Python – Perusteet

Materiaalia

Tero Tuovinen
Tietotekniikan laitos
tttuovin@cc.jyu.fi

6. lokakuuta 2004

Sisältö

1 Yleistä **3**

2 Kommentointi **3**

3 Tyypitys **4**

4 Luokka **5**

5 Oliot **6**

6 Lista **6**

7 Tuple **7**

8 Palautusarvo **8**

9 Tiedostoihin jakaminen **9**

10 Tiedostosta lukeminen ja siihen kirjoittaminen **9**

11 Poikkeukset **10**

12 Yield **10**

13 Dict **11**

14 Regexp **12**

15 Lähteitä **13**

1 Yleistä

Python on tulkattava ja interaktiivinen ohjelmointikieli. Tulkattavuudesta aiheutuu se, ettei Python ole niin nopeaa kuin esim. optimoitu C++. Interaktiivisuus pythonissa tarkoittaa sitä, että Pythonia voi kirjoittaa suoraan komentoriville, ilman, että lähdekoodia tarvitsee mitenkään tallentaa erilliseen tiedostoon ja ajaa. Interaktiivisuus helpottaa suunnattomasti koodin kirjoittamista (vaikka ohjelmat kirjoitettaankin käytännössä tiedostoihin). Tällä hetkellä uusin versio pythonista on 2.4.

Python käyttää sisentämistä koodilohkojen määrittämiseen (sama c++ = {}). Tämä tarkoittaa joskus joissakin käyttöjärjestelmissä tabulaattorin mittaista sisennystä, joissakin käyttöjärjestelmissä se voidaan korvata n:llä kappaleella välilyöntejä.

2 Kommentointi

Pythonissa kommentointimerkki on #. Tulkki ei enää lue tämän jälkeen mitään mitä on kirjoitettu rivin loppuun. Tätä ei tule kuitenkaan käyttää oikeassa kommentoinnissa, vaan kommentointi tulee kirjoittaa osaksi koodia seuraavasti:

```
""" G  is a list of numbers 1-10  """
G = [1,2,3,4,5,6,7,8,9,10]
```

Kommentoitaessa koodin sisälle (docstring) hyötynä on se, että komentotulkissa käskyllä `help(function_name)` saadaan näytölle tulostumaan funktion kommentointi. Lisäksi on ohjelmia, jotka keräävät kommentoinnin järkevästi kasaan, ja antavat näin hyvän pohjan muulle koodin dokumentoinnille.

Kommentoinnissa tulee huomioida se, ettei Pythonissa funktioiden parametreille tarvitse määrittää mitään tyyppiä. Kuitenkin monesti odotetaan tietyntyyppisiä parametrejä, joten halututtujen parametrien tyypit on hyvä kirjoittaa muistiin kommentteihin.

Tehtävä 0. Hae komentotulkissa str - funktion kommentointi.

3 Tyypitys

Pythonissa on automaattinen tyypitys. Tämä merkitsee sitä, että ei tarvitse määritellä mitä tyyppiä muuttuja on. Lista

```
L  = ["cat", 1 , 0.1]
```

on siis täysin sallittu.

Mikäli halutaan lisätä merkkijonoon `1`merkkijono `2`–> `12`, tämä tulee sanoa `inttostr(1)+"2"`, muuten vastaus on 3. Mahdollista on myös määrittellä tietyt muuttujat kiinteästi merkkijonoiksi tai numeroiksi.

```
t = 2            #  If you want use numbers:
t = str(t)       #  t = int(t)
print t + 'g'    #  print t + 4
```

Täysin sallittua on myös sanoa `a=3*"kissa"`. Tällöin `a:n`arvo on `kissakissakissa`. Kuten yllä `a:han`on myös mahdollista lisätä toinen merkkijono.

```
a=a+"koira"
```

Nyt `a:n`arvo on `kissakissakissakoira`. Lisäksi `a:ta`voi käsitellä kuten listaa kirjaimia. Ensimmäisen kirjaimen indeksi on 0.

```
b=a[3:]
```

Tarkoittaa, että `b:hen`tallennetaan `a:n`kirjaimet neljännestä loppuun asti.

```
a[:-3]
```

Tarkoittaa a:n kaikkia paitsi kolmea viimeistä kirjainta.

