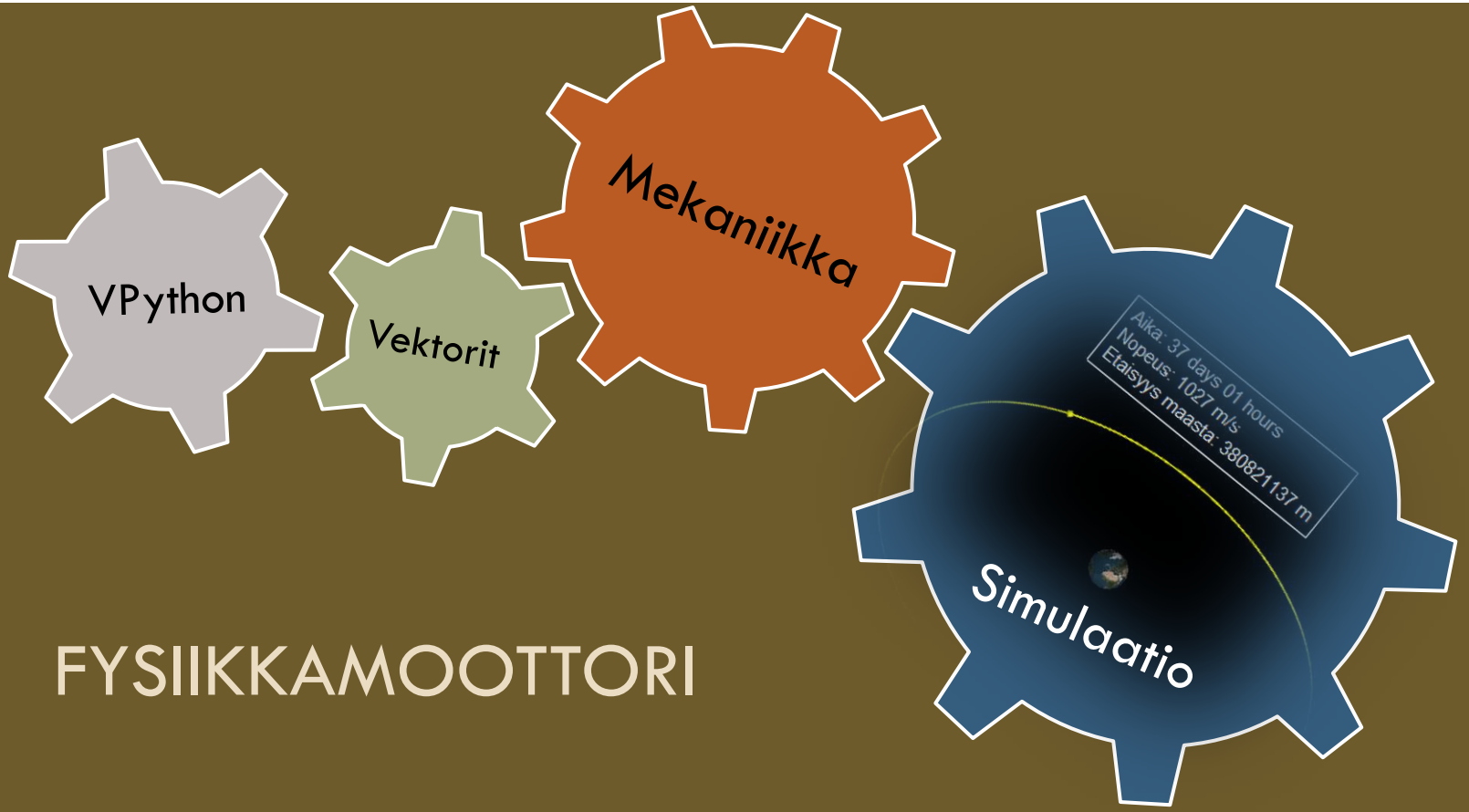
SIMULAATIOT JA VPYTHON 7



FYSIIKKAMOOTTORI

Kari Peisa 2020

Python (>=)3.5.3 ohjelmointikieli ja VPython 7 grafiikkamoduuli

Ohjelmointiympäristön asennus

- Ympäristön (VPython versio 7, Windows) lataaminen Python-ohjelmaan vaatii, että versio Python 3.5.3 tai uudempi on asennettuna tietokoneella https://www.python.org/downloads/
- VPython moduli ladataan versioon DOS-ikkunan komentokehotteella

  ```
  C:\> pip install vpython
  ```

