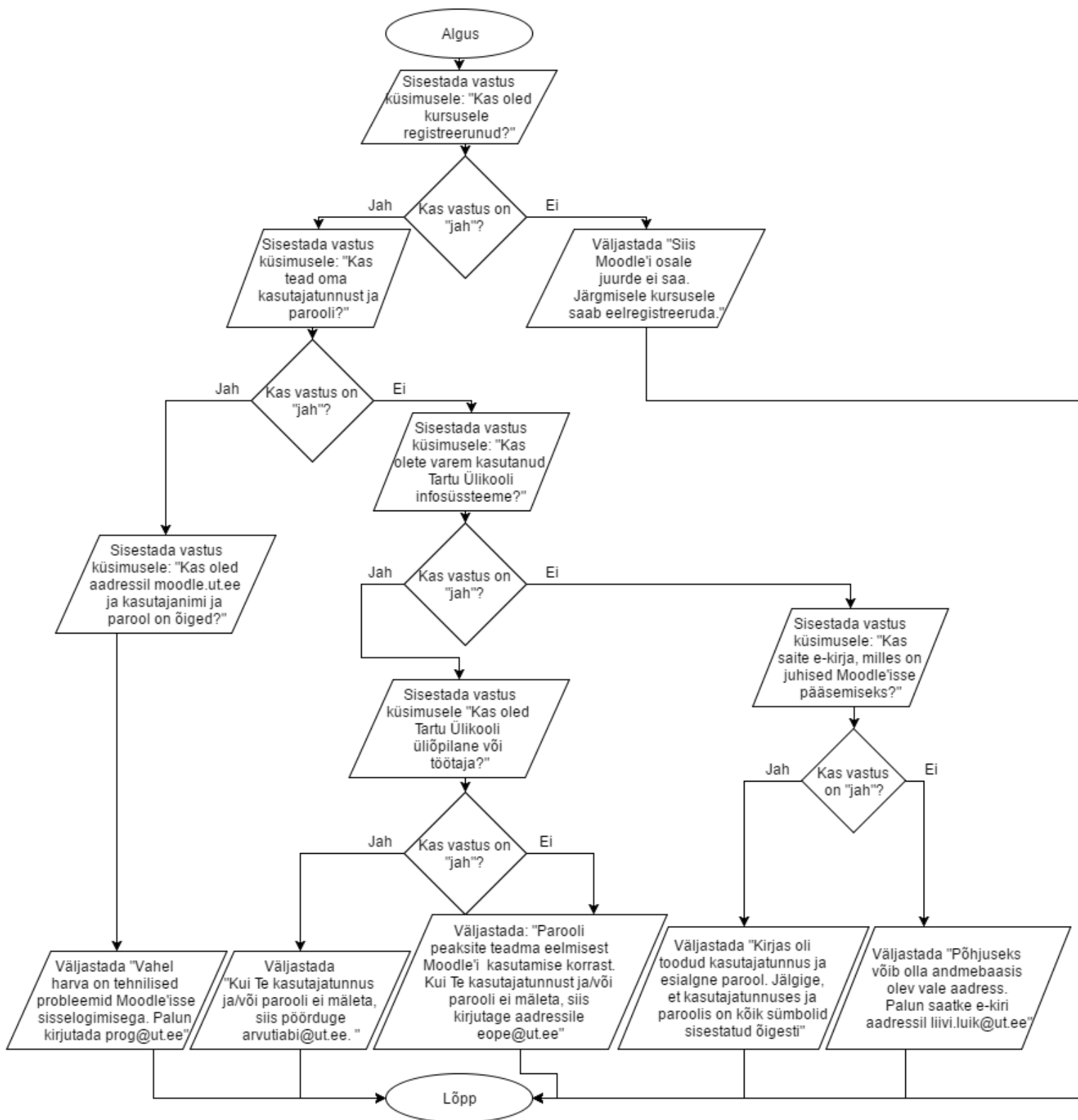
Ülesanne

Milline on eelnevas skeemis sobiv tekst ????? asemel?

- ☐ Vali a on suurem
- ☐ Vali a ei ole suurem
- ☐ Vali a on väiksem
- ☐ Vali a ei ole väiksem

Algoritmi koostamisel tuleb olla väga tähelepanelik ka kõikvõimalike erijuhtude arvestamiseks. Näiteks eelmise algoritmi puhul peab väljastatav tekst olema õige ka juhul, kus sisestatud arvud a ja b on võrdsed.

Enne programmide juurde minekut vaatleme näitena algoritmi, mis kirjeldab sellise programmi tööd, mis püüab aidata kursuslast, kes ei pääse kursuse Moodle'i keskkonda. Siin on see toodud plokkskeemina. (Skeemil klõpsates näeb seda suuremana.)



Enam-vähem sama algoritmi järgi on korraldatud ka [murelahendaja](#).

1.2 Programm

PROGRAMMEERIMISKEELED

Mingi ülesande lahendamiseks vajaliku algoritmi kirjapanek plokk skeemina võib küll olla arusaadav inimesele, aga see ei ole väga sobiv arvutile algoritmi arusaadavaks tegemiseks. Selleks otstarbeks tuleb lahendus esitada **programmina** (vt [Eesti keele seletavast sõnaraamatust](#)). Programm kirjutatakse mingis programmeerimiskeeles. Programmeerimiskeeli on olemas mitmeid sadu, võib-olla tuhandeid. Nagu loomulike keelte puhul, on ka üks programmeerimiskeel mõne teisega sarnasem, mõnest jälle erinevam. Näiteks eesti keel on suhteliselt sarnane soome keelega, vene keel ukraina keelega ja rootsi keel norra keelega. Samas näiteks kuigi eesti keel ja katalaani keel ei ole väga sarnased, on neis mõlemas ikkagi nimisõnad ja tegusõnad ja muud paljudele keeltele omased asjad. Nii on ka erinevates programmeerimiskeeltes palju sarnaseid struktuure. Kuidagi peab olema organiseeritud, et

- kui teatud tingimus on täidetud, siis tehakse ühtmoodi, kui pole täidetud, siis teistmoodi;
- mingit tegevust tehakse teatud arv kordi;
- mingit tegevust tehakse niikaua, kui teatud tingimus on täidetud;
- mingeid programmiosasid saab suhteliselt iseseisvalt kasutada jne...

Programmeerimiskeeli saab tinglikult jaotada põlvkondadesse. (Põlvkondadest ja teistest ajaloolistest asjadest on juttu silmaringimaterjalis [Ajaloost](#).) Rõhuv enamik programme kirjutatakse tänapäeval mingis kõrgtasemekeeles. Neist levinumad on näiteks C (ka C++, C#), Java, PHP ja Python. Erinevate keelte kasutuslevik on pidevas muutumises ja seda on huvitav jälgida näiteks [siin](#).

Meie kursuse aluseks on võetud programmeerimiskeel Python. Ühelt poolt on tegemist keelega, milles tõesti ka päriselt tööstuses programmeeritakse. Teiselt poolt on see sellistest keeltest sobivaim just esimeseks õpitavaks keeleks, sest juba midagigi tegeva programmi saame kirjutada ühe rea abil. Programmeerimiskeele Python esimene versioon loodi 1989. aasta lõpus. Sellel kursusel kasutame versiooni Python 3. Täpsemalt pole versioon (nt 3.4.3 või 3.5.1) nii oluline, kuna meie jaoks on nende erinevused vähetähtsad.

