LUT Python ohjelmointiopas 2023 - osa 3
Sisällysluettelo

Luku 3: Valintarakenne 1
Ehdollinen koodi 1
Sisennys eli koodilohko 1
Ehtolauseke, totuusarvo, loogiset operaattorit ja suoritusjärjestys 2
if-valintarakenne 7
Yleisiä huomioita valintarakenteesta 9
Valintarakenteen tyypillisiä sovelluskohteita 11
Yhteenveto 14

Luku 3: Valintarakenne

Tähän asti olemme tehneet ohjelmia, joissa on ollut aina joukko samassa järjestyksessä suoritettavia peräkkäisiä käskyjä. Entäpä, jos haluaisimme koodin tekevän vertailuja tai laittaa mukaan osioita, jotka ajetaan ainoastaan tarvittaessa? Esimerkiksi miten tekisimme ohjelman, joka sanoo ”Hyvää huomenta” tai ”Hyvää iltaa” kellonajan perusteella?

Kuten varmaan arvaat, tarvitaan tässä vaiheessa koodin tekemiseen ohjausrakenteita. Python käyttää kolmea ohjausrakennetta, `if`, `for` ja `while`, joista tässä luvussa tutustutaan ensimmäiseen eli if-valintarakenteeseen, jonka avulla voimme luoda ”ehdollista” koodia.

Ehdollinen koodi

Tähän asti ohjelmamme ovat suorittaneet kaikki ohjelmarivit ohjelman alusta loppuun asti. Ehdollinen koodi tarkoittaa sitä, ettei kaikkia rivejä suoriteta, vaan tietyt rivit suoritetaan vain nimetyn ehdon toteutuessa. Esimerkin 3.1 ohjelma sisältää yhden ehdollisesti tehtävän tulosteen. Ohjelma kysyy käyttäjältä luvun ja muuttaa sen kokonaisluvuksi, jonka jälkeen ohjelma vertaa annettua lukua lukuun 1 ja mikäli nämä ovat yhtä suuria, ohjelma tulostaa tekstin ”Luku on 1.”. Ohjelman päätteeksi se tulostaa aina standardifraasiin *Kiitos ohjelman käytöstä.* Jos käyttäjä antaa jonkun muun luvun kuin 1, jää ehdollinen tulostus tekemättä eli tekstiä ”Luku on 1.” ei tulosteta esimerkin Suoritus 2 mukaisesti.

Esimerkki 3.1. Ehdollinen koodi

```
Luku = int(input("Anna luku: "))

if (Luku == 1):
    print("Luku on 1.")

print("Kiitos ohjelman käytöstä.")
```

Tuloste

Suoritus 1:

```
Anna luku: 1
Luku on 1.
Kiitos ohjelman käytöstä.
```

Suoritus 2:

```
Anna luku: 2
Kiitos ohjelman käytöstä.
```

Esimerkki 3.1 on yksinkertaisin muoto ehdollisesta koodista eli if-valintarakenteesta. Valintarakenteen käytössä on useita huomioitavia asioita ja siitä on olemassa useita erilaisia muotoja, joten käydään ne läpi seuraavaksi.

Sisennys eli koodilohko

Esimerkin 3.1 kohdalla herää kysymys, mistä ohjelma tietää, mitkä koodirivit suoritetaan ehdollisesti? Pythonissa **tyhjät merkit (whitespace) rivin alussa ovat merkitseviä merkkejä**. Tätä sanotaan **sisennykseksi**. Nämä rivin alun ”tyhjät merkit” eli välilyönnit ja tabulaattori-merkit määrittelevät loogisen rivin sisennyksen syvyyden, jolla määritellään mihin loogiseen joukkoon rivi kuuluu. Tämä tarkoittaa sitä, että loogisesti samaan ryhmään kuuluvat koodirivit sijoitetaan samalle sisennystasolle. Sisennystasoa, joka taas sisältää

loogisen Python-käskyn/käskyjä, sanotaan osioksi tai koodilohkoksi. Esimerkin 3.1 ehdollisesti suorittava koodirivi on sisennetty ja jos tähän ehtoon liittyen tulisi suorittaa useita käskyjä, tulisi ne kaikki sisentää samalle tasolle.

Aloitteleva koodari ei aina ymmärrä välilyöntien tärkeyttä ja voi tehdä koodausvirheitä:

```
Arvo = 5
  print("Arvo on", Arvo) # Rivin alussa on välilyöntejä eli koodausvirhe
print("Arvo siis on", Arvo)
```

Jos tallennat tiedoston nimellä whitespace.py ja yrität ajaa sen, saat virheilmoituksen:

```
  File "whitespace.py", line 2
    print("Arvo on", Arvo) # Rivin alussa on välilyöntejä eli koodausvirhe
    ^
IndentationError: unexpected indent
```

Vaihtoehtoisesti ohjelma saattaa myös näyttää “syntax error” –ikkunan, mutta edelleen tietokone kieltäytyy suorittamasta kirjoittamaasi koodia. Virheen syynä on toisen rivin alussa olevissa ylimääräisissä välilyönneissä. *Syntax error* tarkoittaa sitä, että Python-koodissa on jotain kieliopin vastaista, eikä tulkki pysty tulkitsemaan koodia yksikäsitteisesti. Yllä oleva *IndentationError* kertoo sisennyksen olevan pielessä. Käytännössä sinun tulee muistaa tästä osiosta se, että **et voi aloittaa koodiriviä satunnaisesta paikasta vaan sinun on noudatettava sisennysten ja osioiden kanssa loogista rakennetta**.

Python-tulkki tarkastaa, että yhden koodilohkon rivit on aina sisennetty samalle tasolle. Monissa muissa ohjelmointikielissä koodilohko tunnistetaan koodilohkon alku- ja loppumerkeistä kuten esim. C- ja Java-kielissä. Pythonissa sisennyksiin on kuitenkin otettu tiukka kanta, jotta ohjelmista saataisiin helpommin luettavia ja ymmärrettäviä. Tämän vuoksi esim. C-kielessä on erillisiä ohjelmia, joilla voidaan muokata ohjelmista systemaattisesti oikein sisennettyjä. Ohjelmien lukemisen ja ymmärtämisen helpottamiseksi kannattaa siis muistaa aiemmin puhuttu loogisten ja fyysisten rivien synkronointi eli yksi rivi ja yksi asia sekä systemaattiset koodin sisennykset.

