4.1 Sõned

JUTUMÄRGID VÕI ÜLAKOMAD?

Andmetüüpide juures rääkisime põgusalt ka andmete tekstilisest esitamisest ehk **sõnedest**. Oli ka juttu, et sõne esitatakse jutumärkide (või ülakomade) vahel ja sõne ingliskeelne vaste on *string* ning lühend *str*.

Vaatleme nüüd sõnesid lähemalt. Nagu öeldud, võib sõne panna ka jutumärkide või ülakomade vahele. Näiteks saab sõne esitada nii "Sisenesid pangaautomaati!" kui ka 'Sisenesid pangaautomaati!'. Küll aga ei saa sõne ees olla üks märk (nt jutumärk) ja järel teine (nt ülakoma). Tuleks panna tähele, et jutumärgid ja ülakomad ise ei kuulu sõne sisu juurde. Seda saame proovida *print* käsuga:

- **Esimene võimalus**

```
print("Sisenesid pangaautomaati!")
```

- **Teine võimalus**

```
print('Sisenesid pangaautomaati!')
```

Mõlema võimaluse puhul väljastatakse ekraanile samasugune tekst:

```
>>>
Sisenesid pangaautomaati!
```

Milleks need jutumärgid ja ülakomad on vajalikud? Neid on vaja selleks, et muidu võib Python sõned segamini ajada näiteks muutujanimede või arvudega. Proovige näiteks eelmist käsku ilma ülakomadeta. Mis juhtub?

Seejuures väärib märkimist, et jutumärkide või ülakomade ümberpanemine muutujanimele ei muuda selle muutuja väärtust sõneks. Vastava sõne saamiseks on funktsioon `str()`.

Ülesanne

Mis on muutuja sõneA väärtus?

```
arvA = 17
sõneA = "arvA"
```

Vali 17

Vali "17"

Vali arvA

Vali "arvA"

Tegelikult saab programmeerimisel üsna sageli sama tulemuse saamiseks erinevaid võimalusi kasutada. Iseküsimus on, kui põhjalikult neid erinevaid võimalusi saab ja tuleb õpetamisel välja tuua. Sellel kursusel otsustasime mõlemaid variante (nii jutumärke kui ülakomasid) küllaltki põhjalikult tutvustada. Kui ülesande tekstis ei ole eraldi märgitud, siis võib ülesannete lahendamisel ükskõik kumba varianti kasutada.

AGA KUI TEKST SISALDAB JUBA JUTUMÄRKE VÕI ÜLAKOMASID?

Asi läheb veidi keerulisemaks, kui sõne sees on vaja kasutada jutumärke, ülakomasid või muid erisümboleid. Järgnevalt demonstreerime erinevaid viise selle probleemi lahendamiseks.

- **1. võimalus.** Kui tekstis on ülakomasid, siis kõige lihtsam on kasutada piiritlejaks jutumärke ja vastupidi:

```
print("Rock 'n' roll")
print('Jim ütles vaid: "Siin see on."')
```

Ülesanne

Mis väljastatakse ekraanile?

```
print("Jim ütles vaid: "Siin see on."")
```

Vali Jim ütles vaid: "Siin see on."

Vali Jim ütles vaid:

Vali Veateade

- **2. võimalus.** Kui tekstis on vaja kasutada nii jutumärke kui ka ülakomasid, siis pole eelmisest soovitusest abi. Sellisel juhul tuleb üks neist (nt jutumärgid) ikkagi valida piiritlejaks. Kui neid on aga vaja tekstis kasutada, siis tuleb kasutada langkriipsu jutumärkide ees. See annab Pythonile märku, et tegemist pole veel teksti lõpuga, vaid sooviti kirja panna piiritlejaks valitud sümbolit ennast.

```
print("Jack vastas: \"Rock 'n' roll\".")
print('Jack vastas: "Rock \'n\' roll".')
```

Mõlemal juhul väljastatakse ekraanile tekst:

```
Jack vastas: "Rock 'n' roll".
>>>
```

(Juhime tähelepanu sellele, et langkriips \ ja kaldkriips / on erinevad märgid.)

Langkriipsu saab kasutada ka muul otstarbel, nt reavahetusi saab esitada kombinatsiooniga *n*.

```
print("Seda kuupaistet!\nOh muutuksin sündides\nmänniks mäetipul!\n-
Ryota")
```

Tulemus:

```
>>>
Seda kuupaistet!
Oh muutuksin sündides
männiks mäetipul!
-Ryota
```

Nagu näha, on langkriips spetsiaalse tähendusega. Kuidas aga esitada langkriipsu ennast? Lihtne, see tuleb ära märgistada ... langkriipsuga!:

```
print("C:\\kaustanimi\\failinimi.txt")
```

Tulemus:

```
>>>
C:\kaustanimi\failinimi.txt
```

Ülesanne

Mis väljastatakse ekraanile? (Väljatrükk on nupu "Vali" all.)

```
print("Minu lemmikraamatud on \n\"Kevade\" ja \n\"Rehepapp\"")
```

Vali

Minu lemmikraamatud on \n\"Kevade\" ja \n\"Rehepapp\"

Vali

Minu lemmikraamatud on
"Kevade" ja
"Rehepapp"

Vali

Minu lemmikraamatud on n"Kevade" ja n"Rehepapp"

Vali

Minu lemmikraamatud on
Kevade ja
Rehepapp

Vali Veateade

- **3. võimalus**. Kui tekstis on vaja kasutada palju reavahetusi, ülakomasid või jutumärke, siis võib tulemus muutuda kõigi nende *n*, \' või \" tõttu väga kirjuks. Seetõttu on Pythonis veel üks sõne kirjapaneku viis – kolmekordsete ülakomade või jutumärkide vahel saab vabalt kasutada tavalisi reavahetusi, ülakomasid ja jutumärke. Katsetage iseseisvalt:

```
print("""Jack vastas: "Rock 'n' roll".""")
print('''Jack vastas: "Rock 'n' roll".''')
print("""Seda kuupaistet!
Oh muutuksin sündides
männiks mäetipul!
--Ryota""")
print("""
   _____
  / ____|
 | |   __  __ _ _ __ ___    ___    _____   _____ _ __
 | | |_ |/ _` | '_ ` _ \ / _ \  / _ \ \ / / _ \ '__|
 | |__| | (_| | | | | | |  __/ | (_) \ V /  __/ |
  \_____|\__,_|_| |_| |_|\___|  \___/ \_/ \___|_|
""")
```

Ülesanne

Mis väljastatakse ekraanile?

```
print("""Rock 'n' rolli Jim ütles vaid: "Siin see on." """)
```

Vali Rock 'n' rolli Jim ütles vaid: "Siin see on."

Vali "" Rock 'n' rolli Jim ütles vaid: "Siin see on." ""

Vali Veateade

TEHTED SÕNEDEGA

Sõnedega saab teha erinevaid tehteid. Me oleme neid juba liitnud. See käis plussmärgi abil.

```
"Tere " + "hommikust"
```

Lisaks on olemas funktsioonid, mis "ühendatakse" sõne külge punkti abil. (Neid nimetatakse ka meetoditeks.) Nende funktsioonide nime lõppu pannakse samuti sulud, mis võivad olla tühjad või mittetühjad. Selliste funktsioonidega saab palju huvitavat ja kasulikku korda saata!

```
print("tartu".capitalize())
print("Tartu".endswith("tu"))
print("Tartu".lower())
print("Tartu".upper())
print("Tartu".startswith("tu"))
print("Kauneim linn on Eestis Tartu".title())
```

Katsetage seda programmi ja püüdke aru saada, mida need funktsioonid teevad.

Funktsioonid töötavad ka muutujatega, mille väärtuseks on sõne.

```
linn = "Tartu"
print(linn.lower())
```

Kõikide sõnemeetoditega saab tutvuda
aadressil http://docs.python.org/3/library/stdtypes.html#string-methods.

Sõne koosneb sümbolitest ja neid saab eraldi käsitleda. Näiteks tavamõttes esimene sümbol on indeksiga 0.

```
linn = "Tartu"
print(linn[0])
```

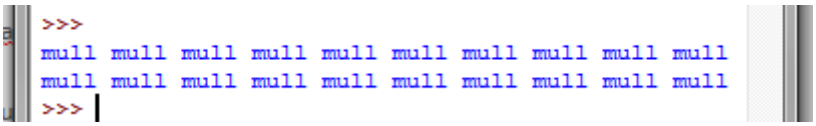
Leedu perenimede kontrollülesandes juba tegelikult sümbolite abi kasutasime ka, aga põhjalikumalt räägime neist järgmisel nädalal.

