

Programmeerimise aine lisamaterjalid

1. Standardteek ja moodulid

Õpiku esimeses peatükis sai uuritud math moodulit ning selle funktsioone ja konstante. Pythonis on palju rohkem selliseid mooduleid erinevate probleemide lahendamiseks.

Näiteks:

- Kuidas saada kätte homne kuupäev? Kuu lõpus võib muutuda küsimus väga keeruliseks, kui seda ise üritada arvutada. Mitu päeva on praeguses kuus? Veebruari lõpus: kas on liigaasta?
- Kuidas panna elemendid juhuslikku järjekorda?
- Kuidas uurida kausta sisu Pythoniga?
- Kuidas muuta faili nimesid?

Pythonisse sisse ehitatud moodulid teevad lahendused sellistele küsimustele lihtsaks. Selles peatükis vaatame mõne huvitavama mooduli kasutamise näited. Uurime, kuidas moodulid üldse töötavad ja kuidas teha enda lihtne moodul.

Ettevalmistus

Selle peatüki läbimiseks piisab Pythoni installatsioonist ning õpiku esimese peatüki lugemisest. Väikesel määral käsitletakse andmestruktuure, mida pole veel tutvustatud.

Et kõigest aru saada, on soovitatav tutvuda funktsioonidega ([peatükk 4](#)) ja järjenditega ([peatükk 7](#)).

Moodulite importimine

Tuletame kõigepealt meelde, kuidas mooduleid importida. Seda saab teha import lause abil:

```
>>> import math
>>> math.cos(math.pi)
-1.0
```

Moodulitele saab anda ka nimesid as lausega. Seda tehakse pigem harva ja pikemate või väga sagedasti kasutatavate nimedega. Tavaks on lühendada pandas moodul pd-ks ja numpy moodul np-ks.

```
>>> import math as m
>>> m.cos(m.pi)
-1.0
```

Moodulitest saab importida ainult vajalikud muutujad kasutades from lauset. Tervet moodulit sellega ei impordita. Mitu muutujat saab eraldada komaga.

```
>>> from math import pi, cos
>>> pi
3.141592653589793
>>> cos(pi)
-1.0
>>> math.pi
NameError: name 'math' is not defined
```

Tärni abil saab importida kõik mooduli muutujad.

```
>>> from math import *
>>> pi
3.141592653589793
>>> e
2.718281828459045
>>> tau
6.283185307179586
>>> cos(pi)
-1.0
```

Standardteek

Pythoniga tuleb kaasa palju mooduleid, mida nimetatakse kollektiivselt standardteegiks. Nende täisnimekiri on saadaval [Pythoni dokumentatsioonis](https://docs.python.org/3/library/math.html), aga vaatame lähemalt mõnda huvitavamat.

Moodul math

Selle mooduli abil saab ligi matemaatilistele funktsioonidele, mida Pythonis vaikimisi ei ole. Täisdokumentatsioon: <https://docs.python.org/3/library/math.html>

Importimine:

```
>>> import math
```

Kasulikud konstandid ja funktsioonid:

```
>>> math.pi # pii
3.141592653589793
>>> math.inf # lõpmatus (kõik arvud on võrdlemisel sellest väiksemad)
inf
>>> math.floor(1.7) # alla ümardamine
1
>>> math.ceil(1.3) # üles ümardamine
2
```

```
>>> math.sqrt(2) # ruutjuur
1.4142135623730951
>>> math.factorial(6) # faktoriaal
720
>>> math.log(128, 2) # Logaritmid
7.0
>>> math.cos(math.pi) # trigonomeetria
-1.0
```

```
>>> math.isclose(1.45, 1.46, rel_tol=0.01) # tõeväärtus, kas kaks arvu
on lähedikkude rel_tol piires
True
```

Mõned kasulikud matemaatilised funktsioonid on Pythonisse juba sisse ehitatud:

```
>>> round(1.7) # ümardamine
2
>>> round(1.337, 2) # kahe komakohani ümardamine
1.34
>>> abs(-5) # absoluutväärtus
5
>>> 2**0.5 # ruutjuur ilma math moodulita
1.4142135623730951
```

Moodul random

Selle mooduliga saab genereerida [pseudojuhuslikke](https://docs.python.org/3/library/random.html) arve, et tuua programmidesse juhuslikkust. Täisdokumentatsioon: <https://docs.python.org/3/library/random.html>

Importimine:

```
>>> import random
```

Kasulikud funktsioonid:

```
>>> random.randint(1, 10) # juhuslik täisarv kahe arvu vahel
7
>>> random.randint(1, 10)
10
```

```
>>> random.random() # juhuslik ujukomaarv 0 ja 1 vahel
0.8426622092891721
>>> random.uniform(1.5, 3) # juhuslik ujukomaarv kahe arvu vahel
1.9124507393369863
```

```
>>> pakk = ["ärtu", "ruutu", "poti", "risti"]
>>> random.choice(pakk) # juhuslik valik järjendist
"ruutu"
>>> random.shuffle(pakk) # järjendi segamine
>>> pakk
['ruutu', 'risti', 'poti', 'ärtu']
```

Moodul datetime

Selle mooduli abil saab käsitleda kuupäevi ja aegu. Siin näites luuakse objekte, millest räägitakse täpsemalt peatükis "Objektorienteeritud programmeerimine". Praeguseks olgu need lihtsalt keerulised muutujad, millega saab erinevaid asju teha. Täisdokumentatsioon: <https://docs.python.org/3/library/datetime.html>

Importimine:

```
>>> import datetime
```

Datetime objekti loomine (aasta, kuu, päev, tund, minut, sekund):

```
>>> kuupäev = datetime.datetime(2020, 2, 29, 12, 34, 56)
>>> kuupäev
datetime.datetime(2020, 2, 29, 12, 34, 56)
>>> kuupäev.year
2020
>>> kuupäev.second
56
```

Praeguse ajahetke saamine:

```
>>> praegu = datetime.datetime.now()
>>> praegu
datetime.datetime(2020, 2, 29, 18, 20, 33, 30651)
```

Datetime objekti vormindamine sõneks:

```
>>> praegu.strftime("%Y-%m-%d %H:%M:%S")
'2020-02-29 18:20:33'
```

Vormindamise märgendite nimekiri:

<https://docs.python.org/3/library/datetime.html#strftime-and-strptime-format-codes>

Datetime objekti parsimine sõnest.

```
>>> datetime.datetime.strptime("2020-02-29 18:20:33", "%Y-%m-%d
%H:%M:%S")
datetime.datetime(2020, 2, 29, 18, 20, 33)
```

Aegadele saab kestusi juurde liita ja maha lahutada, mille kaudu tekib uus datetime objekt. Liidame enne loodud kuupäevale juurde 13 tundi ja 37 minutit:

```
>>> kestus = datetime.timedelta(hours=13, minutes=37)
>>> kuupäev + kestus
datetime.datetime(2020, 3, 1, 2, 11, 56)
>>> kuupäev - kestus # Lahutada saab ka
datetime.datetime(2020, 2, 28, 22, 57, 56)
```

Seega homse kuupäeva saab kätte nii:

```
>>> homme = praegu + datetime.timedelta(days=1)
>>> homme
datetime.datetime(2020, 3, 1, 18, 20, 33, 30651)
```

Moodul os

Selle mooduliga saab ligi operatsioonisüsteemi funktsionaalsusele, näiteks failide asukohtadele.

Täisdokumentatsioon: <https://docs.python.org/3/library/os.html>

Importimine:

```
>>> import os
```

Praeguse tee (current working directory) saamine:

```
>>> os.getcwd()
'/home/kasutaja/Python/Moodulid/'
```

Failinimede järjendi saamine praeguses kaustas ja määratud kaustas:

```
>>> os.listdir()
['programm.py', 'pilt.jpg', 'Uus kaust', 'Lõputöö.pdf']
>>> os.listdir('/home/kasutaja/Python/Moodulid/Uus kaust')
['saladused.txt']
```

Failinime muutmine:

```
>>> os.rename("pilt.jpg", "foto.jpg")
>>> os.listdir()
['programm.py', 'foto.jpg', 'Uus kaust', 'Lõputöö.pdf']
```

Kausta loomine:

```
>>> os.mkdir("Teine kaust")
>>> os.listdir()
['programm.py', 'foto.jpg', 'Uus kaust', 'Lõputöö.pdf', 'Teine kaust']
```

Proovi kirjutada programm, mis loeb sisse kausta kõik failid ja väljastab nende sisu. Siin tuleb kasuks [for-tsükkel](#). Katseta seda kaustaga, mis sisaldab tekstifaile.

Moodul sys

Selle mooduliga saab ligi süsteempõhisele funktsionaalsusele. Täisdokumentatsioon: <https://docs.python.org/3/library/sys.html>

Importimine:

```
>>> import sys
```

Käsurea argumentidele saab ligi `sys.argv` muutujast:

```
>>> sys.argv  
['programm.py', 'esimene', 'teine']
```

Selline väärtus tuleb, kui käivitada Thonnys `programm.py` järgmise käsuga:

```
>>> % Run programm.py esimene teine
```

Või käsurealt:

```
kasutaja@arvuti ~/Python/Moodulid$ python3 programm.py esimene teine  
C:\Users\kasutaja\Python\Moodulid> python3 programm.py esimene teine
```

Täisarvu maksimaalne väärtus praeguses arvutis:

```
>>> sys.maxsize  
9223372036854775807
```

Pythoni interpretaatori versiooni saamine:

```
>>> sys.version  
'3.7.6 (default, Jan 19 2020, 22:34:52) \n[GCC 9.2.1 20200117]'
```

Programmi töö lõpetamine:

```
>>> sys.exit("Põhjus")
```

Moodul this

Selle mooduli importimine väljastab Pythoni põhimõtted (*The Zen of Python*). Proovi seda importida.

Moodul antigravity

Seda proovi ise importida. :)

Kuidas moodulid töötavad?

Importisime palju erinevaid mooduleid ja kasutasime nende konstante ja funktsioone, aga kuidas need üldse töötavad? Kuidas luua ise üks moodul?

Mooduli importimine tegelikult otsib üles sellenimelise Pythoni faili ja käivitab selle. Kõiki mooduli failis defineeritud muutujaid saab importivas programmis kasutada.

Isetehtud moodul

Proovime ise mooduli teha. Kirjutame programmi minumoodul.py, kus defineerime ühe konstandi ja lihtsa funktsiooni. Lisame ka ühe print-lause, et tõestada, kuidas kogu programm käivitub.

```
konstant = "Tere"

def ruut(n):
    return n**2

print("Moodul imporditud!")
```

Nüüd avame samas kaustas interpretaatori või loome uue Pythoni faili ja proovime tehtud moodulit importida ning muutujaid kasutada.

```
>>> import minumoodul
Moodul imporditud!
>>> minumoodul.konstant
'Tere'
>>> minumoodul.ruut(17)
289
```

Kõik muutujad töötavad ning sõnum väljastatakse. Üldiselt välditakse moodulite importimisega millegi väljastamist. Teised moodulid ju seda ei teinud.

Mooduli kõikide muutujate loetlemiseks on olemas sisseehitatud funktsioon `dir()`. Kui sulud tühjaks jätta, näidatakse kõiki muutujaid, mis on kogu programmis, s.h imporditud moodulid. Kui sulgude sisse panna muutuja, siis loetletakse selle muutujaga seotud muutujaid.

```
>>> dir()
['__annotations__', '__builtins__', '__doc__', '__loader__', '__name__',
 '__package__', '__spec__', 'math', 'minumoodul']
>>> dir(minumoodul)
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__',
 '__name__', '__package__', '__spec__', 'konstant', 'ruut']
>>> dir(math) # kui on imporditud
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos',
```

```
'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign',  
'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs',  
'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot',  
'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma',  
'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'pi', 'pow', 'radians',  
'remainder', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']
```

Kui kunagi satub ette võõras muutuja, on mõistlik jooksutada selle peal `dir()` funktsioon, et näha kõiki selle võimalusi. Katseta ka `help()` funktsiooni.

Saame ka uurida erinevate objektide funktsioone ja muutujaid. Vaatame eelnevalt mainitud `datetime` objekti võimalusi.

```
>>> from datetime import datetime  
>>> praegu = datetime.now()  
>>> dir(praegu)  
['__add__', '__class__', '__delattr__', '__dir__', '__doc__', '__eq__',  
 '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__',  
 '__init__', '__init_subclass__', '__le__', '__lt__', '__ne__',  
 '__new__', '__radd__', '__reduce__', '__reduce_ex__', '__repr__',  
 '__rsub__', '__setattr__', '__sizeof__', '__str__', '__sub__',  
 '__subclasshook__', 'astimezone', 'combine', 'ctime', 'date', 'day',  
 'dst', 'fold', 'fromisocalendar', 'fromisoformat', 'fromordinal',  
 'fromtimestamp', 'hour', 'isocalendar', 'isoformat', 'isoweekday',  
 'max', 'microsecond', 'min', 'minute', 'month', 'now', 'replace',  
 'resolution', 'second', 'strftime', 'strptime', 'time', 'timestamp',  
 'timetuple', 'timetz', 'today', 'toordinal', 'tzinfo', 'tzname',  
 'utcfromtimestamp', 'utcnow', 'utcoffset', 'utctimetuple', 'weekday',  
 'year']  
>>> praegu.year  
2020
```

Leia vastused küsimustele:

- Mida teeb `help()` funktsioon?
- Millised funktsioonid ja konstandid on `re` moodulil?
- Millised funktsioonid ja muutujad on `timedelta` objektil?

Moodulite asukohad

Kirjutasime enda mooduli interpretaatoriga samasse kausta ja saime seda importida, aga muud moodulid ei asu selles kaustas ja neid saab ikka importida. Kus need asuvad?

Pythonil on erinevate moodulite asukohtadest nimekiri nimega `path`. Sellele saab ligi mooduli `sys` muutujast.

```
>>> import sys  
>>> sys.path
```



```
[', '/usr/lib/python3.7.zip', '/usr/lib/python3.7',  
'/usr/lib/python3.7/lib-dynload',  
'/home/kasutaja/.local/lib/python3.7/site-packages',  
'/usr/local/lib/python3.7/dist-packages',  
'/usr/lib/python3/dist-packages', '/usr/lib/python3.7/dist-packages']
```

Mooduli importimisel otsitakse need kaustad järjest läbi, alustades esimesest. Esimene kaust on tühisõne ehk esimesena vaadatakse üle kaust, kus programm käivitati.

Imporditud moodulite failide asukohta saab ka näha:

```
>>> import turtle  
>>> turtle  
<module 'turtle' from '/usr/lib/python3.7/turtle.py'>  
>>> turtle.__file__  
'/usr/lib/python3.7/turtle.py'
```

Mõned moodulid on Pythoni lähtekoodi sisseehitatud ja nende lähtekoodile nii lihtsalt ligi ei pääse.

```
>>> import math  
>>> math  
<module 'math' (built-in)>  
>>> math.__file__  
AttributeError: module 'math' has no attribute '__file__'
```

Proovi uurida moodulite `this` ja `antigravity` lähtekoode. Kas need vastavad ootustele?

Kas programm käivitati või imporditi?

Igal programmil (ja seega moodulil) on muutuja `__name__` (kaks alakriipsu ees ja taga), mille väärtuseks on mooduli nimi sõnena. See muutuja on ka käivitataval programmil, aga selle väärtus on siis hoopis `"__main__"`.

```
>>> __name__  
'__main__'
```

Kui `__name__` väärtuseks on mooduli nimi, siis järelikut programm imporditi. Kui selle väärtus on `"__main__"`, siis see käivitati.

Tihti lisatakse programmi ette if-lause kontrolliga, kas programmi käivitati. Nii saavad teised arendajad programmi importida ja selle funktsioone kasutada ilma, et programmi põhiosa käivitub.

```
if __name__ == "__main__":  
    print("Programm käivitati, ei imporditud!")
```

Proovi seda järelle `minumoodul.py` programmis.

Enesekontrolliküsimused

1. Kuidas importida kõik mooduli muutujad?
 - `import all from moodul`
 - `from moodul import all`
 - `import *`
 - `import * from moodul`
 - `from moodul import *`
2. Mis mooduli ja muutujaga saab kätte käsurea argumendid?
 - `os.argv`
 - `cmd.args`
 - `sys.args`
 - `os.args`
 - `sys.argv`
3. Miks tuleb kontrollida programmis `if __name__ == "__main__"`?
 - Et programm näeks professionaalsem välja
 - Et teisi programme saaks programmi importida
 - Et seda programmi saaks importida teistesse programmidesse
 - Et Python teaks, millist kohta tuleb käivitada
 - Et programmi põhiosa ei käivituks importimisel

Ülesanded

1. Muuda `minumoodul.py` programmi nii, et see väljastab sõnumi ainult siis, kui see käivitatakse. Importimisel ei tohi midagi väljastada.

```
>>> %Run minumoodul.py
Programm käivitati, ei imporditud!
>>> import minumoodul
>>>
```

2. Kirjuta programm, mis võtab käsurealt sisendiks päevade arvu ning väljastab kuupäeva ja aja pärast sisestatud päevade arvu.

Näide:

```
>>> %Run ajaarvutaja.py 1337
Praegune aeg on 2020-02-29 18:20:33.
1337 päeva pärast on aeg 2023-10-28 18:20:33.
```

3. Google Mapsi koordinaatsüsteemi laiuskraadide piirid on -85 kuni +85 ning pikkuskraadide piirid on -180 kuni +180. Et minna Google Mapsi rakenduses mingitele koordinaatidele, saab minna veebilehitsejaga aadressile <https://maps.google.com/?q=laiuskraad,pikkuskraad>. Näiteks Delta hoone puhul <https://maps.google.com/?q=58.385894,26.725829>.

Kirjuta programm, mis genereerib juhusliku koordinaadipaari ja avab veebilehitsejaga selle asukoha Google Mapsis. Veebilehitseja peaks avama lehekülge samamoodi, nagu seda teeb `antigravity` moodul. Programmi koodi kirjuta kommentaar huvitava asukohaga, kuhu see sind viis.

4. Kirjuta programm, mis kasutab asjalikult kolme erineva mooduli funktsionaalsust. Võib kasutada mooduleid, mida materjalides ei ole käsitletud.

2. Rakendusliidesed

Õpiku teises peatükis räägiti enda arvuti failide lugemisest, aga alati ei pruugi vajalikud andmed olla enda failides olemas. Võibolla on vaja saada ajakohast infot muutuvate andmete kohta, näiteks ilmaennustused või valuutakursid. Võibolla on vaja midagi tõlkida Google Translate'iga, teha midagi Spotify andmetega, saada mingi sõna definitsioon või postitada midagi Twitterisse.

Sarnaselt lokaalsete failide lugemisega on võimalik lugeda veebilehti. Kõik veebilehed on justkui failid kellegi teise arvutis, mida veebilehitseja oskab graafiliseks liideseks teha. Suvaliste veebilehtede lugemine võib osutuda keeruliseks, aga Internetis on olemas palju selliseid lehti, mis on mõeldud programmidega lugemiseks. Neid nimetatakse rakendusliidesteks (ingl. k *API* ehk *Application Programming Interface*). Siin peatükis uurime, kuidas avada veebilehti nagu faile ja saada rakendusliidestelt andmeid.

Ettevalmistus

Enne jätkamist tuleb paigaldada moodul [requests](#). Seda saab teha käsuga `pip install requests`. Katsetamiseks kirjuta Pythonis `import requests`. Kui erindit ei visata, on moodul paigaldatud. Kui ei ole kindel, kuidas mooduleid installida, siis on õpikus [moodulite paigaldamise juhised](#).

Sellest peatükist arusaamiseks on vaja läbida õpiku esimesed 2 peatükki ning tutvuda 10. peatüki [sõnastiku andmestruktuuriga](#).

Veebilehtede lugemine

Pythoniga veebilehtede lugemise teeb väga lihtsaks moodul `requests` ja selle funktsioon `get()`, mis tagastab päringu objekti. Selle päringu lähtekoodi sõnena saab kätte väljagae `text`. Avame näiteks Tartu Ülikooli veebilehe ja väljastame selle lähtekoodi.

```
>>> import requests
>>> päring = requests.get("https://ut.ee/")
>>> päring.text
'<!DOCTYPE html PUBLIC "-//W3C//DTD HTML+RDFa 1.1//EN">\n<html
lang="et"...'
```

Väljastatud HTML-kood ei ole kasutajale ega programmile hästi loetav. Pythoniga on võimalik HTML-koodi parsida kasutades mooduleid [BeautifulSoup](#) või [lxml](#), aga siin me HTML-iga rohkem ei tegele. Vaatame hoopis rakendusliideseid, mis teevad programmidele andmete kättesaamise lihtsaks.

JSON-vorming

Et teha andmed programmile kergesti loetavaks, kasutavad paljud rakendusliidesed JSON-vormingut. See vorming lubab sõnedes hoida JavaScripti objekte (sealt nimi *JavaScript Object Notation*), mis on väga sarnased Pythoni [sõnastikega](#).

Üks JSON objekt näeb välja selline:

```
tudeng = {
    "nimi": "Algo",
    "vanus": 19,
    "oskab_pythonit": true,
    "hobid": ["Python", "Netflix"]
}
```

Requests moodul oskab sellised leheküljed lugeda Pythoni sõnastikeks meetodiga `json()`. Näiteks saab tudengi nime saada kätte koodiga `tudeng["nimi"]` ja esimese hobi saab kätte koodiga `tudeng["hobid"][0]`. Väärtusteks võib olla ka teine JSON-objekt ehk sõnastik. Milline näeks `tudeng` välja siis, kui võtme "hinded" all oleks sõnastik hinnetega? Kuidas siis saada kätte aine "Programmeerimine" hinne?

Rakendusliideste pärimine

Leiame ühe rakendusliidese, mis tagastab JSON-vormingus informatsiooni ja proovime seda lugeda. Üks selline on näiteks <https://exchangeratesapi.io/>, mis annab meile praegused valuutakursid. Proovime kätte saada EUR/USD kursi.

```
>>> import requests
>>> aadress = "https://api.exchangeratesapi.io/latest"
>>> päring = requests.get(aadress)
>>> vastus = päring.json()
>>> vastus
{'rates': {'CAD': 1.5623, 'HKD': 8.3849, 'ISK': 152.2, 'PHP': 55.593,
'DKK': 7.4731, 'HUF': 356.06, 'CZK': 27.606, 'AUD': 1.8635, 'RON':
4.8445, 'SEK': 11.1523, 'IDR': 17187.09, 'INR': 81.14, 'BRL': 5.6037,
'RUB': 86.9346, 'HRK': 7.608, 'JPY': 118.63, 'THB': 35.076, 'CHF':
1.0535, 'SGD': 1.5643, 'PLN': 4.5604, 'BGN': 1.9558, 'TRY': 7.0625,
'CNY': 7.6849, 'NOK': 12.3165, 'NZD': 1.8903, 'ZAR': 18.68, 'USD':
1.0801, 'MXN': 26.2955, 'ILS': 3.9802, 'GBP': 0.92985, 'KRW': 1364.14,
'MYR': 4.7686}, 'base': 'EUR', 'date': '2020-03-19'}
>>> vastus["rates"]["USD"]
1.0801
```

Tihti on vaja päringuid täpsustada. Näiteks eelmises näites tagastati meile kõikide valuutade väärtused euro suhtes, aga kuidas täpsustada baasvaluutat? Nagu dokumentatsioon meile

ütleb, tuleb pärida aadressi <https://api.exchangeratesapi.io/latest?base=USD>. See on tegelikult sama aadress mis enne, aga lisatud on parameeter "base" väärtusega "USD".

