1. LOGO ja Pythoni kilpkonnagraafika (moodul Turtle)

Lastele programmeerimise tõdede õpetamine ei ole mingi viimase aja leitutis. Cynthia Solomon ja Seymour Papert koos meeskonnaga tegelesid sellega juba üle 50 aasta tagasi. Tulemuseks oli keel LOGO ning mitmed artklid õpetamise kohta (kaks viidet 1971 ja 1976 a artiklitele MIT raamatukogus leiad ka kursuse veebist), samuti matemaatika õpetamisest programmeerimise toel (viide kursuse veebis).

Võid uurida lisaks LOGO ajalugu vikipeediast (https://en.wikipedia.org/wiki/Logo_(programming_language)),

Mitmete piltidega illustreeritud LOGO kilpkonna ajaloo leiad siit:

(http://cyberneticzoo.com/cyberneticanimals/1969-the-logo-turtle-seymour-papert-marvin-minsky-et-al-american/)

LOGO kilpkonna põhimõttel töötavad tänapäevased lasterobotid - nt Bee Botid jms. S. Papert osales Lego Mindstorm komplekti väljatöötamisel, mis omakorda sai nime S. Paperti raamatu järgi (https://dl.acm.org/doi/pdf/10.5555/1095592).

1.1. Elementaarsed liikumised LOGO keeles

Katsetame kõigepealt LOGO veebirakenduses: https://www.calormen.com/jslogo/

PKilpkonna liikumise põhimõtted on lihtsad: kilpkonn liigub sinna, kuhu pea näitab. Saab määrata sammude arvu. Kilpkonn liigub ka tagurpidi (saba ees). Kilpkonn pöörab nii vasakule kui paremale. Peale pööramist liigub ta edasi uues suunas. Pöördenurk antakse kraadides.

Vt ka 1. joonistamise varianti Turtle mooduli peatükis 1.3.1.

Valik käske:

`forward 100` ehk `fd 100` - kilpkonn liigub edasi 100 sammu

`back 100` ehk `bk 100` - kilpkonn liigub tagasi 100 sammu

`clearscreen` ehk `cs` - kustutab kõik ja kilpkonn läheb lähteasendisse

`right 90` ehk `rt 90` - kilpkonn pöörab 90° päripäeva (paremale) pööramisel saab kasutada kraade 0 kuni 360

`right 180` ehk `rt 180` - kilpkonn pöörab 180° vastupäeva (vasakule)

Teeme mõned katsed, ülesanded pärinevad materjalist "Programming in LOGO". Materjali link on kursuse veebis.

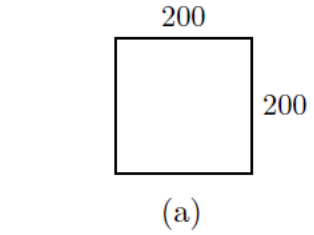
Ülesanne 1 Kilpkonn liigub

Sisesta järgmised käsud (võivad olla nii üksteise all kui kõrval) ja **mõtle täpselt läbi**, mida mingi käsk teeb:

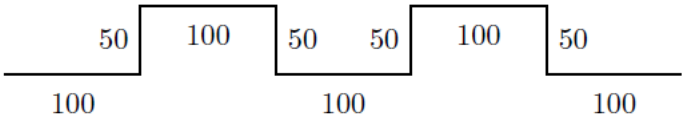
```
fd 100 rt 90 fd 150 rt 90 fd 50 lt 90 fd 150 rt 90 fd 50
```

Ülesanne 2 Kujundid

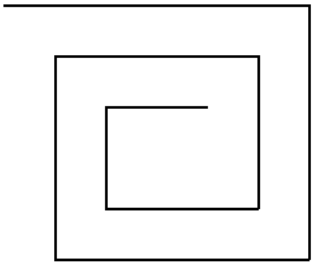
Proovi eelmise ülesande käske kasutades joonistada järgmiseid kujundeid

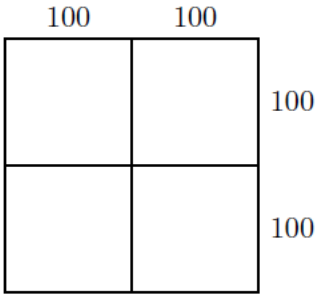


(a)



(b)





1.2. Juhuslikud arvud ja muu juhuslikkus Pythonis

Juhuslikest arvudest on tegelikult juba juttu olnud.

