Tallinna Ülikool

Digitehnoloogiate Instituut

Täiendavad Python-keele õppematerjalid: Time-teek ja ajaarvamine

Seminaritöö

Autor: Harry Kaarma

Juhendaja: Inga Petuhhov

Tallinn 2015

Sisukord

Sissejuhatus

Ajaarvamine programmeerimiskeeltes on üks nendest asjadest, mis võib segadust tekitada isegi näiliselt kogenud programmeerijates. See pole iseenesest üllatav: ajaarvamisega seotud probleemid on tihti ootamatult erineva ülesehitusega, ja vajavad lahendamiseks teistsuguseid lahendusi. Selliste lahenduste leidmisel on suureks abiks, kui on olemas kindlad teadmised mingis antud programmeerimiskeeles ajaarvamisega seonduvatest põhitõdedest, ja juba olemasolevatest tööriistadest. Selliseid teadmisi aitab kergemini ja kindlamalt omastada õppematerjal. Probleem seisneb selles, et Python-keele jaoks ei ole veel olemas eesti keelseid ajaarvamisega seotud õppematerjale. Seega seadsingi endale eesmärgiks luua selline õppematerjal, mis käsitleks ajaarvamise põhitõdesid Python-keeles, ja demonstreeriks keelega kaasas olevates teekides sisalduvate funktsioonide kasutamist ajaarvamisega seotud probleemide lahendamiseks. See õppematerjal peaks olema kindla fookusega just ajaarvamisele: liiga laialivalguv lähenemine võib muuta selgete teadmiste omandamise keerulisemaks. Samuti peaks see materjal olema piisavalt lihtne, et seda suudab mõista inimene, kellel on Pythonist kindlalt olemas vaid algpõhised teadmised. Põhiliseks sihtgrupiks oleks seega näiteks „Programmeerimise alused” kursuse lõpetanud tudeng. Loodav materjal peaks sisaldama nii seletusi ja näiteid, kui ka lihtsamaid ülesandeid, et meeldejätmist soodustada. Materjali plaanin üles ehitada Pythoni ametlikule dokumentatsioonile tuginedes (Python Software Foundation, 2015). Loodetavasti saab see õppematerjal olema kasulik kõigile, kes pärast esimesi programmeerimiskursusi Pythoni kasutamist jätkavad.

1. Sihtgrupp ja metoodika

Nagu sissejuhatuses mainitud, oleks siin loodava õppematerjali sihtgruppiks üliõpilane (või miks mitte ka südikas iseõppija), kes mõistab vähemalt Python-keele aluspõhimõtteid (süntaksi ja põhilist programmeerimise loogikat). Kuna need asjad peaks olema väga selged igaühele, kes on läbinud TLU „Programmeerimise alused” kursuse, võib neid tudengeid lugeda põhiliseks sihtgrupiks.

Õppematerjali ajaarvamise kohta Python-keeles oleks esiteks vaja seetõttu, et eesti keelset materjali antud kujul veel ei ole (vähemalt mitte ülikoolile kättesaadavat), ja teiseks seetõttu, et isegi olemasolevad Inglise keelsed juhendid ja õppematerjalid on minu arvates kas üpris pinnapealsed, või veidi liiga laialivalguvad. Seega tundub mulle otstarbekama lahendusena kirjutada antud teemal lihtsalt uus eesti keelne materjal, kui et hakata tõlkima mõnda inglise keelset.

Sisu koha pealt leian, et õppematerjalis oleks mõtekas käsitleda kahte põhilist Python-keeles juba olemasolevat ajaarvamisega seotud teeki: *time*, ja *datetime*. Teised Pythonis ajaarvamisega seotud kolmanda osapoole teegid on vähemalt mingil määral nende põhjal või nende eeskujul üles ehitatud, ja täidavad enamasti üpris spetsiifilisi ülesandeid. Seega arvan, et loodavas õppematerjalis oleks mõtekam põhjalikumalt keskenduda kahele Pythonis olemasolevale teegile, et luua teadmised, mille abil oskaks materjali kasutajad hiljem vajaduse korral ka mistahes neil kahel teegil põhinevaid juurdearendusteeke mõista. Veel võib potensiaalsetest teekidest ära mainida Pythonis olemasoleva *calendar* teegi, mis on ehituselt ja otstarbelt piisavalt iseennastseletav (see koostab automaatselt kalendreid), et ajaarvamise põhiteekidega tuttav inimene peaks ilma suure vaevata suutma seda lihtsalt dokumentatsiooni abil kasutada, ja selle käsitlemine materjalis ei oleks seega eriti mõtekas.

Ehituselt peaks materjal olema illustreeriva kallakuga: Iga paari kirjeldava tekstilõigu juurde peaks kuuluma ka koodinäide koos oodatud väljundiga. See võimaldab hoida teksti konkreetse, lastes samas koodinäidetel iseenda eest rääkida. Loodava õppematerjali oodatud kasutusviis on reaalsete katsetustega paralleelselt. Seega peaks ka koodinäited olema teksti kujul, mitte pildid. Nii saab õpilane soovi korral need otse materjalist oma programmi ümber tõsta, ja neid modifitseerides ka ise lihtsasti katsetusi teha. Pärast piisava hulga uute funktsioonide tutvustust peaks olema väike komplekt harjutusi, mis peaks tähelepaneliku õpilase jaoks olema ette antud informatsiooni ja koodinäidete põhjal lihtsasti lahendatavad. Nende ülesannete idee pole õpilasi liigselt pinnida, vaid aidata praktilise kasutuse näidete abil neil just loetu olemust meelde jätta. Kõige olulisemate struktuuride kohta peaks lisama ka paar tabelit, mis nende ehitust ja selle iseärasusi kirjeldavad. Tähtsate struktuuride all mõtlen siin struktuure, millele suur osa teegi funktsionaalsusest mingil viisil baseerub, näiteks *time*-teegi *struct_time*, ja *datetime* teegi *datetime* objekt, mis koosnevad

mitmest väljast, ja mille täpne olemus võib algul materjali lugejale segaseks jääda.