Parameetreid saab manuaalselt aadressi taha küsimärgiga lisada, nagu on näidisaadressilt näha. Lihtsam on lisada parameetrite sõnastik päringule kaasa. Eriti siis, kui peab lisama mitu parameetrit või kui parameetrite väärtused sisaldavad sümboleid.

```
>>> import requests
>>> aadress = "https://api.exchangeratesapi.io/latest"
>>> parameetrid = {"base": "USD"}
>>> päring = requests.get(aadress, params=parameetrid)
>>> päring.url
'https://api.exchangeratesapi.io/latest?base=USD'
>>> vastus = päring.json()
>>> vastus["base"]
'USD'
>>> vastus["rates"]["EUR"]
0.9258402
```

Proovi teha päring valuutakursside ajaloole: <https://api.exchangeratesapi.io/history>. Vaata dokumentatsioonist, millised parameetrid juurde lisama peab.

POST-päringud

Siiani oleme veebilehtede lugemisel tegelikult taustal teinud GET-päringuid. See on lihtsalt päring, mis annab veebilehele teada, et me tahame saada selle sisu. Neid kasutatakse siis, kui me ainult pärima ja tulemusena midagi ei muudeta. Kui serveris midagi muudetakse, kasutatakse POST-päringuid. POST-päringutega saadetakse veebilehtedele andmeid ilma, et neid aadressi taha lisatakse. Ka nende päringutega saadetakse üldiselt midagi vastu.

Lühendame näiteks internetiaadressi, kasutades teenust <https://rel.ink/>. Selle käigus tehakse serverisse kirje, et pikale aadressile vastab uus väiksem aadress, mille tõttu peab tegema POST-päringu. Rakendusliidese dokumentatsioon on kirjas, et peab tegema POST-päringu aadressile <https://rel.ink/api/links/> ja andma kaasa "url" parameetriga aadressi, mida tahame lühendada.

```
>>> import requests
>>> aadress = "https://rel.ink/api/links/"
>>> andmed = {"url": "https://progeopik.cs.ut.ee/"}
>>> päring = requests.post(aadress, data=andmed)
>>> päring.json()
{'hashid': '9mvm2g', 'url': 'https://progeopik.cs.ut.ee/', 'created_at': '2020-03-20T16:47:21.317166Z'}
```

<https://rel.ink/9mvm2g> viib nüüd aadressile <https://progeopik.cs.ut.ee/>.

Rakendusliideste nimekiri

Internetis leidub palju rakendusliideseid, mis tagastavad JSON-vormingus andmeid. Mõned näited:

- Valuutakursid: <https://exchangeratesapi.io/>
 - Praegused kursid euro suhtes: <https://api.exchangeratesapi.io/latest>
- Internetiaadresside lühendamine: <https://rel.ink/>
- Juhuslik "yes" või "no" ja vastav gif: <https://yesno.wtf/api>
- Chuck Norrisi naljad: <https://api.chucknorris.io/>
 - Juhuslik nali: <https://api.chucknorris.io/jokes/random>
- Ilmaennustus (ilma Eestita): <https://www.metaweather.com/api/>
 - Ilm Helsingis: <https://www.metaweather.com/api/location/565346/>
- Pokémonide andmed: <https://pokeapi.co/>
 - Pikachu: <https://pokeapi.co/api/v2/pokemon/pikachu>
- Koeratõud ja nende pildid: <https://dog.ceo/dog-api/documentation/>
 - Juhuslik pilt: <https://dog.ceo/api/breeds/image/random>
- Kasside faktid: <https://alexwohlbruck.github.io/cat-facts/docs/>
 - Juhuslik fakt: <https://cat-fact.herokuapp.com/facts/random>
- Nimepäevad: <https://api.abalin.net/documentation>
 - Täna eesti nimepäevad: <https://api.abalin.net/today?country=ee>

Paljud rakendusliidesed ei ole avalikult kättesaadavad ning nõuavad autentimist, et päringuid piirata. Nende kasutamiseks peab registreerima kasutajaks ning iga päringuga kaasa saatma kasutajaga seotud võtme.

Mõned näited autentimisega rakendusliidestest:

- Sõnastik: <https://developer.wordnik.com/>
- Ilmaennustus OpenWeatherMap (Eestiga): <https://openweathermap.org/api>
- Google Translate: <https://cloud.google.com/translate/docs/apis>
- Spotify: <https://developer.spotify.com/>

Sellistele teenustele tehakse tavaliselt eraldi Pythoni moodulid, et neid oleks kergem kasutada:

- Sõnastik: [wordnik-python](#)
- OpenWeatherMap: [pyowm](#)
- Google Translate: [googletrans](#)
- Spotify: [spotipy](#)
- Twitter: [python-twitter](#)
- Facebook Messenger: [fbchat](#)
- Reddit: [praw](#)

Mahukad nimekirjad rakendusliidestest:

- <https://github.com/public-apis/public-apis>
- <https://any-api.com/>
- <https://apilist.fun/>
- <https://public-apis.xyz/>

Enesekontrolliküsimused

1. Millal tehakse POST-päringuid GET-päringute asemel?
 - Siis, kui GET-päringuid on juba liiga palju tehtud
 - Siis, kui selle tulemusel serveris midagi muudetakse
 - Siis, kui on vaja päringut täpsustada
 - Alati peab tegema POST-päringuid, GET-päringud on ainult veebilehitsejatele
 - Siis, kui kuskile sisse logitakse
2. Kuidas saada kätte JSON-vormingus lehekülje andmed Pythoni sõnastikuna?
 - `requests.urlopen(aadress).to_dict()`
 - `requests.open(aadress).to_json()`
 - `requests.get(aadress, format="JSON")`
 - `requests.get(aadress).json()`
 - `json.loads(urllib.request.urlopen(aadress))`
3. Kuidas saata leheküljele andmeid POST-päringuga?
 - `requests.post(aadress, data=andmed)`
 - `requests.data(aadress, data=andmed)`
 - `requests.push(aadress, data=andmed)`
 - `requests.send(aadress, data=andmed)`
 - `requests.send_data(aadress, data=andmed)`

Ülesanded

1. Kirjuta valuutakalkulaator, kuhu sisestatakse nii baasvaluuta, soovitud valuuta ja raha kogus.

Näide:

```
>>> %Run valuutaarvutaja.py
Sisesta baasvaluuta: GBP
Sisesta teine valuuta: IDR
Sisesta kogus: 1
1 GBP on väärt 18262.95 IDR
```

2. Kirjuta programm, mis küsib kasutajalt sisendit, kasutab mõnda huvitavat rakendusliidest ning väljastab vastavalt sisendile rakendusliidestest saadud infot.

3. Regulaaravaldised

Paljud programmeerijatel ette tulevad ülesanded on seotud sõnade ja teksti analüüsiga. Sellised ülesanded võivad osutuda üpris keeruliseks. Näiteks, kuidas teha kindlaks, et kasutaja sisestas ikka korrektse e-posti aadressi? Kuidas leida üles tekstist kõik telefoninumbrid? Kuidas[1] eemaldada[5] Vikipeedia[15] artiklist[43] kõik[119] viited[327]? Neid ülesandeid võib ju proovida puhta Pythoniga lahendama hakata, aga selliseid ja keerukamaidki ülesandied on palju lihtsam ja kiirem lahendada regulaaravaldistega.

Ettevalmistus

Selle peatüki läbimiseks piisab Pythoni installatsioonist.

Et peatükist aru saada, peab läbima [õpiku](#) esimesed 3 peatükki ja tutvuma [järjenditega](#) 7. peatükist.

Regulaaravaldised

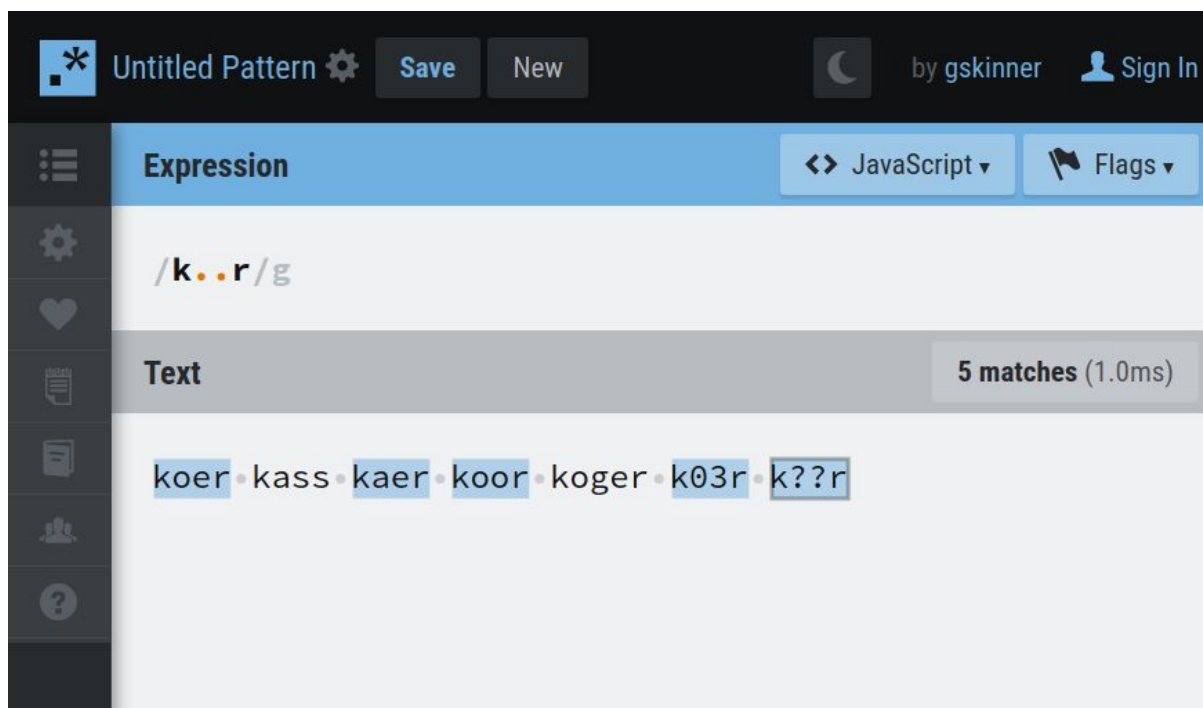
Regulaaravaldis on lihtsalt muster erinevatest sümbolitest, mille abil sõnade osasid otsida.

Kõige lihtsam regulaaravaldise muster oleks mingi sõna ise. Näiteks mustriga "koer" saab teha kindlaks, kas sõne algab osasõnega "koer", leida sõnest kõik osasõne "koer" esinemised või asendada sõnes kõik osasõned "koer" millegi muuga.

Selliseid ülesandeid on muidugi lihtne täita ka Pythonisse sisseehitatud funktsioonidega, vastavalt sõne meetodid `startswith()`, `find()` ja `replace()`. Ülesanne muutub keeruliseks, kui otsitav muster ei ole lihtsalt mingi sõne. Näiteks, kuidas leiame kõik neljatähelised sõnad, mis algavad k-tähega ja lõppevad r-tähega?

Suvalisi sümboleid saab regulaaravaldistes märkida punktiga. Eelmises lõigus esitatud küsimuse vastuseks sobib muster "k..r".

Regulaaravaldisi saab katsetada lehekülgedel <https://regexr.com/> ja <https://regex101.com>. Ülemisse tekstikasti kirjuta muster ja alumisse kirjuta tekste, millega mustrit katsetada. Mustrile vastavad osad värvitakse ära ning nende peale vajutades saab lisainfot. Katseta edaspidi kõik regulaaravaldiste oskused nende rakendustega läbi.



Erinevaid viise sümbolite tähistamiseks mustrites

Saime teada, et punktidega tähistatakse ükskõik mis sümboleid. Kui meile sobivad ainult mõned sümbolid, kasutame nurksulgi. Nende sisse lähevad sümbolid, mida otsime.

Sümbol	Tähendus
.	Sobivad kõik sümbolid.
[abc]	Sobivad sümbolid a, b ja c. Nurksulgude vahele võib kirjutada mistahes sümboleid.
[a-z]	Sobivad tähed a kuni z ladina tähestikus.
[A-Za-z0-9]	Sobivad tähed A kuni Z, a kuni z ja numbrid 0-9.
\w	Sobivad kõik numbrid ja tähed (k.a täpitähed) ja alakriipsud.
\d	Kõik arvud. Sama, mis [0-9].

Näited:

Muster	Sobivad	Ei sobi
koer	koer	kõik muu
k..r	koer, kaer, koor, k03r, ...	koger, kass, kor, kr, ...
[ktpb]ass	kass, tass, pass, bass	kõik muu
[0-9][0-9][0-9][A-Z][A-Z][A-Z]	123ABC, 420BLZ, 111YKS jms tavalised auto	kõik, mis ei ole tavalised auto numbrimärgid

	numbrimärgid	
\w\w\w\w\w\w	kõik 6-tähemärgilised sõnad, mis sisaldavad tähti, numbreid ja alakriipse	kõik muu

Enesekontroll

1. Millise regulaaravaldisega leiame kõik eestikeelsed neljatähelised sõnad, mis algavad k- ja lõpevad r-tähega? Vahepealsed tähed ei tohiks olla numbrid ega sümbolid.
 - k.r
 - k\\r
 - k\w\wr
 - k[a-z][a-z]r
 - k[a-zžšõäöü][a-zžšõäöü]r
2. Millised sõned sobivad mustriks "l.h."?
 - lehm
 - lagi
 - lehed
 - l3h?
 - lohe
3. Millised sõned sobivad mustriks "\w\w\w\d"?
 - 123a
 - nCk1
 - kaks
 - 1337
 - o_o7

Üks või teine, grupeerimine

Erinevate tähtede võimalusi sai määrata nurksulgudega. Pikemaid võimalusi tuleb eraldada püstkriipsudega ja vajadusel paigutada sulgude sisse.

Sümbol	Tähendus
	Või-märk. Sobib vasakule poole jääv grupp või paremale poole jääv grupp.
()	Grupeerimismärgid. Grupid tuleb ümbritseda sulgudega.

Näited:

Muster	Sobivad	Ei sobi
koer kass	koer, kass	kõik muu
k(oer ass)	koer, kass	kõik muu
k(o a)er	koer, kaer	kõik muu

(ko ka)(er ss)	koer, kass, kaer, koss	kõik muu
(je bo ka)ss	jess, boss, kass	kõik muu
aias sadas (sai leib)a	aias sadas saia, aias sadas leiba	kõik muu

Enesekontroll

1. Millised sõned sobivad mustriks "sõitsin (rat|au)t(o|a)ga koju"?
 - ☒ sõitsin autoga koju
 - ☒ sõitsin rattaga koju
 - ☐ sõitsin rotiga koju
 - ☐ sõitsin ratoga koju
 - ☒ sõitsin autaga koju
2. Millised sõned sobivad mustriks "(ab|cd)(ef|gh)(ij|kl)"?
 - ☐ abcdef
 - ☐ aeibfj
 - ☐ cdefgh
 - ☒ abghij
 - ☐ abefijkl

Kordused

Mustriks saab veel ära määrata, et mõni sümbol või grupp saab olla sõnes mitu korda.

Sümbol	Tähendus
?	Eelnev sümbol/grupp kas on või ei ole.
*	Eelnevat sümbolit/gruppi on null või rohkem kordi.
+	Eelnevat sümbolit/gruppi on üks või rohkem kordi.
{5}	Eelnevat sümbolit/gruppi on täpselt 5 korda.
{5,}	Eelnevat sümbolit/gruppi on 5 või rohkem kordi.
{1,5}	Eelnevat sümbolit/gruppi on 1 kuni 5 korda (1 ja 5 kaasa arvatud).

Näited:

Muster	Sobivad	Ei sobi
jaa?	ja, jaa	kõik muu
(mitte)?olla	olla, mitte olla	kõik muu
ja*	j, ja, jaa, jaaa, ...	kõik muu
ja(ja)*	ja, jaja, jajaja, ...	kõik muu

<code>ja+</code>	ja, jaa, jaaa, jaaaa, ...	kõik muu
<code>ja{2,4}</code>	jaa, jaaa, jaaaa	kõik muu
<code>[0-9]{3}[A-Z]{3}</code>	tavalised auto numbrimärgid	mitte tavalised auto numbrimärgid
<code>\w{6}</code>	kõik 6-tähemärgilised sõnad, mis sisaldavad tähti, numbreid ja alakriipse	kõik muu
<code>.+</code>	kõik sõned, kus on vähemalt midagi	tühisõne

Enesekontroll

1. Millised sõned sobivad mustriksile "ha{3,}"?
 - ☐ hahahaha
 - ☐ hahaha
 - ☒ haaaa
 - ☐ haahaahaa
 - ☐ haha
2. Millised sõned sobivad mustriksile "(no)*(nii)+"?
 - ☒ nonii
 - ☐ noniinonii
 - ☒ niinii
 - ☒ nononinii
 - ☐ nonono

Algus ja lõpp

Et täpsustada sõne algust ja lõppu, on olemas märgid `^` ja `$`.

Sümbol	Tähendus
<code>^</code>	Sõne algus.
<code>\$</code>	Sõne lõpp.

Näited:

Muster	Sobivad	Ei sobi
<code>^a</code>	aabits..., arvuti..., algebra..., a123..., a...	muude sümbolitega algavad sõned
<code>ass\$</code>	...kass, ...bass, ...kontrabass, ...	muude sümbolitega lõppevad sõned
<code>^koer\$</code>	koer	kõik muu

Erisümbolid

Oleme vaadanud palju erisümboleid, millega regulaaravaldise mustreid koostada:

. [] \ | () ? * + { } ^ \$

Aga kuidas neid sümboleid sõnedes otsida? Samamoodi, nagu Pythonis tehakse sõne sees jutumärke, saab ka regulaaravaldistes teha erisümboleid: nende ette tuleb panna langkriips. Inglise keeles öeldakse selle kohta *escaping*. Lühidalt: et otsida punkti, peab otsima "\.", küsimärgi otsimiseks peab otsima "\?", plussi otsimiseks "\+" jne.

Muster	Sobivad	Ei sobi
\.\.\.	...	kõik muu
youtube\.com	youtube.com	kõik muu
\(.*\)	(kass), (koer), (abc123), ...	mitte sulgudes olevad sõned
\[\d+\]	[1], [42], [1632], ...	[], mitte nurksulgude vahel olevad arvud
~_\(ツ\)_\/~	~_ (ツ) _/ ~	kõik muu

Harjutamine

Regulaaravaldisi saab harjutada leheküljel [RegexOne](#). Proovi ka regulaaravaldiste [ristsõnu](#) või [golfi](#).

Regulaaravaldised Pythonis

Kasutame regulaaravaldisi lõpuks praktikas. Et Pythonis regulaaravaldisi kasutada, tuleb importida moodul `re`. Selle nimi on lühend ingliskeelsest terminist *regular expressions*.

```
>>> import re
```

Mustreid defineeritakse justkui sõnesid, aga jutumärkide ette tuleb panna r-täht. See lubab sõnedes kasutada langkriipse ilma teise langkriipsuta.

```
>>> muster = r"\[\d+\]"
>>> muster
'\\[\\d+\\]'
```

Sõne vastavus mustriga

Uurime, kas muster `"k..r"` vastab erinevatele sõnedele. Seda saab teha funktsiooni `re.match()` abil. See funktsioon tagastab objekti, kui sõne vastab mustrile. Vastasel juhul ei tagasta see midagi.

```
>>> muster = r"k..r"
```

```
>>> re.match(muster, "koer")
<re.Match object; span=(0, 4), match='koer'>
>>> re.match(muster, "kaer")
<re.Match object; span=(0, 4), match='kaer'>
>>> re.match(muster, "kass")
>>>
```

Tegelikult kontrollib `re.match()` ainult, kas muster vastab sõne algusele. Et kontrollida tervet sõnet, on mõistlik ette panna `^` ja `taha $`.

```
>>> re.match(r"k..r", "koerad")
<re.Match object; span=(0, 4), match='koer'>
>>> re.match(r"^k..r$", "koerad")
```

Seda saab ära kasutada if-lauses:

```
import re

kasutajanimi = input("Sisesta kasutajanimi: ")
kasutajanimi_regex = r"^[a-z0-9_-]{3,20}$"

if re.match(kasutajanimi_regex, kasutajanimi):
    print("See kasutajanimi sobib!")
else:
    print("See kasutajanimi ei sobi.")
```

Millised kasutajanimid programmile meeldivad? Proovi erinevaid variante.

Sarnast kontrolli saab teha e-posti aadressidega, aga muster on keerulisem:

```
# https://emailregex.com/
email_regex = r"^[a-zA-Z0-9_+-.]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-]+$.+$)"
```

Proovi muuta programmi nii, et kontrollitakse telefoninumbri korrektsust.

Mustrite otsimine sõnedes

Funktsioon `re.findall()` leiab üles tekstist kõik mustri vastavused:

```
>>> tekst = "koer kass kaer koor koger k03r k??r"
>>> muster = r"k..r"
>>> re.findall(muster, tekst)
['koer', 'kaer', 'koor', 'k03r', 'k??r']
```

Proovi leida tekstist erinevaid mustreid: auto numbrimärgid, kasutajanimed, e-posti aadressid, telefoninumbrid.

Mustrite asendamine sõnedes

Funktsiooniga `re.sub()` (sõnast *substitute*) saab asendada tekstis kõik mustri vastavused:

```
>>> tekst = "koer kass kaer koor koger k03r k??r"
>>> muster = r"k..r"
>>> asendus = "kass"
>>> re.sub(muster, asendus, tekst)
'kass kass kass kass koger kass kass'
```

Selle koodiga asendasime kõik mustrile `"k..r"` vastavad osasõned sõnega `"kass"`.