Sisennyksissä ei kannata käyttää välilyöntejä ja tabulaattoreita sekaisin. Kaikki editorit eivät käsittele välilyöntejä ja tabulaattoreita samalla tavalla, jolloin siirrettäessä koodia editorista toiseen, esim. koodin automaattitarkastajaan, voivat sisennykset mennä rikki ja johtaa em. virheisiin. Siksi kannattaa käyttää editoria, joka laskee yhden tabulaattorimerkin neljäksi välilyönniksi – kuten esimerkiksi IDLE tekee – ja käyttää sisennyksissä ainoastaan tabulaattorimerkin pituisia välejä tasolta toiselle siirryttäessä. Koodin ymmärrettävyys on edelleen tärkeä asia, sillä Python-tulkki tarkastaa sisennyksen määrän, mutta se ei pysty tarkastamaan, onko kaikki koodirivit sisennetty loogisesti oikein. Tämä jää ohjelmoijan vastuulle.

Ehtolauseke, totuusarvo, loogiset operaattorit ja suoritusjärjestys

Ehtolauseke ja totuusarvo

Esimerkin 3.1 if-valintarakenteen sisennys käsiteltiin edellä ja toinen uusi asia rakenteessa oli ehtolauseke eli ”jos Luku on yhtä suuri kuin 1” -osio.

```
if (Luku == 1):
```

Tässä ehtolausekkeessa on muuttujan Luku ja numeron 1 vertailu eli ovatko ne yhtä suuria, ja jos näin on, suoritetaan seuraavassa koodilohkossa oleva tulostuskäsky. Käytännössä `if`-lause arvioi suluissa olevan ehtolausekkeen totuusarvon ja jos se on tosi, suoritetaan seuraava koodilohko. Koska totuusarvoja on vain kaksi eli TOSI ja EPÄTOSI (True/False), on koodilohkon suorittamisesta päättäminen näennäisen helppoa. Käytännössä ehtolausekkeet voivat olla moninaisia ja moniosaisia. Ehtolausekkeiden ytimenä on

tyypillisesti matematiikasta tutut vertailuoperaatiot suurempi, pienempi ja yhtäsuuruus sekä niiden variantit sekä sisällytysoperaattori `in`, ks. Taulukko 3.1. Edelleen esimerkin 3.1 tapauksessa muuttujan Luku arvolla 1 ehtolauseke selvittää, onko 1 yhtä suuri kuin 1, päätyy totuusarvoon TOSI ja tulostaa koodilohkon sisältämän tulosteen.

Taulukko 3.1. Vertailuoperaattorit

Operaattori	**Nimi**	**Selite**	**Esimerkki**
<	Pienempi kuin	Palauttaa tiedon onko x pienempi kuin y. Vertailu palauttaa arvon `False` tai `True`.	`5 < 3` palauttaa arvon `False` `3 < 5` palauttaa arvon `True`.
>	Suurempi kuin	Palauttaa tiedon onko x suurempi kuin y.	`5 > 3` palauttaa arvon `True`. Jos molemmat operandit ovat numeroita, ne muutetaan automaattisesti vertailukelpoisiksi eli samantyyppisiksi. Merkkijonoista verrataan ensimmäisiä (vasemmanpuolimmaisia) kirjaimia merkistön mukaisessa järjestyksessä. Tähän palataan luvussa 12.
<=	Pienempi tai yhtä suuri	Palauttaa tiedon onko x pienempi tai yhtä suuri kuin y.	`3 <= 6` palauttaa arvon `True`.
>=	Suurempi tai yhtä suuri	Palauttaa tiedon onko x suurempi tai yhtä suuri kuin y.	`4 >= 3` palauttaa arvon `True`.
==	Yhtä suuri kuin	Testaa ovatko operandit yhtä suuria.	`2 == 2` palauttaa arvon `True`. `"str" == "stR"` palauttaa arvon `False`. `"str"== "str"` palauttaa arvon `True`.
!=	Erisuuri kuin	Testaa ovatko operandit erisuuria.	`2 != 3` palauttaa arvon `True`.
in	Sisältää	Testaa onko osa joukkoa	`"i" in "aeiouyääö"` palauttaa arvon `True`.
not in	Ei sisällä	Testaa ettei ole osa joukkoa	`"i" not in "aeiouyääö"` palauttaa arvon `False`.

Vertailuoperaattoreilla voidaan vertailla 2 operandia kerrallaan, esim. Luku1 ja Luku2, sekä selvittää niiden suuruuteen ja suhteeseen liittyviä tietoja Taulukon 3.1 mukaisesti. Sisällytysoperaattori `in` sopii tässä vaiheessa merkkijonoille, mutta sille on enemmän käyttöä myöhemmin. Merkkijonovertailun perusteisiin palataan myöhemmin, mutta esim. ”a” on ennen ”b”:tä eli ”a” < ”b”. Pienet kirjat ovat ennen isoja, joten isoilla ja pienillä kirjaimilla on merkitystä ja tässä vaiheessa kannattaa vertailla vain samalla tavalla kirjoitettuja merkkijonoja. Esimerkki 3.2 esittelee tyypillisiä ehtolausekkeita ja vertailuja.

Esimerkki 3.2. Ehtolauseke, vertailu ja joukko-operaattori

```
print("Ehtolauseke:")
print("True == 1:", True == 1)
print("False == 0:", False == 0)
print("1 == 2:", 1 == 2)
print("1 != 2:", 1 != 2)
print("1 < 2:", 1 < 2)
print()

Luku1 = 3
Luku2 = 4
print("Vertailu yhteensopivilla muuttujilla, Luku1="
      +str(Luku1)+" ja Luku2="+str(Luku2)+":")
print("Luku1 == Luku2:", Luku1 == Luku2)
print("Luku1 > Luku2:", Luku1 > Luku2)
print()

print("Merkkijonovertailut vastaavalla tavalla:")
print("'eka' == 'toka':", "eka" == "toka")
print("'eka' != 'toka':", "eka" != "toka")
print("'eka' > 'toka':", "eka" > "toka")
print()

print("Joukko-operaattori in:")
if ("a" in "apina"):
    print("Kirjain a esiintyy merkkijonossa 'apina'.")
```

Tuloste

```
Vertailulauseke:
True == 1: True
False == 0: True
1 == 2: False
1 != 2: True
1 < 2: True

Vertailu yhteensopivilla muuttujilla, Luku1=3 ja Luku2=4:
Luku1 == Luku2: False
Luku1 > Luku2: False

Merkkijonovertailut vastavalla tavalla:
'eka' == 'toka': False
'eka' != 'toka': True
'eka' > 'toka': False

Joukko-operaattori in:
Kirjain a esiintyy merkkijonossa 'apina'.
```