KORRUTADA SAAB KA

Sõnedega saab veel erinevaid toredaid asju teha. Näiteks saab sõnesid "korrutada".

```
lause = "mull " * 20
print(lause)
```

Tulemuseks saame:



Ülesanne

Mida väljastatakse ekraanile?

```
a = "1"
b = 1
print("Arvud on " + 5 * a + " ja " + str(5 * b))
```

Vali Arvud on "5" ja 5

Vali Arvud on 5 ja "5"

Vali Arvud on 5 ja 11111

Vali Arvud on 11111 ja 5

Vali Veateade

VÄLJUND

Programmis tuleb aeg-ajalt suhelda kasutajaga ehk tegeleda sisendi ja väljundiga. Me oleme eespool funktsioone `input` ja `print` juba kasutanud, aga räägime nüüd just viimasest põhjalikumalt. Nagu olete juba õppinud, saab programmis väärtusi ekraanile kuvada käsuga *print*. Nüüd uurime seda käsku veidi lähemalt.

Vaatame näidet:

```
print(32 * 57)
```

See, mis ekraanile kuvatakse, antakse ette sulgudes - argumendina. Eelmises näites on 32 * 57 käsu *print* argumendiks. Kui kõik läheb ilusti, siis programm kuvab ekraanile 1824 ja lõpetab töö. Funktsioonid ja nende argumendid on programmeerimises väga tähtsal kohal. Põhjalikumalt käsitleme neid kahe nädala pärast. Oleme neid aga kasutanud ja kasutame edasi juba praegu.

Käsule *print* võib ette anda ka mitu argumenti, sel juhul trükitakse ekraanile samale reale mitu asja järjest, kusjuures need asjad on eraldatud tühikutega. Järgnev näide demonstreerib kahte samaväärset viisi, kuidas trükkida ekraanile mitu infokildu korraga. Esimene variant kombineerib komponendid kõigepealt üheks sõneks ja kasutab seda *print*-i argumendina, teine variant annab kõik komponendid eraldi argumentidena.

```
eesnimi = "Peeter"
perenimi = "Paan"
vanus = 21
print(eesnimi + " " + perenimi + " vanus: " + str(vanus))


eesnimi = "Peeter"
perenimi = "Paan"
vanus = 21
print(eesnimi, perenimi, "vanus:", vanus)
```

Mõlemad programmid väljastavad ekraanile:

```
>>>
Peeter Paan vanus: 21
```

Eraldi argumentidega variant on küll lühem kirja panna (eriti mugav on see, et arve ei pea ise *str* käsuga sõneks teisendama), aga mõnikord see siiski ei sobi, näiteks siis, kui me ei soovi väljundis argumentide vahele tühikut. Niisiis on esimene variant pisut paindlikum, kuid teine pisut lihtsam (lihtsam nii kirjutada kui ka hiljem lugeda).

Seesuguseid dilemmasid tuleb programmeerijatel sageli ette: kas valida lihtsam või paindlikum tee? Kuidas nende vahel kompromissi leida? Ühe poole püüeldes jääb teine tihti unarusse.

Ülesanne

Mis väljastatakse ekraanile? (Väljatrükk on nupu "Vali" all.)

```
nimi1 = "Otto"
nimi2 = "Triin"
print("Tere,",nimi1)
print(nimi2 + "!")
```

Vali

Tere,OttoTriin!

Vali

Tere, Otto Triin!

Vali

Tere,Otto
Triin!

Vali

Tere, Otto
Triin!

Vali Veateade

(Otto-Triin oli üheksakümnendate aastate keskel lastesaate tegelane.)

4.2 Graafikaraamistik *tkinter*. Tahvel

TK JA TKINTER

Pythoni graafiliste võimalustega oleme juba kokkupuutunud seoses kilpkonnaga (moodul *turtle*). Sel nädalal tegeleme Pythoni graafikaraamistikuga **tkinter**. Eesmärke on seejuures mitu. Ühelt poolt annab see võimaluse saada graafilise kujundusega programme - on ju enamik igapäevaselt kasutatavaid programme graafilised. Teiselt poolt tutvume natuke teistmoodi programmeerimisstiiliga.

Moodul *tkinter* (koos alammooduliga tkinter.ttk) kuulub Pythoni standardsete moodulite hulka ja töötab erinevatel operatsioonisüsteemidel. Nimetatud moodulid põhinevad levinud teegil nimega **Tk**, mida kasutatakse ka teistes programmeerimiskeeltes.

Käesolev materjal põhineb *tkinter*i versioonil 8.5, mis on Python 3 (ja ka Thonnyga) kaasas. Selles versioonis tehtud kasutajaliidesed on vastava operatsioonisüsteemi (Windows, Mac OS, Linux) konkreetsele platvormile omase välimusega.

Tkinter on väga suurte võimalustega, millest siin vaatleme vaid väikest osa. Paraku ei ole head ühist dokumentatsiooni, kust kõike vaadata saaks. Mõned materjalid sisaldavad siiski päris palju infot ka.

- http://www.tkdocs.com/ – hea materjal Tk ja tkinter-i põhimõtete õppimiseks.

Käesolevas osas tutvume *tkinteri* abil tehtavate programmide üldise ülesehitusega ja võimalusega teha pilte tahvlil (*canvas*). Nuppude, siltide jt kasutajaliidese võimalustega tegeleme osas 4.3.

ESIMENE GRAAFIKAPROGRAMM

Harilik graafikaaken koosneb:

- **raamist**
- **tahvlitest** ja
- **atomaarsetest komponentidest**.

Raam kujutab endast tavaliselt akna põhimikku, mille ülemises ääres on tiitliriba. Erirolli mängib *juurraam*, mis algatab tkinteri elutsükli. Nii raam kui ka tahvel on *konteinerid*, mille sisse on võimalik paigutada teisi graafikaelemente. Nii võime näiteks lisada raamile mõne omaloodud tahvli ning sellele omakorda atomaarseid elemente. Viimased kujutavad endast joonise nähtavaid komponente, näiteks programmide kasutajaliidestest tuttavad nupud, sildid ning muud taolised elemendid. Lisaks sellele on võimalik tahvlile otse mitmesuguseid graafilisi kujutisi joonistada.

Koostame programmi, mis loob ja kuvab juurakna.

Alustame sellest, et impordime kõik moodulis tkinter olevad klassid ja funktsioonid:

```
from tkinter import *
```

Siis loome juurakna. Raami tiitliks paneme "Tere, maailm!":

```
raam = Tk()
raam.title("Tere, maailm!")
```

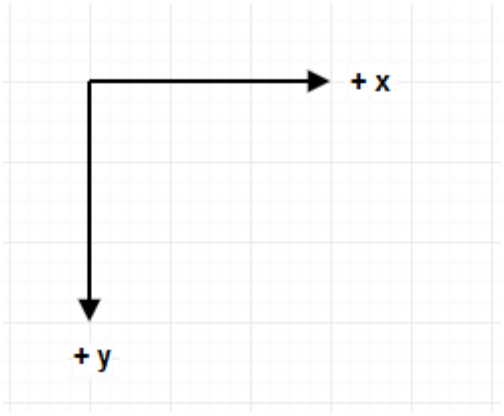
Pärast graafiliste elementide loomist tuleb siseneda Tkinteri põhitsüklisse — vastasel juhul ei toimu ekraanil midagi:

```
raam.mainloop()
```

Olles selle tsükli sees, teostab programm raamis olevate elementide paigutamist ning jälgib kasutaja sisendit. Programm töötab tsüklis kuni põhiraami sulgemiseni.

TAHVEL (CANVAS)

Tahvel kujutab endast ristkülikukujulist ala, mille peale saab joonistada graafilisi kujutisi. (Sõna *canvas* üks vasteid on *lõuend*, mis ehk olekski mõneti täpsem - lõuendile ju joonistataksegi. Mitmetes materjalides on siiski juurdunud vasteks *tahvel*.) Paneme tähele, et koordinaadistiku alguspunkt asub tahvli ülemises vasakus nurgas ning x-koordinaat suureneb vasakult paremale ja y-koordinaat ülevalt alla. Mõõtühikuteks on pikslid (ehk ekraanitäpid) ja seega koordinaatide väärtusteks on täisarvud.



Lisame nüüd tühja tahvli raamile loomist:

```
from tkinter import *
raam = Tk()
raam.title("Tühi tahvel")
# loome tahvli laiusega 600px
tahvel = Canvas(raam, width=600)
# paigutame tahvli raami ja teeme nähtavaks
tahvel.pack()
# siseneme põhitsüklisse
raam.mainloop()
```

Peale laiuse võib tahvlile ette anda mitmeid teisi argumente. Nende loetelu kuvatakse, kui küsida dokumentatsiooni tahvli kohta (ehk kirjutada käsurealt):

```
>>> print(Canvas.__init__.__doc__)
```

Kopeerige ja käivitage ülalolev tühja tahvliga raami programm.

