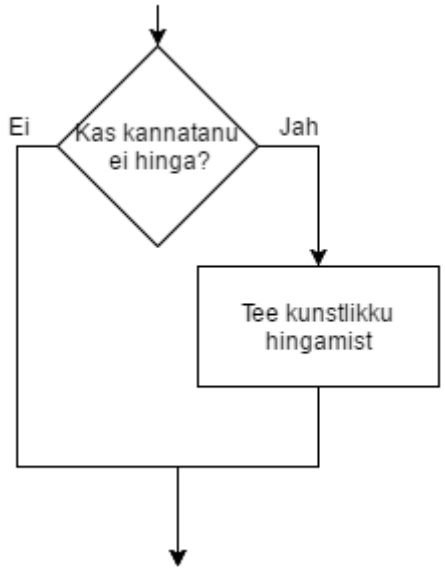
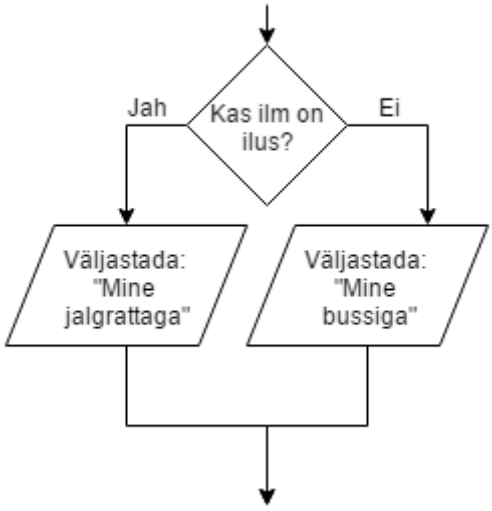
2.1 Tingimuslause

SISSEJUHATUS

Eelmisel nädalal vaatasime algoritme, kus mitmes kohas olenes järgmine samm teatud tingimuse täidetusest. Näiteks õnnetuse puhul: "Kui kannatanu ei hinga, tee kunstlikku hingamist." Sellisel juhul on kirjeldatud tegevus selliseks juhuks, kui tingimus on täidetud ja kui pole täidetud, siis jäetakse see samm lihtsalt vahele. Siinkohal on veel huvitav see, et tingimus on eituse kaudu esitatud - "kannatanu ei hinga".



Tihti aga on tegevused nii selleks juhuks, kui tingimus on täidetud, kui ka selleks, kui tingimus pole täidetud.



Programmides realiseeritakse selliseid valikuid **tingimuslausete** (ehk **valikulausete** ehk ***if*-lausete**) abil.

SÕNEDE VÕRDLEMINE TINGIMUSES

Tahame näiteks teha programmi, mis küsib kasutajalt sisenemiseks **PIN-koodi**ja kontrollib selle õigsust. Programm võiks

- õige parooli sisestamisel öelda: “Sisenesid pangaautomaati!”
- vale parooli sisestamisel aga hoopis: “Vale parool! Enesehävitusrežiim aktiveeritud: 3 ... 2 ... 1 ....”.

Programmi alguses tuleb kasutajalt PIN-kood küsida, seda saab teha järgmise lausega:

```
sisestatud_pin = input("Sisesta PIN-kood: ")
```

Kui kasutada programmeerimiskeskkonda Thonny, siis ootab `input` kasutaja sisestust alumisel paneelil (*Shell*).

Nüüd on vaja kontrollida, kas parool on õige või ei ole. Selleks saab kasutada **tingimuslauset**:

```
if sisestatud_pin == "1234":
    print("Sisenesid pangaautomaati!")
```

Mis see **tingimuslause** (ka **valikulause** või ***if*-lause**) siis on? Tingimuslauses on kesksel kohal tingimus, mille alusel otsustatakse, kuidas programm edasi töötab. Tingimus on esitatud tõeväärtuse tüüpi avaldisena, mille väärtus on tõene (`True`) või väär (`False`) - tingimus on täidetud või ei ole täidetud. Kõige sagedamini on tingimus esitatud teatud võrdlemisena. Võrrelda võib näiteks muutuja väärtust ja sõne nagu eelnevas näites. Oluline on tähele panna, et muutuja väärtus on samuti sõnetüüpi. Topeltvõrdusmärgiga (`==`) avaldis on väärtusega `True` , kui mõlemad pooled on võrdsed ja väärtusega `False` , kui ei ole võrdsed.

Võrrelda saab ka näiteks muutuja väärtust ja arvu. Näiteks avaldis `a >= 1` on tõene, kui a on suurem kui üks või ühega võrdne. Samuti võib võrrelda omavahel kahe muutuja väärtust. Tegelikult võib tingimusena olla kirjas ka lihtsalt `True` (või `False`), aga siis ei toimu tegelikult mingit valikut. Avaldis `5 == 4` on ka näiteks alati väär. Tingimuslausel on sisuline mõte siis, kui tingimusavaldis võib omandada erinevaid tõeväärtusi.

Kindlasti ei tohi unustada koolonit (`:`) `if` -rea lõpust. Paneme tähele, et `print` on taandatud võrreldes eelmise reaga paremale. Seda saab teha kas vajutades TAB-klahvi või 4 korda tühikut. Selliselt tuleb käituda, et `print` käsk käiks ikkagi `if` -osa juurde ja sellist tegevust nimetatakse treppimiseks. Head programmeerimiskeskkonnad (näiteks Thonny) toetavad treppimist sellega, et pärast eelmise rea koolonit viivad kursori juba sobivalt taandesse.

Ülaltoodud programm juba kuidagi töötab, aga mitte siis, kui parool on vale. Tingimuslausel võib olla `else`-osa, millega määratakse, mis juhtub siis, kui tingimus ei ole täidetud (vastava avaldise väärtus on `False`).

Hetkel lisame järgmise lõigu:

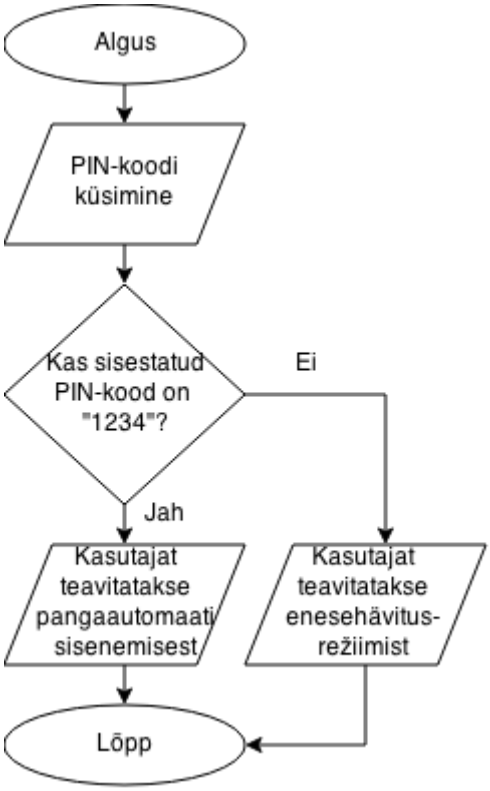
```
else:
    print("Vale parool! Enesehävitusrežiim aktiveeritud: 3 ... 2 ...
1 ....")
```

Kokku sai siis selline kood:

```
sisestatud_pin = input("Sisesta PIN-kood: ")
if sisestatud_pin == "1234":
    print("Sisenesid pangaautomaati!")
else:
    print("Vale parool! Enesehävitusrežiim aktiveeritud: 3 ... 2 ...
1 ....")
```

Palun katsetage seda programmi Pythoniga. Kui eelnevas näites midagi segaseks jäi, siis vaadake kokkuvõtvat videot programmi koostamise kohta:

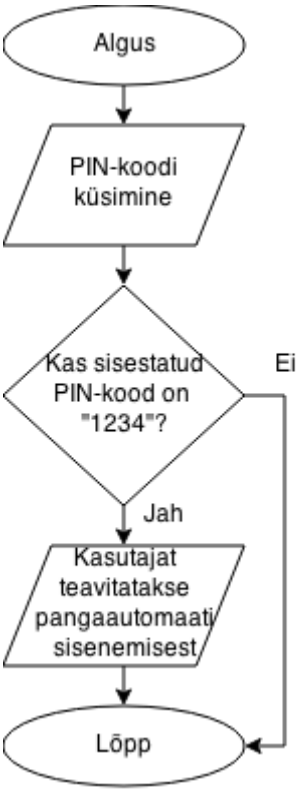
Antud programmi plokkskeem näeb välja selline:



Sellist ühes või teises suunas otsuste tegemist ja liikumist vastavalt tingimuse kontrollile nimetatakse programmeerimises ka **hargnemiseks**. Programm otsustab, millist haru pidi

edasi minna. Kui tingimus on täidetud, siis minnakse edasi ühte haru pidi, kui ei ole, siis teist haru pidi. Rõhutada tuleb, et tingimuslauses on harusid alati kaks, isegi kui `else`-osa "näha" pole. Kui `else` puudub, siis ei tehta väära tingimusavaldise korral mitte midagi, sama hästi võiks `else`-osa kirjutada ja tühjaks jätta.

Kui eelnevas programmis `else`-osa poleks, siis plokkskeem oleks selline:



Ülesanne

Meenutame, et materjalides on enesekontrolliülesanded, mille lahendamise tulemusi ei salvestata. Võite julgesti ka valesti vastata, aga püüdke ikka õigesti.

Mida väljastatakse ekraanile?

```
vastus = "0" + "0" + "7"
if vastus == "007":
    print("On võrdsed!")
else:
    print("Ei ole võrdsed!")
```

Vali 7

Vali 007

Vali On võrdsed!

