LUT Python ohjelmointiopas 2023 - osa 2
Sisällysluettelo

Luku 2: Tietotyypit, muuttujien roolit, tulostaminen ja perusohjelma

Tutustuimme tietotyyppeihin jo ensimmäisessä luvussa ja nyt katsomme vähän tarkemmin mistä merkkijonoissa, kokonaisluvuissa ja desimaaliluvuissa on kysymys sekä miten tietotyyppejä muutetaan Python-ohjelmissa. Merkkijonon osalta keskeisiä asioita ovat perusteiden lisäksi niiden yhdistely sekä alimerkkijonojen käsittely, kun taas desimaalilukuja pitää pystyä pyöristämään ja muuttamaan kokonaisluvuiksi. Tutustuimme muuttujiin jo edellisessä luvussa, mutta nyt katsomme lyhyesti mistä muuttujien rooleissa on kysymys, sillä ne auttavat tekemään selkeitä ohjelmia. Luvun päätteeksi palaamme tietojen tulostamiseen ja katsomme tapoja muokata tulosteita käyttäjän haluamaan muotoon.

Merkkijonot

Merkkijono on jono peräkkäisiä merkkejä. Merkkijonot voivat olla esimerkiksi sanoja tai lauseita, mutta merkkijonoina niissä on mikä tahansa joukko merkkejä. Merkkijonoja käytetään ohjelmoinnissa paljon, joten tämä osio kannattaa lukea ajatuksella lävitse ja palata siihen aina tilanteen niin vaatiessa.

Lainausmerkki (”) ja heittomerkki (')

Voit määritellä merkkijonoja käyttäen lainausmerkkejä, esimerkiksi näin: `”Luota minuun tässä asiassa.”`. Kaikki ei-näkyvät merkit kuten välilyönnit tai sisennykset tallentuvat kuten tulostus näyttää ne eli omille paikoilleen lainausmerkkien väliin. Heittomerkki (’) toimii samalla tavoin kuin lainausmerkki. Tässäkin tapauksessa kahden merkin väliin jäävä osa luetaan merkkijonona, esimerkiksi: `’Elämme kovia aikoja ystävä hyvä’`.

Pythonin kieliopin kannalta lainaus- ja heittomerkillä ei ole eroa, joskaan ne eivät toimi keskenään ristiin. Tämä tarkoittaa sitä, että `"Tämä on yritelmä'` ei ole kelvollinen merkkijono, vaikka se teknisesti onkin oikeiden merkkien rajoittama. Lainaus- ja heittomerkkejä tulee käyttää aina pareittain, esim. `"Homma on 'bueno'."`

Merkkijonon tunnuksena kannattaa käyttää lainausmerkkejä. Heittomerkki sopii sanojen korostamiseen merkkijonon sisällä edellisen esimerkin mukaisesti ja sitä tarvitaan myös lauseiden sisällä: `”It’s time to go.”`.

Ohjausmerkit

Ajoittain tulee vastaan tilanteita, jolloin merkkijonossa on pariton määrä heitto- tai lainausmerkkejä, esim. merkkijono `vaa'an alla`. Ongelmaa ei ole, jos käyttää merkkijonon tunnisteena lainausmerkkiä, mutta aina joskus törmäämme `'vaa'an alla'` -tyyliseen merkkijonoon, joka ei siis toimi Pythonissa. Onneksi tämä ongelma on ratkaistavissa ohjausmerkillä (\) eli takakenolla tai kenoviivalla.

Merkkijonon `'vaa'an alla'` ongelma on se, ettei tulkki tiedä, mihin heittomerkkiin merkkijono päättyy. Tämä onnistuu ohjausmerkillä (\), jonka avulla voit merkata yksinkertaisen heittomerkin ohitettavaksi tyyliin \’. Nyt esimerkkirivi `'vaa\'an alla'` toimii ilman ongelmia. Ohjausmerkillä voi myös ”maskata” eli peittää parittoman lainausmerkin kahden lainausmerkin välissä. Tälle tosin on harvoin tarvetta, koska lainausmerkkejä käytetään määritelmän mukaan pareittain.

Kenoviivan käyttömahdollisuudet eivät lopu tähän. Esimerkiksi itse kenoviivan merkitsemiseen käytetään ohitusmerkkiä, jolloin merkintä tulee näin `\\`. Tässä vaiheessa kannattaa muistaa, että kenoviiva on ohjaus- eli erikoismerkki ja se voi muuttaa tulosteita.

Tekstiä voi tulostaa monelle riville käyttämällä rivinvaihtomerkkiä (\n). Rivinvaihtomerkki tulee näkyviin tekstiin normaalisti kenoviiva-n -yhdistelmänä, mutta tulkissa tulostuu rivinvaihtona. Esimerkiksi `"Tämä tulee ensimmäiselle riville. \n Tämä tulee toiselle riville."` Toinen vastaava hyödyllinen merkki on sisennysmerkki (\t), joka vastaa tabulaattorimerkkiä ja jolla voimme tasata kappaleiden reunoja. Ohjausmerkeistä on hyvä tietää lisäksi se, että yksittäinen kenoviiva rivin päässä tarkoittaa sitä, että merkkijono jatkuu seuraavalla rivillä. Tämä aiheuttaa sen, että tulkki ei lisää rivin päähän rivinvaihtoa vaan jatkaa tulostusta samalle riville. Esimerkiksi,

```
"Tämä on ensimmäinen rivi joka tulostuu. \
Tämä tulee ensimmäisen rivin jälkeen."
```

On sama kuin `"Tämä on ensimmäinen rivi joka tulostuu. Tämä tulee ensimmäisen rivin jälkeen."`

Täydellinen lista ohjausmerkeistä löytyy mm. Python Software Foundationin dokumenteista, jotka löytyvät osoitteesta www.python.org.

Esimerkkejä

Alla on esimerkkejä merkkijonoista ja miten ne tulostuvat Pythonissa. Huomaa heitto-, lainaus- ja ohjausmerkkien näkyminen tulosteessa.