Kopeerisid Vikipeediast pika artikli, aga teksti sisse jäävad nurksulgudes viited.

```
>>> tekst = "Python was conceived in the late 1980s[34] by Guido van
Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a
successor to the ABC language (itself inspired by SETL),[35] capable of
exception handling and interfacing with the Amoeba operating system.[8]
Its implementation began in December 1989.[36]"
```

Nende eemaldamine manuaalselt on tüütu ja neid koodiga eemaldada võib esmapilgul tunduda keeruline, aga regulaaravaldised teevad selle väga lihtsaks:

```
>>> muster = r"[\d+]"
>>> re.sub(muster, "", tekst)
'Python was conceived in the late 1980s by Guido van Rossum at Centrum
Wiskunde & Informatica (CWI) in the Netherlands as a successor to the
ABC language (itself inspired by SETL), capable of exception handling
and interfacing with the Amoeba operating system. Its implementation
began in December 1989.'
```

Veel regulaaravaldisi?

Regulaaravaldistega on muidugi võimalik palju rohkem teha. Sellel kursusel rohkemat ei käsitleta. Regulaaravaldisi vaadatakse põhjalikumalt aines "Automaadid, keeled ja translaatorid" (LTAT.03.006).

Üks kasulik mainimata jäänud regulaaravaldise võimalus on gruppidest väärtuste saamine.

Tasub ka uurida mooduli `re` dokumentatsiooni: <https://docs.python.org/3/library/re.html>

Enesekontrolliküsimused

1. Mis on vahet sõne meetodil `replace()` ja regulaaravaldiste funktsioonil `sub()`?
 - Neil ei ole mingit vahet
 - `sub()` töötab kiiremini kui `replace()`
 - `sub()` kasutab otsimiseks regulaaravaldisi, `replace()` ainult sõnesid
 - `replace()` asendab ainult esimese leitud osa
 - `sub()` asendab ainult esimese leitud osa
2. Mis funktsioonidega saab regulaaravaldistega kontrollida vastavust, leida kõik vastavused ja asendada kõik vastavused millegi muuga?

- `match()`, `findall()`, `sub()`
 - `compare()`, `search()`, `replace()`
 - `match()`, `find()`, `replace()`
 - `compare()`, `findall()`, `replace()`
 - `match()`, `find()`, `sub()`
3. Millised sõned sobivad mustriks "`^#?[a-f0-9]{6}|[a-f0-9]{3}$`"?
- `#2196f3`
 - `bada55`
 - ☐ `#cabage`
 - `#fb1`
 - ☐ `3f51p5`

Ülesanded

1. Kirjuta programm `numbrileidja.py`, mis leiab failist üles kõik telefoninumbrid järgmise vorminguga:

- Alguses on +372, aga mitte alati.
- Järgmisena võib tulla tühik.
- Järgmisena tuleb 3 või 4 numbrit.
- Järgmisena võib tulla tühik.
- Järgmisena tuleb 4 numbrit.

Mõne telefoninumbri näited:

- `+372 1234 5678`
- `1234 5678`
- `123 4567`
- `+37212345678`
- `+3721234567`
- `+372 12345678`
- `+372123 4567`

Näiteks `kontakt.txt` sisu (andmed pärit [Riigikogu leheküljelt](#)):

Riigikogu
Henn Põlluaas +372 631 6301 henn.polluaas@riigikogu.ee
Helir-Valdor Seeder +3726316311 helir-valdor.seeder@riigikogu.ee
Siim Kallas +372 6316321 siim.kallas@riigikogu.ee
Tiiu Pohl 6316302 tiuu.pohl@riigikogu.ee

`numbrileidja.py` käivitamine:

Sisesta failinimi: `kontakt.txt`
Leitud telefoninumbrid:
+372 631 6301
+3726316311
+372 6316321
6316302

4. Andmebaasid

Siiani oleme andmeid lugenud ja salvestanud tekstifailidesse. Väheste andmetega on see lihtne ja kiire lahendus, aga mida rohkem andmeid koguneb, seda aeglasemaks läheb nende otsimine. Mida keerulisemaks lähemad andmestruktuurid, seda keerulisem on neid failides hoida. Kõigi selle lahendamiseks on välja töötatud andmebaasisüsteemid, mis lubavad hoida palju keerulisi andmeid organiseeritult nii, et neile pääseb kiiresti ligi.

Mõned sellised süsteemid on MySQL, PostgreSQL, MongoDB, Oracle Database ja Sybase SQL Anywhere. Siin õppematerjalides keskendume hoopis süsteemile SQLite. See on üks lihtsamatest süsteemidest, mida kasutavad tihti rakendused andmete hoidmiseks. Seda kasutavad näiteks Apple, Google, Facebook, Firefox ja [paljud muud](#). Kõige tähtsam: see on Pythonisse sisse ehitatud!

Ettevalmistus

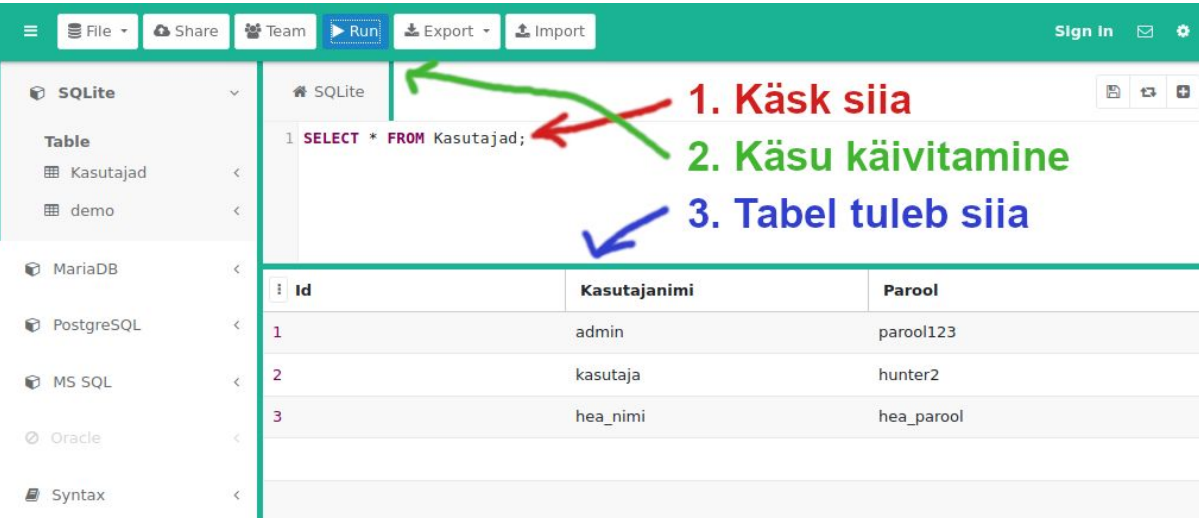
Jätkamiseks ei pea paigaldama lisatarkvara. Võibolla tuleb kasuks [SQLite Browser](#).

Et peatükist aru saada, peab läbima [õpiku](#) esimesed 3 peatükki ning tutvuma 7. peatüki [järjendite](#) ja [ennikutega](#).

SQL keel

Paljudel eelmainitud andmebaasi süsteemidel on midagi ühist: need kasutavad SQL-i. SQL (*Structured Query Language*) on päringukeel, mida kasutatakse andmebaaside loomiseks, andmete kirjutamiseks, pärimiseks ja muutmiseks ning paljuks muuks.

SQLite'i saab katsetada lehel [SQLite Online](#). Kõik edaspidised käsud tasub sinna kirjutada ja läbi proovida.



The screenshot shows the SQLite Online web interface. On the left, there is a sidebar with a list of databases: SQLite, MariaDB, PostgreSQL, MS SQL, Oracle, and Syntax. The 'SQLite' database is selected. In the main area, a SQL query is entered: `1 SELECT * FROM Kasutajad;`. Below the query, the results are displayed in a table with three columns: Id, Kasutajanimi, and Parool. The table contains three rows of data. Three annotations with arrows point to the query: a red arrow points to the word 'FROM' with the text '1. Käsk siia', a green arrow points to the word 'SELECT' with the text '2. Käsü käivitamine', and a blue arrow points to the table name 'Kasutajad' with the text '3. Tabel tuleb siia'.

Id	Kasutajanimi	Parool
1	admin	parool123
2	kasutaja	hunter2
3	hea_nimi	hea_parool

Tabeli loomine

Enne, kui saame andmeid andmebaasi salvestada, peame looma tabeli, mis kirjeldab andmete struktuuri.

Oletame, et meil on veebileht, kus saab kasutajaks registreerida ja sisse logida. Igal kasutajal on mingi järjenumber ehk ID, tekstiline kasutajanimi ja tekstiline parool. Selliseid omadusi nimetatakse väljadeks.

Looime tabeli selliste väljadega ja paneme selle nimeks "Kasutajad". Tabelit saab luua käsuga `CREATE TABLE Tabel (...)` ning sulgudesse lähevad väljade nimed ja nende tüübid:

```
CREATE TABLE Kasutajad (id INTEGER PRIMARY KEY, kasutajanimi TEXT,
parool TEXT);
```

Kõik käsud tuleb lõpetada semikooloniga.

PRIMARY KEY tähendab seda, et see on põhiline väli, millega sellele kasutajale viidata saab ning et iga järgmine kasutaja saab ühe võrra suurema järjenumbri.

Andmete lisamine tabelisse

Nüüd on tabel olemas ja puudu on ainult andmed. Andmeid saab tabelisse lisada käsuga `INSERT INTO Tabel (...) VALUES (...)` nii, et esimestesse sulgudesse lähevad väljade nimed ja teistesse lähevad nende väärtused:

```
INSERT INTO Kasutajad (kasutajanimi, parool) VALUES ('admin',
'parool123');
```

Väljale ID ei pea väärtust omistama, sest siis määratakse selle väärtuseks automaatselt alguses 1, edaspidi tuleb selle väärtus eelmisest ühe võrra suurem. See on PRIMARY KEY tõttu. Sõned peavad olema ümbritsetud ülakomadega, mitte jutumärkidega.

Lisame veel mõned kasutajad:

```
INSERT INTO Kasutajad (kasutajanimi, parool) VALUES ('kasutaja',
'hunter2');
INSERT INTO Kasutajad (kasutajanimi, parool) VALUES ('hea_nimi',
'hea_parool');
```

NB! Paroole ei tohi kunagi salvestada tavalise tekstina. Neid tuleks räsida ühepoolse algoritmiga nagu SHA-256. Näiteks tuleb "parool123" asemel salvestada andmebaasi räsi "f9c80861456cdd34bebfa8886ae3436f22bbc7343e27df6a3376bcce23ed330d". Parooli kaudu on võimalik saada räsi, aga räsi kaudu ei ole võimalik saada parooli. Kui keegi tahab sisse logida, tuleb sisselogimisparool ka räsida ja seejärel räsidsid võrrelda. Kui räsidsid on võrdsed, siis sisestatud parool on õige. Räsimine on tähtis, sest kui keegi kunagi andmebaasile ligi saab, ei saada kätte paroole. Pythonis saab räsidsid arvutada näiteks [bcrypt](#) mooduliga.

Andmete pärimine tabelist

Kui andmed on tabelisse lisatud, peab neid kuidagi kätte saama. Seda tehakse SELECT käsuga. Et kuvada kõik tabeli read, saab kirjutada käsu:

```
SELECT * FROM Kasutajad;
```

id	kasutajanimi	parool
1	admin	parool123
2	kasutaja	hunter2
3	hea_nimi	hea_parool

Tärn tähendab seda, et tagastatakse kõik väljad. Saab ka valida, milliseid tagastada:

```
SELECT id, kasutajanimi FROM Kasutajad;
```

id	kasutajanimi
1	admin
2	kasutaja
3	hea_nimi

Saame täpsustada, milliseid andmeid tahame võtmesõnaga WHERE:

```
SELECT parool FROM Kasutajad WHERE kasutajanimi='admin';
```

parool
parool123

Andmete muutmine

Kui on vaja muuta näiteks kasutaja parooli, tuleb teha UPDATE käsk.

```
UPDATE Kasutajad SET parool='hunter3' WHERE kasutajanimi='kasutaja';
```

Tabeli kustutamine

Tabelit saab kustutada käsuga DROP TABLE. Selle käigus kustutatakse ka kõik read, seega selle käsuga peab olema ettevaatlik.

```
DROP TABLE Kasutajad;
```

Veel SQL-i!

SQL-il on veel palju rohkem võimalusi nagu mitme tabeli omavahelised suhted, vaated, funktsioonid jne. Siin kursusel keerulisemat SQL-i ei käsitleta. Andmebaasidesse ja SQL-i süvenetakse täies rauas aines "Andmebaasid" (LTAT.03.004). Enne seda saab SQL-i harjutada ja edasi õppida [SQLZOO](#) leheküljel.

Enesekontroll

1. Kuidas luua andmebaas linnade salvestamiseks nimega ja rahvaarvuga?
 - CREATE TABLE Linnad (TEXT nimi, INTEGER rahvaarv);
 - TABLE CREATE Linnad (NAME nimi, NUMBER rahvaarv);
 - CREATE TABLE Linnad (nimi TEXT, rahvaarv INTEGER);
 - TABLE Linnad CREATE (TEXT nimi, INTEGER rahvaarv);
 - TABLE CREATE Linnad (nimi TEXT, rahvaarv INTEGER);
2. Kuidas lisada sinna tabelisse Seoul rahvaarvuga [38000000](#)?
 - INSERT INTO Linnad (nimi, rahvaarv) VALUES ('Seoul', 38000000);
 - ADD TO Linnad ("Seoul", 38000000);
 - TABLE Linnad (nimi, rahvaarv) ADD VALUES ("Seoul", 38000000);
 - INSERT INTO Linnad ('Seoul', 38000000);
 - ADD TO Linnad (nimi, rahvaarv) VALUES ('Seoul', 38000000);
3. Kuidas pärida sellest tabelist kõik linnade nimed?
 - SELECT * FROM Linnad;
 - SELECT nimi FROM Linnad;
 - FROM Linnad SELECT nimi;
 - GET (nimi) FROM Linnad;
 - QUERY FROM Linnad (nimi);

Andmebaasid Pythoniga

Kõige põhilisemad SQL-käsud on nüüd selged. Proovime teha samasuguse andmebaasi Pythonis. Nagu eelnevalt mainitud, on SQLite Pythonisse sisseehitatud. Selle kasutamiseks tuleb importida moodul sqlite3.

```
>>> import sqlite3
```

Mooduli dokumentatsioon: <https://docs.python.org/3/library/sqlite3.html>

Andmebaasiga ühendamine

Et andmebaasiga suhelda, tuleb kõigepealt luua ühendus. See on sarnane faili avamisele. Ühtlasi hoitakse SQLite andmebaase failides. Teiste andmebaasisüsteemidega on tavaliselt keerulisem ühendada.

```
>>> ühendus = sqlite3.connect("andmebaas.db")
```

Kui sellise failinimega andmebaasi veel pole, siis see luuakse automaatselt.

Et andmebaasi kärke saata, peab veel tegema kursori, mille kaudu seda teha.

```
>>> kursor = ühendus.cursor()
```

Nüüd saab saata SQL-kärke `execute()` meetodiga. Looime kõigepealt sama tabeli, mille lõime enne.

```
>>> käsk = "CREATE TABLE Kasutajad (id INTEGER PRIMARY KEY, kasutajanimi  
TEXT UNIQUE, parool TEXT);"  
>>> kursor.execute(käsk)
```

Veel ei ole tegelikult andmebaasis ühtegi muudatust tehtud. Kui käivitatakse kaks, mis andmebaasis midagi muudavad, on vaja need ka kinnitada.

```
>>> ühendus.commit()
```

Sarnaselt failidega tuleb andmebaasi ühendus pärast kasutamist kinni panna:

```
>>> ühendus.close()
```

Kui ühendus on avatud, ei saa teised programmid samal ajal sama andmebaasiga ühendada: visatakse erind, et andmebaas on lukus.

Andmete lisamine

Looime uuesti ühenduse andmebaasiga.

Andmete lisamiseks saab muidugi lihtsalt teha `kursor.execute()` nagu enne ja sisestada `INSERT` käsk, aga siin tuleb väga ettevaatlik olla, sest andmebaasi lisatavad andmed tulevad tihti kasutajatelt.

Oletame, et saame kasutajalt kasutajanime ja parooli ning meie käsk on järgmine:

```
>>> käsk = "INSERT INTO Kasutajad (kasutajanimi, parool) VALUES ('" +  
kasutajanimi + "', '" + parool + "');"
```

Kui sisestatud kasutajanimi ja parool on tavalised, siis on käsk korralik:

```
>>> käsk  
"INSERT INTO Kasutajad VALUES ('kasutaja', 'hunter2');"
```

Aga mis siis, kui muutuja `kasutajanimi` saab sellise väärtuse?

```
>>> kasutajanimi = "'; DROP TABLE Kasutajad;"
```

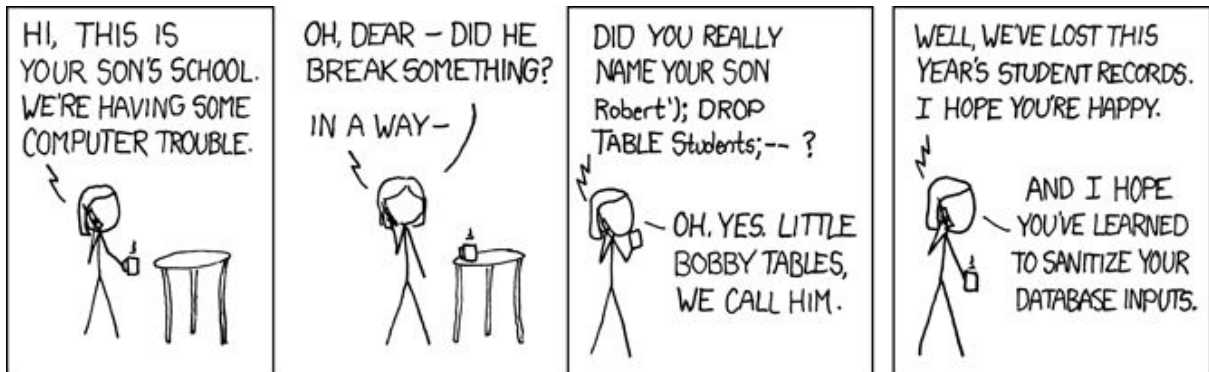
Siis terve käsk oleks selline:

```
>>> käsk
```

```
"INSERT INTO Kasutajad VALUES (''; DROP TABLE Kasutajad; , '');"
```

INSERT käsk lõpetatakse varakult ära, sest kasutajanimi lõpetas ülakomaga sõne ära ja semikooloniga. See ei ole korrektne käsk, sest üks muutuja on puudu ja seda ignoreeritakse. Siis tehakse käsk DROP TABLE Kasutajad, mis kustutab terve tabeli ära. Lõpus on mõned sümbolid, mis ei tee ka midagi. Kasutajal õnnestus andmebaasile ligipääsuta kustutada ära terve tabel.

Sellist manöövrit kutsutakse [SQL-i süstimiseks](#) (ingl. k. *SQL injection*) ja see on väga levinud rünnak rakenduste vastu:



[XKCD 327: Exploits of a Mom](#)

Selle vältimiseks on sqlite3 moodulil võimalused olemas: muutujad tuleb asendada küsimärkidega ja nende väärtused lähevad ennikuna teise parameetrisse.

```
>>> kasutajanimi = "admin"
>>> parool = "parool123"
>>> käsk = "INSERT INTO Kasutajad VALUES (NULL, ?, ?);"
>>> kursor.execute(käsk, (kasutajanimi, parool))
```

Nii ei lõpeta ülakoma sõne ja kasutajanimeks saab päriselt see, mis on muutuja väärtus.

NB! Ühe elemendiga ennik on Pythonis vormingus (muutuja,). Lihtsalt sulgudes muutuja on sama, mis muutuja ise.

Lisa ka teised kasutajad tabelisse. Et muutused sisse läheks, peab jälle ühenduse kinnitama.

```
>>> ühendus.commit()
```

Andmete pärimine

SELECT käskude tulemused saab kätte kursorist fetchall() meetodiga.

```
>>> kursor = ühendus.cursor()
>>> kursor.execute("SELECT * FROM Kasutajad;")
>>> kursor.fetchall()
[(1, 'admin', 'parool123'), (2, 'kasutaja', 'hunter2'), (3, 'hea_kasutaja', 'hea_parool')]
```


Tagastatakse järjend ennikutest, mis sisaldavad iga rea andmeid soovitud järjekorras. Kui järjekorda pole täpsustatud ja on kasutatud täрни, võetakse järjekord tabeli loomise käsust: id, kasutajanimi, parool.

Keerulisemad andmebaasid

Siin kursusel rohkem andmebaase ei käsitleta. Kui sarnaselt jätkata `sqlite3` mooduliga, võivad mahukad projektid minna liiga keeruliseks. Selle vastu on loodud moodulid nagu [SQLAlchemy](#) ja [Peewee](#), mis viivad vastavusse andmebaasi tabelid ja Pythoni objektid (ingl. k. *Object Relational Mapping*). See teeb andmebaasiga suhtlemise koodi loetavamaks ning SQL-i ei pea kirjutamagi. Pythoni objektidest räägitakse lähemalt peatükis "Objektorienteeritud programmeerimine".

Enesekontrolliküsimused

1. Millised on korrektsed tabeli loomise SQL käsud?
 - ☐ `CREATE TABLE riigid {TEXT nimi, INTEGER rahvaarv, INTEGER pindala};`
 - ☒ `CREATE TABLE autod (ID INTEGER PRIMARY KEY, mark TEXT, mudel TEXT, uste_arv INTEGER);`
 - ☐ `CREATE TABLE tudengid (TEXT matriklinr, TEXT eesnimi, TEXT perenimi);`
 - ☐ `CREATE TABLE ained (TEXT ainekood, TEXT nimi, NUMBER kohtade_arv);`
 - ☒ `CREATE TABLE Registreeringud (tudeng_matriklinr TEXT, aine_kood TEXT);`
2. Mis käskudega saab lisada, pärida ja muuta ridu SQL-ga?
 - ☐ ADD, GET, UPDATE
 - ☐ INSERT, GET, MODIFY
 - ☒ INSERT INTO, SELECT, UPDATE
 - ☐ PUT, FETCH, MODIFY
 - ☐ INSERT, SELECT, MODIFY
3. Kuidas luua ühendus SQLite andmebaasiga failis `database.db`?
 - ☐ `ühendus = sqlite3.connect("andmebaas.db")`
 - ☐ `ühendus = sqlite3.create_connection("database.db")`
 - ☐ `ühendus = sqlite3("database.db")`
 - ☒ `ühendus = sqlite3.connect("database.db")`
 - ☐ `ühendus = sqlite.connect("database.db")`

Ülesanded

1. Kirjuta programm `loo_andmebaas.py`, mis loob tabeli, millel on 4 välja:
 1. id: arv, primaarne võti
 2. kasutajanimi: tekst
 3. parool: tekst
 4. lemmikarv: täisarv

2. Kirjuta programm `registreeri.py`, millega luuakse eelmises ülesandes loodud tabelisse uus kasutaja.

```
Sisesta kasutaja: admin  
Sisesta parool: parimparoolmaailmas  
Sisesta lemmik arv: 1337  
Kasutaja loodud!
```

```
Sisesta kasutaja: admin  
Kasutajanimi on juba võetud!
```

3. Kirjuta programm `logisisse.py`, milles saab sisse logida eelmises ülesandes loodud kasutajatesse.

```
Sisesta kasutaja: admin2  
Kasutajat ei leitud!  
Sisesta kasutaja: admin  
Sisesta parool: kõigeparemparoolmaailmas  
Vale parool!
```

```
Sisesta kasutaja: admin  
Sisesta parool: parimparoolmaailmas  
Edukalt sisse logitud! Sinu lemmik arv on 1337.
```

5. Veebirakenduste loomine

Kindlasti kasutad igapäevaselt veebilehti nagu Google, YouTube, Facebook, Instagram, Twitter, Vikipeedia või vähemalt oled neist kuulnud. Need on kõik tegelikult veebirakendused, mille taga jookseb mingi kood, mis saadab veebilehitsejate päringutele andmeid vastu. Uurime, kuidas luua enda lihtne veebirakendus ja kuidas seda Internetti püsti panna.