Programmeerimiskeele Python looja nime juurde jõuame aga läbi järgmise küsimuse. Ka edaspidi kasutame vahel küsimusi juba siis, kui vastust veel tekstis toodud pole. Ühelt poolt tundub see võib-olla mõnevõrra kummaline - kuidas küsida õppurilt sellist asja, mida pole veel seletatudki!? Teiselt poolt aga näitavad kogemused, et sellised küsimused aitavad õppija vaimu virge hoida ja paremini teemasse sisse minna. Lisaks ei lähe vastuse hinne kuhugi arvesse, seega pole oma tulemuse pärast vaja liigselt muretseda. Enesekontrolli ülesannete puhul võite julgesti ka meelega valesti vastata, sageli ilmuvad sellisel juhul õpetlikud kommentaarid.

Ülesanne

Kes oli programmeerimiskeele Python looja?

☐ Vali Jean-Claude van Damme

☐ Vali Marco van Basten

☐ Vali Guido van Rossum

☐ Vali Vincent van Gogh

Programmeerimiskeele Python autori kohta leiab natuke infot [eesti keeles](#) ja natuke rohkem [inglise keeles](#).

ESIMENE PROGRAMM

Üks põhjus, miks Python on esimese keelena suhteliselt sobiv, on see, et päris esimese programmi kirjutamine on lihtne. Nii saame teha näiteks üherealise programmi

```
print("Tere!")
```

See programm toob ekraanile sõna "Tere!".

Tavaliselt muidugi on programmis rohkem ridu. Täiendame meiegi oma programmi.

```
print("Tere!")  
print("Head aega!")
```

THONNY JA PYTHONI PAIGALDAMINE ARVUTISSE

Selleks, et programmeerimiskeeles Python programmeerima hakata, on vaja Python oma arvutisse saada. Pythoni saab paigaldada eraldi, aga meie kasutame oma kursusel keskkonda Thonny, millega Python juba kaasas on. Tegelikult on olemas ka veebipõhiseid võimalusi Pythonis programmeerimiseks ja mitmeid teisi keskkondi peale Thonny. Materjalides siiski eeldame, et kasutatakse Thonnyt, sest see võimaldab meil teemasid teatud Thonny abivahenditega illustreerida. Näiteks kasutame Thonny võimalusi muutujate väärtuste näitamiseks ja programmi sammhaaval läbimiseks.

Niisiis palun paigaldage enda arvutisse Thonny. Juhise selleks leiate [siit](#). Palume uue versiooni paigaldada ka neil, kel vana versioon on varasemast juba arvutis olemas. Põhimõtteliselt saab kursuse praktiliselt kõiki ülesandeid lahendada ka nt keskkonna IDLE või mõne teise keskkonna abil, aga materjalid on koostatud Thonnyt silmas pidades.

Püüdke nende juhiste abil eespool olnud lühike Tere-programm käivitada. Tegemist on selle kursuse jaoks väga olulise etapiga. Kui saate esimese programmi tööle, siis on suur võimalus, et saate järgmisedki! Esialgu võivad need sammud võõrad tunduda, aga lähinädalatel hakkate programme koostama, käivitama, parandama ja jälle käivitama kümneid ja sadu kordi.

Pythoni (nagu mistahes teise keele) programm on ühelt poolt täiesti tavaline tekstifail, mida võime kirjutada tavalise tekstiredaktoriga (näiteks Notepadi või Wordiga). Siiski on seda mugavam teha spetsiaalse redaktoriga, näiteks Thonnyga. Tavaliselt on Pythoni programmide failidel laiendiks py, Thonny lisab selle salvestamisel ise.

Kui Thonny paigaldamine oma arvutisse ei õnnestunud, siis palun kirjutage aadressil prog@ut.ee.

Lühema programmiga saime hakkama, püüame natuke pikemaga ka. Selle võite ise sisse kirjutada või siit kopeerida. Proovige programm käima saada!

```
arv = 1
if arv > 0:
    print("positiivne")
else:
    print("mittepositiivne")
```

Pythonis on programmi struktuuri korraldamisel äärmiselt oluline õige taandamine ehk tühikutega mingi rea paremale poole lükkamine. (Teistes keeltes võivad selles rollis olla näiteks looksulud { }). Proovige näiteks taanet niimoodi muuta:

```
arv = 1
if arv > 0:
    print("positiivne")
else:
    print("mittepositiivne")
```

Kui mitte varem, siis nüüd juhtus selline asi, et me tahtsime käivitada vigast programmi. Seda juhtub programmeerimisel alati ja seda ei tohi sugugi karta. Proovige programmi parandada ja uuesti käivitada.

VAHEKOKKUVÕTE

Loodetavasti olete nüüd midagi teada saanud algoritmist ja programmist ning esimesed programmid töölegi saanud.

Nüüd võite proovida leida (ja Moodle'is esitada) ülesande 1.1. lahenduse [kohustuslikest kontrollülesannetest](#). Võite aga kõiki ülesandeid ka lahendada pärast järgmiste selle nädala teemade läbimist.

1.3 Andmetüübid

ANDMETÜÜBID

Eelmises osas oli meil programm

```
arv = 1
if arv > 0:
    print("positiivne")
else:
    print("mittepositiivne")
```

Selles on näha mitut tüüpi andmeid. Paistavad arvud `1` ja `0` ning tekstilised suurused `"positiivne"` ja `"mittepositiivne"`. Pythonis (nagu ka teistes programmeerimiskeeltes) ongi võimalik kasutada erinevaid andmetüüpe. Meie jaoks on esialgu olulised järgmised Pythoni andmetüübid:

- **Täisarvud** (näiteks `10`, `-4`, `0`);
- **Ujukomaarvud** (näiteks `2.5`, `3.14`, `-7.23`, `2.0`);
- **Sõne / tekst** (näiteks `"kass"` (või `'kass'`), `"koer ja hobune"` (või `'koer ja hobune'`));
- **Tõeväärtused** (`True` või `False`).

Täisarvude kasutamine Pythonis on ehk kõige sarnasem (kooli)matemaatikale. Oma tüüp on koma sisaldavatele arvudele, selliseid arve nimetatakseujukomaarvudeks. Koma rollis kasutatakse Pythonis punkti (näiteks `3,2` kirjutatakse Pythonis `3.2`). Tekstide jaoks on sõnetüüp, mille tunnuseks on jutumärgid või ülakomad. Sõne ingliskeelne vaste on *string* (lühendatult ka *str*). Tõeväärtuste tüübis on vaid kaks võimalikku väärtust - `True` (tõene) ja `False` (väär).