Esimerkki 2.1. Merkkijonoja ja ohjausmerkkejä tulosteissa

```
Sana = "kinkkumunakas"
print(Sana)
Sana = "vaa'an"
print(Sana)
Sana = 'raa\'at'
print(Sana)
Lause = "'Kyllä', hän sanoi."
print(Lause)
Lause = "\"Kyllä\", hän sanoi."
print(Lause)
Lause = "'Vaa\'an alla', mies sanoi."
print(Lause)
Lause = "\"Vaa'an alla\", mies sanoi."
print(Lause)
Lause = '"Vaa\'an alla", mies sanoi.'
print(Lause)
Lause1 = "Tämä tulostuu yhdelle riville."
Lause2 = "Tämä tulostuu toiselle riville."
Lause3 = "Numeroita erotettuna kenoviivoilla: 1\\2\\3"
Lause4 = "Numeroita erotettuna sarkainmerkeillä: 1\t2\t3"
Lause5 = "Numeroita erotettuna rivinvaihtomerkeillä:\n1\n2\n3"
print(Lause1)
print(Lause2)
print(Lause3)
print(Lause4)
print(Lause5)
```

Tuloste

```
kinkkumunakas
vaa'an
raa'at
'Kyllä', hän sanoi.
"Kyllä", hän sanoi.
'Vaa'an alla', mies sanoi.
"Vaa'an alla", mies sanoi.
"Vaa'an alla", mies sanoi.
Tämä tulostuu yhdelle riville.
Tämä tulostuu toiselle riville.
Numeroita erotettuna kenoviivoilla: 1\2\3
Numeroita erotettuna sarkainmerkeillä: 1  2     3
Numeroita erotettuna rivinvaihtomerkeillä:
1
2
3
```

Merkkijonojen yhdistäminen

Usein merkkijonoja joutuu yhdistämään toisiin merkkijonoihin tai numeroihin ja nämä molemmat tietysti onnistuvat, vaikka ne tehdäänkin eri tavoin. Heti alkuun on syytä huomata, että jos laitat kaksi merkkijonoa vierekkäin, Python yhdistää ne automaattisesti. Esimerkiksi merkkijonot ”`Vaa'an`” ” `alunen`” yhdistyvät tulkin tulostuksessa merkkijonoksi "`Vaa'an alunen`".

Tyypillinen tarve on yhdistää kaksi erillistä merkkijonoa yhdeksi merkkijonoksi. Kuten luvussa 1 oli puhetta, voidaan merkkijonoja yhdistä toisiinsa ”+”-operaattorilla. Ja ihan vastaavasti kuin matematiikassa monta summausoperaatiota voidaan hoitaa kertomerkillä, Pythonissa saman merkkijonon voi tulostaa monta kertaa ”*”-operaattorilla.

 # Esimerkki 2.2. Merkkijonojen yhdistäminen

```
Sana1 = "Ko"
Sana2 = "ralli"
print(Sana1 + Sana2)
print((Sana1 + "-") * 3 + Sana1 + Sana2)
```

Tuloste

```
Koralli
Ko-Ko-Ko-Koralli
```

Merkkijonon ja luvun yhdistäminen vastaa omenan ja appelsiinin yhdistämistä eli se ei onnistu, koska ne ovat erilaisia asioita eikä ole selvää, mitä lopputuloksen tulisi olla – omena, appelsiini vai hedelmäsalaatti. Toinen asia on, että nämä kaksi eri asiaa voidaan tulostaa yhdessä print-lauseessa ja toisaalta ne voidaan muuttaa samantyyppisiksi, jolloin ne voidaan yhdistää. Jos lukua ja merkkijonoa yrittää yhdistää toisiinsa print-lauseessa, tulee siihen tyypillisesti seuraavanlaisia kommentteja tulkilta:

```
>>> Ika = 3
>>> print(Ika + "-vuotias poika.")
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    print(Ika + "-vuotias poika.")
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>>
```

tai

```
TypeError: Can't convert 'int' object to str implicitly
```

Kun koodia on enemmän, myös virheilmoitus vaihtuu hieman. Molemmat kertovat kuitenkin samasta virheestä: merkkijonon ja numeron yhdistäminen ei onnistu.

Tarve on selkeä – haluamme tulostaa lauseen, jossa `Ika`-kokonaislukumuuttujaa seuraa tekstiä, nyt ”`-vuotias poika.`”-merkkijono. Ongelma voidaan ratkaista muuttamalla kokonaisluku merkkijonoksi `str`-funktiolla:

Esimerkki 2.3. Numeron ja merkkijonon yhdistäminen yhdeksi merkkijonoksi

```
Ika = 3
Lause = str(Ika) + "-vuotias poika."
print(Lause)
```

Tuloste

```
3-vuotias poika.
```

Muodostimme nyt yhden merkkijonomuuttujan `Lause`, joka alkaa merkkijonoksi muutetulla `Ika`-kokonaislukumuuttujan arvolla, jonka jatkona on merkkijono `"-vuotias poika."`. Seuraavalla rivillä tämä merkkijono tulostetaan näytölle. Tämä on tyypillinen ja suositeltava tapa yhdistää numeroita ja tekstiä tulostettaviksi lauseiksi. Palaamme tähän asiaan luvun lopussa Tulosteiden muotoilu –kohdassa, kun tutustumme `print`-käskyn ominaisuuksiin tarkemmin.

Merkkijonon pituus ja indeksit

Pythonin merkkijono on taulukko, jossa merkit ovat jokainen omalla paikallaan ja paikoilla on numerot eli *indeksit*. Alla olevassa merkkitaulukossa on merkkijono ”ApuVa” ja jokainen merkki on omassa kolossaan. Merkkijono sisältää 5 merkkiä, ensimmäinen merkki on ’A’, viimeinen merkki on ’a’ ja 3. ja 4. merkin sisältävä alimerkkijono on ”uV”. Jos tätä merkkijonoa ajattelee Python-koodina, ovat *ensimmäinen* ja *viimeinen* merkki samoja, mutta 3. ja 4. merkin sisältävä alimerkkijono olisi ”Va”. Äkkiseltään tämä ei kuulosta loogiselta, joten käydään läpi merkkijonon perusasiat Python-ohjelmassa.

```
+---+---+---+---+---+
| A | p | u | V | a |
+---+---+---+---+---+
```

Merkkijonon pituuden saa selville Pythonissa sisäänrakennetulla funktiolla `len` (length/pituus). Funktiolle annetaan syötteenä merkkijono tai muuttuja, ja funktio palauttaa pituuden merkkeinä esimerkin 2.4 mukaisesti.