Vali Ei ole võrdsed!

Vali Veateade

NB! Sõnede võrdlemisel loetakse suurtähed ja väiketähed erinevateks. Näiteks `"A"` ja `"a"` ei ole võrdsed. Mõnede ülesannete puhul me tahame nad siiski võrdseks lugeda. Näiteks saab sisendi puhul kõik tähed väikeseks teisendada nii:

```
sisestatud_väike = input().lower()
```

Analoogiliselt saab kõik tähed suureks teisendada:

```
sisestatud_suur = input().upper()
```

ARVUDE VÕRDLEMINE

Eelmises näites kontrollisime koodi võrdsust sõnega `"1234"`. Oluline on siinkohal just see, et jutumärgid teevad sellest muidu arvuna näivast asjast sõne. Loomulik on aga paljusid asju käsitleda ka arvuna.

Tingimusi saab (muuhulgas) moodustada järgmiste märkidega:

- `<`, `>`, `<=`, `>=` võrratuste kehtivuse kontrollimiseks;

- `==` võrdsuse kontrollimiseks;
- `!=` mittevõrdsuse kontrollimiseks.

Kasutame nüüd arvude võrdlemisel võrratust. Teeme programmi, mis küsib kasutajalt vanust ja kui ta on noorem kui 14, siis öeldakse, et ta on selle mängu mängimiseks liiga noor. Täisarvu sisestamiseks peame sisendiks saadud sõne täisarvuks teisendama. Seda saab teha funktsiooniga `int`. Näiteks: `arv = int(input())`.

```
print("Kui vana oled?")
vanus = int(input())

if vanus < 14:
    print("Oled natukene noor edasipääsuks.")
else:
    print("Sisenesid võlumaailma.")
```

Ülesanne

Mida väljastatakse ekraanile?

```
vastus = 2 + 2 * 3
if vastus == 12:
    print("On võrdsed!")
else:
    print("Ei ole võrdsed!")
```

Vali 8

Vali 12

Vali On võrdsed!

Vali Ei ole võrdsed!

Vali Veateade

Ülesandeid huvilisematele

Siin on nüüd mõned ülesanded, mille võivad need, kes end ebakindlamalt tunnevad, vahele jätta. Kuid soovitame soojalt siiski proovida!

- Kirjutada PIN-koodi küsimise programm nii ümber, et sisestatud koodi võrreldakse arvuga `1234` (just arvuga, mitte sõnega). Programm võiks ka öelda, kas sisestatud vale kood oli "liiga suur" või "liiga väike".
- Katsetada, kas ka sõnesid saab võrrelda märkide `<`, `<=`, `>=` ja `>` abil. Kuidas tulemust kommenteerida?

2.2 Tingimuslause tingimuslause sees

Tingimuslause tingimuslause sees

Valikulausete puhul on kesksel kohal tingimus, mille kehtimisest sõltub, millise haruga edasi minnakse. Tingimus esitatakse tõeväärtustüüpi avaldisena, näiteks `sisestatud_pin == "1234"`. Avaldis võib olla ka märksa keerulisem. Mõningaid selliseid vaatame käesoleva nädala järgmistes materjalides.

Praegu aga jätkame varasemat programmi, kus küsitakse PIN-koodi ja õige koodi korral "sisenetakse" pangaautomaati. Mis aga saab edasi? Tundub, et programm peab edasi ka valikutega minema. Teeme nüüd nii, et pangaautomaati sisenedes näidatakse, mitu eurot pangakontol on, ning küsitakse, mitu eurot soovitakse välja võtta. Kui kasutaja soovib välja võtta rohkem raha, kui on pangakontol, siis teavitatakse kasutajat, et see pole võimalik.

Eelnevalt oli meil olemas pangaautomaati sisenemise programm:

```
sisestatud_pin = input("Sisesta PIN-kood: ")
if sisestatud_pin == "1234":
    print("Sisenesid pangaautomaati!")
else:
    print("Vale parool! Enesehävitusrežiim aktiveeritud: 3 ... 2 ...
1 ....")
```

Esiteks võiks programmi alguses kasutusele võtta muutuja, mille väärtus näitab, kui palju pangakontol raha on. Näiteks teeme nii, et algselt on kasutaja pangakontol 100 eurot:

```
kontoseis = 100
```

Nüüd teeme programmiosa, mis töötab siis, kui kasutaja on automaati sisse saanud. Selles osas väljastatakse ekraanile pangakontol olev summa ning küsitakse, mitu eurot soovitakse välja võtta. Küsitud väljavõetava raha teisendame seekord koheselt täisarvutüüpi (`int`) väärtuseks:

```
print("Sisenesid pangaautomaati! Pangakontol on " + str(kontoseis) +
" eurot.")
print("Sisesta, mitu eurot soovid välja võtta:")
soovitud_raha = int(input())
```

Seejärel oleks vaja kontrollida, et väljavõetav rahasumma oleks väiksem või võrdne kontol olevast. Kui nii on, siis võetakse vastav summa kontolt maha. Kui ei ole, siis väljastatakse vastav veateade:

```
if soovitud_raha <= kontoseis:
    kontoseis = kontoseis - soovitud_raha
    print("Välja võetud: " + str(soovitud_raha) + " eurot. Ärge
unustage raha masinast ära võtta!")
else:
    print("Kontol ei ole nii palju raha!")
```

Juba eelmisel nädalal oli näide, muutujale anti väärtuse tehtega, milles kasutame sedasama muutujat ennast (siin `kontoseis = kontoseis - soovitud_raha`). Muutuja uue väärtuse arvutamine vana põhjal on isegi nii levinud, et kasutusel on tehtemärgid (nt `+=`, `-=`, `*=`, `/=`), mis võimaldavad seda lühemalt kirja panna.

Nii saaks asendada vastava rea eeltoodud programmis järgmisega:

```
kontoseis -= soovitud_raha
```

Lähme nüüd tagasi pangaautomaadi programmi juurde. Kõige lõpuks võime veel ekraanile väljastada, kui palju raha pangakontole jäi.

```
print("Pangakontol on nüüd: " + str(kontoseis) + " eurot.")
```

Kui nüüd lisada need read algse programmi `if`-osa sisse, siis saame kokkuvõttes järgneva programmi. Märgiga `#` tähistatakse programmis kommentaare, mis on inimeste jaoks ja mida Python ei arvesta. Kõik, mis samal real `#` märgile järgneb, on kommentaar.

```
kontoseis = 100

sisestatud_pin = input("Sisesta PIN-kood: ")
if sisestatud_pin == "1234":
    print("Sisenesid pangaautomaati! Pangakontol on " +
str(kontoseis) + " eurot.")
    print("Sisesta, mitu eurot soovid välja võtta:")
    soovitud_raha = int(input())
    if soovitud_raha <= kontoseis:
        kontoseis = kontoseis - soovitud_raha #kontoseisu väärtus
väheneb
        print("Raha välja võetud: " + str(soovitud_raha) + "
eurot.")
    else:
        print("Kontol ei ole nii palju raha!")
    print("Pangakontol on järgi: " + str(kontoseis) + " eurot.")
else:
    print("Vale parool! Enesehävitusrežiim aktiveeritud: 3 ... 2 ...
1 ....")
```

Palun mõelge selle programmi loogika rahulikult läbi ning seejärel katsetage selle tööd.

Ülesanne

Mida väljastatakse ekraanile?

```
a = 1
b = 2
if a > b:
    if a == 1:
        print(a)
    else:
        print(b)
else:
  print("Olen siin")
```

Vali 1

Vali a

Vali 2

Vali b

Vali Olen siin

Vali Veateade

2.3 Mitmeosaline tingimus. Loogilised tehted ja avaldised

TINGIMUS ON KEERULISEM

Eespool käsitlesime olukorda, kus mingi tingimuslause oli teise tingimuslause ühes harus. Nüüd aga vaatleme, kuidas saab kasutada keerulisemaid tingimusi. Tegemist on siis ikkagi ühe tingimusega, aga see koosneb mitmest osast. Võib ette kujutada mitmeid elulisi situatsioone, kus on vaja kontrollida mitut tingimust korraga. Olgu näiteks selline seif, mis avaneb ainult siis, kui kaks erinevat koodi õigesti sisestatakse. See võimaldaks näiteks kindlalt teada, et seifi avamisel peavad mõlemad partnerid korraga kohal olema.

Tingimusi saab kombineerida loogiliste tehete abil. Üks nendest on näiteks `and`, mis ongi "ja" tähenduses. Tehtega `and` seotud avaldis on tõene siis ja ainult siis, kui mõlemad avaldised, millega tehe tehakse, on tõesed. Järgnevas programmis ongi seda ära kasutatud.

```
kood1 = "1234"
kood2 = "0000"
print("Sisesta 1. salakood:")
pakutud_kood1 = input()
print("Sisesta 2. salakood:")
pakutud_kood2 = input()

if pakutud_kood1 == kood1 and pakutud_kood2 == kood2:
    print("Seif avaneb!")
else:
    print("Salakoodidega on kehvasti. Tõstke käed üles!")
```

Katsetage seda programmi erinevate sisenditega. Näiteks, kui

- mõlemad koodid on õiged;
- 1. kood on õige, 2. kood vale;
- 1. kood on vale, 2. kood õige;
- mõlemad on valed.

Kas tulemus vastas teie ootustele?

