# LUT Python ohjelmointiopas 2023 - osa 8
# Sisällysluettelo

# Luku 8: Kirjastot ja moduulit

Funktioista puhuttaessa näimme, kuinka pystymme käyttämään uudelleen aikaisemmin muualla kirjoitettua koodia kuten `print`, `input` ja `len` -funktioita. Entäpä jos haluaisimme käyttää aikaisemmin tekemäämme funktiota jossain toisessa ohjelmassa eli tiedostossa? Pythonissa tämä voidaan toteuttaa käyttämällä moduuleja. Moduuli on periaatteessa lähdekooditiedosto, joka sisältää funktioita, joita voidaan tuoda käytettäväksi sisällyttämisen (eng. `import`) avulla. Todellisuudessa olemme jo aiemmin tehneet moduuleja, sillä kaikki aliohjelmia sisältävät Python-tiedostot voivat toimia toisissa ohjelmissa moduuleina. Lisäksi Pythonin mukana tulee perusfunktioiden lisäksi suuri joukko moduuleja, jotka mahdollistavat erilaisia toimintoja. Ohjelmointiympäristön mukana toimitettavista lisätoimintoja tarjoavista moduuleista käytetään usein nimitystä kirjastomoduuli (erityisesti Python) tai funktiokirjasto (C, C++).

## *Kirjastojen käyttö*

Kirjastomoduulin käyttö alkaa sisällyttämällä se ohjelmaan `import`-komennolla. Sisällyttämisen jälkeen moduulin funktioita voidaan käyttää kuten normaaleja lähdekoodin sisäisiä funktioita.

Esimerkissä 8.1 näkyy kirjastojen peruskäyttö. Osassa 1 käytetään `math`-kirjastossa määriteltyä kiintoarvoa `pi` ja funktiota `pow`, jotka ovat piin likiarvo sekä potenssiin korotusfunktio normaalien ympyrä-laskujen mukaisesti. Osassa 2 ympyrän säde arvotaan `random`-kirjaston funktiolla `randint`, joka palauttaa kokonaisluvun väliltä 0 ja 30 tai jommankumman raja-arvoista. Osassa 3 tulostetaan kellonaika 3 kertaa toistorakenteessa pitäen tulosteiden välissä aina 6 sekunnin tauko.

**Esimerkki 8.1. Kirjastojen peruskäyttö**

```
# Osa 1. math-kirjaston käyttö
import math
print("math-kirjaston kiintoarvon ja funktion käyttö:")
Sade = 5
Keha = 2 * math.pi * Sade
Ala = math.pi * math.pow(Sade, 2)
print("Säde {0:d}, kehä {1:.2f} ja pinta-ala {2:.2f}.\n".format(Sade, Keha, Ala))

# Osa 2. random-kirjaston käyttö
import math
import random
print("Luvun arpominen random-kirjaston randint-funktiolla väliltä 0-30:")
Sade = random.randint(0,30)
Keha = 2 * math.pi * Sade
Ala = math.pi * math.pow(Sade, 2)
print("Säde {0:d}, kehä {1:.2f} ja pinta-ala {2:.2f}.\n".format(Sade, Keha, Ala))

# Osa 3. time-kirjaston käyttö
import time
print("Kelloajan tulostus 6 sekunnin välein:")
for i in range(3):
    Nyt = time.localtime()                  # Aika sisäisessä esitysmuodossa
    Kello = time.strftime("%H:%M:%S", Nyt)  # Aika merkkijonoksi
    print("Kello on " + Kello)
    time.sleep(6)
```

### *Tuloste*

```
math-kirjaston kiintoarvon ja funktion käyttö:
Säde 5, kehä 31.42 ja pinta-ala 78.54.

Luvun arpominen random-kirjaston randint-funktiolla väliltä 0-30:
Säde 7, kehä 43.98 ja pinta-ala 153.94.

Kelloajan tulostus 6 sekunnin välein:
Kello on 10:50:16
Kello on 10:50:22
Kello on 10:50:28
```

Kaikki esimerkin 3 osaa ovat itsenäisesti toimivia osia, mikä näkyy siinä, että jokainen osa sisällyttää kaikki tarvitsemansa kirjastot osan alussa `import`-käskyillä. Käytännössä tämä ilmoittaa tulkille, että tulemme käyttämään nimettyä moduulia ohjelmassa. Viitattaessa moduulissa määriteltyihin rakenteisiin tulee viittaus aloittaa `kirjastonimi.` -etuliitteellä eli kirjastonimen jälkeen tulee piste ja haluttu funktio tai vakio. Pistenotaatiota tarvitaan, jotta tulkki ymmärtää hakea ne kirjastosta eli osaa etsiä tunnuksia kirjaston nimiavaruudesta. Tulkki etsii `import`-komennossa annettua kirjastoa ensin samasta kansiosta lähdekooditiedoston kanssa ja sen jälkeen tulkille erikseen määritellyistä kansioista. Nämä määritellään asennuksen yhteydessä, joten standardikirjastojen osalta asia on kunnossa.

Kirjastojen nimi paljastaa sen sisällä olevat rakenteet tai ainakin pyrkii antamaan selkeän vihjeen niistä. `math`-moduuli sisältää matematiikkaan liittyviä rakenteita eli vakioita ja aliohjelmia; `random`-moduuli sisältää satunnaislukuihin liittyvät rakenteet ja `time` moduuli sisältään aikaan liittyviä rakenteita.

Tarpeeseen sopivia kirjastoja ja rakenteita etsitään dokumenttien yms. avulla, kunnes ne oppii muistamaan aktiivisen käytön myötä. Esimerkin 8.1 `math`-kirjasto, `pi`-vakio ja `pow`-funktio eivät ole suuria yllätyksiä, ja monet oppivat käyttämään niitä kuultuaan ne pari kertaa. Satunnaisluvut ovat usein vieraampia ja etenkin pseudosatunnaisluvut vaativat usein ajatuksen kanssa asiaan paneutumista, ennen kuin ne tuntuvat luontevilta käyttää. Osan 3 `sleep` eli odotusfunktio on varsin selkeä eli parametrina annettavat sekunnit määräävät odotettavan ajan. Myös ajan käsittely näyttää selkeältä eli `time`-moduulissa on `localtime`-funktio systeemikellon ajan hakemiseen, ja tämä aika voidaan muotoilla `strftime`-funktiolla haluttuun merkkijonomuotoon tulostettavaksi näytölle. Ajan käsittely tietokoneessa on laaja konsepti ja palaamme siihen myöhemmin tässä luvussa. Tässä vaiheessa oleellista on hahmottaa kirjastojen peruskäyttö sisältäen seuraavat asiat:

1. Kirjaston sisällyttäminen ohjelman alkuun `import` käskyllä, esim. `import math`
2. Kirjaston vakioiden ja funktioiden käyttö pistenotaatiolla, esim. `math.pi` ja `math.pow(2, 3)`

## *Omien moduulien tekeminen ja käyttäminen*

Omien moduulien tekeminen on helppoa ja olemme itse asiassa tehneet Python-moduuleja koko ajan tässä oppaassa. Syy tähän on se, että *moduuli* tarkoittaa tässä kohdin Python-tiedostoa eli kaikki tähän asti tekemämme .py-päätteiset tiedostot ovat moduuleja. Näin ollen moduulien teko onnistuu ja voimme keskittyä useista moduuleista, eli tiedostoista, koostuvan ohjelman toimintaan ja ymmärrettävyyteen. Keskitymme tässä oppaassa kahdesta itse tehdystä moduulista muodostuviin ohjelmiin. Yksinkertaisuuden vuoksi toinen moduuli sisältää pääohjelman ja toinen aliohjelmat eli toimii kirjastona. Pääohjelman sisältävän tiedoston nimi noudattaa kurssin tyyliohjetta eli nimetään samalla tavalla kuin aiemminkin ja toisen tiedoston nimi on sama lisättynä Kirjasto-tarkenteella.

Esimerkissä 8.2 on itse tehty Python kirjasto-moduuli `L08E2Kirjasto.py`. Koska tiedoston nimen pitää olla täsmälleen oikea, toistuu se tässä esimerkissä monta kertaa eli ole tarkkana tiedostonimien kanssa. Itse tiedostossa on kaksi kiintoarvoa ja kolme aliohjelmaa, joten tämä tiedosto näyttää normaalilta tähän asti tehdyltä Python-ohjelmalta yhdellä erolla eli kirjastotiedostossa ei ole pääohjelmaa eikä kutsua siihen. Tiedostossa on vain aliohjelmia ja niihin liittyviä kiintoarvoja (ja luokkia tarvittaessa) ja jos tämän tiedoston suorittaa, mitään ei tapahdu, koska päätasolla/sisentämättä ei ole yhtään suoritettavaa käskyä. Huomaa myös, että kaikki tiedonsiirto aliohjelmiin tapahtuu parametreilla ja paluuarvolla. Olemme tähän asti korostaneet parametrien ja paluuarvojen tärkeyttä tiedonsiirrossa, sillä ohjelman jakautuessa useisiin tiedostoihin ne ovat luonnollinen tapa siirtää tietoa tiedostojen välillä. Tiivistetysti kirjaston teko onnistuu aiemmin annetuilla ohjeilla jättämällä pääohjelma pois.

**Esimerkki 8.2. Oma kirjasto, tallennettu nimellä L08E2Kirjasto.py**

```
# Kirjastomoduuli L08E2Kirjasto.py
VERSIO = 1.0
MM_TO_INCH = 25.4

def tulosta(Teksti):
    print(Teksti)
    return None

def summa(Luku1, Luku2):
    Tulos = Luku1 + Luku2
    return Tulos

def mmTuumiksi(Millit):
    Tuumat = Millit / MM_TO_INCH
    return Tuumat

#eof
```

Esimerkissä 8.3 on edellä esiteltyä kirjasto-moduulia hyödyntävä pääohjelma tallennettuna tiedostoon `L08E2.py`. Molempien tiedostojen pitää olla samassa kansiossa ohjelmaa suoritettaessa ja koska molemmat tiedostot liittyvät samaan ohjelmaan, ne on nimetty yhteisellä osalla `L08E2`. Suorittaminen alkaa kirjaston sisällyttämisellä ja jatkuu pääohjelmaan, joka kutsuu aliohjelmia aiemmin puhutun mukaisesti sillä erolla, että aliohjelman nimen lisäksi nyt pitää kertoa, mistä moduulista aliohjelma löytyy. Itse tehdyssä kirjastossa olevia aliohjelmia kutsutaan samalla tavoin kuin muitakin kirjastoaliohjelmia eli esimerkin 8.1 mukaisesti aliohjelmakutsu on nyt esim. `L08E2Kirjasto.tulosta(Tuloste)`. Myös kirjaston kiintoarvoja voidaan käyttää esimerkin 8.1 mukaisesti pistenotaatiolla.