Loogiset operaattorit

Loogisilla operaattoreilla, Taulukko 3.2, voidaan yhdistää eri totuusarvoja ja siten myös useita erillisiä ehtolausekkeita. Kahden ehdon tapauksessa voidaan vaatia, että molempien ehtojen pitää olla tosia (AND) tai riittää, että toinen on tosi (OR). Yksittäisen arvon vastakohta saadaan NOT operaattorilla. Mikäli ehtoja olisi useita, esim. 10, katsotaan jokaisen ehdon totuusarvo erikseen ja sen jälkeen yhdistetään ne AND/OR -operaattoreilla. Jotta yhdistäminen menisi oikein, kannattaa vertailtavat asiat laittaa sulkuihin ja katsoa aina kahta ehtoa yhdellä kertaa, ts. ovat ne molemmat tosia (AND) vai riittääkö, että toinen on tosi (OR). Näitä yhdistelemällä voidaan selvittää monimutkaistenkin ehtojen perusteella ehtolausekkeen arvo, joka on siis aina tosi tai epätosi. Nämä loogiset operaattorit ovat toiselta nimeltään Boolen operaattoreita ja niillä muodostettuja lauseita kutsutaan Boolen algebraksi tai Boolen logiikaksi. Lisätietoa niistä löytyy esim. Wikipediasta. Tällä kurssilla meidän pitää ajoittain yhdistellä useita ehtoja Esimerkin 3.3 mukaisesti.

Taulukko 3.2 Loogiset operaattorit

Operaattori	Nimi	Selite	Esimerkki
not	Boolean NOT	Jos x on `True`, palautuu arvo `False`. Jos x on `False`, palautuu arvo `True`.	`not True` palauttaa arvon `False`.
and	Boolean AND	`x and y` palauttaa arvon `False` jos x on `False`, muulloin palauttaa y:n totuusarvon.	`False and True` palauttaa `False`. Tässä tapauksessa Python ei edes testaa jälkimmäistä arvoa, koska se tietää varman vastauksen. Tätä sanotaan pikatestaukseksi, ja se lyhentää laskenta-aikaa.
or	Boolean OR	`x or y` palauttaa `True`, jos x on `True`, muulloin palauttaa y:n totuusarvon.	`True or False` palauttaa `True`. Yllä olevan kohdan maininta pikatestauksesta pätee myös tässä.

Esimerkki 3.3. Useiden ehtojen yhdistäminen, Boolen logiikka/algebra

```
Ehto1 = True
Ehto2 = False

print("Ehtolausekkeita, Ehto1="+str(Ehto1)+" Ehto2="+str(Ehto2)+":")
if (Ehto1 and Ehto2):
    print("Ehto1 ja Ehto2 tosia")

if (Ehto1 or Ehto2):
    print("Ehto1 tai Ehto2 tosi, tai molemmat tosia")
print()

# Useista vertailuista koostuva ehtolauseke - huomaa sulut
Ika = 17
Kello = 22
print("Ehtolausekkeita, Ika="+str(Ika)+" Kello="+str(Kello)+":")
if ((Ika < 18) and (Kello > 21)):
    print("Olet alle 18v ja kello on yli 21, joten et voi tulla sisälle.")
```

Tuloste

```
Ehtolausekkeita, Ehto1=True Ehto2=False:
Ehto1 tai Ehto2 tosi, tai molemmat tosia

Ehtolausekkeita, Ika=17 Kello=22:
Olet alle 18v ja kello on yli 21, joten et voi tulla sisälle.
```

Esimerkissä 3.3 kannattaa huomata, että käytetyillä Ehto1ja Ehto2 arvoilla ensimmäinen `if`-lauseke ei tulosta mitään. Vaihtamalla muuttujien arvoja, vaihtuvat myös tulostettavat asiat. Jälkimmäisessä kohdassa ehdot on kirjoitettu `if`-lauseeseen, joka on tyypillinen tapa ohjelmoinnissa. Tällöin on syytä käyttää sulkuja, jotta operaattoreilla yhdistettävät ehdot ovat yksikäsitteisiä ja selkeästi nähtävillä. Monimutkaisissa ehtolausekkeissa ehtojen arviointi erikseen ja tuloksen kirjoittaminen esimerkin mukaisesti omiin tilapäismuuttujiin helpottaa oikein toimivan ohjelman tekemistä.

Operaattoreiden suoritusjärjestys

Jos sinulla on vaikkapa lauseke `2 + 3 * 4`, niin suorittaako Python lisäyksen ennen kertolaskua? Jo peruskoulumatematiikan mukaan tiedämme, että kertolasku tulee suorittaa ensin, mutta kuinka Python tietää siitä? Tämä tarkoittaa sitä, että kertolasku on suoritusjärjestyksessä ennen yhteenlaskua.

Taulukossa 3.3 on listattuna Pythonin operaattorit niiden suoritusjärjestyksen mukaisesti

ensimmäisestä viimeiseen. Tämä tarkoittaa sitä, että useita operaattoreita sisältävässä lausekkeessa niiden toteutusjärjestys on listan mukainen.

Taulukko on oiva apuväline esimerkiksi testauslausekkeita tarkastettaessa, mutta käytännössä kannattaa käyttää sulkeita laskujärjestyksen varmistamiseen. Esimerkiksi `2 + (3 * 4)` on lukijan kannalta paljon selkeämpi kuin `2 + 3 * 4`. Sulkeiden kanssa kannattaa kuitenkin käyttää järkeä ja välttää turhien sulkeiden käyttöä, koska se haittaa laskutoimituksen luettavuutta ja ymmärrettävyyttä. Esimerkiksi laskutoimitus `(2 + ((3 + 4) - 1)+(3))` on hyvä esimerkki siitä, mitä ei pidä tehdä.

Valintarakenteessa kannattaa laittaa jokainen vertailulauseke sulkujen sisään. Näin jokaisen sulkulausekkeen totuusarvon voi selvittää helposti ja tarpeen tullen Boolen operaattoreilla yhdistetyn lausekkeen tulos on helppo katsoa niiden pohjalta. Ja kun `if`-lauseen perässä on aina yhdet sulut ennen kaksoispistettä, tulee ohjelmasta selkeä ja ymmärrettävä sekä tulkille että käyttäjälle, ts. `if ((Ehto1) and (Ehto2) and (Ehto3)):`.

