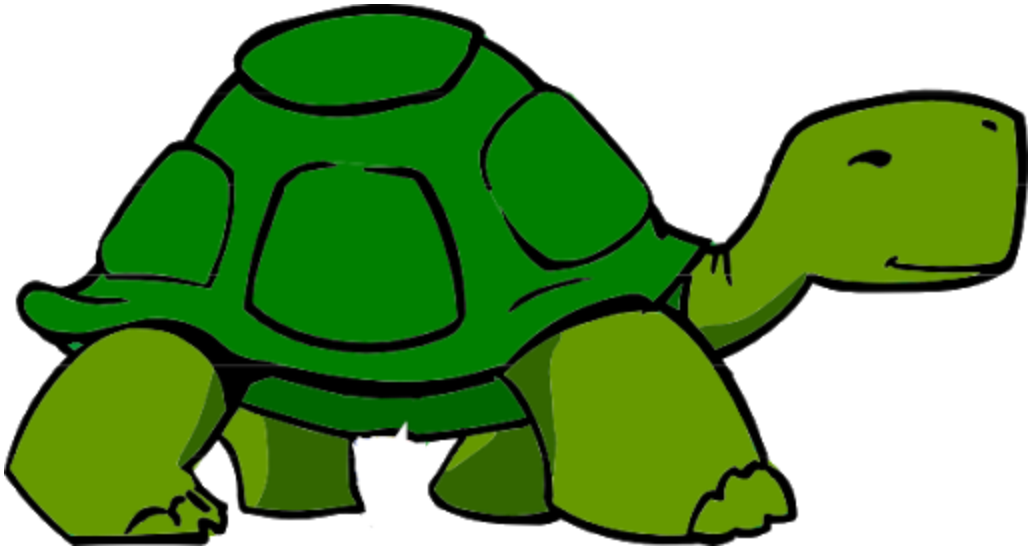
Python Turtle’ õppematerjal



Sten Markus Laht

2020

Sisukord

Sissejuhatus

Tere tulemast kõikidele programmeerimishuvilistele! Käesoleva õppematerjali läbimine annab läbi graafilise väljundi esmase ettekujutuse, mida programmeerimine endast laias laastus kujutab. Et materjali sisust aru saada, võiksite teada, mida tähendavad järgnevad mõisted:

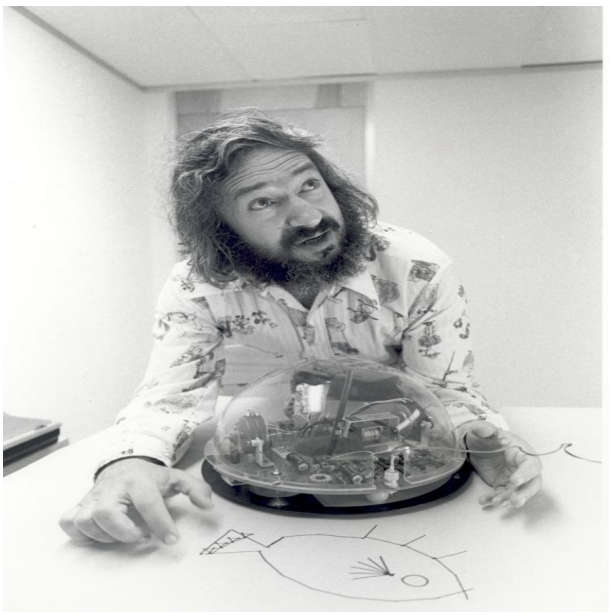
- muutuja
- põhilised andmetüübid (int, float, boolean)

Juhul kui tunnete, et mõisted on ebaselged, siis soovitan vaadata järgnevat lehekülge: http://www.cs.tlu.ee/~inga/progbaas/Materjalid/Python_muutujad_2019.pdf

Käesoleva materjali eesmärk on tutvustada läbi Turtle' mooduli programmeerimise esmaseid põhitõdesid. Samuti on rõhku pandud algoritmilise mõtlemise arendamisele. Aga mida tähendab algoritmiline mõtlemine? Carnegie Melloni ülikooli arvutiteaduse professor Jeannette Wing on välja pakkunud ühe võimaliku selgituse: algoritmiline mõtlemine on mingisuguse probleemi keerukuse vähendamine abstraktset mõtteviisi kasutades ning probleemi ulatuse vähendamine kasutades automatiseerimist. Loodetavasti tunnete end materjali läbi töötades targemini kui enne ning näiteks mõisted for-tsükkel, if-tingimuslause või funktsioon enam hirmu ja segadust ei tekita. Edu ja jõudu õppimisel!