Esimerkki 2.4. len-funktion käyttö

```
print(len("A"))

Merkkijono1 = "ApuVa"
print(len(Merkkijono1))

Merkkijono2 = "Apumiehensijaisentuuraajankorvaajanlomittajanpaikka"
Pituus = len(Merkkijono2)
print(Pituus)
```

Tuloste

```
1
5
51
```

Ensimmäisellä rivillä `len`-funktio selvittää merkkijonon ”A” pituuden ja `print`-funktio tulostaa sen. Toisessa kohdassa määritellään merkkijonomuuttuja, selvitetään muuttujassa olevan merkkijonon pituus ja tulostetaan se. Kolmannessa kohdassa määritellään merkkijonomuuttuja, selvitetään merkkijonomuuttujan pituus ja tallennetaan se kokonaislukumuuttujaan sekä lopuksi tulostetaan pituus-muuttujan arvo. Kuten ***Tuloste***-kohdasta näkyy, kaikki tavat johtavat pituuden tulostumiseen näytölle, mutta sisäisesti asiat on toteutettu eri tavoilla. Mikä näistä tavoista on paras ratkaisu, riippuu tilanteesta eli kannattaa lukea tehtävänanto ja tarkistaa, mitä pitää tehdä.

Edellä oli puhetta *ensimmäisestä* merkistä. Ensimmäinen merkki on taulukon ensimmäisessä paikassa, jonka indeksi Pythonissa on 0. Indeksoinnin aloittamista nollasta on perusteltu monilla tavoin. Jonkun mielestä indeksi kertoo siirtymän sanan alusta ja koska sanan ensimmäinen merkki on jo alussa, on siirtymä 0 ja siten indeksi on sama 0. Toinen perustelu on *matemaattinen kauneus* eli nolla on pienin ei-negatiivinen kokonaisluku. Käytännössä indeksoinnin alku 0:lla on ohjelmoinnissa tyypillinen tapa ja sama toistuu monissa muissa paikoissa. Ole aina tarkka *ensimmäisen* alkion kanssa ja varmista, onko sen indeksi 0 vai 1.

Viimeisen merkin kohdalla tulee myös olla tarkkana. Jos indeksointi alkaa luvusta 0 eikä 1, on viimeisen merkin indeksi (pituus - 1). Käytännössä `len`-funktio palauttaa merkkijonon pituuden ja viimeisen merkin indeksi taulukossa on sitä edeltävä kokonaisluku. Esimerkissä 2.4 `Merkkijono1`:n pituus oli 5 ja `Merkkijono2`:n pituus 51, joten näiden merkkijonojen viimeiset merkit saadaan indekseillä 4 ja 50, ts. `Merkkijono1[4]` ja `Merkkijono2[50]`.

Tiivistetysti merkkijonon yksittäisiin merkkeihin viitataan indekseillä. Indeksit voivat aluksi tuntua hankalilta, mutta niihin tottuu, kunhan niitä käyttää tarpeeksi monta kertaa. Yksi keino indeksien ja merkkijonon leikkausten lasketaan on ajatella numeroiden sijaan niiden välejä alla olevan esimerkin mukaisesti. Kuvan numerorivi kertoo merkin sijainnin laskettuna normaalisti vasemmalta oikealle.

```
+---+---+---+---+---+
| A | p | u | V | A |
+---+---+---+---+---+
0   1   2   3   4
```

Merkkijonojen leikkaukset

Merkkijonojen yhdistäminen onnistuu ”+”-operaattorilla, mutta niiden jakaminen ei onnistu ”-”-operaattorilla. Joskus merkkijonosta pitää kuitenkin ottaa kiinnostava kohta erilleen tarkempaa käsittelyä varten. Esimerkiksi suomalainen henkilötunnus sisältää paljon tietoa ja irrottamalla siitä sopivia osia saadaan selvitettyä ko. henkilön syntymävuosi, josta taas voidaan laskea ikä. Nämä alimerkkijonot eli merkkijonojen leikkaukset perustuvat merkkitaulukoiden indekseihin, joiden toiminta käytiin läpi edellä.

Merkkijonon leikkaukset määritellään hakasuluilla ja numerosarjalla, jossa ensimmäinen numero kertoo aloituspaikan, toinen lopetuspaikan ja kolmas siirtymävälin eli [alku:loppu:siirtymä]. Hakasulkujen sisällä numerot erotellaan toisistaan kaksoispisteillä. Kaikissa tilanteissa kaikkia kolmea indeksiä ei ole pakko käyttää, kuten alla olevat esimerkistä 2.5 näkyy.

Esimerkki 2.5. Merkkijonojen perusleikkaukset

```
Sana = "Teekkari"
print("Perusleikkauksia sanasta '"+Sana+"'.")
print("Yksi merkki indeksillä 3:", Sana[3])
print("Alusta 3 merkkiä:", Sana[0:3])
print("Keskeltä 4 merkkiä:", Sana[4:8])
print("Joka toinen merkki:", Sana[0:8:2])
print()

print("Leikkauksia oletusarvoilla")
print("Alusta 2 merkkiä:", Sana[:2])
print("Indeksistä 2 loppuun:", Sana[2:])
print("Askel 2 oletusarvon +1 sijasta:", Sana[0::2])
print("Kaikki oletusarvoilla:", Sana[::])
```

Tuloste

```
Perusleikkauksia sanasta 'Teekkari'.
Yksi merkki indeksillä 3: k
Alusta 3 merkkiä: Tee
Keskeltä 4 merkkiä: kari
Joka toinen merkki: Tekr

Leikkauksia oletusarvoilla
Alusta 2 merkkiä: Te
Indeksistä 2 loppuun: ekkari
Askel 2 oletusarvon +1 sijasta: Tekr
Kaikki oletusarvoilla: Teekkari
```

Indeksit voivat tuntua vaikeilta ja helposti tulee mieleen, että miksi asiat on tehty näin vaikeasti. Käyttämällä merkkijonoja ja leikkauksia enemmän toteutuksen edut tulevat vastaan ja mieli voi muuttua. Esimerkiksi leikkaus ”`Sana[:i] + Sana[i:]`” on sama kuin `Sana`.

Leikkausten yhteydessä kannattaa muistaa, mitä ne ovat eli leikkauksia olemassa olevasta merkkijonosta. Käytännössä tästä seuraa se, ettei leikkaukseen voi sijoittaa uusia merkkejä. Tämä ei kuitenkaan ole ongelma, sillä leikkauksilla voi irrottaa itseä kiinnostavia merkkejä ja alimerkkijonoja, ja muodostaa niistä uusia merkkijonoja tulosteiksi tai jatkokäsittelyyn esimerkin 2.6 mukaisesti.

