Sisukord

Kursuse "Teeme ise arvutimänge - algus"

1. RAAMAT

OLULISED MÕISTED JA SISSEJUHATUS

Tiina Kull

Tartu Ülikool

2012

Eessõna

Tavaliselt on eessõnad need kohad, kus enamus õpikute lugejaid sujuvalt üle libisevad. Loomulikult, ka seekord võid sa seda teha ja asuda kohe nö asja juurde, kuid iial ei tea tegelikult, millest sa ilma jääd... Seega, kuna järgnev tekst ei ole väga pikk, siis viskad ehk pilgu peale.

Kogu kursuse õppematerjal koosneb **kuuest** eraldi **raamatust**, iga nädal üks raamat. Viimaseks nädalaks uut õppematerjali ettenähtud ei ole. Laias laastus on iganädalased teemad raamatutes järgmised:

1. **raamat**: esimene tutvus Pythoniga, sisend, väljund, muutujad jms sissejuhatus
2. **raamat**: hargnemine ja otsustuste tegemine, tsüklid ja mängu kavandamine
3. **raamat**: tsüklid tsüklites ja listid
4. **raamat**: funktsioonid, objektid ja ussimäng
5. **raamat**: graafika ja animatsioon
6. **raamat**: veel mängudes kasulikke programmilõike ja heli

Raamatute materjalides on palju igasuguseid ikoone ja pildikesi.

Selline pilt ükskõik kus materjalides tähendab seda, et peaksid ilmtingimata proovima antud näidet ka ise oma arvutis Pythoniga katsetada.

See pilt tähendab video vaatamise võimalust. Video vaatamiseks tuleb vajutada pildil, mille tulemusena avaneb eraldi brauseri aknas videolõik. Videod on mp4 formaadis, kuid kui peaks sellega probleeme tekkima, siis saab ka kiiresti näiteks AVI vms tuntud formaati video ümber renderdada. Sellisest soovist pead aga mulle eraldi teada andma.

Kuidas saada maksimaalne kasu?

- Ükskõik millist kursust sa ka ei võtaks, ikka on kursus üsna suures ulatuses kursuse läbiviija nägu. Seega, et saada maksimaalset kasu antud ainest, siis tuleb ise olla suht aktiivne ja mitte jääda ootama, millal puder suhu tõstetakse. Arusaamatuste või mistahes muude küsimuste korral tuleb KINDLASTI küsida. Seletajale (loe: kursuse läbiviijale) tundub oma jutt alati teistele arusaadav:), see on inimpsüholoogiale väga omane. Seisa siis enese eest ise.

- Teine väga oluline asi, millest on vaja kohe aru saada, on see, et programmeerimise õppimise juures pea 95% programmeerimise juures tekkinud probleemidest, teadmatustest, küsimustest jne lahendatakse praktikas **interneti** vahendusel. Siin see kursus ja tema maht on selleks liiga väikesed, et saada kirjeldada kasvõi kümnendikkugi tegelikust programmeerimise maailmast. Mina annan sulle selle kursuse jooksul vaid tugipunktid - nö ABC, millede läbimisel saad edaspidi iseseisvalt edasi minna. **Väga oluline** on see, et õpiksid selle kursuse käigus tihedalt interneti abi kasutama.

Hakkame pihta!

Julge hundi rind on rasvane, seega kaotada ei ole mitte midagi - hakkame pihta!

Kõigepealt, et üldse mingite mängude programmeerimisest saaks juttu teha, peame

1. enda arvutisse installeerima selleks vajaliku programmeerimiskeele (antud kursusel siis Pythoni ja see on tasuta).
2. tegema selgeks põhilised programmeerimise algtõed - sellel nädalal saame teada, mis on muutujad, andmetüübid, sisend, väljund, kuidas ja miks on vaja teha arvutusi. See nädal tegeleme seega suures osas esmaste teadmiste ettevalmistamisega ja veidi vähem konkreetsete mängudega.

Kuid kõigepealt installeerima!

Pythoni installeerimine on väga lihtne asi. Kõigepealt tuleb sul minna järgmisele veebilehele http://python.org/download/ ja valida sealt vastavalt sinu arvuti operatsioonisüsteemile kõige viimane Pythoni versioon ehk siis hetkel Python3.2.2.

Kui oled õige rea üles leidnud ja selle lingile vajutanud, tuleb sul tegutseda juba vastavalt tekkinud dialoogiakna juhistele.

Siin on sulle ka väike videolõik installeerimise teostamiseks:)

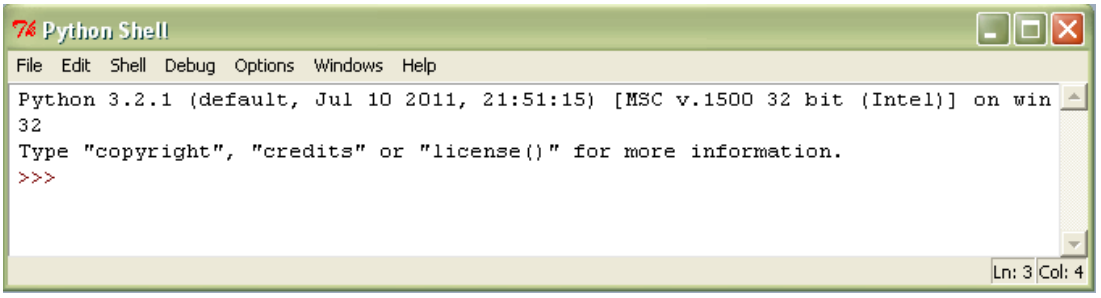
Esimesed katsetused

Nii, Pythoni programmeerimiskeele keskkond peaks sul nüüd arvutis olemas olema ja proovime sellega midagi ka teha. Pythoni kasutamiseks programmeerimise eesmärgil on paar erinevat võimalust. Esimene neist on **IDLE-keskkonna** (i.k. *Integrated DeveLopment Environment*) kasutamine ja seda me just tegama hakkamegi.

NB! Kuna mina kasutan Windows operatsioonisüsteemi, siis paljud juhised on just selle opsüsteemi järgi, kuid kuna opsüsteemid üksteisest nüüd nii väga ka ei erine, siis ehk ei ole see suur takistus. Vajadusel kirjuta foorumisse, sest saan kergesti katsetada ka teiste süsteemidega.

ALUSTA!

Ava **Start menüü** alt Python3.2.2 ja vali sealt **IDLE(Python GUI)**. Nüüd avaneb IDLE aken, mis peaks välja nägema midagi sellist, kus kolme noolekesega real vilgub kursor.



PS: GUI tähendab i.k. *graphical user interface* ehk siis graafiline kasutajaliides. See tähendab seda, et programmil on aken, menüü, nupud, kerimisriba jne. Programme, millel need puuduvad, nimetatakse teksti-režiimi programmideks või konsoolprogrammideks või käsurea programmideks. Seega on ka Pythoni IDLE graafilise kasutajaliidesega.