Nagu mainisin, otsustasin õppematerjali luua ise, mitte lihtsalt ümber tõlkida, nii et töö juures kasutatavaks kirjanduseks loeksin vastavad dokumentatsioonilehеküljed Pythoni ametlikus dokumentatsioonis, seega time teegi dokumentatsiooni (Python Software Foundation, 2015), ja datetime teegi dokumentatsiooni (Python Software Foundation, 2015). Lisaks veel mõne entsüklopeediasissekande ajaarvamise, täpsemalt ajatsoonide teemal, mis oleks abiks sissejuhatava põhimõtete peatüki kirjutamise juures (Wikimedia Foundation, 2015).

2. Valmis õppematerjal

Õppematerjal on siinkohal esitatud ilma tiitellehe ja sisukorrata.

Eelsõna

Ajaarvamine programmeerimiskeeltes on tihti üheks programmeerijate suurimaks komistuskiviks. See on iseenesest täiesti mõistetav: paigas olevad (ja samas funktsioonides kasutatavad) ajaarvamissüsteemid ei hiilga tihti oma inimloetavuse poolest, ja nendega tehete tegemine vajab veidi teistsugust lähenemist kui tavalised andmeformaadid. Suurimaks probleemiks aga on tihti just teadmatus programmeerimiskeeles olemas olevatest funktsioonidest: väga suur osa ajaarvamisprobleemidest on tekkinud tänu programmeerijate kalduvusele hakata ratast uuesti leiutama. See õppematerjal ongi eelkõige mõeldud demonstreerima Python-keeles juba olemasoleva ajaarvamise teegi võimekust. Õppematerjal peaks olema selgelt loetav „Programmeerimise Alused” kursuse läbinud inimesele. See tähendab, et Pythoni põhifunktsioonide ja süntaksi mõistmine on eeldatud, ja neid pikalt üle ei seletata.

Antud õppematerjal koostati Tallinna Ülikoolis, seminaritöö raames. Näidisprogrammide loomiseks kasutati Python 3.5.0 64-bitist versiooni.

2.1. Ajaarvamise põhimõtted

Enne põhilise materjali juurde asumist kordame üle mõned mõisted ja algtõed. Nagu programmeerimise puhul ikka, on väga tähtis, et kõik osapooled oleksid „samal leheküljel”.

Meile tuttavate metoodikate järgi jaotatakse aeg ühikutesse järgmiselt: aastad, kuud, nädalad, päevad, tunnid, minutid, sekundid. Programmeerimise juures aga jaotatakse aeg kõige madalamal tasandil ainult **epohhist möödunud sekunditeks (*seconds since epoch*)**. **Epohhiks (*epoch*)** ehk aja alguseks loetakse Unix-põhistel süsteemidel kuupäeva 12:00(AM) 01.01.1970. Need sekundid teisendatakse arvutis ümber suuremateks ajaühikuteks ainult inimloetavuse eesmärgil: ajaga seonduvad tehted tehakse algtasandil ikkagi epohhist möödunud sekundite põhjal. Seda tehakse üsna lihtsal põhjusel: sekundite lugemine mingist hetkest on alati selle hetke suhtes samas proportsioonis. Kõik sekundist suuremad ajaühikud aga on mingil määral kontekstitundlikud: mitte ainult ajatsoonide mõttes, vaid ka sellepärast, et nad kuuluvad kalendrisse, ja kalender peab jääma planeet Maa aastaaegadega sünkrooni. Selleks lisatakse kalendrisse vahel lisa-, ehk liigpäevi, või sekundeid. Seega, kui palju aastaid, päevi või minuteid on kahe ajahetke vahel möödunud oleneb kalendri iseärasustest antud perioodil, aga see, kui palju sekundeid on möödunud, on konkreetne arv. Korrektsete tehete tegemiseks on vaja just konkreetseid arve. Antud materjalides käsitletud ajaarvamisteekide mõte seisnebki suuresti sellest, et nende funktsioonid suudavad kasutatava kalendri põhjal kiiresti arvutada epohhist möödunud sekundid ümber korrektse vormistusega kuupäevaks (ja vastupidi), säästes nii kasutajat liigsest matemaatilisest peavalust.

Eelnevaga seonduvalt on väga tähtis mitte segi ajada möödunud aega, ja kellaaega. Möödunud aeg on kõikjal sama (fookuse hoidmise huvides ärme laskume siinkohal relatiivsuse teemadesse), kuid kellaaeg, millele see vastab, oleneb kohast planeedil. Üldistavalt võib öelda, et analoogkell peaks lõpetama ühe ringi (mõlemad seierid näitavad 12 peale) öö kõige pimedamal hetkel (kesköö), ja teise ringi päeva kõige valgemal hetkel (lõuna). Sellise seaduspärasuse tagamiseks on iga koht meie planeedil liigitatud mingi **ajatsooni (*time zone*)** alla. Üldiselt väljendatakse kokkuleppe põhjal ajatsoone selle järgi, kui mitme tunni võrra erineb nende kellaaeg planeet maa null-meridiaanil asuva Greenwichi observatooriumi kellaajast. Greenwichi enda ajatsooni tähistatakse **GMT (*Greenwich Mean Time*)**+0, või vahel ka teise lühendiga, **UTC (*Coordinated Universal Time*)**+0. Idapoolsetes ajatsoonides liidetakse kellaajale tunde, (ajatsoonid UTC+1, UTC+2, jne.), Läänepoolset lahutatakse kellaajast tunde (UTC-1, UTC-2, jne.), et keskpäev oleks ikka umbes kell 12:00. Lisaks eristamisele UTC+/- järgi on igal ajatsoonil ka oma nimetus, ja sellele vastav lühend.