Konkreetses väärtuse tüüpi saab teada käsku `type()` kasutades. Täisarvu tüüpi märgitakse Pythonis `int`,ujukomaarvu tüüpi `float`, sõne tüüpi `str` ja tõeväärtuse tüüpi `bool`. Proovige näiteks:

```
print(type(4))
print(type(4.5))
print(type("Maria"))
print(type(True))
```

Ülesanne

Mis ilmub ekraanile?

```
print(type("5.0"))
```

Vali ☐ <class 'int'>

Vali ☐ <class 'float'>

Vali ☐ <class 'str'>

Vali ☐ Veateade

Ülesanne

Mis ilmub ekraanile?

```
print(type(5,0))
```

Vali ☐ <class 'int'>

Vali ☐ <class 'float'>

Vali ☐ <class 'str'>

Vali ☐ Veateade

Tüüpidega samanimeliste funktsioonide abil saame leida konkreetse väärtuse teist tüüpi vaste. Vahest on siin tulemus selgesti ettearvatav, vahel aga võib see suhteliselt ootamatu tunduda. Nii on näiteks `int("1")` väärtuseks täisarv `1`, sest sõnele `"1"` on just täisarvudest `1` loomulik vaste. Samuti vastupidi - `str(1)` väärtuseks on sõne `"1"`. Kontrollige ka, mida Python ise nende andmete tüüpidest arvab:

```
print(type(str(1)))  
print(type(int("1")))
```

Mõnedel juhtudel pole tüübteisenduse funktsiooni väärtuse ennustamine nii lihtne. Proovige näiteks järgmisi näiteid.

```
print(bool(7))  
print(bool(0))  
print(float(True))  
print(int(1.7))
```

Vaatame viimast rida, mis andis `int(1.7)` väärtuseks `1`. Nagu näeme ei ole tegemist ümardamisega, kuna ümardamisreeglite järgi peaks väärtus olema `2`. Ümardamiseks on Pythonis eraldi funktsioon `round`. Näiteks `round(1.7)` on tõesti `2`.

Tuleme teiste näidete juurde tagasi hiljem, kui vastavaid tüüpe põhjalikumalt käsitleme.

Erinevat andmetüüpi väärtusi on vaja erinevates olukordades, nendega arvutused toimuvad erinevalt. Erinevate tüüpidega saab teha erinevaid tehteid. Ka näiliselt sama tehtemärgiga võivad erinevad tüübid erinevalt käituda. Oluline on ka see, et erinevat tüüpi väärtusi kujutatakse arvuti mälus erinevalt - ka ruumi võtavad nad mälus erinevalt.

1.4 Muutujad

MUUTUJAD

Kui need andmed juba mälus on, siis on loomulik tahtmine neid ka kasutada. Konkreetsele mälupiirkonnale, kus teatud väärtus on, saab anda eraldi nime. Sellist nimega mälupiirkonda kutsutakse **muutujaks**. Muutujal on nimi ja muutujale väärtuse andmine käib Pythonis võrdusmärgi abil.

Tinglikult võiks muutujaid võrrelda maitseainete topsidega. Selleks, et maitseaineid lihtsamini üles leida, on mõistlik topsid vastavalt sildistada. Sildi järgi leitakse vastav maitseaine paremini üles.



Muutujaid võiks võrrelda maitseainete topsidega. Allikas: Eno Tõnissoni erakogu

Enne oli juttu, et erinevat tüüpi väärtusi kujutatakse mälus erinevalt. Laias laastus on seegi võrreldav maitseainetopsidega - need võivad näiteks olla erineva suurusega.

MUUTUJALE VÄÄRTUSE ANDMINE

Muutuja on kasulik siis, kui talle anda mingi väärtus - määratakse andmed, mida muutujale vastavas mälupiirkonnas hoitakse. Umbes nii, et topsis, mille peale on kirjutatud kaneel, on hoiul kaneel.

Pythonis muutuja nime valimisel on omad reeglid ja tavad nagu ka näiteks tavaelus lapsele nime panemisel. (Reeglid ise on küll lapsele nimepanemise omadest mõnevõrra erinevad.) Pythonis

muutujale nime panemisel tuleb silmas pidada, et nimi ei tohi sisaldada tühikuid. Tühiku asemel kasutatakse vajadusel alakriipsu (nt `vanima_lapse_vanus`) või jäetakse vahe üldse ära (nt `vanimaLapseVanus`). Lisaks on tavaks, et muutuja nimi on kirjutatud väikeste tähtedega (va sõnade algustähed alates teisest). Nimi ei tohi alata numbriga, aga alates teisest sümbolist võib numbraid olla küll. Nimedena ei saa kasutada mõningaid Pythonis eritähendusega sõnu, näiteks `and`, `def`, `elif`, `else`, `for`, `from`, `if`, `import`, `in`, `not`, `or`, `pass`, `return`, `while` ja tõeväärtusi `True` ja `False`.

Erinevalt mitmetest teistest programmeerimiskeeltest (nt Java) ei määrata Pythonis programmeerija poolt ilmutatult, mis tüüpi väärtused antud muutujale sobivad ehk mis tüüpi muutuja on. Python saab ise aru, et kui väärtuseks on `12`, siis on tegemist täisarvuga, kui `12.1`, siis ujukomaarvuga, kui `"Kaua Sa kannatad kurbade naeru"`, siis sõnega.

Soovi korral saab lugeda erinevate nimevaliku soovitude kohta [ingliseelsest Wikipedia artiklist](#).

Järgmises videos ja sellele järgnevas tekstis tegeletakse sarnase näitega, soovitame valida endale sobiva või vajadusel mõlemaid uurida: <https://youtu.be/Vz8EfZRSMB0>

Vaatame nüüd näidet, kus väärtustame erinevaid muutujaid. Anname muutujale `poisi_vanus` väärtuseks `12`. Selleks käivitame Thonny ning trükime uude programmiaknasse (File -> New) esimesele reale:

```
poisi_vanus = 12
```

Toome mängu vanaema, andes teisel real muutujale `vanaema_vanus` väärtuseks `59`:

```
vanaema_vanus = 59
```

Talletatud infot saame kasutada samade muutujate abil. Võime näiteks leida poisi ja vanaema vanuste summa, trükkides programmi kolmandale reale:

```
vanuste_summa = poisi_vanus + vanaema_vanus
```

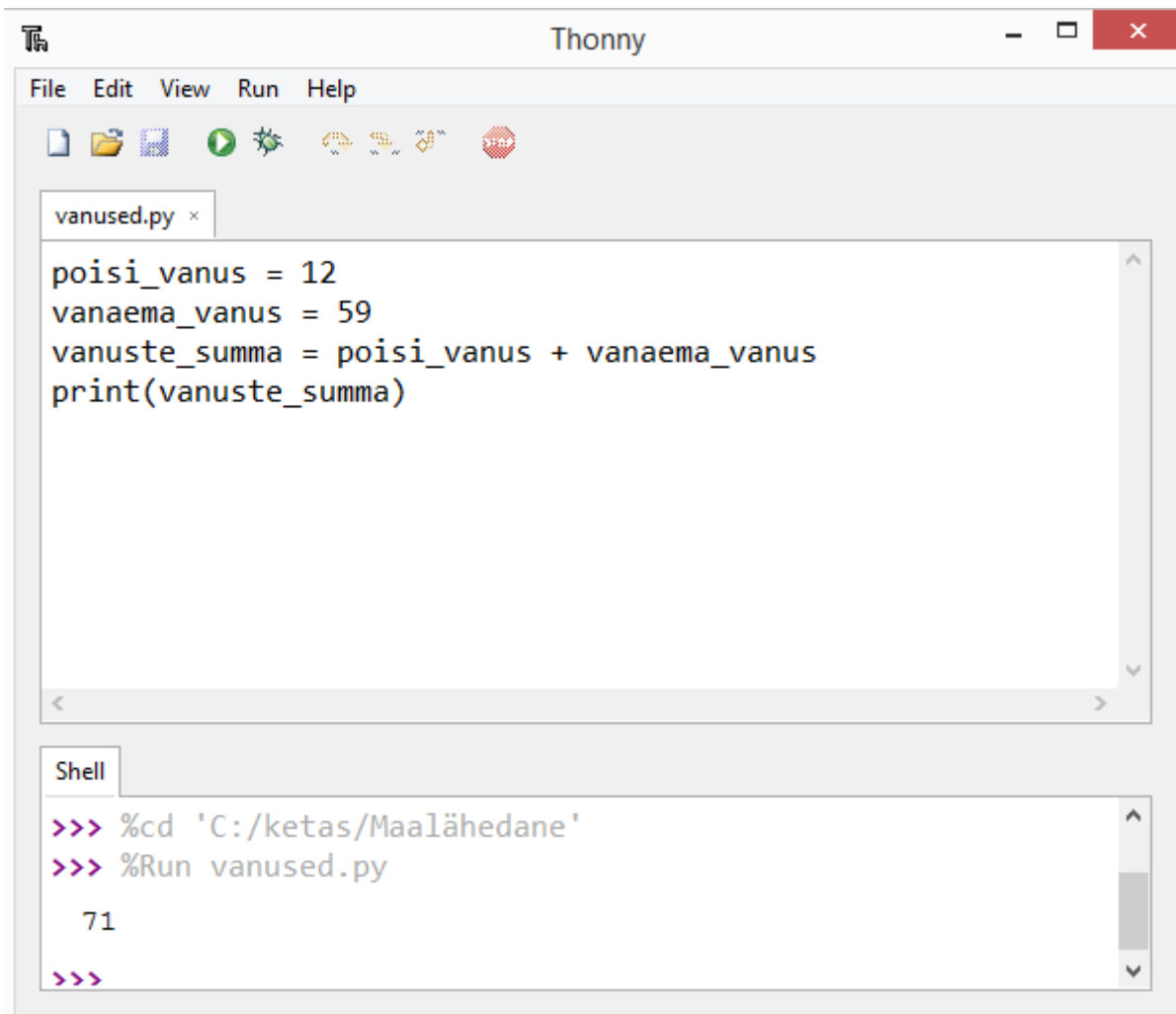
Ekraanile väljastamiseks tuleks lisada käsk `print`.

```
print(vanuste_summa)
```