Mängude ja muidu lõbusate asjade tegemiseks on tihti abi juhuslikest valikutest. Juhuslike valikute tegemiseks aitavad kaasa juhuslikud arvud. Nende arvude väljaarvutamiseks on mitmeid matemaatilisi meetodeid. Seega pole need arvud midagi nii väga juhuslikud. Õigem ongi neid kutsuda pseudojuhuslikeks arvudeks.

Juhuslike arvude funktsioonide kasutamiseks tuleb eelnevalt importida moodul `random`:

```
import random
```

Moodulis `random` on mitmeid funktsioone, kuid kasulikumad võiksid olla:

`random.randint(algus,lõpp)` - täisarv antud vahemikust

`random.choice(list)` - `list` on väärtuste jada (arvud, stringid jms), valitakse suvaline element nende hulgast

`random.shuffle(list)` - vahetab etteantud listi elemendid suvalisse järjekorda ja pärast võib neid sealt näiteks järjest võtta ja millekski kasutada.

Pane tähele, et iga funktsiooni nime ette on kirjutatud mooduli nimi!

Näited:

Moodustame listi (järjendi) nimega `värvid`:
```
värvid = ['red','blue','green','magenta']
```
Laseme arvutil valida loetelust juhuslikult ühe värvi `juhuslik_värv`:
```
juhuslik_värv = random.choice(värvid)
```
Laseme arvutil valida juhusliku täisarvu (`juhuslik_arv`) vahemikust 1 kuni 100:
```
juhuslik_arv = random.randint(1,100)
```

1.3. Turtle moodul

Katsetame edasi Pythoni Turtle mooduliga - kilpkonnagraafika funktsioonikoguga.

Et hakata programmi kirjutama, tuleb Pythoni töökeskkond (*IDE, Integrated Development Environment*) käima panna. Selleks keskkonnaks on Thonny (või kui Sul varasem kogemus mõne teise töökeskkonnaga, siis sobib ka see).

Mooduli funktsioonide kasutamiseks tuleb moodul importida:

```
from turtle import *
```
Seekord näeb importimise käsk välja pisut teistmoodi, et mooduli nime iga funktsiooni (käsu) ees mitte korrata.

Järgnevas tekstis on nimetatud mitmeid kilpkonna-funktsioonide nimesid. Nime järele on alati lisatud ka sulupaar, sest üldiselt programmeerides järgneb selline sulupaar alati funktsiooni nimele funktsioonikutses. Funktsioonikutse on funktsiooni käivitamine, tema tööle panemine.

Joonistatakse pliiatsiga (*pen*, mis on nagu kilpkonna saba). Kui saba on maas (`down()`), jääb jälg. Kui saba on üleval (`up()`), siis jälge ei jää.

Järgnevas tekstis mainitud näiteprogrammid leiad kursuse veebilehelt lingi „Turtle näited” alt.

NB! Pane tähele, et ehkki kilpkonna tegevus Pythoni Turtle mooduli käskudega (funktsioonidega) ning päris LOGO-keeles on sarnane, erineb käskude kasutamine siiski mingil määral - pööra tähelepanu sulgude kasutamisele!

1.3.1. Joonistamine: variant 1. Kilpkonn liigub sinna, kuhu pea (või saba) näitab

Kilpkonnal on suund. Kui kilpkonnale öelda liigu edasi 50 sammu (pikslit), siis ta liigub selles suunas, kuhu pea näitab. Kilpkonn saab ka liikuda tagasi (`forward()`, `backward()`, ka `fd()`, `bk()`). Kilpkonna saab pöörata praeguse suuna suhtes (`right()`, `left()`, ka `rt()`, `lt()`), andes ette pöördenurga kraadides (vaikimisi) või radiaanides.

Näiteks joonistame täisnurga (kopeeri käsud arvutisse, enne seda mooduli import!!):

```
forward(100)# Liigu edasi 100 pikslit
left(90)          # Pööra vasakule 90°.
forward(100)# Liigu edasi 100 pikslit
```

Varem on koodi lõppu soovitatud kirjutada `done()`, mis lõpetab täitmise. Aga praegu tundub, et see hoopis takistab programmi oma tööd lõpetamast. Nii et ärgem siis seda kirjutagem ...