Taulukko 3.3. Operaattoreiden suoritusjärjestys

Operaattori	Selite	
`sisältö, ...`	Merkkijonomuunnos	**korkea prioriteetti**
{avain:tietue ...}	Sanakirjan tulostaminen	
[sisältö, ...]	Listan tulostaminen	
(sisältö, ...)	Tuplen luominen tai tulostaminen	
f(argumentti ...)	Funktiokutsu	
x[arvo:arvo]	Leikkaus	
x[arvo]	Jäsenyyden haku	
x.attribuutti	Attribuuttiviittaus	
****	**Potenssi**	
+x, -x	Positiivisuus, negatiivisuus	
***, /, %**	**Kertominen, jakaminen ja jakojäännös**	
+, -	**Vähennys ja lisäys**	
<, <=, >, >=, !=, ==	**Vertailut**	
is, is not	Identiteettitesti	**matala prioriteetti**
in, not in	Jäsenyystesti	
not x	Boolean NOT	
and	Boolean AND	
or	Boolean OR	

Listalla on operaatioita, joita et vielä tunne. Niihin palataan myöhemmin.

Operaattorit arvioidaan yleisellä tasolla vasemmalta oikealle, mikäli niiden suoritusjärjestys on samaa tasoa. Esimerkiksi `2 + 3 + 4` on käytännössä sama kuin `(2 + 3) + 4`. Jotkin operaattorit, esim. sijoitusoperaattori, toimivat oikealta vasemmalle, jolloin lauseke `a = b = c` tulkitaan olevan `a = (b = c)`.

if-valintarakenne

Palataanpa takaisin asiaan eli `if`-valintarakenteeseen, nyt kun koodilohkojen ja ehtolausekkeiden toiminta on tuttua. Tähän asti olemme tutustuneet vain `if`-rakenteen yksinkertaisimpaan muotoon eli ”jos niin” osaan ja kuten edellä huomasimme, voi käyttäjä jäädä ihmettelemään mitä ohjelmassa tapahtui kun ehdollinen osa jäi suorittamatta. Ehdollinen koodi on hyvä työkalu, muttei sovi kaikkiin tarpeisiin ja siksi sitä voidaan laajentaa tarpeen mukaan. Usein väittämään liittyy kaksi vaihtoehtoista toimintoa, joista ensimmäinen tehdään väittämän ollessa tosi (”jos tosi niin toimi näin”) ja väittämän ollessa epätosi tehdään toinen toiminto (”muutoin toimi näin”). Tämä toinen osio toteutetaan Pythonissa `else`-osiona eli `if...else...`. Tämäkään ei aina riitä, vaan joskus tarvitaan vielä useampi vaihtoehto tilanteeseen sopivaan toimintaan. Näitä varten valintarakenteessa voi olla myös `elif` (else if)-osioita, joilla useita `if`-lauseita voidaan ketjuttaa peräkkäin ja testata useita eri vaihtoehtoja samassa rakenteessa. `elif`-osioita voi `if`-rakenteessa olla mielivaltainen määrä; `else`-osio on vapaaehtoinen eli se voi puuttua tai se voi olla mukana, mutta kumpaakaan ei voi olla olemassa ilman `if`-osiota, joita voi olla ainoastaan yksi per rakenne.

`if`-lauseen osat (`if…elif…else`) ovat aina samalla tasolla ja eri haaroihin liittyvät koodirivit on aina sisennetty seuraavalle tasolle. `if`-rakenteen jälkeen koodi jatkuu alkuperäisellä tasolla, ts. sama sisennys kuin `if`-sanalla oli. Kuulostaako jossittelulta? Ei hätää, sillä Esimerkki 3.4 selventää asiaa.

Esimerkki 3.4. if -valintarakenteen variaatiot

```
# Esimerkki 3.4. if-valintarakenteen variaatiot
print("Ehdollinen koodi: if-lause")
Jaettava = float(input("Anna jaettava: "))
Jakaja = float(input("Anna jakaja: "))
if (Jakaja != 0):
    print(Jaettava / Jakaja)
print("Valintarakenteen jälkeen koodi palaa päätasolle.")
print()

############################################################
print("Ehdollinen koodi: else-haara")
Jaettava = float(input("Anna jaettava: "))
Jakaja = float(input("Anna jakaja: "))
if (Jakaja != 0):
    print(Jaettava / Jakaja)
else:
    print("Jakaja oli nolla, ei voi laskea.")
print("Valintarakenteen jälkeen koodi palaa päätasolle.")
print()

############################################################
print("Ehdollinen koodi: if-elif ilman else-haaraa")
Jatketaan = False
Jatko = input("Haluatko jatkaa ohjelman suorittamista (k/e): ")

if (Jatko == "k"):
    Jatketaan = True
elif (Jatko == "K"):
    Jatketaan = True

if (Jatketaan == True):
    print("Päätit jatkaa ohjelman suorittamista.")
print()
```

```
#########################################################
Luku1 = int(input("Anna luku 1: "))
Luku2 = int(input("Anna luku 2: "))
print("Yleinen valintarakenne, Luku1="+str(Luku1)+" ja
Luku2="+str(Luku2))
if (Luku1 == Luku2):
    print("Luvut ovat yhtä suuria.")
    print("Koodilohkossa voi olla monta riviä sisennettynä samaan
tasoon.")
elif (Luku1 > Luku2):
    print("Luku1 on suurempi kuin Luku2")
elif (Luku1 < Luku2):
    print("Luku1 on pienempi kuin Luku2")
else:
    print("Tänne else-haaraan ei pitäisi päätyä koskaan.")
print("Valintarakenteen jälkeen koodi palaa päätasolle.")
print("Kiitos ohjelman käytöstä.")
```

Tuloste

```
Ehdollinen koodi: if-lause
Anna jaettava: 4
Anna jakaja: 0
Valintarakenteen jälkeen koodi palaa päätasolle.

Ehdollinen koodi: else-haara
Anna jaettava: 4
Anna jakaja: 0
Jakaja oli nolla, ei voi laskea.
Valintarakenteen jälkeen koodi palaa päätasolle.

Ehdollinen koodi: if-elif ilman else-haaraa
Haluatko jatkaa ohjelman suorittamista (k/e): k
Päätit jatkaa ohjelman suorittamista.

Anna luku 1: 3
Anna luku 2: 3
Yleinen valintarakenne, Luku1=3 ja Luku2=3
Luvut ovat yhtä suuria.
Koodilohkossa voi olla monta riviä sisennettynä samaan tasoon.
Valintarakenteen jälkeen koodi palaa päätasolle.
Kiitos ohjelman käytöstä.
```