Esimerkki 2.6. Merkkijonojen leikkaus ja yhdistely

```
Sana = "Teekkari"
print("Merkkijonojen leikkaus ja yhdistely")
print("Teekkari jaettuna indeksillä 2:", Sana[:2] + Sana[2:])
print("Teekkari jaettuna indeksillä 4:", Sana[:4] + Sana[4:])
print()

print("Merkkijonojen yhdistely")
print("Pen" + Sana[3:])
print(Sana[:3] + "hetki")
print()
```

Tuloste

```
Merkkijonojen leikkaus ja yhdistely
Teekkari jaettuna indeksillä 2: Teekkari
Teekkari jaettuna indeksillä 4: Teekkari

Merkkijonojen yhdistely
Penkkari
Teehetki
```

Leikkausten indeksit voivat olla negatiivisia. Käytännössä indeksit voi ajatella lukujanana, jossa mennään positiivisista luvuista nollaan ja jatketaan negatiivisilla luvuilla samalla tavoin kuin lämpömittari. Negatiiviset luvut helpottavat indeksien kanssa toimimista etenkin merkkijonon lopussa, sillä ne lasketaan merkkijonon lopusta. Myös siirtymäväli voi olla negatiivinen eli laskenta on lopusta alkuun.

Esimerkki 2.7. Negatiiviset indeksit

```
Sana = "Teekkari"
print("Negatiivisten indeksien käyttö")
print("Viimeinen merkki, ts. 1 merkki lopusta:", Sana[-1])
print("Toiseksi viimeinen merkki:", Sana[-2])
print("Lopusta 2 viimeistä merkkiä:", Sana[-2:])
print("Alusta loppuun ilman 2 viimeistä merkkiä:", Sana[:-2])
print()

Sana = "Robottikana"
print("Sana '"+Sana+"' takaperin eli lopusta alkuun '"+Sana[::-1]+"'.")
print("Merkkijonon joka toinen merkki lopusta alkaen:", Sana[::-2])
print("Merkkijonon indeksinä -0 on sama kuin 0:", Sana[-0])
```

Tuloste

```
Negatiivisten indeksien käyttö
Viimeinen merkki, ts. 1 merkki lopusta: i
Toiseksi viimeinen merkki: r
Lopusta 2 viimeistä merkkiä: ri
Alusta loppuun ilman 2 viimeistä merkkiä: Teekka

Sana 'Robottikana' takaperin eli lopusta alkuun 'anakittoboR'.
Merkkijonon joka toinen merkki lopusta alkaen: aaitbR
Merkkijonon indeksinä -0 on sama kuin 0: R
```

Kertauksena vielä yllä ollut kuva, jossa ylempi numerorivi kertoo kirjaimen sijainnin laskettuna vasemmalta oikealle ja alempi rivi negatiivisilla luvuilla laskettuna oikealta vasemmalle. Ja indeksiä -0 ei käytetä, koska se on sama kuin 0.

```
 +---+---+---+---+---+
 | A | p | u | V | A |
 +---+---+---+---+---+
 0   1   2   3   4   5
-5  -4  -3  -2  -1
```

Tyypillisiä virheilmoituksia

Merkkijonot ja niiden indeksit ovat Pythonissa varsin joustavia eli niillä voi tehdä monenlaisia asioita. Kaikki ei kuitenkaan ole mahdollista ja merkkijono-operaatioista voi tulla virheilmoituksia, vaikka joskus Python olettaa käyttäjän tarkoittavan jotain.

Merkkijonoalueen yli meneviä leikkauksia kohdellaan hienovaraisesti. Jos annettu numeroarvo ylittää merkkijonon rajat tai aloituspaikka on lopetuspaikkaa suurempi, tulee vastaukseksi tyhjä jono:

```
>>> sana[1:100]
'eekkari'
>>> sana[10:]
''
>>> sana[2:1]
''
```

Tämä ei koske tilannetta, jossa merkkijonosta otetaan yksittäinen merkki leikkauksen sijaan:

```
>>> sana[100]
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    sana[111]
IndexError: string index out of range
>>>
```

Ja kuten aiemmin oli puhetta, leikkauksiin ei voi sijoittaa uusia arvoja. Merkkijono on vakiotietotyyppi eikä muuttuja:

```
>>> sana[0] = 'x'
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: 'str' object doesn't support item assignment

>>> sana[:1] = 'Splat'
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: 'str' object doesn't support slice assignment
```

Lukujen tyyppimuunnokset ja pyöristys

Pythonissa on tyyppimuunnosfunktiota tietotyypin vaihtamiseen. Lähtökohta muunnoksille on luonnollisesti hyväksyttävä muutettava eli esim. numeroista koostuvia merkkijonoja voidaan muuttaa luvuiksi. Peruskäskyt ovat `int()`, `float` ja `str()`, jotka muuttavat annetun syötteen kokonaisluvuksi, desimaaliluvuksi ja merkkijonoksi. `int()` muuttaa syötteen kokonaisluvuiksi, mutta ei pyöristä vaan katkaisee luvun desimaaliosan pois. `float()` muuttaa syötteen desimaaliluvuksi, joka näkyy kokonaisluvun kohdalla siinä, että tulosteeseen tulee `.0` mukaan. `str()` muuttaa syötteen merkkijonoksi ja sitä käytetään erityisesti tulosteita tehdessä, jolloin luvut muutetaan ensin merkkijonoiksi ja sitten yhdistetään muihin merkkijonoihin esimerkin 2.8 mukaisesti.

Esimerkki 2.8. Tietotyyppimuunnokset

```
Merkkijono = "32"
Kokonaisluku1 = int(Merkkijono)
Kokonaisluku2 = int(3.99999)
Kokonaisluku3 = int(-267)

Desimaaliluku1 = float(Merkkijono)
Desimaaliluku2 = float(Kokonaisluku2)
Pii = float("3.14159")

Merkkijono2 = str(Kokonaisluku1)
Merkkijono3 = str(Pii)

Nimi = input("Anna nimi: ")
Syote = input("Anna lukumäärä: ")
Lukumaara = int(Syote)
print()

print("Merkkijono:", Merkkijono)
print("Merkkijonomuuttuja muutettuna kokonaisluvuksi:", Kokonaisluku1)
print("Desimaaliluku muutettuna kokonaisluvuksi:", Kokonaisluku2)
print("Negatiivinen kokonaisluku muutettuna kokonaisluvuksi:", Kokonaisluku3)
print("Merkkijonomuuttuja muutettuna desimaaliluvuksi:", Desimaaliluku1)
print("Kokonaisluku muutettuna desimaaliluvuksi:", Desimaaliluku2)
print("Merkkijono muutettuna desimaaliluvuksi:", Pii)
print("Kokonaisluku muutettuna merkkijonoksi:", Merkkijono2)
print("Desimaaliluku muutettuna merkkijonoksi:", Merkkijono3)
print("Nimi '"+Nimi+"' tulostettuna "+str(Lukumaara)+" kertaa:", Nimi*Lukumaara)
```