Loogilisi tehteid on veel, näiteks `or` ja `not`. Sõna "or" tähendab tõlkes "või". Asendage valikulauses tehe `and` tehtega `or`:

```
if pakutud_kood1 == kood1 or pakutud_kood2 == kood2:
```

Katsetage nüüd programmi tööd erinevate sisenditega. Jälle, kui

- mõlemad koodid on õiged;
- 1. kood on õige, 2. kood vale;
- 1. kood on vale, 2. kood õige;
- mõlemad on valed.

Püüdke `or`-tehte olemust nüüd selgitada. Kuidas on seos tavakeele sõnaga "või"?

Ülesanne

Milline järgnevatest tavakeelsetest seletustest vastab kõige paremini järgmisele avaldisele?

```
on_soe or on_vihmane
```

Vali Ilm on soe või vihmane, aga mitte mõlemat korraga

Vali Ilm on soe või vihmane või mõlemat korraga

Vali Ilm on soe ja vihmane.

Vali Ilm on soe, aga ei ole vihmane

Eelmise näite puhul olid välja pakutud mõned variandid, mida katsetamisel kasutada. Need polnud valitud suvaliselt, vaid püüdsid hõlmata kõiki põhimõtteliselt erinevaid variante. Testimine on programmeerimises väga oluline. Vähegi suurema programmi puhul on mõistlike testide koostamine omaette suur töö. IT-firmades on lausa testija ametikohad ja vastavad osakonnad.

LOOGILISED TEHTED JA AVALDISED

Meil on kasutada tõeväärtused `True` ja `False`, mis sageli otseselt "nähtavad" polegi ja programmis paistavad näiteks hoopis mingite võrdlemistena.

`pakutud_kood1 == "1234"`
`vanus < 14`
`soovitud_raha <= kontoseis`

Olenevalt muutujate konkreetsetest väärtustest on nende avaldiste väärtusteks `True` või `False`. Loogiliste tehete abil saame keerulisemaid avaldisi moodustada. Tehete `and` ja `or` puhul on meil vaja kahte operandi, millega tehe toimub.

Selliseid tehteid nimetatakse kahekohalisteks ehk binaarseteks. Tehe `not` töötab ühe operandiga (on ühekohaline ehk unaarne). Tehe `not` "pöörab" tõeväärtuse vastupidiseks. Näiteks avaldise

```
not 1 == 1
```

väärtus on `False`, sest `1 == 1` on `True`.

Võib öelda, et `and`, `or` ja `not` tehete kasutamisel on tulemused järgnevad:

- `väide1 and väide2` - tõene ainult siis, kui mõlemad väited on tõesed
- `väide1 or väide2` - tõene siis, kui **vähemalt üks** väidetest on tõene
- `not väide1` - tõene ainult siis, kui `väide1` on väär.

Tehted ja avaldised on meile põhimõtteliselt tuttavad juba koolimatemaatikast - aritmeetikast ja algebrast. Loogilised tehted ja avaldised vajavad veidi harjumist. Ärge väga muretsege, kui esimese hooga kõik veel selgeks ei saa. Põhimõtteliselt on tegemist siiski loogilise asjaga. :-)

Loogilised avaldised muutuvad tingimusteks, kui neid selles rollis kasutatakse - näiteks tingimuslauses.

Vahel on tingimused keerulisemad ja programmi korraliku töö huvides tuleb need põhjalikult läbi mõelda. Olgu meil tahtmine teha salatit - kartuli- või makaronisalatit. Lihtsustatult oleks meil siis vaja kartuleid või makarone ja salatikastet.

Jätame praegu täpsustamata, millest see salatikaste tehtud on ja muid detaile ka. Tegelikult kipubki päriselu loogilistes avaldistes kirjeldamine päris keeruline olema. Programmide kirjutamisel aga vähemalt mingil tasemel seda teha tuleb.

Loogilise avaldise koostamiseks võtame kasutusele muutujad `kartul_olemas`, `makaron_olemas` ja `salatikaste_olemas`. Vastavalt siis sellele, kas vastav koostisosa on olemas või mitte, on selle muutuja väärtus `True` (on olemas) või `False` (ei ole olemas).

Püüame nüüd kirja panna, kas saame salatit teha või mitte. Siin on tegelikult võimalikud mitu mõttekäiku ja ka avaldis võib lõpuks tulla (näiliselt) erinev.

Üks võimalus oleks mõelda nii, et meil on vaja kahte koostisosa: n-ö nimiosa ja salatikastet. Nimiosas võib olla kartul või makaron - panemegi kirja `kartul_olemas or makaron_olemas`. Ja salatikaste - `(kartul_olemas or makaron_olemas) and salatikaste_olemas`. Sulud tähistavad siin (nagu ka aritmeetikas ja algebras) seda, et vastav tehe tuleb enne ära teha.

Järgmises näites on meil siis kartul olemas ja salatikaste, aga makarone pole. Kui meie mõttekäik on õige olnud, siis peaks ekraanile tulema `True`, sest tõesti saame salatit teha. Proovige!

```
kartul_olemas = True
makaron_olemas = False
salatikaste_olemas = True
print((kartul_olemas or makaron_olemas) and salatikaste_olemas)
```

Proovige nüüd programmi tööd muutujate `kartul_olemas`, `makaron_olemas` ja `salatikaste_olemas` erinevatel väärtustel. Enne kui käivitate, mõelge milline tulemus võiks tulla. Kas programm töötab vastavalt teie ootustele?

Eelnevalt oli juttu, et sulgude kasutamine suunab tehete järjekorda. Sulgudes olev tehakse varem. Mis aga juhtuks, kui näiteks avaldises `(a or b) and c` sulud ära jätaksime? Mis mõte neil üldse siin on? Kas siis nagunii ei tehtaks siin `or`-tehet enne, asub see ju eespool?

Tõepoolest on sulud siin olulised, sest kui neid poleks, siis tehtaks esimesena hoopis `and`-tehe. Nimelt on kokkulepe, et `and` on kõrgema prioriteediga kui `or`, analoogiliselt tavalise aritmeetikaga, kus ka korrutamine tehakse enne liitmist. Näiteks *2 + 3 * 4* on *14*, aga mitte *20*.

Ülesanne

Kuivõrd võib kartmata samaväärsust rikkuda sulud ära jätta avaldises

```
(kartul_olemas and salatikaste_olemas) or
(makaron_olemas and salatikaste_olemas)
```

- Vali Ei tohi ära jätta
- Vali Võib esimesed ära jätta, aga teisi mitte
- Vali Võib teised ära jätta, aga esimesi mitte
- Vali Võib mõlemad ära jätta

VIDEO

Järgmises videos on veel üks programmi näide, kus on keerulisem tingimus ja erinevad loogilised tehted (http://www.uttv.ee/naita?id=23879).

2.4 Mitmeharuline tingimuslause elif abil

Eespool juba vaatlesime olukordi, kus tingimuslause harudes oli omakorda tingimuslauseid. Siin vaatame, kuidas vahel saab selliseid olukordi lühemalt kirja panna.

Vaatame sellist näidet, kus programmile öeldakse punktisumma ja programm teatab, mis hinde see summa annab.

Ülikoolis pannakse tihti hindeid järgmise skeemi järgi:

Tulemus (%)	Hinne
>90 .. 100	A
>80 .. 90	B
>70 .. 80	C
>60 .. 70	D
>50 .. 60	E
<=50	F

Programm, mis saab sisendiks punktid ja kontrollib, kas selle eest saab hinde "A", oleks järgnev:

```
punktid = int(input("Sisesta punktide arv"))

if punktid > 90:
    print("Hinne A")
else:
    print("Ei ole hinne A")
```

Lisame nüüd kontrolli ka hindele "B":

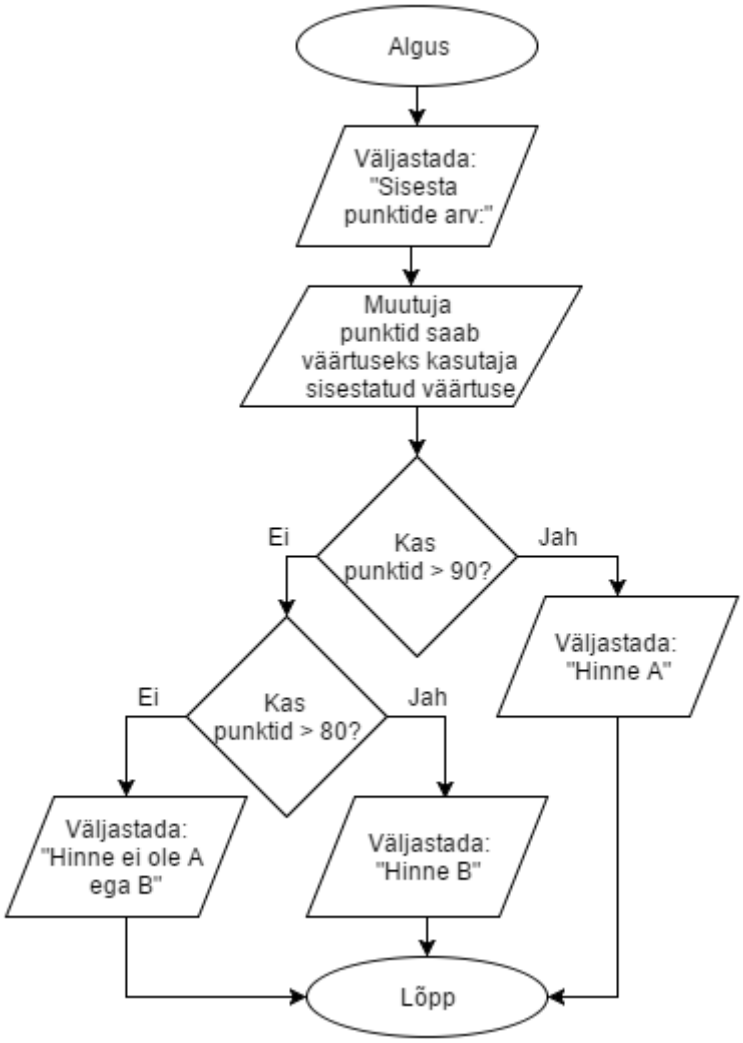
```
punktid = int(input("Sisesta punktide arv"))

if punktid > 90:
    print("Hinne A")
else:
    if punktid > 80:
        print("Hinne B")
    else:
        print("Hinne ei ole A ega B")
```

Kuna hinnetele vastavaid vahemikke on palju, siis kipub programm jooksma treppides liiga paremale. Mugavamaks programmeerimiseks võib kasutada ka `elif`-osa, mis on nii kirjapildi kui ka tähenduse poolest kombinatsioon `else`-ist ja talle järgnevast `if`-ist.