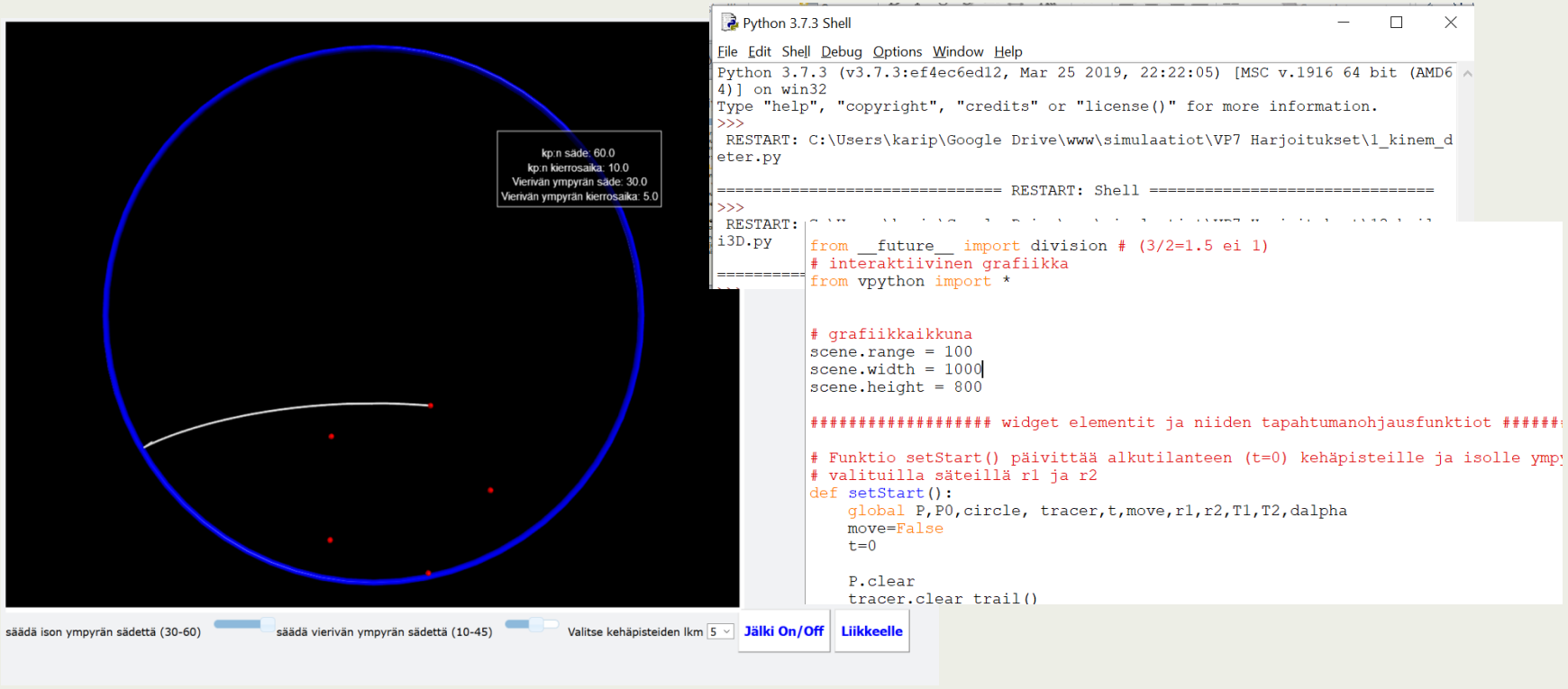
  (kts. https://vpython.org/presentation2018/install.html)

  Huom! Jos on useita Python versioita koneella, lataaminen pip -ohjelmalla edellyttää, että PATH -ympäristömuuttujassa on viittaus sen Python-version kansioihin, johon paketti aiotaan asentaa. Jos ei ole, pitää siirtyä siihen kansioon, jossa kyseisen version pip.exe sijaitsee. Vaihtoehtoisesti voi asentaa Anaconda -version hallintajärjestelmän.
- Moduli otetaan käyttöön Python-koodissa komennolla

  ```
  from vpython import *
  ```
- VPython versiota 6 (Classic) ei enää tueta, vaan kehitys on siirtynyt versioon 7, joka toimii täysin eri periaatteella ja eri ympäristössä kuin versio 6. Version 6 asentamista, toimintakuntoon saattamista ja käyttöä ei tässä käsitellä.