**Esimerkki 8.3. Oman kirjaston käyttö, tallennettu nimellä L08E2.py**

```
# Pääohjelma L08E2.py
import L08E2Kirjasto

def paaohjelma():
    Tuloste = "Tässä on tulostettava testiteksti."
    Milleja = 123
    L08E2Kirjasto.tulosta(Tuloste)
    Summa = L08E2Kirjasto.summa(3, 9)
    print("Annettujen lukujen summa on", Summa)
    Tuumia = L08E2Kirjasto.mmTuumiksi(Milleja)
    print("{0:d} mm = {1:.2f} tuumaa muuntokertoimella {2:.2f}.".
            format(Milleja, Tuumia, L08E2Kirjasto.MM_TO_INCH))
    print("Käytetyn kirjaston versio on", L08E2Kirjasto.VERSIO)
    return None

paaohjelma()
#eof
```

### *Tuloste*

```
Tässä on tulostettava testiteksti.
Annettujen lukujen summa on 12
123 mm = 4.84 tuumaa muuntokertoimella 25.40.
Käytetyn kirjaston versio on 1.0
```

Omien kirjastojen teko ja käyttö ei siis ole monimutkaista, kunhan on tarkkana teknisten yksityiskohtien kanssa.

## *Valmiiden kirjastojen käyttö ja käyttöohjeita*

Aloitimme kirjastojen käytön esimerkissä 8.1 ja varsin pian tuli selväksi, että kirjastojen käyttäminen on vaikeaa, jos ei tiedä mitä kirjastoja on olemassa, mitä ne sisältävät ja miten tarjolla olevia aliohjelmia käytetään. Tutustutaan seuraavaksi muutamiin yleisiin kirjastomoduuleihin sekä niiden sisältämiin rakenteisiin, jotta voimme ruveta hyödyntämään niitä ohjelmissamme eikä meidän tarvitse kirjoittaa kaikkea tarvittavaa koodia itse. Ja ennen moduulien esittelyä katsotaan, mistä ja miten ne löytyvät Pythonin perusasennuksessa.

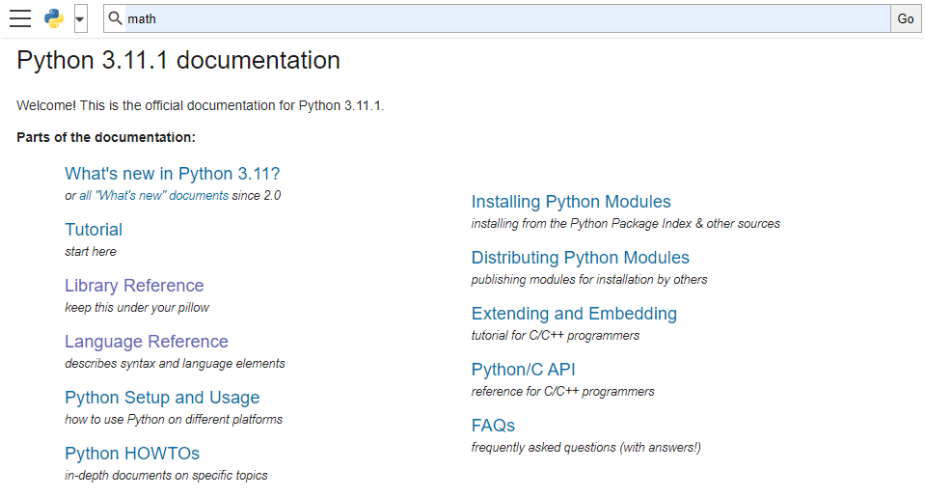
### Python dokumentaatio ja kirjastojen esittelyt

Seuraavissa alakohdissa mainittavat kirjastot ja funktiot riittävät tyypillisesti tähän oppaaseen liittyvissä tehtävissä, mutta käytännössä kirjastojen perustiedot etsitään ohjelmointityökalujen dokumenteista ja sitten kokeillaan, saadaanko haluttu asia tehtyä. Jos löytynyt ohjelma ei vastaa tarvetta, etsimistä voi jatkaa standardikirjastojen ulkopuolelta kyselemällä muilta vinkkejä sopivista kirjastoista jne.

Python versio 3.11.1 Windows-asennus toimii kirjastojen osalta seuraavalla tavalla. Usein asennukset poikkeavat eri järjestelmissä ja versioissa, joten jos joku kohta näyttää erilaiselta, etsimistä kannattaa jatkaa vastaavia avainsanoja ja -konsepteja katsellen.

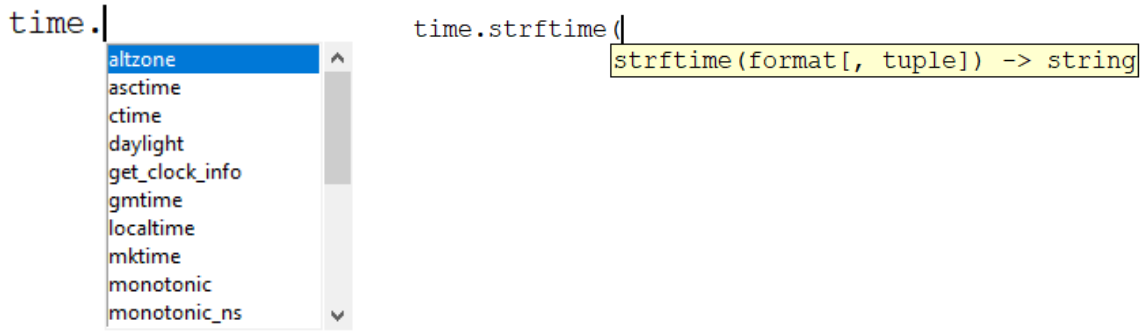
Moduulin dokumentaatioon pääse helpoiten käsiksi IDLE-editorissa laittamalla kursori moduulin nimen, esim. `math`-sanan, päälle ja valitsemalla F1. Tämän jälkeen selain avaa asennuksen yhteydessä tietokoneelle asennetun Python-dokumentaation, esim. Python 3.11.1 documentation, kuvan 8.1 mukaisesti. Näytön yläreunassa näkyy etsi-laatikko, johon voit kirjoittaa haettavan kirjaston nimen, esim. `math`, tai kirjastoja voi etsiä myös Library Reference -osiosta. Tässä tapauksessa Etsi-toiminto löytää 140 sivua, joista ensimmäinen on ”math (Python module, in math — Mathematical functions)”. Tällä sivulla on listattuna noin 60 `math`-kirjaston funktiota ja kiintoarvoa eli tämän jälkeen voi katsoa, löytyykö näistä apua ja mitä parametreja sekä paluuarvoja yms. niihin liittyy.

**Kuva 8.1. math-kirjaston haku Python 3.11.1 dokumentaatiosta**




IDLE-editoria kannattaa hyödyntää kirjastojen kanssa. Kirjoittamalla editoriin moduulin nimen ja pisteen IDLE tuo pudotusvalikon näkyville, josta voi valita käytettävissä olevat funktiot ja attribuutit. Myös ToolTip'iä kannattaa käyttää eli kirjoita moduulin nimi, piste ja funktion nimi sekä aukeava sulku, jonka jälkeen ToolTip näyttää funktion nimen, parametrit ja paluuarvon kuvan 8.2 mukaisesti.

**Kuva 8.2. IDLE-editori ja funktio/attribuuttilista sekä ToolTip**




Kirjastomoduulien käytön lähtökohta on, että etsii itselle höydyllisiä kirjastoja ja funktioita eli tietää, mitä on tarjolla ja missä. Sen jälkeen käytön yksityiskohdat eli parametrit ja paluuarvot sekä niiden järjestyksen ja tyypit voi tarkistaa dokumenteista. Aikaa myöten käytön myötä ne oppii muistamaan ulkoa, mutta käytön yksityiskohdat löytyvät asennuksen mukana tulevista työkaluista eikä niitä tarvitse opetella ulkoa.

## math-moduuli

`math`–moduulin avulla pääsemme käsiksi erilaisiin matemaattisiin funktioihin, joilla voimme laskea mm. trigonometrisia arvoja ja logaritmeja. Kirjasto sisältää myös vakioita kuten pii ja e.

**Esimerkki 8.4. math-moduuli**

```
import math
print(math.cos(math.pi / 4.0))
print(math.log(1024, 2))
```

### *Tuloste*

```
0.7071067811865476
10.0
```

## random-moduuli

`random`-moduuli mahdollistaa satunnaislukujen käytön. Sen avulla voimme ottaa käyttöön mm. satunnaiset valinnat sekä suorittaa arvontoja koneen sisällä. Tässä oppaassa riittää tyypillisesti käyttää `randint`-funktio, joka palauttaa satunnaisen kokonaisluvun annetulta väliltä. Jokaisen funktion yksityiskohdat kannattaa tarkastaa aina dokumenteista, esim. tuleeko satunnaislukujen raja-arvot mukaan, molemmat tai toinen.