Kuna mõnedes paikades muutub päevavalguse režiim aastaaegade vahetumise tõttu üsna palju,

kasutatakse nendes vahest suvel ja talvel erinevaid ajatsoone. Seda praktikat nimetatakse **suve/talveajaks**, või inglise keeles **DST (*Daylight Savings Time*)**. Näiteks Eestis kasutatakse talvel ajatsooni UTC+2, mida nimetatakse ka EET (*Eastern European Time*), ja suvel ajatsooni UTC+3, mida nimetatakse ka EEST (*Eastern European Summer Time*).

Kellaaega ise kujutatakse ühel kahest põhilisest viisist: kas **24-tunnise kellana (*24-hour clock*)**, nagu osadel digitaalkelladel, kus päev kestab 00:00 – 23:59, või **12-tunnise kellana (*12-hour clock*)**, mis teeb kaks 12-tunnist ringi, kus kellaaeg tähistatakse esimesel ringil kui AM (enne keskpäeva, ladina k. *ante meridiem)*, ja teine kui PM (peale keskpäeva, ladina k. *post meridiem)*, nagu tüüpiliselt analoogkelladel, kus päev kestab 12:00AM – 11:59PM.

Lõpuks tasub veel märkida, et kuna programmeerimise juures võib vahel olla vaja äärmist täpsust, on enamik arvutite ajamõõtjaid vähemalt millisekundi täpsusega. Sellepärast on ka epohhist möödunud sekundite arv mällu salvestatud ujukomaarvuna (*float*).

2.2. Time teek

Alustuseks tasub ära märkida, et kõik ajafunktsioonide käest saadud ajad sõltuvad arvuti enda kellast, ja võivad vahel riistvara ja tarkvara iseärasuste tõttu erineda.

Kui on soov Pythonis mingi uue teegiga töötada, tuleks see esimese asjana importida (või siis lihtsalt sellest vajalikud funktsioonid)

üldiselt:

```
import time
```

või spetsiifiliselt:

```
from time import localtime, strftime
```

Järgnevad näited on loodud esimese, üldise impordiga, ja sellepärast on funktsiooni ette kirjutatud teegi nimi, kujul „time.funktsioon()”. Spetsiifilise impordi korral pole „time.” ette kirjutamine vajalik.

2.2.1. Aja initsialiseerimine ja kuvamine

Nagu eelnevalt mainitud, arvestab Python aega sekundites pärast epohhi, ehk aja algust. Selle sekundite arvu saab kätte, kui omistada muutujale funktsioon *time.time()*. Seda väärtust hoitakse ujukomaarvuna (*float*), ja seega saab sellega soovi korral teha ka arvutustehteid.

```
import time

sekundid = time.time()
print ("epohhist möödunud sekundeid: ", sekundid)
```

Väljund:

```
epohhist möödunud sekundeid:  1446029217.9237607
```

Aja niimoodi muutujasse salvestamine on hea tehete jaoks, aga ei ole just inimloetav. Inimloetava kuupäeva vormimiseks peab läbima veel paar sammu. Funktsioonide *time.localtime(t)* ja *time.gmtime(t)* abil saab epohhist möödunud sekundite väärtusest genereerida järgmise ehitusega enniku (tuple) (vt tabel 1):

Indeks	Välja nimi	Välja väärtus
0	tm_year	4-kohaline aasta, 0000-9999
1	tm_mon	Kuu number, 1-12
2	tm_mday	Päeva number, 1-31
3	tm_hour	Tunni number, 0-23
4	tm_min	Minuti number, 0-59
5	tm_sec	Sekundi number, 0-61(võimaldab lugeda liigsekundeid)
6	tm_wday	Nädalapäeva number, 0-6(0 on esmaspäev)
7	tm_yday	Aasta päev Juliuse kalendri järgi, 1-366
8	tm_isdst	Kas suve/talveaeg on kasutusel, 0(ei ole), 1(on), -1(käivitamisel automaatselt määratud)

Tabel 1: Ajaenniku struct_time komponendid

Seda ennikut võib täpsemalt määratleda ajastruktuuriks *time.struct_time*. Funktsioon *time.localtime(t)* genereerib enniku arvuti enda ajasätetes määratud ajatsooni järgi, *time.gmtime(t)* teeb seda Greenwichi aja järgi (UTC+0).

```
import time

sekundid = time.time()
aeg = time.localtime(sekundid)
gmaeg = time.gmtime(sekundid)

print ("epohhist möödunud sekundeid: ", sekundid)
print ("Meie ajatsooni ajaennik: ", aeg)
print ("Greenwichi ajatsooni ajaennik: ", gmaeg)
```

Väljund:

```
epohhist möödunud sekundeid:  1446031611.6488638
Meie ajatsooni ajaennik:  time.struct_time(tm_year=2015,
tm_mon=10, tm_mday=28, tm_hour=13, tm_min=26, tm_sec=51,
tm_wday=2, tm_yday=301, tm_isdst=0)
Greenwichi ajatsooni ajaennik:
time.struct_time(tm_year=2015, tm_mon=10, tm_mday=28,
tm_hour=11, tm_min=26, tm_sec=51, tm_wday=2, tm_yday=301,
tm_isdst=0)
```

Tulemus sisaldab nüüd vormistatud infot, aga täieliku inimloetavuse jaoks peab tegema veel ühe sammu. Antud ennikule funktsiooni *time.asctime()* rakendades genereeritakse ajaenniku põhjal inimloetav kuupäev.

```
import time

sekundid = time.time()
aeg = time.localtime(sekundid)
gmaeg = time.gmtime(sekundid)
kuupaev = time.asctime(aeg)
gmkuupaev = time.asctime(gmaeg)

print ("Praegune kuupäev on: ", kuupaev)
print ("Praegune kuupäev UTC järgi on: ", gmkuupaev)
```