Veebirakendusi kirjutatakse keeltega PHP, Java, Ruby ja muidugi ka Python. Veebirakenduste jaoks on Pythonil palju erinevaid [raamistikke](#): Django, TurboGears, Pyramid, Flask ja väga palju muid. Selles peatükis keskendume [Flaskile](#), mis lubab veebirakenduse püsti saada vaid mõne reaga.

Ettevalmistus

Enne jätkamist tuleb paigaldada moodul [flask](#). Seda saab teha käsuga `pip install flask`. Katsetamiseks kirjuta Pythonis `import flask`. Kui erindit ei visata, on moodul paigaldatud. Kui ei ole kindel, kuidas mooduleid installida, siis on õpikus [moodulite paigaldamise juhised](#).

Et peatükist aru saada, peab olema läbitud õpiku [esimene osa](#). Kasuks tuleb HTML-i tundmine, aga selle peatüki läbimiseks ei pea seda oskama. HTML-i õppematerjalid: [eesti keeles](#), [inglise keeles](#).

Flask

Flask on Pythoni moodul lihtsate veebirakenduste loomiseks, mis keskendub kiiresti alustamisele ja lihtsusele. Selle lihtsus ei takista ka suuremate veebirakenduste loomist. Seda kasutavad teiste raamistike kõrval näiteks Netflix, Airbnb, Reddit, Uber [ja muud](#). Selles peatükis kirjutame lihtsa veebirakenduse, mis võtab kasutajalt sisendit ja arvutab selle põhjal tulemuse. Vaatame ka, kuidas seda Internetti üles laadida ja teistele jagada.

Tere, maailm!

Paneme kõigepealt püsti kõige lihtsama võimaliku veebirakenduse: üksainus lehekülg, mis tagastab mingit teksti. Salvesta järgnev kood faili `teremaailm.py`.

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def index():
    return "Tere, maailm!"

if __name__ == "__main__":
    app.run(debug=True)
```

Proovi seda programmi jooksutada. Peaks ilmuma selline väljund:

```
* Serving Flask app "teremaaailm" (lazy loading)
* Environment: production
  WARNING: Do not use the development server in a production
environment.
  Use a production WSGI server instead.
* Debug mode: on
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 123-456-789
```

Koodi jooksutamine käivitas arvutis veebiserveri aadressil <http://127.0.0.1:5000/>. 127.0.0.1 on IP aadress, mis tähendab praeguse arvuti võrku ja 5000 tähendab porti, mille peal server jookseb praeguse arvuti võrgus. Nüüd proovi minna veebilehitsejas sinna aadressile ja seal peaks olema tekst "Tere, maailm!":



Pärast külastamist ilmub konsooli ka selline väljund:

```
127.0.0.1 - - [06/Apr/2020 14:24:47] "GET / HTTP/1.1" 200 -
```

See tähendab, et lehekülge "/" külastati sellel kellaajal IP-aadressilt 127.0.0.1.

Kui programm ei käivitunud õigesti, siis Flaski dokumentatsioon pakub välja erinevaid lahendusi:

<https://flask.palletsprojects.com/en/1.1.x/quickstart/#what-to-do-if-the-server-does-not-start>

Mis just juhtus?

Käime rakenduse ridahaaval läbi.

```
from flask import Flask
app = Flask(__name__)
```

Impordime Flaski rakenduse klassi ja salvestame sellest isendi muutujasse app. Parameeter `__name__` annab Flaskile teada, kuidas programmi käivitati.

```
@app.route("/")
```

See on dekoraator, mis ütleb rakendusele, et järgmine funktsioon kutsutakse välja, kui minnakse aadressile `/`. Kaldkriips tähendab juurt ehk antud juhul <http://127.0.0.1:5000/>. `/tere` vastaks aadressile <http://127.0.0.1:5000/tere>. Dekoraator teeb põhimõtteliselt seda, et sellele järgnevat funktsiooni kasutatakse ühes teises funktsioonis, aga sellest ei ole Flaski kasutamisel vaja aru saada.

```
def index():
    return "Tere, maailm!"
```

See funktsioon määrab ära, mida veebirakendus teeb, kui minnakse dekoraatoris määratud leheküljele. Funktsiooni tagastus saadetakse lehe külastajale. Praegu tagastatakse lihtsalt tekst `"Tere, maailm!"`, aga tavaliselt laaditakse kuskilt andmeid, tehakse arvutusi ja tagastatakse HTML-kood, mille veebilehitseja teeb ilusaks veebileheks. Siin saab ka vastu võtta kasutaja sisendeid ja nende põhjal muuta tagastust. Funktsiooni nimi ei ole praegu tähtis.

```
if __name__ == "__main__":
    app.run(debug=True)
```

Siin käivitatakse rakendus, kui programm käivitati, mitte ei imporditud.

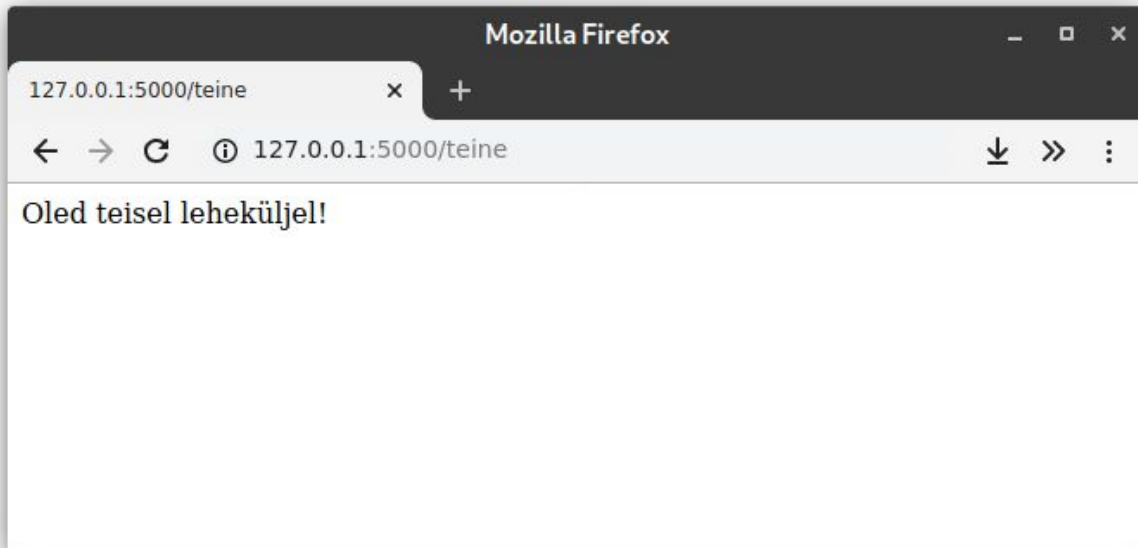
Debug tähendab, et kui lehekülje laadimisel juhtub viga, siis seda kuvatakse veebilehitsejas, mitte ainult konsoolis. See teeb ka seda, et kui programmi koodi muudetakse, siis programmi ei pea uuesti jooksutama, vaid see taaskäivitatakse automaatselt.

Proovi muuta väärtust, mida `index()` funktsioon tagastab ja vaata, mis konsooli ilmub. Ilma programmi taaskäivitamiseta, külasta veebilehitsejaga aadressi uuesti. Kas veebilehel on uus tekst?

Lehekülgede aadressid

Esimeses näites tegime ühe lehekülje aadressile "/". Teeme teise lehekülje teisele aadressile:

```
@app.route("/teine")
def teine():
    return "Oled teisel leheküljel!"
```

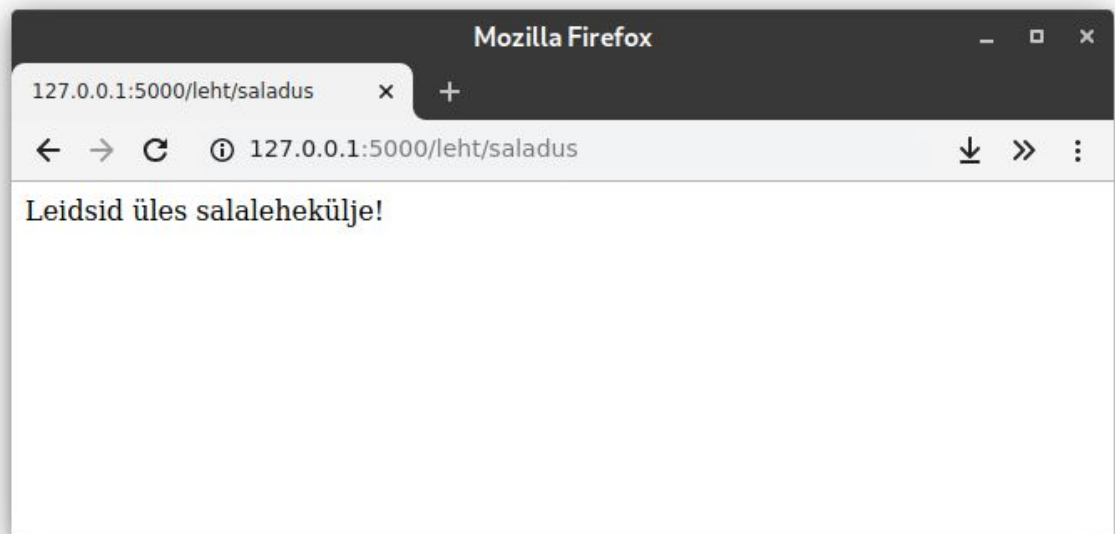


Ühel leheküljel võib olla mitu aadressi:

```
@app.route("/teine")
@app.route("/2")
def teine():
    return "Oled teisel leheküljel!"
```

Aadressid võivad olla isegi dünaamilised, s.t aadressi sees on muutuja. Selleks tuleb aadressis teravnurksulgude vahele panna muutujanimi ning sama muutuja panna funktsiooni parameetritesse. Seda muutujat saab nüüd funktsioonis kasutada.

```
@app.route("/leht/<arv>")
def leht(arv):
    if arv == "saladus":
        return "Leidsid üles salalehekülje!"
    return "Oled praegu leheküljel: " + arv
```



Katseta erinevaid aadresse. Leia vastused küsimustele:

- Mis juhtub, kui minna lehele "/leht/" ilma muutujata?
- Kui defineerida aadress "/leht<number>", kas see töötab ka?
- Kas aadressil võib olla mitu muutujat?

Huvitavam sisu

Lehekülje sisu ei pea samuti staatiline olema. Oletame, et lehekülg random.org on maas ja kirjutame sellest enda versiooni. Algul tagastame lihtsalt juhusliku arvu 1-st 10-ni:

```
from flask import Flask
import random

app = Flask(__name__)
```

```

@app.route("/")
def index():
    juhuslik_arv = random.randint(1, 10)
    return "Sinu juhuslik arv on {}".format(juhuslik_arv)

if __name__ == "__main__":
    app.run(debug=True)

```

Proovi täiendada seda rakendust nii, et kasutaja saab ise määrata juhusliku arvu ülem- ja alampiiri.

Kui külastada lehekülje juurt ehk "/", on piirid 1 ja 10.

Kui külastada aadressi "/<maksimum>", on piirid 1 ja <maksimum>.

Kui külastada aadressi "/<miinimum>/<maksimum>", on piirid <miinimum> ja <maksimum>.

Seda peaks lahendama ainult ühe funktsiooniga.

Vihje: tuleta meelde, kuidas määrata funktsiooni parameetritele vaikeväärtused.

Aadressiparameetrid

Eelmise ülesandega tegime rakenduse, millel on kaks parameetrit: ülem- ja alampiir. On kolm võimalust neid sisestada: mitte ükski; ainult ülempiir; nii alampiir kui ülempiir. Dekoraatorid tulevad sellised:

```

@app.route("/")
@app.route("/<maksimum>")
@app.route("/<miinimum>/<maksimum>")

```

Aga mis siis, kui tahame, et kasutaja sisestab ainult alampiiri ning ülempiir saab vaikeväärtuse? Mis siis, kui meil on palju parameetreid ja neil on kõigil mingid vaikeväärtused? Saaksime teha mingi väärtuse, mille puhul võetakse vaikeväärtus, praegu näiteks "/<miinimum>/default", aga see ei ole väga ilus lahendus. Selle jaoks on olemas võimalus lisada aadressidele parameetrid.

Vaatame YouTube'i video linki: <https://www.youtube.com/watch?v=dQw4w9WgXcQ&t=43>

Selle aadress on tegelikult lihtsalt <https://www.youtube.com/watch>, aga sellel on kaasas kaks parameetrit:

```

{
    "v": "dQw4w9WgXcQ", # mis videot näidatakse
    "t": 43 # mitmendalt sekundilt videot alustada
}

```

Flaskis saab parameetrite väärtustele ligi `requests.args` muutuja kaudu, mille peab flask moodulist importima. Väärtused saab kätte `get()` meetodiga. Neile saab ka määrata vaikeväärtuse ja tüübi. Sama lehekülj oleks Flaskis pool-pseudokoodina umbes selline:


```

from flask import Flask, request, redirect

app = Flask(__name__)

@app.route("/watch")
def watch():
    video_id = request.args.get("v")
    start_time = request.args.get("t", default=0, type=int)
    if not video_id:
        return redirect("/")
    ...

```

Mida YouTube video_id-ga teeb, on ärisaladus. Muutuja start_time tehakse automaatselt täisarvuks ja selle vaikeväärtus on 0.

Proovi lisada oma juhusliku arvu generaatorile võimalus valida alam- ja ülempiirid aadressiparameetritega. Kasuta YouTube'i näidet, et parameetrite väärtused kätte saada.

Kui külastada lehekülje juurt ilma parameetriteta ehk aadressi "/", on piirid 1 ja 10.

Kui külastada aadressi "/?maksimum=20", on piirid 1 ja 20.

Kui külastada aadressi "/?maksimum=20&miinimum=10", on piirid 10 ja 20.

Ilusamad leheküljed

Siiani oleme tagastanud ainult teksti, aga leheküljed ei ole ju tavaliselt sellised. Lehekülgedel on struktuuri, värve, suuri tekste, väikeseid tekste, pilte, linke. Et anda leheküljele struktuur, tuleb kasutada HTML-koodi. Ilusaks saab seda teha CSS-ga. Selle peatüki eesmärk ei ole neist kumbagi õpetada. Kasutame olemasolevat disaini ja vaatame, kuidas Flaskiga seda tagastada.

Flask nõuab, et lehekülje failid on spetsiifilistes kaustades. HTML-failid lähevad kausta templates/ ning CSS-failid, pildid ja muud staatilised failid lähevad kausta static/. Failipuu peaks tulema järgmine, kui app.py on Pythoni fail, kus käivitatakse Flaski rakendus:

```

templates/
├── index.html
└── about.html
static/
├── style.css
├── pilt.jpg
└── script.js
app.py

```

Salvesta järgmine HTML-kood faili index.html ja paiguta see kausta templates/.

```

<!DOCTYPE html>
<html>
    <head>
        <title>Juhusliku arvu generaator</title>
        <meta charset="UTF-8">

```

```

    <link rel="stylesheet" type="text/css" href="/static/style.css">
</head>
<body>
    <h1>Juhusliku arvu generaator</h1>
    <h2>Sinu juhuslik arv on: 3</h2>
    <form action=".">
        <label for="miinimum">Alampiir:</label>
        <input id="miinimum" name="miinimum" type="number" value="1"><br>
        <label for="maksimum">Ülempiir:</label>
        <input id="maksimum" name="maksimum" type="number" value="10"><br>
        <input type="submit" value="Genereeri">
    </form>
</body>
</html>

```

Salvesta järgmine CSS-kood faili `style.css` ja paiguta see kausta `static/`.

```

body {
    font-family: sans-serif;
    text-align: center;
}
input {
    margin: 0.5rem 0;
}

```

Nüüd on vaja Flaskiga HTML tagastada. Seda saab teha funktsiooniga `render_template()`. Sulgudesse läheb HTML-faili nimi.

```

from flask import Flask, render_template
app = Flask(__name__)

@app.route("/")
def index():
    return render_template("index.html")

if __name__ == "__main__":
    app.run(debug=True)

```

Kui HTML- ja CSS-fail on õigesti paigutatud, tuleb lehekülg selline:



Lehekülg näeb nüüd välja nagu lehekülg. Muidugi, genereeritav arv on alati 3 ja nupule vajutamine seda ei muuda. Järgmisena tuleb see tööle saada.

Kui vajutada nupu "Genereeri" peale, siis teeb veebilehitseja leheküljele värskenduse ja aadressi lõppu lisatakse juba parameetrid sisendikastidelt:

```
127.0.0.1:5000/?minimum=1&maksimum=10
```

Need tulevad tänu sellele, et `<form>` elemendil on atribuudid `action="."` ja `method="GET"`. Need määravad ära, et nupu (`<input type="submit">`) vajutamisel minnakse aadressile `"."` (mis on praegune lehekülg) ja lisatakse parameetrid vastavalt sisendkastidele ja nende `"name"` atribuutidele.

Aadressiparameetreid oskame juba lugeda. Nüüd on jäänud ainult küsimus: kuidas me sellisel leheküljel andmeid kuvame?

Leheküljel andmete kuvamine

Kui lehekülg oli ainult üks sõne, siis oli sinna andmeid lihtne sisestada:

```
return "Sinu juhuslik arv on {}".format(juhuslik_arv)
```

Kui kasutame HTML-koodi ja `render_template()` funktsiooni, siis on lahendus natuke keerulisem, aga siiski sarnane. HTML-koodis tuleb lisada muutujad kahe loogelise sulu vahele:

```
<h2>Sinu juhuslik arv on: {{arv}}</h2>
```

Need väärtused tuleb lisada `render_template()` funktsiooni parameetritena:

```
jhuslik_arv = random.randint(1, 10)
return render_template("index.html", arv=jhuslik_arv)
```

See töötab, sest Flask kasutab [Jinja](#) malle. Jinjaga saab teha palju enam, kui ainult andmeid kuvada. Näiteks saab selle abil lisada HTML-koodi [if-lauseid](#) (kas muutuja saadeti üldse kaasa? kas on vaja kuvada veateade? kas kasutaja on sisselogitud?) või [for-tsükleid](#) (kuva kõik järjesti elemendid). Siin me neid täpsemalt läbi ei katseta.

Proovi panna kokku kõik siiaani tehtud kood ja saada juhusliku arvu generaator lõpuks valmis.



Rakenduse avalikustamine

Kui Flaski server jookseb, saab samas arvutis sellele ligi aadressil 127.0.0.1:5000, aga teised sellele ligi ei saa. Idee poolest saab ligi samas võrgus teisest arvutist või isegi teisest võrgust, kui ruuter seda lubab, aga eesmärk on saada lihtne ligipääs kõigile. Selle jaoks tuleb panna rakendus püsti kuskile serverisse. Seda lubavad lihtsasti teenused nagu [PythonAnywhere](#) ja [Heroku](#). Saab ka kasutada virtuaalservereid nagu [DigitalOcean](#) ([õpetus](#)), mis on natuke keerulisem.

PythonAnywhere lubab Flaski veebirakendust tasuta üles laadida mõningate piirangutega. Veebirakendus ilmub aadressile <https://kasutajanimi.eu.pythonanywhere.com/>.

Juhised:

1. Tee uus *Beginner* kasutaja ja logi sisse:
<https://www.pythonanywhere.com/registration/register/beginner/>
2. Kodulehel vali "Open web tab" ja "Add a new web app".
3. Läbi protsess vajutades "Next" nupule. Valida "Flask" raamistik ja kõige uuem Pythoni versioon.
4. Rakenduse faili asukoht võib jääda samaks.
5. <https://kasutajanimi.eu.pythonanywhere.com/> viib nüüd rakenduseni, mis ütleb "Hello from Flask!".
6. Liigu kodulehele ja vajuta "Browse files" nupule. Vasakul liigu kausta `mysite`. Seal on `flask_app.py`, mille sisu tuleb muuta. Kopeeri sinna selle peatüki raames tehtud kood.
7. Loo uued kaustad `templates` ja `static` ja laadi üles failid nendesse kaustadesse.
8. Rakenduse leheküljel vajuta "Reload" nupule.
9. Rakendus jookseb nüüd aadressil <https://kasutajanimi.eu.pythonanywhere.com/>.

Kui rakendus avalikuks teha, peaks koodist ära võtma `debug=True`.

Kokkuvõte

Ehitasime väga lihtsa veebirakenduse ja panime selle Internetti üles. Veebirakendused võivad muidugi olla palju keerulisemad ja järgmise YouTube'i, Instagrami või ÕIS-i versiooni loomiseks on veel palju õppida. [Developer Roadmap](#) teeb hea ülevaate sellest, mida kõike peab korralik veebiarendaja teadma.

Flaskiga jätkamiseks tasub järgida selle [dokumentatsiooni õpetust](#), kus kirjutatakse võimas blogirakendus. Kui Flask ei meeldi, siis [Django](#) raamistik teeb rohkem asju sinu eest ära, aga sellega alustamine võib olla keerulisem.

Informaatika õppekavas minnakse süvitsi veebirakendustesse aines "Veebirakenduste loomine" ([LTAT.05.004](#)).

Enesekontrolliküsimused

1. Kuidas tagastada Flaskiga HTML-faili `about.html` sisu nii, et Jinja muutuja `{{nimed}}` saab väärtuse Pythoni muutujast `andmed`?
 - `return render_template("about.html", nimed=andmed)`
 - `return render_html("about.html", andmed=nimed)`
 - `return render_template("index.html", andmed=nimed)`
 - `return render_html("index.html", nimed=nimed)`
 - `return render_html("about.html", andmed=nimed)`

2. Leheküljele tehakse päring /submit?kogus=42. Kuidas saame Flaski funktsioonis kätte parameetri kogus väärtuse? Vaikeväärtus olgu 1 ja muutujatüübiks olgu täisarv.
 - request.get_arg("kogus", default=1, type=int)
 - flask.get_args["kogus"].type("int").default(1)
 - request.args.get("kogus", default=1, type=int)
 - flask.params.get("kogus", default=1, type="int")
 - request.args.param("kogus", default=1, type="int")
3. Tahame kuvada aadressil /galerii/suvi2020/14 pilti 14 galeriist "suvi2020". Milline dekoraatori ja funktsiooni kasutus on õige?
 - @route("galerii/<nimi>/<pilt>") ja def galerii(nimi, pilt)
 - @flask.route("/galerii/<nimi>/<pilt>") ja def galerii()
 - @app.path("/galerii/<nimi>/<pilt>") ja def galerii()
 - @flask.path("/galerii/<nimi>/<pilt>") ja def galerii(nimi, pilt)
 - @app.route("/galerii/<nimi>/<pilt>") ja def galerii_view(nimi, pilt)

Ülesanded

1. Juhusliku arvu generaatoril lähevad alam- ja ülempiirid nupu vajutamisel tagasi vaikeväärtustele. Täienda rakendust nii, et need säilitavad oma väärtused vastavalt parameetritele.
2. Täienda juhusliku arvu generaatorit nii, et genereerida saab mitu juhuslikku arvu. Selleks tuleb lisada uus aadressiparameeter, mis määrab ära, mitu arvu peab genereerima. Tuleb ka lisada HTML-koodi sisendikast, mis lubab seda parameetrit muuta. Näide:

```
<label for="kogus">Kogus:</label>
<input id="kogus" name="kogus" type="number" value="1"><br>
```



3. Kirjuta uus kasulik veebirakendus, mis kasutab kasutajasisendit. Mõned ideed:

- Kuupäeva kalkulaator (peatükk "Standardteek ja moodulid")
- Mingist rakendusliidesest saadud andmete kuvamine
 - Pokémoni andmed ja pilt - <https://pokeapi.co/>
 - Koerte pildid - <https://dog.ceo/dog-api/documentation/>
 - Kasside faktid - <https://cat-fact.herokuapp.com/facts/random>
- Vikipeedia artiklist viidete eemaldaja (peatükk "Regulaaravaldised")
 - Siin tuleks kasutada `<textarea>` HTML elementi
- Anonüümne foorum
 - Sisestatud tekst lisatakse tekstifaili ja faili sisu kuvatakse veebilehel
 - Veel parem, kui kasutada andmebaasi
 - Siin tuleks kasutada [POST-päringut](#)

Soovitav on taaskasutada juhusliku arvu generaatori HTML- ja CSS-koodi.

6. Objektorienteeritud programmeerimine

Siiamaani oleme salvestanud andmeid erinevatesse andmestruktuuridesse: järjendid, ennikud, hulgad, sõnastikud. Vaatamata on veel jäänud vaieldamata kõige võimsam andmestruktuur üldse. See on andmestruktuur, millega luuakse teisi andmestruktuure. See on andmestruktuur, mida ei saa võibolla isegi kutsuda andmestruktuuriks, sest see on omaette klass.

Jutt käib justnimelt klassidest ja nendega loodavatest objektidest. Tegu on objektorienteeritud programmeerimise peatükiga. Vaatame, kuidas oleme juba varem klassidega kokku puutunud, kuidas luua enda klasse, mis nendega teha saab ning kus neid hästi rakendada saab. Proovime luua enda versiooni mõnest populaarsest rakendusest.

Ettevalmistus

Selle peatüki läbimiseks piisab Pythoni installatsioonist: midagi paigaldama ei pea.

Et peatükist aru saada, peab olema läbitud õpiku [esimene osa](#).

Tuttavate objektide uurimine

Kuigi siin kursusel pole veel klasside telgitagustesse vaadatud, oleme mõningaid klasse kasutanud. Tuletame meelde `datetime` moodulit. See on moodul, mis lubab salvestada ajahetki, sh kuupäevi ja kellaaegu.

Mooduli importimine:

```
>>> import datetime
```

Moodulis `datetime` asub klass nimega `datetime`. Klassidega saab teha isendeid ehk objekte. Need on muutujad, mille küljes võib olla teisi muutujaid ehk atribuute. Need muutujad võivad olla ka funktsioonid, mida kutsutakse meetoditeks. Et luua uut objekti, peab klassi välja kutsuma kui funktsiooni, mis tagastab selle klassi isendi. Proovime luua `datetime` isendi. Funktsiooni parameetriteks lähevad aasta-aeg, kuu, päev, tund, minut, sekund ja mikrosekund.

```
>>> kuupäev = datetime.datetime(2020, 2, 29, 12, 34, 56, 789)
```

Muutujasse `kuupäev` on nüüd salvestatud klassi `datetime` isend ehk objekt. Uurime, mis on selle muutujatüüp (`type()`) ning mis muutujad sellega seostuvad (`dir()`).

```
>>> type(kuupäev)
<class 'datetime.datetime'>
>>> dir(kuupäev)
['__add__', '__class__', '__delattr__', '__dir__', '__doc__', '__eq__',
```



```
['__format__', '__ge__', '__getattribute__', '__gt__', '__hash__',
 '__init__', '__init_subclass__', '__le__', '__lt__', '__ne__',
 '__new__', '__radd__', '__reduce__', '__reduce_ex__', '__repr__',
 '__rsub__', '__setattr__', '__sizeof__', '__str__', '__sub__',
 '__subclasshook__', 'astimezone', 'combine', 'ctime', 'date', 'day',
 'dst', 'fold', 'fromisocalendar', 'fromisoformat', 'fromordinal',
 'fromtimestamp', 'hour', 'isocalendar', 'isoformat', 'isoweekday',
 'max', 'microsecond', 'min', 'minute', 'month', 'now', 'replace',
 'resolution', 'second', 'strftime', 'strptime', 'time', 'timestamp',
 'timetuple', 'timetz', 'today', 'toordinal', 'tzinfo', 'tzname',
 'utcfromtimestamp', 'utcnow', 'utcoffset', 'utctimetuple', 'weekday',
 'year']
```

Näeme, et tegemist on tõepoolest isendiga klassist `datetime.datetime`, millel on palju seotud muutujaid ehk välju. Näiteks on seal väljad `year`, `month`, `day`, `hour`, `minute`, `second` ja `microsecond`, millega saame kätte arvud, millega me isendi lõime. Neid välju nimetatakse **isendiväljadeks**.

```
>>> kuupäev.year
2020
>>> kuupäev.month
2
>>> kuupäev.microsecond
789
```

Klassidel saab tihti isendiväljade väärtusi ise määrata (nt `kuupäev.year = 2021`), aga `datetime` seda ei luba.

Loodud isendil on ka funktsioone, mis kasutavad mõnda isendivälja. Neid nimetatakse **isendimeetoditeks**. Näiteks on üks selline `strftime()`, mis tagastab kuupäeva sõnena soovitud formaadis.

```
>>> kuupäev.strftime
<built-in method strftime of datetime.datetime object at 0x7f279e256570>
>>> kuupäev.strftime("%Y-%m-%d %H:%M:%S")
'2020-02-29 12:34:56'
```

Klassidel on tihti isendimeetodeid, mis muudavad isendivälju, aga `datetime` klassil selliseid pole.

Klassidel on ka funktsioone, mis ei vaja isendeid ega isendivälju. Nendega saab üldiselt teha midagi selle klassiga seotut, näiteks sama klassi isendi tagastamist. Selliseid funktsioone kutsutakse **staatilisteks meetoditeks**. Sisuliselt on need lihtsalt funktsioonid, mis on klassi küljes. Klassil `datetime` on näiteks staatiline meetod `now()`, mis tagastab praeguse ajahetke jaoks `datetime` isendi.

```
>>> datetime.datetime.now()
datetime.datetime(2020, 2, 29, 18, 20, 33, 30651)
```

Selliseid meetodeid saab ka välja kutsuda isendite kaudu, aga üldiselt nii ei tehta.

```
>>> kuupäev.now()
datetime.datetime(2020, 2, 29, 18, 20, 33, 30652)
>>> datetime.datetime.now().now().now().now()
datetime.datetime(2020, 2, 29, 18, 20, 33, 30653)
```

Klassidel võib ka olla staatilisi välju, mis on kõikidel isenditel samad. Klassil `datetime` on näiteks staatilised väljad `min` ja `max`, mis näitavad minimaalseid ja maksimaalseid kuupäeva väärtuseid.

```
>>> datetime.datetime.min
datetime.datetime(1, 1, 1, 0, 0)
>>> datetime.datetime.max
datetime.datetime(9999, 12, 31, 23, 59, 59, 999999)
```

Väljade ja meetodite selgituste saamiseks saab kasutada [dokumentatsiooni](#) ja `help()` funktsiooni.

```
>>> help(datetime.datetime)
Help on class datetime in module datetime:

class datetime(date)
|   datetime(year, month, day[, hour[, minute[, second[, microsecond[, tzinfo]]]])
|
|   The year, month and day arguments are required. tzinfo may be None, or an
|   instance of a tzinfo subclass. The remaining arguments may be ints.
|
|   Method resolution order:
|       datetime
|       date
|       builtins.object
|
|   Methods defined here:
|
|   __add__(self, value, /)
|       Return self+value.
|
|   ...
```

Järgmisena vaatame, kuidas klasse luua ning kuidas need töötavad.

Klasside loomine

Loome enda kuupäeva klassi. Et vähendada tööd, mõtleme ainult kuupäevadele ilma kellaaegadeta ehk meie kuupäeva klassil peaks olema 3 isendivälja: aasta, kuu ja päev. Meie ideaalses maailmas on kalendris igal kuul täpselt 31 päeva, muidu peaks kontrollima kuu päevade arvu vastavalt kuule.

Klassidele on tavaks panna suure algustähga nimi, seega paneme selle nimeks Kuupäev. Klasse saab luua class käsuga. Selle käsu alla saab defineerida meetodeid ja välju. Lisame alguses ainult aasta välja ning määrame selle väärtuseks 2020.

```
class Kuupäev:  
    aasta = 2020
```

Nüüd saame luua selle klassi isendi. Selleks kutsume seda välja kui funktsiooni. Täpselt nagu tavaliste muutujatega, saame isendivälju kasutada ning üle kirjutada. Saame ka uusi välju omistada ning neid kasutada.

```
>>> kuupäev = Kuupäev()  
>>> kuupäev  
<__main__.Kuupäev object at 0x7fa1cf208e80>  
>>> kuupäev.aasta  
2020  
>>> kuupäev.aasta = 2021  
>>> kuupäev.aasta  
2021  
>>> kuupäev.kuu = 2  
>>> kuupäev.kuu  
2
```

Konstruktorid

Praegu on igal uuel loodud isendil aasta väärtus automaatselt 2020. See ei ole väga ettenägelik lähenemine, sest sellest järgnevatel aastatel ei ole see enam kasulik ja iga uue isendi loomisel peab selle üle kirjutama. Samuti on tüütu isendi loomisel manuaalselt teisi muutujaid määrata.

Kui me lõime uue datetime isendi, kutsusime me klassi välja nagu funktsiooni ja lisasime muutujate väärtused parameetritena.

```
>>> kuupäev = datetime.datetime(2020, 2, 29, 12, 34, 56, 789)
```

Et sellist võimalust enda klassile lisada, peame defineerima erilise funktsiooni, mida kutsutakse **konstruktoriks**. Selle funktsiooni nimi Pythonis on `__init__` ning selle esimene parameeter peab olema `self`, mille väärtuseks läheb alati käesolev isend. Järgnevad parameetrid võivad olla ise määratud. Need on need parameetrid, mis lähevad funktsiooni sulgudesse selle väljakutsumisel.

```
class Kuupäev:  
    def __init__(self, aasta, kuu, päev):  
        self.aasta = aasta  
        self.kuu = kuu  
        self.päev = päev
```

Defineerisime klassile Kuupäev konstruktori kolme parameetriga: aasta, kuu ja päev. Kuna self tähendab käesolevat isendit, siis määrame uuele isendile uued isendiväljad, mille väärtused võtame funktsiooni parameetritelt samamoodi, nagu me enne omistasime isendile välju.

Nüüd saame luua uue isendi ise valitud algväärtustega.

```
>>> kuupäev = Kuupäev()
TypeError: __init__() missing 3 required positional arguments: 'aasta',
'kuu', and 'päev'
>>> kuupäev = Kuupäev(2020, 2, 29)
>>> kuupäev.aasta
2020
>>> kuupäev.kuu
2
>>> kuupäev.päev
29
```

Konstruktoris saame ka kontrollida väärtuseid. Kui kuu või päeva väärtus on sobivatest väärtustest väljaspool, viskame erindi. Meie klassil on kõikidel kuudel on teadagi 31 päeva.

```
class Kuupäev:
    def __init__(self, aasta, kuu, päev):
        self.aasta = aasta

        if 1 <= kuu <= 12:
            self.kuu = kuu
        else:
            raise ValueError("Kuu peab olema 1 kuni 12.")

        if 1 <= päev <= 31:
            self.päev = päev
        else:
            raise ValueError("Päev peab olema 1 kuni 31.")
```

```
>>> kuupäev = Kuupäev(2020, 13, 1)
ValueError: Kuu peab olema 1 kuni 12.
>>> kuupäev = Kuupäev(2020, 2, -3)
ValueError: Päev peab olema 1 kuni 31.
>>> kuupäev = Kuupäev(2020, 2, 29)
>>>
```

Isendimeetodid

Kui me tahame praegu kuupäeva isendit teisendada sõneks, siis peame kokku liitma 3 isendivälja.

```
>>> kuupäev = Kuupäev(2020, 2, 29)
>>> "{}-{:02d}-{:02d}".format(kuupäev.aasta, kuupäev.kuu, kuupäev.päev)
'2020-02-29'
```

Võime lisada klassile isendimeetodi, mis seda ise teeb ning tagastab kuupäeva sõnevormingus. Asendame lihtsalt isendi viite muutujaga self.

```
def sõnena(self):
    return "{}-{:02d}-{:02d}".format(self.aasta, self.kuu, self.päev)
```

```
>>> kuupäev = Kuupäev(2020, 2, 29)
>>> kuupäev.sõnena()
'2020-02-29'
```

Tegelikult peaks igal klassil olema selline meetod, mis väljendab isendit sõnena. Kui proovime isendit praegu sõneks teisendada, siis antakse meile klassi nimi ja mäluviide, aga isendi sisu ei näidata.

```
>>> str(kuupäev)
'<__main__.Kuupäev object at 0x7feca1b2cc70>'
>>> kuupäev
<__main__.Kuupäev object at 0x7feca1b2cc70>
```

Kui aga datetime klassi isendit sõneks teisendada, siis tuleb ülevaade isendiväljadest.

```
>>> kuupäev = datetime.datetime(2020, 2, 29, 12, 34, 56, 789)
>>> str(kuupäev)
'2020-02-29 12:34:56.000789'
>>> kuupäev
datetime.datetime(2020, 2, 29, 12, 34, 56, 789)
```

See, mida sõneks teisendamisel tehakse, oleneb jälle ühest erilisest meetodist nimega `__str__`. Kui sõneks teisendatakse, võetakse selle meetodi tagastus. Kirjutame oma kuupäeva klassile selle meetodi. Võime kasutada sõnena meetodit selle tagastuses.

```
def __str__(self):
    return self.sõnena()
```

```
>>> kuupäev = Kuupäev(2020, 2, 29)
>>> str(kuupäev)
'2020-02-29'
>>> kuupäev.__str__()
'2020-02-29'
>>> kuupäev
<__main__.Kuupäev object at 0x7f7f85ea5c70>
```

Sõneks teisendamine töötab hästi, aga niisama isendi kirjutamine tagastab jälle klassi nime ja mäluviite. See on sellepärast, et see väärtus võetakse teisest meetodist nimega `__repr__` (tuleb sõnast *representation*). Saame enda klassile selle meetodi kirjutada. Tavaks on sellel meetodil tagastada sõne, mille abil saab konstruktorit välja kutsuda, et saada selline isend.

```
def __repr__(self):  
    return "Kuupäev({}, {}, {})".format(self.aasta, self.kuu, self.päev)
```

```
>>> kuupäev = Kuupäev(2020, 2, 29)  
>>> kuupäev  
Kuupäev(2020, 2, 29)  
>>> repr(kuupäev)  
'Kuupäev(2020, 2, 29)'
```

Lisame veel isendimeetodi, mis muudab isendiväljade väärtuseid. Näiteks võiks olla meetod aastate juurde lisamiseks.

```
def lisa_aastaid(self, aastate_arv):  
    self.aasta += aastate_arv
```

```
>>> kuupäev = Kuupäev(2020, 2, 29)  
>>> kuupäev.lisa_aastaid(5)  
>>> kuupäev  
Kuupäev(2025, 2, 29)
```

Kuude lisamine on keerulisem, sest kui kuu väärtus on üle 12, siis peab vastavalt aastaid juurde lisama.

```
def lisa_kuid(self, kuude_arv):  
    self.kuu += kuude_arv  
    if self.kuu > 12:  
        # Lisame aastaid juurde nii palju, kui neid 12-st üle läheb  
        self.lisa_aastaid((self.kuu - 1) // 12)  
        # Uus kuude arv on praeguse arvu jääk jagamisel 12-ga  
        self.kuu = (self.kuu - 1) % 12 + 1
```

```
>>> kuupäev = Kuupäev(2020, 2, 29)  
>>> kuupäev.lisa_kuid(13)  
>>> kuupäev  
Kuupäev(2021, 3, 29)  
>>> kuupäev.lisa_kuid(120)  
>>> kuupäev  
Kuupäev(2031, 3, 29)
```

Proovi ise lisada päevade lisamise meetod. Kui kasutada `lisa_kuid` meetodit, siis ei pea aastaid eraldi juurde lisama. Lihtsuse mõttes võib ikka eeldada, et igal kuul on 31 päeva.

Kui kõik kolm meetodit on tehtud, saab ka teha ühise liitmismeetodi.

```
def lisa(self, aastate_arv=0, kuude_arv=0, päevade_arv=0):
    self.lisa_aastaid(aastate_arv)
    self.lisa_kuid(kuude_arv)
    self.lisa_päevi(päevade_arv)
```

```
>>> kuupäev = Kuupäev(2020, 2, 29)
>>> kuupäev.lisa(aastate_arv=1, kuude_arv=2, päevade_arv=3)
>>> kuupäev
Kuupäev(2021, 5, 1)
```

Staatilised meetodid

Lisame oma klassile ühe staatilise meetodi. Staatiliste meetodite puhul peab lihtsalt parameetri `self` ära jätma, sest ei kasutata ühtegi isendivälja ega -meetodit. Kirjutame meetodi, mis tagastab isendi praeguse kuupäeva väärtustega. Võtame väärtused `datetime` isendist.

```
def praegu():
    nüüd = datetime.datetime.now()
    return Kuupäev(nüüd.year, nüüd.month, nüüd.day)
```

```
>>> Kuupäev.praegu()
Kuupäev(2020, 2, 29)
```

Proovi kirjutada staatiline meetod `parsi`, mis võtab parameetrina sõne formaadis "YYYY-MM-DD", nt "2020-02-29" ning tagastab uue kuupäeva isendi nende väärtustega.

```
>>> Kuupäev.parsi("2020-02-29")
Kuupäev(2020, 2, 29)
```

Kokkuvõte

Vaatasime natuke Pythoni objektorienteeritud programmeerimise võimalusi: mis asjad on klassid ja isendid, kuidas neid luua ja kasutada ning mis on, isendiväljad ja meetodid. Mõned võimalused jäid vaatamata, näiteks privaatsed väljad, klasside pärilus, itereeritavad isendid, objektide käitumine liitmisel, lahutamisel, võrdlemisel jne. Nendega saab tutvuda [Pythoni dokumentatsioonis](#).

Programmeerimise paradigmat, mis kasutab objekte, nimetatakse objektorienteeritud programmeerimiseks. See on väga levinud viis, kuidas programme kirjutada ning sellest räägitakse rohkem aines "Objektorienteeritud programmeerimine" ([LTAT.03.003](#)), kus

kasutatakse Java keelt. Kui on plaanis edaspidi Pythoniga tegeleda, siis on Pythoni objektorienteerituse tundmine väga kasulik.

Enesekontrolliküsimused

1. Mis eristab staatilist meetodit isendimeetodist?
 - Staatilistel meetoditel ei ole parameetreid.
 - Staatilised meetodid ei sõltu isendiväljadest.
 - Isendimeetodid on defineeritud klassi sees, staatilised meetodid on defineeritud klassist väljas.
 - Isendimeetodeid saab kasutada läbi klassi, staatiliste meetodite välja kutsumiseks peab olema isend.
 - Need on tegelikult täpselt samad asjad.
2. Miks töötab koodijupp `datetime.datetime.now().now().now().now()`?
 - Sest meetod `now()` töötab iga Pythoni objekti peal.
 - Sest `now()` on meetod, mis tagastab uue meetodi.
 - Sest `now()` on staatiline meetod, mis tagastab uue `datetime.datetime` objekti, millel on ka `now()` meetod.
 - Sest `datetime.datetime` on staatiline klass.
 - Kõik vastusevariandid on õiged.
3. Mis on konstruktor?
 - Meetod, millega väärtustatakse isendivälju
 - Meetod, mis kutsutakse välja enne teiste meetodite välja kutsumist
 - Meetod, mis kutsutakse välja isendi loomisel
 - Meetod, mis tagastab isendi kõik muutujad
 - Meetod, kus defineeritakse klassi kuju

Ülesanded

1. Lõpeta kuupäeva klass. Peavad olema realiseeritud meetodid `lisa_päevi` ja `parsi`. Meetodid peavad arvestama päevade arvuga kuudes: jaanuaris on 31, veebruaris on 28-29, märtsis on 31, aprillis on 30 jne. Veebruari päevade arv sõltub aastast: kui aasta jagub arvuga 4, siis on 29, välja arvatud siis, kui see jagub arvuga 100, siis on 28, välja arvatud siis, kui see jagub arvuga 400, siis on jälle 29. Aastal 1900 oli veebruaris 28 päeva, aastal 2000 oli veebruaris 29 päeva.

2. Mõttele välja klass, mis võib sulle kasuks tulla ning kirjuta programm selle klassi defineerimise ja kasutusnäidetega. Klassil peab olema konstruktor, vähemalt üks isendiväli, vähemalt üks isendimeetod ning meetodid `__str__` ja `__repr__`.

Mõned ideed:

- Tabeli klass, kuhu saab lisada ridu ja veerge. Meetodid võiksid olla rea ja veeru kätte saamiseks indeksitega. Tabelit võiks saada teisendada CSV-vormingusse.
- Koordinaadi klass, millel on x- ja y-koordinaatide väärtuste isendiväljad. Meetod tagastab teise koordinaadi objekti kauguse.

- Kujundi klass (näiteks kolmnurk, ristkülik või ring), millel on külgede pikkused ja nurkade suurused. Meetodid pindala ja ümbermõõtude arvutamise jaoks. Võiks ka olla meetod, mis joonistab selle kujundi mooduliga turtle.

7. Graafiliste mängude loomine

Mängude loomine on [väga suur](#) valdkond, kus programmeerijad saavad oma võimeid kasutada. Mänge on muidugi võimalik ka Pythoniga luua. Selle jaoks on loodud erinevaid teekke: [Pygame](#), [pyglet](#), [Cocos2d](#), [Panda3D](#) ja [palju muid](#). Siin peatükis vaatame, kuidas luua üks lihtne mäng Pygame'iga.

Pygame on populaarne Pythoni mängude loomise teek, mis lubab lihtsasti tekitada graafikat ja heli ning saada kasutajalt sisendit mitmel erineval moel. See on hea viis, kuidas alustada oma mänguarendaja karjääri. See pole ka ainult alustamiseks: [päris palju](#) mänge on Pygame'iga tehtud.

Ettevalmistus

Enne jätkamist tuleb paigaldada moodul [pygame](#). Seda saab teha käsuga `pip install pygame`. Katsetamiseks kirjuta Pythonis `import pygame`. Kui erindit ei visata, on moodul paigaldatud. Kui ei ole kindel, kuidas moduleid installida, siis on õpikus [moodulite paigaldamise juhised](#).

Et peatükist aru saada, peab olema läbitud õpiku [esimene osa](#). Soovitatav on läbida peatükk "Objektorienteeritud programmeerimine" läbimine, aga see ei ole vajalik.

Mängu akna kuvamine

Kui moodul `pygame` importida, siis saab Pygame'i käivitada käsuga `pygame.init()`. Enne seda peab veel määrama akna suuruse käsuga `pygame.display.set_mode()`, mille parameetrikaks on ennik ekraani laius ja kõrgus pikslites. Pygame'i akent saab sulgeda käsuga `pygame.quit()`. Teeme akna laius 640 pikslit ja kõrgus 480 pikslit.