Märki **">>>"** rea alguses nimetatakse **käsuviibaks** ehk **käsureaks**. Kui selline rida on olemas, tähendab see seda, et Python on valmis ootama sinult käske ja juhiseid, mida täitma hakata.

Python: "Anna käske!!!"

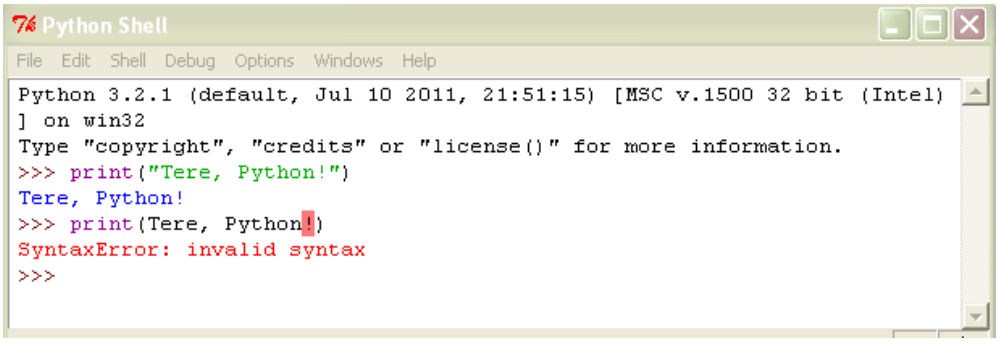
Kirjuta käsureale käsuviiba >>> taha:

```
print("Tere, Maailm!")
```

vajuta Enter ja kirjuta

```
print(Tere, Python!)
```

ja vajuta Enter - jälgi kahe käsu vahet!



Seega iga kord, kui sa tahad, et Python sinu käsu täidaks, kirjuta käsk ja vajuta Enter. Ning pea meeles, Python on üks igavane salakaval madu ja igasugused sõnad, mida me käsureale kirjutame, ei tööta. Nagu sa pildilt näha võid, teine käsk meil läbi ei läinud (jutumärgid on puudu!). Seega Pythoni taltsutamiseks peame enne veidi ussikeelt õppima.

Üks oluline asi IDLE keskkonnas on panna tähele seda, et kui sa oled käsu õigesti kirjutanud, nagu `print` näiteks, siis see värvub violetseks (vaikimisi seaded). Käsu täidab Python **sinisega** ja veateated annab **punaselt**. Pane ka tähele, et tekst kirjutatakse alati **jutumärkide** vahele, siis saab arvuti aru, et tegemist on tekstiga:) ja see värvub kenasti **roheliseks**, jällegi sulle kinnituseks, et oled õigesti kirjutanud.

Proovime veel midagi!

Kirjuta käsureale `print(3+4)` või `print(6*4)` või `print(15/3)`. Mis Python teeb?

Proovi veel sama asja, kuid sulgude sisu pane jutumärkidesse!

Või on igav?!?

Kirjuta siis käsureale

```
print("tee"+"rull"+"uisud")
```

või

```
print("ahv "*20)
```

Esimene päris programm

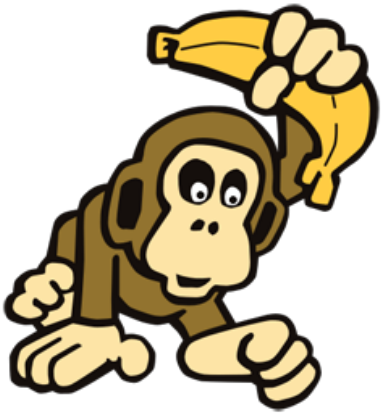
Praeguseks proovisime Pythoniga suhelda käsurea kaudu, kuid egas siis nii programme ei kirjutata.

Jaa, nii võib ju kaa, kui sul on rohkelt aega ja salvestamist ei soovi:)

Aga tegelikult, et teha seda päris-päris asja, tuleb sul selles samas IDLE aknas avada **File > New Window**. Avaneb täiesti tühi aken ilma igasuguste käsuviipadeta, kuhu saab hakata kirjutama käske üksteise alla. Seda akent või keskkonda nimetatakse **IDLE'i tekstiredaktoriks** või **teksti editoriks**. Käsud, mis sinna aknasse kirjutatakse, slavestatakse eraldi failina ja seda faili ehk programmi saab igal suvalisel hetkel tervikuna käima panna.

Kuidas alustada?

1. Ava IDLE tekstiredaktor
2. Salvesta uus programm ehk tühi aken (mis siis, et sul ei ole veel ühtegi rida kirjutatud)
    - soovitan sul teha proovitööde jaoks täiesti oma kaust
    - pane pealkirjaks näiteks *esimene.py*
    - ära unusta laiendit ***.py***, see tagab värvid Idle's
3. Programmi sisuks kirjuta vabalt üks lugu, näiteks:



```
print("Mulle meeldivad ahvid!")

print("ahvid "*20)

print("huuh "*50)

print("Aitab! Kalad on paremad!")
```

4. Salvesta uuesti (**CTRL+s**)
5. Pane programm käima - vajuta **F5** või menüüst **Run > Run Module**

Mis siis juhtub, kui tuleb veateade ja minu kirjutatud programm ei lähe tööle?

Laias laastus on veateateid kahte sorti:

1. **Syntax errors**

   need on vead, mis tulevad tavaliselt välja juba siis, kui programm ei ole veel päris käima pandud. Nimelt IDLE kontrollib enne su koodi üle, et kas see on korrektselt kirjutatud, kas kõik käsud on tema jaoks arusaadavad, kas kõik reavahetused, sulud, jutumärgid, koolonid jms on õigetes kohtades. Kui mitte, annab ta Syntax errori (ehk nö grammatika vea, kirjutamise vea teate) ja märgib punase kastikesega ligikaudse koha koodis, kus Pythoni arvates viga baseerub.

1. **Runtime errors**

   Need on sellised vead, mis tulevad alles siis välja, kui programm on saadud juba käima panna.

   Näiteks kui ma kirjutaksin käsureale nii:

   ```
   print("Tere" + 5)
   ```

   siis kood on grammatiliselt õigesti kirjutatud, kuid tegelikult Python sellist tehet teha ei oska ning annab veateate. Proovi, milline see veateade välja näeb!

Miks oskab Python peale hakata `print("Tere" * 5)`-ga aga `print("Tere" + 5)` peale annab veateate?

Nii, siin mõtle natuke, see on koht, kus tuleb loogika mängu panna!

....

Vastus! Põhimõtteliselt võin ma seda võrrelda olukorraga kui mul on üks pliiats ja ma korrutan seda viiega, siis ma saan viis pliiatsit (teen koopiad), kuid kui mul on pliiats ja liidan 5 juurde, tekib esiteks küsimus, mida ma juurde liidan, kas kartuleid? Kuna tekstile liidetakse juurde arv, siis on see sama kui pliiatsitele kartulite liitmine. Kes ütleb, mis siis kokku tuli, kas kartulid või pliiatsid või kartulpliiatsid?