Väljund:

```
Praegune kuupäev on:  Wed Oct 28 13:38:23 2015
Praegune kuupäev UTC järgi on:  Wed Oct 28 13:38:23 2015
```

Nagu te võibolla juba märkasite, on see funktsioon küll mugav, kuid sellel on paar olulist puudujääki. Esiteks on tagastatav kuupäev nüüd sõne (*stringi*) kujul, nii et sellega ei saa enam teha relevantseid tehteid. Teiseks ei saa antud funktsiooni puhul määrata, mis kujul kuupäev vormistatakse. Kui on tarvis kuupäeva konkreetsel viisil vormistada, tuleks kasutada funktsiooni *time.strftime(formaat, ajaennik)*, kus formaat on soovitud väljendust kujutav string. Näiteks kui soovime kuvada Euroopas tavalisemat päev.-kuu.-aasta kuupäevaehitust, ja kellaaega pole vaja, siis saaks seda teha järgmise koodiga:

```
import time

sekundid = time.time()
aeg = time.localtime(sekundid)
formaat = "%d.%m.%Y"
vormkpv = time.strftime(formaat,aeg)

print ("Praegune kuupäev on: ", vormkpv)
```

Väljund:

```
Praegune kuupäev on:  28.10.2015
```

Vormistamisstringi saab lisada veel kõiksugu muid argumente, mistahes soovitud kombinatsioonides. Kõik võimalikud argumendid on kirjeldatud järgnevas tabelis (vt tabel 2).

Argument	Tähendus
%a	Nädalapäeva lühendatud nimi
%A	Nädalapäeva täisnimi
%b	Kuu lühendatud nimi
%B	Kuu täisnimi
%c	Arvuti kellasätete järgi kohandatud kuupäeva ja aja väljendus(sama, mille genereerib time.asctime())
%d	Päev kuus täisarvu kujul (01-31)
%H	Tund täisarvu kujul (24 tunnise kella järgi) (00-23)
%I	Tund täisarvu kujul (12 tunnise kella järgi) (01-12)
%j	Päev aastas täisarvu kujul (001-366)
%m	Kuu täisarvu kujul (01-12)
%M	Minut täisarvu kujul (00-59)
%p	Arvuti kellasätetele vastav AM/PM indikaator
%S	Sekund täisarvu kujul (0-61)
%U	Nädala number aastas (Pühapäev loetakse nädala alguseks) (00-53)
%w	Nädalapäev täisarvuna (0(pühapäev)-6(laupäev))
%W	Nädala number aastas (Esmaspäev loetakse nädala alguseks) (00-53)
%x	Arvuti kellasätete järgi kohandatud kuupäeva väljendus(üks pool %c-st)
%X	Arvuti kellasätete järgi kohandatud aja väljendus(teine pool %c-st)
%y	Aasta number ilma sajandita täisarvu kujul (00-99)
%Y	Aasta number koos sajandiga täisarvu kujul (0000-9999)
%z	Ajatsooni muutefaktor UTC suhtes (-23:59 - +23:59) See argument on mittesoovitatav (*deprecated*), ja tuleviku versioonides ei pruugi seda enam leiduda.
%Z	Ajatsooni nimi (Jääb tühjaks kui ajatsoon pole määratud)
%%	Lihtsalt sümbol „%” (Juhuks kui seda peaks olema vaja kuupäeva väljatrükis kasutada)

Tabel 2: Kuupäeva vormistusstringi võimalikud argumendid

2.2.2. Harjutused

Arenda programmi edasi nii, et see trükiks käivitamisel eraldi ridades välja:

- Praeguse kuu ja nädalapäeva nime (tekstina)
- Praeguse kellaaja 12 tunnises süsteemis, sekundi täpsusega
- Praeguse nädala numbri aastas eeldusel, et pühapäev on nädala algus

2.2.3. Aja käsitsi sisestamine ja tehted

Ajaga tehete tegemiseks on oluline lisaks võimalusele aega automaatselt initsialiseerida süsteemi kella järgi ka võimalus sisestada ise kuupäevi, millega võrdlusi teostada. Kõige mõistlikum viis selleks on kasutada funktsiooni *time.strptime(string, formaat)*. Tegemist on põhimõtteliselt tagurpidi *strftime()* funktsiooniga, mis toodab stringikujulise kuupäeva, ja seda kirjeldava vormistusstringi põhjal *struct_time* ajaenniku.

```
import time
vormkpv = "16.03.1994"
formaat = "%d.%m.%Y"
kuupaev = time.strptime(vormkpv, formaat)

print ("Algne kuupäev oli selline: ", vormkpv)
print ("Ennik näeb välja selline: ", kuupaev)
```

Väljund:

```
Algne kuupäev oli selline:  16.03.1994
Ennik näeb välja selline:  time.struct_time(tm_year=1994,
tm_mon=3, tm_mday=16, tm_hour=0, tm_min=0, tm_sec=0,
tm_wday=2, tm_yday=75, tm_isdst=-1)
```

