Luku 3. Ensimmäinen Python-ohjelma

Jos olet ottanut Python-työkalut käyttöön kuten luvussa 2 kuvailtiin, olet varmaan myös jo kokeillut Python-tulkin toimivuutta muutamilla syötteillä. Se kannattaa tehdä viimeistään nyt ennen kuin mennään eteenpäin.

Python-tulkkia ei voi rikkoa omalla ohjelmallaan, mutta sen voi saada hämmentymään ja tulostamaan ilmoituksia virheistä. Esimerkiksi jos tarkoitus oli laskea yhteen 2 + 2, mutta syöte olikin **2 2 +**, niin Python-tulkki kertoo mielipiteensä, koska tällainen syöte ei ole Python-kielen mukaista:

```
>>> 2 2 +
  File "<stdin>", line 1
    2 2 +
      ^
SyntaxError: invalid syntax
```

Ilmoitus `SyntaxError: invalid syntax` tarkoittaa, että syötteen muoto eli *syntaksi* on virheellinen. Ilmoituksessa on myös nuolenkärki, joka osoittaa virheelliseen kohtaan. Useimmat Python-tulkin antamat virheilmoitukset ovat samantapaisia.

Arvaatko luvun?

Python-ohjelmointia voi opiskella joko pala palalta, vähitellen yhdistellen asioita toimivaksi kokonaisuudeksi, tai sitten lähteä liikkeelle hieman isommasta ja pilkkoa sitä vähitellen helpommin ymmärrettäviin palasiin.

Tässä luvussa lähdetään liikkeelle pienestä, mutta kokonaisesta Python-ohjelmasta, joka sisältää monia ohjelmoinnin perusteisiin liittyviä asioita. Samalla tulee myös esille joitakin Pythonin erityispiirteitä.

Ensimmäinen Python-ohjelma laittaa sinut arvaamaan luvun, jota tietokone "ajattelee". Tähän voi tosiaan laittaa lainausmerkit ympärille, koska tietokone ei toki varsinaisesti ajattele yhtään mitään, vaan ainoastaan suorittaa sille annettuja käskyjä erittäin nopeasti. Ohjelman voi kuitenkin laatia siten, että syntyy uskottava vaikutelma ajattelevasta, älykkäästä koneesta.

Ohjelma sisältää 12 riviä, jotka voi antaa Python-tulkin suoritettavaksi. Näiden rivien syöttäminen suoraan tulkille voi olla hieman hankalaa, koska Pythonissa rivien sisennys on erittäin tärkeää. Niinpä onkin parempi kirjoittaa kaikki rivit ensin tekstieditoriohjelmassa ja antaa ne sitten Python-tulkin suoritettavaksi kaikki yhdellä kertaa.

```
luku = 42
meni_oikein = False
while not meni_oikein:
    arvaus_teksti = input('Anna arvauksesi: ')
    arvaus = int(arvaus_teksti)
    if arvaus < luku:
        print('Se on isompi!')
    elif arvaus > luku:
        print('Se on pienempi!')
    else:
        meni_oikein = True
        print(f'Arvasit oikein, se oli {luku}!')
```

Käynnistä IDLE-ohjelma ja tee uusi tiedosto. Kirjoita yllä olevat rivit tiedostoon. Jos sinulla on käytössä tämän kirjan PDF-versio, voit myös kopioida rivit siitä leikepöydän kautta IDLE:n editoriin. Tärkeintä on huolehtia siitä, että ensimmäiset kolme riviä alkavat ensimmäisestä sarakkeesta, ja rivit neljännestä alkaen alkavat yhden sarkainvälin verran oikealle päin. Joukossa on myös sellaisia rivejä, jotka alkavat vielä kauempaa oikealta, ja tällä kaikella on merkitys.

Sisennys määrittää rakenteen

Se Pythonin piirre, joka selkeimmin erottaa sen muista ohjelmointikielistä, on pakotettu rakenne. Jotta ohjelman suoritus etenisi oikein, ja jotta ohjelman rakenne olisi helppo ymmärtää, Python pakottaa ohjelmoijan syöttämään ohjelmansa tietyssä muodossa. Tämä muoto perustuu lähes yksinomaan sisentämiseen.

Sisennys (engl. *indentation* tai *indent*) tarkoittaa ohjelmarivin aloituskohdan siirtämistä oikealle päin, niin että rivin alkuun lisätään tyhjää (joko välilyöntejä tai sarkainmerkkejä).

Python-ohjelmateksti alkaa aina rivin vasemmasta reunasta, ja sitä mukaa kun siinä esiintyy sisäkkäisiä rakenteita, niitä pitää sisentää vastaavassa suhteessa. Sisennyksenä käytetään useimmiten neljän välilyönniin ryhmää, mutta itse asiassa välilyöntejä voi olla myös enemmän tai vähemmän, kunhan niitä on yhdenmukainen määrä. Käytännössä Python-ohjelmissa esiintyy vain kahden tai neljän välilyönnin sisennyksiä. Sisennyksen pitää olla sama kautta ohjelman, eli eri kokoisia sisennyksiä ei voi käyttää ristiin ohjelman eri riveillä.

Tämä Pythonin piirre jakaa mielipiteitä, jopa niin vahvasti, että joidenkin mielestä se on parasta ikinä, ja toisten mielestä taas maailman typerin keksintö. Totuus lienee jälleen kerran jossain sillä välillä: toisaalta sisennyksen käyttö todellakin tekee Python-ohjelmista helpompia lukea ja ymmärtää, ja toisaalta taas se voi aiheuttaa erityisesti vasta-alkajille hankalasti selvitettäviä ongelmia ellei asiasta tiedä. Nyt tiedät, että Python-ohjelmoinnissa *välillä on väliä*.