Ajalugu

Kilpkonnagraafika (ingl Turtle graphics) alguseks võib pidada aastat 1967 kui loodi Logo õppeprogrammeerimiskeel. Logot tuntaksegi eelkõige kilpkonnagraafika loomise võimaluse poolest. Aga kuidas 1960. aastatel graafilisi kujundeid programmeeriti? Massachusettsi tehnikaülikooli teadlane Paul Wexelblat ehitas valmis juhtmevaba programmeeritava roboti "Irving", mille põhja külge oli kinnitatud pliiats, mis vastavalt kasutaja suunistele töötas.



Joonis 1. Logo programmeerimiskeele üks autoritest, Seymour Papert koos "kilpkonnaga", nagu taolisi roboteid tol ajal hellitavalt nimetati (Solomon, 1971)

Joonis 2. Lapsed mängimas Irvinguga (Papert 1980)

Algus Turtle'ga ja esimesed käsud

Alustuseks võite avada oma Pythoni interpretaatori. Juhul kui teil seda veel pole, siis soovitan kasutada Thonnyt (https://thonny.org/), mis on Tartu Ülikooli arvutiteadlaste poolt arendatud interpretaator. Aga kui teil on juba mõni muu lemmik, mida kasutate, siis loomulikult pole keelatud ka sellega töötada.

Kui interpretaator on olemas, siis avage see ja saamegi hakata juba otsast koodi kirjutama. Kõige esimese asjana kirjutage esimesele reale **import turtle**. See võimaldab meil kasutada Turtle moodulit ja sellega kaasnevaid käske. Selleks, et ka midagi joonistama hakata, tuleb meil luua kilpkonna objekt ja sellele nimi anda. Nime võite oma suva järgi valida, praegusel juhul kasutan nime kilbu, koodirida näeb välja selline: **kilbu = turtle.Turtle()**. Ja järgmise asjana saamegi juba hakata kilbule käske andma. Tavalise joone tõmbamiseks on Turtle's käsk **forward**, millele saab anda parameetri, kui pikk joon tõmmatakse; praegusel juhul testime näiteks suurust 100, koodireana **kilbu.forward(100).** Selleks, et joone liikumise suunda muuta, on käsud **left** ja ***right***, kuhu saab anda parameetritena kraadid 0st- 360ni. Näiteks **kilbu.left(150)** ja **kilbu.right(25)**. Võite erinevaid nurki iseseisvalt läbi proovida. Joonele värvi määramiseks tuleb enne liikumise algust määrata värv kas ingliskeelse värvi nimega või siis näiteks kuueteistkümnendsüsteemi teisaldatud RGB (red, green, blue) värvidega. Näited: **kilbu.color("yellow")** või **kilbu.color("#9f2121")**. **Ülesanne:** Joonistage T täht ning number 4.

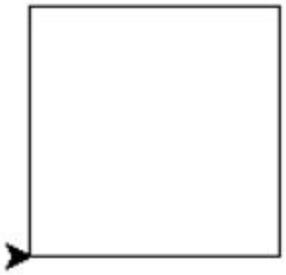
```
import turtle

kilbu = turtle.Turtle()

kilbu.color("#9f2121")
kilbu.forward(100)
kilbu.right(120)
kilbu.forward(200)
kilbu.left(120)
kilbu.forward(100)
```