Ohjelman ensimmäinen osa demonstroi nollalla jakoa ja ohjelman logiikka toimii eli ohjelma ei kaadu virheeseen, mutta käyttäjä jää ihmettelemään mitä tapahtui. Toisessa osassa tämä ongelma on korjattu `else`-haaralla, jossa käyttäjälle kerrotaan ongelmasta. Hyvä tapa olisi neuvoa käyttäjää tässä vaiheessa ja kertoa, miten ongelman voi välttää, ja siksi palaamme tähän myöhemmin. Kolmannessa osassa on `if-elif` -rakenne ilman `else` osaa, jossa näkyy, miten käyttäjän antama iso tai pieni k-kirjan mahdollistaa ohjelmassa oma koodilohko suorittamisen. Tässä käytetään yksisuuntaista lippu -roolia ja se olisi voitu korvata yhdellä `if`-lauseella laittamalla ehdoksi `((Jatko == 'k') or (Jatko == 'K'))`. Neljäs osa esittelee yleistä valintarakennetta, jossa vertaillaan kahta eri lukua. Ensimmäisessä haarassa on kaksi tulostusta eli koodilohko tunnistetaan sisennyksistä ja kaikki yhdessä haarassa olevat samaan tasoon sisennetyt käskyt suoritetaan. Riippuen käyttäjän antamista luvuista ohjelma päätyy eri haaroihin ja tulostaa niissä sopivat tiedot käyttäjälle. Huomaa, että `else`-haaraan ei pitäisi päätyä koskaan, joten se on tässä laitettu mukaan varmuuden vuoksi ja mahdollistamaan odottamattomien tilanteiden hallinta. Tällä erää haarassa olevaa tekstiä ei pitäisi koskaan tulostua näytölle. Tässä tiiviissä esimerkissä on *Kiitos ohjelman käytöstä.* -kommentti, sillä muutoin ohjelman loppumista on vaikea tunnistaa luotettavasti.

Yleisiä huomioita valintarakenteesta

`if`-rakenteen käyttö vaatii tarkkuutta eli muista laittaa `if`-, `elif`- ja `else`-osien perään kaksoispiste, jotta Python-tulkki tietää uuden osion alkavan siitä. Toiseksi myös sisennykset vaativat huolellisuutta, sillä etenkin viimein koodilohkoon kuuluva rivi jää helpolla sisentämättä, jolloin ohjelma ei toimi oikein. Kolmas huomion arvoinen asia on yhtäsuuruusoperaattorin (”==”) ja sijoitusoperaattorin (”=”) ero. **Vertailua ei koskaan toteuteta vain yhdellä yhtäsuuruusmerkillä!** Tulkki kyllä huomauttaa, jos tällaista yritetään.

Valintarakenteissa tulee käyttää tietotyyppien luonnollista tietotyyppiä eikä esim. merkkijonoa sen vuoksi, että tieto oli alun perin siinä muodossa. Jos siis käsitellään merkkijonona, tehdään vertailut merkkijonoille, mutta jos käsitellään lukuja, muutetaan muuttujat ensin luvuiksi ja valintarakenteessa käsitellään sen jälkeen lukuja.

Sisäkkäiset valintarakenteet

Valinrakenteessa voi olla monta haaraa ja ehtolauseessa voi olla monta ehtoa. Näiden lisäksi valintarakenteet voivat olla sisäkkäin. Lähtökohta on yksinkertainen – aiemmin kaikki koodi oli päätasolla, ja nyt valintarakenteen haarojen koodilohkot ovat sisennettyjä. Käytännössä sisennys on uusi koodin perustaso ja siinä voi olla kaikki samat käskyt kuin päätasolla. Näin ollen koodilohkossa voi olla valintarakenne tilanteeseen sopivassa muodossa omilla sisennyksillään. Ja tämän sisällä voi olla omia valintarakenteita taas sisennettynä oikein. Todellisuudessa sisäkkäisten rakenteiden kanssa tulee olla tarkkana, jotta ohjelman logiikka toimii eikä siihen tule virheitä. Siksi sisäkkäisiä rakenteita ei kannata käyttää turhaan, mutta niitä tarvitaan ajoittain ja ne ovat luonnollinen osa ohjelmia. Katsotaan Esimerkki 3.5, jossa on valintarakenteen sisällä toisia valintarakenteista ja muista, että rakenteet voivat olla sisäkkäin, kunhan ne on sisennetty oikein.

Esimerkki 3.5. Sisäkkäiset valintarakenteet ja muuttujan rooli *yksisuuntainen lippu*

```
Kayttajatunnus = input("Anna käyttäjätunnus: ")
Salasana = input("Anna salasana: ")
Tunnistettu = False

if (Kayttajatunnus == "Matti"):
    if (Salasana == "abc123#!?EFG"):
        Tunnistettu = True
elif (Kayttajatunnus == "Erja"):
    if (Salasana == "omaSalasana"):
        Tunnistettu = True
else:
    print("Tuntematon käyttäjätunnus.")

if (Tunnistettu == True):
    print("Käyttäjätunnus ja salasana olivat oikein.")
else:
    print("Käyttäjätunnus ja salasana eivät täsmää.")
```

Tuloste

```
Anna käyttäjätunnus: Erja
Anna salasana: omaSalasana
Käyttäjätunnus ja salasana olivat oikein.
```

Esimerkissä 3.5 kysytään käyttäjältä käyttäjätunnus ja salasana, jonka jälkeen tarkistetaan, että ne ovat oikein. Jos tiedot ovat oikein, on käyttäjä tunnistettu ja ohjelman lopussa tämä kerrotaan käyttäjälle. Vastaavasti jos tiedot eivät täsmää, kerrotaan siitä käyttäjälle. Esimerkin sisäkkäiset valintarakenteet voisi korvata AND-operaattorilla eli molempien ehtojen pitää toteutua. Tekemällä erilliset valintarakenteet ohjelmoija tietää, mikä ehto toteutui ja mikä ei, joten käyttäjää helpottavan palautteen antaminen on helpompaa tällä ratkaisulla. Huomaa myös esimerkissä käytetty yksisuuntainen lippu -muuttuja, joka alustetaan EPÄTOSI-arvolla ja jos käyttäjä tunnistetaan, muutetaan sen arvoksi TOSI. Yksisuuntainen tarkoittaa sitä, ettei TOSI-arvoa saanutta muuttujaa koskaan muuteta enää EPÄTOSI-arvoksi. Näin muuttujan käyttö on selkeää ja sen oikea käyttö helppo varmistaa eli muuttuja alustetaan EPÄTOSI-arvolla ja tarvittaessa se muutetaan TOSI-arvoksi.