**Esimerkki 8.5. random-moduuli**

```
import random

print("random-moduulin generoimia satunnaislukuja ja joukkoja:")
print("Satunnainen kokonaisluku a <= x <= b (a=0, b=12):", random.randint(0, 12))
print("Satunnainen desimaaliluku 0 <= x < 1:", random.random())
print("Satunnainen valinta listalta ['apple', 'pear', 'banana']:",
        random.choice(['apple', 'pear', 'banana']))
print("Satunnainen 10 alkion joukko listana väliltä [0-99]:")
print(random.sample(range(100), 10))
print()

print("Satunnaisluvut ovat näennäisiä eli pseudosatunnaislukuja:")
print("random ilman alustusta 1:", random.randint(0,30))
print("random ilman alustusta 2:", random.randint(0,30))
print("random ilman alustusta 3:", random.randint(0,30))
random.seed(1)
print("random alustettu arvolla 1:", random.randint(0,30))
random.seed(1)
print("random alustettu arvolla 1:", random.randint(0,30))
random.seed(1)
print("random alustettu arvolla 1:", random.randint(0,30))
```

### *Tuloste*

```
random-moduulin generoimia satunnaislukuja ja joukkoja:
Satunnainen kokonaisluku a <= x <= b (a=0, b=12): 7
Satunnainen desimaaliluku 0 <= x < 1: 0.6512680740546052
Satunnainen valinta listalta ['apple', 'pear', 'banana']: apple
Satunnainen 10 alkion joukko listana väliltä 0-100:
[67, 34, 95, 92, 82, 20, 66, 71, 3, 49]

Satunnaisluvut ovat näennäisiä eli perustuvat kelloon:
random ilman alustusta 1: 25
random ilman alustusta 2: 11
random ilman alustusta 3: 25
random alustettu arvolla 1: 4
random alustettu arvolla 1: 4
random alustettu arvolla 1: 4
```

Käytännössä random-kirjaston erilaiset variantit vähentävät tarvetta soveltaa ja tehdä asia monessa eri vaiheessa. Tämän oppaan kannalta satunnaisuus on vain yksi asia ja siksi keskitymme `randint`-funktion käyttöön. Sen lisäksi on syytä huomata `seed`-funktion käyttö, joka antaa satunnaislukugeneraattorille siemenluvun. Käytännössä tämä tarkoittaa sitä, että samalla siemenluvulla ”satunnaisluvut” ovat aina identtisiä eli ne perustuvat samaan matemaattiseen kaavaan Python-dokumentaation mukaisesti. Mikäli siemenlukua ei aseteta, käyttää `random`-kirjasto tyypillisesti sisäisen kellon tuhannesosia siemenlukuna, ja satunnaisluvut ovat käytännössä satunnaisia. Tämän oppaan kannalta satunnaisuus on riittävää, mutta tällä moduulilla ei kannatta lähteä jakamaan rahaa esim. lauantai-illan Lotossa.

## fractions-moduuli

Pythonin matemaattisen taidot eivät rajoitu pelkästään kokonais- ja desimaalilukujen käsittelyyn, vaan mukaan voidaan ottaa murtoluvutkin. Eksakti matemaattinen numeroiden pyörittely vaatii usein murtolukuja ja tällöin Pythonista voi olla paljon apua:

Esimerkissä 8.6 määrittelemme ensin `Murtoluku` -muuttujan arvoksi murtoluvun 1/3. Sen jälkeen tulostamme arvon ja saamme – yllätys yllätys – vastaukseksi murtoluvun 1/3. `Murtoluku` kerrottuna 4:llä on 4/3 ja vielä 3:lla on 3/3, jonka Python automaattisesti supistaa arvoon 1/1 eli 1.

**Esimerkki 8.6. fractions-moduuli**

```
import fractions

print("fractions-moduuli eli murtoluvut:")
Murtoluku = fractions.Fraction(1, 3)
print("Murtolukuesitys Pythonin tapaan:", Murtoluku)
print("Edellinen murtoluku kerrottuna 4:llä:", 4 * Murtoluku)
print("Edellinen murtoluku kerrottuna 3:llä:", 3 * Murtoluku)
```

### *Tuloste*

```
fractions-moduuli eli murtoluvut:
Murtolukuesitys Pythonin tapaan: 1/3
Edellinen murtoluku kerrottuna 4:llä: 4/3
Edellinen murtoluku kerrottuna 3:llä: 1
```

## urllib-moduuli

`urllib` poikkeaa aiemmin katsomistamme moduuleista. Se ei ole pelkkä tietokoneen sisäinen moduuli, vaan se osaa hakea Internetin verkkoresurssien kuten www-sivujen tai uutissyötteiden tietoja omien funktioidensa avulla Internet-protokolliin perustuen.

Esimerkin 8.7 mukaisesti `urllib`-moduulin toimintaidea noudattaa tiedoston lukemista eli avaan, luetaan ja suljetaan. Koska www-sivut noudattavat omia protokollia, pitää sisältöä tyypillisesti muokata ja tällä kertaa keskitytään ääkkösten huomioimiseen. Itse sisällön tulostus onnistuu, mutta IDLE varoittaa, että tuloste on 6449 riviä pitkä ja suosittelee mieluummin kopioimaan sen muualle. Koodieditorille tämä ei ole ongelma, joten sisällön voi liittää tyhjään koodi-ikkunaan ja katsella tulostetta siellä.

**Esimerkki 8.7. urllib-moduuli**

```
import urllib.request
Sivu = urllib.request.urlopen("http://www.lut.fi")
Sisalto = Sivu.read()              # Luetaan sivun sisältö
Sivu.close()
Tekstina = Sisalto.decode("utf-8") # Muunnetaan merkistö oikeanlaiseksi
print(Tekstina)
```

### *Tuloste*

```
[IDLE varoittaa, että 6449 riviä voi hidastaa sen toimintaa.]
```

Tällä muutaman rivin ohjelmalla saa siis haettua halutun verkkosivun sisällön. Tämä ohjelma tarvitsee luonnollisesti oikeuden käyttää verkkoyhteyttä, joka saattaa joissain tapauksissa aiheuttaa hälytyksen palomuurissa. Jos näin käy, tiedät mistä on kysymys ja voit sallia uuden yhteyden. Luonnollisesti, jos ohjelma ei saa yhteyttä verkkoon, ei funktiokaan saa mitään haettua.

Käytännössä www-sivujen sisällön käsittely on rutiiniasia Pythonilla ja se näkyy siinä, että

asia onnistuu muutamalla kirjastosta löytyvällä funktiolla. Näin ollen ohjelmoijan tehtäväksi jää tyypillisesti haetun sisällön analysointi sopivalla tavalla.

## *Aikatiedon käsittely*

Aika on laaja konsepti ja se voi tarkoittaa esimerkiksi päivämääriä ja vuosia, kellonaikoja eri puolilla maapalloa ja aikaa mitattaessa voidaan tarvita tuhannesosien tarkkuutta sekä tietoa monista teknisistä yksityiskohdista. Siksi Python tarjoaa ajan käsittelyyn useita eri kirjastoja, jotka keskittyvät erilaisiin tarpeisiin. Yleisimmin käytetyt aikaan liittyvät kirjastot ovat `time`, `datetime`, `calendar` ja `zoneinfo` sekä `dateutil`, joista tutustumme seuraavaksi kahteen ensimmäiseen eli `time` ja `datetime` -kirjastoihin. `calender` keskittyy yleisiin kalenteri-asioihin, `zoneinfo` aikavyöhykkeisiin ja `dateutil` on erikseen asennettava paketti, joka tarjoaa laajempaa tukea aikavyöhykkeiden ja ajan käsittelyyn. Tutustumme tämän oppaan kannalta keskeisiin ajankäsittelyrutiineihin ja mikäli sinulla on erikoistarpeita aikaan liittyen, kannattaa katsoa muita kirjastoja sekä tarvittaessa jatkaa sopivien kirjastojen etsimistä.

Ajan käsittely poikkeaa jonkin verran eri käyttöjärjestelmissä kuten Windows, Linux ja Mac. Tämä opas on tehty Windows-ympäristössä käyttäen perusrakenteita, joiden pitäisi toimia myös muissa ympäristöissä. Jos alla olevat rakenteet eivät toimi omassa ympäristössä, kannattaa mahdolliset käyttöjärjestelmäerot tarkistaa dokumenteista luvun alussa kerrotulla tavalla. Eri moduuleissa on myös samoja sekä vastaavanlaisia rakenteita ja jäsenfunktioita, jotka toimivat eri tavoin eri moduuleissa. Siksi ongelmatilanteissa kannattaa tarkastaa ensin, että käytettävän moduulin nimi on varmasti sama kuin alla olevissa esimerkeissä. Edelleen, omalla koneelle asennettu Python-dokumentaatio liittyy omaan Python-asennukseen, joten sitä eli F1-helppiä ja ToolTip'iä kannattaa hyödyntää aktiivisesti kirjastojen kanssa.

### time-moduuli ja systeemin aika

Ajankäsittelyn peruskirjasto on `time` ja keskitymme tässä oppaassa siihen. Muissa moduuleissa on samantyylisiä toimintoja, jäsenfunktioita ja -muuttujia, joten mahdollisissa ongelmatilanteissa kannattaa ensin varmistua, että käytät varmasti `time`-moduulia.

Aika-käsite kiehtoo ihmisiä ja siihen liittyen on kehitelty monenlaisia teorioita ja fiktiiviset elokuvat tuovat lisää väriä asiaan. Pythonissa ja tietokoneissa laajemminkin ajankäsittely perustuu insinöörimatematiikkaan eli kysymys on aikajanasta, jossa on nolla-piste ja siitä eteenpäin aika kasvaa sekä ennen sitä aika menee ”taaksepäin”. Käytännössä idea vastaa vanhan analogisen lämpömittarin indikaattoria, joka nousee/menee oikealla lämpötilan noustessa ja alas/vasemmalla lämpötilan laskiessa. Tietokoneissa tuo ”nollapiste” on 1.1.1970 klo 00:00:00, josta mennään tulevaisuutta kohti positiivisilla liukuluvuilla ja taaksepäin negatiivisilla luvuilla. Tämä aikajana-ajatus mahdollistaa aikaleimojen vertailun eli pienempi on aiemmin ja isompi on myöhemmin.