Tehtävä 1: Tee lista, joka sisältää listan koiria, kissoja ja numeroitai.

Tehtävä 2: Käsittele merkkijonoa listana, ja poista siitä 4 viimeistä ja 2 ensimmäistä kirjainta.

4 Luokka

Luokka määritellään Pythonissa seuraavasti.

```
class Classname(baseclass):
        """comments"""
        def __init__(self, values, default=0):
                self.values=values
                self.default=default
                """if there is no value to default,
                this will be zero"""
        def function(self)
                """First function"""
                return
```

'Classname'on luokan nimi, 'baseclass'kertoo, mistä yläluokasta luokka on peritty. Python tukee moniperintää, joten luokka voidaan periä useammasta yläluokasta, ja saada niiden kaikkien ominaisuudet. Luokkaa ei ole pakko periä toisesta luokasta, usein on jopa suositeltavaa käyttää Pythonin muita ominaisuuksia perinnän tilalla."""comments"""- rivin jälkeen tulee alustus, joka suoritetaan, kun olio luodaan. Ensimmäisenä parametrinä on osoitin olioon itseensä ''self'', jonka jälkeen tulevat muut parametrit. Oletusarvo määritellään = -merkin avulla. **Huomaa, ettei parametriä 'self'lasketa mukaan argumentteihin!**

Tehtävä 3: Tee 'Pokemon' - luokka jossa on pokemonin 'hp' ja muita arvoja.

Tehtävä 4: Tee luokkaan funktio 'damage' johon viedään arvo siitä miten monta pistettä luokan instanssilta häviää iskussa.

Tehtävä 5: Tee peritty luokka 'Charmander'.

Tehtävä 6: Tee perittyyn luokkaan arvo 'Fire_damage' ja funktio 'Fireattack'.

5 Oliot

Luokan instanssi, olio, luodaan ja sitä käytetään seuraavasti:

```
object = Classname(2,3) # 2 and 3 are arguments
object.funtion()
```

Tässä tehdään `object`-niminen olio jonka *values* arvo on 2ja *default* arvo on `3.`Oliolla on lisäksi funktio `'function'`,jota kutsutaan ilman parametrejä ja joka ei palauta mitään. Olioon on mahdollista lisätä ominaisuuksia lennosta. Tämän ominaisuuden ansiosta on helppo toteuttaa toiminnallisuus erillisiin funktioihin ja sitä mukaan kun ohjelma edistyy ja olio tarvitsee uusia toimintamalleja, ne liitetään yksittäiseen olioon.

```
def MyFunction(self)
      return """Returnvalue"""


object.fun = MyFunction
object.fun()
```

Yllä tehdään funktio ja liitetään se olioon.

Tehtävä 7: Tee olio luokista 'Charmander' ja 'Pokemon'.

Tehtävä 8: Tee kaksi funktio, joka tulostaa jonkun hyökkäyksen (keksi itse :)).

Tehtävä 9: Liitä toinen hyökkäys toiseen olioon ja molemmat toiseen olioon. Käytä hyökkäyksiä.

6 Lista

Listat ovat pythonin yksi tehokkaimmista aseista. Niiden luominen on helppoa ja läpikäyminen vaivatonta. Monesti listojen avulla pystytään korvaamaan if -rakenteita koodista. Seuraavassa määritellään tyhjälista ja siihen lisätään alkio.

```
a=[]
a.append(1) # append 1 to emptylist
```

Lista voi sisältää kaikenlaista tavaraa. Samassa listassa voi olla numeroita, merkkijonoja, toisia listoja, funktioita, tyhjiä listoja jne. Olemassaolevalta listalta voidaan kysyä onko se tyhjä

```
if a!=[]:
     do_something .
```

Kun luodaan listaa, tulee varsin useasti esiin tilanne, jossa `range`-funktio on hyvin käyttökelpoinen.

```
""" G is a list of numbers 1-10"""
G = range(10)
```

`Range`- funktion avulla on mahdollista toteuttaa helposti monenlaisia listoja. Niistä tarkemmin *Python Library referencessä.*

Listan läpikäynti tapahtuu `for`- funktion avulla.

```
for x in Lista:
       print x
       print sin(x)
       if x > 5
             print x + x
```

Jokaiselle listan alkiolle tehdään määrätyt tehtävät.