Näiteks eelnev programmilõik oleks siis selline:

```
punktid = int(input("Sisesta punktide arv"))

if punktid > 90:
    print("Hinne A")
elif punktid > 80:
    print("Hinne B")
else:
    print("Hinne ei ole A ega B")
```

Plokkskeem on siis hetkel järgmine.



Proovige lõpetada programm iseseisvalt.

Ülesanne

Mida väljastatakse ekraanile?

```
arv = 6
if arv > 6:
print("Arv on liiga suur!")
elif arv < 6:
print("Arv on liiga väike!")
else:
print("Õige arv!")
```

Vali Arv on liiga suur!

Vali Arv on liiga väike!

Vali Õige arv!

Vali Veateade

2.5 Juhuslik arv

JUHUSLIKU ARVU GENEREERIMINE

Vahest läheb programmeerimises vaja, et arvuti oskaks genereerida juhuslikke arve. Näiteks proovime teha programmi, mis simuleeriks mündiviskamist ja saaks tulemuseks kulli või kirja.

Kuidas panna arvuti juhuslikult valima kahe valiku vahel?

Selles tulebki meile appi juhusliku arvu genereerimine. Funktsiooniga `randint(1,2)` saame võrdse tõenäosusega ühe arvu vahemikust 1-st 2-ni (mõlemad kaasaarvatud). Kuna sinna vahemikku jäävadki vaid kaks arvu: 1 ja 2, siis valitakse nende vahel. Kumb igal konkreetsel juhul tuleb, ei ole ette teada. Võrdse tõenäosuse puhul on mõlema juhu tõenäosus 50% (sageli öeldakse ka 0,5). Kui teha piisavalt palju katseid, siis umbes samapaljudel juhtudel tuleb tulemuseks 1 kui 2. Oluline on siin märkida, et väikestel katsearvudel võib erinevus olla täiesti märgatav. Näiteks vabalt võib juhtuda, et kümnest juhust on seitsmel juhul 1 ja kolmel juhul 2. Järgmisel nädalal asume juhuslikkust tsükliliselt "kontrollima".

Sageli on meil aga vaja rohkem variante kui kaks. Näiteks `randint(1,3)` annab kas arvu 1, 2 või 3. Igaühe tõenäosus on siis ligikaudu 33,3% (1/3).

Funktsiooni `randint()` saab aga kasutada ainult siis, kui esimesele reale kirjutada, et seda funktsiooni programmis kasutatakse:

```
from random import randint
```

See tähendab, et me impordime moodulist `random` funktsiooni `randint`. Seal moodulis on veel teisi huvitavaid funktsioone ka. Vajadusel saab importida ka kõiki neid korraga:

```
from random import *
```

Proovige aga nüüd järgmist programmi:

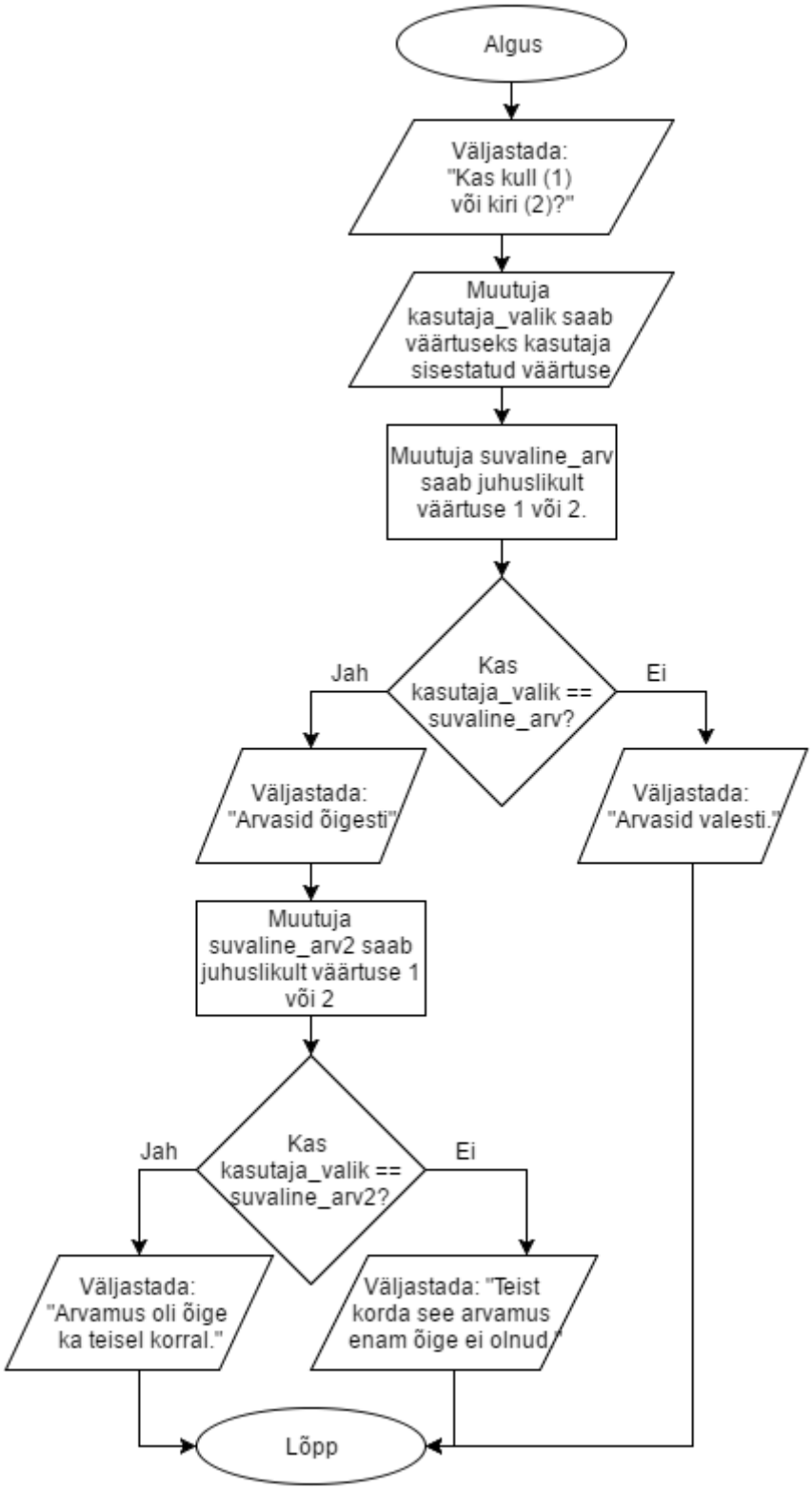
```
from random import randint

suvaline_arv = randint(1, 2)
if suvaline_arv == 1:
    arvuti_valik = "kull"
else:
    arvuti_valik = "kiri"

print("Mündiviskes võitis: " + arvuti_valik)
```

Kui nüüd küsida kasutajalt, kas ta valib kulli või kirja, siis saamegi kulli ja kirja viskamise mängu:

```
from random import randint

print("Kas kull (1) või kiri (2)?")
kasutaja_valik = int(input())
suvaline_arv = randint(1, 2)

if kasutaja_valik == suvaline_arv:
    print("Arvasid õigesti.")
else:
    print("Arvasid valesti.")
```

Teeme nüüd programmi ümber lisades selle tingimuslause ühte harusse teise tingimuslause.

```
from random import randint

print("Kas kull (1) või kiri (2)?")
kasutaja_valik = int(input())
suvaline_arv = randint(1, 2)

if kasutaja_valik == suvaline_arv:
    print("Arvasid õigesti.")
    suvaline_arv2 = randint(1, 2)
    if kasutaja_valik == suvaline_arv2:
        print("Arvamus oli õige ka teisel korral.")
    else:
        print("Teist korda see arvamus enam õige ei olnud.")
else:
    print("Arvasid valesti.")
```


Algus
Väljastada: "Kas kull (1) või kiri (2)?"
Muutuja kasutaja_valik saab väärtuseks kasutaja sisestatud väärtuse
Muutuja suvaline_arv saab juhuslikult väärtuse 1 või 2.
Kas kasutaja_valik == suvaline_arv?
Jah
Ei
Väljastada: "Arvasid õigesti"
Väljastada: "Arvasid valesti."
Muutuja suvaline_arv2 saab juhuslikult väärtuse 1 või 2
Kas kasutaja_valik == suvaline_arv2?
Jah
Ei
Väljastada: "Arvamus oli õige ka teisel korral."
Väljastada: "Teist korda see arvamus enam õige ei olnud."
Lõpp

Ülesanne

Oletame, et eelmises programmis valiks kasutaja kulli (1). Kui suur on tõenäosus, et ekraanile ilmuks

```
Arvasid õigesti.
Arvasid valesti.
```

Vali 0

Vali 0,25

Vali 0,5

Vali 1

Kes tahab sel teemal edasi mõtiskleda, võiksid läbi mõelda, mis oleks teisiti, kui

`suvaline_arv2 = randint(1, 2)` asemel oleks `suvaline_arv2 = randint(1, 3)`

või

`suvaline_arv2 = randint(1, 2)` asemel oleks `suvaline_arv1 = randint(1, 2)`

VIDEO

Kui materjalis juhuslikust arvust midagi selgusetuks jäi, siis vaadake kokkuvõtvat videot (http://www.uttv.ee/naita?id=23878).