Esimerkissä 8.8 näkyy `time`-moduulin tarjoamia funktioita systeemin kellonajan hakemiseen. Palataan ajan esitusasun muotoiluun seuraavassa esimerkissä ja keskitytään tässä vaiheessa ajan selvittämisen funktioihin ja tietorakenteisiin. Käytämme nyt `time`-moduulia, joten se pitää ensin ottaa käyttöön `import time` -käskyllä. Tämän jälkeen ajan haku onnistuu esim. `gmtime`, `localtime`, `asctime` ja `tmtime` -funktioilla, joilla on pieniä mutta merkittäviä eroja:

1. `gmtime`. Palauttaa UTC-ajan, joka on siis 2/3 tuntia Helsinki-aikaa jäljessä.
2. `localtime`. Palauttaa asetusten mukaisen ”paikallisen” kellonajan, joka on käyttäjän kannalta tyypillisesti ”oikea” aika. **Tässä oppaassa kannattaa käyttää tätä funktiota ajan hakemiseen**, ellei muuta ohjetta anneta.
3. `asctime`. Palauttaa paikallisen ajan muotoiltuna merkkijonona. Merkkijonosta

näkee helposti kaikki oleelliset asiat, mutta esitystapa ei noudata suomalaista käytäntöä. Tätä voi käyttää kehitysvaiheessa, mutta lopullisessa ohjelmassa suomenkielisten tulosteiden seassa tämä ei ole hyväksyttävä muoto.

4. `tmtime(0)`. `tmtime`-funktio palauttaa parametrina olevaa sekuntia vastaavan ajan. Antamalla parametriksi 0 sekuntia näemme, että tietokoneen ajanlaskun nollapiste on 1.1.1970 klo 00:00:00. Tämä on ajan lähtökohta ja siksi se hyvä tietää, vaikka käytännössä tätä tarvitsee harvoin.

**Esimerkki 8.8. time-moduulin peruskäyttö**

```
import time

Aikarakenne = time.gmtime()
Aika = time.localtime()
AikaMerkkijono = time.asctime()
AikaEpoch = time.gmtime(0)

print("Aika suoraan time-kirjaston funktioilla:")
print("time.gmtime():", Aikarakenne)
print("time.localtime():", Aika)
print("time.asctime():", AikaMerkkijono)
print("time.gmtime(0) eli ajanlaskun alku, epoch aika:", AikaEpoch)
```

### *Tuloste*

```
Aika suoraan time-kirjaston funktioilla:
time.gmtime(): time.struct_time(tm_year=2023, tm_mon=10, tm_mday=26,
tm_hour=12, tm_min=47, tm_sec=38, tm_wday=3, tm_yday=299, tm_isdst=0)
time.localtime(): time.struct_time(tm_year=2023, tm_mon=10, tm_mday=26,
tm_hour=15, tm_min=47, tm_sec=38, tm_wday=3, tm_yday=299, tm_isdst=1)
time.asctime(): Thu Oct 26 15:47:38 2023
time.tmtime(0) eli ajanlaskun alku, epoch aika:
time.struct_time(tm_year=1970, tm_mon=1, tm_mday=1, tm_hour=0, tm_min=0,
tm_sec=0, tm_wday=3, tm_yday=1, tm_isdst=0)
```

Esimerkin tulosteesta näkyy, että `asctime` palauttaa merkkijonon ja muut funktiot palauttavat `time.struct_time` -rakenteen. Rakenteen nimen jälkeen suluissa on sen jäsenmuuttujat ja niiden arvot. Käymme tämän rakenteen tarkemmin läpi kohta, sillä siitä löytyy keskeiset aikaan liittyvät tiedot.

## Aikatiedon esitysmuodoista

Esimerkissä 8.9 aika-tietoa muotoillaan `time`-moduulin `strftime`-funktiolla. Esimerkin alussa haetaan aika `gmtime` ja `localtime`-funktioilla kahteen muuttujaan, ja näissä muuttujissa olevat tiedot muotoillaan `time`-moduulin `strftime`-funktiolla merkkijonoksi. Kuten tulosteessa näkyy, esitystapa näyttää nyt suomalaiselta eli päivä, kuukausi, vuosi, tunnit, minuutit ja sekunnit ovat ryhmiteltyinä pisteillä ja kaksoispisteillä. Tärkeintä on huomata, että `localtime`-funktio palauttaa saman kellonajan, mikä näkyy esim. Windowsin tehtäväpalkin kellossa eli sitä kannattaa käyttää. `strftime` saa kaksi parametria, joista ensimmäinen on sisältää tulostettavan merkkijonon muotoilumerkeillä. Nämä muotoilumerkit on koottu alla olevaan taulukkoon 8.1, mutta esimerkistä näkyy päivän tulostus numerona (%d), lyhyenä merkkijonona (Fri, %a) sekä pitkänä merkkijonona (Friday, %A). Kuukauden osalta vastaavat merkit ovat %m, %b ja %B, vuosi tulostuu neljällä numerolla %Y:llä ja kahdella numerolla %y:llä, kun taas tunnit, minuutit ja sekunnit tulostuvat %H, %M ja %S -muotoilumerkeillä. `strftime`-funktio hakee edellä mainitusta `struct_time` -tietorakenteesta nämä tiedot ja tulostaa ne muotoilumerkeillä ilmoitetussa muodossa. Esimerkin lopussa näkyy vielä muotoilumerkit %u ja %V, joilla saa tulostettua

viikonpäivän indeksin ISO-standardin mukaisesti eli viikon ensimmäinen päivä on maanantai ja sen indeksi on 1, sekä viikon indeksin. Viikkonumeroiden kohdalla tulee huomata tekniset yksityiskohdat eli mikä on määritelmän mukaan ensimmäinen viikko (indeksi 1) ja indeksille 0 kuuluvat päivät, joista tarkemmin alla ja Python-dokumentaatiossa.

Esimerkin 8.9 mukaisesti `strftime`-funktion käänteisfunktio on `strptime`. Tällöin lähtökohtana on merkkijono, joka halutaan muuttaa sisäiseksi tietorakenteeksi. `strptime` ottaa parametreina aikatiedon sisältävän merkkijonon ja toisena parametrina muotoilumerkkijonon, jonka mukaisesti aikatieto sijoitetaan `struct_time`:iin. Kun merkkijono on muutettu `struct_time`-muotoon, se voidaan tulostaa esim. muutetussa muodossa näytölle tai kirjoittaa tiedostoon, ja aikatietoja voidaan myös vertailla normaalisti eli selvittää, mikä aikaleima on ennen toista ja laittaa ne aikajärjestykseen tms.

**Esimerkki 8.9. Ajan muotoilu time-moduulin strftime ja strptime -funktioilla**

```
import time

AikaUTC = time.gmtime()
AikaLocal = time.localtime()

print("Aika muotoiltuna strftime:lla:")
AikaUTCMerkkijono = time.strftime("%d.%m.%Y %H:%M:%S", AikaUTC)
AikaLocalMerkkijono = time.strftime("%d.%m.%Y %H:%M:%S", AikaLocal)
print("time.gmtime() + strftime:", AikaUTCMerkkijono)
print("time.localtime() + strftime:", AikaLocalMerkkijono)
print()

print("Aika-muuttujan esitysasu muotoiltuna time.strftime()-jäsenfunktiolla:")
print("Viikonpäivä:", time.strftime("Lyhyt %%a '%a' ja pitkä %%A '%A'", AikaLocal))
print("Kuukausi:", time.strftime("Lyhyt %%b '%b' ja pitkä %%B '%B'", AikaLocal))
print("Viikonpäivän indeksi %u, maanantai=1:", time.strftime("%u", AikaLocal))
print("Viikon indeksi %V:", time.strftime("%V", AikaLocal))
print()

print("Päivämäärät, merkkijonot, sisäinen esitys ja vertailu:")
Juhannus = "20230621 182017"
Joulu = "20231224 181102"
AikaJuhannus = time.strptime(Juhannus, "%Y%m%d %H%M%S") # struct_time
AikaJoulu = time.strptime(Joulu, "%Y%m%d %H%M%S")
JuhannusMuoto2 = time.strftime("%d.%m.%Y %H:%M", AikaJuhannus)
print("Juhannuksen päivämäärä alkuperäisessä muodossa:", Juhannus)
print("Juhannuksen päivämäärä muokattuna:", JuhannusMuoto2)
print("Juhannus on ennen joulua:", AikaJoulu > AikaJuhannus)
```

## *Tuloste*

```
Aika muotoiltuna strftime:lla:
time.gmtime() + strftime: 27.10.2023 08:18:49
time.localtime() + strftime: 27.10.2023 11:18:49

Aika-muuttujan esitysasu muotoiltuna time.strftime()-jäsenfunktiolla:
Viikonpäivä: Lyhyt %a 'Fri' ja pitkä %A 'Friday'
Kuukausi: Lyhyt %b 'Oct' ja pitkä %B 'October'
Viikonpäivän indeksi %u, maanantai=1: 5
Viikon indeksi %V: 43

Päivämäärät, merkkijonot, sisäinen esitys ja vertailu:
Juhannuksen päivämäärä alkuperäisessä muodossa: 20230621 182017
Juhannuksen päivämäärä muokattuna: 21.06.2023 18:20
Juhannus on ennen joulua: True
```

Tässä oppaassa tavoitteena on ymmärtää ja pystyä tekemään perusoperaatiot aikaan liittyen. Käytännössä tämä tarkoittaa seuraavia asioita, jotka näkyvät koodina esimerkissä 8.10:

1. Selvitä tietokoneen järjestelmän aika
2. Tulosta sisäisessä muodossa oleva aikatieto merkkijonona näytölle ja tiedostoon
3. Muuta merkkijonona oleva aikatieto järjestelmän sisäiseen esitysmuotoon
4. Vertaile aikatietoa, esim. onko aika1 sama kuin aika2, vai aiemmin vai myöhemmin

**Esimerkki 8.10. Keskeiset ajankäsittelytoiminnot**

```
import time

print("Tietokoneen aika merkkijonoksi ja näytölle:")
AikaNyt = time.localtime()
AikaMerkkijono = time.strftime("%d.%m.%Y %H:%M", AikaNyt)
print("Tietokoneen aika on nyt:", AikaMerkkijono)
print()

print("Merkkijono-muotoinen aika sisäiseen muotoon ja vertailu:")
AattoMerkkijono = "20231224"
PaivaMerkkijono = "20231223"
AikaAatto = time.strptime(AattoMerkkijono, "%Y%m%d")
AikaPaiva = time.strptime(PaivaMerkkijono, "%Y%m%d")

if (AikaPaiva == AikaAatto):
    print("Nyt on jouluaatto.")
elif (AikaPaiva < AikaAatto):
    print("Jouluaatto on edessä.")
elif (AikaPaiva > AikaAatto):
    print("Jouluaatto meni jo.")
else:
    print("Onko joulu peruttu?")
```

## *Tuloste*

```
Tietokoneen aika merkkijonoksi ja näytölle:
Tietokoneen aika on nyt: 28.10.2023 12:28

Merkkijono-muotoinen aika sisäiseen muotoon ja vertailu:
Jouluaatto on edessä.
```

Keskeiset aikaan liittyvät muotoilumerkit on koottu taulukkoon 8.1. Tässä oppaassa keskitymme päivämääriin ja kellonaikoihin, ja muut muotoilumerkit demonstroivat aikamuunnosten koko kirjoa. Jos siis tulee tarve selvittää aikaan liittyviä tietoja ja muotoilla niitä, Pythonin valmiista kirjastoista löytyy paljon työkaluja näihin tehtäviin. Ja tarvittaessa tarkempia tietoja kannattaa katsoa Python-dokumenteista.