Kokku saime järgneva programmikoodi:

```
poisi_vanus = 12
vanaema_vanus = 59
vanuste_summa = poisi_vanus + vanaema_vanus
print(vanuste_summa)
```

Programmi käivitades (Run -> Run current script) peaks tulemus olema selline:



The screenshot shows the Thonny Python IDE interface. The main editor window displays a Python script named 'vanused.py' with the following code:

```
poisi_vanus = 12
vanaema_vanus = 59
vanuste_summa = poisi_vanus + vanaema_vanus
print(vanuste_summa)
```

Below the editor is a 'Shell' window showing the execution output:

```
>>> %cd 'C:/ketas/Maalähedane'
>>> %Run vanused.py
71
>>>
```

Nuputa! Kuidas leida poisi ja vanaema keskmine vanus?

Muutujatega tegutsemisel tuleb meele pidada, et muutuja peab enne selle kasutamist olema väärtustatud. Proovige käivitada juba varasemalt vaadeldud näidet sellisena:

```
poisi_vanus = 12
vanuste_summa = poisi_vanus + vanaema_vanus
vanaema_vanus = 59
print(vanuste_summa)
```

Millise teate väljastas Python? Miks selline veateade ilmnes ja kuidas viga parandada saab?

ARVULISTE VÄÄRTUSTE OMISTAMINE MUUTUJALE

Muutujale arvulise väärtuse omistamist juba vaatasime. Käib see siis näiteks nii:

```
lemmik_arv = 7  
print(lemmik_arv)
```

Lisaks sellele on võimalik arvudega sooritada erinevaid tehteid. Vaadake tähelepanelikult, sest mõned märgid tähendavad teisi asju kui me harjunud oleme:

- `+` liitmine
- `-` lahutamine
- `/` jagamine
- `*` korrutamine
- `**` astendamine
- `%` jäägi leidmine
- `//` täisosa leidmine

Lisaks tehtemärkidele on olulisel kohal erinevad funktsioonid, nii saame näiteks funktsiooniga `max` leida mitmest suurusest suurima. Näiteks `max(23, 56)` väärtus on `56`.

Võimalusi on veel hulga rohkem, võite nendega tutvuda [Pythoni dokumentatsioonis](#).

Ülesanne

Mida väljastatakse ekraanile?

```
arv = 3**2  
print(arv)
```

Vali 1.5

Vali 3

Vali 6

Vali 9

Vali Veateade

Järgmine näide on ujukomaarvude omistamisest kahele muutujale, nende summa arvutamisest ja ekraanile väljastamiseks:

```
esimene_arv = 3.2
teine_arv = 10.15
arvude_summa = esimene_arv + teine_arv
print(arvude_summa)
```

Muutuja väärtust saab programmis ka muuta. Proovige järgi, mis tulemus tuleb sellisel juhul:

```
esimene_arv = 3.2
teine_arv = 10.15

esimene_arv = 5

arvude_summa = esimene_arv + teine_arv
print(arvude_summa)
```

Ülesanne

Mida väljastatakse ekraanile?

```
a = 1
b = a
a = 2
print(b)
```

Vali 1

Vali 2

Vali 3

Vali b

Vali Veateade

TEKSTILISE VÄÄRTUSE ANDMINE MUUTUJALE. SÕNE

Eelmistes näidetes andsime muutujale väärtuseks arvu (täisarvu või ujukomaarvu). Nüüd vaatame, kuidas muutujale anda väärtuseks tekst sõnena. Tekstiline väärtus omistatakse muutujale jutumärkide (või ülakomade) vahel. Mitu teksti saame kokku panna, kasutades `+` märke.

Proovige näiteks, kuidas töötab järgnev programm:

```
nimi1 = "Mihkel"
nimi2 = "Mari"
tervitus = "Tere, " + nimi1 + " ja " + nimi2 + "!"
print(tervitus)
```

See on üks nendest näidetest, kus üks ja sama tehtemärk tähendab erinevate andmetüüpide puhul erinevaid asju:

- Arvude puhul tähendab `+` märk kahe arvu liitmist (`5 + 3` annab tulemuseks `8`).
- Sõnede puhul tähendab `+` märk kahe sõne ühendamist üheks sõneks (`"tere" + "tore"` annab tulemuseks `"teretore"`). Sel tehtel on ka peenem nimi - *konkatenatsioon*.

Nagu eelmisest programminäitest näha, siis sõned kirjutatakse jutumärkidega, muutujanimed aga ilma.

NB! Sellisel viisil saab omavahel ühendada **ainult** sõnesid. Arvu lisamiseks antud lausesse peaks kasutama funktsiooni `str()`, et arv esmalt sõneks teisendada. Näiteks kui muutuja `arv` sisaldaks väärtust `5`, siis `str(arv)` annaks väärtuse `"5"` (sõnena), mida saaks juba teiste sõnedega liita.

Ülesanne

Mida väljastatakse ekraanile?

```
a = 4
b = 5.5
s = a + b
print("Summa on " + s)
```

Vali Summa on 9.5

Vali Summa on9.5

Vali Summa on 9,5

Vali Summa on9,5

Vali Veateade

TÕEVÄÄRTUSE OMISTAMINE MUUTUJALE

Kui arvud ja sõned on tuttavamana tunduvad andmetüübid (oleme ju arvude ja tekstiga ikka tegutsenud), siis tõeväärtused on võib-olla mõnevõrra võõramad. Nimelt saab muutujale väärtuseks anda ka väärtusi `True` (tõene) või `False` (väär):

```
liigaasta = False  
print(liigaasta)
```

Tõeväärtuste tüüpi nimetatakse algebralise loogika looja [George Boole](#)'i järgi inglise keeles *boolean* tüübiks. Kohata võib ka sõna *boolean* lühendatud varianti *bool*. Põhjalikumalt vaatame tõeväärtuste rakendusi tingimuslausete peatüki juures.