Pea meeles, et kunagi ei saa programmeerimises liita tekstile juurde arve ilma et sa eelnevalt arvu tekstiks muudaksid või vastupidi. Selle muutmisest aga räägime mõni peatükk hiljem.

Pea ka seda meeles, et veateated on väga loomulikud asjad. Ilma veateadeteta ei saa läbi ükski programmeerija, isegi mitte see, kes on 30a programmerija olnud. Veateadetaga harjumiseks ja nende tähenduste tõlgendamiseks soovitan sul külastada Pythoni veateadete lehekülge ja seda veidi uurida: http://docs.python.org/dev/tutorial/errors.html

Esimene mäng

Arva ära, millise arvu peale ma praegu mõtlen?

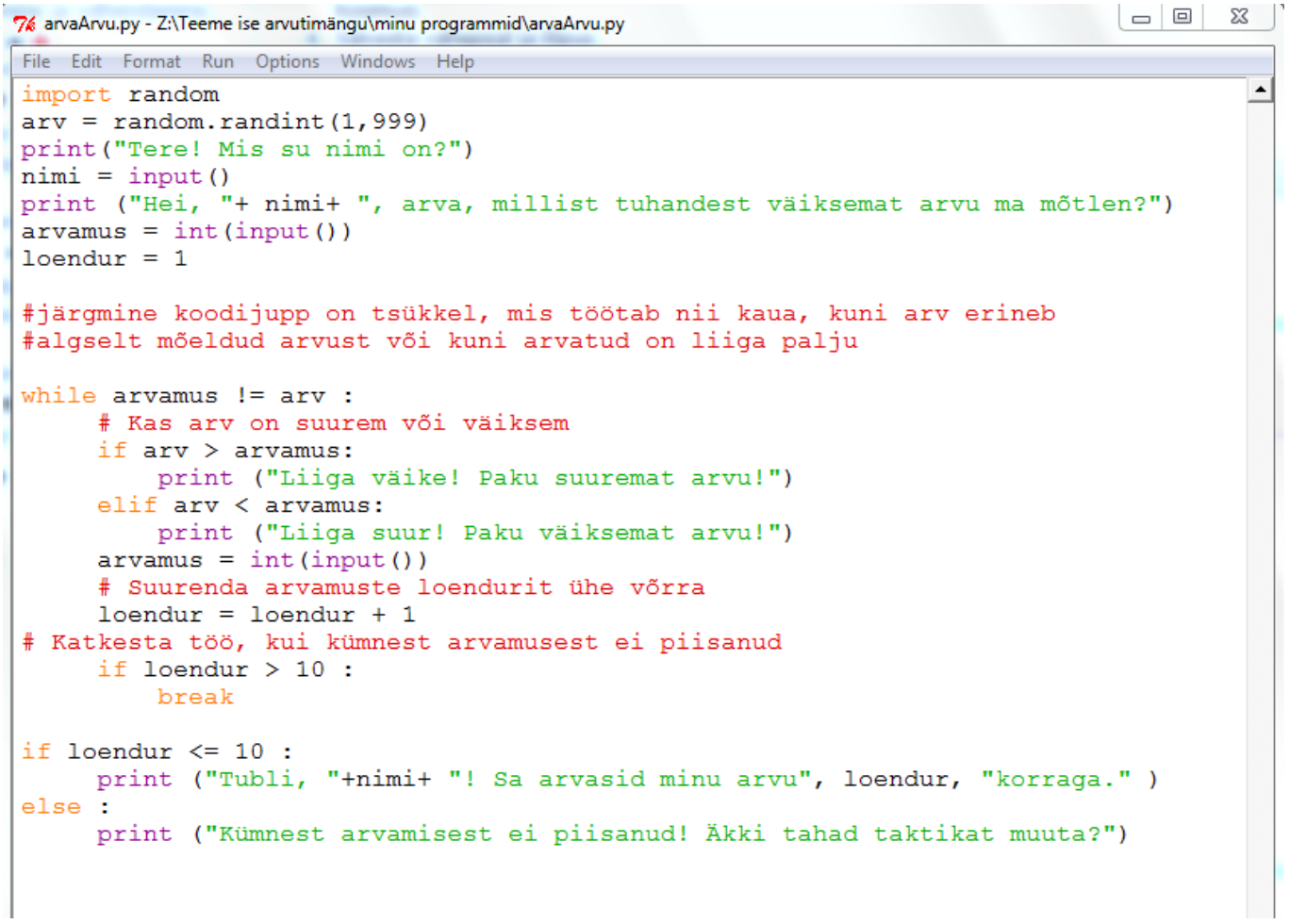
Esimene programm, mida sa Pythoniga kirjutasid, ei teinud just suurt midagi - kirjutas vaid **`print`** käsu abil ekraanile mõned read teksti. Kuna see kursus sai aga välja reklaamitud kui mängude programmeerimise kursus, siis hakkame aga mängudega pihta.

Kõigepealt, et tuure natuke maha võtta, pean sulle ütlema, et lihtsalt paari rea koodiga ei saa ühtegi mängu teha. Kui tahta teha sellist väikest lihtsat mängu (ilma erilise graafikata), peab kirjutama u poolesaja rea koodi lähedale, kui tahta teha veidi suuremat või lausa mega-giga mängu, siis tuleb arvestada tuhandete ridadega programmidega. Need mängud, mida me sellel kursusel vaatame, on nii keskmiselt 50-200 rida pikad.

Algatuseks aga üks pisikene arvu arvamise mäng:

Kuna sa suurt midagi Pythoni käskudest veel ei tea, siis iseseisvalt täiesti uut mängu oleks päris raske teha. Seetõttu, minu strateegia siin kursusel ongi läbi olemasolevate programmikeste õpetada sulle programmeerimise ja mängude kirjutamise põhitõed selgeks.

1. Ava IDLE keskkond ja uus tühi aken. **File > New**
2. Salvesta veel tühi programm ja pane talle nimeks näiteks *ArvaArvu.py*
3. Trüki ise tähthaaval ümber allolev kood. Miks trükkida, kui võiks kopeerida? Annan sulle lihtsalt head nõu kudas kõige paremini/kiiremini programmeerimine selgeks saada. Lihtsalt kopeerides ei mõtle sa programmi sisule ja ei hakka juurdlema, miks siia kirjutatakse koolon või siia taane. Ja saad ise trükkides programmi kirjutamise loogikast kiiremini aru ja tulevikus küsida foorumites asjalikke küsimusi.
4. Väga oluline on koodi liigendus ehk taanded, iga taane võiks olla 4 tühikut.
5. Salvesta vahepeal ja lõpus.
6. Pane programm käima **F5** või **Run > Run Module**
7. Nutikat mängimist!



Olulised mõisted

Nüüd kui oled natuke juba "oma loodud" mängu mängida saanud, siis tahaks ju teada täpsemalt, kuidas ise analoogset programmi teha?