Yksi vai monta valintarakennetta

Valintarakenteessa voi olla monta haaraa ja valintarakenteita voi olla monta. Nämä toimivat kuitenkin loogisesti eri tavoin eli yksi monihaarainen valintarakenne tuottaa erilaisen tuloksen kuin monta erillistä valintarakennetta – tai voi tuottaa syötteistä riippuen. Käytännössä ohjelman suunnitteluvaiheessa tulee miettiä, tarvitaanko yksi vain monta valintarakennetta ja minkälaisia haaroja niissä on. Lähtökohta on aina toteuttaa ohjelma siten, että se toimii käyttäjän haluamalla tavalla eikä sinne päin.

Esimerkissä 3.6 on kolme eri tavoin toteutettua osiota, jotka kaikki toimivat mutta hieman eri tavoilla.

Esimerkki 3.6. if-rakenne: yksi vai monta ja oikea vai väärä logiikka

```
Luku = int(input("Anna kokonaisluku: "))
print("Yksi if-rakenne usealla haaralla ja oikealla logiikalla:")
if (Luku < 10):
    print("Luku on pienempi kuin 10.")
elif (Luku < 100):
    print("Luku on pienempi kuin 100.")
elif (Luku < 1000):
    print("Luku on pienempi kuin 1000.")
else:
    print("Luku on suurempi tai yhtä suuri kuin 1000.")
print()

print("Monta peräkkäistä if-rakennetta:")
if (Luku < 10):
    print("Luku on pienempi kuin 10.")
if (Luku < 100):
    print("Luku on pienempi kuin 100.")
if (Luku < 1000):
    print("Luku on pienempi kuin 1000.")
if (Luku >= 1000):
    print("Luku on suurempi tai yhtä suuri kuin 1000.")
print()

print("Yksi if-rakenne usealla haaralla ja väärällä logiikalla:")
if (Luku < 1000):
    print("Luku on pienempi kuin 1000.")
elif (Luku < 100):
    print("Luku on pienempi kuin 100.")
elif (Luku < 10):
    print("Luku on pienempi kuin 10.")
else:
    print("Luku on suurempi tai yhtä suuri kuin 1000.")
print()
```

Tuloste

```
Anna kokonaisluku: 50
Yksi if-rakenne usealla haaralla ja oikealla logiikalla:
Luku on pienempi kuin 100.

Monta peräkkäistä if-rakennetta:
Luku on pienempi kuin 100.
Luku on pienempi kuin 1000.

Yksi if-rakenne usealla haaralla ja väärällä logiikalla:
Luku on pienempi kuin 1000.
```

Kuten tulosteista näkyy, ohjelmat toimivat hieman eri tavoin käyttäjän syötteellä 50. Kaikki tulosteet ovat totta, mutta tarkin tulos on ensimmäinen eli pienempi kuin 100. Pienempi kuin 100 sekä 1000 on totta kuten myös pienempi kuin 1000, mutta tarkin vastaus on ensimmäinen ”pienempi kuin 100”.

Erot johtuvat suoritusjärjestyksestä. Useista valintarakenteista muodostuvassa osiossa koodi suorittaa jokaisen `if`-lauseen huolimatta siitä, onko aiempi `if`-lause ollut tosi. Tämä on `if-elif-else`-rakenteen ja `if-if-else`-rakenteiden ero: `elif`-väitteet tutkitaan ainoastaan, mikäli aiemmat `if`- tai `elif`-väitteet ovat saaneet arvon `False`. Lisäksi rakenteesta poistutaan heti, kun ensimmäinen `elif`-lause saa arvon `True`. Sen sijaan erillisten `if`-lauseiden kohdalla jokainen `if`-väittämä testataan erikseen siitä huolimatta, oliko aiempi samaan rakenteeseen kuuluva `if`-lause ollut totta.

Huomaa, että tulkin kannalta koodissa ei ole mitään ongelmaa – mikäli koodia käytettäisiin vaikkapa 10-potenssin tunnistamiseen, olisi vastaus hyödytön vaikkakin teknisesti täysin oikein. Ohjelma voi siis sisältää loogisia virheitä, vaikka se olisi syntaksin osalta kirjoitettu oikein. Tyypillisesti monta vaihtoehtoa käsitellään monihaaraisella valintarakenteella.

Valintarakenteen tyypillisiä sovelluskohteita

Henkilötunnus ja luvun parillisuus

Tässä vaiheessa opasta henkilötunnus on tuttu asia, sillä se sopii erittäin hyvin useiden asioiden esittelyyn. Henkilötunnus on siis merkkijono, jossa on koodattuna paljon tietoa, mm. päivämäärä, vuosisata, järjestysnumero ja sukupuoli. Valintarakenteen osalta erityisen kiinnostavia asioita ovat vuosiluku ja sukupuoli. Vuosiluku muodostuu kahdesta osasta, vuoden ja vuosikymmenen sisältävästä osasta sekä vuosisadan määrittävästä merkistä. Sukupuoli taas on koodattu yksilöivään järjestysnumeroon eli naisilla luku on parillinen ja miehillä pariton. Esimerkki 3.7 sisältää näitä asioita käsittelevän koodin.

Esimerkki 3.7. Henkilötunnuksen syntymävuosi ja luvun parillisuus

```
Henkilotunnus = input("Anna henkilötunnuksesi: ")
VuosiLyhyt = int(Henkilotunnus[4:6])
Vuosisata = Henkilotunnus[6:7]
Jarjestysnumero = int(Henkilotunnus[7:10])

# Syntymävuoden selvittäminen
if (Vuosisata == "+"):
    Vuosi = 1800 + VuosiLyhyt
elif (Vuosisata == "-"):
    Vuosi = 1900 + VuosiLyhyt
elif (Vuosisata == "A"):
    Vuosi = 2000 + VuosiLyhyt
else:
    print("Vuosisadan tunnus tuntematon, tarkista se.")

# Sukupuolen selvittäminen, Jarjestysnumero on naisilla parillinen
if ((Jarjestysnumero % 2) == 0):
    Sukupuoli = "Olet nainen."
else:
    Sukupuoli = "Olet mies."

print("Olet syntynyt vuonna", Vuosi)
print(Sukupuoli)
```

Tuloste

```
Anna henkilötunnuksesi: 120302A213K
Olet syntynyt vuonna 2002
Olet mies.
```

Ohjelmassa otetaan ensin leikkaukset kiinnostavista tiedoista, joita käsitellään tarkemmin. Vuosi ja vuosikymmen ovat yhdessä, kun taas vuosisata on koodattu merkillä ’+’, ’-’ ja ’A’ eli näitä merkkejä vastaavat vuosisadat ovat 1800, 1900 ja 2000 koodin mukaisesti. Syntymävuosi saadaan yhdistämällä nämä luvut. Vastaavasti järjestysnumerosta saadaan selvitettyä parillisuus jakamalla se kahdella ja katsomalla jakojäännöstä, jolloin 0 tarkoittaa parillista ja 1 paritonta lukua. Näin kahdella valintarakenteella saatiin selvitettyä pyydetyt tiedot.