**Taulukko 8.1 Päivämäärä-tietojen muotoilumerkkejä, tiivistelmä Python Help:stä**

| Directive | Meaning | Example |
|---|---|---|
| %a | Weekday as locale's abbreviated name. | Sun, Mon, …, Sat (en_US) |
| %A | Weekday as locale's full name. | Sunday, Monday, …, Saturday (en_US) |
| %w | Weekday as a decimal number, where 0 is Sunday and 6 is Saturday. | 0, 1, …, 6 |
| %d | Day of the month as a zero-padded decimal number. | 01, 02, …, 31 |
| %b | Month as locale's abbreviated name. | Jan, Feb, …, Dec (en_US) |
| %B | Month as locale's full name. | January, February, …, December (en_US) |
| %m | Month as a zero-padded decimal number. | 01, 02, …, 12 |
| %y | Year without century as a zero-padded decimal number. | 00, 01, …, 99 |
| %Y | Year with century as a decimal number. | 0001, 0002, …, 2013, 2014, …, 9998, 9999 |
| %H | Hour (24-hour clock) as a zero-padded decimal number. | 00, 01, …, 23 |
| %I | Hour (12-hour clock) as a zero-padded decimal number. | 01, 02, …, 12 |
| %M | Minute as a zero-padded decimal number. | 00, 01, …, 59 |
| %S | Second as a zero-padded decimal number. | 00, 01, …, 59 |
| %z | UTC offset in the form ±HHMM[SS[.ffffff]] (empty string if the object is naive). | (empty), +0000, -0400, +1030, +063415, -030712.345216 |
| %Z | Time zone name (empty string if the object is naive). | (empty), UTC, EST, CST |
| %j | Day of the year as a zero-padded decimal number. | 001, 002, …, 366 |
| %W | Week number of the year (Monday as the first day of the week) as a decimal number. All days in a new year preceding the first Monday are considered to be in week 0. | 00, 01, …, 53 |
| %% | A literal '%' character. | % |
| %u | ISO 8601 weekday as a decimal number where 1 is Monday. | 1, 2, …, 7 |
| %V | ISO 8601 week as a decimal number with Monday as the first day of the week. Week 01 is the week containing Jan 4. | 01, 02, …, 53 |

## Aikatiedot `struct_time`-tietorakenteessa

Edellisissä esimerkeissä käytettiin `time`-moduulin `struct_time`-rakennetta, joten esimerkissä 8.11 näkyy tämän rakenteen jäsenmuuttujat. Kyseessä on tietorakenne eli se sisältää vain tietoa eikä jäsenfunktioita, ja tällä kertaa nämä tiedot ovat kokonaislukuja. Esimerkin tulosteen mukaisesti rakenteessa on vuosi, kuukausi, päivä, tunti, minuutti ja sekunti, jotka ovat kokonaislukuja seuraavasti:

1. sekunti, 0 <= x <= 61
2. minuutti, 0 <= x < 60
3. tunti, 0 <= x < 24
4. viikko, 0 <= x <= 53

5. kuukausi, 1 <= x <= 12
6. päiviä kuukaudessa, 1 <= x <= 31
7. päiviä vuodessa, 1 <= x <= 366

Niin, siis sekunteja voi olla 61, joista 61 on tuettu historiallisista syistä ja 60 on mukana karkaussekuntien takia. Nämä otettiin käyttöön 1972, jonka jälkeen 27 sekuntia on lisätty atomikellon ja maan pyörimisnopeuden synkronoimiseksi. Asia on käsitelty tarkemmin esim. Wikipediassa, Leap second, jossa on kuva 8.2 eli karkaussekunti kellossa 31.12.2016.

**Kuva 8.2. Karkaussekunti kellossa 31.12.2016 (Wikipedia, time.gov)**


Esimerkin 8.11 lopussa tulostetaan monesko viikonpäivä ja vuoden päivä on kyseessä. Nämä ovat hyödyllisiä tietoja tietyissä tehtävissä, joten niiden olemassaolo on syytä huomata. Toinen asia on, että viikonpäivän kohdalla laskenta alkaa maanantaista luvulla 0. Tämä on yhteensopiva listan indeksinä eli listan ensimmäiseen alkioon saa helposti maanantain tiedot. Mikäli maanantai on helpompi käsitellä numerona 1, voi käyttää `strftime`-funktiota ja %u -muotoa yllä mainitun mukaisesti, joka palauttaa numeroita 1-7. Muutenkin päivämäärien kanssa työskennellessä on syytä muistaa seuraavat asiat:

1. minuutissa on aina 60 sekuntia
2. tunnissa on aina 60 minuuttia
3. päivässä on aina 24 tuntia, paitsi kesä-/talviaikasiirrot
4. viikossa on aina 7 päivää, paitsi vuoden ensimmäinen ja viimeinen viikko
5. kuukaudessa olevien päivien lukumäärä vaihtelee, karkauspäivä lisää epäsäännöllisyyttä
6. vuodessa olevien päivien lukumäärä vaihtelee karkauspäivän takia.

**Esimerkki 8.11. Aikatiedot `struct_time`-tietorakenteessa**

```
import time

Aika = time.localtime()
print("time kirjaston struct_time:n keskeiset jäsenmuuttujat numeroina:")
print("Vuosi Aika.tm_year:", Aika.tm_year)
print("Kuukausi Aika.tm_mon:", Aika.tm_mon)
print("Päivä Aika.tm_mday:", Aika.tm_mday)
print("Tunti Aika.tm_hour:", Aika.tm_hour)
print("Minuutti Aika.tm_min:", Aika.tm_min)
print("Sekunti Aika.tm_sec:", Aika.tm_sec)
print("Monesko viikonpäivä, maanantai=0, Aika.tm_wday:", Aika.tm_wday)
print("Monesko päivä vuodessa Aika.tm_yday:", Aika.tm_yday)
```

## *Tuloste*

```
time kirjaston struct_time:n keskeiset jäsenmuuttujat numeroina:
Vuosi Aika.tm_year: 2023
Kuukausi Aika.tm_mon: 10
Päivä Aika.tm_mday: 27
Tunti Aika.tm_hour: 10
Minuutti Aika.tm_min: 14
Sekunti Aika.tm_sec: 21
Monesko viikonpäivä, maanantai=0, Aika.tm_wday: 4
Monesko päivä vuodessa Aika.tm_yday: 300
```

## Suoritusajan selvittäminen

Edellä keskityimme päivämääriin ja kellonaikoihin eli kalenteri-tason asioihin, mutta joskus tarvitaan tarkempaa seurantaa sekunnin tuhannesosien luokassa. Tyypillinen esimerkki on ohjelman hitaus ja siten tarve selvittää, missä ohjelman osassa menee minkäkin verran aikaa, kuinka paljon aikaa oikeasti menee ja olisiko joku toinen ratkaisu nopeampi.

Esimerkissä 8.12 otetaan selvää, kuinka paljon yksi funktiokutsu vie aikaa. Asiaa selviää `time`-moduulin funktioilla ja esimerkissä käytetään kahta varianttia samasta kutsusta. Ensin kutsutaan `perf_counter`-funktiota, joka on tehty lyhyen ajan mittaamiseen tarkasti. Funktio palauttaa ajan sekunteina, ja kun näitä kutsuja tehdään kaksi, on meillä tiedossa funktiokutsujen väliin jäänyt aika. Esimerkissä 8.12 kutsujen välissä on `sleep(3)` käsky, eli systeemi odottaa 3 sekuntia ja tämä näkyy myös aikaleimojen erotuksessa, joka on nyt 3.0006355 sekuntia eli hieman yli 3 sekuntia. Esimerkissä tehdään seuraavaksi sama asia uudestaan käyttäen `perf_counter_ns`-funktiota, joka palauttaa ajan kokonaislukuna yksikkönä nanosekunnit. Kuten olemme nähneet, desimaalilukujen kanssa tulee aina jossain vaiheessa pyöristysvirheitä ja tämä eliminoi ne.

**Esimerkki 8.12. Suoritusajan selvittäminen**

```
import time

print("Suoritusajan selvittäminen, 2 aikaleimaa sleep(3) välissä:")
print("time.perf_counter() eli sekunteina desimaalilukuina:")
AikaLeima1 = time.perf_counter()
time.sleep(3)
AikaLeima2 = time.perf_counter()
print("Aikaleimat {} ja {}, erotus {}.".format(AikaLeima1,
            AikaLeima2, AikaLeima2-AikaLeima1))
print()

print("time.perf_counter_ns() eli nanosekunteina kokonaislukuina:")
AikaLeima1 = time.perf_counter_ns()
time.sleep(3)
AikaLeima2 = time.perf_counter_ns()
print("Aikaleimat {} ja {}, erotus {}.".format(AikaLeima1, AikaLeima2,
            (AikaLeima2-AikaLeima1)/1000000000))
```

### *Tuloste*

```
Suoritusajan selvittäminen, 2 aikaleimaa sleep(3) välissä:
time.perf_counter() eli sekunteina desimaalilukuina:
Aikaleimat 14083.9877843 ja 14086.9884198, erotus 3.0006355000004987.

time.perf_counter_ns() eli nanosekunteina kokonaislukuina:
Aikaleimat 14087012783600 ja 14090013509000, erotus 3.0007254.
```

Unix-järjestelmissä ajanotto toimii hieman eri tavalla, mutta antaa samankaltaisen vastauksen. Tämä johtuu siitä, että Unixissa aikaa ei oteta järjestelmärajapinnan apufunktiolta, vaan se otetaan suoraan prosessin käyttämältä prosessorilta. Käsite ”prosessoriaika” sisältää useita huomioon otettavia asioita, mutta siitä huolimatta näillä funktioilla voidaan selvittää suorituskykyä ja optimoida sitä molemmilla alustoilla. Funktiota käyttäessä on muistettava, että saadut ajat ovat suhteellisia ja mitattu tulos on vertailukelpoinen vain samassa ajoympäristössä samalla työasemalla.