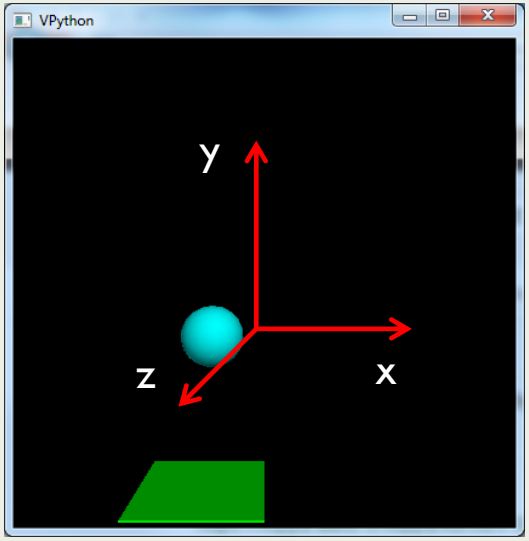
Ohjelmointiympäristön osat

- Python Shell komentotulkki (Python on dynaaminen kieli)
- Jokin koodieditori esimerkiksi Pythonin oma IDLE
- Grafiikkaikkunana toimii selainohjelma



Grafiikkaympäristöstä

- VPython on erityisesti suunniteltu 3D kineettisten tapahtumien simuloimiseen
- Python kieli on perinteinen olio-ohjelmointikieli ja sen syntaksi on helposti omaksuttavaa.
- Grafiikkaikkuna avautuu automaattisesti selain-ohjelmaan ajettaessa (run) ohjelma
- Koordinaatiston oletussuunta ohessa
- Ikkunan (oletus) keskus (0,0,0)
- Katselukulman vaihto:
  - Hiiren oikea painike alas ja vetäminen = katselukulman (viewpoint) pysty- ja vaakasuuntainen kierto
  - Hiiren molemmat painikkeet alas painettuna vetäminen tai työntäminen = loitonna tai lähennä näkymää



Koodieditorista IDLE

- IDLE tunnistaa syntaksin värein. Python-kielen syntaksi noudattaa tarkoin ohjelmalohkojen sisennyksiä
- Jos kopioi IDLE-editoriin Python koodia esimerkiksi webistä, eivät mm. sisennykset eivätkä kaikki merkistöt tule välttämättä syntaksin mukaisesti liitetyksi
- Erityistä: Kun IDLE-editorissa talletetaan Python-koodi, tulee tiedostonimen perään itse laittaa tarkennin .py, sillä muuten se jää pois ja kääntäjä tulkitsee tiedoston tekstitiedostoksi

Ohjelmointi VPython modulilla

vector objekti

- 3D objekti (vektorimuuttuja), jonka avulla kaikki fysiikka toteutetaan simulaatiossa. Vektori-objektia ei voi tulostaa grafiikkaikkunaan, se on vain laskemista varten. (on olemassa arrow -grafiikkaobjekti)
- Luotaessa uusi vektoriobjekti, nimetään se ja annetaan konstruktorissa vektorin komponentit

```
velocity=vector(10,5,5)# vektori 10i+5j+5k
```

- Vektorioperaatioita (x ja y vektoriobjekteja)

Yhteen- ja vähennyslasku x+y, x-y ja reaaliluvulla kertominen 3*x

mag(x) ja mag2(x)	vektorin x pituus ja sen neliö
norm(x)	x:n suuntainen yksikkövektori
dot(x,y) tai x.dot(y)	x:n ja y:n pistetulo (huom x*y → virheilmoitus ajossa)
cross(x,y) tai x.cross(y)	x:n ja y:n ristitulo
proj(x,y) tai x.proj(y)	x:n projektiovektori y:n suunnassa
diff_angle(x,y) tai x. diff_angle(y)	x:n ja y:n välinen kulma (radiaaneissa)

Ohjelmointi VPython modulilla

- Graafiset VPython-objektit tuodaan ikkunaan kutsumalla objektin konstruktoria, jolla asetetaan riittävä määrä perusominaisuuksia (ainakin paikka ja koko). Esimerkiksi pallo-objektin luominen ja identifioiminen

```
ball1 = sphere(pos=vector(-5,0,0), radius=3, color=color.cyan)
```