2.6 Kilpkonnagraafika

Pythoni dokumentatsioon

Seni oleme saanud programmide töö tulemused tekstilisel kujul ekraanile väljastatuna. Samuti oleme sisendi andnud tekstilisel kujul. Reaalselt kasutatavad programmid on sageli hoopis graafilise kasutajaliidesega. Sellisteni jõuame mõne nädala pärast, aga natuke graafikat toome juba praegu sisse. Selles peatükis vaatame lähemalt Pythoni moodulit `turtle` (tõlkes "kilpkonn"), mille abil saab ekraanile kujundeid joonistada. Joonistamiseks tuuakse mängu tegelane - kilpkonn, kes oma virtuaalmaailmas ringi liikudes võib jätta maha jälje. See jälg võib teinekord päris huvitava ja isegi kunstilise kujuga olla.

AJALUGU

Selline virtuaalne kilpkonn ilmus esmakordselt hariduslikus programmeerimiskeeles LOGO, mille lõid aastal 1967 Wally Feurzeig ja Seymour Papert. See keel oli mõeldud just lastele algoritmilise mõtlemise arendamiseks. Praeguseks on kilpkonnagraafikat võimalik kasutada mitmetes programmeerimiskeeltes, ka Pythonis. Viimasel ajal eriti populaarses hariduslikus programmeerimiskeeles *Scratch* on samuti tuntavaid kilpkonnagraafika mõjutusi. LOGO programmeerimist võib proovida siin. Võib-olla on huvitav teada, et üks esimesi interneti kaudu toimunud kursusi (aastal 1996, tol ajal e-posti kaudu) Eestis oli just programmeerimiskeele Logo kursus.

ALUSTAMINE JA LIIKUMINE

Järgnevalt uurime, kuidas kilpkonna Pythonis kasutama hakata ning vaatame, kuidas kujundeid joonistada. Kokkuleppeliselt nimetatakse kilpkonnaks väikest kolmnurka joonistamisväljal.

Järgnevat videot võite vaadata kohe või pärast mõningat iseseisvat katsetamist.

Selleks, et joonistamisega alustada, tuleb esmalt importida vajalikud käsud kilpkonna moodulist. Selleks kirjutame programmi esimesele reale:

```
from turtle import *
```

Järgnevalt proovime kirjutada programmi, mille abil joonistataks ruut. Selleks kasutame sobivaid kilpkonnakäske:

- `forward(n)` - liigu edasi (ingl *forward*) *n* sammu võrra (ühe sammu pikkus on 1 piksel ehk üks täpike ekraanil);
- `back(n)` - liigu tagasi (ingl *back*) *n* sammu võrra;

- `left(d)` - pööra vasakule (ingl *left*) *d* kraadi;
- `right(d)` - pööra paremale (ingl *right*) *d* kraadi.

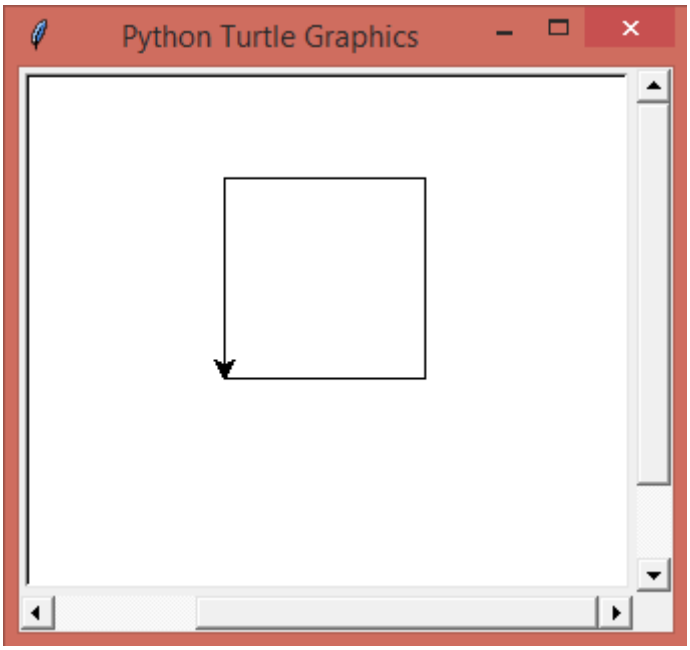
Eelnevaid käske sobivas järjekorras rakendades saame järgneva programmi.

Näiteprogramm. Ruut

```
from turtle import *                # * lisamisel imporditakse kõik
kilpkonna käsud

forward(100)                        # Kilpkonn liigub edasi 100
pikslit
left(90)                            # Kilpkonn pöörab 90° vasakule
forward(100)                        # Kordame eelnevaid käske, sest
ruudul on neli külge
left(90)
forward(100)
left(90)
forward(100)

exitonclick()                       # Saame akna sulgeda
hiireklõpsuga
```

NB! Ärge pange oma Pythoni programmi nimeks *turtle.py*. See ajab Pythoni segadusse.

Tulemuseks saame



Tegelikult võib kilpkonn järjekindlalt pöörata ka 90° paremale, tulemuseks on ikka ruut.

Näiteprogrammis märkame, et kilpkonn peab täitma korduvalt samu käske: minema 100 pikslit edasi ja pöörama seejärel 90° vasakule. Seega on siin tegemist tsüklilise tegevusega. Tsüklitest aga räägime edasistes osades.

Selleks, et vähendada kirjutamisvaeva, saab kilpkonna liikuma panna ka lühemate käskudega, kasutades pikemate esituste esimesi ja viimaseid tähti.

- `fd(n)` - võrdne käsuga `forward(n)`;
- `bk(n)` - võrdne käsuga `back(n)`;
- `lt(d)` - võrdne käsuga `left(d)`;
- `rt(d)` - võrdne käsuga `right(d)`.

See robot-kilpkonn oskab teha veel palju muudki. Täpsemalt võib vaadata Pythoni dokumentatsioonist.

VÄRVIMINE

Kilpkonn oskab ka joonistatud kujundeid värvida ja muuta taustavärvi. Selleks, et värvima hakata, tuleb esmalt valida sobiv värv. Seda saab teha käsuga `color(värvi_nimetus)`, kus värv antakse sõnena, mille väärtuseks on värvi ingliskeelne vaste või värvi kuueteistkümnendkoodis esitus. Näiteks musta värvi esitus kuueteistkümnendkoodis on *#000000*. Värvikoode saab vaadata siit.

Näiteks

```
color("red")                          # Kilpkonn muutub punaseks
```

Või

```
color("#008000")                      # Kilpkonn muutub tumeroheliseks
```

Kui värv on valitud, siis tuleb kilpkonnale käsuga `begin_fill()` teada anda, et nüüd lähebki värvimiseks. Seda peab tegema vahetult enne kujundi joonistamist, mida värvida soovime. Värvimise lõpetamiseks tuleb peale kujundi lõppu kirjutada käsk `end_fill()`. Järgmiseks kirjutame programmi, mis joonistab punase ringi. Kilpkonn oskab joonistada ringi (ingl *circle*) käsuga `circle(r)`, kus *r* on ringi raadius pikslites.

Näiteprogramm. Jaapani lipp

```
from turtle import *

color("red")                          # Kilpkonn muutub punaseks
begin_fill()                          # Kilpkonn alustab ringi
värvimist
circle(100)                           # Kilpkonn joonistab ringi
raadiusega 100 pikslit
end_fill()                            # Kilpkonn lõpetab ringi
värvimise
exitonclick()
```

Kui eelmisel programmi puhul oli täpselt teada, mis toimub, siis täiendame seda programmi juhuslikkusega. Selles programmis imporditakse käske lausa kahest moodulist.

Näiteprogramm. Sinine, must või valge

```
from turtle import *
from random import randint

värv = randint(1, 3)
if värv == 1:
    color("blue")                   # Sinine
if värv == 2:
    color("black")                  # Must
if värv == 3:
    color("white")                  # Valge valgel taustal
begin_fill()                        # Kilpkonn alustab ringi
värvimist
circle(100)                         # Kilpkonn joonistab ringi
raadiusega 100 pikslit
end_fill()                          # Kilpkonn lõpetab ringi
värvimise

exitonclick()
```

Praegu tõime kilpkonna mängu selleks, et edasisi natuke keerulisemaid teemasid paremini illustreerida. Aga mitte ainult - hakkame ikka ilusaid pilte ka tegema.

2.7 Silmaring. Arvusüsteemid

Kümnendsüsteem

Selle nädala silmaringi materjal on **arvusüsteemidest**. Alustame terminitest "arv" ja "number", mida paraku üsna sageli segi aetakse. Sellele lisab hoogu asjaolu, et inglise keeles on sõna "arv" vasteks just "number" ja juhtub, et tõlkides jäetakse ka eesti keeles sõnaks "number". Samuti on eesti keeles mitmeid termineid, kus "number" on kinnistunud, aga tegelikult pigem arvu tähenduses on (nt telefoni number, kinga number, passi number).