Nagu märgata võib, siis enniku loomiseks kasutatakse ainult stringist välja loetud andmeid, ja mitte esitatud andmed kas arvutatakse automaatselt (loogiliselt tuletatavad asjad, antud näite puhul nädalapäev, päeva number aastas), või siis sisestatakse nullina (antud näite puhul kellaaeg). Ajaenniku saab edasi taandada epohhist möödunud sekunditeks kasutades *time.mktime()* funktsiooni, mis on põhimõtteliselt tagurpidi töötav *localtime()/gmtime()* funktsioon. Siinjuures tasub mainida, et kui üritada *mktime()* funktsiooni rakendada kuupäevale, mis oli enne epohhi, siis tagastab programm veateate piiri ületanud väärtuse kohta (*Overflow error*), kuna epohhist möödunud sekundite andmeväli pole mõeldud töötama negatiivsete arvudega. Epohhist varasemate kuupäevadega arvutuste tegemiseks on kasulik järgmises peatükis käsitletud datetime teek.

```
import time
vormkpv = "16.03.1994"
formaat = "%d.%m.%Y"
kuupaev = time.strptime(vormkpv, formaat)
aeg = time.mktime(kuupaev)

print ("Algne kuupäev oli selline: ", vormkpv)
print ("Ennik näeb välja selline: ", kuupaev)
print ("Epohhist oli sellel kuupäeval möödunud: ", aeg, "
sekundit")
```

Väljund:

```
Algne kuupäev oli selline:  16.03.1994
Ennik näeb välja selline:  time.struct_time(tm_year=1994,
tm_mon=3, tm_mday=16, tm_hour=0, tm_min=0, tm_sec=0,
tm_wday=2, tm_yday=75, tm_isdst=-1)
Epohhist oli sellel kuupäeval möödunud:  763768800.0
sekundit
```

Kui meil on olemas soovitud kuupäeva sekundiline väärtus, saame sellega kasvõi käsitsi lihtsamaid tehteid teha, näiteks kaks kuupäeva teineteisest lahutades saame nende vahele jääva sekundite arvu. Sekundite arvu võime seejärel paari kavala jagamise abil teisendada aastateks/kuudeks/päevadeks. Antud lähenemine ei anna päriselt täpset tulemust, kuna ei arvesta liigaastatega. Meetodeid ajavahede täpsemaks ja lihtsamaks arvutamiseks uurime lähemalt järgmises peatükis, mis käsitleb *datetime* teeki.

```
import time
vormkpv = "16.03.1994"
formaat = "%d.%m.%Y"
kuupaev = time.strptime(vormkpv, formaat)
aeg = time.mktime(kuupaev)
aeg2 = time.time()
ajavahe = int(aeg2 - aeg)
sekundid = ajavahe % 60
minutid1 = ajavahe // 60
minutid = minutid1 % 60
tunnid1 = minutid1 // 60
tunnid = tunnid1 % 24
paevad1 = tunnid1 // 24
paevad = paevad1 % 365
aastad = paevad1 //365

print ("Algne kuupäev oli selline: ", vormkpv)
print ("Epohhist oli sellel kuupäeval möödunud: ", aeg, "
sekundit")
print ("Epohhist on praeguseks möödunud: ", aeg2, "
sekundit")
print ("See kuupäev oli ", ajavahe, "sekundit tagasi")
print ("Ehk siis ", aastad, " aastat, ", paevad, " päeva, ")
print (tunnid," tundi, ",minutid, " minutit, ja " ,sekundid,
" sekundit tagasi")
```

Väljund:

```
Algne kuupäev oli selline:  16.03.1994
```

```
Epohhist oli sellel kuupäeval möödunud:  763768800.0
sekundit
Epohhist on praeguseks möödunud:  1446046947.2592518
sekundit
See kuupäev oli oli  682278147 sekundit tagasi
Ehk siis  21  aastat,  231  päeva,
17  tundi,  42  minutit, ja  27  sekundit tagasi
```

Praktilise edasiarendusena juba õpitu põhjal võime konstrueerida näiteks lihtsa äratuskella. Selleks on kasulik *sleep()* funktsioon, millega olete tõenäoliselt juba enne kokku puutunud. Sleep võimaldab oma programmi soovitud ajaks „magama panna": programm ei kasuta nii tühjalt ressursse, ja jätkab operatsioonide täitmist pärast aja möödumist. Pythoni sleep funktsioon on programmeerimiskeelte seas kohati omapärane, sest kasutab sisendina sekundeid. Enamik teisi keeli kasutab millisekundeid.

```
import time

print ("Sisesta alarmi kuupäev ja kellaaeg kujul DD.MM.YYYY
HH:MM")
alarmiaeg = input()
alarmikpv = time.strptime(alarmiaeg, "%d.%m.%Y %H:%M")
alarmiaegsek = time.mktime(alarmikpv)
praeguneaeg = time.time()
print("Alarm paigas! Vajuta Ctrl+C et tühistada...")
time.sleep(int(alarmiaegsek - praeguneaeg))
print("ALARM!")
```

Väljund:

```
Sisesta alarmi kuupäev ja kellaaeg kujul DD.MM.YYYY HH:MM
28.10.2015 18:06
Alarm paigas! Vajuta Ctrl+C et tühistada...
ALARM!
```

2.2.4. Harjutused

- Lisa äratuskellale veakontroll, mis tuvastab et aeg on ikka tulevikus, ja et on mõtet oodata
- Kirjuta ise (näiteks ajavahemikku arvutava programmi põhjal) programm, mis lubab kasutajal sisestada kaks kuupäeva, ja arvutab, kumb oli varem, ning palju on nende vahel aega.
- *Kirjuta ise(näiteks ajavahemikku arvutava programmi põhjal) programm, mis arvutab käivitamisel, kui kaua on mõne sinu jaoks tähtsa (fikseeritud kuupäevanumbriga) päevani sellel aastal aega, või kas see on juba möödas. (Vihjeks: Pythonis saab stringe liita)

2.2.5. Teised time-teegi funktsioonid

Järgneb kompaktne loetelu koos kirjeldustega teistest märkimisväärsetest time-teegi funktsioonidest, mida me eelnevates näidetes ei käsitlenud, kuid mis võivad spetsiifilistes olukordades kasulikud olla.