Mikäli ohjelmien suorituskyky kiinnostaa enemmän, kannattaa tutustua standardikirjaston lukuun Python Profilers, jossa käydään läpi Pythonin tarjoamia ratkaisuja ohjelmien profilointiin eli suorituskyvyn arviointiin.

## Päivämäärillä laskeminen

Pythonissa on perustyökalut päivämäärillä laskemiseen. Laskennan lähtökohta on päivämäärät, sillä ne perustuvat vakioarvoihin paria poikkeusta lukuun ottamatta. `struct_time`-kohdan mukaisesti minuutissa on 60 sekuntia, tunnissa 60 minuuttia ja päivässä 24 tuntia. Näihin liittyy kuitenkin karkaussekunnit ja -päivät sekä kesä-/talviaika muunnokset, ja tarve huomioida nämä laskennassa on arvioitava aina tapauskohtaisesti.

Esimerkissä 8.13 näkyy aikatietojen laskennan perusteet. Aluksi on syytä huomata, että käytämme nyt `datetime`-moduulia `time`-moduulin sijasta. Tämä johtuu sitä, että laskenta on toteutettu eri periaatteilla kuin aiempi aika-tietojen käsittely ja se perustuu olioihin. Myös seuraava esimerkki 8.14 perustuu `datetime`-moduuliin ja olioihin, joten tässä vaiheessa kannattaa toimia seuraavasti:

1. Mikäli tarvitset **aika-tietoa**, päivämääriä, kellonaikoja jne., käytä **`time`-moduulia**
2. Mikäli sinun tarvitse **laskea päivämäärillä**, käytä **`datetime`-moduulia**. Laskenta tarkoittaa päivämäärien erotusta ja yhteenlaskua eli siirtymiä.

Esimerkissä 8.13 merkkijonona oleva aikaleima muutetaan sisäiseen esitysmuotoon `strptime`-funktiolla, joka saa parametreina päivämäärän merkkijonona ja muotoilumerkkijonon eli vastaavalla tavalla kuin aiemminkin – paitsi että nyt `time.strptime`-funktion sijasta kutsumme `datetime.datetime.strptime`-funktiota. Kyseessä on siis `datetime`-moduulin `datime`-luokan jäsenfunktio `strptime`, kun aiemmin käytimme `time`-moduulin `strptime`-funktiota. Ero on pieni, joten ongelma on lähinnä muistaa funktiokutsun oikea muotoa. Tästä ei kuitenkaan tarvitse tehdä ongelmaa, sillä ToolTip tuo aina pisteen jälkeen ko. rakenteeseen kuuluvat rakenteet näkyville, ja niistä voi valita itselle sopivan. Ja lopuksi laittamalla funktion nimen perään aukeavan sulun, tulee parametrilista näkyville. Tällä tavoin saat merkkijonona olevan aikatiedon sisäiseen esitysmuotoon esim. muuttujaan `Aika1`, ja voit laskea sillä esimerkin mukaisesti erotuksen sekä tulostaa `Erotus`-olion sisältämän `days`-jäsenmuuttujan arvon. `Erotus`-olion luokka on `timedelta`, joka on hyvä tietää katsoessa tarkempia tietoja helpistä, mutta muutoin sitä ei tarvitse huomioida.

**Esimerkki 8.13. Päivämäärillä laskenta perustuu datetime-kirjastoon**

```
import datetime

print("Laskenta datetime-moduulilla")
print()
print("2023 kevät- ja syyspäiväntasausten väli:")
TasausKevatpaiva = "20.3.2023 21:24:24"
TasausSyyspaiva = "23.9.2023 06:49:56"
Aika1 = datetime.datetime.strptime(TasausKevatpaiva, "%d.%m.%Y %H:%M:%S")
Aika2 = datetime.datetime.strptime(TasausSyyspaiva, "%d.%m.%Y %H:%M:%S")
Erotus = Aika2 - Aika1
print("Tasausten välissä on {0:d} päivää.".format(Erotus.days))
print()

print("datetime.timedelta huomioi karkauspäivän:")
Aika1 = datetime.datetime.strptime("28.2.2024", "%d.%m.%Y")
Aika2 = datetime.datetime.strptime("1.3.2024", "%d.%m.%Y")
print("28.2.-1.2.2024:", (Aika2-Aika1).days, "päivää")
Aika1 = datetime.datetime.strptime("28.2.2023", "%d.%m.%Y")
Aika2 = datetime.datetime.strptime("1.3.2023", "%d.%m.%Y")
print("28.2.-1.2.2023:", (Aika2-Aika1).days, "päivää")
```

### *Tuloste*

```
Laskenta datetime-moduulilla

2023 kevät- ja syyspäiväntasausten väli:
Tasausten välissä on 186 päivää.

datetime.timedelta huomioi karkauspäivän:
28.2.-1.2.2024: 2 päivää
28.2.-1.2.2023: 1 päivää
```

Pythonin peruskirjastot huomioivat standardipoikkeamat ajankäsittelyssä eli karkauspäivät ja kesä-/talviaika. Esimerkin 8.13 mukaisesti karkauspäivä näkyy oikein laskennassa ja jos kesä-/talviaikaa tarvitsee käsitellä tarkemmin, `struct_time`-rakenteen muuttuja `tm_isdst` kertoo, onko aika kesäaika (Daylight Savings Time, DST). Ajan tarkasti ottaen oikea käsittely on haastavaa ja työlästä. Siksi tämän oppaan laajuudessa riittää ymmärtää, minkälaiset asiat vaikuttavat ajan oikeaan käsittelyyn ja miten asiaa voi lähteä selvittämään tarkemmin tarvittaessa.

## Päivämäärien siirtymät

Esimerkin 8.13 yhteydessä käytiin läpi päivämäärillä laskennan perusteet ja esimerkissä 8.14 esitellään miten siirtymiä voi hyödyntää ohjelmissa. Kuten aiemmin on ollut puhetta, kuukauden viimeinen päivä ei ole itsestään selvä asia vaan se muuttuu kuukausittain ja lisäksi karkausvuosi muuttaa helmikuun päivien lukumäärää. Toisaalta jokaisessa kuukaudessa on ensimmäinen päivä, joten edellisen kuukauden viimeiseen päivään voi siirtyä kuukauden ensimmäisestä päivästä vähentämällä ajasta yhden päivän esimerkin mukaisesti `datetime.timedelta(days=-1)` -kutsulla.

Toinen tyypillinen tehtävä aika-tiedon kanssa on tietojen luokittelu suurempiin kokonaisuuksiin. Esimerkiksi sähkölaitokselta pystyy lataamaan sähkönkulutustiedot tunnin tarkkuudella, ja vaikka tämä on hyvä ja käytännöllinen esitysmuoto, on joskus helpompi katsoa tietoja päivittäin, viikoittain, kuukausittain ja vuosittain. Kuten aiemmin oli puhetta, päivässä on (melkein) aina 24 tuntia, viikossa on 7 päivää, kuukaudessa on monta päivää, ja vuodessa on aina 12 kuukautta sekä vaihteleva määrä viikkoja. Jotta käyttäjällä – tai asiakkaalle – näytettävät tulosteet olisivat selkeitä Excel-tyylisiä taulukoita, joutuu ohjelmoija usein tulkitsemaan annettua dataa ja tässä kohdin aikasiirtymät on hyvä työkalu. Esimerkissä 8.14 näkyy, miten lähtötietona annettu ”30.10.2023 on maanantai” mahdollistaa seuraavan maanantain selvittämisen `datetime.timedelta(weeks=1)` siirtymän avulla. Esimerkin 8.11 mukaseisti `struct_time` sisältää paljon tietoa ajasta, ja `strftime`-funktio palauttaa halutessa vastaavia tietoja taulukon 8.1 mukaisesti. Näillä numeerisilla tiedoilla tietoja voi ryhmitellä esim. listassa samaan ryhmään, eikä tietenkään kannata unohtaa esimerkissä 6.13 tehtyä tietojen ryhmittelyä sopiviin osajoukkoihin.

**Esimerkki 8.14. Päivämäärien siirtymät**

```
import datetime

print("Siirtymät datetime-moduulin timedelta-luokalla")
print()
print("Kuukauden viimeinen päivä:")
MaaliskuunEnsimmainenStr = "1.3.2023"
MaaliskuunEnsimmainen = datetime.datetime.strptime(MaaliskuunEnsimmainenStr,
                                "%d.%m.%Y")
HelmikuunViimeinen = MaaliskuunEnsimmainen + datetime.timedelta(days=-1)
HelmikuunViimeinenStr = HelmikuunViimeinen.strftime("%d.%m.%Y")
print("Kuukauden ensimmäinen päivä:", MaaliskuunEnsimmainenStr)
print("Edellisen kuukauden viimeinen päivä:", HelmikuunViimeinenStr)
print()

print("Viikkoa myöhemmin:")
MaanantaiStr = "30.10.2023" # Kalenterin mukaan tämä päivä on maanantai
Maanantai = datetime.datetime.strptime(MaanantaiStr, "%d.%m.%Y")
SeuraavaMaanantai = Maanantai + datetime.timedelta(weeks=1)
SeuraavaMaanantaiStr = SeuraavaMaanantai.strftime("%d.%m.%Y")
print("Maanantai:", MaanantaiStr)
print("Seuraava maanantai:", SeuraavaMaanantaiStr)
print()
```

### *Tuloste*

```
Siirtymät datetime-moduulin timedelta-luokalla

Kuukauden viimeinen päivä:
Kuukauden ensimmäinen päivä: 1.3.2023
Edellisen kuukauden viimeinen päivä: 28.02.2023

Viikkoa myöhemmin:
Maanantai: 30.10.2023
Seuraava maanantai: 06.11.2023
```

# *Muuta huomionarvoista*

## Ulkoisten kirjastojen lisääminen Pythoniin

Pythonin mukana tulevat kirjastot saa käyttöön `import`-käskyllä ja niiden lisäksi on kirjastoja, jotka voi itse lisätä omaan Python asennukseen. Asennuksen jälkeen kirjastoja voi ottaa käyttää aiempien kirjastojen tavoin `import`-käskyllä. Esimerkkejä tällaisista kirjastoista ovat seuraavat kolme pakettia:

1. **numpy** – Numerical Python, erityisesti matriisit/taulukot.
2. **matplotlib** – graafien ja diagrammien piirtämiseen tehty kirjasto
3. **svgwrite** – vektorigrafiikan piirtämiseen tehty kirjasto, jolla pystyy tekemään svg-tiedostoformaatin mukaisia piirustuksia (Scalable Vector Graphics).