> Ülesanne 3. Täisnurkne kolmnurk
>
> Kuidas saada soovitud suuruses võrdhaarset täisnurkset kolmnurka? Täienda programmi selliselt, et kasutajalt küsitakse kolmnurga kaatetite pikkused (käski `input()`) ja edasi joonista kolmnurk. Ehkki on ka mugavam viis joonistamise alguspunkti liikumiseks, proovi natuke matemaatikat - leia hüpotenuusi pikkus. Milline on nurk ehk kui palju ja kuhu poole peab kilpkonna pöörama?
>
> Ülesannet saab ajada matemaatilisemaks suvaliste kaatetite pikkustega. Mida sel juhul veel lisaks arvutada tuleks?

Pythoni abi matemaatika funktsioonide jaoks leiab siit: https://docs.python.org/3/library/math.html

Saab ka öelda, et keera pea sellesse või teise suunda sõltumata sellest, kuhu poole ta hetkel vaatab (`setheading()`). Suunda saab määrata arvudega:

`setheading(0)` - ida

`setheading(90)` - põhi

`setheading(180)` - lääs

`setheading(270)` - lõuna.

Peale kilpkonna pööramist saab teda liigutada uude suunda.

> Ülesanne 4 Moodustame tähti
>
> Proovi ise joonistada tähti või numbreid, kujundades nad kandilistena. Kas õnnestub oma nimi kirja panna? Sõltuvalt nimest võib see osutuda üsna tülikaks. Aga paar tähte võiks ikka proovida :). Et tähtedele vahed tekiksid, tõsta kilpkonna saba üles (`up()`), liigu uue tähe algusesse ja langeta saba (`down()`).

1.3.2. Tsüklilause ja joonistamine

Eespool toodud piltidest nii sakke (b) kui ka kandilist spiraali saab joonistada tsüklit kasutades. Tegelikult ka ruutu.

> **Näide ruutu joonistamisest**:
>
> ```
> for külg in range(4):
>     fd(100)
>     rt(90)
> ```

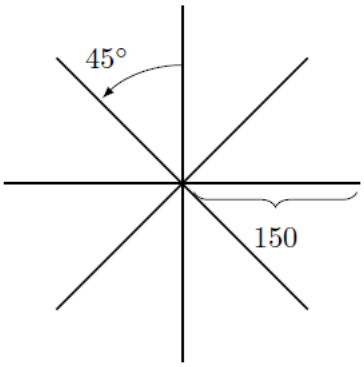
For-tsükli selgitus

```
for muutuja in range(N):
```

Keelesõna `for` järel peab olema vabalt valitav muutuja nimi. Funktsiooni `range()` järel sulgudes aga korduste arv `N`. Lisaks veel - muutujasse pannakse igas tsükli korduses uus arv: esimene arv on `0`, siis `1`, siis `2` jne kuni `N-1`. For-tsükkel on tegelikult keerukam "nähtus", aga esimeses lähenduses soovitud korduste arvu saamiseks piisab sellestki teadmisest.

Üldistus: **hulknurga joonistamine**. Kuidas arvutada välja, mitu kraadi peab kilpkonn pöörama, et näiteks 8-nurka joonistada? `360 / 8`. Kas see teooria sobib ka eelneva ruudu joonistamisega?

> Ülesanne 5 Täheke
>
> Kuidas joonistada täheke? (Vihje: kilpkonnaga saab ka tagurpidi liikuda.). Aga 16-haruline täheke? Kas oskame teha üldisema programmi, kus saame määrata tähekese harude arvu ning selle järgi muu vajaliku välja arvutada?



Sellise meetodi abil saab joonistada kujundeid ka polaarkoordinaate kasutades – loodan, et sõnastasin matemaatiliselt õigesti (vt `arhimedese_spiraal.py`)

1.3.3. Isetehtud käsud ehk funktsioonid

Programmeerimiskeele käskudest saab moodustada uusi käske. Näiteks võime teha ise käsud tähekese või hulknurga joonistamiseks. Selliseid käske nimetatakse funktsioonideks.

Järgnev funktsioon `hulknurk` joonistab hulknurga sellise külgepikkuse ja nurkade arvuga, nagu soovitakse. Tähele tuleb panna, et funktsioon ei hakka ise vabatahtlikult tööle. Ta tuleb tööle panna ehk välja kutsuda ja selle käigus täpsustatakse hulknurga andmed.

```
def hulknurk(küljepikkus, nurkadearv):
    nurk = 360 / nurkadearv
    for nurk in range(nurkadearv):
        fd(küljepikkus)
        rt(nurk)
```

```
hulknurk(100, 3)   # Funktsioonikutse, küljepikkus 100, nurkade arv 3
```

Kas oskaksid teha funktsiooni tähekese joonistamiseks?