```
import pygame

pygame.display.set_mode((640, 480))
pygame.init()

pygame.quit()
```

Kui see programm käivitada, siis ilmub korra must aken suurusega 640 korda 480 pikslit ja läheb kinni.

Pygame'i loodud aken töötab eraldiseisvalt Pythoni programmile. See tähendab, et pärast `pygame.init()` käsku kuvatakse eraldi aken, kuhu saab hakata joonistama ning programmi kood jookseb edasi. Aken läheb kinni alles siis, kui programm jõuab sulgemiskäsu `pygame.quit()`. Kui enne sulgemiskäsku jookseb programm 3 sekundit, siis aken on püsti 3 sekundit.

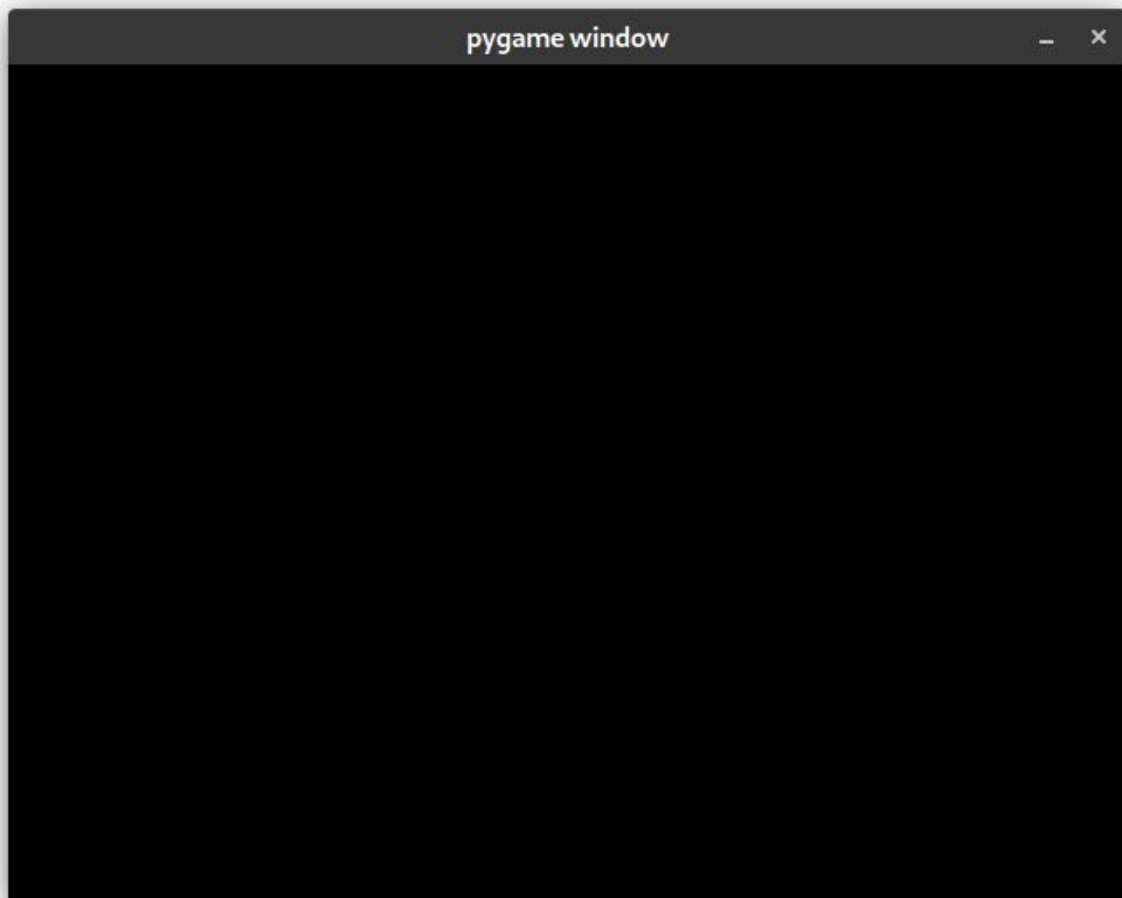
Kui enne sulgemiskäsku jooksutada lõpmatu tsükkel, siis aken ei lähegi kinni ning ristik sulgemine ka ei aita. Programmi töö peab muudmoodi lõpetama. Saame selles lõputus while-tsükli kontrollida, kas kasutaja on vajutanud sulgemisnupule ja sel juhul saame tsükli lõpetada.

```
import pygame

pygame.display.set_mode((640, 480))
pygame.init()

while True:
    sisend = pygame.event.poll()
    if sisend.type == pygame.QUIT:
        break

pygame.quit()
```



Kui see programm käivitada ja ristile vajutada, siis aken läheb kinni. Oleme kirjutanud väga algelise programmi, millega kasutaja saab suhelda! Vaatame hiljem, kuidas tsükklisse muud funktsionaalsust lisada ja teistmoodi kasutajalt sisendit saada. Kõigepealt proovime aknasse midagi joonistada.

Joonistamine

Siiamaani oleme loonud musta akna, mis on graafilisest mängust tsipa kaugel. Järgmisena proovime selle akna millegi värvilisega täita.

Värvid

Saame täita selle akna mingi värviga, mida saab esindada erinevatel viisidel, aga Pygame'is on tavaks kasutada [RGB mudelit](#), mis koosneb punasest, rohelisest ja sinisest värvi kogusest, 0 kuni 255. Neid esindatakse kolmeliikmeliste ennikutena. Näiteks on must värv (0, 0, 0), valge värv (255, 255, 255), punane värv (255, 0, 0), akvamariinsinine (128, 255, 212) jne. Lihtne viis enda värvi valimiseks on [guugeldada "colour picker"](#) ja võtta valitud värvi RGB väärtused.

Täidame akna enda soovitud värviga. Selleks peame ekraanisuuruse muutmise funktsiooni tagastuse salvestama muutujasse ekraan. See on muutuja, kuhu hakkame edaspidi kõiki joonistusi lisama. Ekraani saab täita meetodiga `fill()`, mille parameetriks läheb värv. Tehtud muudatused tuleb salvestada käsuga `pygame.display.flip()`.

```
import pygame

ekraan = pygame.display.set_mode((640, 480))
pygame.init()

ekraan.fill((128, 255, 212))
pygame.display.flip()

while True:
    sisend = pygame.event.poll()
    if sisend.type == pygame.QUIT:
        break

pygame.quit()
```

Programm käivitab nüüd akna, mis pole enam must, vaid akvamariinsinine. Edusammud!



Kujundid

Proovime lõpuks aknasse midagi lisada. Pygame võimaldab joonistada erinevaid kujundeid. Ristkülikut saab ekraanile joonistada funktsiooniga `pygame.draw.rect()`. See nõuab kolme parameetrit:

1. pind, mille peale joonistada (meie puhul muutuja `ekraan`)
2. värv, millega ristkülik täita (värve juba oskame)
3. asukoha nelik: alguspunkti x-koordinaat, alguspunkti y-koordinaat, ristküliku laius, ristküliku kõrgus (kõik pikslites)

Koordinaatide lugemist alustatakse Pygame'is ülevalt vasakult ehk (0, 0) on üleval vasakul, (640, 0) on üleval paremal, (0, 480) on all vasakul ja (640, 480) on all paremal. Kõik vahepealsed on kuskil ekraani sees.

Joonistame roosa ristküliku, mis algab meie akna keskpunktist ehk (320, 240) ning on 200 piksli laiune ja 100 piksli kõrgune.

```
import pygame

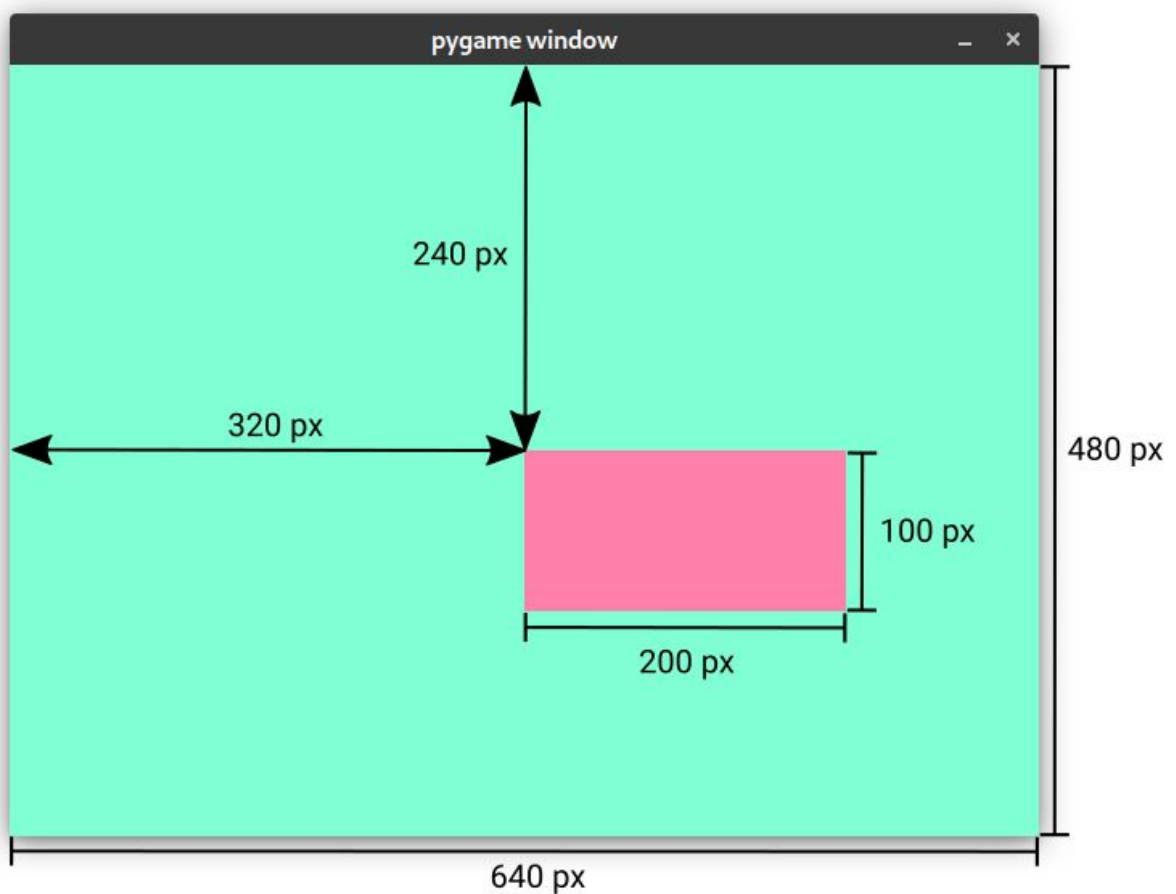
ekraan = pygame.display.set_mode((640, 480))
pygame.init()
```

```
ekraan.fill((128, 255, 212))

pygame.draw.rect(ekraan, (255, 128, 171), (320, 240, 200, 100))
pygame.display.flip()

while True:
    sisend = pygame.event.poll()
    if sisend.type == pygame.QUIT:
        break

pygame.quit()
```



Pygame lubab ka teisi kujundeid joonistada. Loe [dokumentatsioonist](#) erinevaid võimalusi ja proovi ekraanile joonistada ring, kolmnurk ja joon.

Pildid

Sarnaselt kujunditele saab mängu lisada pilte. Kõigepealt peame leidma pildi, mida mängus kasutada. Neid saab muidugi ise luua, aga Internetis on suur hulk vabalt kasutatavaid materjale olemas: itch.io/game-assets, opengameart.org [jpm](#). Kui neid kasutada mängudes,

on vaja autorile viidata. Siin peatükis kasutame rott, mis on võetud [siit](#), ja juustu, mis on võetud [siit](#).



Et joonistada pilt aknasse, tuleb see kõigepealt sisse laadida pildiobjektina funktsiooniga `pygame.image.load()`. Sulgudesse läheb failitee. Praegu paigutame pildifailid programmifailiga samasse kausta, aga suurema mängu puhul võiks need olla eraldi kaustas.

```
rott = pygame.image.load("rott.png")
```

Muutuja `rott` on nüüd pind ([Surface](#)), mida saab joonistada ekraanile. Pindade võimaluste kohta saab lugeda [Pygame'i dokumentatsioonist](#).

Saame roti paigutada ekraanile meetodiga `blit()`. Sulgudesse läheb pildifail ning koordinaadipaariga asukoht. Paigutame loodud roti objekti ekraani koordinaatidele (100, 200).

```
ekraan.blit(rott, (100, 200))
```



Roti laiuse ja kõrguse eraldi saame kätte meetoditega `get_width()` ja `get_height()`. Proovi paigutada rott täpselt ekraani keskele, kasutades seda meetodit, et arvutada paigutuse koordinaadid. (320, 240) ei ole õige siin vastus.

```
>>> rott.get_width()
116
>>> rott.get_height()
72
```

Tekst

Mängudes on tavaliselt teksti, mis näitab vestluseid, õpetusi, skoori, menüüd jpm. Pygame võimaldab teksti kuvada samamoodi nagu piltegi.

Kõigepealt tuleb valida font, milles teksti kuvada ja teha sellest objekt funktsiooniga `pygame.font.SysFont()`. Esimeseks parameetriks läheb fondi nimi sõnena ja teiseks läheb fondi suurus. Kui fondi nimeks valida `None`, siis valib Pygame vaikimisi fondi. Funktsiooniga `Font()` saab ka fondi võtta failist. See on kasulik, kui fonti ei pruugi olla teiste arvutis.

```
font = pygame.font.SysFont("Comic Sans MS", 24)
font = pygame.font.SysFont(None, 32)
font = pygame.font.Font("Futura-Bold-Italic.ttf", 40)
```

Kasutades fonti saame meetodiga `render()` luua pinna objekti. Sulgudesse läheb tekst sõnena, tõeväärtus `True` ning teksti värv. Tõeväärtuse kohta võib lugeda [dokumentatsioonist](#). Loodud objekt on põhimõtteliselt pilt ehk seda saab meetodiga `blit()` paigutada ekraanile. Loome teksti "Tere, maailm!" musta fondiga ning paigutame selle koordinaatidele (40, 40).

```
tekst = font.render("Tere, maailm!", True, (0, 0, 0))
ekraan.blit(tekst, (40, 40))
```




Kogu kood siamaani:

```
import pygame

ekraan = pygame.display.set_mode((640, 480))
pygame.init()

ekraan.fill((128, 255, 212))

rott = pygame.image.load("rott.png")
ekraan.blit(rott, (100, 200))

font = pygame.font.SysFont(None, 40)
tekst = font.render("Tere, maailm!", True, (0, 0, 0))
ekraan.blit(tekst, (40, 40))

pygame.display.flip()

while True:
    sisend = pygame.event.poll()
    if sisend.type == pygame.QUIT:
```

```
break
```

```
pygame.quit()
```

Pinna objekte saab moondada: pööramine, peegeldamine, suurendamine jne. Uuri dokumentatsiooni: [pygame.transform](#).

Harjuta ekraanile joonistamist. Leia vastused küsimustele:

- Kas ühte pinda on võimalik mitu korda ekraanile joonistada?
- Kas pinda saab paigutada ekraanist väljapoole?
- Mis juhtub, kui joonistada pilt enne ja täita ekraan mingi värviga hiljem?
- Mis käskudega saab kätte peegelpildi ja pööramised?
- Kuidas võiks toimuda animeerimine?

Animeerimine

Siiani oleme joonistanud ekraanile ühe kaadri. Et saada paigutatud objekte liikuma, ei saa me lihtsalt muuta nende asukohti: me peame iga kaadri algusest peale joonistama. See võib kõlada ebaintuiitsena, aga Pygame töötabki just niimoodi.

Jätame alles roti ekraanile, aga liigutame tema joonistamise meie põhitsüklisse, mis programmi käimas hoiab.

```
import pygame

ekraan = pygame.display.set_mode((640, 480))
pygame.init()

rott = pygame.image.load("rott.png")

while True:
    sisend = pygame.event.poll()
    if sisend.type == pygame.QUIT:
        break

    ekraan.fill((128, 255, 212))
    ekraan.blit(rott, (100, 200))
    pygame.display.flip()

pygame.quit()
```

Kui programm käivitada, siis midagi väga muutunud ei ole. Rott on ikka paigal, ta lihtsalt joonistatakse igal tsükli käigul uuesti. See on kasulik, sest nüüd me saame näiteks muuta igal käigul roti asukohta.

Salvestame programmi alguses roti koordinaadid mingitesse muutujatesse, näiteks `rott_x` ja `rott_y`, algul olgu need 100 ja 200. Nüüd igal tsükli korral suurendame `rott_x` muutujat ühe võrra ja käivitame programmi uuesti.

```
rott_x = 100
rott_y = 200

...

rott_x += 1
```

Kui see programm käivitada, siis rott tõepoolest liigub väga kiiresti paremale ja jääbki liikuma. Proovi täitmiskäsk ekraanil `fill()` tsüklit ära võtta ja ainult programmi algusesse panna. Mis juhtub?

Saime roti liikuma, aga mitte väga hästi. Probleem on praegu selles, et tsükli käik ei ole kuidagi ajaliselt piiratud: see kestab täpselt nii kaua, kui Pythonil läheb selle jooksutamiseks. See on esiteks väga lühike aeg: peaksime roti koordinaati liigutama palju vähem (nt 0.1 pikslit) iga kord. Teiseks: see võib erineda erinevates arvutites ja isegi samas arvutis erinevatel aegadel. Tahame, et mäng jookseks kõikides arvutites samamoodi igal hetkel.

Selle probleemi parandamiseks kasutame Pygame'i kella. Looime programmi alguses uue kella objekti:

```
kell = pygame.time.Clock()
```

Iga tsükli alguses peaks kell tiksuma ehk kutsume välja meetodi `tick()`. Sulgudesse läheb arv, mitu kaadrit peaks sekundis kuvama. Valime 60, sest enamik kuvaritest värskenduvad 60 korda sekundis. Selle meetodi käivitamisel ootab programm umbes 1/60 sekundit.

```
dt = kell.tick(60)
```

See meetod tagastab, mitu millisekundit on kulunud eelmisest kella tiksumisest. 60 kaadri sekundis puhul see on enamasti 16 või 17. Seda on väga kasulik teada, sest kui arvuti näiteks kiilub korraks kinni ja mõnda kaadrit joonistatakse natuke kauem, siis meile on see teada ja me saame seda oma mängus arvestada nii, et mängu tegevus ei kiiluks kinni, vaid liiguks vastavalt sellele, kui palju aega on kulunud. Kui kaader peaks kestma 16 millisekundit ja kestab hoopis 32, siis mängu tegevus peaks sellel kaadril liikuma 2 korda kiiremini. Selle abil tagame ka selle, et mäng jookseb kõikidel arvutitel sama kiiresti.

Seega peame igal liikumisel arvestama kulunud aega (muutujat `dt`). Roti liikumisel korrutame sellega läbi.

```
rott_x += 0.25 * dt
```

0.25 on siin roti kiirus, mis väljendab mitu pikslit me tahame, et rott liiguks millisekundis. 0.25 pikslit millisekundis on teisendatult 250 pikslit sekundis. Mitme sekundiga jõuab rott ekraanist välja, kui ekraani laius on 640 ja roti algne x-koordinaat on 100?

Kogu kood:

```
import pygame

ekraan = pygame.display.set_mode((640, 480))
pygame.init()

kell = pygame.time.Clock()

roott = pygame.image.load("roott.png")
roott_x = 100
roott_y = 200

while True:
    dt = kell.tick(60)

    sisend = pygame.event.poll()
    if sisend.type == pygame.QUIT:
        break

    ekraan.fill((128, 255, 212))
    ekraan.blit(roott, (roott_x, roott_y))
    pygame.display.flip()

    roott_x += 0.25 * dt

pygame.quit()
```



Proovi muuta kaadrisagedust näiteks 24 peale, mida kasutatakse tavaliselt filmides. Kas rott jõuab erinevate kaadrisagedustega ekraanist välja joosta täpselt samade ajakuludega? Kui jah, siis programm töötab igas arvutis sama kiiresti, lihtsalt aeglase arvuti puhul võivad mõned kaadrid vahele jääda.

Proovime veel animatsiooni huvitavamaks teha. Kui rott jõuab teisele poole, siis paneme ta ümber pöörama. Selleks salvestame roti kiiruse eraldi muutujasse.

```
roti_kiirus = 0.25
...
...
...
    rott_x += roti_kiirus * dt
```

Kui roti parem ots jõuab paremalt poolt ekraanist välja, siis lihtsalt teeme tema kiiruse negatiivseks. Teeme sama, kui roti vasak ots jõuab vasakult poolt ekraanist välja.