Täiendage tahvli loomise käsku, muutes

- tahvli taustavärvi (nt `background="red"`);
- tahvli kõrgust (*height*).

Tahvlile saab lisada mitmesuguseid graafikaelemente: jooni, kaari, pilte, ringe, hulknurkasid, ristkülikuid ja teksti. Iga sellise objekti loomiseks on olemas vastav käsk (mis algab *create_*), millele antakse ette kujundi positsioon ja stiil. Vaatleme mõnda neist lähemalt.

Murdjoon, mille tippudeks on (x0, y0), (x1, y1), ..., (xn, yn), joonistatakse `create_line(x0, y0, x1, y1, ..., xn, yn, option, ... )` abil.

Näited:

```
# üks horisontaalne sirglõik
tahvel.create_line(50,50,100,50)

# horisontaalne sirglõik ja vertikaalne sirglõik
tahvel.create_line(50,150,100,150,100,200)

# sirglõik paksusega 4px
tahvel.create_line(50,100,100,250, width=4)

# rohelist värvi nool paksusega 4px
tahvel.create_line(50,350,100,350, width=4, fill="green",
arrow=LAST)
```

Argumendi väärtus *arrow=LAST* kujundab joone lõppu nooleotsa.

Ristküliku, mille ülemise vasakpoolse tipu koordinaadid on (x0, y0) ja alumise parempoolse tipu koordinaadid on (x1, y1), saame `create_rectangle (x0, y0, x1, y1, option, ... )` abil.

Näited:

```
# seest tühi ristkülik (ruut) mustade servadega
tahvel.create_rectangle(150,50,200,100)

# seest tühi ristkülik kollaste servadega, mille paksus on 2px
tahvel.create_rectangle(150,150,200,200, width=2, outline="yellow")

# mustade servadega roheline ristkülik
tahvel.create_rectangle(150,250,200,300, fill="green")
```

Hulknurga tippudega (x0, y0), (x1, y1), ... saame `create_polygon(x0, y0, x1, y1, ..., option, ...)` abil.

Näited:

```
# mustade servadega punane kolmnurk
tahvel.create_polygon(150,350,150,400,200,375, fill="red",
outline="black")
```

Ovaali joonistab `create_oval(x0, y0, x1, y1, option, ... )`.

Ovaal on mõtteliselt piiratud ristkülikuga, mille ülemise vasakpoolse tipu koordinaadid on (x0, y0) ja alumise parempoolse tipu koordinaadid on (x1, y1). Ovaali joonistamisel peate seega arvestama ristkülikuga, mis seda ümbritseb.

Näited:

```
# roheliste servadega punane ovaal(ring)
tahvel.create_oval(10,10,100,100, fill="red", outline="green")
```

Teksti saame tahvlile `create_text(x0, y0, option, ... )` abil. Tekst tuleb ette anda argumendiga `text`. Teksti koht sõltub koordinaatidest (x0, y0) ning argumendist `anchor`. Vaikimisi on punkt (x0, y0) teksti keskel.

Näited:

```
# musta värvi tekst "Tere!"
tahvel.create_text(50,50, text="Tere!")

# sinist värvi teksti "Tere!" alumine vasak punkt on (50, 50)
tahvel.create_text(50,50, text="Tere!", anchor=SW, fill="blue")
```

Argumendi `anchor` teised võimalikud väärtused on CENTER ja ilmakaartele vastavad N, NE, E, SE, S, W ja NW.

TSÜKLID

Kui joonistusel on korduvaid elemente, siis võib olla abi tsüklitest. Teeme näiteks Haapsalu ja Sillamäe lipud.

```
from tkinter import *

raam = Tk()
raam.title("Haapsalu lipp")
# loome tahvli laiusega 600px ja kõrgusega 300 px
tahvel = Canvas(raam, width=600, height = 300)
# ühe joone kõrgus
kõrgus = 100
# tsüklimuutuja
i = 0
while i < 3:
    # esimene ja kolmas joon
    if i == 0 or i == 2:
    #if i % 2 == 0:
        # sinine väike ristkülik
        tahvel.create_rectangle(0, i * kõrgus, 200, (i + 1) *
kõrgus, fill="blue", outline="blue")
        # valge suur ristkülik
        tahvel.create_rectangle(200, i * kõrgus, 600, (i + 1) *
kõrgus, fill="white", outline="white")
    # teine joon
    else:
        # valge väike ristkülik
        tahvel.create_rectangle(0, i * kõrgus, 200, (i + 1) *
kõrgus, fill="white", outline="white")
        # sinine suur ristkülik
        tahvel.create_rectangle(200, i * kõrgus, 600, (i + 1) *
kõrgus, fill="blue", outline="blue")
    # tsüklimuutuja suurendamine
    i += 1
# paigutame tahvli raami ja teeme nähtavaks
tahvel.pack()
# siseneme põhitsüklisse
raam.mainloop()
```

```
from tkinter import *

raam = Tk()
raam.title("Sillamäe lipp")
# loome tahvli laiusega 880px ja kõrgusega 560px
tahvel = Canvas(raam, width=880, height = 560)

#sinine taust
tahvel.create_rectangle(0, 0, 880, 560, fill="blue", outline="blue")

# tsüklimuutuja
i = 0
while i < 5:
    # kollane ristkülik vasakul
    tahvel.create_rectangle(0+i*80, 560-3*70-i*70, 2*80+i*80, 560-
i*70, fill="yellow", outline="yellow")
    # kollane ristkülik paremal
    tahvel.create_rectangle(880-2*80-i*80, 560-3*70-i*70, 880-
i*80, 560-i*70, fill="yellow", outline="yellow")
    i += 1

#tipp
tahvel.create_rectangle(880-6*80, 0, 880-5*80, 70, fill="yellow",
outline="yellow")

# paigutame tahvli raami ja teeme nähtavaks
tahvel.pack()
# siseneme põhitsüklisse
raam.mainloop()
```

Vaata ka näidislahenduse videot: http://uttv.ee/naita?id=23946

LISALUGEMINE. FUNKTSIOONI GRAAFIK

Ülesannete lahendamiseks pole seda materjali tarvis tingimata läbida.

Koostame programmi, mis joonistab mõne teie poolt valitud (matemaatilise) funktsiooni (nt $y=x^3$) graafiku.

Kuigi Tkinter sobib hästi graafikute joonistamiseks, tekitab mõningast ebamugavust teistmoodi koordinaatide süsteem – oleme ju harjunud, et y kasvab ülespoole, mitte aga alla ja koordinaatide alguspunkt on meil ka harjumuspäraselt olnud pigem joonise keskel. Võtame abiks klassi Canvas funktsiooni `move`, mis võimaldab tahvlil olevaid objekte horisontaalset ja vertikaalset telge mööda ümber tõsta. Seega paigutame kõik objektid harilikku koordinaadistikku ja siis rakendame funktsiooni `move`. Peegelduse x-telje suhtes korraldame sellega, et funktsiooni väärtuste kogumisel kogume y asemel -y. Näiteprogramm püüab teha y=x graafikut:

```
from tkinter import *
raam = Tk()
```

```
# tahvli laius
w = 500
# tahvli pikkus
h = 500
tahvel = Canvas(raam, width=w, height=h, bg="white")

# vertikaalne telg
tahvel.create_line(0, h/2, 0, -h/2, arrow=LAST)
# horisontaalne telg
tahvel.create_line(-w/2, 0, w/2, 0, arrow=LAST)

# joonistame lõigud (x1,f(x1)) - (x2,f(x2))
x1 = -w//2
while x1 < w//2:
    x2 = x1+1
    # olgu alguseks lineaarne funktsioon
    y1 = x1
    y2 = x2
    # -y on selleks, et peegeldada x-telje suhtes
    tahvel.create_line(x1, -y1, x2, -y2)
    x1 += 1

# nihutame kõik objektid 250px võrra paremale ja alla
tahvel.move(ALL, w/2, h/2)

tahvel.pack()

raam.mainloop()
```

Kopeerige ja käivitage ülalolev kood. Katsetage programmi ka teiste funktsioonidega (nt 5x, x**3, ...)

LISALUGEMINE. LIIKUVAD KUJUTISED

Ülesannete lahendamiseks pole seda materjali tarvis tingimata läbida.