Tuloste

```
Anna nimi: Kalle
Anna lukumäärä: 3

Merkkijono: 32
Merkkijonomuuttuja muutettuna kokonaisluvuksi: 32
Desimaaliluku muutettuna kokonaisluvuksi: 3
Negatiivinen kokonaisluku muutettuna kokonaisluvuksi: -267
Merkkijonomuuttuja muutettuna desimaaliluvuksi: 32.0
Kokonaisluku muutettuna desimaaliluvuksi: 3.0
Merkkijono muutettuna desimaaliluvuksi: 3.14159
Kokonaisluku muutettuna merkkijonoksi: 32
Desimaaliluku muutettuna merkkijonoksi: 3.14159
Nimi 'Kalle' tulostettuna 3 kertaa: KalleKalleKalle
```

Huomaa, että tulosteista on mahdoton tietää, onko tuloste luku vai merkkijono, sillä niissä ei ole eroa. Laskentaoperaatiota tehdessä tietotyyppi kuitenkin paljastuu. Ja muutettaessa merkkijono luvuksi ei kirjainmerkkien muuttaminen luvuksi onnistu:

```
>>> int("Hello")
ValueError: invalid literal for int() with base 10: 'hello'
```

Tyyppimuunnoksiin liittyy läheisesti **lukujen pyöristys**. Käyttäjän kannalta ei ole mielekästä katsella pitkää desimaalijonoa ruudulla etenkin, jos pienempi määrä on informatiivisempi. Esimerkissä 2.9 näkyy pyöritykseen liittyvät perusasiat. Pyöristäminen tehdään `round()`-funktiolla, johon voidaan laittaa kaksi parametriä eli pyöristettävä desimaaliluku ja pilkun jälkeen kokonaisluku, joka kertoo halutun desimaalien määrän. Jos desimaalien määrää ei anneta, `round()` palauttaa kokonaisluvun. Muutoin palautettavassa luvussa on pyydetty määrä desimaaleja, mutta `round()` ei lisää turhia nollia tulokseen – lukuun ottamatta desimaaliluvun tunnisteena toimivaa `.0` -osaa.

Esimerkki 2.9. Lukujen pyöristäminen

```
Desimaaliluku1 = 234.5647292340234
Desimaaliluku2 = round(Desimaaliluku1)
Desimaaliluku3 = round(Desimaaliluku1, 0)
Desimaaliluku4 = round(Desimaaliluku1, 3)

Desimaaliluku5 = 3.14
Desimaaliluku6 = 1043.55
Osamaara = Desimaaliluku6 / Desimaaliluku5
Tulos1 = round(Osamaara, 2)
Tulos2 = round(2/1, 2)
Tulos3 = round(1/2, 2)
Tulos4 = round(1/3, 2)

print("Desimaaliluku pyöristettynä kokonaisluvuksi:", Desimaaliluku2)
print("Desimaaliluku pyöristettynä 0 desimaalille:", Desimaaliluku3)
print("Desimaaliluku pyöristettynä 3 desimaalille:", Desimaaliluku4)
print("Kahden desimaaliluvun osamäärä pyöristämättä:", Osamaara)
print("Kahden desimaaliluvun osamäärä 2 desimaalilla:", Tulos1)
print("Kahden kokonaisluvun osamäärä 2 desimaalilla:", Tulos2)
print("Kahden kokonaisluvun osamäärä 2 desimaalilla:", Tulos3)
print("Kahden kokonaisluvun osamäärä 2 desimaalilla:", Tulos4)
```

Tuloste

```
Desimaaliluku pyöristettynä kokonaisluvuksi: 235
Desimaaliluku pyöristettynä 0 desimaalille: 235.0
Desimaaliluku pyöristettynä 3 desimaalille: 234.565
Kahden desimaaliluvun osamäärä pyöristämättä: 332.34076433121015
Kahden desimaaliluvun osamäärä 2 desimaalilla: 332.34
Kahden kokonaisluvun osamäärä 2 desimaalilla: 2.0
Kahden kokonaisluvun osamäärä 2 desimaalilla: 0.5
Kahden kokonaisluvun osamäärä 2 desimaalilla: 0.33
```

Normaalien desimaalilukujen pyöristäminen on intuitiivista `round()`-funktiolla, mutta sen erityispiirteet on syytä huomata ja tarkistaa aina tarvittaessa. Näitä ovat kokonaisluvuksi pyöristäminen ja se, että `round()` laskee arvon sekä pyöristää se haluttuun laskennalliseen tarkkuuteen. `round()` ei kuitenkaan muokkaa tulosta haluttuun esitysasuun, esim. lisää nollia tulosteeseen. Palaamme tulostusasiaan myöhemmin, sillä tulosteiden muotoilu on monille tärkeä asia ja siksi siihen on olemassa nykyistä monipuolisempia työkaluja.

Tyypillisiä virheilmoituksia

Tyyppimuunnosfunktio samoin kuin pyöristysfunktio eivät muuta alkuperäistä tietoa vaan tekevät siitä uuden esitysmuodon. Tämä tarkoittaa sitä, että jos uutta esitysmuotoa halutaan hyödyntää myöhemmin, se pitää tallentaa muuttujaan. Esimerkki havainnollistaa tätä tyypillistä ongelmaa:

```
>>> Luku = "2323" # Alustetaan Luku merkkijonona
>>> int(Luku)
2323
>>> Luku + 1 # Muuttuja ei tyyppimuunnoksesta huolimatta ole numero, \
             koska sitä ei tallennettu mihinkään.
Traceback (most recent call last):
  File "<pyshell#19>", line 1, in -toplevel-
    Luku + 1
TypeError: Can't convert 'int' object to str implicitly
>>> Luku = int(Luku) # Tallennetaan muutos, jolloin Luku todella on \
                     numero
>>> Luku + 1 # Nyt lukuun voidaan lisätä
2324
>>>
```

Yllä tyyppimuunnoksen jälkeen tieto tallennetaan alkuperäiseen muuttujaan, mikä on tässä tapauksessa luonteva ratkaisu, koska muuttujan nimestä ”`Luku`” voidaan olettaa, että sillä voi laskea jotain matemaattisiin kaavoihin perustuen. Käytännössä tyyppimuunnettu arvo tulee sijoittaa muuttujan arvoksi sijoitusoperaattorilla (”=”).