Valikkorakenne

Tässä oppaassa keskitytään merkkipohjaisiin tietokoneohjelmiin. Oppaan lopussa katsotaan myös graafisten käyttöliittymien tekemistä, mutta tässä vaiheessa ohjelmoinnin opiskelua merkkipohjaiset ohjelmat ovat sopivampia. Lähtökohtana on, että käyttäjän tulee pystyä ohjaamaan ohjelmaa eli kertomaan ohjelmalle mitä sen tulee tehdä. Tämä tavoite voidaan saavuttaa monella tavalla, mutta yksi hyvä keino tähän on valikon käyttö ja siksi tutustumme seuraavaksi valikkopohjaisen ohjelman ideaan. Meillä on tässä vaiheessa vielä rajallinen valikoima ohjelmointirakenteita käytössä, mutta valikon ja ohjelmaa toimintaa ohjaavan valintarakenteen toteutus onnistuu.

Esimerkissä 3.8 näkyy valikkorakenteen idea, joka perustuu kahteen osaan. Ensimmäinen osa on valikon tulostus käyttäjälle. Käytännössä valikko muodostuu joukosta käskyjä, joissa on vakiorakenne: ohje, valinnat ja valinnan kysyminen käyttäjältä. Tämän jälkeen käyttäjän valinta käsitellään valintarakenteessa, jossa on vastaavat haarat kuin käyttäjälle näytetyissä valinnoissa. Tällä tavoin käyttäjä voi valita, mitä hän haluaa tehdä ja sen jälkeen ohjelmassa on selkeät vastaavat kohdat, joissa kaikki valinnat käydään läpi.

Esimerkki 3.8. Valikkorakenteen idea

```
Luku1 = int(input("Anna ensimmäinen luku: "))
Luku2 = int(input("Anna toinen luku: "))
print()

print("Valitse haluamasi toiminto:")
print("1) Laske lukujen summa")
print("2) Laske lukujen erotus")
print("0) Lopeta")
Syote = input("Anna valintasi: ")
Valinta = int(Syote)
print()

if (Valinta == 1):
    print("Summa on", Luku1+Luku2)
elif (Valinta == 2):
    print("Erotus on", Luku1-Luku2)
elif (Valinta == 0):
    print("Lopetetaan.")
else:
    print("Tuntematon valinta.")

print("Kiitos ohjelman käytöstä.")
```

Tuloste

```
Anna ensimmäinen luku: 3
Anna toinen luku: 1

Valitse haluamasi toiminto:
1) Laske lukujen summa
2) Laske lukujen erotus
0) Lopeta
Anna valintasi: 1

Summa on 4
Kiitos ohjelman käytöstä.
```

Esimerkin 3.8 ohjelmassa käyttäjä voi valita lukujen summan laskimisen antamalla valinnaksi numeron 1, laskea erotuksen numerolla 2 ja lopettaa ohjelma valinnalla 0. Summan ja erotuksen lasku tapahtuu valintarakenteessa omissa haaroissa ja ohjelman lopetukselle on oma valinta näiden jälkeen. Tämä rakenne mahdollistaa valikkorakenteen laajentamisen eli uusien toimintojen lisäämisen valikkoon kohtina 3, 4, jne., ja silti ohjelman lopetus tapahtuu aina samalla vakiosyötteellä 0. Huomaa, että ohjelman lopetus eli valinta 0 on valintarakenteessa viimeisenä ehdollisena haarana ja valintarakenteessa on myös else-haara, jolla voidaan käsitellä muut käyttäjän antamat syötteet ja varoittaa tätä virheestä.

Tämä valikkorakenne on lähtökohta useimmille isommille ohjelmille tässä oppaassa eli tämä rakenne kannattaa opetella tekemään. Rakenne muuttuu ja kehittyy, kun pääsemme oppaassa eteenpäin, mutta keskeisimmät osat löytyvät jo tästä esimerkistä. Valikkorakenteessa tulee aina olla em. `else`-haara tuntemattomien valintojen käsittelyyn, mutta siinä ei tarvitse olla virheentarkistusta muiden virheiden varalta. Palaamme tarkemmin virheenkäsittelyyn myöhemmin oppaassa.

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.

- Ehdollinen koodi eli if-rakenne (E3.1, E3.4)
- Koodilohko eli sisennys
- Ehtolauseke, totuusarvo, loogiset operaattorit ja suoritusjärjestys (Taulukko 3.1, 3.2, 3.3; E3.2, E3.3)
- Sisäkkäiset valintarakenteet (E3.5, E3.9)
- Muuttujien roolit: yksisuuntainen lippu (E3.5)
- Yksi vai monta valintarakennetta, valintarakenteen logiikka (E3.6)
- Luvun parillisuus (E3.7, E3.9)
- Valikkopohjainen ohjelma (E3.8, E3.9)

Palaamme useampiin asioihin uudestaan myöhemmin kurssilla. Esimerkiksi valintarakenteen hyväksyttävä koko muuttuu jatkossa, kun tutustumme muihin ohjelmointi- ja tietorakenteisiin. Samoin virheenkäsittelyyn palataan myöhemmin tarkemmin ja tässä vaiheessa tulee suorittaa tehtävänannossa erikseen mainitut virheentarkistukset.

Pienen Python-perusohjelman tyyliohjeet

Tähän lukuun liittyvät tyyliohjeet on koottu alle.