Mitä tapahtuu jos sisennys menee väärin? Python-tulkki kieltäytyy suorittamasta ohjelmaa ja näyttää virheilmoituksen. Seuraavassa esimerkissä `print`-aliohjelman kutsun edellä on kaksi välilyöntiä:

```
>>>   print('Hello, world!')
File "<stdin>", line 1
  print('Hello, world!')
IndentationError: unexpected indent
```

Sana *indent* tai *indentation* tarkoittaa sisennystä.

Kirjainkoolla on väliä

Pythonissa erotellaan isot ja pienet kirjaimet toisistaan tarkasti. Kaikki kielen avainsanat, kuten `if`, `while` ja monet muut, kirjoitetaan kokonaan pienellä. Vastaavasti taas esimerkiksi aliohjelmilla on aina tasan tarkkaan se kirjoitusasu, jonka alkuperäinen tekijä on niille antanut. Esimerkiksi `print` on eri asia kuin `Print` tai `PRINT`.

Python-tulkki antaa usein ihan hyödyllisiä virheilmoituksia sellaisista virheistä, jotka se tunnistaa. Jos esimerkiksi olet vahingossa kirjoittanut `Print`, niin virheilmoitus näyttää seuraavanlaiselta:

```
>>> Print('Moro!')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'Print' is not defined. Did you mean: 'print'?
```

Jos Python-ohjelmasi ei toimi ja saat outoja virheilmoituksia, niin kannattaa tarkistaa, että olet kirjoittanut kaikki muuttujien nimet samalla tavalla aina silloin kun tarkoitat samaa asiaa: esimerkiksi `arvaus` ja `Arvaus` ovat Pythonille kaksi eri asiaa.

Ohjelman ajaminen

Avaa IDLE:ssä uusi tiedosto valitsemalla File → New File. Syötä ohjelmasi tähän ikkunaan ja tallenna se levytiedostoon valitsemalla File → Save. Sen jälkeen voit ajaa sen Python-tulkissa. Valitse editori-ikkunassa komento Run → Run Module.

Ohjelma käynnistyy IDLE Shell -ikkunassa, tulostaa tekstin `Anna arvauksesi:` ja jää sitten odottamaan että kirjoitat arvaamasi luvun näppäimistöltä ja painat sitten **Enter**-näppäintä.

Tämä ohjelma ei tarkista mitenkään sitä onko syöttämäsi tieto jokin luku, kuten **`42`**, vai mahdollisesti esimerkiksi tekstiä, kuten **`neljätoista`**. Jos syötät ohjelmalle jotain mikä ei sisällä numeroita, ohjelman suoritus pysähtyy virheeseen. Jatkossa tulemme tarkistamaan paremmin käyttäjän syötteet, mutta juuri nyt emme jää ratkomaan sitä ongelmaa.

Seuraavassa on esimerkkinä yksi ohjelman ajokerta, jonka aikana olen syöttänyt ohjelmalle useita lukuja, ja ohjelma on tulostanut reaktionsa niihin.

```
Anna arvauksesi: 32
Se on isompi!
Anna arvauksesi: 66
Se on pienempi!
```

```
Anna arvauksesi: 54
Se on pienempi!
Anna arvauksesi: 48
Se on pienempi!
Anna arvauksesi: 40
Se on isompi!
Anna arvauksesi: 42
Arvasit oikein, se oli 42!
```

Itse asiassa tiesin, että luku jota tietokone "ajattelee" on 42, koska olin sen itse ohjelmoinut, mutta voit salaa vaihtaa tämän luvun tilalle jonkin muun luvun, ja pyytää vaikka opiskelutoveria ajamaan ohjelman niin että hän ei näe mitä lukua käytit. Myöhemmin opettelemme tekemään Pythonilla satunnaisesti valitun luvun tietyltä lukuväliltä, jolloin voit itsekin ihan oikeasti pelata tätä arvauspeliä.

Mitähän mahtaa tapahtua jos syöttää luvun tilalle tekstiä? Siitä on helppo ottaa selvää:

```
Anna arvauksesi: kolme
Traceback (most recent call last):
  File "<stdin>", line 3, in <module>
ValueError: invalid literal for int() with base 10: 'kolme'
```

Python-tulkin virheilmoituksen nimi on tällä kertaa `ValueError`, ja sen selitteenä on `invalid literal for int() with base 10: 'kolme'`. Selitys tälle on, että Pythonin `int`-aliohjelma odotti saavansa sellaisen parametrin, joka olisi muodollisesti pätevä kymmenjärjestelmän luku, mutta saikin merkkijonon, jonka sisältönä oli teksti **`kolme`**.

Python-tulkki siis valvoo ohjelman suoritusta, ja mikäli eteen tulee tilanne, josta ei pääse eteenpäin, tulkki keskeyttää ohjelman suorituksen. Tämä on yleensä parempi vaihtoehto kuin jatkaminen eteenpäin virheistä huolimatta, koska näin vältetään isommat vahingot.

Tässä ensimmäisessä ohjelmassa oli paljon sellaisia asioita, joita ei vielä tässä vaiheessa voi eikä tarvitsekaan osata. Sen tarkoitus on ainoastaan antaa käsitys siitä, millaisia ohjelmia Pythonilla voi tehdä. Myöhemmin tässä kirjassa teemme tällaisia ja vielä paljon monipuolisempiakin ohjelmia.