Niisiis püüdes sellest mitte liiga suurt numbrit teha, peame siiski vähemalt antud materjalis (ja tegelikult kogu sel kursusel) neil mõistetel vahet tegema.

Ajalooliselt oli **arv** algul loendamise tulemus. Loendades saame naturaalarvud, 1, 2, 3, 4, 5 jne. 0 võetakse mõnes kontekstis naturaalarvude hulka, mõnes mitte. Eks siin olegi mõneti filosoofiline küsimus - kas siis, kui midagi ei ole loendatud, saab üldse rääkida loendamise tulemusest! Hiljem arvude mõistet laiendati - räägiti täisarvudest, ratsionaalarvudest, reaalarvudest, kompleksarvudest. Meie jääme siin põhiliselt täisarvude juurde.

Igatahes tekkis ajalooliselt vajadus arve kirja panna. Kasutusel on olnud väga erinevaid süsteeme. Tänapäeval on kõige levinum viis panna arve kirja **araabia numbrite** abil. Algselt India päritolu märgid jõudsid Araabia kaudu Euroopasse. Seejuures 0 võeti teistest numbritest märgatavalt hiljem kasutusse. Numbreid on siis klassikaliselt kasutusel kümme. Need on 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Numbrid on seega sümbolid - märgid, mille abil arve kirja pannakse.

Arve, mida kirja panna tuleb, on muidugi palju rohkem kui kümme. Teoreetiliselt on neid lausa lõpmatult palju. Kui väikeste naturaalarvude puhul saame ühe numbriga arvu kirja, siis edasi peab mingit süsteemi kasutama. Nii ongi kujunenud, et lisaks numbri enda väärtusele loeb ka tema positsioon. Näiteks arvus 77 on esimese 7 tähendus tegelikult 70 ja teisel 7. Sellist süsteemi nimetatakse **positsiooniliseks** arvusüsteemiks. Positsioon on oluline!

Kui nüüd nt 2016 juppideks võtta saame 2016 = 2000 + 10 + 6. Näeme, et 0 on väga oluline ja näitab, et sajalisi pole. Veel detailsemalt saame, et $2016 = 2 \cdot 10^3 + 0 \cdot 10^2 + 1 \cdot 10^1 + 6 \cdot 10^0$. Tähele tuleb panna, et arvu väärtuse leidmisel hakatakse tegelikult pihta paremalt. Paremalt esimene **number** näitab üheliste ($10^0 = 1$) arvu, teine **number** kümneliste ($10^1 = 10$) jne. Lühematel juhtudel näeme muidugi peale vaadates ära, mis positsioonil paremalt viimane ehk vasakult esimene number on. Pikematel juhtudel aga tulebki paremalt lugema hakata.

Ülesanne

Märkida, milliseid tähendusi omavad numbrid 5 arvus 15525.

- [ ] 5
- [ ] 50
- [ ] 500
- [ ] 5000
- [ ] 50000

Kuna on kasutada kümme erinevat numbrit, siis nimetatakse seda süsteemi **kümnendsüsteemiks**. See on tõesti laialt levinud, seejuures ka näiteks mõõtühikute (sh rahaühikute) juures. Siiski pole tegemist ainukese võimaliku arvusüsteemiga. Aegade jooksul on kasutuses olnud väga erinevad süsteemid. (Näiteks ka rahaühikute korral - Suurbritannia naelsterling jaotub 100 penniks alles alates 1971. aastast.)

Arvusüsteemid pole alati ka (puhtalt) positsioonilised olnud. Näiteks rooma numbrite puhul on positsioon küll oluline, aga mitte ülal toodud süsteemi järgi. Nüüd aga jätkame ikkagi positsiooniliste süsteemidega.

Kahendsüsteem

Võib tekkida õigustatud küsimus, et kas kümme on ainuke võimalik numbrite arv ja arvusüsteemi alus. Kas võib süsteemi üles ehitada ka vähemate numbritega või hoopis enamate? Jah, saab nii vähemate kui enamatega. Vaatleme nüüd sellist süsteemi, kus numbreid on ainult kaks: 0 ja 1. Tegemist on **kahendsüsteemiga**. Arvude üles kirjutamise idee on samasugune nagu kümnendsüsteemis. Esialgu saame hakkama ühe numbriga. Kui loendatavaid elemente pole, siis on neid 0 ja kui on üks, siis on 1. Rohkem aga erinevaid numbreid pole. Tekib samasugune olukord nagu kümnendsüsteemis arvu 9 puhul, mille järel enam uusi numbreid pole. Nagu kümnendsüsteemiski tuleb nüüd kasutusele kirjapilt 10, mis tähendab siis tavamõttes kahte, arvutus oleks selline: $1 \cdot 2^1 + 0 \cdot 2^0$.

Rõhutada tuleb, et tegemist on loendamise mõttes ikka sama arvuga, lihtsalt kirjapilt on erinev. Selleks, et eristada erinevaid aluseid, märgitakse neid alaindeksitega, seega $10_2 = 2_{10}$. Pythonis aga pannakse kahendarvudele ette *0b*. Näiteks 0b10 on siis tavamõttes 2.

Toome tabeli mõningate arvudega.

Kümnendarv	**Kahendarv**
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
14	1110
15	1111
16	10000
31	11111
32	100000
63	111111
64	1000000

Näeme, et erinevates süsteemides on "ümmargused" hoopis erinevad arvud. Hea võimalus näiteks juubelite pidamiseks!

Kahendarvudel on praktiline väärtus just arvutitega seoses. Nimelt on kahte võimalikku seisundit palju hõlpsam realiseerida kui näiteks kümmet erinevat. Lamp põleb / ei põle. Mälupesas on midagi / ei ole. Kahendarv koosneb kahendnumbritest. Inglise keeles on kahendnumber *binary digit*, millest on tuletatud mõõtühiku nimeks *bit*. Bitt ongi informatsiooni põhiühikuks.

Kahendarvudega saab tehteid teha analoogiliselt kümnendarvudega, aga seda me siinkohal ei vaatle, mis ei takista siiski selleteemalise küsimuse esitamist.

Ülesanne

Mis on kahendsüsteemis 1 + 1 väärtus?

Vali 2

Vali 10

Vali 11

Vali Selline tehe pole võimalik.

Neile, kes kahendsüsteemist väga vaimustatud on, võivad näiteks huvi pakkuda kahendsüsteemis (tegelikult küll modifitseeritud viisil) aega näitavad kellad.

Kaheksandsüsteem. Kuueteistkümnendsüsteem

Kahendarvud lähevad üsna kiiresti pikaks ja see on üks põhjus, miks kasutatakse ka kaheksandarve ja kuueteistkümnendarve. Kuna 8 ja 16 on 2 astmed, siis on konverteerimine näiteks kuueteistkümnend ja kahendsüsteemi vahel oluliselt lihtsam kui kümnendsüsteemi ja kahendsüsteemi vahel.

Kaheksandsüsteemis saab ikka tavalisi numbreid kasutada - lihtsalt 8 ja 9 jäävad kasutamata. Kuueteistkümmendsüsteemis aga kümnest märgist ei piisa. Spetsiaalseid numbreid pole siiski juurde mõeldud, kasutatakse tähti A, B, C, D, E ja F.

Lisamegi nüüd eelmisele tabelile kaheksandarvude ja kuueteistkümnendarvude veerud.

Kümnendarv	Kahendarv	Kaheksandarv	Kuueteistkümnendarv
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A

11	1011	13	B
14	1110	16	E
15	1111	17	F
16	10000	20	10
31	11111	37	1F
32	100000	40	20
63	111111	77	3F
64	1000000	100	40

Kuidas siis mõista kaheksandarvu 137? Nüüd on siis aluseks 8, mis tähendab, et

$137_8 = 1 \cdot 8^2 + 3 \cdot 8^1 + 7 \cdot 8^0 = 64 + 24 + 7 = 95$

Vaatleme ka kahte kuueteistkümnendarvu. Neis võivad numbritena olla tähed tähestiku algusest, aga ei pruugi olla. Näiteks arvestades, et A on kümnendsüsteemi 10 ja D on 13, saame

$A2D_{16} = 10 \cdot 16^2 + 2 \cdot 16^1 + 13 \cdot 16^0 = 2560 + 32 + 13 = 2605$

ja

$1010_{16} = 1 \cdot 16^3 + 0 \cdot 16^2 + 1 \cdot 16^1 + 0 \cdot 16^0 = 4096 + 16 = 4112.$

Pythonis on kaheksandsüsteemis arvu märkimiseks kasutusel prefiks (eesliide) *0o* ja kuueteistkümnendarvude jaoks *0x*.

Kuueteistkümnendarve kasutatakse sageli näiteks mäluaadressides ja ka näiteks värvide märkimiseks. Erinevate värvide kuueteistkümnend koode saab vaadata siit. Neid saame ka kilpkonnagraafikas kasutada. Näiteks `turtle.pencolor('#85e085')` mõjul muutub kilpkonna sule heleroheliseks.

Kaheksandsüsteemi arve kasutatakse näiteks mõnedes operatsioonisüsteemides failidele juurdepääsu õiguste määramiseks. Näiteks Unixis annab `chmod 664 mingifail.txt` kasutajale ja rühmale õiguse lugeda ja kirjutada, aga mitte käivitada. Teised saavad ainult õiguse lugeda.

Ülesanne

Millised järgnevatest on kuueteistkümnendsüsteemi arvud?