Nämä kirjastot toimivat esimerkkeinä siitä, miten Pythonin käyttömahdollisuuksia voi entisestään laajentaa ja tehostaa, ja asiasta kiinnostuneet voivat etsiä lisäinformaatiota yllä olevilla hakusanoilla Internetistä. numpy-kirjaston matriiseihin tutustutaan tarkemmin luvussa 10, mutta muutoin näiden kirjastojen käyttö jää oman harrastuksen varaan.

Mainitut kirjastot voi asentaa Pythoniin sen omalla asennustyökalulla PIP. Koska numpy-kirjasto asentuu matplotlib-kirjaston asennuksen yhteydessä, sitä ei tarvitse asentaa erikseen. Alla on askeleet Pythonin asennukseen, jonka jälkeen siihen lisätään kirjastot. Mikäli Pythonin asennus on kunnossa, ei sitä tarvitse toistaa vaan voit siirtyä suoraan kirjastojen asennukseen PIP:llä.

Tee Pythonin perusasennus koneelle asennusohjelman ohjeiden mukaisesti ja huomaa seuraavat asiat

1. Perusasennuspaketti, esim. https://www.python.org/downloads/release/python-3111/
2. Asennuksessa tulee olla mukana IDLE, PIP, dokumentaatio, launcher ja ympäristömuuttujat tulee päivittää, erityisesti Python tulee lisätä polku/PATH-muuttujaan

Onnistuneen Python-perusasennuksen jälkeen lisätään kirjastot PIP:llä:

1. Käynnistä Windowsin Komento-ikkunaa (cmd) ja mene Pythonin asennushakemistoon
2. Tarkista, että asennus on mennyt oikein ja seuraaviin käskyihin tulee järkevät versionumerot eikä ilmoitus, ettei ohjelmaa löydy tai muuta virheilmoitusta. Huomaa, että ”version” sanan edessä on 2 kpl tavuviivoja eli --
    a. python --version
    b. pip --version
3. Asenna ensin matplotlib ja sen jälkeen svgwrite alla olevilla käskyillä
    a. pip install matplotlib==3.6.3
    b. pip install svgwrite==1.4.3

Molempien PIP-käskyjen pitäisi tuottaa useita rivejä tietoa asennuksen etenemisestä ja onnistumisesta. Huomaa, että em. versionumerot 3.11.1, 3.6.3 ja 1.4.3 eivät ole satunnaisia vaan näiden ohjelmistojen yhteensopivia versioita. Koska matplotlib sisältää numpy-kirjaston, ei sitä tarvitse asentaa erikseen.

## Esikäännetyt pyc-tiedostot

Moduulin sisällyttäminen on Python-tulkille suhteellisen raskas ja aikaa vievä toimenpide, joten Python nopeuttaa asiaa muutamalla tempulla. Yksi temppu on luoda esikäännettyjä tiedostoja, jotka tunnistat tiedostopäätteestä pyc. Tämä pyc-tiedosto nopeuttaa Pythonin toimintaa huomattavasti, koska tällainen tiedosto on jo valmiiksi käännetty tulkille muotoon, jossa se voidaan nopeasti sisällyttää toiseen ohjelmaan. Jos tiedostoa ei ole olemassa, joutuu tulkki suorittamaan ensin esikäännöksen ja tämän jälkeen tulkitsemaan sen osaksi koodia. Periaatteessa pyc-tiedostoista riittää tietää, että ne ovat tulkin luomia tiedostoja ja nopeuttavat ohjelman toimintaa. Niitä ei kuitenkaan voi muokata käsin, eikä niitä voi tulkita kuin konekielen ohjelmia. Niiden poistaminen on sallittua; jos tiedostoa ei ole olemassa, tekee tulkki uuden käännöksen, kunhan alkuperäinen lähdekooditiedosto (.py) on saatavilla.

## from...import -sisällytyskäsky

Joskus voimme haluta moduulin funktiot ja muuttujat käyttöön ilman pistenotaatiota. Eli käyttää esimerkiksi `math`-kirjaston funktioita ilman `math`-etuliitettä pelkästään funktionimellä `sin`.

Tämä onnistuu notaatiolla `from x import y`, jossa x korvataan halutun moduulin nimellä ja y funktiolla tai sen muuttujan nimellä, joka halutaan ottaa suoraan käyttöön. Jos esimerkiksi haluaisimme ottaa kaikki `math`-moduulin rakenteet suoraan käyttöömme, voisimme tehdä sen käskyllä `from math import *`, jossa ”*” `import`-osan jälkeen tarkoittaa, että haluamme tuoda kaikki funktiot ja muuttujat käyttöömme. Toisaalta, jos haluaisimme tyytyä pelkkään `sin`iin, niin voisimme tehdä sen käskyllä `from math import sin`.

Tähän sisällytystapaan liittyy yksi vakava riski, joka on ylikirjoittumisvaara. Jos kaksi moduulia sisältää samannimisen funktion, kirjoitetaan ensin sisällytetyn moduulin funktio yli eikä siihen päästä enää käsiksi. Oletetaan, että meillä on esimerkin 8.17 mukaisesti kaksi moduulia nimeltään `L08E17Nastola` ja `L08E17Hattula`, ja molemmissa on moduulin nimen tulostava funktio `kerronimi`. Esimerkissä 8.15 toimitaan tähän asti oppaassa neuvotulla tavalla eli `import moduulinimi`, ja funktiokutsuissa on ensin moduulinimi

ja sitten pisteen perässä funktionimi. Esimerkissä 8.16 taas on `from … import *` -sisällytys ja funktioita kutsutaan vain niiden nimillä – tai siis nyt kaksi kertaa samaa funktiota, koska molemmissa moduuleissa on saman niminen funktio.

Esimerkissä 8.15 ohjelma toimii aiemmin kuvatun mukaisesti ilman yllätyksiä. Esimerkin 8.16 ohjelmassa tulee aika nopeasti mieleen, että miksi kutsutaan kaksi kertaa samaa funktioita, sillä tuloksenhan pitäisi olla sama? Ja näinhän se on – ohjelmassa `L08E17Nastola`-sisällytyksen jälkeen `kerronimi`-funktio on käytössä ja viittaa `L08E17Nastola`-moduulin funktioon. Mutta `L08E17Hattula`-moduulin sisällytyksen yhteydessä `kerronimi`-funktion tilalle tulee `L08E17Hattula`-moduulin funktio ja aiempaa `L08E17Nastola`-moduulin funktio tulee ylikirjoitetuksi eikä sitä enää löydy.

**Esimerkki 8.15. import ja kirjastonimen käyttö funktiokutsuissa**

```
import L08E17Nastola
import L08E17Hattula

print("import ja kirjastonimen käyttö funktiokutsuissa:")
print("L08E17Nastola.kerronimi():", L08E17Nastola.kerronimi())
print("L08E17Hattula.kerronimi():", L08E17Hattula.kerronimi())
```

## *Tuloste*

```
import ja kirjastonimen käyttö funktiokutsuissa:
L08E17Nastola.kerronimi(): L08E17Nastola
L08E17Hattula.kerronimi(): L08E17Hattula
```

**Esimerkki 8.16. 'from ... import *' -toiminta samannimisellä funktiolla**

```
from L08E17Nastola import *
from L08E17Hattula import *

print("'from ... import *' -toiminta samannimisellä funktiolla:")
print("kerronimi():", kerronimi())
print("kerronimi():", kerronimi())
```

## *Tuloste*

```
'from ... import *' -toiminta samannimisellä funktiolla:
kerronimi(): L08E17Hattula
kerronimi(): L08E17Hattula
```

**Esimerkki 8.17. Kahdessa kirjastossa L08E17Nastola ja L08E17Hattula sama funktio kerronimi()**

```
# L08E17Nastola.py
def kerronimi():
    return "L08E17Nastola"
# eof

# L08E17Hattula.py
def kerronimi():
    return "L08E17Hattula"
# eof
```

Esimerkissä 8.16 emme pääsisi käsiksi `L08E17Nastola`-moduulin funktioon `kerronimi` millään muulla tavalla kuin sisällyttämällä funktion uudelleen. Yleisesti ottaen kannattaa välttää `from...import`-syntaksia, koska se hyödyttää harvoin koodia niin paljoa, että sekoittumis- ja ylikirjoitusriski nimiavaruuksien kanssa olisi perusteltua. Tähän sääntöön on olemassa muutama yleisesti hyväksytty poikkeus, kuten myöhemmin näemme. Mutta omien moduulien kanssa työskennellessä `from...import` kannattaa jättää väliin.

# *Yhteenveto*

## Osaamistavoitteet

Tämän luvun keskeiset asiat on nimetty alla olevassa listassa. Tarkista sen avulla, että tunnistat nämä asiat ja sinulla on käsitys siitä, mitä ne tarkoittavat. Mikäli joku käsite tuntuu oudolta, käy siihen liittyvät asiat uudestaan läpi. Nämä käsitteet ja asiat on hyvä muistaa ja tunnistaa, sillä seuraavaksi teemme niihin perustuvia ohjelmia.