1.5 Sisend kasutajalt: input()

Varasemad näited olid meil sellised, kus andmed kirjutati programmiteksti sisse. Sageli on aga vaja andmed programmi töö vältel juurde saada. Failides (sh internetis) olevatele andmetele juurdepääsust räägime mõne nädala pärast. Praegu aga arutame seda, kuidas programm saaks kasutajalt andmeid küsida ja siis neid ka kasutada. Kui vaatame programmi poolt, siis on kasutajalt saadavad andmed sisendiks. Vastava funktsiooni nimi `input` just tõlkes sisendit tähendabki.

Võite kohe edasi lugeda või vaadata hoopis videot, mis ka sama teemat tutvustab:

<https://youtu.be/5hsnPZn0oYo>

Alustame programmist, mis küsib kasutajalt nime ja siis tervitab teda nimepidi. Nagu ikka, saab siingi programmi mitmel viisil koostada. Alustame sellest variandist, kus kõigepealt väljastatakse ekraanile küsimus, siis väärtustatakse muutuja kasutajalt saadud vastusega ning lõpuks väljastatakse tervitus.

```
print("Palun sisestage nimi")
nimi = input()
print("Tere, " + nimi + "!")
```

Programmi Thonnys käivitades ilmub küsimus Thonny allosas asuvasse *Shell*-paneelile ja sinna tuleb ka vastus kirjutada. Klaviatuuri reavahetusklahviga (Enter) sisestus aktsepteeritakse. Proovige programm käivitada!

Programmis saab küsimuse teksti esitada ka koos funktsiooniga `input`:

```
nimi = input("Palun sisestage nimi")
print("Tere, " + nimi + "!")
```

See programm ei tööta siiski täpselt samamoodi kui eelmine. Nüüd oodatakse kasutaja vastust küsimusega samal real. Selleks, et küsimus ja vastus eralduksid, lisame küsimuse teksti taha kooloni ja tühiku:

```
nimi = input("Palun sisestage nimi: ")
print("Tere, " + nimi + "!")
```

Juba varasemast teame, et Pythonis on erinevat tüüpi andmeid. Selgitame, mis tüüpi väärtuse saame funktsiooni `input` abil:

```
nimi = input("Palun sisestage nimi: ")
print(type(nimi))
```

Näeme, et tegemist on sõnega (`str`). Tänu sellele saime ka programmis selle plussmärgi abil sujuvalt teiste sõnedega (nt "Tere, " ja "!") ühendada. Paljudel juhtudel aga on meil vaja kasutajalt küsida näiteks hoopis arve.

Küsime siis näiteks hoopis vanust. Muutujale paneme nimeks `vanus`.

```
vanus = input("Palun sisestage oma vanus: ")  
print(type(vanus))
```

Proovige programmküsimusele mingi arvuga vastata.

Näeme endiselt, et tegemist on sõnega. Sellest, et muutuja nime vahetasime, ei muutunud tegelikult programm üldse. Muutujate nimede sisulisus on tähtis programmeerijale, Python tegelikult nime konkreetsest tähendusest aru ei saa.

Senikaua kui meil pole vaja teha n-ö arvulisi tehteid, kõlbavad sõne kujul arvud ka.

```
vanus = input("Palun sisestage oma vanus: ")  
print("Teie vanus on " + vanus)
```

Näiteks, kui sisestate vastuseks 31, siis oleks see sama, kui programmis oleks rida `vanus = "31"`.

Tegelikult on meil ju olemas vahendid mingi väärtuse teises andmetüübis kujutamiseks.

Näiteks `str` sõnena kujutamiseks, `int` täisarvuna kujutamiseks, `float` ujukomaarvuna kujutamiseks.

Kui tahame sisestatut täisarvuna käsitleda, siis võime kirjutada `vanus = int(input("Palun sisestage oma vanus: "))`. Sellisel juhul on vanus juba täisarv ja sellega saame teha vastavaid tehteid.

```
vanus = int(input("Palun sisestage oma vanus: "))  
vanus_viie_aasta_pärast = vanus + 5  
print("Teie vanus 5 aasta pärast on " +  
      str(vanus_viie_aasta_pärast))
```

Selles näites kasutatakse kaht tüübiteisendust: `vanus` peab olema täisarv, et saaks teha arvutusi ja väljatrükkimiseks jälle on vaja hoopis sõnet.

Kui selle programmi puhul sisestada midagi, mida `int` ei saa täisarvuks teisendada, siis tekib veateade.

Selle kursuse materjalides eeldame reeglina, et kasutaja sisestab seda tüüpi asju, mida küsitakse. Päriselt muidugi nähakse väga suurt vaeva, et kasutaja (kas siis kogemata või meelega) ei saaks sisestada midagi sellist, mis pole oodatud ja võib-olla programmi hoopis katki teeb.

Järgmises programmis küsitakse kasutajalt täisarvud ja need liidetakse kokku.

```
esimene_arv = int(input("Sisesta esimene liidetav täisarv: "))  
teine_arv = int(input("Sisesta teine liidetav täisarv: "))  
arvude_summa = esimene_arv + teine_arv  
print(arvude_summa)
```

Siin me väljastamisel sõneks teisendamist ei kasutanud, sest `print` tuleb arvude väljastamisega toime küll. Varasemates näidetes oli `str` vajalik just sõnede ühendamiseks.

Siin programmis küsitakse kahte arvu. See, millises järjekorras neid sisestatakse, pole siin oluline. Kui kaks esimest rida ära vahetada, siis tulemus on ikka sama.

```
teine_arv = int(input("Sisesta teine liidetav täisarv: "))
esimene_arv = int(input("Sisesta esimene liidetav täisarv: "))
arvude_summa = esimene_arv + teine_arv
print(arvude_summa)
```

Küll aga võib sisestuse järjekord olla oluline mitmete teiste programmide korral. Meie kursusel võib õige sisestamise järjekord olla oluline lahenduste automaatkontrolli puhul. Sel juhul on see ka ülesande püstituses kirjas.

Lõpetuseks koostame programmi, mis küsib kasutajalt täisarvu ning seejärel

- liidab talle temast suuruse poolest järgmise arvu;
- liidab tulemusele arvu 9;
- jagab tulemuse arvuga 2;
- lahutab tulemusest esialgse arvu.

Eeltoodud tingimustele vastab järgmine programm.

```
algne_arv = int(input("Sisestage täisarv: "))
järgmine_arv = algne_arv + 1
tulemus = algne_arv + järgmine_arv
tulemus = tulemus + 9
tulemus = tulemus / 2
tulemus = tulemus - algne_arv
print(tulemus)
```

Võib tunduda kummaline, et kuidas me saame muutujale väärtuse anda tehtega, milles kasutame sedasama muutujat ennast (nt `tulemus = tulemus + 9`). Tegelikult on see aga täiesti lubatud ja sageli kasutatav! Nii saabki muutuja väärtust muuta.

Proovige programmi erinevate arvudega! Milliseid tulemusi saate?

Tulemus paistab olevat ujukomaarv. Mis hetkel see tekib? Thonnyga võime jälgida muutujate väärtusi programmi jooksutamise ajal. Selleks tuleb muutujate paneel avada (View --> Variables). Kui tavalisel moel programm käivitada (Run --> Run current script või vastav rohelise ringiga nupp või klahv F5), siis tehakse programmi töö pärast kasutajalt sisendi saamist ühe hooga ära ja paistavad muutujate lõppväärtused.