Joonis 3 ja 4. Z-tähe joonistamine

Esimesed kujundid ja for-tsükkel

Kui olete kilpkonna liikumisele pihta saanud, siis võime edasi liikuda juba erinevate kujundite joonistamise juurde. Proovime esmalt joonistada näiteks võrdkülgset kolmnurka. Kasutame samu käske, mida eelmises peatükiski, ehk **forward** ja **left**. Selleks, et leida, kui palju nurka keerata tuleb, peab jagama täispöörde (360°) kujundi nurkade arvuga, praegusel juhul on vastuseks 120°. Sarnase loogikaga võib programmeerida ükskõik millise võrdkülgse kujundi. Kuid mida rohkem külgi kujundil on, seda rohkem tuleb koodiridu kirjutada ja see muutub ajapikku väga tüütuks ning raskesti loetavaks. Selle probleemi vältimiseks võime kasutada **for**-tsüklit. For-tsükli abil saame määrata mitu korda mingeid koodiridu käivitatakse. For-tsüklil on aga oma süntaks; kui võtame näiteks ruudu, siis näeks for-tsükli algus välja selline: **for i in range(4):**. i tähistab abimuutujat, mille väärtus vaikimisi on 0 ning see suureneb iga kord peale tsükli läbimist ühe võrra, kuni jõuab soovitud väärtuseni, praegusel juhul neljani. For-tsükli puhul on oluline ka treppimine, inglise keeles *indentation*. See tähendab taanet ridade kohta, mis asuvad tsükli sees. Üldiselt kasutatakse selle jaoks *tab* - nuppu, levinud on ka nelja tühiku kasutamine. Taane on vajalik interpretaatorile aru saamiseks, kus mõni tsükkel või funktsioon algab või lõpeb. Lisaks muudab treppimine ka inimese jaoks tsükli ja selle sisu palju loetavamaks.

```
import turtle

kilbu = turtle.Turtle()
i = 0
nurk = 360/4
for i in range(4):
    kilbu.forward(100)
    kilbu.left(nurk)
```



Joonis 4 ja 5. Kood ning ruut

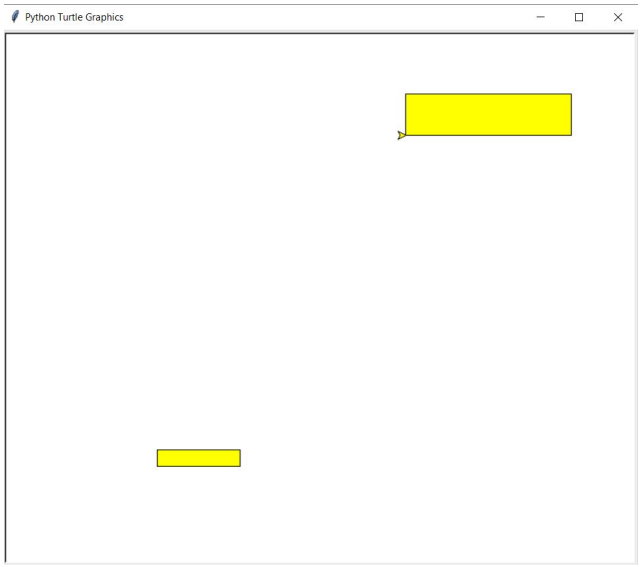
Ülesanne: Joonistada võrdkülgne kaheksanurk.

Asukoha muutmine, funktsioonid ja värvid

Siiamaani oleme joonistanud vaid lõuendi keskele, vaatame nüüd, kuidas asukohta muuta. Selle jaoks on käsk **setpos()**. Ta nõuab kahte argumenti - x ja y. Võitegi lõuendit võtta kui x-y teljestikku. Näiteks käsk setpos(-200, -200) viib asukoha alla vasakusse nurka ning käsk setpos(200, 200) ülesse paremale, kusjuures asukoht (0, 0) on lõuendi keskpunkt. Kui proovite käsku aga käivitada, siis märkate, et asukohta muutudes viiakse ka alguspunktist joon vastavasse punkti. Selle olukorra vältimiseks on käsud **penup()** ja **pendown()**; penup tõstab "pastaka" lõuendilt ülesse ja pendown paneb ta uuesti lõuendile.

Edasi vaatame funktsioone. Funktsioonid aitavad meil koodi arusaadavamaks muuta ning liigset koodiridade kopeerimist vältida. Pythonis defineeritakse funktsioone nii: **def funktsiooniNimi()**:. Testime funktsiooni koostamist ja väljakutsumist eelmise peatüki ruudu näite põhjal. Paneme ruutu joonistava for-tsükli funktsiooni sisse (NB! Treppimine toimub funktsioonis samamoodi, kui for-tsüklis). Veidi värvi lisamiseks saame kasutada käske **begin_fill()** ja **end_fill()**, mis täidavad loodud kujundi piirjooned värviga. Begin_fill() kirjutatakse enne kujundi joonestamise algust ning end_fill() kirjutatakse peale piirjoonte tõmbamist lõppu. Et täitevärvi määrata, võime kasutada eelpool mainitud, käsku color, mille esimeseks argumendiks on joone värv ning teiseks täitevärv. Kui veel funktsioonide juurde tagasi tulla, siis saab ka nende enda defineerimisel parameetreid määrata. Kui võtame näiteks ristküliku, siis oleks mõeldav funktsiooni defineerimisel kindlaks määrata ristküliku pikkus ja laius. Vaadake allolevat näidet:

```
import turtle

def ristkylik(pikkus, laius):
    kilbu.color("black", "yellow")
    kilbu.begin_fill()
    for i in range(2):
        kilbu.forward(pikkus)
        kilbu.left(90)
        kilbu.forward(laius)
        kilbu.left(90)
    kilbu.end_fill()

kilbu = turtle.Turtle()

kilbu.penup()
kilbu.setpos(-200, -200)
kilbu.pendown()

ristkylik(100, 20)

kilbu.penup()
kilbu.setpos(100, 200)
kilbu.pendown()

ristkylik(200,50)
```