Proovime joonistada osutitega kella, mis ennast aja jooksul värskendaks. Võrreldes eelmiste ülesannetega, kus tegemist oli sisuliselt staatiliste kujutistega, on meie praeguseks eesmärgiks uurida, kuidas võib muuta graafikaobjektide olekuid rakenduse töö ajal. Graafikaobjektide loomisel saab neile anda unikaalseid nimesid, mille järgi saab need hiljem tahvlil üles leida, nt

```
id = tahvel.create_line(x0,y0,...,xn,yn)
```

Kasutades nime saab näiteks objekti kustutada, nihutada või muuta tema argumente. Objektidega manipuleerimiseks kasutame klassis Canvas defineeritud meetodeid.

kustutamine

```
tahvel.delete(id)
```

```
# nihutamine
tahvel.move(id, x, y)

# objekti argumentide tagastamine
tahvel.itemcget(id, "width")

# koordinaatide uuendamine
tahvel.coords(id, x0,y0,...,xn,yn )
```

Antud ülesande kontekstis huvitab meid põhimõtteliselt viimane meetod, mille abil me saame osutite positsiooni uuendada. Tekitame uue raami ja tahvli. Kella keskpunkt olgu tahvli keskel.

```
from tkinter import *
raam = Tk()
raam.title("Kell")
# tahvli laius
w = 500
# tahvli pikkus
h = 500

tahvel = Canvas(raam, width=w, height=h, bg="white")

# kella raam
tahvel.create_oval(10,10,w-10,h-10)
# kella keskpunkt
tahvel.create_oval(w/2-5,h/2-5,w/2+5,h/2+5,fill="black")
```

Joonistame sekundiosuti (joon) ja salvestame tema andmedmuutujasse `sek_id`

```
sek_id = tahvel.create_line(w/2,h/2,w/2,20,fill="red")
```

Kuna osuti üks ots on fikseeritud kella keskel, siis meid huvitavad ainult liikuva otsa koordinaadid mingil ajahetkel t. Kui on antud sekundite arv `sekundid`, siis on võimalik arvutada vastavad punkti koordinaadid x ja y:

```
from math import *

# osuti liikuva tipu koordinaadid
# arvutame sekundiosuti pikkuse
r = min(w/2,h/2)-20

# arvutame x koordinaadi
x = r*cos(pi/2-sekundid/60.0*2*pi)

# arvutame y koordinaadi
y = -r*sin(pi/2-sekundid/60.0*2*pi)
```

Järgmise sammuna loome funktsiooni, mis loeb jooksvalt aega ja uuendab sekundiosuti positsiooni. Uus funktsioon luuakse võtmesõna `def` abil, funktsiooni sisu on taandatud. Täpsemalt käsitleme seda kahe nädala pärast.

```
import time

def uuenda():
    # loeme jooksva sekundi
    sekundid = time.localtime().tm_sec

    # osuti liikuva tipu koordinaadid
    # arvutame sekundiosuti pikkuse
    r = min(w/2,h/2)-20

    # arvutame x koordinaadi
    x = r*cos(pi/2-sekundid/60.0*2*pi)

    # arvutame y koordinaadi
    y = -r*sin(pi/2-sekundid/60.0*2*pi)


    # uuendame osuti positsiooni
    tahvel.coords(sek_id, 0, 0, x, y)

    # nihutame keskele
    tahvel.move(sek_id, w/2, h/2)

    # ootame 1 sekundi ja siis uuendame kellaaega uuesti
    raam.after(1000, uuenda)
```

Kutsuge funktsioon `uuenda` välja enne Tkinteri põhitsüklisse sisenemist.

```
uuenda()
tahvel.pack()
raam.mainloop()
```

Kokkupandud kood on järgmine:

```
from tkinter import *
from math import *
import time
raam = Tk()
raam.title("Kell")
# tahvli laius
w = 500
# tahvli pikkus
h = 500

tahvel = Canvas(raam, width=w, height=h, bg="white")

# kella raam
tahvel.create_oval(10,10,w-10,h-10)
# kella keskpunkt
tahvel.create_oval(w/2-5,h/2-5,w/2+5,h/2+5,fill="black")
sek_id = tahvel.create_line(w/2,h/2,w/2,20,fill="red")

def uuenda():
    # loeme jooksva sekundi
    sekundid = time.localtime().tm_sec

    # osuti liikuva tipu koordinaadid
    # arvutame sekundiosuti pikkuse
    r = min(w/2,h/2)-20

    # arvutame x koordinaadi
    x = r*cos(pi/2-sekundid/60.0*2*pi)

    # arvutame y koordinaadi
    y = -r*sin(pi/2-sekundid/60.0*2*pi)

    # uuendame osuti positsiooni
    tahvel.coords(sek_id, 0, 0, x, y)

    # nihutame keskele
    tahvel.move(sek_id, w/2, h/2)

    #ootame 1 sekundi ja siis uuendame kellaaega uuesti
    raam.after(1000, uuenda)

uuenda()
tahvel.pack()
raam.mainloop()
```

Käivitage rakendus. Täiendage kella. Lisage minuti- ja tunniosuti, mis samuti muudaks aja jooksul oma positsiooni.

4.3 Kasutajaliides *Tkinteriga*

KASUTAJALIIDESE KOMPONENDID

Eelmises osas tegelesime piltidega. Nüüd vaatame graafilise kasutajaliidesega programmide tegemist mooduli *tkinter* baasil. Alustuseks toome ära ühe lihtsa tkinter-i programmi:

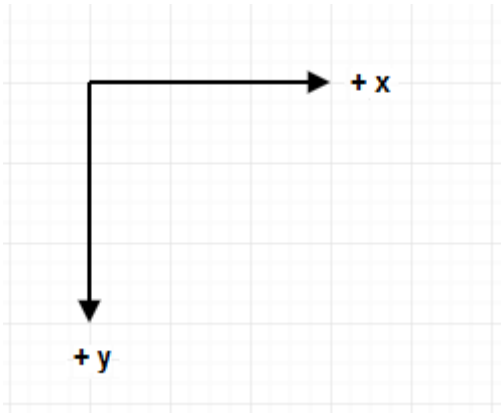
```
# impordi tk vidinad ja konstandid
from tkinter import * # tkinteri põhivahendid
from tkinter import ttk # platvormi ühise stiili saamiseks
from tkinter import messagebox

# see funktsioon käivitatakse nupule klõpsamisel
def tervita():
    tervitus = 'Tere, ' + nimi.get()
    messagebox.showinfo(message=tervitus)


# loome akna
raam = Tk()
raam.title("Tervitaja")
raam.geometry("300x100")

# loome tekstikasti jaoks sildi
silt = ttk.Label(raam, text="Nimi")
silt.place(x=5, y=5)

# loome tekstikasti
nimi = ttk.Entry(raam)
nimi.place(x=70, y=5, width=150)

# loome nupu
nupp = ttk.Button(raam, text="Tervita!", command=tervita)
nupp.place(x=70, y=40, width=150)

# ilmutame akna ekraanile
raam.mainloop()
```

Seda programmi käivitades peaksid saama ühe väikese akna, milles on tekstikast nime sisestamiseks ning nupp, mida vajutades saad nimelise tervituse.

Funktsioon *tervita* on mõeldud käivitamiseks nupule klikkimise korral. Funktsiooni kehas küsitakse allpool defineeritud tekstikasti sisu (nimi.get()), moodustatakse selle põhjal tervitusega sõne ning näidatakse seda kasutajale väikses lisaaknas. (Selles funktsioonis oleme kasutanud ühte globaalset muutujat – `nimi` pole ei funktsiooni argument ega lokaalne muutuja, vaid funktsioonist väljaspool defineeritud muutuja.)

Programmis kasutatakse 3 kasutajaliidese komponenti e vidinat (*ingl widget*):

Funktsioon ttk.Label loob ühe **sildi** (s.o vidina teksti näitamiseks). Funktsiooni esimese argumendiga näitasime, et me soovime seda silti kasutada eespool loodud aknas. Kasutades nimelist argumenti *text*, andsime sellele sildile ka soovitud teksti. Käsk `silt.place(...)` paigutas loodud sildi soovitud koordinaatidele (ühikuteks on pikslid, punkt (0,0) paikneb akna sisuosa ülemises vasakus nurgas ning koordinaadid kasvavad paremale/alla liikudes).



Järgmises plokis lõime ja seadsime paika **tekstisisestuskasti** (`ttk.Entry`). Selle paigutamisel näitasime ära ka soovitud laiuse.

Nupu (`ttk.Button`) loomisel määrasime argumendiga *command* ära, mida tuleb teha nupule klikkimise korral. Paneme tähele, et argumendi väärtuseks on ainult funktsiooni nimi, mitte funktsiooni väljakutse (see oleks olnud koos tühjade sulgudega). Põhjus on selles, et me ei taha seda funktsiooni käivitada mitte nupu loomise ajal, vaid siis, kui nuppu klikitakse.