time.CLOCK_ - funktsioonid:

See, ja sellega sarnased funktsioonid tagastavad ajamõõtmisandmeid spetsiifilistelt arvutisüsteemi komponentidelt. Põhiliselt on need funktsioonid mõeldud arvuti jõudlustestimiseks (kui kaua mingi protsess aega võtab, jne.), sest on veidi täpsemad, kuid samas väga kontekstitundlikud (erinevates operatsioonisüsteemides tuleb neid erinevalt välja kutsuda), ja neid tuleks seega alati kasutada dokumentatsiooni abiga.

time.ctime(s):

Teisendab sekundeid-pärast-epohhi väärtuse otse kohalikus formaadis kuupäevaks. Ilma argumendita käivitamise korral kasutab arvuti kohalikku aega. Lihtne, aga ei võimalda käsitsi vormistamist.

time.monotonic():

Tagastab süsteemi monotoonkella väärtuse: seda kella ei mõjuta tavalise süsteemikella muutmine. See tähendab, et ta on alati iseenda suhtes „õige”, samas kui tavaline kell võib teoreetiliselt programmi toimimise ajal välismõjude tõttu sattuda olukorda, kus ta on näiteks ühel kontrollimise hetkel vähem, kui eelmisel.

time.tzname:

Tagastab kahest stringist koosneva enniku: kohaliku ajatsooni nimi, ja kohaliku suve/talveaja ajatsooni nimi. Lihtne viis kontrollimaks, kas antud arvuti kell kasutab suve/talveaega.

Põhjaliku ülevaate kõigist time-teegi funktsioonidest võib leida Pythoni ametlikust time-teegi dokumentatsioonist, mis on veebis saadaval aadressil https://docs.python.org/3.5/library/time.html (Inglise keeles).

2.3. Datetime teek

Datetime teek on ehituselt objektorienteeritud, mis tähendab, et siin ei ole aja muutujad enam lihtsalt stringid, ennikud või arvud, vaid objektid, mis sisaldavad mitut atribuuti, ja millel on oma sisseehitatud meetodid. Osad siinsed asjad on otstarbelt time teegis leiduvatele sarnased, kuid lihtsustatud (näiteks *date* ja *time* objektid, mis töötavad idealiseeritud ajas, kus oli ja on alati sama kalender, ja liigsekundeid/aastaid ei esine), teised on aga täiesti erinevad (näiteks *timedelta* funktsioonid). Üldiselt keskendume siin just funktsionaalsustele, mida time teegis ei olnud.

2.3.1. Initsialiseerimine ja defineerimine

Samamoodi nagu eelmises peatükis on siin kasutatud üldist importi, nii et objektide ja meetodite ette tuleb antud näidetes ikka lisada teegi nimi, „datetime.”. Kuupäeva omistamine muutujale datetime teegiga käib üsna oodatule sarnaselt. Tasub lihtsalt meeles pidada, et nii ei teki enam lihtsalt ennik, vaid objekt. Praeguse süsteemikuupäeva saab kätte meetodiga *today()* või *now().* (Now on erinev, sest sellele saab lisaks lisada argumendina ajatsooni, mille praegust aega soovitakse.) Olenevalt sellest, mis klassi kasutada, saab nii kätte kas kuupäeva (kui kutsuda välja objekt date) või kuupäeva ja kellaaja (kui kutsuda välja objekt datetime). Üksinda kellaaja objekti ei saa üksi luua, ja see tuleb kellaaja ja kuupäeva ühisobjektist luua.

```
import datetime

# muutuja = teeginimi.objektinimi.meetod(argumendid)
kpv = datetime.date.today()
kpvkell = datetime.datetime.today()
kell = kpvkell.time()

print(kpv)
print(kell)
print(kpvkell)
```

Väljund:

```
2015-10-28
23:02:27.625983
2015-10-28 23:02:27.625983
```

Põhilised datetime objektid koosnevad ehituselt järgmises tabelis toodud atribuutidest (ja selles järjekorras tuleks sisestada ka argumendid, kui objekti käsitsi initsialiseerida) (vt tabel 3).

Atribuut	Sisu	Leidub objektides
year	Aasta väärtus, arv süsteemi poolt määratud lubatud vahemikus.	datetime, date
month	Kuu väärtus, 1-12	datetime, date
day	Päeva väärtus, 1-(antud kuu maksimaalne päev)	datetime, date
hour	Tunni väärtus, 0-23	datetime, time
minute	Minuti väärtus, 0-59	datetime, time
second	Sekundi väärtus, 0-59	datetime, time
microsecond	Mikrosekundi väärtus, 0-999999	datetime, time
tzinfo	Ajatsooniinfo alamobjekt. Võib jätta tühjaks.	datetime

Tabel3: Datetime objektide ehitus

Neid välju võib objektist ka spetsiifiliselt välja küsida. Näiteks nii:

```
import datetime
kpvkell = datetime.datetime.today()
print("Kell on ", kpvkell.hour, ":", kpvkell.minute)
```

Väljund:

```
Kell on  23 : 59
```

Sarnaselt time teegiga võib datetime objekti käsitsi loomiseks initsialiseerida kas epohhist möödunud sekundite järgi, kasutades *fromtimestamp(sekundid)* funktsiooni, või initsialiseerida stringi + vormistusstringi järgi, kasutades *strptime(string, formaat)* funktsiooni (vormistusstringis kehtivad samad argumendid, mis time teegi versiooniski). Erinevalt time teegist saab datetime teegi objekte ilma pika ümberteisendamiseta ka täiesti käsitsi luua ja muuta. Loomiseks saab lihtsalt datetime objekti initsialiseerida koos argumentidega (ainult aasta, kuu ja päev on kohustuslikud), muutmiseks saab kasutada *replace(omadus=uus väärtus)* meetodit.

```
import datetime

kpvkell = datetime.datetime.strptime("08.08.2010","%d.%m.%Y")
print(kpvkell)
kpvkell2 =   datetime.datetime(2010, 10, 10)
print(kpvkell2)
kpvkell3 = kpvkell2.replace(year=2014)
print(kpvkell3)
```