- [ ] 1CE
- [ ] 100
- [ ] -1A
- [ ] AG3
- [ ] F1

Veel lugemist

Ise saate arve erinevate alustega süsteemidesse konverteerida näiteks siin või ka siin.

Nagu paljudest muudest asjadestki on arvusüsteemidest juttu "Matemaatika õhtuõpikus", mille saab ka alla laadida.

2.8 Silmaring. Tõeväärtused

TÕEVÄÄRTUSTABELID

Eespool vaatlesime salatinäidet.

```
kartul_olemas = True
makaron_olemas = False
salatikaste_olemas = True
print((kartul_olemas or makaron_olemas) and salatikaste_olemas)
```

Soovitati ka proovida programmi tööd muutujate `kartul_olemas`, `makaron_olemas` ja `salatikaste_olemas` erinevatel väärtustel. Kerkib küsimus, mitu erinevat varianti üldse on? Igaüks kolmest muutujast võib omada kahte erinevat väärtust. Kui olekski ainult üks muutuja, oleks võimalusi kaks:

- `True`
- `False`

Kui oleks ainult kaks muutujat, siis võimalusi oleks neli:

- `True`, `True`
- `True`, `False`
- `False`, `True`
- `False`, `False`

Kui on kolm muutujat, siis on võimalusi kaheksa:

- `True`, `True`, `True`
- `True`, `True`, `False`
- `True`, `False`, `True`
- `True`, `False`, `False`
- `False`, `True`, `True`
- `False`, `True`, `False`
- `False`, `False`, `True`
- `False`, `False`, `False`

Ülesanne

Kui palju oleks erinevaid variante, kui oleks 10 erinevat muutujat ja igal oleks võimalikud väärtused True ja False.

Vali 100

Vali 512

Vali 1024

Vali 2048

Meie näites on salati jaoks vaja kartulit **või** makaroni. Sõnal "või" on tavakeeles mõnevõrra teistsugune tähendus, kui programmeerimisel (või ka matemaatilises loogikas). Tavakeeles on "või" sageli välistav - kas see või teine (aga mitte mõlemad). Loogikas ja programmeerimisel on aga tehe `or` tõene ka siis, kui mõlemad operandid on tõesed. Seda saab kujutada ka tõeväärtustabelina. Loogikas tähistatakse *or*-tehet sageli märgiga `V`. Tõene ja väär on tähistatud vastavalt *t* ja *v*.

A	B	A v B
t	t	**t**
t	v	**t**
v	t	**t**
v	v	**v**

Toome ka tõeväärtustabeli `and`-tehte jaoks, mida sageli tähistatakse märgiga `&`.

A	B	A & B
t	t	**t**
t	v	**v**
v	t	**v**
v	v	**v**

Meie salatinäites on tõeväärtustabel arvutatud kahes järgus kõigepealt A v B, mis siis meil tähendab `kartul_olemas or makaron_olemas` ja pärast siis kogu avaldis `(kartul_olemas or makaron_olemas) and salatikaste_olemas`.

A	B	C	(A v B) & C
t	t	t	**t**
t	t	v	**v**
t	v	t	**t**
t	v	v	**v**
v	t	t	**t**
v	t	v	**v**
v	v	t	**v**
v	v	v	**v**

LOOGILISTE AVALDISTE SAMAVÄÄRSUS

Salatinäite puhul võime läheneda ka natuke teistmoodi. Kartulisalati jaoks oleks meil vaja kartulit ja salatikastet. Makaronisalati jaoks oleks vaja makarone ja salatikastet. Kokkuvõttes aga piisab, kui vähemalt ühe jaoks neist on materjal olemas. Vastav loogiline avaldis on siis järgmine.

```
(kartul_olemas and salatikaste_olemas) or (makaron_olemas and salatikaste_olemas)
```

Tegelikult ongi see samaväärne avaldisega, mis meil enne oli.

```
(kartul_olemas or makaron_olemas) and salatikaste_olemas
```

Loogiliste tehete puhul kehtivad mitmed seadused (analoogiliselt algebrale). Näiteks

- avaldis `a and b` on samaväärne avaldisega `b and a`;
- avaldis `a or b` on samaväärne avaldisega `b or a`;
- avaldis `(a or b) and c` on samaväärne avaldisega `(a and c) or (b and c)`;

Loogilistes tehetes orienteeruda on programmeerijale väga oluline ja nii on seda arendavad õppeained ka ülikoolide õppekavades.

2.9 Teise nädala kontrollülesanded 2.1, 2.2, 2.3

Teisel nädalal tuleb esitada nelja kohustusliku ülesande lahendused. Neljanda ülesande puhul on võimalik valida lahendamiseks vähemalt üks järgmistest ülesannetest, kas 2.4a, 2.4b või 2.4c (võib ka kaks või kolm lahendada). Lahendused tuleb esitada *Moodle* 'is, kus need kontrollitakse automaatselt. *Moodle* 'is on ka nädalalõputest 10 küsimusega, millest tuleb vähemalt 9 õigesti vastata.

Kontrollülesanne 2.1. Jäätumine

Mitmed autod hoiatavad võimaliku jää eest, kui temperatuur õues on 4,0 või alla selle.

Koostada programm, mis

- küsib kasutajalt õhutemperatuuri,
- väljastab ekraanile ***Ei ole jäätumise ohtu***, kui sisestatu on üle 4,0,
- väljastab ***On jäätumise oht***, kui temperatuur on 4,0 või alla selle.

Temperatuuri võib sisestada nii täisarvuna kui ka ujukomaarvuna, nt -1.3.

Näited programmi tööst:

```
>>> %Run yl2.1.py

  Sisesta õhutemperatuur: 10.5
  Ei ole jäätumise ohtu

>>>

>>> %Run yl2.1.py

  Sisesta õhutemperatuur: -1.3
  On jäätumise oht

>>>
```

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 2.2. Spämm

Kirjade seast rämpsposti (spämmi) leidmiseks saab kasutada filtreid, mis filtreerivad välja konkreetsetele tingimustele vastavaid kirju. Kalmer teeb filtrit, kus filtreeritakse välja kirjad, mille kohta on vähemalt üks järgmistest tingimustest tõene:

- kirjal ei ole teema pealkirja,
- kiri sisaldab manusena faili ja kirja suurus ületab 1 MB.

Koostada Kalmeri jaoks programm, milles

1. küsitakse kirja suurust megabaitides (kasutaja sisestab ujukomaarvu),
2. küsitakse kirja teema pealkirja (kasutaja sisestab teema pealkirja või kasutaja sisestus on tühi),
3. küsitakse, kas kirjaga on kaasas fail (kasutaja sisestab ***jah*** või ***ei***),
4. väljastatakse ekraanile ***Kiri on spämm***, kui kiri filtreeritakse välja, vastasel juhul väljastatakse ***Kiri ei ole spämm***.

Proovige kirjutada programm, kasutades ainult ühte tingimuslauset. Kui see ei õnnestu, siis võib ka mitmega.

NB! Kasutaja käest peab kindlasti küsima kolm korda.

Näited programmi tööst:

```
>>> %Run yl2.2.py

  Sisestage kirja suurus: 0.7
  Sisestage kirja teema pealkiri: Ülesanne 2.2
  Kas kirjaga on kaasas fail? jah
  Kiri ei ole spämm

>>>
>>> %Run yl2.2.py

  Sisestage kirja suurus: 0.8
  Sisestage kirja teema pealkiri:
  Kas kirjaga on kaasas fail? ei
  Kiri on spämm

>>>
```

Kontrollülesanne 2.3. Leedu perenimed

Inimese nimede osas on erinevatel maadel erinevaid kombeid ja vähemalt naabrite puhul oleks hea neid teada (areneva Balti koostöö mõttes).

Traditsiooniliselt näitab leedu naiste perekonnanimedes nime lõpp perekonnaseisu. Näiteks on Adamkienė abielus ja Adamkutė mitte. Alates 2003. aastast on lubatud ka lühem vorm, mis perekonnaseisu ei näita, nt Adamkė. Huvi korral uuri lähemalt siit.

Koostada programm, mis küsib kasutajalt Leedu perekonnanime ja väljastab ekraanile

- ***Abielus***, kui nimi lõpeb tähtedega "ne",
- ***Vallaline***, kui nimi lõpeb tähtedega "te",

- ***Määramata***, kui nimi lõpeb tähega "e" (aga mitte "ne" ja "te"),
- ***Pole ilmselt leedulanna perekonnanimi***, kui nimi ei lõpe tähega "e".

Lihtsuse mõttes kasutame tavalist tähte "e", jätame punkti peale panemata.

Sõne `nimi` kahe viimase tähe kontrollimiseks saab kasutada näiteks võrdlemist `nimi[-2:] == "ne"`. Viimase tähe kontrollimiseks sobib `nimi[-1] == "e"`.

Näited programmi tööst:

```
>>> %Run yl2.3.py

  Sisestage Leedu perekonnanimi: Adamkiene
  Abielus

>>>
>>> %Run yl2.3.py

  Sisestage Leedu perekonnanimi: Adamke
  Määramata

>>>
```

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.

2.10 Teise nädala kontrollülesanded 2.4abc

Teisel nädalal tuleb esitada nelja kohustusliku ülesande lahendused. Neljanda ülesande puhul on võimalik valida lahendamiseks vähemalt üks järgmistest ülesannetest, kas 2.4a, 2.4b või 2.4c (võib ka kaks või kolm lahendada). Lahendused tuleb esitada *Moodle* 'is, kus need kontrollitakse automaatselt. *Moodle* 'is on ka nädalalõputest 10 küsimusega, millest tuleb vähemalt 9 õigesti vastata.