```
if (rottp_x + rottp.get_width()) > 640 or rottp_x < 0:
    roti_kiirus *= -1
```

Kui roti kiirus on negatiivne, siis tema x-koordinaat väheneb igas tsükli käigus ja ta liigub vasakule.

Pöörame ka roti ümber. Rohkem infot [dokumentatsioonis](#).

```
rott = pygame.transform.flip(rott, True, False)
```

Kogu kood:

```
import pygame

ekraan = pygame.display.set_mode((640, 480))
pygame.init()

kell = pygame.time.Clock()

rott = pygame.image.load("rott.png")
rott_x = 100
rott_y = 200

roti_kiirus = 0.25

while True:
    dt = kell.tick(60)

    sisend = pygame.event.poll()
    if sisend.type == pygame.QUIT:
        break

    if rott_x > (640 - rott.get_width()) or rott_x < 0:
        roti_kiirus *= -1
        rott = pygame.transform.flip(rott, True, False)

    ekraan.fill((128, 255, 212))
    ekraan.blit(rott, (rott_x, rott_y))
    pygame.display.flip()

    rott_x += roti_kiirus * dt

pygame.quit()
```

Rott nüüd pörkab lõputult akna parema ja vasaku ääre vahel. Proovi panna rott ringi liikuma, nagu vana DVD-mängija ekraanisäästja: kui jõuab ääreni, siis pörkab. Vihje: liikuvale kahemõõtmelisel esemel saab olla 2 kiirust: horisontaalkiirus ja vertikaalkiirus.



Kasutajasisend

Panime roti liikuma, aga tavaliselt saab mängu mõjutada kasutaja. Paneme roti liikuma vastavalt klahvivajutustele!

Siiani oleme tegelikult mingit kasutajasisendit küsinud: kui ristile vajutada, siis mäng läheb kinni. Senist sisendipüüdmist peame natuke muutma. Kahe tsükli käigu vahel võib toimuda mitu sisendit, aga me oleme iga kord ainult ühe küsinud - peaksime kõike kontrollima. Kõik sisendid saab kätte funktsiooniga `pygame.event.get()`. Kuna need peab tsükliga läbi käima, siis `break` viiks meid ainult sellest tsüklist välja. Asendame selle `sys.exit()` väljakutsega, mis lõpetab programmi töö.

```
for sisend in pygame.event.get():
    if sisend.type == pygame.QUIT:
        sys.exit()
```

Samamoodi, nagu kontrollime ristile vajutamist, saame kontrollida ka klaviatuurivajutusi. Selle jaoks tuleb kontrollida, kas sisendi tüüp on `pygame.KEYDOWN`. Kui on, siis saame vaadata selle klahvi väärtust. Nooleklahvid on näiteks `pygame.K_UP`, `pygame.K_DOWN`,

pygame.K_LEFT, pygame.K_RIGHT. Vastavalt nendele võiksime rottii liigutada. Erinevaid nuppe saab kätte [dokumentatsioonist](#).

```
for sisend in pygame.event.get():
    if sisend.type == pygame.QUIT:
        sys.exit()
    elif sisend.type == pygame.KEYDOWN:
        if sisend.key == pygame.K_UP:
            rotti_y -= rotti_kiirus * dt
        elif sisend.key == pygame.K_DOWN:
            rotti_y += rotti_kiirus * dt
        elif sisend.key == pygame.K_LEFT:
            rotti_x -= rotti_kiirus * dt
        elif sisend.key == pygame.K_RIGHT:
            rotti_x += rotti_kiirus * dt
```

Kui nupule vajutada, siis rotti tõepoolest liigub... natuke. Kui tahame, et nuppu all hoides rotti jääkski liikuma, siis võiksime koordinaatide muutmise asemel hoopis rotti kiiruseid muuta ning igal tsükli käigul rottii liigutada vastavalt kiirustele.

```
rotti_kiirus = 0.25
rotti_x_kiirus = 0
rotti_y_kiirus = 0

...
    if sisend.key == pygame.K_UP:
        rotti_x_kiirus = 0
        rotti_y_kiirus = -rotti_kiirus
    elif sisend.key == pygame.K_DOWN:
        rotti_x_kiirus = 0
        rotti_y_kiirus = rotti_kiirus
    elif sisend.key == pygame.K_LEFT:
        rotti_x_kiirus = -rotti_kiirus
        rotti_y_kiirus = 0
    elif sisend.key == pygame.K_RIGHT:
        rotti_x_kiirus = rotti_kiirus
        rotti_y_kiirus = 0
...

rotti_x += rotti_x_kiirus * dt
rotti_y += rotti_y_kiirus * dt
```

Kui nüüd nupule vajutada, siis rotti jääbki liikuma. See võib mõnes mängus olla soovitud, näiteks ussimängus, aga vaatame, kuidas nupust lahti laskmisel rotti seisma jääks. Saame vaadata, et kui sisenditüüp on pygame.KEYUP ehk klahvist lastakse lahti, siis nullime kiirused vastavalt klahvile, mis lahti lasti. Kogu kood siiani:


```

for sisend in pygame.event.get():
    if sisend.type == pygame.QUIT:
        sys.exit()
    elif sisend.type == pygame.KEYDOWN:
        if sisend.key == pygame.K_UP:
            roti_y_kiirus = -roti_kiirus
        elif sisend.key == pygame.K_DOWN:
            roti_y_kiirus = roti_kiirus
        elif sisend.key == pygame.K_LEFT:
            roti_x_kiirus = -roti_kiirus
        elif sisend.key == pygame.K_RIGHT:
            roti_x_kiirus = roti_kiirus
    elif sisend.type == pygame.KEYUP:
        if sisend.key == pygame.K_UP or sisend.key == pygame.K_DOWN:
            roti_y_kiirus = 0
        elif sisend.key == pygame.K_LEFT or sisend.key == pygame.K_RIGHT:
            roti_x_kiirus = 0

```

Selle programmiga on veel probleeme. Kui hoida all paremat noolt, seejärel hoida all vasakut noolt ja lasta parem nool lahti, siis rott jääb seisma, kuigi vasak nool on veel all. Saame igal hetkel kontrollida vajutatud klahve funktsiooniga `pygame.key.get_pressed()`. See on sõnastik, mille võtmeteks on klahvid ja väärtusteks on tõeväärtused, kas nuppu hoitakse all või mitte.

```

import pygame
import sys

ekraan = pygame.display.set_mode((640, 480))
pygame.init()

kell = pygame.time.Clock()

rott = pygame.image.load("rott.png")
rott_x = 100
rott_y = 200

roti_kiirus = 0.25
roti_x_kiirus = 0
roti_y_kiirus = 0

while True:
    dt = kell.tick(60)

    for sisend in pygame.event.get():
        if sisend.type == pygame.QUIT:
            sys.exit()

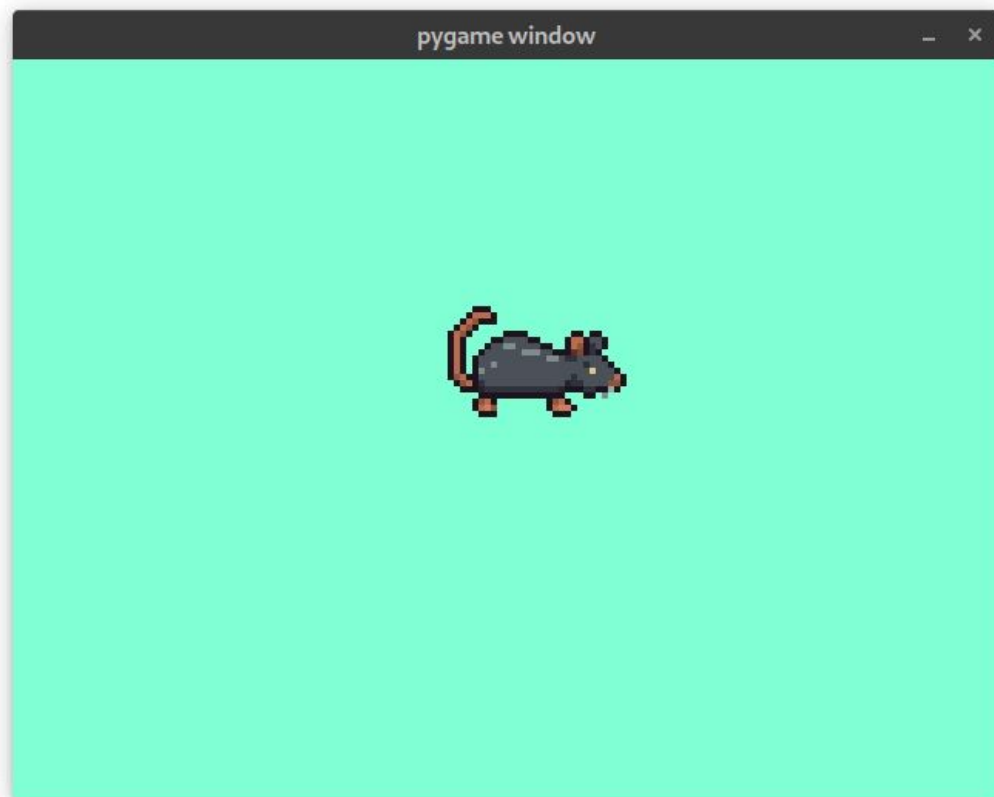
```

```
vajutatud = pygame.key.get_pressed()
if vajutatud[pygame.K_UP]:
    roti_y_kiirus = -roti_kiirus
elif vajutatud[pygame.K_DOWN]:
    roti_y_kiirus = roti_kiirus
else:
    roti_y_kiirus = 0

if vajutatud[pygame.K_LEFT]:
    roti_x_kiirus = -roti_kiirus
elif vajutatud[pygame.K_RIGHT]:
    roti_x_kiirus = roti_kiirus
else:
    roti_x_kiirus = 0

roott_x += roti_x_kiirus * dt
roott_y += roti_y_kiirus * dt

ekraan.fill((128, 255, 212))
ekraan.blit(roott, (roott_x, roott_y))
pygame.display.flip()
```

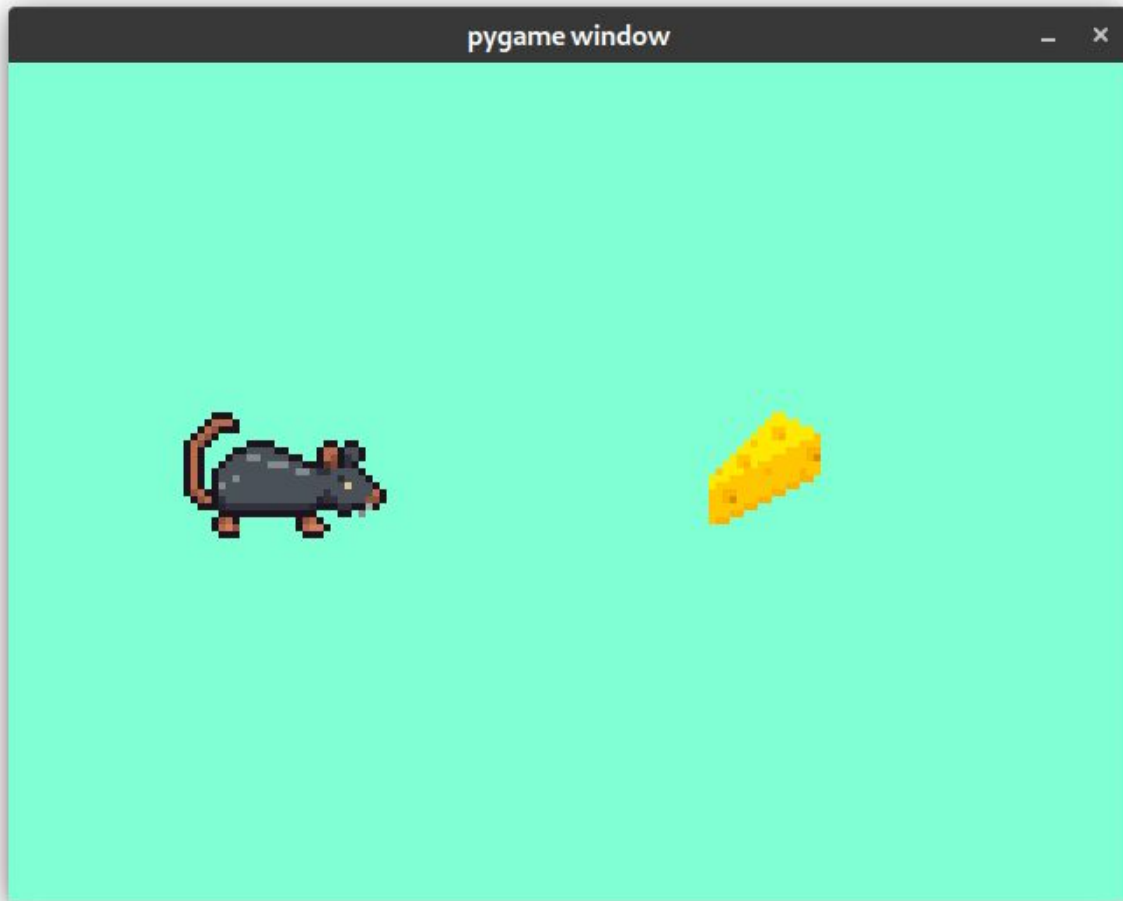


Mõningaid probleeme veel leidub. Kui hoida all nii üles- kui allanoole klahv, siis liigub rott alati ülespoole, sest seda kontrollitakse esimesena. Roti kiirus peaks igas suunas olema võrdne, aga diagonaalis liikudes on ta kiirem. Praeguseks on aga liikumine piisavalt hea, liigume edasi.

Juust

Mis mäng on mäng ilma eesmärgita? Lisame mängu juustu, mille rott peab kätte saama.

Laadi alla `juust.png` ülevalt ja tuleta meelde, kuidas laadida see mängu ning joonistada see. Juustul võiks olla ka koordinaadid muutujatena, sarnaselt rotile.



Kui proovida juustu peale liikuda, siis midagi ei juhtu. Paneme kokkupuutumisel juustu juhuslikku asukohta hüppama!

Kuidas me kontrollime kokkupuudet? Saaksime manuaalselt vaadata, kui roti koordinaadid on juustu koordinaatide lähedal, aga see on palju tööd ning Pygame lubab seda palju lihtsamalt teha.

Kui me paigutame pinna ekraanile meetodiga `blit()`, siis see tegelikult tagastab meile ühe ristküliku objekti ([Rect](#)). See on kasulik objekt, millelt saab kätte pildi koordinaate: keskpunkt, nurgad, servade keskpunktid, laius, kõrgus jne. Muuseas saab sellega kontrollida kahe ristküliku kokkupuutumist üksteisega, kasutades `colliderect()` meetodit. Liigutame kokkupuutumisel juustu uuele juhuslikule asukohale.

```
juust_rect = ekraan.blit(juust, (juust_x, juust_y))
rott_rect = ekraan.blit(rott, (rott_x, rott_y))

if rott_rect.colliderect(juust_rect):
    juust_x = random.randint(0, 640 - juust.get_width())
    juust_y = random.randint(0, 480 - juust.get_height())
```



Lõplik kood kommentaaridega:

```
import pygame
import sys
import random
# Loome akna ning käivitame Pygame'i
ekraan = pygame.display.set_mode((640, 480))
pygame.init()
# Alustame Pygame'i kella
kell = pygame.time.Clock()
# Laadime sisse roti pildi
rott = pygame.image.load("rott.png")
# Algväärtustame roti asukoha
rott_x = 100
rott_y = 200
# Kui kiiresti rott liigub klahvivajutustel
roti_kiirus = 0.25
# Kui kiiresti rott parajasti liigub
roti_x_kiirus = 0
roti_y_kiirus = 0
```

```

# Laadime sisse juustu pildi
juust = pygame.image.load("juust.png")
# Algväärtustame roti asukohta
juust_x = 400
juust_y = 200
# Põhitsükkel, mis hoiab akent lahti
while True:
    # Tiksume kella (ootame umbes 1/60 sekundit)
    # Salvestame kulunud aja eelmisest tiksumisest millisekundites
    dt = kell.tick(60)

    # Käime läbi kõik sisendid eelmisest tiksumisest saati
    for sisend in pygame.event.get():
        # Kui on soovitud aken sulgeda, sulgeme programmi
        if sisend.type == pygame.QUIT:
            sys.exit()

    # Võtame kõik parajasti allavajutatud klahvid
    vajutatud = pygame.key.get_pressed()
    # Kui üles- või allanool on all, määrame roti vertikaalkiiruse
    if vajutatud[pygame.K_UP]:
        roti_y_kiirus = -roti_kiirus
    elif vajutatud[pygame.K_DOWN]:
        roti_y_kiirus = roti_kiirus
    else:
        # Vastasel juhul peatame roti vertikaalselt
        roti_y_kiirus = 0

    # Kui vasak- või paremnool on all, määrame roti horisontaalkiiruse
    if vajutatud[pygame.K_LEFT]:
        roti_x_kiirus = -roti_kiirus
    elif vajutatud[pygame.K_RIGHT]:
        roti_x_kiirus = roti_kiirus
    else:
        # Vastasel juhul peatame roti horisontaalselt
        roti_x_kiirus = 0

    # Liigutame roti asukohta vastavalt kiirustele
    # Korrutame liikumist kulunud ajaga, et rott liiguks sama
    # palju sõltumatult vahele jäänud kaadritest
    rott_x += roti_x_kiirus * dt
    rott_y += roti_y_kiirus * dt

    # Täidame akna akvamariinsinise värviga
    # Sellega kustutame ära kõik, mis siamaani ekraanil on
    ekraan.fill((128, 255, 212))

```

```
# Joonistame pildid nende koordinaatidele
# Saame nende ristküliku objektid
juust_rect = ekraan.blit(juust, (juust_x, juust_y))
rott_rect = ekraan.blit(rott, (rott_x, rott_y))

# Kui juustu ja hiire ristkülikud puutuvad kokku
if rott_rect.colliderect(juust_rect):
    # Liigutame juustu juhuslikele koordinaatidele
    juust_x = random.randint(0, 640 - juust.get_width())
    juust_y = random.randint(0, 480 - juust.get_height())

# Värskendame ekraani
pygame.display.flip()
```

Kui Pythonis klasside kasutamine on selge, siis võiks igale objektile teha enda klassid, millel on isendiväljad suurustest ja kiirustest ning meetodid liigutamise ja muude tegevuste jaoks: <https://stackoverflow.com/a/35642955/12123296>

Kokkuvõte

Vaatasime, kuidas Pygame töötab, joonistasime kujundeid, pilte ja teksti, panime pildi liikuma, liigutasime ise pilti klahvivajutustega, kontrollisime kahe eseme kokkupuudet ja saime lõpuks valmis midagi, mida saab enam-vähem mänguks nimetada. Graafiliste mängude loomise alustalad on nüüd all, aga väga palju jäi käsitlemata.

Et edasi õppida, pakub Pygame erinevaid õpetusi oma [veebilehel](#). Tasub ka uurida Pygame'i [projektide kogumikku](#). Sinna on postitatud palju erinevaid Pygame'iga tehtud mängu koos lähtekoodidega (Releases all), mida tasub uurida.

Ülikoolis saab arvutigraafikaga ja mängudega jätkata ainetes "Arvutigraafika" ([MTAT.03.015](#)) ning "Arvutimängude loomine ja disain" ([MTAT.03.263](#)).

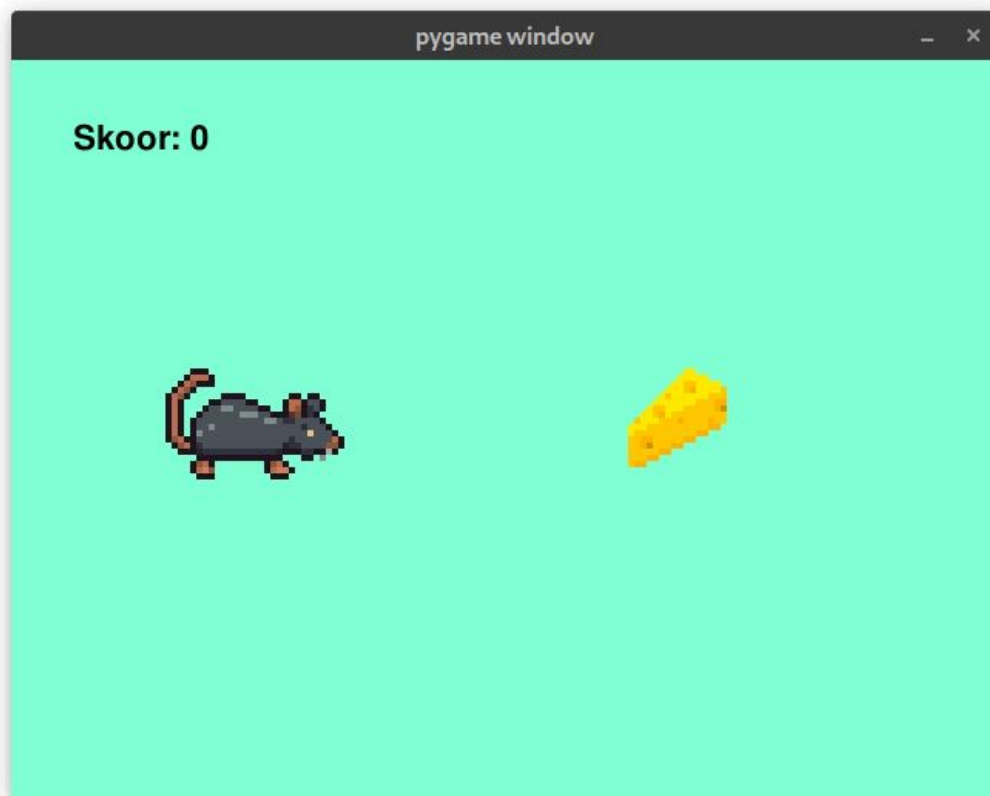
Enesekontrolliküsimused

1. Kuidas joonistada aknasse ristkülik?
 - `pind.draw.rect(värv, x, y, laius, kõrgus)`
 - `pygame.draw.rect(pind, värv, (x, y, laius, kõrgus))`
 - `pind.blit(pygame.draw.rect(värv, laius, kõrgus), x, y)`
 - `pygame.draw(pind, x, y, pygame.rect(värv, laius, kõrgus))`
 - `pygame.draw_rectangle(pind, laius, kõrgus, värv, x, y)`
2. Miks on vaja koordinaatide muutmised korrutada kulunud ajaga?
 - Sest ilma selleta liiguvad esemed liiga aeglaselt
 - Sest ilma selleta liiguvad esemed liiga kiiresti
 - Sest ilma selleta sõltub esemete liikumine kaadrisagedusest
 - Sest ilma selleta on koordinaadid alati ebatäpsed

- Sest see hoiab mängu kaadrisagedust stabiilsena
- 3. Mis on vahet Rect objektile ja Surface objektile?
 - Rect objekt kirjeldab pildi asukohta ekraanil.
 - Rect objektile on kaasas eseme liikumiskiirused.
 - Surface objektile on lisaks pildile kaasas selle asukoht ekraanil.
 - Surface objektile on meetod, mis võimaldab kontrollida kahe pildi kokkupuudet.
 - Need on tegelikult samad.

Ülesanded

1. Täienda peatüki jooksul loodud mängu skooriga, mis loeb juustu kättesaamise arvu. Kuva see skoor kuskil mänguaknas tekstina.



2. Kirjuta Pygame'iga animatsioon, kus mingi pilt või kujund liigub sinusoidi trajektoori või ringjoone järgi.

3. Kirjuta ise Pygame'iga üks mäng, millel on mingi eesmärk. Graafika joonistada ise või võtta Internetist. Viimase puhul tuleb lähtekoodis kommentaarina allikale viidata. Mõned ideed:

- Labürindist pääsemise mäng. Labürindi võib näiteks võtta tekstifailist, sarnaselt [pykkarile](#).
- Lihtne [Flappy Birdi](#) kloon. Gravitatsiooni asemel võib vabalt ise kontrollida üles-alla liikumist.
- Esemel lendavad mängija suunas, kes peab eest ära hüppama, et ellu jääda.

8. Keerulisemad Pythoni võimalused

Käesolev Pythoni kursus teeb hea sissejuhatuse Pythoniga programmeerimisse, aga puhtalt aine läbimisega ennast võluriks kutsuda ei saa. Selle jaoks peab läbima selle viimase silmaringimaterjalide peatüki.

Siin peatükis vaatame, kuidas teha lihtsaid asju keerulisemalt, et vähendada üleliigse koodi kirjutamist. Õpime keerulisemaid Pythoni võimalusi, et teha elu lihtsamaks.

Ettevalmistus

Selle peatüki läbimiseks piisab Pythoni installatsioonist: midagi paigaldama ei pea.

Et peatükist täiesti aru saada, peab olema läbitud õpiku [esimene](#) ja [teine](#) osa.

Lühem if-lause

Vahepeal on vaja millegi väga lihtsa jaoks kasutada if-lauset. Näiteks, sõna kääne muutub osastavaks, kui see viitab mitmele esemele: ostukorvis on 1 **toode**, aga ostukorvis võib olla mitu **toodet**.

Sellises olukorras peame kasutama if-lauset:

```
if toodete_arv == 1:
    print("Ostukorvis on {} toode".format(toodete_arv))
else:
    print("Ostukorvis on {} toodet".format(toodete_arv))
```

Et vähendada kirjutatud koodi, saab if-lause kirjutada ühele reale:

```
print("Ostukorvis on {} {}".format(toodete_arv, "toode" if toodete_arv == 1 else "toodet"))
```

Lühema if-lause valem:

```
väärtus_kui_tõene if tingimus else väärtus_kui_väär
```

Näited:

```
>>> "tõene" if True else "väär"
'tõene'
>>> "tõene" if False else "väär"
'väär'
>>> "paaris" if 20 % 2 == 0 else "paaritu"
'paaris'
>>> "paaris" if 21 % 2 == 0 else "paaritu"
```

```
'paaritu'
>>> "tühi" if len([]) == 0 else "täis"
'tühi'
>>> "tühi" if len([5]) == 0 else "täis"
'täis'
```

Proovi kirjutada lühike tingimuslause, mis annab väärtuseks "fizz", kui arv jagub kolmega ning arvu ise, kui see ei jagu kolmega.

Meil on järjend, mis sisaldab esemeid, mida on keelatud lennukile kaasa võtta keelatud.

```
keelatud = ["veepudel", "kahvel", "žilett", "ilutulestikud", "elevant"]
```

Kirjuta programm, mis küsib kasutajalt eset ning väljastab, kas see on keelatud või mitte nii, et programmil ei ole ühtegi taanet.

```
Sisesta ese: telefon
telefon ei ole keelatud lennukile kaasa võtmiseks
```

```
Sisesta ese: veepudel
veepudel on keelatud lennukile kaasa võtmiseks
```

Järgendi hõlmamine

Kui tahame luua uut järjendit ja täita seda mingite andmetega, oleme siamaani loonud uue järjendi ning seejärel for-tsükliga sinna elemente juurde lisanud.

Näiteks, meil on järjend sõnadest ja me tahame saada järjendit nende pikkustest.

```
sõned = ["Python", "on", "üldotstarbeline", "interpreteeritav",
"programmeerimiskeel"]

pikkused = []
for sõne in sõned:
    pikkused.append(len(sõne))

print(pikkused) # [6, 2, 15, 16, 19]
```

Sama asja saab kirja panna vähema koodiga:

```
sõned = ["Python", "on", "üldotstarbeline", "interpreteeritav",
"programmeerimiskeel"]

pikkused = [len(sõne) for sõne in sõned]

print(pikkused) # [6, 2, 15, 16, 19]
```

Seda võtet nimetatakse järjendi hõlmamiseks (ingl. k. *list comprehension*).

Veel üks näide: tahame saada kahe astmeid. Pikemalt saab teha nii:

```
kahe_astmed = []
for arv in range(10):
    kahe_astmed.append(2**arv)

print(kahe_astmed) # [1, 2, 4, 8, 16, 32, 64, 128, 256, 512]
```

Järjendi hõlmamisega:

```
kahe_astmed = [2**arv for arv in range(10)]

print(kahe_astmed) # [1, 2, 4, 8, 16, 32, 64, 128, 256, 512]
```

Ülesanne võib olla natuke keerulisem. Näiteks soovime kahe astmeid ainult siis, kui need lõppevad neljaga.

```
kahe_astmed_neljaga = []

for arv in range(23):
    if 2**arv % 10 == 4:
        kahe_astmed_neljaga.append(2**arv)

print(kahe_astmed_neljaga) # [4, 64, 1024, 16384, 262144, 4194304]
```

Ka lisatingimusi saame panna järjendi hõlmamisse:

```
kahe_astmed_neljaga = [2**arv for arv in range(23) if 2**arv % 10 == 4]

print(kahe_astmed_neljaga) # [4, 64, 1024, 16384, 262144, 4194304]
```

Proovi järjendi hõlmamisega luua järjend, mis sisaldab ainult arve, mis ei jagu arvudega 3 ega 5:

```
[1, 2, 4, 7, 8, 11, 13, 14, 16, 17, 19, ...]
```

Meil on järjend, mis sisaldab kõige sagedasemaid sõnade põhivorme (lemmasid) koos nende järkudega:

```
[(1, "olema"), (2, "ja"), (3, "see"), (4, "tema"), (5, "mina"), (6, "ei"), (7, "et"), (8, "kui"), (9, "mis"), (10, "ka")]
```

Proovi järjendi hõlmamisega luua järjend, mis jätab alles ainult sõnad:

```
["olema", "ja", "see", "tema", "mina", "ei", "et", "kui", "mis", "ka"]
```

Proovi hõlmamist ka kuhjadega, sõnastikega ja ennikutega.

Funktsioonide kasutamine parameetrites

Sorteerimine

Kui me tahame sorteerida arvude järjendit, siis saab kasutada funktsiooni `sorted` või meetodit `sort`.

```
>>> arvud = [1, 5, 9, 2, 6, 5]
>>> sorted(arvud) # ei sorteeri muutujat, tagastab sorteeritud järjendi
[1, 2, 5, 5, 6, 9]
>>> arvud
[1, 5, 9, 2, 6, 5]
>>> arvud.sort() # sorteerib muutuja
>>> arvud
[1, 2, 5, 5, 6, 9]
```

Mis siis, kui meil on vaja midagi keerulisemat sorteerida? Näiteks meil on järjendisse salvestatud ennikud, mille üks element on arv. Kuidas öelda sorteerimisfunktsioonile, et sorteerida arvu järgi?

```
maletajad = [("Ding Liren", 3), ("Nepomnjaštši", 4), ("Caruana", 2), ("Carlsen", 1)]
```

Sorteerimisfunktsioonidel on parameeter `key`, millele saab panna väärtuseks funktsioone. See funktsioon jooksutatakse iga järjendi elemendi peal läbi nii, et parameetriks pannakse element ning järjend sorteeritakse selle funktsiooni tagastuste põhjal. Meie järjendi puhul saaksime teha funktsiooni, mis tagastab enniku teise liikme.

```
def teine_liige(ennik):
    return ennik[1]
```

Nüüd peab meelde tuletama, et funktsioonid on ka muutujad, lihtsalt neid saab välja kutsuda. Seega meie funktsioon on salvestatud muutujasse `teine_liige`. Sorteerime järjendi nii, et `key` parameetri väärtuseks läheb see funktsioon.

```
>>> maletajad
[('Ding Liren', 3), ('Nepomniachtchi', 4), ('Caruana', 2), ('Carlsen', 1)]
>>> maletajad.sort(key=teine_liige)
>>> maletajad
[('Carlsen', 1), ('Caruana', 2), ('Ding Liren', 3), ('Nepomniachtchi', 4)]
```

Töötas! Proovime veel. Kui meil on sõnede järjend, sorteeritakse neid tavaliselt tähestiku järjekorras:

```
>>> sõned = ["aaaa", "b", "ccc", "dddd", "ee"]
```

```
>>> sorted(sõned)
['aaaa', 'b', 'ccc', 'dddd', 'ee']
```

Aga mis siis, kui tahame neid sorteerida sõnepikkuse järjekorras? Peaksime tegema funktsiooni, mis tagastab sõne pikkuse.

```
def sõne_pikkus(sõne):
    return len(sõne)
```

Oota, selline funktsioon on juba olemas ja seda kasutasime isegi siin funktsioonis: len. Paneme sorteerimisel parameetri key väärtuseks lihtsalt selle!

```
>>> sorted(sõned, key=len)
['b', 'ee', 'ccc', 'aaaa', 'dddd']
```

Saime isegi meeldiva ja loetava koodi.

Kui funktsiooni pole veel kirjutatud, peame selle ise defineerima, nagu tegime esimeses näites. Üsna tüütu on nii lihtsat funktsiooni ise defineerida. Võiks ju olla lihtsam viis funktsiooni defineerimiseks ilma, et peab mitu rida koodi juurde kirjutama.

Lambda-funktsioonid

Et vähendada koodi kirjutamist, saame kasutada lambda-funktsioone. Need on funktsioonid, mida saab kirjutada lühidalt ja kasutada anonüümselt ehk neid ei pea muutujasse salvestama.

Vaatame funktsiooni, mida me enne kirjutasime:

```
def teine_liige(ennik):
    return ennik[1]
```

Seda saab samaväärselt kirjutada lambdana:

```
teine_liige = lambda ennik: ennik[1]
```

Defineerisime just funktsiooni teine_liige, mis võtab parameetriks muutuja ennik ning tagastab selle teise liikme.

Saame neid samamoodi kasutada teiste funktsioonide parameetrites:

```
>>> maletajad
[('Ding Liren', 3), ('Nepomniachtchi', 4), ('Caruana', 2), ('Carlsen', 1)]
>>> maletajad.sort(key=lambda ennik: ennik[1])
>>> maletajad
[('Carlsen', 1), ('Caruana', 2), ('Ding Liren', 3), ('Nepomniachtchi', 4)]
```

Tegelikult on [sellised funktsioonid](#) ka juba Pythoni standardteegis olemas, aga nende kasutamiseks peab ühe mooduli importima ning nendega sai teha hea sissejuhatuse lambdadesse.

Vaatame veel funktsioone, mis nõuavad parameetritena teisi funktsioone.

Funktsioon map

Üks asi, mis võib tihti ette tulla, on kogu järjendi elementidele mingi funktsiooni rakendamine. Näiteks on vaja arvusõnede järjend muuta arvude järjendiks. Selle jaoks peaks need for-tsükliga läbi käima ja kuskile uude järjendisse lisama.

```
sõned = ["3", "1", "4", "1", "5", "9"]

arvud = []
for sõne in sõned:
    arvud.append(int(sõne))

print(arvud) # [3, 1, 4, 1, 5, 9]
```

See on päris palju koodi nii lihtsa asja jaoks. Et säästa koodi kirjutamist, on Pythonis funktsioon map. Sellele saab parameetriteks panna ühe funktsiooni ning järjendi, et jookсутada seda funktsiooni kõikide järjendi elementide peal. See tagastab objekti, mida saab läbida for-tsükliga või muuta järjendiks funktsiooniga list. Eelmine kood uuesti, kasutades funktsiooni map:

```
sõned = ["3", "1", "4", "1", "5", "9"]

arvud = list(map(int, sõned))

print(arvud) # [3, 1, 4, 1, 5, 9]
```

Tegime palju lühema koodiga sama asja.

Võimalik, et funktsioon nõuab mitut parameetrit, näiteks round, mille esimene parameeter on ümardatav arv ja teine on kohtade arv, mitmeni peaks ümardama.

```
>>> round(3.14159265, 2)
3.14
```

Sel juhul saab map parameetriteks panna 2 järjendit: esimesed parameetrid ja teised parameetrid. Ümardame näiteks ujukomaarvude järjendi arvud vastavalt teisele järjendile.

```
arvud = [3.14159, 2.71828, 1.41421, 6.28318, 1.61803]
kohtadeni = [2, 3, 4, 3, 2]

ümardatud = list(map(round, arvud, kohtadeni))
```

```
print(ümardatud) # [3.14, 2.718, 1.4142, 6.283, 1.62]
```

Proovi map funktsiooniga kõik järjendi sõned suurtähestada (sõne meetod upper).

Funktsioon filter

Vahepeal on vaja järjendeid filtreerida. Näiteks tahame järjendist jätta alles ainult positiivsed arvud. Võib jälle teha for-tsükli:

```
arvud = [3, -6, 1, -2, 4, -8, 1, -3]
positiivsed = []

for arv in arvud:
    if arv > 0:
        positiivsed.append(arv)

print(positiivsed) # [3, 1, 4, 1]
```

Seda saab lühemalt teha funktsiooniga filter. See funktsioon nõuab parameetritesse funktsiooni ning järjendit. Tagastatud järjendisse jäetakse alles ainult need elemendid, mille peal funktsioon tagastab True. Eelmine koodijupp funktsiooniga filter:

```
arvud = [3, -6, 1, -2, 4, -8, 1, -3]

positiivsed = list(filter(lambda arv: arv > 0, arvud))

print(positiivsed) # [3, 1, 4, 1]
```

Seda saab ka kasutada sõnede peal. Näiteks saame alles jätta kõik tähed ja tühikud tekstis ilma muude sümboliteta:

```
lause = "„Funktsionaalprogrammeerimine on lihtne,” ütles mitte keegi."

print("".join(filter(lambda x: x.isalpha() or x == " ", lause)))
# Funktsionaalprogrammeerimine on lihtne ütles mitte keegi
```

Kuna filter tagastab järjendilaadse objekti, peab sõne saamiseks selle kokku liitma sõne meetodiga join.

Kasuta filter funktsiooni, et saada kätte salajane sõnum järgnevast tekstist:

```
XXSXSSSSSSXAXSSSSSSLXXXXXXAXSSSSSSJXXXXXXAXSSSSSNXXXXXEXSSSSX
XXXXXXSSSSSSXÖXXXXXXNXXXXXXUXXXXXXMXXX
```

Veel üks huvitav funktsioon on [reduce](#).

Funksioon zip

Kui meil on mitu järjestit sarnaste andmetega ja me tahame neid kokku pakkida, siis peaksime need for-tsükliga indekseerima:

```
eesnimed = ["Jüri", "Mailis", "Tõnis", "Tanel", "Urmas"]
perenimed = ["Ratas", "Reps", "Lukas", "Kiik", "Reinsalu"]

ministrid = []

for i in range(len(eesnimed)):
    ministrid.append((eesnimed[i], perenimed[i]))

print(ministrid)
# [('Jüri', 'Ratas'), ('Mailis', 'Reps'), ('Tõnis', 'Lukas'), ('Tanel', 'Kiik'), ('Urmas', 'Reinsalu')]
```

Funksioon zip lubab seda palju lihtsamalt teha:

```
eesnimed = ["Jüri", "Mailis", "Tõnis", "Tanel", "Urmas"]
perenimed = ["Ratas", "Reps", "Lukas", "Kiik", "Reinsalu"]

print(list(zip(eesnimed, perenimed)))
# [('Jüri', 'Ratas'), ('Mailis', 'Reps'), ('Tõnis', 'Lukas'), ('Tanel', 'Kiik'), ('Urmas', 'Reinsalu')]
```

See on kasulik, kui meil on näiteks erinevaid andmeid samade asjade kohta ja me tahame neid kokku liita.

Funksioonile zip võib sisse sööta lõpmatu arvu parameetreid:

```
>>> list(zip([1], [2], [3], [4], [5]))
[(1, 2, 3, 4, 5)]
```

Seega, kui me sisestame saadud paarid zip funktsiooni parameetritesse, siis saame tagasi eesnimede ja perenimede järjestid:

```
>>> list(zip(('Jüri', 'Ratas'), ('Mailis', 'Reps'), ('Tõnis', 'Lukas'),
('Tanel', 'Kiik'), ('Urmas', 'Reinsalu')))
[('Jüri', 'Mailis', 'Tõnis', 'Tanel', 'Urmas'), ('Ratas', 'Reps', 'Lukas', 'Kiik', 'Reinsalu')]
```

Aga kui meil on paarid järjendi sees, siis kuidas kõiki elemente funktsiooni parameetritesse saada?

Et järjestit funktsiooni parameetritena kasutada, tuleb järjest panna parameetriks ning selle ette kirjutada tärn. Proovime algul lihtsama näitega: paneme funktsiooni round parameetriks kaheliikmelise järjendi.

```
>>> round(*[3.14159265368, 2])  
3.14
```

Proovime sama võtet zip funktsiooni peal:

```
>>> ministrid = [('Jüri', 'Ratas'), ('Mailis', 'Reps'), ('Tõnis',  
'Lukas'), ('Tanel', 'Kiik'), ('Urmas', 'Reinsalu')]  
>>> list(zip(*ministrid))  
[('Jüri', 'Mailis', 'Tõnis', 'Tanel', 'Urmas'), ('Ratas', 'Reps',  
'Lukas', 'Kiik', 'Reinsalu')]
```

Töötab! Oleme edukalt järjestid kokku ja lahti pakkinud.

Kokkuvõte

Kasutasime erinevaid võtteid, et vähendada üleliigset koodi kirjutamist. Võibolla tundub tõesti, et see teeb lihtsaid asju keerulisemaks, aga piisava harjutamisega muutuvad need lihtsamaks ning pikema koodi kirjutamise soov kaob ära.

Kuigi õpitud võtted lubavad lahendada peaaegu kõiki ülesandeid ühe reaga, peab meelde tuletama, et [Pythoni ametlik stiiliõpetus](#) nõuab, et read on maksimaalselt 79 tähemärgi pikkused. Päris programmides tuleks kood mõistlikult paigutada mitmele reale.

Paljud siin peatükis rakendatud funktsioonid on seotud funktsionaalprogrammeerimisega. Sellesse teemasse süveneb aine "Programmeerimiskeeled" ([MTAT.03.006](#)) keeltega Haskell ja Scala.

Enesekontrolliküsimused

1. Millised on korrektsed järjendi hõlmamised?
 - `[float(arv) for arv in ["3.14", "2.71", "1.41"]]`
 - `[for arv in [1, 2, 3, 4]: arv**2]`
 - `[arv if arv % 2 == 0 for arv in range(10)]`
 - `[arv if arv % 2 == 0 else 0 for arv in range(10)]`
 - `[arv for arv in range(10) if arv % 2 == 0]`
2. Mis juhtub, kui funktsioonidele zip või map antud järjendid on eri pikkusega?
 - Visatakse erind
 - Lühemale järjendile liidetakse nulle või tühisõnesid juurde, olenevalt teisest elemendist
 - Pikema järjendi üleliigne osa ignoreeritakse
 - Lühemale järjendile selle viimaseid elemente juurde
 - Lühema järjendi elemente hakatakse algusest võtma

3. Mida tagastab järgmine koodilõik?

```
list(map(lambda x, y: x - y, [10, 8, 6], [1, 2, 3]))
```

- [9, 6, 3]
- [10, 8, 6]
- [10, 8, 6, 1, 2, 3]
- [11, 10, 9]
- [10, 16, 24]

Ülesanded

1. Antud on järjestid nimedest, matriklinumbritest ja keskmistest hinnetest.

```
nimed = ["Jaan", "Martin", "Katrin", "Margus", "Tiiu", "Jüri", "Anna",  
"Sirje", "Ülle", "Kristjan", "Anne", "Julia", "Andres", "Marina",  
"Rein", "Aivar", "Tiina", "Urmas", "Toomas", "Maria"]  
matriklinumbrid = ["B69310", "B28761", "B79826", "B14207", "B16122",  
"B61619", "B14708", "B59695", "B50264", "B32270", "B88961", "B73302",  
"B29125", "B87856", "B48386", "B22124", "B52814", "B80444", "B56290",  
"B57742"]  
hinded = [4.15, 3.61, 3.59, 3.28, 4.5, 4.6, 4.16, 3.83, 4.97, 4.26,  
4.92, 3.93, 3.64, 4.12, 4.03, 4.75, 4.38, 4.65, 3.09, 4.04]
```

Kirjuta **ühe reaga** funktsioon, mis tagastab ennikud nendest, kelle keskmine hinne on vähemalt 4.6, sorteeritud keskmise hinde järgi kahanevalt.

```
>>> stipisaajad(nimed, matriklinumbrid, hinded)  
[('Ülle', 'B50264', 4.97), ('Anne', 'B88961', 4.92), ('Aivar', 'B22124',  
4.75), ('Urmas', 'B80444', 4.65), ('Jüri', 'B61619', 4.6)]
```

2. Tekstifaili evolutsioon.txt on salvestatud tekst imelikul kujul: tekst algab ülevalt paremalt ja liigub alla. Kirjuta **ühe reaga** funktsioon korrasta, mis võtab parameetrina failinime ning tagastab õiges järjekorras loetud teksti. Lahendamiseks piisab siin peatükis kasutatud funktsioonidest, v.a faili sisselugemine. Faili sulgemine siin ülesandes pole tähtis.

```
nne mE  
g,raon  
, fnsd  
audtl  
enl e  
vd mbs  
o hoes  
laasa  
vrvtuf  
eee to  
d wir  
.bbofm  
eenus  
iedl
```

```
>>> with open("ü11.py") as f: print(len(f.readlines()))
1
>>> %Run 'ü11.py'
>>> korrasta("evolutsioon.txt")
'Endless forms most beautiful and most wonderful have been, and are
being, evolved. '
```