Lisaks on programmis read, mis olid juba eelmise osa näidetes ja on üldjuhul olemas igaks tkinteri programmis. Programmi sisuline olemus sõltub sellest, milliseid vidinaid aknasse paigutatakse ning kuidas kasutaja tegevustele reageeritakse. Võimalike vidinate valimiseks võib uurida alustuseks lehekülge aadressil http://www.tkdocs.com/tutorial/widgets.html. Kasutaja tegevustele reageerimisel saab rakendada kogu oma programmeerimisvõtete arsenali.

PAREM VIIS VIDINATE PAIGUTAMISEKS

Eelmist näiteprogrammi käivitades ei pruugita olla rahul olla vidinate paigutusega. Paigutust saab muuta korrigeerides etteantud koordinaate ja mõõtmeid. Paraku on selline pikselhaaval timmimine üsna tänamatu töö, kuna mõnes teises operatsioonisüsteemis (või ka teiste seadetega arvutis) ei pruugi seatud paigutus sobida.

Vidinate paigutust on võimalik muuta ka natuke üldisemalt kui pikslite tasemel, lubades sellega Tk-l valida vastavalt olukorrale kõige parem konkreetne paigutus. Järgnevas programmis on vidinate paigutamiseks kasutatud meetodi `place` asemel meetodit `grid` , mis jagab kasutajaliidese mõtteliselt ruudustikuks ning paigutab iga vidina soovitud lahtrisse vastavalt argumentidele *column* ja *row*.

```
from tkinter import *
from tkinter import ttk
from tkinter import messagebox

def tervita():
    tervitus = 'Tere, ' + nimi.get()
    messagebox.showinfo(message=tervitus)


raam = Tk()
raam.title("Tervitaja")
# raam.geometry("300x100") # akna algne suurus määratakse vastavalt
sisule


# paigutame sildi ruudustiku ülemisse vasakusse lahtrisse (column ja
row)
# soovime, et sildi ümber jääks igas suunas 5 pikslit vaba ruumi
(padx ja pady)
# soovime, et silt "kleepuks" oma lahtris ülemisse vasakusse nurka
(sticky)
# N - north, W - west
silt = ttk.Label(raam, text="Nimi")
silt.grid(column=0, row=0, padx=5, pady=5, sticky=(N, W))

# tekstikasti puhul soovime, et ta kleepuks nii ida- kui lääneserva
külge
# st ta peab venima vastavalt akna suurusele
nimi = ttk.Entry(raam)
nimi.grid(column=1, row=0, padx=5, pady=5, sticky=(N, W, E))

# soovime, et nupp veniks nii laiuses kui ka kõrguses
nupp = ttk.Button(raam, text="Tervita!", command=tervita)
nupp.grid(column=1, row=1, padx=5, pady=5, sticky=(N, S, W, E))

# soovime, et akna suuruse muutmisel muudetakse veeru 1 ja rea 1
mõõtmeid
# (st. veerg 0 ja rida 0 jäävad sama laiaks/kõrgeks)
raam.columnconfigure(1, weight=1)
raam.rowconfigure(1, weight=1)

# kuvame akna ekraanile
raam.mainloop()
```

Lisaks meetoditele *place* ja *grid* võib kohata veel paigutusmeetodit *pack*, mida kasutasime tahvli puhul eelmises osas. Rohkem infot saab siit: http://www.tkdocs.com/tutorial/concepts.html#geometry.

PILDID TAHVLILE

Kuigi tahvliga tegutsesime juba eelmises osas, toome tahvlile pildi lisamise siin. Tahvlile saab panna .gif, .pgm, või .ppm formaadis pilte. Järgmise näite proovimiseks salvesta programmiga samasse kausta järgmised failid: kuusk.gif, lill.gif.

```
from tkinter import *

raam = Tk()
raam.title("Tahvel")
tahvel = Canvas(raam, width=400, height=400, background="white")
tahvel.grid()

# pildi kuvamisel on vaja kõigepealt laadida pilt ja panna see siis
tahvlile
kuusk = PhotoImage(file="kuusk.gif")
img = tahvel.create_image(250, 80, image=kuusk)

# activeimage määrab pildi, mida näidatakse, kui hiirekursor on
pildi kohal
# anchor näitab, mille järgi pilt paigutatakse (antud juhul ülemise-
vasaku nurga järgi)
lill = PhotoImage(file="lill.gif")
img = tahvel.create_image(50, 200, image=kuusk, activeimage=lill,
anchor=NW)

raam.mainloop()
```

NÄIDE

Järgmine programm kõlbaks kontrollülesande 4.2c lahenduseks, aga nüüd peate midagi muud välja mõtlema või antud programmi oluliselt muutma.

```
#Impordime kõik vajalikud teegid
from tkinter import *
from tkinter import ttk

#Loome raami ja paneme raamile pealkirja
raam = Tk()
raam.title("Fonolukk")

#Fonoluku ekraani osa
kood = ttk.Entry(raam).grid(row = 0, column = 0, columnspan = 3,
rowspan = 1, sticky = (N, W, E), padx=5, pady=5)

#Lisame fonolukule numbrid
rida = 1
veerg = 0
arv_nupul = 1
while arv_nupul < 10:
    #Lisame nupu
    ttk.Button(raam, text = str(arv_nupul)).grid(row = rida,
column = veerg, padx = 5, pady = 5)
    #Kui arv_nupul jagub 3-ga, siis paneme järgmise arvu uude ritta
ja nullime veeru väärtuse
    if arv_nupul % 3 == 0:
        rida += 1
        veerg = 0
    #Kui arv_nupul ei jagu 3-ga, siis suurendame veeru väärtust.
    else:
        veerg += 1

    #Suurendame arvu väärtust
    arv_nupul += 1

#Lisame ka teised fonolukul esinevad sümbolid
ttk.Button(raam, text = "*").grid(row = 4, column = 0, padx = 5,
pady = 5)
ttk.Button(raam, text = "0").grid(row = 4, column = 1, padx = 5,
pady = 5)
ttk.Button(raam, text = "#").grid(row = 4, column = 2, padx = 5,
pady = 5)

#Kuvame ekraanile
raam.mainloop()
```

4.4 Silmaring: Arvutigraafika

Autor: Raimond Tunnel, illustratsioonid: Kristel Sergo. (Materjal on lühendatud kursuse autorite poolt.)

Informaatikas on arvutigraafika väga lai valdkond, millel on omakorda palju spetsialiseerumisvõimalusi. Rääkides arvutigraafikast, mõtleme me valdkonda, mis tegeleb arvutis näha soovitava pildi genereerimisega. Selleks võib olla näiteks nupp või tekst, kuid samas ka erinevad efektid filmides ning loomulikult kogu 3D mängude visuaalne pool.

Vastavalt olukorrale tuleb arvutis graafika joonistamiseks mõelda erinevatele asjadele. Näiteks **nupu** korral on oluline:

- Kuhu me nupu ekraanil joonistame?
- Kas nupul on piirjoon? Kui lai? Kuidas see joonistatakse?
- Kuidas nupu sees olev tekst joonistada?
- Kas nupp on läbipaistev? Kas nupu taga on veel midagi?

Kaks küsimust nendest küsivad "kuidas" ja see ongi põhiline arvutigraafika eesmärk:

- Defineerida **kuidas** midagi jõuab meie programmist ekraanile.

Lahendus sellele küsimusele peab olema ka üpriski **kiire**. Ainuüksi käesoleval lehel on meil tuhandeid tähemärke, igal tähemärgil on mitmeid jooni ning vastavalt teksti suurusele täidavad need jooned mingi arvu piksleid.

- **Piksel** - ekraanipunkt, minimaalne värvitav element ekraanil (või digitaalsel pildil).

Arvutigraafikas peab vastama küsimusele, kuidas leitakse üles ning värvitakse sobiva värviga õiged pikslid ja seda näiteks 30 korda sekundi jooksul, kui kasutaja lehekülge vertikaalselt liigutab.

Arvutigraafika valdkonnal on ühisosa ka näiteks **digitaalse pilditöötluse** valdkonnaga. Viimane nimelt uurib, kuidas muuta olemasolevaid pilte (näiteks fotosid) paremini nähtavateks (nii inimese kui ka arvuti jaoks). Digitaalse pilditöötluse valdkonda jääb ka olemasolevate piltide pealt kujundite tuvastus. Näiteks võib arvuti programmeerida satelliitpildi pealt üles leidma autosid.