Kui aga kasutada Thonny samm-sammulise läbimise võimalusi, siis näeme muutujate väärtuste muutumist ka samm-sammult. Selleks tuleb programm käivitada silumisrežiimis (Run --> Debug current script või putukaga nupp või Ctrl + F5). Näeme, et programmi esimene rida on esiletoodud.

Nüüd saame sammukaupa programmis edasi minna (Run --> Step over või vastav noolega nupp või klahv F6).

Proovige! Millal saab muutuja `tulemus` väärtuseks ujukomaarvu?

Proovige, mis muutub, kui jagamismärgi `/` asemel kasutada hoopis kahte jagamismärki kõrvuti `//`. Tegemist on täisarvulise jagamisega. Kahe arvu jagamisel on tulemuseks alati täisarv (või sellele vastav ujukomaarv). Näiteks `5 // 3` väärtuseks on `1`, `5.1 // 3` väärtuseks aga `1.0`. Tuleb tähele panna, et tulemuse saamiseks jagatist ei ümardata, vaid tulemuseks on kõige suurem täisarv, millest jagatis ei ole väiksem.

Vastake enesetesti küsimusele ning katsetage ka vastava programmiga.

Ülesanne

Millised tehted järgmistest annavad tulemuseks 2?

- ☐ `5 // 2`
- ☐ `-5 // -2`
- ☐ `5.0 // 2`
- ☐ `5 / 2`
- ☐ `4 / 2`
- ☐ `5.0 // 2.0`

1. nädala kontrollülesandeks 1.3 on astendamise programmi koostamine. Ülesandega kohanemiseks võiks lihtsalt erinevaid astmeid leida.

```
print(3 ** 2)
print(10 ** 100)
```

Teine arv on siis 1 saja nulliga. Lõpetame siin aga enesetesti küsimusega, mille eest meie kursusel ei saa punktigi, sest enesetesti küsimuste eest meil punkte ei saadagi. 2001. aastal aga Suurbritannia telemängus "Kes tahab saada miljonäriks?" oli see miljoni naela küsimus.

Ülesanne

Kuidas nimetatakse arvu, milles on 1 saja nulliga?

Vali gigabitt

Vali gugol

Vali megatron

Vali nanomool

Sellest arvust ja ka sellest küsimusest on juttu [siin](#). Vastajast (inglise keeles) [siin](#).

1.6 Silmaring. Katkeid programmeerimise ajaloost

Ajaloost

Ühelt poolt on programmeerimise ajalugu küllaltki lühike, eriti võrreldes mõnede teiste valdkondadega. Teiselt poolt on aga siingi juhtunud palju huvitavaid sündmusi, on olnud erinevaid ajajärke ja tegutsenud värvikaid persoone. Siin saame tutvustada vaid vähest, aga loodetavasti mingi (küll väga fragmentaarset) taustapildi siiski saab.

Programmeerimise ajalugu algab juba enne seda, kui arvutid füüsiliselt valmis said. Nimelt leidis inglise matemaatik [Charles Babbage](#) 1812. aastal, et teatud arvutusi võiks teha hoopis masin. Aastakümnete jooksul tegeles ta selle mõtte realiseerimisega. 1842. aastal tutvustas ta loengus universaalse mehaanilise arvuti - analüütilise masina ideed. Teatud eeskujuks olid automaatsed kangasteljed. Varasematest ideedest eristas seda masinat just see, et protsessi pidi juhitama varemkoostatud juhiste järgi. C. Babbage püüdis seda masinat ka valmis ehitada, aga tolleaegsete tehniliste võimaluste piiratuse tõttu see ei õnnestunud. Masina lihtsam kuju tehti valmis aastaid hiljem. Mitmed tema põhimõtted aga leidsid rakendamist hilisemates arvutites.

Analüütilise masina loengu põhjal avaldati artikkel, mille C. Babbage palus inglise keelde tõlkida [Ada Lovelace'il](#), kes oli luuletaja lord Byroni tütar. Lisaks tõlkimisele lisas A. Lovelace artiklile ka kommentaare, mille hulgas olid ka juhised, kuidas selle masina abil leida Bernoulli arve. Hiljem on neid juhiseid hakatud pidama ajaloo esimeseks programmiks ja Ada Lovelace'i esimeseks programmeerijaks. Rohkem programme ta teadaolevalt ei kirjutanud ja ei saanud seda ainsatki masinal reaalselt testida. See, et esimene programmeerija oli naine, on vahel olnud inimestele üllatav. Tegelikult saab programmeerimisega muidugi tegeleda soost sõltumata.

SALADUSED

Hüppame nüüd ajas umbes sada aastat edasi. Vahepeal oli masina abil näiteks edukalt analüüsitud USA rahvaloenduse andmeid, kusjuures masina kasutamine andis tohutu ajavõidu. [Perfokaarte](#) oli kasutatud näiteks raamatupidamises. Teise maailmasõja eel ja ajal oli suur osa tegevusest seotud sõjandusega. Võib-olla isegi sõja lõpptulemust oluliselt mõjutanud saaga on seotud saksa šifreerimisaparaadi Enigma koodi lahtimurdmisega. (Sellest on ka juttu filmides, nt [Imiteerimismäng \(The Imitation Game\)](#).) Nimelt vahetasid saksa staabid, allveelaevad jm omavahel sõnumeid Enigmaga šifreeritult. Selle koodi lahtimurdmisega tegeles spetsiaalne töörühm Londoni lähistel Bletchley Parkis. Töörühmal dešifreerimine õnnestuski ja nende töö andis ka impulsi inglise elektronarvuti Colossus loomiseks. Sakslaste sõnumitest arusaamine võimaldas oma tegevust paremini planeerida ja kui see polnudki põhiline põhjus, miks sõda sedapidi lõppes, siis olulise panuse see kindlasti andis. [Bletchley Parki](#) saab ka külastada.

Üks Bletchley Parki töörühma liidreid oli kahtlemata värvikas isiksus Alan Turing. Ühelt poolt oli tegemist kindlasti äärmiselt andeka matemaatikuga ja informaatikuga. Tema tööd on arvutiteaduses fundamentaalse tähtsusega: [Turingi masin](#) ja [Turingi test](#) kannavad lausa tema nime.

Teiselt poolt oli tegu näiteks innustunud pikamaajooksjaga, kes võis Bletchley Parkist üle 60 km kaugusele Londonisse koosolekule joosta. Tema maratonijooksu rekord oli üsna arvestataval tasemel. Kuna ta kannatas heinapalaviku all, siis teatud perioodidel sõitis ta jalgrattaga tööle, gaasimask peas. Isiklik elu oli A. Turingil traagiline. 1952. aastal mõisteti ta süüdi ebasüüdsuse paragrahvi alusel, mille alla homoseksuaalsus tol ajal käis. 1954. aastal leiti Alan Turing tsüaniidimürgituse tagajärjel surnuna. Käeulatuses oli pooleldisöõdud õun ... Kuninganna Elisabeth II tühistas süüdimõistva otsuse 2013. aastal.