Kiirustada aga ei tasu. Enne kui sulle selgitama hakkan iga rea tähendust eelmises mängus, tuleb kõigepealt selgeks saada mõned väga olulised mõisted. Need on **arvuti mälu**, **sisend**, **sisendi töötlemine**, **väljund** ja **muutujad**.

Sisend, sisendi töötlemine ja väljund

Sinu päris esimesel programmil puudus sisend ja väljundiks oli ka vaid programmi sisse kirjutatud tekstid, sellepärast see programm ei olnudki just väga huvitav.

Teises programmis (arvu arvamine) olid olemas aga kõik kolm komponenti, mis tegid programmi koheselt palju lõbusamaks.

- **Sisend** (ehk *input*) olid arvu arvamise katsed (arvamus) või mängija nimi
- **Sisendi töötlemine** toimus siis, kui programm kontrollis, kas arv on võrdne mõeldud arvuga
- **Väljundiks** (ehk *print*) olid teated arvamise õnnestumisest või ebaõnnestumisest.

Selge on see, et üks korralik programm vajab sisendit, kuid kuidas jätab arvuti sisendi meelde? Kuidas mingi Python saab arvutile öelda, et jäta nüüd see teade meelde, aga seda ära jäta?

Loomulikult sa tead, et arvutil on olemas mälu. **Mälu** võib vaadelda kui tohutu suurt pesade hulka, mis seisavad ühes kindlas olekus seni, kuni sa neid muudad. Olek tähendab kahendkoodis infot, mis on sellesse mälupessa salvestatud. Sellist struktuuri võibki piltlikult nimetada nö meelde jätmiseks. Iga pesa fikseeritakse kahendkoodiga teatud olekusse ja ta jääb sellesse seniks, kuni toimub taas muutus. Seega alati saab kas muuta mälupesa (programmeerida mälupesa ümber) või ainult lugeda mälupesast seal olevat kahendkoodis infot ilma muutmata.

Kuidas siis Python või mis iganes programmeerimiskeel mälupesi muudab? Ja kui ta on ühe pesa sisu muutnud, kuidas programmeerimiskeel selle pesa uuesti üles leiab?

Programmeerides, kui sa tahad, et arvuti jätaks mingi olulise info meelde, siis tuleb sellele infole anda alati **nimi**. Kohe kui nimi on antud, saab Python aru, et nüüd on vaja arvutimälu poole pöörduda ja see info (olgu see üksainus arv, sõna, lause, pilt või muusika) mällu salvestada. Python kinnitab **nime** külge arvuti mälupesa aadressi, mille kaudu saabki ta alati nime kasutades nime taga oleva info arvuti mälust kätte. Nii lihtne see ongi.

Nii tehti ka arvu arvamise mängus. Kui programmis küsiti mängijalt arvu, siis sellele arvule (ehk siis info, mis tuli meeles pidada), tuli anda **nimi**, mis aitas arvu arvutimällu salvestada. Selleks nimeks oli antud programmis *arvamus* ja info seoti temaga võrdusmärgi abil.

```
    print ("Liiga suur!
arvamus = int(input())
# Suurenda arvamuste loe
```

Muutujad

Kui ma eelmisel lehel rääkisin, et iga info, mis tahetakse meelde jätta, tuleb varustada nimega, siis programmeerijate maailmas kutsutakse selliseid ise välja mõeldud nimesid **muutujateks**.

Ning kui ma **võrdusmärgi** abil annan **muutujale** mingi väärtuse, siis nimetatakse sellist protseduuri **muutujale väärtuse omistamiseks**.

Teeme ühe muutuja väärtustamise Idle käsurea keskkonnas (Python Shell) reaalselt läbi.

Omistame muutujale mingi väärtuse. Mida see tähendab? Kirjutame käsureale ühe nime (ehk muutuja), mille kaudu Python salvestab muutuja taga võrdusmärgi järel oleva info mälupessa. Kui me tahame seda infot mälupesast uuesti kasutada, siis saame selle kätte jälle selle sama muutuja kaudu, millele Python kleepis külge ka mälupesa aadressi.

Kuna üks pilt on palju kõnekam kui tuhat sõna, vaata muutuja defineerimise protseduuri siit:

NB! Kas on kitsendusi muutuja nime valimisel?

Laias laastus võid sa muutuja nimeks valida mistahes tähtede kombinatsiooni - peaaegu. Nimi võib ka olla kui tahes pikk (ülipikad nimed aga halvendavad koodi lugemist) ja see võib sisaldada numbreid ja alakriipse (_). Kuid on paar kitsendavat reeglit:

- Muutuja nimed on tõstutundlikud st ühe ja sama tähe suur ja väike variant on erinevad sümbolid - need ei ole üks ja seesama täht. Näiteks muutuja *koer* ja *kOer* on erinevad muutujad.
- Muutuja nimi peab alati algama tähe või alakriipsuga. Näiteks *4koera* ei saa olla muutuja nimi.
- Muutuja nimi ei tohi sisaldada tühikuid, sümboleid, kirjavahemärke jms.

Näited muutujatest ja nende väärtustest:

- `eesnimi = "Tiina"`
- `esimene_vastus = 45`
- `_õpilase_nimi = "Juku"`
- `Minu_loendaja_2 = 0`
- `SinuTeineVastus = "kolm karu"`

Kuidas mõtleb programmeerija?

Kui muutujale on omistatud mingi väärtus, siis see väärtus on salvestatud mällu ja seda infot kutsutakse nüüdsest muutuja nimega. Enamus programmeerimiskeeltes öeldakse sellise olukorra kohta, et me hoiame väärtusi muutujas. Pythonis käivad aga muutujate asjad veidi teisiti kui enamus programmeerimiskeeltes (minu arvates lihtsamalt). Nimelt, selle asemel et hoida väärtuseid muutujates, antakse pigem väärtustele nimesid. Mõned Pythoni programmeerijad lausa ütlevad, et Pythonis ei ole "muutujaid" vaid on "nimed". Kuid nad käituvad üsna ühte moodi. Seega, et sa teaksid siis, et kui ma kasutan tekstis sõnu muutuja, nimi või muutuja nimi, siis need tähendavad laias laastus ühte ja seda sama asja. Tegelikult ei ole väga suurt vahet kuidas neid kutsuda, põhiline on, et sa mõistaksid, kuidas muutujad käituvad ja kuidas neid oma programmides kasutada.

Kuidas muutujad muutuvad?

Muutuja teema juures võib õigustatult tekkida küsimus, miks mällu salvestatavale infole antud nime üldse kutsutakse muutujaks?

Sellepärst, et sageli on vaja programmides kasutatavate nimede taga olevat väärtust muuta. Näiteks see sama arvu arvamise mäng. Kõigepealt saab selles programmis muutuja *arvamus* näiteks väärtuse 56 ja kui osutub, et see ei ole õige vastus, siis ei ole tegelikult ju vaja seda 56 rohkem meeles pidada. Küsime kasutajalt hoopis uue arvamuse ja omistame vanale muutujale *arvamus* uue arvamise sisu, olgu selleks näiteks 75, mida tahame jällegi korraks mälus hoida. Seega oleme eelmise väärtuse nö kustutanud ja muutuja *arvamus* on saanud uue väärtuse 75.