Muuttujien roolit

Olemme puhuneet muuttujista ja nimensä mukaisesti muuttuja sisältää tietoa, joka voi muuttua. Muuttujilla on ohjelmoinnissa useita erilaisia käyttötarkoituksia, joita voidaan kutsu *rooleiksi*. Tutustumme keskeisimpiin muuttujien rooleihin, sillä niiden ymmärtäminen auttaa ymmärtämään ohjelmaa ja kirjoittamaan oikein toimivia ohjelmia. Olemme käyttäneet kaikkia nyt esiteltäviä muuttujien rooleja, jotka ovat tuoreimman säilyttäjä, kiintoarvo ja tilapäissäilö.

Esimerkissä 2.10 näkyvä muuttuja Nimi on *tuoreimman säilyttäjä* eli samaan muuttujaan luetaan ensin käyttäjän antama nimi ja sitten se tulostetaan – tämän jälkeen muuttujaa käytetään uudestaan samalla tavalla seuraavalle nimelle. Muuttuja siis säilyttää tuoreimman käyttäjän antaman arvona.

Esimerkki 2.10. Muuttujan rooli *tuoreimman säilyttäjä*

```
Nimi = input("Anna nimi: ")
print(Nimi)
Nimi = input("Anna toinen nimi: ")
print(Nimi)
```

Tuloste

```
Anna nimi: Kalle
Kalle
Anna toinen nimi: Ville
Ville
```

Esimerkissä 2.11 näkyvä muuttuja PII on *kiintoarvo*. Kiintoarvo säilyttää tiedon ilman muutoksia. Kiintoarvo saa arvonsa yhden kerran tyypillisesti heti ohjelman alussa ja se säilyttää saman arvon ohjelman loppuun asti. Teknisesti `PII` on Pythonissa muuttuja, mutta kiintoarvo-roolin takia arvoa ei tule muuttaa missään kohdassa ohjelmaa.

Esimerkki 2.11. Muuttujan rooli *kiintoarvo*

```
PII = 3.14159
print("Piin arvo on", PII)
```

Tuloste

```
Piin arvo on 3.14159
```

Esimerkissä 2.12 näkyy kolmas tyypillinen muuttujan rooli *tilapäissäilö*, jossa muuttuja on tilapäinen tiedon säilytyspaikka. Tilapäissäilö-muuttujia käytetään ymmärrettävyyden vuoksi ja ohjelma voidaan tehdä ilman niitä. Esimerkissä 2.12 `Syote` on tilapäissäilö, johon käyttäjän syöte tallennetaan myöhempää käyttöä varten eli kunnes se muunnetaan kokonaislukumuuttujaksi `Numero1`. `Numero2`:n arvo kysytään ja muutetaan kokonaisluvuksi yhdellä kertaa, joten tilapäissäilöä ei käytetä. Molemmat tavat ovat teknisesti toimivia, mutta etenkin useita eri vaiheita edellyttävien operaatioiden välitulosten tallennus tilapäissäilöihin auttaa varmistamaan ohjelman oikean toiminnan.

Esimerkki 2.12. Muuttujan rooli *tilapäissäilö*

```
Syote = input("Anna numero: ")
Numero1 = int(Syote)
print("Numero kerrottuna itsellään on", Numero1 * Numero1)

Numero2 = int(input("Anna numero: "))
print("Numero kerrottuna itsellään on", Numero2 * Numero2)
```

Tuloste

```
3
Numero kerrottuna itsellään on 9
2
Numero kerrottuna itsellään on 4
```

Tulevissa luvuissa tutustumme muutamaan muuhunkin rooliin. Näistä on huomautus aina, kun uusi rooli esitellään.

Tulosteiden muotoilu

Pythonissa tulostus tapahtuu `print`-käskyllä niin kuin kaikissa tähän asti tehdyissä ohjelmissa näkyy. Tosin tähän asti olemme käyttäneet vain `print`-käskyn perusmuotoa, ja käsky tarjoaa tähän mennessä nähtyä enemmän mahdollisuuksia tulosteiden muotoiluun.

Esimerkissä 2.13 näkyy, että `print`-käsky tulostaa sille suluissa annetun parametrin näytölle. Voimme siis tulostaa useita eri arvoja, parametreja, näytölle pilkuilla eroteltuina ja ne kaikki tulostetaan samalla riville välilyönnein eroteltuna. Toinen tähän mennessä nähty vaihtoehto tulostamiseen on muodostaa tulostettava merkkijono omana vaiheenaan ja sen jälkeen käyttää `print`-käskyä muodostetun merkkijonon tulostamiseen alla olevan mukaisesti.

Esimerkki 2.13. Tulosteiden muotoilu tähän asti

```
# print-lauseen parametrien tulostus
print(5)
Nimi = "Ville"
print(Nimi)
print("Moi", Nimi, 5, "vuotta.")

# Merkkijonon muodostus + tulostus print-lauseella
Tuloste = "Moi " + Nimi + ", " + str(5) + " vuotta."
print(Tuloste)
```

Tuloste

```
5
Ville
Moi Ville 5 vuotta.
Moi Ville, 5 vuotta.
```

`print`-käskyllä saa tulostettua kaikki halutut tiedot, mutta tuloste ei ole loppuun asti viimeistelty. Yllä olevassa tervehdyksessä pilkun laittaminen nimen perään ilman välilyöntiä ei onnistu tulostettaessa parametreja. Ongelma voidaan ratkaista esimerkiksi muodostamalla tulostettava merkkijono yhdistämällä sopivia merkkijonoja. Voidaan tietysti miettiä, onko yhdestä välilyönnistä eroon pääseminen kaiken tämän vaivan arvoista. Mutta jos maksava asiakas pyytää viimeisteltyä tulostusasua, niin silloin on hyvä olla keinoja siihen, ettei menetä rahoja ja sen lisäksi vielä asiakastakin.

`print`-käskyllä on nopea tulostaa arvoja parametreina eli erottamalla tulostettavat tiedot/muuttujat pilkuilla. Jos taas tarvitaan viimeisteltyä tulostusasua, kannattaa tuloste muodostaa merkkijono-operaatioilla ja käyttää `print`-lausetta valmiin merkkijonon tulostamiseen. Tällöin nämä kaksi tehtävää erotetaan toisistaan, jotta ne molemmat voidaan hoitaa tehokkaasti – ensinnäkin merkkijonon muodostaminen ja toiseksi sen tulostaminen. Merkkijonojen kanssa on tärkeää muistaa, että kaikki yhdistettävät osat ovat merkkijonoja eikä välissä ole lukuja.