- Graafisille objekteille alustetaan konstruktorissa sisäänrakennetut perusattribuutit, joihin viitataan pistenotaatiolla ja nimellä. Näiden arvoja voidaan myös muuttaa. Esimerkiksi pallo-objejktille `sphere()` mm.
  - `pos` paikkavektori , `ball1.pos=vector(-5,0,0)`
  - radius pallon säde, `ball1.radius=14.5`
  - `color` väri, jonka arvo voidaan asettaa color -luokan attribuuteilla (red, green, blue, yellow, magenta, cyan, orange, black, white),

```
ball1.color=color.green
```

  - `ball1.pos.x, ball1.pos.y ja ball1.pos.z` `objektin` paikkavektorin koordinaatit

Ohjelmointi VPython modulilla

- Graafisille objekteille ohjelmoija asettaa uusia (omia) ominaisuuksia esittelemällä pistenotaatiolla uuden attribuutin nimi ja asettamalla sille jokin arvo. Attribuutin tyyppi määräytyy dynaamisesti arvon perusteella
  - `ball.mass=12.4`
  - `ball.velocity=vector(10,10,0)`
- Graafisen objektin liikuttaminen ikkunassa toteutetaan pelkästään päivittämällä sen paikkavektoria (alla dt on skalaari, jolla määritetään lyhyt aikaväli simulaation peräkkäisille ruuduille)
  - `ball.pos += ball.velocity*dt`

    `(lyhennys muodosta ball.pos = ball.pos + ball.velocity*dt)`
- VPython objektien ominaisuudet ja käyttö löytyvät VPythonin online –dokumentaatiosta https://www.glowscript.org/docs/VPythonDocs/index.html

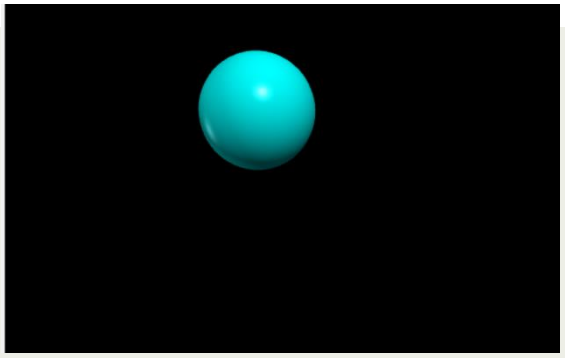
Ohjelmointi VPython modulilla

Esimerkki yksinkertaisesta toimivasta ohjelmasta, jossa pallo liikkuu tasaisella nopeudella (oletuksena ikkunan katselukulma loittonee automaattisesti, kun objekti yrittää poistua ikkunasta)

```
from vpython import *

ball=sphere(pos=vector(-5,0,0),radius=3,color=color.cyan)
ball.velocity=vector(1,1,1)
dt=0.01 # virtuaalinen aika-askel fysiikassa

while(1):
  rate(50) # ikkunoiden päivitysviive 1/50 sek
  ball.pos += ball.velocity*dt
```



Python kielen syntaksista

- Muistuttaa tavanomaisia proseduraalisia olio-ohjelmointikieliä, mutta syntaksi helpompi omaksua, koska
  - Looginen rakenteellisuus lohkoineen määräytyy tarkoilla sisennyksillä, ei sulkeilla eikä lauseiden erotinmerkeillä (puolipiste erottaa kuitenkin samalla rivillä olevat lauseet toisistaan)
  - Edellisen vuoksi Python kieli on helposti luettavaa (readability). Lisäksi se on hyvin ilmaisuvoimaista (expressiveness), mikä näkyy mm. listojen läpikäynneissä (kts. myöh.)

Seuraavissa Python kielen syntaksia kuvaavissa esimerkeissä on merkitty punaisella kohdat, jotka ovat kielen omia syntaktisia elementtejä. Mustalla on merkitty puolestaan ohjelmoijan itsensä määräämät lausekkeet tai lauseet

Laskeminen Pythonilla

- Skalaarimuuttujien aritmeettiset operaattorit ja niiden suoritusjärjestys

1. `a**b` potenssiin korotus a^b
2. `+x -x` positiivisuus/negatiivisuus
3. `*  /  %` kertominen, jakaminen ja jakojäännös
4. `+  -` yhteenlasku ja vähennyslasku

- Matemaattiset funktiot kirjastossa math

`from math import *` (kuuluvat VPython-moduuliin)

- Esimerkiksi (x ja y reaalilukuja)

`sqrt(x)`, **`sin(x)`**, **`cos(x)`**, **`tan(x)`**, **`pow(x,y)`** (= **`x**y`**), **`exp(x)`**, **`log(x[,kanta])`**, **`degrees(x)`**, **`radians(x)`** sekä vakiot

`pi` (=3.1415...) ja **`e`** (=2.7182...)