Arvud ja sõned

Loodan, et sa panid eelmisel lehel tähele, et näidis muutujatele oli antud kahte erinevat tüüpi väärtuseid: arvulisi ja sõnalisi. Arvulise väärtusega muutujaid kutsutakse **arvulisteks** muutujateks, aga sõnalise väärtusega muutujaid kutsutakse **sõnedeks**.

Näited arvulise väärtusega muutujatest:

- `esimene = 4`
- `teine = 6`

Näited sõnemuutujatest:

- `vastus = "Küll on kena kelguga hangest alla lasta"`
- `eesnimi  = 'Tiina'`
- `arv_tekstina = "23"`

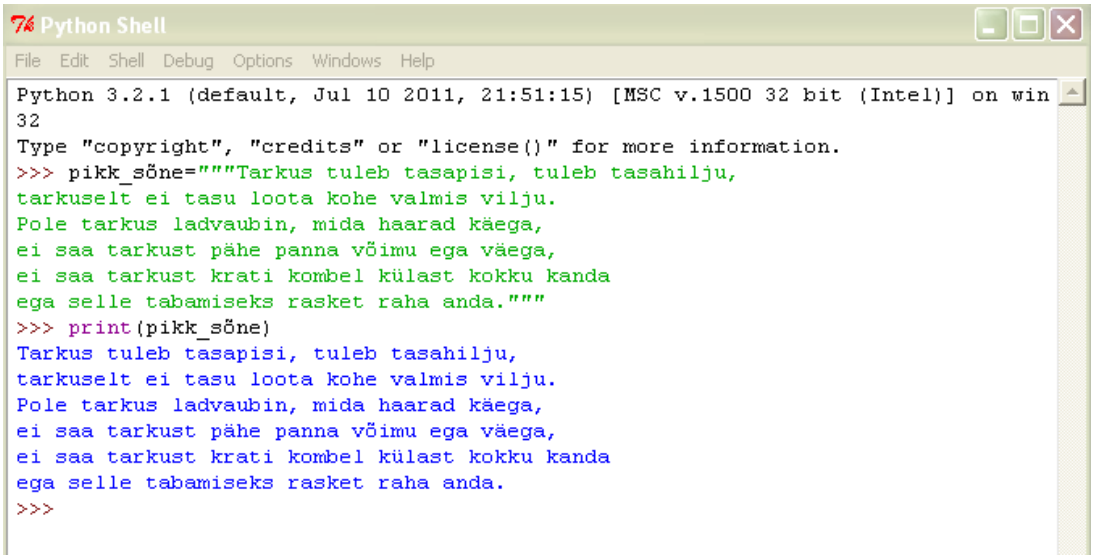
Kuidas Python saab aru, kas tegemist on arvu või sõnega (`arv_tekstina = "23"`)? Vahe on jutumärkides - kui pannakse arvule jutumärgid ümber, siis on kohe tegemist sõnega ja mingit arvutamist sellise muutujaga teha ei saa.

Kui olid tähelepanelik, siis märkasid et sõnede näide nr 2 on teistsuguste jutumärkidega kui ülejäänud. Pythonis ei olegi tegelikult vahet, kas kasutad kahekordseid või ühekordseid jutumärke, peaasi, et sõne alguses ja lõpus on samad jutumärgid.

Eriti pikad sõned:

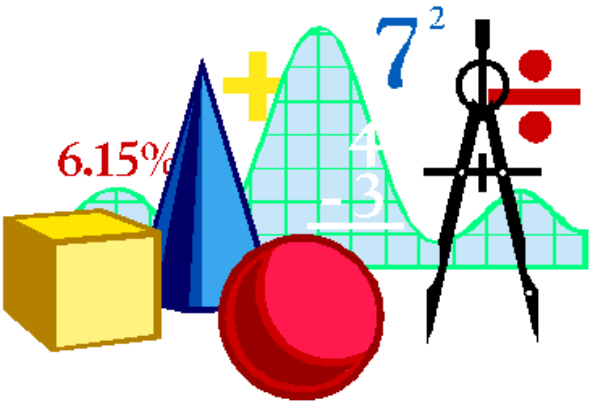
Kui sul on vaja aga muutuja väärtustada eriti pika sõnega (näiteks üks salm luuletusest vms), siis tasub sellise sõne ümber kasutada kolmekordseid jutumärke. **Kolmekordsed jutumärgid** lubavad teha jutumärkide vahel ka **reavahetusi**.

Näiteks:



Uurime, milliseid põnevaid tehteid saab erinevat tüüpi muutujatega teha:

Matemaatika



Huvitav, pidime siin ju mänge programmeerima, aga nüüd hakkame hoopis rääkima matemaatikast. See võib tunduda klišeena, kui öelda, et matemaatika on igal pool ja sellepärast peab seda mõistma. Sageli ei ole ju kooli matemaatikaülesanded kuidagi seotud sellega, mis on päriselus vaja olnud või kui siis eriti vähe. Tegelikult aga on matemaatika eriti oluline just programmeerimise juures ja veel eriti mängude programmeerimise juures. Kui silme ette manada üks tüüpiline arvutimäng, siis seal on palju erinevaid objekte erineva suuruse ja erinevate füüsikaliste omaduste jms-ga. Tüüpiliselt liiguvad nad ekraanil ringi (ekraan ei ole aga midagi muud kui üks suur koordinaatteljestik) - seega peab programm pidevalt oskama arvutada objektide liikumise kiirust, suunda, paiknemist, värvust (ka värvid on ju kodeeritud arvudeks). Programme kirjutab aga programmeerija ja kui programmeerija on matemaatikas... hmm ütleme nii, et pole piisavalt selle ainega tegelenud, siis on ka keeruliste konstruktsioonide ja geniaalsete tulemuste saavutamine suht lootusetu.

Seega, vaatame kõigepealt üle, kuidas käib nö elementaarmatemaatika Pythonis. Siinkohal ei saa mainimata jätta, et enamus programmeerimiskeeltes käib arvudega opereerimine sarnaselt, seega kui oled ühe korra võtted selgeks saanud, ei ole mingi probleem samu asju teha mõnes teises programmeerimiskeeles.

Põhitehted

Põhitehteid Pythonis ja enamus teistes programmeerimiskeeltes tehakse järgmiste sümbolite kaudu:

Tehtemärk	Mida see teeb? + kommentaar
+	tavaline kahe arvu liitmine
-	tavaline kahe arvu lahutamine
*	tavaline kahe arvu korrutamine
/	kahe arvu jagamine, tulemus on ALATI reaalarv
**	astendamine
%	jäägi leidmine
//	kahe arvu täisosaline jagamine
int()	reaalarvu täisarvuks muutmine - lõigatakse komakohad tagant ära
round()	ümardamine

Igaks juhuks, enne kui läheme konkreetsete näidete juurde, tuletame meelde, mis vahe on **täisarvul** (i.k *integer*) ja **reaalarvul** (i.k *decimal number*).