PROGRAMMEERIMISKEELED

Räägime nüüd ka natuke sellest, kuidas aegade jooksul arvutile oma soove on teada antud. Juba mehaanilistest masinatest peale on olulisel kohal olnud suhteliselt kahevalentne lähenemine nii programmide kui andmete osas. Nii on näiteks perfokaardil või perfolindil mingis konkreetses kohas auk või seda ei ole, mingi lamp põleb või ei põle, mingis pesas on midagi või pole. Arvude kujul on seda mõistlik kirja panna vaid kahe numbri, 1 ja 0 abil.

Programmeerimiskeeled võib (mõnevõrra tinglikult) jaotada põlvkondadesse. Nii saab eristada näiteks:

1. põlvkond - masinkood
2. põlvkond - assemblerkeeled
3. põlvkond - kõrgtasemekeeled

Masinkoodis programmid koosnevad tinglikult ainult ühtedest ja nullidest, näiteks

00000000001000100011000000100000

võib tähendada "liita aadressidel 1 ja 2 olevad arvud ning salvestada resultaata aadressile 6". Sellist programmi on inimesel raske lugeda ja kirjutada, masinale on see aga hästi "seeditav". Assemblerkeeles programmeerimiseks on juba inimesele natuke paremini mõistetav, näiteks

add \$1, \$2, \$3

Siiski loetakse assemblerkeeles kirjutatud programme madalatasealiseks. See tähendab siinkohal arvutilähedust - assemblerkeelt valdavad programmeerijad on ise reeglina just kõrgetasealised. :-)

Kõrgtaseme keeles programmi suudab ettevalmistunud inimene hästi kirjutada ja lugeda, masina jaoks tuleb seda aga transleerida. (Transleerimine ongi konkreetne termin, kuigi olemuselt see muidugi tõlkimist tähendabki.) Järgmiste põlvkondade programmeerimiskeeled peaksid olema veelgi rohkem programmeerijasõbralikumad ja tehisintellekti abil ülesandeid pigem ülesande (inimkeelse?!) kirjelduse kui juba etteantud lahendussammude järgi lahendama.

1.7 Esimese nädala kontrollülesanded

Esimesel nädalal tuleb esitada nelja kohustusliku ülesande lahendused. Neljanda ülesande puhul on võimalik valida lahendamiseks kas 1.4a või 1.4b (või mõlemad). Lahendused tuleb esitada *Moodle* 'is, kus need kontrollitakse automaatselt. *Moodle*'is on ka nädalalõputest 10 küsimusega, millest tuleb vähemalt 9 õigesti vastata.

Kontrollülesanne 1.1. Tervitus (teemad 1.1. Algoritm. 1.2. Programm)

Koostada programm, mis väljastaks ekraanile teksti *Tere, maailm!* täpselt sellisel kujul - koma ja hääumärgiga.

Näide programmi tööst:

```
>>> %Run yl1.1.py
      Tere, maailm!
>>> |
```

Automaatkontroll on nii korraldatud, et tõesti pole vähimadki kõrvalekalded tekstis lubatud. Nii võib teil programm täiesti töötada Thonnys ilma igasuguste veateadeteta, aga kui väljastatakse näiteks *tere, maailm*, siis automaatkontroll seda õigeks ei loe.

Kontrollülesanne 1.2. Aasta tegija (teemad 1.3. Andmetüübid. 1.4. Muutujad)

Koostada programm, mille

- 1. real luuakse muutuja nimega `aasta` ning antakse sellele väärtuseks 2017 (arvuna);
- 2. real luuakse muutuja nimega `muld` ning antakse sellele väärtuseks "leedemuld" (sõnena);
- 3. real luuakse muutuja nimega `lause_keskosa` ning antakse sellele väärtuseks ". aasta mullaks on " (sõnena);
- 4. real luuakse muutuja nimega `lause`, mille väärtuse saamiseks ühendatakse üheks sõnaks muutujad `aasta`, `lause_keskosa` ja `muld` (vajadusel tuleb kasutada funktsiooni, mis teisendab arvu sõneks);
- 5. real väljastatakse muutuja `lause` väärtus ekraanile.

Kuigi tegelikult pannakse lause lõppu punkt, siis siin ärge pange. (Automaatkontroll isegi annab punkti või mõne muu üleliigse osa eest veateate.)

Näide programmi tööst:

```
>>> %Run yl1.2.py  
2017. aasta mullaks on leedemuld  
>>>
```

Aasta mulla kohta võib lugeda [siin](#).

Kui olete juba hulk aega proovinud ülesannet iseseisvalt lahendada ja see ikka ei õnnestu, siis võib-olla saate abi [murelahendajalt](#). Püütud on tüüpilisemaid probleemseid kohti selgitada ja anda vihjeid.

Kontrollülesanne 1.3. Astendamine (teema 1.5. Sisend kasutajalt: `input()`)

Koostada programm, mis

- küsib kasutajalt astme aluse (täisarv) ja astendaja (täisarv) (just selles järjekorras);
- arvutab ja väljastab ekraanile astme.

Näiteks kui kasutaja sisestab vastustena 2 ja 4, siis väljastatakse 16, sest $2^4 = 16$. Astme alus on siin 2, astendaja 4 ja aste 16.

Näide programmi tööst:

```
>>> %Run yl1.3.py  
Sisestage astme alus: 2  
Sisestage astendaja: 4  
16  
>>> |
```

Kui olete juba hulk aega proovinud ülesannet iseseisvalt lahendada ja see ikka ei õnnestu, siis võib-olla saate abi [murelahendajalt](#). Püütud on tüüpilisemaid probleemseid kohti selgitada ja anda vihjeid.

Järgmisest kahest ülesandest (1.4a ja 1.4b) tuleb lahendada vähemalt üks.

Kontrollülesanne 1.4a. Nädala ajakulu (terve 1. nädal)

Ülikoolides arvestatakse ühe ainepunkti ajakuluks 26 tundi. Näiteks kolme ainepunkti kursusel on ajakuluks $3 * 26 = 78$ tundi. Kui see jaotada 10 nädala peale, on ühe nädala eeldatav ajakulu $78 / 10 = 7,8$.

Koostada programm, mis

- küsib kasutajalt just sellises järjekorras ainepunktide arvu (täisarvu) ja nädalate arvu (täisarvu);

- arvutab ja väljastab ekraanile ühe nädala eeldatava ajakulu, mis on ümardatud täisarvuni.

Täisarvuks ümardamisel saab kasutada funktsiooni *round*, näiteks `round(ajakulu)` väärtuseks on 8, kui kasutaja on sisestanud 3 ja 10. Funktsioon *int* ei ole siin sobiv, kuna `int(7.8)` on hoopis 7.

Näide programmi tööst:

```
>>> %Run yl1.4a.py
Sisestage ainepunktide arv: 3
Sisestage nädalate arv: 10
8
>>> |
```

Kui olete juba hulk aega proovinud ülesannet iseseisvalt lahendada ja see ikka ei õnnestu, siis võib-olla saate abi [murelahendajalt](#). Püütud on tüüpilisemaid probleemseid kohti selgitada ja anda vihjeid.

Kontrollülesanne 1.4b. Kiiruseületamise trahv (terve 1. nädal)

Liiklusseaduse järgi määratakse suurima lubatud sõidukiiruse ületamise korral hoiatustrahv, mille suurus eurodes saadakse lubatud sõidukiirust ületanud kilomeetrite arvu korrutamisel arvuga 3. Hoiatustrahvi maksimaalmäär on 190 eurot.

Trahvimääradest on juttu [siin](#).