1. Ohjelmarivit sisennetään loogisesti samalla tyylillä koko ohjelmassa. Käytä aina neljää (4) välilyöntiä, joka onnistuu useimmissa koodieditoreissa sarkainnäppäimellä, esim. IDLE ja Visual Studio Code.
2. Ehtolausekkeet tulee laittaa selkeästi sulkuihin, esim.
   ```
   if ((Ehto1 == True) or (Ehto2 == True)):
   ```
3. Valikkopohjaisen ohjelman valintarakenteena käytetään monihaaraista `if-elif-else`-rakennetta:
   a. Valintarakenteen valinnat käydään järjestyksessä alkaen valinnasta 1 valintaan N, jonka jälkeen on valinta 0 eli ohjelman normaali lopetus.
   b. Valintarakenteen viimeinen haara on `else`-haara, jossa käsitellään tuntemattomat valinnat.
   c. Valinta käsitellään kokonaislukuna.
   d. Valikon jokainen rivi tulostetaan omalla `print`-käskyllä.
4. Valikkopohjaisen ohjelman standardifraasit ovat seuraavat:
   a. `"Valitse haluamasi toiminto:"`
   b. `"0) Lopeta"`
   c. `"Anna valintasi: "`
   d. `"Tuntematon valinta."`
   e. `"Lopetetaan."`
   f. `"Kiitos ohjelman käytöstä."`
5. Valikkopohjaisen ohjelman valintahaarassa tulee tehdä kaikki valintaan liittyvät toimenpiteet alusta loppuun asti.
6. Mikäli ohjelman suorituksessa tulee ongelmia, tulee ohjelman kertoa käyttäjälle selkeästi, mikä ongelma on ja miten sen voi ratkaista. Virheenkäsittelyyn palataan laajemmin myöhemmin omassa luvussa.

7. Valintarakenteessa tulee käyttää tietotyyppien luonnollista tietotyyppiä. Jos käsitellään merkkijonona, tehdään vertailut merkkijonoille, ja jos käsitellään lukuja, muutetaan muuttujat ensin luvuiksi ja valintarakenteessa käsitellään sen jälkeen lukuja.
8. Valintarakenteen järkevä pituus määräytyy ehtojen määrän sekä käytettävissä olevien ohjelmointi- ja tietorakenteiden pohjalta.
9. Tässä oppaassa ei käytetä match-case -rakennetta.

Luvun asiat kokoava esimerkki

Esimerkissä 3.9 on valikkopohjainen ohjelma useilla valintarakenteille. Kaikki ohjelmointiin liittyvät asiat on käsitelty aiemmissa esimerkeissä, mutta tässä esimerkissä näkyy ohjelman pituuden kasvu, koska siinä on useita käsiteltävään tietoon perustuvia valintoja. Esimerkiksi kuukausien käsittely edellyttää lähtökohtaisesti omaa haaraa jokaista kuukautta kohti. Käytännössä valintarakenteen järkevä pituus määräytyy ehtojen määrän sekä käytettävissä olevien ohjelmointi- ja tietorakenteiden pohjalta. Esimerkiksi toistorakenteella, lista- ja sanakirjarakenteilla voidaan pienentää valintarakenteen kokoa ja näihin palataan myöhemmin.

Esimerkki 3.9. Valikkopohjaisen ohjelman idea

```
# 20230918 un E03_9.py (c) LUT
##################################
# Kiintoarvojen asettaminen
PAIVA = 0
KUUKAUSI = 2
VUOSI = 4
VUOSISATA = 6
NUMERO = 7

##################################
# Muuttujien alustukset
Tuloste = ""

##################################
# Tiedon lukeminen
Henkilotunnus = input("Anna henkilötunnuksesi: ")
print("Valitse haluamasi toiminto:")
print("1) Tulosta syntymäajan päivä")
print("2) Tulosta syntymäajan kuukausi")
print("3) Tulosta syntymäajan vuosi")
print("4) Tulosta sukupuoli")
print("0) Lopeta")
Syote = input("Anna valintasi: ")
Valinta = int(Syote)
print()

##################################
# Tiedon käsitteleminen
if (Valinta == 1):
    Paiva = int(Henkilotunnus[PAIVA:PAIVA+2])
    Tuloste = "Olet syntynyt "+str(Paiva)+". päivä."
elif (Valinta == 2):
    Kuukausi = int(Henkilotunnus[KUUKAUSI:KUUKAUSI+2])
    if (Kuukausi == 1):
        KuukausiNimi = "tammikuussa"
    elif (Kuukausi == 2):
        KuukausiNimi = "helmikuussa"
    elif (Kuukausi == 3):
        KuukausiNimi = "maaliskuussa"
    elif (Kuukausi == 4):
```

```
        KuukausiNimi = "huhtikuussa"
    elif (Kuukausi == 5):
        KuukausiNimi = "toukokuussa"
    elif (Kuukausi == 6):
        KuukausiNimi = "kesäkuussa"
    elif (Kuukausi == 7):
        KuukausiNimi = "heinäkuussa"
    elif (Kuukausi == 8):
        KuukausiNimi = "elokuussa"
    elif (Kuukausi == 9):
        KuukausiNimi = "syyskuussa"
    elif (Kuukausi == 10):
        KuukausiNimi = "lokakuussa"
    elif (Kuukausi == 11):
        KuukausiNimi = "marrasukuussa"
    elif (Kuukausi == 12):
        KuukausiNimi = "joulukuussa"
    Tuloste = "Olet syntynyt "+KuukausiNimi+"."
elif (Valinta == 3):
    VuosiLyhyt = Henkilotunnus[VUOSI:VUOSI+2]
    Vuosisata = Henkilotunnus[VUOSISATA:VUOSISATA+1]
    if (Vuosisata == "+"):
        Vuosi = int("18" + VuosiLyhyt)
    elif (Vuosisata == "-"):
        Vuosi = int("19" + VuosiLyhyt)
    elif (Vuosisata == "A"):
        Vuosi = int("20" + VuosiLyhyt)
    Tuloste = "Olet syntynyt vuonna "+str(Vuosi)+"."
elif (Valinta == 4):
    Jarjestysnumero = int(Henkilotunnus[NUMERO:NUMERO+3])
    if (Jarjestysnumero % 2 == 0):
        Tuloste = "Olet nainen."
    else:
        Tuloste = "Olet mies."
elif (Valinta == 0):
    print("Lopetetaan.")
else:
    print("Tuntematon valinta.")

###################################
# Tiedon tulostus
print(Tuloste)
print("Kiitos ohjelman käytöstä.")

###################################
# eof
```

Tuloste

```
Anna henkilötunnuksesi: 120302A213K
Valitse haluamasi toiminto:
1) Tulosta syntymäajan päivä
2) Tulosta syntymäajan kuukausi
3) Tulosta syntymäajan vuosi
4) Tulosta sukupuoli
0) Lopeta
Anna valintasi: 2

Olet syntynyt maaliskuussa.
Kiitos ohjelman käytöstä.
```