Täisarvud ehk ***integers*** on kõik arvud ilma komadeta ...-3, -2, -1, 0, 1, 2, 3, ...

Reaalvud on need samad, aga komakohaga + kõik suvalise kahe arvu vahele jäävad arvud lisaks ...-2.0,..., -1.0,...,0.0,...,1.0,...,2.0,...

Programmeerimises ei kasutata terminit reaalarv, vaid selle asemel kasutatakse mõistet **ujukomaarv** (i.k *floating-point numbers*, lühemalt ***floats***).

Näited koos konkreetsete arvudega:

Tehe	Tulemus	Kommentaar
`4 + 5.0`	`9.0`	Täisarv pluss reaalarv annab tulemuseks reaalarvu
`3 - 5`	`-2`	
`3.0 - 5`	`-2.0`	
`6 / 3`	`2.0`	Tavalise jagamise tulemus on ALATI ujukomaarv
`5 // 3`	`1`	Täisosaline jagamine, kolm mahub 5 sisse 1 kord
`5 % 3`	`2`	Jäägi leidmine
`5 * 3`	`15`	Tavaline korrutamine
`5 ** 3`	`125`	Astendamine
`4 ** 0.5`	`2.0`	Juurimine astendamise kaudu, sama mis ruutjuur 4-st
`round(2.6375, 2)`	`2.64`	Ümardamine nõutud täpsusega ehk siis kaks kohta peale koma
`round(2.6375)`	`3`	Ümardamine lähima täisarvuni
`int(2.6375)`	`2`	Täisarvuks teisendamisel ei ümardata, komakohad lõigatakse jõuga ära.
`3 + 5 * 2`	`13`	Python arvestab tehete järjekorda, kui tahad järjekorda muuta, kasuta sulgi.
`(3 + 5) * 2`	`16`	
`6 - 3 - 1`	`2`	Sama prioriteediga tehted tehakse vasakult paremale ...
`6 - (3 - 1)`	`4`	
`2 ** 3 ** 2`	`512`	... v.a. astendamised, mis tehakse paremalt vasakule
`(2 ** 3) ** 2`	`64`	

Programmeerimises kehtib raudne **reegel**, mida rohkem sa praktikas oma näppe koodi kirjutamiseks kulutad, seda kaugemale sa selles vallas jõuad. Nii on ka kõikide siinsete videote ja näidete vaatamisega. Tee kõik näited, mida sa videotes näed ja tekstis leiad ise reaalselt läbi, muidu võid maailmameistriks saada vaid vaatamises ja mitte programmeerimises.

Kuidas arvutada Pythoniga?

Paljudes teistes programmeerimiskeeltes kasutatakse astendamisel katuse märki (**^**). Sa võid seda kasutada ka Pythonis, aga sa saad väga vale vastuse, sest katus tähendab Pythonis hoopis midagi muud kui astendamist. Seega, kui oled harjunud eelnevalt astendamsel katust kasutama, siis ole selle kohapeal väga ettevaatlik, sest selline viga ei anna sul programmi käivitamisel veateadet.

Suurendamine ja vähendamine

Programmeerimises on olemas veel kaks põnevat tehet, selliseid tehteid sa tavaliselt koolitunnis ei kohta: nendeks teheteks on ühe võrra suurendamine ja ühe võrra vähendamine. Kuidas see käib?

Olgu mul näiteks muutuja `number = 7`. Ku ma tahan selle muutuja väärtust ühe võrra vähendada, siis ma kirjutan lihtsalt `number -= 1`

Või kui ma tahan seda ühe võrra suurendada, siis kirjutan `number += 1`.

Vaata ka videot:

Eriti suured ja eriti väiksed arvud

Selle peatüki sisu on pigem silmaringi laiendamise eesmärgiga siia lisatud, kui et sul ilmtingimata seda mängude tegemsiel vaja läheks, aga mine sa tea.

Tavaliselt puutuvad algajad programmeerijad hiigelsuurte või hiigelväikeste arvudega kokku siis, kui on midagi totaalselt valesti arvutatud ja siis arvatakse sageli, et programmeerimiskeel on omadega sassi läinud. Kuid tegelikult mitte.

Proovi näiteks käsureale kirjutada kahe väga suure arvu korrutis ja uuri täpsemalt, mis vastuseks anti.

```
>>>
>>> 35438465836098.567*93086207602726.098
3.2988323879411875e+27
>>>
```

Mida see 'e' täht ja '+' teevad seal arvu keskel?

See '**e**' on üks võimalus, kuidas kirjutada lühemalt väga suuri või väga väikseid arve. Sellist kirjaviisi kutsutakse **e-notatsiooniks**. Kui hakata kirjutama väga pikki arve kõigi oma komakohtadega kenasti ekraanile välja, oleks see üks suur peavalu. Esiteks võtaks selline arv väga palju ruumi ja teiseks ei suudaks keegi seda korralikult lugeda - miljardkohad võivad ju ometi sassi minna:) Seetõttu on teaduses kasutusele võetud e-notatsioon, mis tähendab järgmist:

Kui mul on kirjutatud **4,56E+15** (kas on kasutatud suurt või väikest e tähte, sellel ei ole vahet), siis see tähendab tegelikult sama kui ma kirjutaks **4,56x10^{15}** ehk see on omakorda sama, mis nihutada koma paremale 15 koha võrra ehk arvu suurendada. Kui mul oleks e+15 asemel kirjutatud e-15, siis tuleks komakohta nihutada vastavalt 15 võrra vasakule ehk arvu vähendada. Tulemuseks saame, et

4,56e+15 = 4560000000000000

4,56E-15 = 0,000000000000456

Moodul math

Ma ei hakka väga täpselt siin seletama, mis asi on moodul. Piisab esialgu vast sellest, kui sa mõtled temast kui eraldi programmide paketist, millele on antud oma nimi. Siin peatükis tuleb juttu **math** moodulist.

Igasuguseid lihtsaid arvutusi saab Pythonis teha nö niisama. Muudkui kirjuta arve ja tehetemärke ja ongi korras. Kuid enamus huvitavaid matemaatilisi konstruktsioone tehakse aga erifunktsioonide abil. Taskukalkulaatorit koolis kasutades on sulle paljud nendest funktsioonidest tuttavad. Siinus, koosinus, absoluutväärtus, logaritm jne. Kõik need funktsioonid ja palju rohkemgi on ka Pythonis olemas, kuid nedne kasutamiseks peab alati programmi kõige algusesse kirjutama paar võtmesõna:

```
from math import *
```

Tärn tähendab seda, et ma impordin oma programmi KÕIK **math** mooduli fuktsioonid ja muutujad. Tärni asemel võin ma kirjutada ka ainult nende funktsioonide nimed, mida mul on programmis vaja. Näiteks kui mul on vaja ainult piid (3,14...) ja siinust, siis võin kirjutada nii:

```
from math import pi, sin()
```

Ülevaate, milliseid funktsioone või muutuja väärtusi Puthoni **math** moodul endas peidab, saab **math** mooduli ametlikust dokumentatsioonist, mille leiad sellelt aadressilt:

http://docs.python.org/release/3.0.1/library/math.html

Andmetüübid

Muutujate peatüki all rääkisin sulle, et on olemas erinevat tüüpi muutujaid - **arvulised** ja **sõned**. Tegelikult on erinevaid tüüpe palju rohkem ja neid nimetatakse üldmõistega **andmetüüp**. Praeguseks oleme kokku puutunud kolme tüüpi andmetega: täisarvud (*integer*), ujukomaarvud (*float*) ja sõned (*string*). Algatuseks piisab meile vaid nendest kolmest.