- Kirjastojen käyttö (E8.1)
- Oman kirjaston tekeminen ja käyttö (E8.2, 8.3)
- Valmiiden kirjastojen käyttö (Kuvat 8.1, 8.2)
    - Pythonin keskeisiä kirjastoja: math (E8.4), random (E8.5), fractions (E8.6), urllib (E8.7)
- Aikatiedon käsittely
    - Aika ja sen muotoilu: time, strftime, strptime (E8.8, 8.9, 8.10, Taulukko 8.1)
    - Aikarakenne: struct_time (E8.11)
    - Suoritusaika: (E8.12)
    - Laskenta ja siirtymät: datetime (E8.13, 8.14)
- Muita huomionarvoisia asioita
    - Ulkoisten kirjastojen lisäys: svgwrite, matplotlib
    - Esikäännetyt pyc-tiedostot
    - Älä käytä from … import (E8.15, 8.16, 8.17)
- Yhteenveto, kokoava esimerkki (E8.18)

## Pienen Python-perusohjelman tyyliohjeet

Tähän lukuun liittyvät tyyliohjeet on koottu alle. Alla olevia yleisohjeita noudattamalla vältät yleisimmät kirjastoihin liittyvät ongelmat ja perustellusta syystä niistä voi ja tulee poiketa.

**Kirjaston käyttö**

1. Kurssilla käsittelemättömien kirjastojen käyttö on kielletty
2. Itse tehdyt kirjastotiedostot nimetään `xxKirjasto.py`. Kahdesta tiedostosta muodostuvan ohjelman tiedostot olisivat esimerkiksi pääohjelma `L08T1.py` ja kirjasto `L08T1Kirjasto.py`

**Tiedostorakenne, kun ohjelma koostuu vain yhdestä tiedostosta**

3. Tiedoston alkukommentti
4. Kirjastojen sisällytykset
5. Kiintoarvojen määrittely
6. Luokkien määrittely
7. Aliohjelmien koodi kutsujärjestyksessä
8. Pääohjelman koodi
9. Pääohjelman kutsu

**Tiedostorakenne, kun ohjelma koostuu kahdesta/useista tiedostosta**

**Pääohjelman sisältävän tiedoston rakenne**

10. Tiedoston alkukommentti
11. Pääohjelmassa tarvittavien standardikirjastojen sisällytykset
12. Omien kirjastojen sisällytykset
13. Kiintoarvojen määrittely
14. Luokkien määrittely
15. valikko-aliohjelma
16. Pääohjelman koodi
17. Pääohjelman kutsu

**Kirjastojen tiedostorakenne**

18. Tiedoston alkukommentti
19. Kirjastossa tarvittavien standardikirjastojen sisällytykset
20. Kiintoarvojen määrittely
21. Luokkien määrittely
22. Aliohjelmien koodi kutsujärjestyksessä

**Tietojen analysointi**

23. Analyysi-aliohjelma saa tyypillisesti parametrina kaksi tietorakennetta: analysoitavan datan sisältävän tietorakenteen ja tulostietorakenteen. Analysoitava data on tyypillisesti listassa ja tulostietorakenne on tyypillisesti olio tai lista. Mikäli tulostietorakenne on lista, tulee aiemmat analyysin tulokset poistaa tyhjentämällä lista ennen analyysiä
24. Analyysin jälkeen tulokset sijoitetaan tulostietorakenteeseen
25. Etsittäessä datasta tiettyjä arvoja, esim. minimi tai maksimi, tulee tilapäismuuttujat alustaa datasetin ensimmäisen alkion/olion arvoilla
26. Mikäli datassa on useita ehdon täyttäviä arvoja, esim. useita yhtä suuria minimi-/maksimiarvoja, valitaan näistä
    a. erikseen mainitun ehdon täyttävä arvo, jos tällainen ehto on olemassa
    b. vanhin, jos datassa on aikaleima
    c. alkuperäisen datan ensimmäinen arvo (rivinumeron mukaan)
27. Mikäli tehtävässä käytetään datasetin ominaisuuksia, tulee ne selvittää datasta, esim. alkioiden lukumäärä tai ensimmäisen/viimeisen alkion aikaleima
28. Analyysien tulee hyödyntää käytettävissä olevia tietoja siten, ettei esim. valintarakenteista tule tarpeettoman laajoja (nk. if-pyramidi)
29. Kaikki analyysit tulee suorittaa alkuperäisissä yksiköissä ja mahdollinen tulosten pyöristys tehdään muotoiltaessa lopullisia tulosteita

## Luvut asiat kokoava esimerkki

Esimerkissä 8.18 on luvun kokoava esimerkki. Ohjelma muodostuu kahdesta tiedostosta ja se lukee datatiedoston, jossa olevat tiedot laitetaan listaan olioina eli tekee oliolistan. Oliolista analysoidaan yhdessä aliohjelmassa eli selvitetään suurin ja keskimääräinen askelmäärä sekä minä päivän suurin askelmäärä syntyi. Ryhmittely-vaiheessa vuoden aikana kerätyt askeleet jaetaan viikonpäivittäin eli lasketaan, kuinka monta askelta eri viikonpäivinä on otettu. Lopuksi ohjelma tulostaa tulokset sekä kellonajan ohjelman loppuessa.

**Esimerkki 8.18. Oma kirjasto ja time-kirjaston peruskäskyjä, tiedostot L08E18.py ja L08E18Kirjasto.py**

```
# Pääohjelma: L08E18.py
import time
import L08E18Kirjasto

def paaohjelma():
    Lista = []
    Viikko = []
    Tulokset = L08E18Kirjasto.TULOKSET
    Tiedostonimi = "L08E18D1.txt" # input("Anna luettavan tiedoston nimi: ")
    AlkuMerkkijono = "1.1.2017" # input("Anna ensimmäinen huomioitava päivä: ")
    LoppuMerkkijono = "31.12.2017" # input("Anna viimeinen huomioitava päivä: ")
    Alku = time.strptime(AlkuMerkkijono, "%d.%m.%Y")
    Loppu = time.strptime(LoppuMerkkijono, "%d.%m.%Y")
    Lista = L08E18Kirjasto.lue(Tiedostonimi, Lista, Alku, Loppu)
    Tulokset = L08E18Kirjasto.analysoi(Lista, Tulokset)
    Viikko = L08E18Kirjasto.ryhmittele(Lista, Viikko)
    L08E18Kirjasto.tulosta(Tulokset, Viikko)
    L08E18Kirjasto.tulostaLopetusaika()
    Lista.clear()
    Viikko.clear()
    return None

paaohjelma()
# eof
```

```
# Kirjasto: L08E18Kirjasto.py
# Datatiedoston rivi: ”21-01-2017;3001;14624;10,81;16;507;346;70;26;1826”
import time # Kirjastojen import tiedoston alussa

PAIVIA = 7 # Kiintoarvot
PAIVAT = ["Ma", "Ti", "Ke", "To", "Pe", "La", "Su"]
EROTIN = ';'

class DATA: # Luokat
    Aika = None
    Askelia = None

class TULOKSET:
    AikaMax = None
    AskeliaMax = None
    Keskiarvo = None

def lue(Nimi, Tiedot, Alku, Loppu):
    Tiedot.clear()
    Tiedosto = open(Nimi, 'r')
    Rivi = Tiedosto.readline()
    while (len(Rivi) > 0):
        Sarakkeet = Rivi[:-1].split(EROTIN)
        Aika = time.strptime(Sarakkeet[0], "%d-%m-%Y")
        if ((Aika >= Alku) and (Aika <= Loppu)):
            Data = DATA()
            Data.Aika = Aika
            Data.Askelia = int(Sarakkeet[2]) # askeltiedot 3. sarake
            Tiedot.append(Data)
        Rivi = Tiedosto.readline()
    Tiedosto.close()
    return Tiedot
```

```
def analysoi(Tiedot, Tulokset):
    Lukumaara = 0
    Summa = 0
    AikaMax = Tiedot[0].Aika
    AskeliaMax = Tiedot[0].Askelia

    for Alkio in Tiedot:
        Lukumaara = Lukumaara + 1
        Summa += Alkio.Askelia
        if (AskeliaMax < Alkio.Askelia):
            AikaMax = Alkio.Aika
            AskeliaMax = Alkio.Askelia

    Tulokset = TULOKSET()
    Tulokset.AikaMax = AikaMax
    Tulokset.AskeliaMax = AskeliaMax
    Tulokset.Keskiarvo = Summa / Lukumaara
    return Tulokset

def ryhmittele(Lista, Viikko):
    Viikko.clear()
    for i in range(PAIVIA): # Alustetaan päivien askeleet 0:lla
        Viikko.append(0)
    for Alkio in Lista:     # Lisätään alkion askeleet oikealle päivälle
        Viikko[Alkio.Aika.tm_wday] += Alkio.Askelia
    return Viikko

def tulosta(Tulokset, Viikko):
    Aika = time.strftime("%d.%m.%Y", Tulokset.AikaMax)
    Askelia = Tulokset.AskeliaMax
    Keskiarvo = Tulokset.Keskiarvo
    print("Suurin askelmäärä {0:d} tuli {1:s}.".format(Askelia, Aika))
    print("Keskimäärin askelia kertyi päivässä {0:.0f}.".format(Keskiarvo))

    print("Viikonpäivittäin askelet jakautuivat seuraavasti:")
    for Paiva in range(PAIVIA):
        print("{0:s};{1:d}".format(PAIVAT[Paiva], Viikko[Paiva]))

    return None

def tulostaLopetusaika():
    Aika = time.strftime("Ohjelman suoritus päättyi %d.%m.%Y kello %H:%M.",
                         time.localtime())
    print(Aika)
    return None
##########################################################################
# eof
```

## *Tuloste*

```
Suurin askelmäärä 31589 tuli 09.04.2017.
Keskimäärin askelia kertyi päivässä 10328.
Viikonpäivittäin askelet jakautuivat seuraavasti:
Ma;505631
Ti;502769
Ke;553964
To;516715
Pe;512496
La;592823
Su;585326
Ohjelman suoritus päättyi 30.10.2023 kello 09:06.
```