Ohjausrakenteet Pythonissa

- if -lause (valinta)

```
if ehto1:
   lause1   # risuaita aloittaa kommentin
            # Sisennys on tarkasti määrätty
            # ja on tässä 3 merkkiä
   lause2
elif ehto2:
   lause3
   lause4
else :
   lause5
   lause6
lause7     # lause 7 ei kuulu enää if-lauseeseen
```

Ohjausrakenteet Pythonissa

- while -lause (toisto)

```
while ehto:
    lause1
    lause2
    lause3
lause4     # lause 4 ei kuulu enää while-lauseeseen
```

- Loogiset vertailuoperaattorit samat kuin esimerkiksi C kielessä

`<` `<=` `>` `>=` `==` `!=` sekä lisäksi

Olioiden vertailussa	`is`	`is not` (onko viittaus samaan olioon)
Kuuluminen johonkin kokoelmaan	`in`	`not in` (esimerkiksi onko listassa)

Ohjausrakenteet Pythonissa

- Loogiset vertailulausekkeet, jotka palauttavat joko False (0 tai [] tai {}) tai True (≠False), muodostetaan samalla periaatteella kuin esimerkiksi C kielessä
- Operaattoreiden suoritusjärjestys
  1. Aritmeettiset operaattorit suoritetaan aina ennen vertailuoperaattoreita
  2. Vertailuoperaattoreiden suoritusjärjestys

```
<  <=  >  >=  ==  !=
is   is not
in   not in
not x
and
or
```

Ohjausrakenteet Pythonissa

- for –lause (toisto)

```
for muuttuja in sarja:
    lause1
    lause2
else:
    lause3
    lause4
```

else –osa on valinnainen, joka suoritetaan, mikäli muuttujan arvo ei ole mikään sarjassa oleva

Edellä sarja on jokin seuraavista kokoelmatyypeistä (kts. myöhemmin Rakenteiset tietotyypi):

lista []	list
monikko ()	tuple
merkkijono ” ” tai ’ ’	string

Ohjausrakenteet Pythonissa

- continue ja break–operaattorit
  - `continue` keskeyttää toistokierroksen ja suoritus siirtyy toistolauseen ehto-osaan
  - `break` keskeyttää toistokierroksen ja suoritus siirtyy toistolauseen jälkeiseen lauseeseen

Rakenteisista tietotyypeistä

- Lista
  - Lista voi sisältää mitä tahansa perustietotyyppiä olevia alkioita tai niiden sekoitelmia
  - Listaa voidaan muokata vapaasti, lisätä ja poistaa alkioita tai muuttaa niiden järjestystä listaolion omilla jäsenfunktioilla.
  - Funktioiden range avulla luodaan kokonaislukulista ja funktion arange avulla liukulukulista. Niitä käytetään usein toistorakenteissa (esim. for -rakenteessa)
  - Listan alkioon viitataan C kielen mukaisesti hakasulkeilla ja indeksillä alkaen 0:sta `lista[indeksi]`
- Esimerkkejä

```
lista1= [”Kalle”,10,”Liisa”,12]
range(start, end, step)   arange(start, end, step)
lista2= range(5) # 5 pituinen kokonaislukulista [0,1,2,3,4]
lista3=range(5,10) # [5,6,7,8,9]
lista4= arange(0,2,0.5) # liukulukulista [0.,0.5,1.,1.5,2.]
lista1.append(”Ville”) #[”Kalle”,10,”Liisa”,12,”Ville”]
```

Rakenteisista tietotyypeistä

- Listan generoiminen, esimerkkejä
  - Lista satunnaislukuja
    - ```
      lst=[]
      for i in range(4):
      lst.append(rand())
      ```
    - ```
      lst=[rand() for i in range(4)]
      ```
- Listan läpikäyminen, esimerkkejä

```
lst=[”Kalle”,”Maija”,”Ville”,”Jaana”]
```

- ```
  for i in range(len(lst)):
      print(lst[i])
  ```
- ```
  for i in lst:
      print(i)
  ```