`print`-käskyn muista ominaisuuksista kiinnostavia ovat erityisesti parametrit `end` ja `sep`. Niiden oletusarvot ovat vastaavasti `"\n"` ja `" "` eli rivinvaihtomerkki ja välilyönti. Käytännössä `print`-käsky tulostaa suluissa olevat parametrit näytölle eroteltuina `sep`-parametrin merkillä (ts. separator-merkki, kenttäerotin) ja lisää tulosteen perään `end`-parametrin osoittaman merkin (ts. rivin loppumerkki). Vaihtamalla näiden parametrien arvoja voidaan muuttaa tulosteen rakennetta alla olevan esimerkin 2.14 mukaisesti.

Esimerkki 2.14. Tulosteen muotoilua `end` ja `sep` parametreilla

```
# Esimerkin 2.13 lause tulostettuna sep-parametrin avulla
Nimi = "Ville"
print("Moi ", Nimi, ", ", 5, " vuotta.", sep="")
print("\n\n")

# Tekstin muotoilua rivin loppumerkillä ja kenttäerottimella
Nimi = "Maija"
print("Tekstiä")
print("Nimi on", Nimi)
print("Nimi on " + Nimi + ", joka onkin hyvä nimi.")
print("Tekstiä on tässä. ", end="")
print("Tämä jatkuu samalle riville.")
print()
print("Kenttäerottimella saa muokattua tulostetta: ")
print(1, 2, 3, 4, 5)
print(1, 2, 3, 4, 5, sep="-")
```

Tuloste

```
Moi Ville, 5 vuotta.



Tekstiä
Nimi on Maija
Nimi on Maija, joka onkin hyvä nimi.
Tekstiä on tässä. Tämä jatkuu samalle riville.

Kenttäerottimella saa muokattua tulostetta:
1 2 3 4 5
1-2-3-4-5
```

Kuinka se toimii

Ensimmäinen `print` tulostaa Ville-tekstin samalla tavoin kuin edellisessä esimerkissä, mutta nyt `sep`-parametrillä tulostetta muokaten. Huomaa, että Ville-rivin jälkeen on tyhjiä rivejä. Yhden tyhjän rivin saa tulostettua `print()` -käskyllä, mutta jos lasket tulosteessa olevien tyhjien rivien määrän, niitä onkin 3. Eli tulostettavassa merkkijonossa on 2 rivinvaihtomerkkiä ja `end`-parametrin oletuksena on yksi rivinvaihtomerkki, jolloin tulosteeseen tulee 3 rivinvaihtoa.

Tulostettaessa Maija-nimeä sen perään saadaan lisättyä seuraavaksi merkiksi pilkku ja sen jälkeen tekstissä on `end`-parametri, jolla saadaan seuraava tuloste samalle riville. Käytännössä avainsana `end=""` kertoo `print`-funktiolle, että rivin loppuun tuleva merkki on tyhjä ts. sitä ei ole. Oletusarvona käytössä oleva rivinvaihto (`"\n"`) poistuu ja seuraava rivi tulostuu edellisen perään. `end`-avainsanalla voidaan määrittää mikä tahansa merkkijono rivin loppuun `print`-lausekkeessa, myös nyt käytössä oleva tyhjä merkkijono. `print`-lauseen toinen avainsana on `sep` eli separator. Yllä olevan esimerkin lopussa näkyy, miten oletusarvona olevan välilyönnin korvaaminen tavuviivalla näkyy tulosteessa.

Yleensä eri tapoja ei kannata käyttää ristiin, sillä lopputulos näyttää epäselvältä ja siinä on helpolla virheitä. Näissä esimerkeissä katsottiin eri vaihtoehtojen toimintaa ja tuotettiin viimeisteltyä tekstiä, joten esimerkin vuoksi niissä käytettiin useita eri tapoja tekstin muokkaamiseen. Erityisesti tulostettaessa paljon ja jos tulostetaan numeroita riveillä ja sarakkeilla, ovat ylimääräisten rivinvaihtojen poisto ja lisäys usein tarpeellisia apuvälineitä. Usein samaan lopputulokseen pääsee monilla eri tavoilla, kunhan osaa käyttää näitä työkaluja oikein.

Perusohjelman rakenne

Tehtävien ohjelmien pituus rupeaa kasvamaan ja silloin on syytä alkaa miettiä, miten ohjelman rakenne pidetään selkeänä ja ymmärrettävänä. Tässä vaiheessa meillä on vain peräkkäin suoritettavia käskyjä, joten voimme nyt ryhmitellä samanlaisia käskyjä yhteen mahdollisuuksien mukaan. Alla on tässä vaiheessa tehtäville ohjelmille sopiva perusrakenne, joka kehittyy ja muuttuu kurssin edetessä uusien rakenteiden myötä:

1. Muuttujien ja kiintoarvojen alustus
2. Tiedon lukeminen
3. Tiedon käsitteleminen
4. Tiedon tulostus

Esimerkissä 2.15 näkyy tällä pohjalta muodostuva ohjelma. Huomaa ohjelman alussa oleva kommenttirivi, jossa on ohjelman perustiedot eli sen tekopäivämäärä, tekijä ja tiedoston nimi. Ohjelman lopussa on ”Kiitos ohjelman käytöstä.” -tuloste, joka kertoo käyttäjälle, että ohjelma on loppunut eikä käyttäjän tarvitse miettiä, miksi ohjelma pysähtyi. Vastaavasti kooditiedoston viimeisenä rivinä on kommentti ”EOF” eli End Of File, joka kertoo ohjelmoijalle, että tämä on ohjelman viimeinen rivi eikä sen jälkeen ole enää mitään.

Esimerkki 2.15. Perusohjelman rakenne ja käytettyjen muuttujien roolit

```
# 20230912 un E2_15.py (c) LUT
# Alustukset - kiintoarvo
AANI_NOPEUS = 340
##################################
# Tiedon lukeminen – tuoreimman säilyttäjä
Syote = input("Anna kulunut aika: ")
Aika = int(Syote)
##################################
# Tiedon käsitteleminen - tilapäissäilö
Etaisyys = Aika * AANI_NOPEUS
Etaisyys = round(Etaisyys / 1000, 2)
##################################
# Tiedon tulostus
print("Salama löi " + str(Etaisyys) + " km:n päässä.")
print("Kiitos ohjelman käytöstä.")
##################################
# EOF
```

Tuloste

```
Anna kulunut aika: 3
Salama löi 1.02 km:n päässä.
Kiitos ohjelman käytöstä.
```

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 käskyt on hyvä muistaa ja tunnistaa, sillä seuraavaksi teemme niihin perustuvia ohjelmia.