Ühisosa arvutigraafika valdkonnaga tekib olukorras, kus me tahame ühe korra joonistatud pildi peale lisada mingit efekti, et lõpptulemus oleks veelgi parem. Selliseid olukordi juhtub päris palju, kõige lihtsam näide oleks 3D mängust, kus me soovime kõike värve muuta näiteks natukene kollasemaks, sest mängul peab olema just säärane stiil. Selleks me leiame,

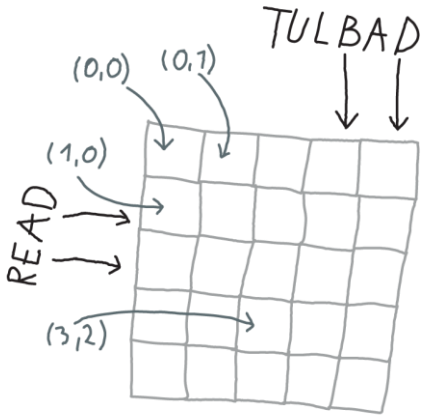
kuidas meie 3D mängu kaader näeks välja tavaliselt ja seejärel muudame pikslite värve vastavalt.

RASTER- JA VEKTORGRAAFIKA

Sissejuhatuses kirjeldatud pikslite värvi järgi millegi joonistamine on **rastergraafika**. Mõiste "**raster**" tähendab antud kontekstis **pikslite ruudustikku**.

Tänapäeval oleme harjunud, et arvutimonitoril olev pilt koosneb pikslitest ning sellest lähtuvalt võime kutsuda monitori ka **rastermonitoriks**.

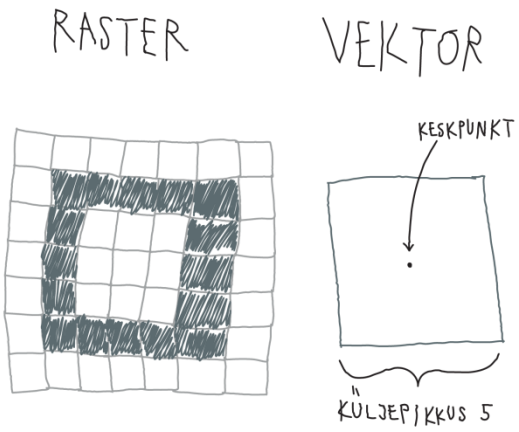
Võib ette kujutada ruudulise vihiku lehte, kus värvime erinevad ruudud erinevat värvi. Igal pikslil on ka selles ruudustikus oma **koordinaat**.



Pikslite koordinaadid 5 x 5 ruudustikus

Näiteks, kui alustame lehe ülevalt vasakult nurgast, siis allapoole jäävad pikslite read ja paremale pikslite tulbad. Võime read ja tulbad nummerdada ja kui me alustame mõlema nummerdusega 0-st, siis esimese ruudu koordinaat on (0, 0). Sellest paremal asub (0, 1) ja all asub (1, 0). Seega esimene arv tähistab rea numbrit ja teine tulba oma.

Tihtipeale aga me tahame defineerida konkreetseid **geomeetrilisi kujundeid**. Näiteks **ruutu** on rastergraafikas võimalik küll joonistada nii, et defineerime 5 pikslit paremale, 4 alla, 4 vasakule ja 4 ülesse tagasi. See on aga päris tüütu ja pigem tahaksime ruudu defineerida näiteks ainult tema **küljepikkuse ja keskpunkti koordinaadiga**. Näiteks olgu ruudu küljepikkus 5 ja keskpunkt näiteks koordinaadil (3, 3). Niimoodi me saame küljepikkust muuta ja lihtsasti defineerida **erineva suurusega ruute**. Võime muuta ka ruudu keskpunkti koordinaati ja selle läbi **kujundit liigutada**. Selliselt joonistatava geomeetria defineerimist nimetatakse **vektorgraafikaks**.



Ruut raster- ja vektorgraafikas

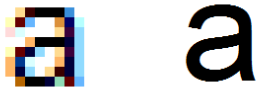
Ajalooliselt olid näiteks ühed esimesed monitorid vektormonitorid, mis joonistasid ekraanile ainult jooni ja neil puudus pikslite ruudustik.

Vektorgraafika on tänapäeval väga **oluline**, sest

- ekraanide mõõtmed on erinevad ja
- see võimaldab lihtsasti objektide suurust muuta.

Täht "a" raster- ja vektorgraafikas

Hea näide sellest on kirjastiilid ehk fondid. Ilmselt me tahame teksti joonistada erineva suurusega. Seda saab näha ka siis, kui me näiteks käesolevat veebilehte suurendame või vähendame (Ctrl klahv + hiire rullik edasi / tagasi). Selle tegevuse tulemusena tähed muutuvad suuremaks ja väiksemaks, kuid on iga kord samasuguse ja sujuva kujuga. Kui me oleksime teinud tähe “a” rastergraafikaga, siis suurendamise korral see täht kaotaks oma kuju.



Täht „a“ raster- ja vektorgraafikas

Eelneval pildil vasakul on näha täht "a", millest tegime ekraanipildi ja suurendasime seda 100 korda. Selle tulemusena näeme suurendatud piksleid. Kusjuures kõik pikslid ei ole mustad. Sellel on ka väga hea põhjendus arvutigraafika valdkonnast. Paremal on aga sama täht "a", kuid oleme kirjasuuruse muutnud 100 korda suuremaks ja siis teinud pildi. On näha, et tähe suurendades pigem ootaksime parempoolset tulemust. Kõik tähed on aga defineeritud erinevate **geomeetriliste valemitega**ja seega on arvutil neid väga lihtne vastavalt vajadusele suurendada ja vähendada.

RASTERISEERIMINE

Kindlasti tekib küsimus, et kui me defineerime kujundid vektorgraafikas, kuid monitorid on rastergraafikas (koosnevad pikslite ruudustikust), siis kuidas me saame ühest teise?

Sellise tegevuse nimi on **rasteriseerimine** ja ka see on üks arvutigraafika eesmärke.

Üks lihtne viis selleks on **panna enda geomeetriline ruum ja ruudustik kohakuti** ning seejärel iga ruudu korral vaadata, kas selle ruudu alla jääb piisavalt suur osa geomeetrilisest kujundist. Kui jääb, siis saab vastava ruudu värvida näiteks mustaks. Vaata ka ingliskeelset videoselgitust kolmnurga rasteriseerimise

Rasteriseerimine

kohta: https://www.youtube.com/watch?v=aweqeMxDnu4&feature=youtu.be&t=5m34s

Tihti on ongi vaja defineerida asju ka rastergraafikas. Näiteks kõiksugused fotod (n. JPG formaadis) just sellisel kujul ongi. Samuti kui me digimaalime pildi mõnes pilditöötlusprogrammis (nt Photoshop), siis ka see on salvestatud raster-kujul. Nimelt fotodes ja digimaalides on tihti väga keerulised kujundid, mida geomeetriliselt hoida ei ole mõtet või on isegi võimatu. Võib ette kujutada, et kui me teeme foto enda kassist, siis me ei leia kassi täielikku geomeetriat ning ei saa pärast arvutis kassi pilti lõpmatuseni detailselt suurendada. Ainus parameeter selle pildifaili juures on meil see, kui suure ruudustiku peale me kassi jäädvustasime.

Juhul kui on soov teha firmale logo, siis tuleks see teha vektorgraafikas- suuremalt ja väiksemalt särkidele ja reklaamlplakatitele trükkides jääb see alati terav. Küll aga kui on soov näiteks printida t-särgile fotot, siis on ainsaks võimaluseks kasutada pilti rastergraafikas.

3D ARVUTIGRAAFIKA

Suur osa arvutigraafikast tegeleb justnimelt **3D graafikaga**, millega paljud on tuttavad nii filmidest kui arvutimängudest. See on väga suur valdkond, mille jaoks on ülikoolides tihti eraldi erialad. Samuti on see ka valdkond, kus tehakse palju koostööd **3D kunstnikega**. Erinevad kolmemõõtmelised kujudid nagu kuup ja silinder on küll toredad, kuid filmis või mängus oodatakse pigem päris maailma meenutavaid ruume, esemeid ja tegelasi. Nende keerulisemate **objektide geomeetriat**, **värve** ja ka **liikumisanimatsioone** valmistavadki 3D kunstnikud.

Informaatika seisukohast on oluline, et defineeritud geomeetria jõuaks **kiiresti ja õigesti** ekraanile, objektid peegeldaksid valgust vastavalt enda värvile ja materjali omadustele. Näiteks metallid ja dielektrik materjalid peegeldavad valgust väga erinevalt. Samuti tegelevad informaatikud sellega, et joonistada erinevaid efekte nagu tuli, suits, vihm või udu.