See, kuidas ühte või teist tüüpi andmeid luua, sellest rääkisime juba muutujate peatükis. Meeldetuletuseks paar näidisrida:

- eesnimi = "Juku" **< < Sõne** ehk ***string*** andmetüüp, kasutatakse jutumärke
- loendaja = 3 **< < Täisarv** ehk ***integer*** andmetüüp, ei kasutata punkti (e.k koma)
- kiirus = 90.25 **< < Ujukomaarv** ehk ***float*** andmetüüp, kasutatakse punkti (e.k koma)

Andmetüüpide muutmine

Küllalt tihti on programmides vaja andmetüüpe muuta. ***Integer float***'ks või ***string integer***'ks vms variant. Selleks on Pythonis kolm erinevat funktsiooni:

- **`float()`** muudab *stringi* või *integer*'i ujukomaarvuks
- **`int()`** muudab ujukomaarvu või sõne *integer*'ks
- **`str()`** muudab *integer*'i või *float*'i *string*'ks

Märkus! Sulud ülal olevate nimetuste taga ütlevad seda, et tegemist ei ole Pythonis tavaliste käskudega, vaid Pythonisse sisse ehitatud funktsioonidega. Funktsioon on alamprogramm, mis läheb käima siis, kui sa ta nimepidi välja kutsud ja nime taha kindlasti ka sulud kirjutad. Sageli on vaja sulgude sisse ka midagi panna, mingeid andmeid, millega funktsioon saab toimetusi teha.

Näiteks, kui ma tahan muuta täisarvu 24 ujukomaarvuks, kutsun välja funktsiooni **float()** ja kirjutan 24 sulgude sisse ja saan tulemuseks 24.0

```
>>> float(24)
24.0
>>>
```

Kuna jällegi on mõistlikum andmetüüpide muutmist vaadata pigem video pealt, kui kirjutada nende omapärad tekstina üles, siis edasi jälgi juba filmi. Ära unusta ise järgi proovimast!

Kuidas teada saada, millise andmetüübiga on tegemist?

```
>>> a=45
>>> b="Tere"
>>> c="3"
>>> e=5.0
>>> type(a)
<class 'int'>
>>> type(e)
<class 'float'>
>>> type(b)
<class 'str'>
>>> type(c)
<class 'str'>
>>>
```

Kui on veidi arusaamatuks jäänud, millist tüüpi üks või teine muutuja on, siis saab seda alati Pythoni käest küsida. Selleks tuleb kasutada funktsiooni **`type()`** ja sulgude sisse kirjutada muutuja nimi, mille kohta infot tahetakse. Vaata vasakul olevat näidet.

Võimalikud veateated

Selge on see, et kui ma tahan teksti või midagi muud mittearvulist muuta arvuks, siis annab süsteem mulle veateate. Python ei oska sõnast arvu teha, kas sina oskad? Aga kui on mõni arv antud sõnena, näiteks *"56"*, siis selle arvuks muutmisega saab Python hakkama küll:)

```
>>> float("Tere")
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    float("Tere")
ValueError: could not convert string to float: 'Tere'
>>>
```

Sisend ja väljund

Väljundi kirjutamist õppisid sa juba päris esimeses selle nädala peatükis. See oli päris lihtne, tuli kasutada vaid **`print()`** funktsiooni ja sulgude sisse kirjutada, mida on vaja väljastada, kas teksti (jutumärkides), mõne muutuja väärtust (sulgudesse tuleb kirjutada selle muutuja nimi) vms.

Programmi sisendi saamisest oleme ka juba natuke rääkinud, kuid mitte väga täpselt. Kui tahetakse programmi kasutajalt, failist, veebist vms kohast saada mingit infot, tuleb Pythonis kasutada sellist funktsiooni nagu **`input()`**.

Näiteks, kui küsida kasutajalt tema nime, siis tehakse seda nii:

```
>>> nimi=input()
Tiina
>>> nimi
'Tiina'
>>>
```

Kõigepealt võetakse kasutusele muutuja, kuhu kasutajalt saadud info salvestatakse. Muutuja saab väärtuseks **`input()`** funktsiooniga saadud info. Kui input on välja kutsutud, vajutatud enter (käsurea korral) või kui on pandud programm eraldi tööle, siis programm enne edasi ei lähe, kui on kasutajalt mingi sisendi saanud.

Kirjutan oma nime ning nüüd, kui ma muutuja uuesti käsureale kirjutasin, siis on sellesse juba minu andmed salvestatud.

Jäta meelde, et **`input()`** funktsioon loeb kasutajalt andmed ainult **sõne** kujul. Seega kui näiteks arvu arvamise mängus on vaja programmil kasutada arvu, sellega teha tehteid jms, siis peab alati sõne kujul oleva sisendi muutma eelnevalt kas täisarvuks või ujukomaarvuks, vastavalt sellele kuidas parasjagu vaja peaks olema.

```
    print ("Liiga suur!
arvamus = int(input())
# Suurenda arvamuste loe
```

"print" käsk, end="", koma ja +

Väljundi esitamisest on juba päris palju juttu olnud erinevate peatükkide juures. Siin aga tutvustan sulle **`print()`** käsu ja koma ja reavahetuse omavahelist seost.

Mäletad, kui tahtsime eriti pikka teksti väljastada nii, et see ka ilus oleks - erinevatele ridadele paigutatuna, siis tuli kasutada kolmekordseid jutumärke. Kui mul on aga vastupidine soov, tahan, et mitu teksti väljastataks kõik ühel real, siis tuleb **`print()`** käsu sulgudes viimasele kohale kirjutada selline käsk nagu **`end=""`** (end="" kustutab ära print funktsiooni vaikimisi sisse kirjutatud reavahetuse). Kuidas? Proovi järgmised kaks rida koodi eraldi programmina salvestada, pane käima ja vaata, mis tulemuse said. Siis kustuta *end* ära ja pane uuesti programm tööle.

```
print("Tere, ", end="")
print("mis su nimi on?")
```

Kui ma tahan aga väljastada ühe print() käsuga erinevat tüüpi andmeid, näiteks teksti, arvu ja siis veel kord teksti, siis tuleb panna erinevate andmetüüpide vahele lihtsalt koma.

```
print("Tere, ", end="")
print("mis su nimi on?")
nimi=input()
print("Tere,", nimi, ", kas tahaksid arvata minu mõeldud arvu?")
```

Panid tähele, et kui programm pöördub nimeliselt kasutaja poole, siis on väljastus valesti kirjutatud, sest koma ette jääb tühik. Kui sellist olukorda ei taheta, siis tuleb komade asemel kasutada + märki ja panna vajalikud tühikud juba jutumärkides oleva teksti juurde.

```
print("Tere, ", end="")
print("mis su nimi on?")
nimi=input()
print("Tere, "+nimi+ ", kas tahaksid arvata minu mõeldud arvu?")
```

Sisendi küsimine failist või veebist

Tavaliselt saadakse programmi sisend kasutajalt: tema nimi, arvu küsimine, nupu vajutus, hiire liigutamine vms. Kuid mõnikord on vaja sisendit saada ka failist või lausa veebist. Kuidas seda tehakse?