Koostada programm, mis

- küsib kasutajalt just sellises järjekorras nime (sõne), lubatud kiiruse (täisarvu) ja tegeliku kiiruse (täisarvu);
- arvutab trahvi vastavalt kasutajalt saadud andmetele ja ülaltoodud reeglitele;
- väljastab nime ja trahvi ekraanile.

Näiteks kui kasutaja sisestab vastustena *Jürgen, 50* ja *60*, siis väljastatakse *Jürgen, kiiruse ületamise eest on teie trahv 30 eurot*.

Selleks, et trahvi maksimaalmääraga arvestada saab kasutada funktsiooni *min*, millega saab leida mitmest väärtusest minimaalse. Näiteks `min(190, esialgne_tulemus)` väärtuseks on 190, kui muutuja `esialgne_tulemus` väärtus, mis on varem defineeritud, on suurem kui 190 (või võrdne 190).

Näiteks kui kasutaja sisestab vastustena *Sal-Saller, 50* ja *172*, siis väljastatakse *Sal-Saller, kiiruse ületamise eest on teie trahv 190 eurot*. Kiirust ületati 122 võrra ja esialgne arvutus näitab trahviks 366 eurot, aga see on üle maksimaalmäära. (Näide on inspireeritud laulust "Lähme sõidame", kus on "viiekümneses tsoonis meil sada seitsekümmend kaks on sees".)

Näide programmi tööst:

```
>>> %Run yl1.4b.py  
  
Sisestage oma nimi: Jürgen  
Sisestage lubatud kiirus (km/h): 60  
Sisestage tegelik kiirus (km/h): 80  
Jürgen, kiiruse ületamise eest on teie trahv 60 eurot.  
  
>>> |
```

Eeldame, et kasutaja sisestab tegeliku kiiruse, mis on lubatud kiirusest suurem. Kui sisestatakse lubatud kiirusest väiksem tegelik kiirus, siis las programm annabki trahviks negatiivse arvu. Päriselus see küll nii just pole...

Kui olete juba hulk aega proovinud ülesannet iseseisvalt lahendada ja see ikka ei õnnestu, siis võib-olla saate abi [murelahendajalt](#). Püütud on tüüpilisemaid probleemseid kohti selgitada ja anda vihjeid.

1.8 Alkeemia ehk Maagia alustalad 101

Selle loo on kirjutanud Ander Peedumäe, kes oli 2016. aastal Tartu Ülikooli informaatika eriala 1.

kursuse tudeng.

Vana alkeemik seisis messingist katla ees ning mõõtis pilguga õpilasi, kes tunni algusest väljagi ei teinud ning omavahel jutustasid. Ta kõhatas valjult ning tõmbas endale kogu auditooriumi tähelepanu. "Järgneva tunni jooksul saate te teada rohkem, kui enda kaaslastega nädal aega järjest vesteldes. Kuidas karjeid pudelisse püüda, paljastada valesid ja hämmastada kõrvalseisjaid? Teretulemast loengusse Alkeemia I ehk Maagia alustalad."

Lektor lõi saapakannaga vastu maad ning raske apteekrikapp sööstis klirisedes toa nurgast saali keskele. Uksed avanesid ja riiulid moodustasid ükshaaval välja libisedes trepi, mille igal astmel seisis erineva suuruse, kuju ja värviga pudelid ning potsikud, igaühel erisugune silt. „Te olete siin, sest teid huvitab keeruliste ja töömahukate probleemide lahendamine high, verd ja pisaraid valamata ning parim viis selleks on hea maagiline retsept.“

Alkeemik võttis riiulist pisikese ümara tindipurgi. „Alustame põhilisest. Kõik pudelid, purgid, potsikud ja astjad, mida siin näete, on Muutujad. Kas keegi tahab pakkuda, miks me neid nii nimetame?“

Saalis valitses vaikus. Tagumises reas kõhatati. Kolmandast reast kostus väsinud tudengi vaikset norskamist. Pingiridade keskelt tõusis kõhklev käsi. „Sest nad muudavad kuju?“

„Jah, võib ka nii öelda, kuigi kuju muutmine on kõrvalmõju. Tegelikult muutub anumate sisu ning anum reageerib sellele. Kuidas oleks, kui keegi mõtleb välja, mille alusel saame erinevaid pudeleid liigitada?“

Pärast paari vaikset hetke tõusis sama käsi. „Kuju järgi?“

Lektor sikutas enda habet ning vaatas ülejäänud õpilastele pahakspanevalt otsa. „Õige! Aga mitte ainult. Kuju on põhiline, kuid targematele võib tulla pähe heita ka pilk sildile. Võtame näiteks selle ümmarguse tindipudeli.“ Ta tõstis selle enda silme ette. „Säärane kuju reedab selle sisu – tegemist on Sõnega, daamid ja härrad! Sõnade kett, mis ootab purgist välja laskmist. Ja see kindel sõne on püütud Muutujasse nimega X.“

Mees muigas ning ronis kuulajate vahele kolmandasse ritta, kus üks väsinud noormees lõunauinakut nautis. Õppejõud eemaldas korgi, pööras pudeliava usina õppuri kõrva suunas ning sosistas midagi, mis kõlas nagu: „Väljasta...“

„TERE MAAILM!“ Pudelist plahvatas röögatus, mis äratas nii unise õpilase kui ka (suure tõenäosusega) kõik surnud terves linnaosas. Auditoorium täitus kohkunud saginaga ning alkeemik navigeeris end tagasi katla kõrvale, asetas pudeli oma kohale tagasi ja pöördus publiku poole. „Natuke tähelepanu ei jookse kunagi mööda külgi maha. Kas ma tohiksin uuesti paluda teie oma?“

Lektor avas apteekrikapi alumise sahtli ning tõstis nähtavale tolmuise pappkarbi. „Ärge mõistke mind valesti, karjete pudelisse villimine on meelelahutuslik kullakaevandus, kuid see on vaid pisike säde selles lõkkes.“ Ta avas karbi ja noppis sealt ühe peenikese violetse pudeli. „See siin on midagi märksa

kummalisemat. Üksinda üsnagi kasutu ning näitab enda tõelist palet alles korralikus katlatäies. Jah, mu saatusekaaslased, see on Tõeväärtus.“

Alkeemik viipas katla poole ning see lohises otse tema jalge ette. „Tõeväärtused on binaarsete omadustega. Nad on kas tõde või väär, mitte midagi vahepealset. Minu ees olevas katlas on segu, mis loeb mõtteid ja suudab tagastada tõeväärtuse „jah või ei“ tüüpi küsimuste puhul. Säärase maagia kasutamine ei ole otseselt seadusevastane, kuid loitsualuse teadmata on see rangelt karistatav. Viimase aasta tudengid kasutavad sarnast katlatäit, et mängida Tõde või Tegu.“

Sõrmenipsu peale lahvatas katla all roheline leek ja võlujook hakkas mulisema. Õppejõud asetas käed mõlemale poole katla äärtele ning vaatas ootavalt läbi aurupilve. „Kes tahab minult midagi küsida?“ Noormees, kes oli mõned minutid tagasi sunniviisiliselt ärganud, haaras hetkegi kaotamata võimalusest: „Kas on tõsi, mida te igal aastal rebastele räägite? See, kuidas te noorena mägedes kolme lohega võitlesite.“

Alkeemiku näos ei liikunud ükski lihas. Katel lõpetas auramise ning selle pinnale kerkis violetse vedelikuga pudel. Ta eemaldas klaasist korgi, lausus „Väljasta.“ ning tõstis selle kõrgele. „Tõde.“ Mehe näole ilmus võidukas muie.