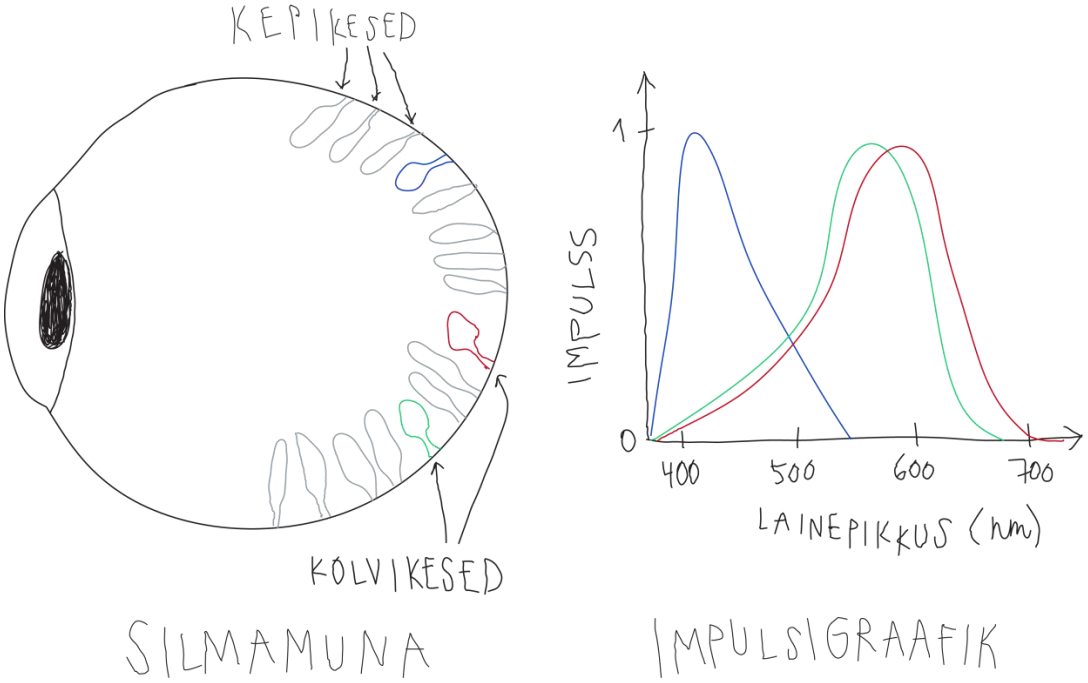
Kogu selle töö käigus kasutatakse programmeerimises palju matemaatilisi objekte ja funktsioone: vektoreid, maatrikseid, nurga koosinust, vektorite skalaar- ja vektorkorrutist, tuletist ja integraali.

Selle kursuse raames me väga sügavuti arvutigraafika matemaatikasse ei lähe. Küll aga huvilistel on võimalus selle kohta lugeda näiteks inglise keelsest materjalist keskkonnas CGLearn.

VÄRVIRUUMID- JA MUDELID

Oleme avastanud, et ekraani peal olevad pikslid on ruudustikus ning igaühel nendest on oma värv. Kuidas aga seda värvi arvutis hoitakse?

Optikast teame, et valgusel on olemas **lainepikkus**. Bioloogiast teame, et inimese silmas on **3 erinevat retseptorit** (kolvikesed), mis reageerivad kõige rohkem **punasele**, **rohelisele** ja **sinisele** valgusele. Erinev kombinatsioon nende kolme retseptori impulsist võimaldab meil tajuda erineva lainepikkusega footoneid ja värve.

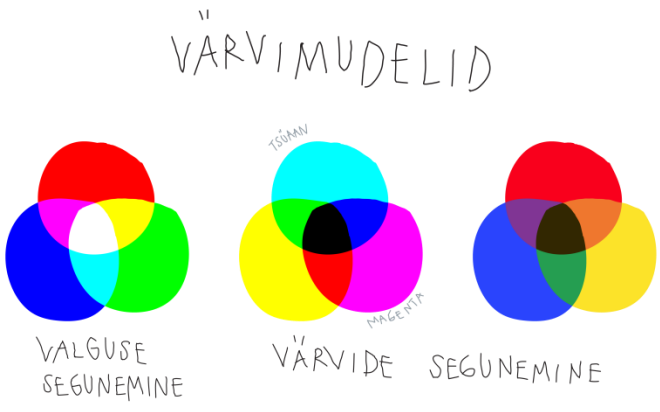


Kepikesed ja kolvikesed inimese silmas ja reageerimisgraafik

Sellest inimeste bioloogilisest omapärast (lindudel on näiteks 4 erinevat tüüpi kolvikest) on disainitud ka meie ekraanid. Monitor kiirgab iga piksli kohta nii punast, rohelist kui sinist valgust. Kombinatsioon nendest kolmest erineva värvusega valgusest võimaldab meil näha erinevaid värve.

Tasub tähele panna, et erineva värvusega valguste koos kiirgamine töötab teistmoodi kui näiteks erinevate guaššvärvide segamine.

Kui meie silm tuvastab korraga punast, rohelist kui ka sinist valgust, siis kõik kolvikesed saadavad impulssi edasi ja me näeme **valget värvi**. Samas, kui me punase, rohelise ja sinise guaššvärvi kokku segame, siis saame tumepruuni värvi. Guaššvärvide segamisel me segame kokku erinevad materjalid, milles neelduvad erinevad lainepikkused. Mida rohkem selliseid materjale me kokku segame, seda rohkemaid lainepikkusi segu neelab ja lõpuks neeldub seal



enamik nähtavast valgusest. Samas erineva värvusega valguste kiirgamisel jõuab lõpuks meie silma rohkem erinevaid lainepikkusi.

Siinkohal olekski oluline teada kahte mõistet:
Värvimudel – kuidas me tähistame arvutis erinevaid värve.
Värviruum – millistele reaalsetele värvidele meie arvutis tähistatud värvid vastavad.

RGB värvimudel ja sRGB värviruum

Kuna monitorid kiirgavad iga piksli kohta eraldi punast (**red**), rohelist (**green**) ja sinist (**blue**) värvi, siis kõige loogilisem variant arvutis värvi hoida olekski arvude kolmikuna, kus on kirjas nende kolme värvi intensiivsus. Sellist värvimudelit kutsutakse värvide inglise keelsete nimetuste esitähtede järgi **RGB värvimudeliks**.

Vastavalt olukorrale võime võtta neid arve

1. ujukomaarvudena vahemikust [0, 1] või
2. täisarvudena vahemikust [0, 255].

Värvide esitamine ujukomaarvudena

Esimesel juhul võime mõelda igast komponendist kui protsendist, kui palju maksimaalsest peab vastavat värvi kiirgama konkreetse piksli sees. Näiteks kolmik (1, 0, 0) tähendaks, et piksel peab olema maksimaalselt punane ja rohelist ning sinist värvi ei kiirga üldse. Kolmik (0.5, 0.5, 0) tähendaks sellisel juhul, et punast ja rohelist on vaja kiirata 50% ning sinist mitte üldse. Selline kolmik annaks meile tumekollase värvi. Tegelikkuses aga, kui me saadame sellise kolmiku monitorile, siis on meie piksel tumedam kui õige 50% kollane, mida monitor näidata suudaks. See on nii selle pärast, et nii ajaloolistel kui ka praktilistel põhjustel saadab monitor kõik värvid läbi funktsiooni tulemus =värv astmelγ, kus kreeka täht γ (**gamma**) võib olla erinevatel monitoridel erineva väärtusega.