Väljund:

```
2010-08-08 00:00:00
2010-10-10 00:00:00
```

```
2014-10-10 00:00:00
```

2.3.2. Timedelta klass

Timedelta funktsioonid võimaldavad kalkuleerida täpselt aegade erinevusi, ja nad on üks põhilisi datetime teegi tugevusi. Timedelta muutujasse saab salvestada mingi ajahulga väärtuse, ning seejärel selle ja mõne päris ajaobjektiga (või ka teise ajahulgaga teises timedelta objektis) tehteid teha. Timedelta abil on ajaarvutusi märkimisväärselt mugavam teha, kui eelmises peatükis proovitud käsitsimeetodil, ja üldiselt on ka tulemused lihtsamini väljendatavad. Näiteks võime võtta kuupäeva, ja sellele ühe päeva liites saame kuupäeva homme samal ajal, päeva lahutades aga kuupäeva eile samal ajal.

```
import datetime

kpvkell = datetime.datetime.today()
ykspaev = datetime.timedelta(days=1)

homme = kpvkell + ykspaev
eile = kpvkell - ykspaev

print("Täna:", kpvkell)
print("Eile:", eile)
print("Homme:", homme)
```

Väljund:

```
Täna: 2015-10-28 23:28:24.202282
Eile: 2015-10-27 23:28:24.202282
Homme: 2015-10-29 23:28:24.202282
```

Samuti saab timedelta objekti luua, kui lahutada teineteisest kaks kuupäevaobjekti:

```
import datetime

kpvkell = datetime.datetime.today()
minukpv = datetime.datetime(1994, 3, 16)
vahe = kpvkell - minukpv

print(vahe)
```

Väljund:

```
7897 days, 0:30:17.412106
```

Timedelta objektile saab loomisel rakendada mitut argumenti, kuid kogu info hoitakse kolmes atribuudis (days, seconds, microseconds) (vt tabel 4).

Argument	Olemus
days	Päevade arv. Hoitakse objektis.
seconds	Sekundite arv. Hoitakse objektis.
microseconds	Mikrosekundite arv. Hoitakse objektis.
milliseconds	Millisekundite arv. Teisendatakse ümber mikrosekunditeks.
minutes	Minutite arv. Teisendatakse ümber sekunditeks.
hours	Tundide arv. Teisendatakse ümber sekunditeks.
weeks	Nädalate arv. Teisendatakse ümber päevadeks.

Tabel 4: Timedelta objekti argumentide olemused

Seega on timedelta objekti loomisel näiteks argumendid *days=7* ja *weeks=1* olemuselt samad.

Timedelta objekti väärtuse saab vajadusel teisendada täielikult sekunditeks, kui kasutada *.total_seconds()* meetodit.

```
import datetime
kpvkell = datetime.datetime.today()
minukpv = datetime.datetime(1994, 3, 16)
vahe = kpvkell - minukpv
print(vahe)
print(vahe.total_seconds())
```

Väljund:

```
7897 days, 0:42:59.571397
682303379.571397
```

2.3.3. Harjutused

- Kirjuta datetime teeki kasutades programm, mis lubab sisestada kahe inimese sünnipäevad, ja arvutab, kumb neist on vanem.
- Kirjuta datetime teeki kasutades programm, mis arvutab käivitamisel välja, mis numbriga kuupäev on kasutaja sisestatud arvu nädalate pärast.

Datetime teegi muude, enamasti meie pool time teegi juures juba ekvivalentselt käsitletud, funktsioonide kohta võib lisaks lugeda pythoni dokumentatsioonist aadressil https://docs.python.org/3.5/library/datetime.html .

2.4. Teised teegid

Siin on lühikirjeldused teistest ajaarvamisega seonduvatest teekidest, millega see õppematerjal ei tegelenud.

Calendar teek

Kui teised seni käsitletud teegid täitsid üldjoontes sarnaseid ülesandeid, siis calendar teek on palju spetsiifilisem: Selle üks ja ainus ülesanne on vormistada ja trükkida välja kalendreid. Antud teek on ehituselt väga lihtne, nii et selle kohta pole siin materjalis midagi tähendusrikast öelda.

Dateutil teek

Kolmanda osapoole teek, mis laiendab datetime teegi funktsioone, näiteks suhteliste timedeltade (mis aksepteerivad lisaks kindlatele väärtustele sisendina ka abstraktsemaid väärtusi, näiteks „aega kuu viimase päevani”) ja ajatsooniarvutuste vallas. Oli selle materjali jaoks veidi liialt spetsiifiline. Saadaval aadressil http://labix.org/python-dateutil .

Kokkuvõte

Ja selline ta siis oligi! Üldiselt võiksin selle töö kohta öelda, et kuigi selle kirjutamine osutus oodatust vaevalisemaks, nautisin ma seda protsessi siiski. Kõige raskem asi töö juures oli vast otsustamine, mis vajaks pikalt seletamist, ja mis mitte. Lõpuks otsustasin vaid kahe teegi pikemalt tutvustamise kasuks, kuna need (time, ja datetime) on Pythoni ajaarvamise põhiliseks aluseks, ja teised sellega seonduvad teegid vajavad efektiivseks toimimiseks suuremal või väiksemal määral nende tundmist. Teistest teekidest, spetsiifikast ja nüanssidest oleks teoreetiliselt võinud kirjutada veel vähemalt sama palju. Selles mõttes olen ma rahul, et suutsin õppematerjali juures säilitada vähemalt mingisuguse fookuse. Töö pidi lõpuks olema sissejuhatava kallakuga õppematerjal ajaarvamisest Python keele abil, ja see ta ka on. Ei midagi enamat ega vähemat. Ülesanded tulid võibolla natuke lihtsad, kuid alati tasub meeles pidada, et programmerimise mõte ongi asjad võimalikult lihtsaks teha. Selliste lihtsamate „klotside” kokku liitmisel, mis nende ülesannete käigus valmivad, ehitataksegi suuremad ja keerulisemad programmid. Jäin oma saavutusega rahule.