1.3.4. Joonistamine: variant 2. Kilpkonn liigub etteantud koordinaatidele

Joonistusaknas on x-y koordinaatteljestik. Punkt `0,0` jääb akna keskele (nagu koolis matematika tunnis).

Kilpkonna käest saab küsida, millistel koordinaatidel ta paikneb (`position()`, `xcor()`, `ycor()`). Ja talle saab ka ette anda koordinaadid, kuhu ta minema peab (`setpos()`, `setx()`, `sety()`). Etteantud koordinaatidele liikumiseks ei ole teda vaja peaga nõutud suunda pöörata. Ta liigub edukalt ka külgsuunas. Liikumine jätab jätkuvalt maha jälje (kui just saba üles pole tõstetud).

Näiteks:

```
x,y = position()    # Saime teada praeguse asukoha
setpos(x+50, y+50)  # Liigutame diagonaalis kirdesse
```

Sarnaselt saab küsida ja muuta ka ainult ühte koordinaati:

```
x = xcor()
setx(x+50)
```

Ülesanne 6 Kilbu läks jalutama

Laseme prooviks kilpkonna suvalist trajektoori mööda jalutama. Kasutame for-tsüklit sammude kordamiseks ning sammu pikkuse ja uue asukoha määramiseks kasutame juhuslikke arve (vt 1.2).

Selles režiimis saab näiteks joonistada ka igasuguseid funktsioonide graafikuid, kui suudame leida parameetrilised võrrandid x ja y (x-i järgi y) väljaarvutamiseks.

Ülesanne 7 Funktsiooni graafik

Kuidas joonistada siinusfunktsiooni graafikut. Siin on vaja moodulist `math` kasutada funktsioone `sin()` ja `radians()`. Esimene neist leiab nurga siinuse, aga see nurk peab olema radiaanides ja selleks on teine funktsioon. Jälle on abiks for-tsükkel, kus seekord for-i järel oleva muutuja väärtust kasutame nurgaks kraadides.

Lisaks saab vaadata näidet epitsükloidi joonistamisest (`epitsykloid.py`). Kui tead veel mõnda põnevat funktsiooni, siis proovi sarnaselt joonistada.

1.3.5. Joonistamine: variant 3. Terved kujundid

On mõned käsud, millega saab joonistada terve kujundi. Näiteks saab joonistada ringi funktsiooniga `circle(raadius, ulatus, samme)`

```
circle(100)       #joonistab ringi raadiusega 100
circle(100,180)   #joonistab poolkaare, selle ulatus on 180
```

Parameetri `samme` abil saab määrata kui "korralik" ring tuleb, sest ring joonistatakse kui hulknurk ja mida rohkem külgi, seda ilusam. Tasub ka tähele panna, et kilpkonna asukoht ringi joonistamise alguses ei ole mitte ringi keskpunktis, vaid hoopis ringjoone servas. Ja kuidas joonistamine edasi kulgeb, sõltub ka sellest, kuhu poole kilpkonn hetkel vaatab (näiteks kui ta vaatab itta, siis tekib ring kipkonna kohale).

Joonistada saab ka täppe (`dot(diameeter, värv)`). Kusjuures diameeter võib olla üsna suvalise suurusega. Kilpkonna asukoht jääb seekord täpi keskpunktiks.

Näiteks:

```
dot(100, "red")   # joonistab punase ringi (täpi) diameetriga 100
```

Ja teisi kujundite funktsioone otseselt ei olegi. Näiteks ristküliku jaoks oleks vaja tõmmata neli joont ja vahepeal kilpkonna vajalikus suunas pöörata. Kui seda vaja teha palju, on kasulik koostada funktsioon. Ruudu saab ka kasutada ringi joonistamise funktsiooni ja määrata sammude arvuks 4.

Vaata näidet `maja_aiaga.py`.

Maja-aiaga-näites on aga kasutatud veel midagi, millest seni juttu ei ole olnud – nimelt erinevaid värve.

1.3.6. Värvid

Kaks peamist värvi määramise võimalust on:

- nime kasutamine;

- RGB värvimudeli kasutamine.

Värvide nimesid on päris palju (näide `t2pid_v2rvilised.py`);

Näites kasutatud nimed (nn *color string*) on olemas Pythoni värvispetsifikatsioonis. Kui õigeid nimesid teada, on nende abil hea värve määrata, sest nimi on kirjeldav ja koodi lugedes inimesele arusaadav. Kuidas saame näiteprogrammi kasutades teada, kui palju nimelisi värve meie kasutuses on?