Huom! Viimeisessä esimerkissä muuttuja i viittaa listan alkioon, ei indeksiin

Rakenteisista tietotyypeista

- Monikko (tuple)
  - On kuten lista, mutta alkioita ei voi muokata luonnin jälkeen
  - Monikko on sisäisesti taulukko kun taas lista on yksiulotteinen sarjarakenne
  - Monikolla ei ole sisään rakennettuja käsittelyfunktioita kuten listalla
  - Esimerkkejä monikosta

```
tuple1= (”Kalle”,”Ville”,”Liisa”,”Maija”)
tuple2=(”Kalle”,”Ville”,(”Liisa”,”Maija”))
   tuple1[2] palauttaa arvon ”Liisa”
   tuple2[2] palauttaa arvon (”Liisa”,”Maija”)
   tuple2[2][1] palauttaa arvon ”Maija”
```

Pääohjelma ja omat funktiot

- Funktio määritellään pääohjelman sisällä omassa lohkossaan (sisennetty koodirivistö), jonka otsikkorivi on muotoa

  `def` funktionnimi(argumentit$_{opt}$) **`:`**

- Funktion määrittely tulee olla koodissa ennen kuin sitä kutsutaan pääohjelmassa
- Funktiossa määritellyt paikalliset muuttujat eivät näy pääohjelmaan (voivat olla siis samannimisiä). Mikäli haluaa käyttää pääohjelmassa määriteltyä muuttujaa `var`, on se esiteltävä funktion rungossa muodossa `global var`
- Jos funktion halutaan palauttaa jokin arvo, tulee sen suoritus päättyä **`return`** –lauseeseen
- Funktion suorituksen jälkeen ohjelma palaa funktiokutsua seuraavaan lauseeseen pääohjelmassa

Pääohjelma ja omat funktiot

- Esimerkki proseduraalisen funktion (aliohjelma) käytöstä

```
lista1=[-1,0,1]
#Määritellään funktio, joka neliöi sille annetun listan alkiot
def nelioiLista(lista):
    for i in range(len(lista)):
        lista[i]=lista[i]**2
#neliöidään listan lista1 alkiot ja tulostetaan
nelioiLista(lista1)
for i in lista1:
    print(i)
```

  - Huom! Tyypin tarkistuksia ei tehdä, koska dynaaminen kieli. Jos funktiota kutsutaan väärän tyyppisellä parametrilla, seuraa ajonaikainen virhe
  - Perustietotyypit (myös string) välitetään arvoparametrina, rakenteiset tyypit viiteparametrina
  - Yllä oleva funktio palauttaa:

```
1
0
1
```

Pääohjelma ja omat funktiot

- Esimerkki funktion käytöstä

```
lst=[0,1,2,3,4]
def summaaLista(lista):
    summa=0             #paikallinen muuttuja
    for i in lista:
        summa+=i        #i viittaa listan alkioon, ei indeksiin
    return summa
#tulostetaan komentotulkkiin listan alkioiden summa
print("Listan alkioiden summa ",summaaLista(lst))
```

Yllä funktio palauttaa: >>>

Listan alkioiden summa 10

Muuttujien näkyvyys ja funktioiden tiedonvälitys

- Paikallinen muuttuja ei näy pääohjelmassa

```
a=2
def fun():
    a=3

fun()
print(a)
```

```
** Python Shell **
>>>
2
```

- Kopiovälitys (PassByCopy) perustietotyypeille

```
a=2
def fun(luku):
    luku=5

fun(a)
print(a)
```

```
** Python Shell **
>>>
2
```

- Viitevälitys (PassByReference) tietorakenteille ja VPython objekteille (olioille)

```
a=[1,2]
def fun(lst):
    lst[0]=5

fun(a)
print(a)
```

```
** Python Shell **
>>>
[5, 2]
```

```
a=sphere(pos=(0,0,0),radius=1)
def fun(ball):
  ball.radius=5

fun(a)
print(a.radius)
```

```
** Python Shell **
>>>
5.0
```

Varovaisuutta funktioiden tiedonvälityksessä

- Edellisessä esimerkissä tuli esille, että listat välitetään viitevälityksellä eli funktio käsittelee argumentissa välitettyä listaa kutsuvan ohjelman tason listana eikä paikallisena muuttujana. Tässä on kuitenkin poikkeuksia. Esimerkiksi, jos funktio muuttaa välitetyn listan rakenteen kokonaan uusiksi, luo funktio paikallisen muuttujan.

```
a=[[1,2],[3,4]]
def fun(lst):
        lst=[5,6]
fun(a)
print(a)
```

```
** Python Shell **
    >>>
[[1,2],[3,4]]
```

Pythonin avainsanat

and assert break class continue def del elif else except exec finally for from global if import in is lambda not or pass print raise return try while yield

Näitä sanoja ei voi käyttää muuttujien niminä