Joonised 5 ja 6. Kood ning ristkülikud

Näide. Lihtsa maja joonestamine

Maja joonestamise näite riputasin veebi üles, saate sealt endale koodi kopeerida ja läbi mängida. Maja näide on üles ehitatud sellele, mida eelmistes osades vaatasime ja õppisime. Programmi koodis on loodud 5 erinevat funktsiooni: **seinad()**, **katus()**, **aknad()**, **uks()** ja **maja().** Nagu nimede järgi arvata võite, on funktsioonide eesmärk koostada eraldi valmis maja osad. Koodis on aga üks uus moodul ja paar uut käsku Turtle'st, mille peab lahti seletama. Esmalt lisasin koodi algusesse rea **import math**, mis on vajalik matemaatiliste funktsioonide kasutamiseks. Esimeses funktsioonis seinad luuakse lihtsalt tavaline ristkülik. Funktsioonis katus liigutatakse "pliiatsiga" ristküliku ülemisse nurka, katuse moodustab täisnurkne kolmnurk, kus kaks kaatetit on võrdse pikkusega. Need saadakse hüpotenuusi jagamisel √2-ga. Ruutjuure võtmiseks on kasutusel funktsiooni **sqrt()**. Funktsioonis *aken* joonistatakse kahe for-tsükli sees kaks ruutu, mis on mõlemad paigutatud **setpos()** käsuga vastasseintest ühekaugele. Funktsioonis uks joonistatakse maja keskele püstine ristkülikukujuline uks. Funktsioon maja kutsub välja kõik eelnevad funktsioonid.

```
import turtle
import math

def seinad():
    kilbu.color("black", "brown")
    kilbu.begin_fill()
    kilbu.penup()
    kilbu.setpos(-120,0)
    kilbu.pendown()
    for i in range(2):
        kilbu.forward(300)
        kilbu.left(90)
        kilbu.forward(150)
        kilbu.left(90)
    kilbu.end_fill()
```

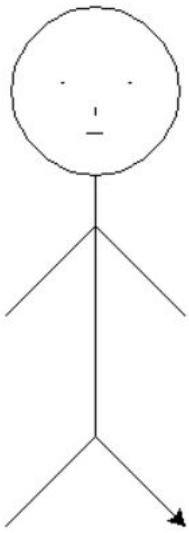
```
def katus():
    kilbu.color("black", "grey")
    kilbu.begin_fill()
    kilbu.penup()
    kilbu.setpos(-120, 150)
    kilbu.pendown()
    kilbu.left(45)
    kilbu.forward(300/math.sqrt(2))
    kilbu.right(90)
    kilbu.forward(300/math.sqrt(2))
    kilbu.setheading(180)
    kilbu.forward(300)
    kilbu.end_fill()
```

```
def aknad():
    kilbu.setheading(90)
    kilbu.penup()
    kilbu.setpos(-90, 50)
    kilbu.pendown()
    kilbu.color("black", "white")
    kilbu.begin_fill()
    for i in range(4):
        kilbu.forward(30)
        kilbu.right(90)
    kilbu.penup()
    kilbu.setpos(150, 50)
    kilbu.pendown()
    for i in range(4):
        kilbu.forward(30)
        kilbu.left(90)
```

```
def uks():
    kilbu.penup()
    kilbu.setpos(12, 0)
    kilbu.pendown()
    for i in range(2):
        kilbu.forward(70)
        kilbu.right(90)
        kilbu.forward(30)
        kilbu.right(90)

def maja():
    seinad()
    katus()
    aknad()
    uks()

kilbu = turtle.Turtle()

maja()
```

Joonised 7, 8, 9 ja 10. Kood

Ülesanne: Joonestada sarnase loogika alusel kriipsujuku.