Kasutatud kirjandus

Python Software Foundation, (2015). *16.3. time — Time access and conversions.* Loetud aadressil https://docs.python.org/3.5/library/time.html

Python Software Foundation, (2015). *8.1. datetime — Basic date and time types.* Loetud aadressil https://docs.python.org/3.5/library/datetime.html

Time zone. (kuupäev puudub). Wikipedia. Loetud 29. oktoober 2015 aadressil https://en.wikipedia.org/wiki/Time_zone

Lisad

Ülesannete võimalikud lahendused

Kuupäeva formaadid

```
import time;

sekundid = time.time()
aeg = time.localtime(sekundid)
formaat1 = "%d.%m.%Y"
formaat2 = "%B, %A" #Kuupäev ja päev tähtedes
formaat3 = "%I:%M:%S" #Tund minut sekund, 12 tunni süsteemis
formaat4 = "%U" #Nädala number, pühapäev on nädala algus
vormkpv1 = time.strftime(formaat1,aeg)
vormkpv2 = time.strftime(formaat2,aeg)
vormkpv3 = time.strftime(formaat3,aeg)
vormkpv4 = time.strftime(formaat4,aeg)
```

Veakontrolliga äratuskell

```
import time;

print ("Sisesta alarmi kuupäev ja kellaaeg kujul DD.MM.YYYY
HH:MM")
alarmiaeg = input()
alarmikpv = time.strptime(alarmiaeg, "%d.%m.%Y %H:%M")
alarmiaegsek = time.mktime(alarmikpv)
praeguneaeg = time.time()
if alarmiaegsek-praeguneaeg <= 0:
    print("See ajahetk on juba möödas!")
else:
    print("Alarm paigas! Vajuta Ctrl+C et tühistada...")
    time.sleep(int(alarmiaegsek - praeguneaeg))
    print("ALARM!")
```

Aegade võrdleja

```
import time;

print ("Palun sisesta kõik kuupäevad ja kellaajad kujul
DD.MM.YYYY HH:MM")
kpv1 = input("Sisesta esimene kuupäev: ")
```

```
kpv2 = input("Sisesta teine kuupäev: ")
kpv1s = time.mktime(time.strptime(kpv1, "%d.%m.%Y %H:%M"))
kpv2s = time.mktime(time.strptime(kpv2, "%d.%m.%Y %H:%M"))
if kpv1s == kpv2s:
    print("Sisestati täpselt samad kuupäevad!")
elif kpv1s < kpv2s:
    small = kpv1s
    large = kpv2s
    print ("Esimene kuupäev on väiksem ")
else:
    small = kpv2s
    large = kpv1s
    print ("Teine kuupäev on väiksem ")

vahe = large - small
sekundid = vahe % 60
minutid1 = vahe // 60
minutid = minutid1 % 60
tunnid1 = minutid1 // 60
tunnid = tunnid1 % 24
paevad1 = tunnid1 // 24
paevad = paevad1 % 365
aastad = paevad1 //365
print (aastad, " aasta, ", paevad, " päeva, ")
print (tunnid, " tunni, ", minutid, " minuti, ja ", sekundid,
" sekundi võrra")
```

Tähtpäeva kontrollija

```
import time;

minukpv = "16.03."
aeg = time.time()
ennik = time.localtime(aeg)
praeguneaasta = str(ennik[0])
otsitavkpv = minukpv + praeguneaasta
otsitavsekund = time.mktime(time.strptime(otsitavkpv, "%d.%m.
%Y"))
ajavahe = int(otsitavsekund - aeg)
if ajavahe < 0:
    print("Sünnipäev oli sellel aastal juba ära...")
elif(ajavahe == 0):
    print("Palju õnne!")
else:
```

```
    sekundid = ajavahe % 60
    minutid1 = ajavahe // 60
    minutid = minutid1 % 60
    tunnid1 = minutid1 // 60
    tunnid = tunnid1 % 24
    paevad1 = tunnid1 // 24
    paevad = paevad1 % 365
    print("Sünnipäevani on jäänud ", paevad, " päeva, ",
tunnid, " tundi, ", minutid, " minutit, ja ", sekundid, "
sekundit!")
```

Datetime vanuste võrdleja

```
import datetime;

print ("Palun sisesta kõik kuupäevad kujul DD.MM.YYYY")
kpv1 = input("Sisesta esimene sünnipäev: ")
kpv2 = input("Sisesta teine sünnipäev: ")
kpv1o = datetime.datetime.strptime(kpv1, "%d.%m.%Y")
kpv2o = datetime.datetime.strptime(kpv2, "%d.%m.%Y")

if kpv1o == kpv2o:
    print("Inimesed on sama vanad!")
elif kpv2o > kpv1o:
    print ("Teine inimene on vanem ")
else:
    print ("Esimene inimene on vanem ")

print (kpv1o - kpv2o, " võrra")
```

Datetime nädalakalkulaator

```
import datetime;

ndl = int(input("Sisesta nädalate arv: "))
vahe = datetime.timedelta(weeks=ndl)
kpv = datetime.datetime.today();
kpv2 = kpv + vahe

print (ndl, "nädala pärast on ", kpv2.day, "kuupäev")
```