Siin on veel üks värvide loetelu:
https://www.wikipython.com/tkinter-ttk-tix/summary-information/colors/

Mis on RGB värvimudel? Millised on põhivärvused ja millised täiendvärvused? Kuidas neid RGB abil määrata saab?

R - punane *(red)*, G - roheline *(green)*, B - sinine *(blue)*. Ja niimoodi kolme värvi abil saabki kirjeldada erinevaid värvuseid.

RGB mudelit saab Turtle moodulis kasutada kahte moodi:

Variant 1: tuleb määrata värvirežiim funktsiooniga `colormode(255)` - sel juhul saab määrata kõik kolm mudeli värvi vahemikus `0 .. 255` ja kirjutada värvide väärtused ükshaaval sulgudesse. Näiteks `fillcolor(255, 0, 0)` annab värviks puhta punase, `fillcolor(255, 0, 255)` aga fuksiinpunase *(magenta)*.

Variant 2: Kolme värvi kombinatsioon esitatakse ühe kuueteistkümnendsüsteemi arvuna, kus kahe positsiooni kaupa on toodud kolme värvi väärtused. Näiteks `fillcolor("#00FFFF")` annab tsüaansinise (*cyan*) tooni (roheline ja sinine on maksimumis ning punane puudub).

Nimede seos RGB-ga: https://www.tcl.tk/man/tcl8.4/TkCmd/colors.htm

Vaatame peamiste värvide jaoks näidet `ringid_ja_v2rvid.py` ning `ringid_ja_v2rvid_viisakam.py`

Värve saab kasutada nii joonte jaoks kui ka mingi kujundi sisu täitmiseks ehk olemas on joone värv ja täidisvärv, nagu tavaliselt graafikas. Saab joonistada kontuuri (jooni) `pencolor()` abil seatud värviga ja lasta kontuur seest värvida värviga, mis on eelnevalt seatud `fillcolor()` abil.

Funktsioonidega `begin_fill()` ja `end_fill()` määratakse värviga täidetav kujund: värvitakse see osa, mis joonistati nende kahe käsu vahel. Näiteks selleks, et saada seest täidetud ristkülik sobivad järgmised koodiread:

```
setpos(200, 200)     # üks nurk paika
pencolor("red")      # punane serva joon
fillcolor("green")   # roheline sisu
begin_fill()         # siit alates algab kujund, mis seest värvitakse
setpos(300, 200)     # joon piki x-telge
setpos(300, 300)     # joon piki y-telge
setpos(200, 300)     # joon piki x-telge
setpos(200, 200)     # joon piki y-telge
end_fill()           # peale seda täidetakse ruut rohelise värviga
```

Loomulikult saab joonistada värvilist riskülikut (ja teisi hulknurki) ka varem vaadatud funktsiooniga.

Joonistamisel kasutatava pliiatsi (kilpkonna saba) jämedust (ehk tekkiva joone paksust) saab määrata funktsiooniga `pensize(joone_paksus)`. Joone paksus on täisarv.

> Ülesanne 8 Täpid rivisse
>
> Joonista üksteise kõrvale 5 punast täppi raadiusega 10 punkti. Siin on abiks funktsioon `dot()`.
>
> Kuidas joonistada üksteise kõrvale kasutaja poolt soovitud arv täppe, mis on kasutaja soovitud raadiusega? Vihje - kasutajalt küsitakse täppide arv ja ühe täpi raadius. Kasutada tuleb tsüklit soovitud arvu täppide saamiseks. Välja saab arvutada, kus on järgmise täpi keskpunkt.

1.4. Lisaks ja lõpetuseks

Kilpkonnafunktsioone on loomulikult veel. Saab tekitada mitu kilpkonna, kes eraldi oma asju ajavad. Muuta joonistamise kiirust, küsida andmeid kilpkonna omaduste, asukoha jms kohta, kustutada jälge, kirjutada joonistamise aknasse teksti ja küsida kasutajalt sisendit (print ja input töötavad käsuaknast) jne. Aga seda kõike võib lisaks uurida Turtle dokumentatsioonist (link kursuse veebis).

Turtle moodulit on käsitletud ka valikkursuse "Rakenduste loomise ja programmeerimise alused" õppematerjalis ning Tartu Ülikooli programmeerimise õpikus (lingid veebilehel).

Ja loomulikult internetist on leitav arvukalt näiteid, videoõpetusi ja muud sarnast.