Joonis 11. Näide kriipsujukust

If-lause ja while-tsükkel, sisend

Käesolevas peatükis teeme tutvust if-lause ja while-tsükliga ning sisendiga. Vaatame näite abil läbi, kuidas tsüklid, tingimuslaused ning sisend toimivad. Vaatleme näidet (kuvatõmmised asuvad peatüki lõpus), kus programm küsib kasutajalt, millist kujundit ta soovib, et lõuendile joonestatakse. Programmi algusesse on loodud boolean tüüpi muutuja **edasi**, millele on määratud vaikeväärtus **True**. While-tsükli eesmärk on endas sisalduvaid koodiridu käivitada nii kaua, kuniks tema töötamise tingimus veel kehtib, praegusel juhul on selleks tingimus, et muutuja edasi väärtus on True (võrdust kontrollitakse Pythonis kahe kõrvutioleva võrdusmärgiga). Tsükli seest leiame, et muutuja kujund saab oma väärtuse **input** funktsioonilt. Funktsioon kuvab terminali küsimuse, millele kasutajal on võimalik vastata. Edasi märkame if-lauset, mis koosneb if tingimusest elif tingimusest ning else tingimusest. Mida need tähendavad? If tingimus kontrollib endas defineeritud tingimust, kas muutuja kujund on võrdne stringiga “kolmnurk”. Juhul kui mitte, siis liigutakse edasi elif tingimuste juurde. Juhul kui ka nende töötamise tingimus pole täidetud, minnakse else tingimusse, mis käivitub siis kui ükski eelnevatest tingimustest polnud täidetud.

Mõtlemine: Miks ei tööta while-tsükkel korrektselt, kui asendada elif-id if-idega? Teoreetiliselt täidavad nad sama ülesannet - kontrollivad, kas neis defineeritud tingimus on täidetud ja tegutsevad vastavalt sellele?

Kui kasutaja kirjutab sisendiks “välju”, muutub muutja edasi väärtus **False**’ks, while-tsüklist väljutakse ning programm lõpetab oma töö. Kui kasutaja annab sisendiks väärtuse, mille olemasolu if-idega ning elif-idega kontrollitud pole, liigutakse else lausesse ning kasutajale öeldakse, et sellist kujundit ei saa joonistada.

```
import turtle

kilbu = turtle.Turtle()

edasi = True

while edasi == True:

    kujund = input("Tere! Millist kujundit soovite joonistada? [kolmnurk/ruut/kuusnurk/välju]")
    if kujund == "kolmnurk":
        for i in range(3):
            kilbu.forward(100)
            kilbu.left(120)
    elif kujund == "ruut":
        for i in range(4):
            kilbu.forward(100)
            kilbu.left(90)
    elif kujund == "kuusnurk":
        for i in range(6):
            kilbu.forward(100)
            kilbu.left(60)
    elif kujund == "välju":
        print("Aitäh kujundeid joonistamast!")
        edasi = False
    else:
        print("Sellist kujundit ei leitud!")
```

Joonis 12. Kood

Huvitavamad käsud

Lisan veel siia paar huvitavamat käsku Turtle'st, lisa saate lugeda Turtle' dokumentatsioonist.

- **pensize()** - lubab muuta "pastaka" suurust.
- **bgcolor()** - muudab taustavärvi.
- **bgpic()** - lubab taustaks valida pildi kohalikust masinast.
- **circle()** - joonistab ringi, esimene argument on ringi raadius, valikuliselt võite teiseks lisada, kui suures ulatuses ringi soovite(kraadides, näiteks 180 joonistab poolringi); kolmanda argumendiga võite kujundi muuta nurgeliseks, ehk võite lisada nurkade arvu (täisarv).
- **speed()** - muudab joonestamise kiirust, argumendid 0-10.
- **backward() -** joon tõmmatakse vastassuunas praegusele suunale, argumendiks joone pikkus.

Kasutatud kirjandus

1. 1969 – The Logo Turtle – Seymour Papert et al (Sth African/American)(2010, 10. jaanuar). Loetud aadressil: http://cyberneticzoo.com/tag/paul-wexelblat/
2. Logo (programming language). (kuupäev puudub). *Wikipedia*. Loetud 16. märts 2020 aadressil https://en.wikipedia.org/wiki/Logo_(programming_language)
3. Turtle graphics. (kuupäev puudub). *Wikipedia*. Loetud 16. märts 2020 aadressil https://en.wikipedia.org/wiki/Turtle_graphics