Tehtävä 10: Tee lista, jossa on viisi kertaa jokin aiemmin tehty funktio.

Tehtävä 11: Aja kaikki listan funktiot 'for':lla.

Tehtävä 12: Lisää listaan numero 1. Aja samalla tavalla läpi kuin ed. tehtävässä.

Tehtävä 13: Tee lista numeroista ja aja kaikki numerot jonkun funktion läpi.

7 Tuple

Eräs vaihtoehtoinen tapa listoille on käyttää `tuplea`.Se on hieman listaa tehokkaampi tapa tallentaa arvoja. `Tuple`määritellään Pythonissa seuraavasti.

```
Tuple_name = 1 , 2 , 3 , 'cat'
""" This is how to nest 2 tuples """
New_My_tuple = (2,3,4) ,Tuple_name
""" One may also separate values """
value1, value2, value3 = Tuple_name
"""Singleton -- Lonely tuple """
Single = (1,)  # <-- Comma !
empty = ()
```

`Tuple`ja lista eroavat toisistaan siten, että `tuplen`arvoja ei voida muuttaa. Tietenkin voidaan luoda uusi `tuple`vanhalla nimellä, mutta yksittäisiä arvoja ei voida vaihtaa.

```
Tuple_name = 1,2,3
List_name = [1,2,3]
List_name[1] = 4  # --> [1,4,3]
Tuple_name[2] = 3
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
  TypeError: object doesn't support item assignment
```

`Tuple`on kätevä tapa esittää esimerkiksi (x,y,z) - koordinaatit. Lisäksi `tuple`sopii hyvin `dictin`avaimeksi, koska sen arvo ei muutu.

Tehtävä 14: Tee 3 tuplea, joissa on Pokemonin koordinaatit. Yhdistä pokemonin nimi ja sen koordinaatit tupleksi. Yhdistä 3 Pokemonia Tupleksi.

8 Palautusarvo

Pyhtonissa on mahdollista palauttaa useampi arvo kerrallaan. Palautusarvoina voidaan myös käyttää toisia funktioita, listoja, olioita yms. Palautus tapahtuu käyttämällä `return`-funktiota.

```
def My_function():
```

```
        return "cat", fun(), [x for x in range(7)]

animal, Myfun , list = My_function()
```

Ylläolevassa koodissa palautetaan kolme asiaa, jotka pitää vain muistaa ottaa kiinni kutsuttaessa funktiota.

Tehtävä 15: Tee funktio jonka palautusarvona on lista ja funktio. Liitä saadut arvot olemassaolevaan olioon.

9 Tiedostoihin jakaminen

Pythonissa tiedostoihin jakaminen ja niiden liittäminen toimii `import`-käskyn avulla. Liitettävän tiedoston tulee olla tavallinen Python-tiedosto.

```
import string
import My_file
```

Tehtävä 16: Tee tiedosto, jossa määritellään yksi funktio. Liitä funktio ja kutsu sitä. Huomioi, että kutsuttaessa funktiota tiedostonimi pitää mainita.

10 Tiedostosta lukeminen ja siihen kirjoittaminen

Tiedostosta lukeminen tapahtuu seuraavasti.

```
data = file(filename)
all_lines = data.readlines
```

Tiedostoon kirjoittaminen:

```
data = file(filename,'w')
s="All saved Data"
data.write(s)
data.close
```

Library referencestä löytyy tarkemmin tietoa eri asetuksia tiedostoon kirjoittamisesta, mm. olemassa olevaan tiedostoon lisäämisestä ja rivi kerrallaan lukemisesta.

Tehtävä 17: Tallenna tiedostoon dataa ja lue se.