Info lugemine failist:

- võtame kasutusele muutuja, näiteks **`fail`**, millele omistame avada tahetava faili nime jutumärkides (fail peab asuma samas kataloogis, kus sinu programmgi. Katsetamiseks loo oma kataloogi üks lihtne tekstifail laiendiga .txt). Faili avatakse käsuga **`open()`**
- loeme faili sisu järgmisesse muutujasse käsuga **`fail.read()`** ja
- prindime sisu välja

```
fail = open("katsetus.txt")
failisisu = fail.read()
print(failisisu)
```

Info saamine veebist:

- veebist info kätte saamiseks peab kõigepealt appi võtma abimooduli **`import urllib.request`**.
- seejärel toimime samamoodi nagu failist lugemise korral. Võtame kasutusele muutuja, näiteks **`veeb`**, millele omistame veebiaadressi jutumärkides ja selle avamiseks kasutame **`urllib`** mooduli käske.
- võtame uue muutuja, kuhu salvestame veebist saadud info
- prindime info välja

```
import urllib.request
veeb = urllib.request.urlopen("http://math.ut.ee/~kull/salakiri.txt")
kiri = veeb.read()
print (kiri)
```

Märkus!
Kui sa katsetad veebist info kätte saamist koolis või kontoris või mõnes muus asutuses, võib juhtuda, et see ei tööta. Seda põhjusel, et paljudes asutustes on kasutusel proxid (programmid, mis on sillaks arvuti lokaalse võrgu ja interneti vahel), mis on nii seadistatud, et igasugused katsed otse internetti pääseda ei oleks võimalik – turvalisuse huvides.

Mida õppisid?

Päris palju uusi asju minu arvates!

Vaatame veelkord selle nimekirja üle. Sa õppisid:

- Pythonit installeerima
- kuidas IDLE käima panna?
- kuidas käsurida kasutada?
- nägid, kuidas Python arvutada oskab ja milleks on seda vaja
- kasutama IDLE tekstiredaktorit oma esimeste programmikeste tegemiseks
- kuidas panna programme Pythonis käima?
- ühtteist veateadetest
- kuidas Python jätab "meelde" asju, kasutades muutujaid?
- et muutujaid kutsutakse ka nimedeks või muutuja nimedeks
- et muutujate sisuks võib olla väga palju erinevat tüüpi infot (arvud, tekstid, muusika, pildid, objektid vms)
- kuidas panna Python arvutama erinevaid tehteid?
- mis vahe on täisarvudel ja ujukomaarvudel ja kuidas üht teiseks muuta?
- kuidas astendatakse Pythonis (see on teiste programmeerimiskeeltega võrreldes erinev)?
- mis on e-notatsioon?
- kuidas jäägiga jagada?
- ühte andmetüüpi teiseks muutma
- kuidas saada teada, millise andmetüübiga on muutuja?
- kuidas saadakse sisendit?
- kuidas väljastada mitme andmetüübiga asju segamini ühel real kasutades koma või +?
- kuidas väljastada pikka sõnet mitmel real?
- kuidas muuta sõnekujuline sisend arvuks?
- Kuidas saada sisendit failist või veebist?

TEE ISE!

Nüüd kõiki eelnevaid teadmisi kasutades, tuleb sul lahendada järgmised 5 ülesannet. Valmis ülesanded tuleb laadida kodutööde esitamise linkide alt Moodle keskkonda üles.

1. **Ülesanne: Tervitus.** Kirjuta programm, mis küsib kasutajalt kõigepealt mis on ta eesnimi, siis küsib tema perekonnanime ning seejärel tervitab kasutajat nii ees- kui perekonnanimega ühel real ja soovib talle edu.

2. **Ülesanne: Puu**. Kirjuta programm, mis küsib kasutajalt puu ümbermõõdu (cm) ja annab vastuseks puu läbimõõdu (diameetri) meetrites ja ümardatud kaks kohta peale koma. Kuna sul tuleb kasutada *piid*(3,14...), siis Pythonis on selleks eraldi käsk **`pi`**, kuid seda saab kasutada ainult siis, kui oled programmi kõige algusesse importinud abimooduli **`math`**. St. sa pead programmi kõige üles kirjutama järgmise rea: **`from math import *`**. Programmi käivitamisel peaks ekraanile ilmuma midagi sellist:

```
Mis on puu ümbermõõt (sentimeetrites)?
110
Puu läbimõõt meetrites on 0.35
```

3. **Ülesanne: Laulusõnad**. Leia veebist tekstifailina (.txt) mõne sulle huvitava laulu sõnad. Kirjuta programm, mis oskab leitud aadressilt need sõnad programmi sisse lugeda ja seejärel tulemuse ekraanile kirjutada.

4. **Ülesanne: Vanus**. Kirjuta programm, mis küsib kasutajalt tema nime ja tema sünniaega ning väljastab kasutajale kui mitu päeva ta vana on, kui on teada ka tänane kuupäev. Teeme hetkel lihtsuse mõttes nii, et kõik aastad on 365 päeva ja kõik kuud on 30 päeva pikad, sest me pole veel keerulisemaid konstruktsioone õppinud. Hiljem täiendame programmi. Programmi töö võiks välja näha midagi sellist:

   ps! Kui tahad aga väljakutset, siis võid uurida juba ette arvu arvamise koodi, kust leiad kindlasti juba huvitavamaid võtteid, kuidas võiks vanuse ülesande lahendada.

```
Mis su nimi on?
Juku
Mis aastal sa sündisid?
2002
Kirjuta ka oma sünnikuu number:
10
ja kuupäev
10
Hei, Juku, sa oled 3960 päeva vana.
```

5. **Ülesanne: ArvaArvu**. Uuri kõigepealt põhjalikult mängu ArvaArvu koodi ning seejärel tee programmis järgmised muudatused:
   - programm peaks mõtlema arvu, mis on alla 20, mitte alla 1000, tee parandus!
   - alla 20-st arvu peaks õige strateegia korral olema võimalik ära arvata 5 korraga. Tee selleks kõik vajalikud muudatused koodis.