Järgmisest kolmest ülesandest (2.4a, 2.4b, 2.4c) tuleb lahendada vähemalt üks.

Kontrollülesanne 2.4a. Pulss

Tervisesport on tervisele kasulik, kui sellega jäädakse mõõdukuse piiridesse. On erinevaid variante sobiva koormuse valimiseks. Näiteks saab kasutada sellist arvestust, et maksimaalne pulsisagedus on meestel 220 miinus vanus ja naistel 206 miinus 88% vanusest. Seejuures erinevate treeningutüüpide puhul peaks pulsisagedus jääma järgmistesse vahemikesse:

- tervisetreening 50-70% maksimaalsest pulsisagedusest,
- põhivastupidavuse treening 70-80% maksimaalsest pulsisagedusest,
- intensiivne aeroobne treening 80-87% maksimaalsest pulsisagedusest.

Koostada programm, mis küsib kasutajalt

- vanuse (täisarvuna aastates),
- soo (kasutaja sisestab ***M***, ***m***, ***N*** või ***n***),
- treeningu tüübi (1 - tervisetreening, 2 - põhivastupidavuse treening, 3 - intensiivne aeroobne treening)

ja lõpuks väljastab pulsisageduse vahemiku vastavatel tingimustel formaadis ***<vähim pulss> kuni <suurim pulss>***, kus vastuses leiduvad arvud on ümardatud täisarvudeks.

Näited programmi tööst:

```
>>> %Run yl2.4a.py

  Sisestage enda vanus: 20
  Sisestage enda sugu: n
  Sisestage treeningu tüüp: 1
  Pulsisagedus peaks olema vahemikus 94 kuni 132.

>>>
```

```
>>> %Run yl2.4a.py

  Sisestage enda vanus: 60
  Sisestage enda sugu: M
  Sisestage treeningu tüüp: 2
  Pulsisagedus peaks olema vahemikus 112 kuni 128.

>>>
```

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 2.4b. Istekoht

Odavlennufirmad pakuvad reisijatele küllaltki soodsaid lennupileteid, kuid lisaväärtuse eest peavad reisijad juurde maksma. Näiteks kui tahetakse lennukis istekoht ise valida, siis ka seda saab lisatasu eest. Istekohta valides küsitakse inimeselt, kas ta soovib istuda akna äärde või mujale. Kui reisija ei taha valida, siis loositakse istekoht nii, et 1/3 tõenäosusega on see akna ääres ja 2/3 tõenäosusega mujal.

Koostada programm, mis vastab järgmistele tingimustele:

1. Küsib kasutajalt, kas ta soovib istekoha ise valida. Ise valimiseks sisestab kasutaja ***"ise"***. Vastasel juhul kirjutab kasutaja ***"loos"***.
   - Kui kasutaja soovis ise valida, siis küsitakse tema käest, kas ta soovib istuda akna ääres (kasutaja sisestab ***"aken"***) või mitte (kasutaja sisestab ***"muu"***).
   - Kui kasutaja valis loosi, siis loositakse talle istekoht nii, et 1/3 tõenäosusega on see akna ääres ja 2/3 tõenäosusega mujal.
2. Väljastatakse, kas kasutaja valis istekoha ise (***"Valisite ise"***) või valiti see loosiga (***"Istekoht loositi"***).
3. Väljastatakse, kas kasutaja istekoht on akna ääres (***"Aknakoht"***) või mitte (***"Vahekäigukoht"***).

Lahendus peab korrektselt kasutama funktsiooni randint() moodulist random.

Näited programmi tööst:

```
>>> %Run yl2.4b.py

  Kas soovite istekohta ise valida ("ise") või loosida ("loos")? ise
  Kas soovite istuda akna ääres ("aken") või mitte ("muu")? aken
  Valisite ise. Aknakoht

>>>
```

```
>>> %Run yl2.4b.py

  Kas soovite istekohta ise valida ("ise") või loosida ("loos")? loos
  Istekoht loositi. Vahekäigukoht

>>>
```

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 2.4c Busside logistika

Meil on vaja transportida teatud arv inimesi mingi arvu identsete bussidega. Eeldame, et reisijaid on vähemalt üks.

Koostada programm, mis küsib transporditavate inimeste arvu ja ühe bussi kohtade arvu (just sellises järjekorras) ning väljastab ekraanile, mitu bussi on vaja ja mitu inimest on viimases bussis (eeldusel, et kõik eelnevad bussid on täis).

Võib-olla on abi nendest tehetest

- // täisarvuline jagamine, 36 // 10 → 3
- % jäägi leidmine 36 % 10 → 6

Testige oma programmi muuhulgas järgmiste algandmetega:

- inimeste arv: 60, kohtade arv: 40;
- inimeste arv: 80, kohtade arv: 40;
- inimeste arv: 20, kohtade arv: 40;
- inimeste arv: 40, kohtade arv: 40.

Püüdke ka mõista, miks just sellised testandmed valiti.

Näited programmi tööst:

```
>>> %Run 2.4c.py

  Inimeste arv: 60
  Kohtade arv: 40
  Busse vaja: 2
  Viimases bussis inimesi: 20

>>>
```

```
>>> %Run 2.4c.py

  Inimeste arv: 80
  Kohtade arv: 40
  Busse vaja: 2
  Viimases bussis inimesi: 40

>>>
```

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.

2.11 Alkeemia II

Alkeemia II

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

„Valikud, valikud," mõtiskles meister valjult, „Igaüks meist teeb enda elus lõpmatult palju valikuid. Üks alkeemia põhiteooriatest näitab, et iga valiku saab taandada labasele 'jah' või 'ei' küsimusele."

Tea, kes oli võtnud kohustuseks panna proovile iga professori väide, tõstis käe ning formuleeris jooksu pealt mitu stsenaariumi. Vana alkeemik märkas küsimust:„Teie, tagumises reas!"

„Ma kardan, et ma ei mõista. Kuidas on võimalik, et kõik keerulised valikud, mis ma teinud olen, oleks saanud lahendada 'jah või 'ei' meetodil? Näiteks siis, kui mul on aega õppida korralikult vaid ühe aine eksamiks ja tulekul on kaks."

Vana mees haaras kriidijupi ning asus vastamise ajal mustale tahvlile kritseldama. „Hea küsimus. Kui me mõtleme, mis on üks keeruline valik, siis leiame tihti, et sellesse on peidetud mitu väiksemat valikut."

Ta joonistas tahvlile puutüve ja kirjutas sellele: „Kas ma õpin täna eksamiks?"

„See on meie esimene küsimus. Peame selle lahendama enne, kui saame edasi liikuda. Mina langetaksin selle valiku erinevate abistavate asjaolude kaudu." Tahvlile tekkisid kiirelt uued kriidijooned – „Kas mul on täna piisavalt aega?", „Kas mul on midagi paremat teha?", „Kas akadeemiline edukus on minu jaoks tähtis?".

„Õppimiseni jõudmiseks peaksid olema vastused 'jah', 'ei' ja 'jah'. Oletame, et see ka nii on", ta heitis tudengitele teeseldud pahaks paneva pilgu ning muigas. „Sellisel juhul lähme uute puuoksade juurde. Me saame teha vaid ühe valiku korraga, kuid mõned valikud võivad välistada teisi. Näiteks, ütleme, et me oleme käinud vähem Alkeemia loengutes, kui Kaugete Maade Ajaloos ja selle tõttu tahame hoopis õppida Alkeemia eksamiks. See on üks lihtne näide, kuid alkeemikud kasutavad säärast loogikat igasugustes keerukates maagilistes esemetes."

Noor naine oli kogu selle aja kaasa mõelnud ning talle tuli meelde ese, mida Ülikooli ajaloo raamatud mainisid. „Kas Rännukivi töötab samal põhimõttel?" päris Tea.

Lektor pani kriidijupi tagasi karbikesse ning toetus mõtlikult enda laua nurgale klassiruumi ees. „Võib-olla on tõesti aeg rääkida teile Rännukivist ja teie kõigi edasistest õpingutest. Hea küll. Esimeste nädalate jooksul olete õppinud sadat sorti katlaid segama, kuid on tähtis

mõista, et alkeemik ei ole katla ori. Katel ja pudelid on tööriistad ning teie teete valiku neid kasutada. Vanade meistrite arvates on parim viis alkeemiat õppida seda rakendades. Seepärast paisatakse kõik piisavalt kogenud tudengid Ülikooli tava järgi maailma eri nurkadesse, siis, kui nad seda kõige vähem ootavad. See on parim viis teile näidata, et alkeemia on mõtteviis, mitte raamatutes tuhlamine ja tõmmistega solgutamine, ning see ei oota kellegi järel.“

Klassiruumis valitses ärev vaikus. Mõned pinginaabrid hakkasid midagi sosinal arutama ning varsti kasvas pingeline nihelemine valjuks jutuajamiseks.

„Vaikust!“ palus alkeemik ja sikutas enda habet. „Rahu! Ärevuseks ei ole põhjust. Mõelge sellest kui keerulisest rühmatööst. Üleüldse peaksite te praegu mõtlema millelegi muule. Teil on heal juhul vaid mõni päev, et saada sinasõbraks ühe vanima alkeemia töövahendiga.“

Mees kõndis enda laua taha, soris sahtlites ning pistis midagi vestitaskusse. Tea oleks võinud vanduda, et tegemist oli väikese kilpkonnaga. „Järgnege mulle, kui julgete!“ lausus õppejõud ning astus tagasi vaatamata klassiruumist välja, kaval naeratus näol.