- Merkkijonot
  - lainausmerkit, heittomerkit, ohjausmerkit \\, \t, \n (E2.1)
  - yhdistäminen +, * (E2.2, E2.3)
  - pituus len(), indeksit, leikkaukset [::] (E2.4, E2.5, E2.6, E2.7)

- Kokonaisluku, desimaaliluku, merkkijono, tyyppimuunnokset, pyöristys
    - int(), float(), str(), round() (E2.8, E2.9)
- Muuttujien roolit: tuoreimman säilyttäjä, kiintoarvo, tilapäissäilö (E2.10, E2.11, E2.12)
- Tulosteiden muotoilu: print, end, sep (E2.13, E2.14)
- Perusohjelma (E2.15)
    - Rakenne: alustus, luku, käsittely, tulostus
    - Lopetus: kiitos ohjelman käytöstä, eof
- Keskeiset käskyt, operaattorit ja parametrit
    - print(), sep, end, str(), int(), float(), len(), [::], round()

Palaamme tulostukseen vielä kerran myöhemmin kurssilla. Tulosteiden muodostaminen on niin tärkeä asia ohjelmoinnissa, että tarvitsemme myöhemmin nykyistä tehokkaampia työkaluja asiaan.

Pienen Python-perusohjelman tyyliohjeet

Tähän lukuun liittyvät tyyliohjeet on koottu alle. Niin kuin aina ohjelmoinnissa, nämä ovat yleisiä ohjeita ja erikoistapauksista yleisistä ohjeista voi ja pitää poiketa perustellusta syystä.

1. Käytä merkkijonojen tunnisteena lainausmerkkiä (”), ei heittomerkkiä (’).
2. Tulosta yksi rivi yhdellä `print`-käskyllä. Älä jaa yhden `print`-käskyn tulostetta useille riveille.
3. Mikäli tulostettava merkkijono muodostuu monista osista, muodosta se ennen tulostamista ja tulosta valmis merkkijono `print`-käskyllä.
4. Tulosteissa rivinvaihtomerkit tulee olla merkkijonon lopussa tai omina erillisinä käskyinä. Älä piilota rivinvaihtomerkkejä merkkijonon alkuun tai muiden merkkien sekaan.
5. Kiintoarvot kirjoitetaan suuraakkosilla, esim. `PII = 3.14159, CM_TUUMA = 2.54`. Kiintoarvoja tulee käyttää luonnonvakioiden ja muunnoskertoimien määrittelyyn.
6. Ohjelmassa ei tule olla ylimääräisiä koodirivejä, jotka eivät tee mitään. Tällaisia ovat mm. seuraavat:
    a. Peräkkäiset `int()` ja `str()` -käskyt, jotka kumoavat toisensa.
    b. Tyyppimuunnokset kuten `int(123)` ja `str(input(…))`, joilla ei ole vaikutusta.
7. Käytä vain oppaassa tähän mennessä käsiteltyjä käskyjä ja rakenteita.

Tässä oppaassa ei opeteta kaikkien Python-kielen käskyjen käyttöä. Tulostamiseen liittyen Pythonissa on mm. f-string, jota ei käytetä ollenkaan ja format-käsky, johon palataan myöhemmin oppaassa. Näitä saa käyttää, mutta vastuu ohjelman selkeydestä ja käskyjen oikeasta käytöstä on rakenteen käyttäjällä, eikä kurssihenkilöstö neuvo näiden käytössä. Esimerkiksi tällä viikolla lukujen pyöristys liittyy matemaattiseen arvoon ja tulostuksen tarkkuus perustuu siihen. Palaamme tulostusasun tarkempaan muotoiluun myöhemmin ja tässä vaiheessa muiden kuin tässä luvussa esitettyjen toimintojen käyttö voi aiheuttaa ongelmia.

Luvun asiat kokoava esimerkki

Esimerkissä 2.16 näkyvä ohjelma selvittää iän vuoden tarkkuudella henkilötunnuksesta olettaen, että käyttäjä on syntynyt 2000-luvulla. Ohjelmassa näkyy kiintoarvojen alustus, tiedon kysyminen, merkkijonoleikkauksia, tyyppimuunnoksia, tiedon käsittely ja tulostus. Kiintoarvojen käyttö vähentää koodissa olevien numeroiden määrää ja helpottaa niiden ylläpitämistä, kun ne on koottu keskitetysti yhteen paikkaan. Ohjelma muuttaa kaikki numerot kokonaisluvuiksi systemaattisuuden vuoksi, vaikka vain vuosi-osalla tehdään laskuja. Tämä muutos näkyy tulosteessa, joka on 12.3. eikä 12.03. Pääsemme ohjelman rajoitteista eroon myöhemmin, mutta tässä vaiheessa meillä on näiden rajoitteiden puitteissa toimiva ohjelma.

Esimerkki 2.16. Perusohjelma

```
# 20230912 un E2_16.py (c) LUT
# Alustukset
###################################
INDEKSI_PAIVA = 0
INDEKSI_KUUKAUSI = 2
INDEKSI_VUOSI = 4
VUOSI_ALKU = 2000
VUOSI_NYT = 2023

###################################
# Tiedon lukeminen
Henkilotunnus = input("Anna henkilötunnuksesi: ")

###################################
# Tiedon käsitteleminen
Merkit = Henkilotunnus[INDEKSI_PAIVA:INDEKSI_KUUKAUSI]
Paiva = int(Merkit)
Merkit = Henkilotunnus[INDEKSI_KUUKAUSI:INDEKSI_VUOSI]
Kuukausi = int(Merkit)
Merkit = Henkilotunnus[INDEKSI_VUOSI:INDEKSI_VUOSI+2]
Vuosi = VUOSI_ALKU+int(Merkit)

###################################
# Tiedon tulostus
print("Olet syntynyt "+str(Paiva)+"."+str(Kuukausi)+". vuonna "+str(Vuosi)+".")
print("Olet nyt noin "+str(VUOSI_NYT - Vuosi)+" vuotta vanha.")
print("Kiitos ohjelman käytöstä.")

###################################
# EOF
```

Tuloste

```
Anna henkilötunnuksesi: 120302A213K
Olet syntynyt 12.3. vuonna 2002.
Olet nyt noin 21 vuotta vanha.
Kiitos ohjelman käytöstä.
```