11 Poikkeukset

Pythonissa on määritelty paljon erilaisia poikkeuksia. Perusperiaate on kuitenkin sama kuin muissakin kielissä.

```
try:
        print "hello"
except:
        print "failure"
```

Tässä yritettiin kirjoittaa sana `''hello''`, ja jos silloin tulisi mikä tahansa poikkeus, niin siirryttäisiin kohtaan `except`. Oikeasti kuitenkin tulee määrittää millaisia poikkeuksia halutaan saada kiinni. Tarkempi listaus poikkeuksista löytyy *Python library reference* :stä.

Tehtävä 18: Yhdistä tiedostosta lukeminen ja poikkeuksen käsittely.

Tehtävä 19: Ota em. tehtävässä kiinni vain oikea virhe.

12 Yield

Yksi monikäyttöisimpiä Pythonin ominaisuuksia on generaattorit. Näistä esimerkkinä on `yield`. Sitä pystyy käyttämään esimerkiksi iteraattorina.

```
def foo():
        print "First I like to say.."
        yield 1
        print "Second... "
        yield 2
```

```
G = foo()
"G is now generator object"
G.next()
G.next()
G.next()
```

Tässä joka kerran funktiota kutsuttaessa aliohjelman suoritus loppuu *yieldin* tullessa vastaan ja ohjelma pomppaa takaisin kutsuvaan ohjelmaan. Seuraavan kerran, kun aliohjelmaa kutsutaan, suoritus jatkuu katkeenees-ta kohdasta. Aina on mahdollista joko luoda uusi generaattori ja aloittaa alusta tai siirtyä edellisessä generaattorissa seuraavaan vaiheesseen.

Enemmän esimerkkejä yieldin käytöstä löytyy osoitteesta:

`http://linuxgazette.net/100/pramode.html`

Tehtävä 20: Tee funktio, joka kutsuttaessa palauttaa satunnaisluvun ja tiedon siitä monesti funktiota on kutsuttu.

13 Dict

Yksi kätevä tapa listata asioita Pythonissa on `dict.`*Dict:iä* voidaan ajatelle listana {arvo, avain} - pareja. Esimerkki *dict:n* käyttämisestä seuraavassa:

```
dict_name={'value1':'key1', 'value2':key2'}
dict_name[key1]
dict_name[key2]
for k, v in dict_name.iteritems():
      print k + " is a key " + v + " is a value"
```

Tehtävä 21: Tee puhelinluettelon alku (Loput kotitehtäväksi :)).

14 Regexp

Pythonissa on paljon valmiiksi tehtyjä kirjastoja ja usein on ohjelmoitaessa syytä kysyä, löytyykö valmis toteutus jostain olemassa olevasta kirjastosta. Esimerkkinä hyvästä kirjastosta esitellään regexp (Regular expression, re). Kirjasto sisältää paljon merkkijonojen käsittelyyn liittyviä funktioita, joita voidaan käyttää tehokkaasti mm. parsereita ohjelmoitaessa.

Seuraavassa malli, mitä `re`-kirjastolla saadaan aikaiseksi.

```
import re
Reg = re.compile('A+')
H1 = "A"
H2 = "B"
print Reg.match(H1) # yes
print Reg.match(H2) # No found
```

Tehtävä 22: Tee lista merkkijonoja ja hae siitä kaikki "K" :lla alkavat sanat re -kirjaston avulla.

15 Lähteitä

Pythonista löytyy paljon lähteitä netistä. *Python tutorial* sisältää kaiken olennaisen ohjelmoinnin aloittamiseksi. *Python Library Reference* sisältää kattavan esityksen suurimmasta osasta Pythonin kirjastoista dokumentaatioineen ja esimerkkeineen. *Numerical Python* sisältää lähinnä työkaluja laskennan toteuttamiseksi pythonilla. *PyOpenGl* kertoo kuinka python voidaan liittää *OpenGl* - grafiikkakirjastoon.

Python tutorial:
http://docs.python.org/tut/tut.html
Python library reference
http://docs.python.org/lib/lib.html
Numerical Python
http://www.pfdubois.com/numpy/
PyOpenGl http://pyopengl.sourceforge.net/