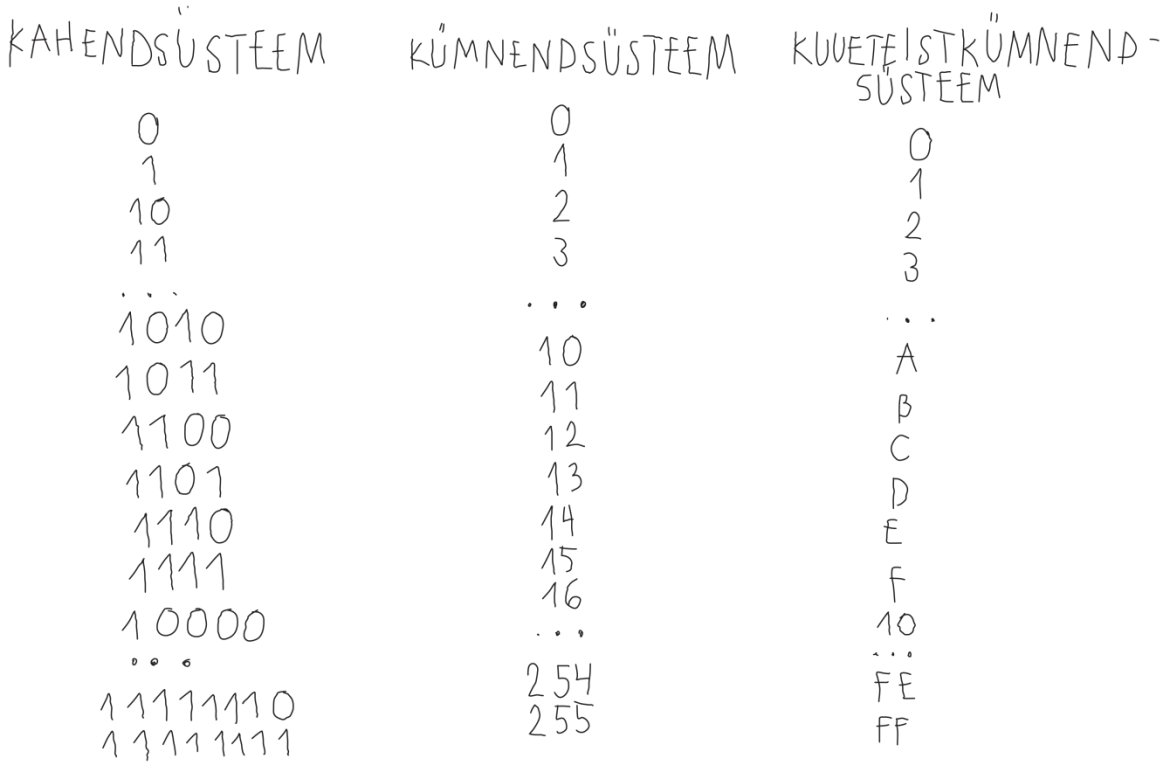
Üldiselt on hea arvestada γ väärtuseks umbes 2,2 (vaata ka näiteks: http://epaperpress.com/monitorcal/gamma.html). Antud funktsioon astendab kõik arvud meie kolmikus astendajaga γ ja kuna meie arvud on vahemikust [0, 1], siis nende väärtused muutuvad väiksemateks. Selle probleemi lahendamiseks tuleks otse monitorile saadetav väärtus enne ise astendada γ pöördväärtusega 1/γ.

Värvide esitamine täisarvudena

Vaatame korraks ka eelnevalt mainitud vahemikku [0, 255] värvikanalite väärtuste jaoks. Sellel vahemikul on mitu põhjust. Esiteks, kuna arvutid hoiavad erinevaid väärtusi binaar- ehk kahendsüsteemis, siis on väärtuste hoidmiseks vahemikust [0, 255] vajalik vähem mälu.

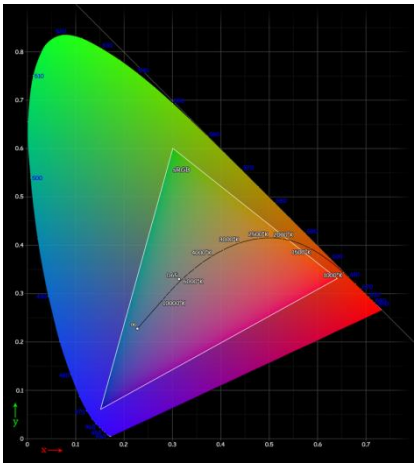
Nimelt kahendsüsteemis hoitakse kõiki väärtusi 0-de ja 1-de kombinatsioonidena ja kui meil on kaheksa kohta arvude 0 või 1 jaoks, siis on meil võimalik kujutada kõiki väärtusi [0, 255], kuid mitte midagi rohkemat (vaata ka videot https://www.youtube.com/watch?v=UBX2QQHlQ_I).

Teine põhjus on, et inimesele on rohkem loetavam (100, 140, 0) kui (0.3922, 0.5490, 0). Kusjuures paljudes kohtades kasutatakse ka heksadetsimaal- ehk kuueteistkümnendsüsteemi. Tolles süsteemis on meil lisaks tavalistele numbritele 0, 1, 2, 3, 4, 5, 6, 7, 8 ja 9 ka veel juures numbrid A, B, C, D, E ja F. Ülal toodud näide kuueteistkümnendsüsteemis oleks (64, 8C, 0). Kõige suurem arv, mida me kahe sümboliga kuueteistkümnendsüsteemis kujutada saame ongi kümnendsüsteemi 255, mis kirjutatakse FF. Veebilehtede arenduses kasutatakse justnimelt seda süsteemi värvide defineerimiseks ning seal kirjutatakse meie kolmik trellide sümboli järele järjest: #648C00.



Arvud kahend-, kümnend- ja kuueteistkümnendsüsteemis

Laiem küsimus on see, et mis värve ikkagi meie RGB või ka mõni teine värvimudel kujutab ehk millisele värviruumile ta vastab. Värvide nägemise ulatus inimestel on suurem kui see, mida monitorid suudavad väljastada. Ei ole pragmaatiline teha sellist monitori, mis suudaks näidata kõiki nähtava spektri (valguse lainepikkuste) värve. Parempoolsel joonisel kolmnurga sees on kõik värvid, mis



on kujutatud enimlevinud sRGB värviruumis. Terve suurem kujund on kogu värvide nägemisulatus inimestel.

4.5 Neljanda nädala kontrollülesanded

Neljandal nädalal tuleb esitada kahe kohustusliku ülesande lahendused. Lahendused tuleb esitada Moodle'is.

Moodle'is on ka nädalalõputest 10 küsimusega, millest tuleb vähemalt 9 õigesti vastata.

Kontrollülesanne 4.1. Suured tähed

Avaldustel/vormidel/lepingutel on inimese nimi (ees- ja perenimi) kirjutatud nii, et esitähed on suured ja teised tähed on väiksed. Aga inimesed kirjutavad eesnime ja perenime erinevalt: ainult väikeste tähtedega, ainult suurte tähtedega või kasutades nii suuri kui ka väikseid tähti segamini.

Koostada programm, mis

- küsib kasutajalt eesnime ja perekonnanime (ühe sisestusena, nii ees- kui perekonnanimi võib koosneda mitmest nimest);
- väljastab nimed nii, et esitähed on suured, teised tähed väiksed.

Näited programmi tööst:

```
>>> %Run yl4.1.py

  Sisestage oma ees- ja perekonnanimi: anton hansen tammsaare
  Anton Hansen Tammsaare

>>>
>>> %Run yl4.1.py

  Sisestage oma ees- ja perekonnanimi: EDUARD VILDE
  Eduard Vilde

>>>
>>> %Run yl4.1.py

  Sisestage oma ees- ja perekonnanimi: augUST gailiT
  August Gailit

>>>
>>> %Run yl4.1.py

  Sisestage oma ees- ja perekonnanimi: ernsT-aleksander joll
  Ernst-Aleksander Joll

>>>
```

Sõnes kõigi sõnade esitähtede suureks ja ülejäänute väikseks tegemiseks on olemas spetsiaalne funktsioon. Tegemist on ühega nendest funktsioonidest, mis ühendatakse sõne külge punktiga.

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 kolmest ülesandest (4.2a, 4.2b ja 4.2c) tuleb lahendada täpselt üks.

Kontrollülesanne 4.2a. Eesti haldusüksuse lipp

Koostada programm, mis kuvab valge taustaga graafikaakna pealkirjaga "Lipp" ja joonistab sinna vabalt valitud (vähemalt 3 värviga või mingi keerulisema kujundiga) Eesti haldusüksuse lipu.

Lipu puhul peab siis olema vähemalt kolm värvi või mingi keerulisem kujund. Eks igaüks valib ise. Narva-Jõesuu lipp on päris tore väljakutse või Tartu või Palamuse. Soovitav oleks valida lipp, kus on midagi korduvat, tsüklilist, et saaks kasutada tsükleid.

Soovi korral võib julgesti jagada pilti oma programmi tulemusest kaaskursuslastega foorumis *Lippude, liiklusmärkide ja majade teosed*. **NB! Juhend ekraanipildi saamiseks.**

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 4.2b. Liiklusmärk

Koostada programm, mis kuvab valge taustaga graafikaakna pealkirjaga "Liiklusmärk" ja joonistab sinna vabalt valitud liiklusmärgi (liiklusseaduses määratud liiklusmärke saab vaadata lisadest).

Mõned innustavad variandid on 112, 175, 178, 438, 712, aga ka siin võiks pigem valida midagi sellist, kus tsüklid oleksid asjakohased.

Soovi korral võib julgesti jagada pilti oma programmi tulemusest kaaskursuslastega foorumis *Lippude, liiklusmärkide ja majade teosed*. **NB! Juhend ekraanipildi saamiseks.**

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 4.2c. Maja

Koostada programm, mis kuvab valge taustaga graafikaakna pealkirjaga “Maja” ja joonistab sinna maja. Kasutatud peab olema vähemalt 3 elementi (uks, aknad, katus, korsten jne) ning lisaks taustale kasutatakse vähemalt 2 värvi.

Soovi korral võib julgesti jagada pilti oma programmi tulemusest kaaskursuslastega foorumis *Lippude, liiklusmärkide ja majade teosed*. **NB! Juhend ekraanipildi saamiseks.**