Ohjelmoinnin perusteet Y Python

T-106.1208

31.1.2011

Luentopalaute kännykällä käynnissä!

- Ilmoittaudu mukaan lähettämällä ilmainen tekstiviesti "Vast ilmo"numeroon 16403 tai vaihtoehtoisesti täytä lomake osoitteessa http://opey.experq.com/register
- Ilmoittautuneille lähetetään luennon jälkeen yksi kysymys luennosta.
- Kysymykseen voi vastata ilmaisella tekstiviestillä, joka on muotoa "Vast *num* *vapaa palaute*", esim. "Vast 3 enemmän esimerkkejä".
- Myös ne ilmoittautuneet, jotka eivät ole kysymystä luennon jälkeen saaneet, voivat lähettää vapaamuotoista palautetta ja kysymyksiä numeroon 16403 tekstiviestillä (max 160 merkkiä), joka on muotoa "Vast *vapaa palaute*", esim "Vast enemmän esimerkkejä".
- Yhteystietoja ei luovuteta kolmansille osapuolille eikä vastauksia liitetä takaisin puhelinnumeroihin.

Funktiot

- Tähän asti esitetyt ohjelmat ovat oleet hyvin lyhyitä. Todellisessa elämässä tarvitaan kuitenkin ohjelmia, jotka ovat tuhansien tai kymmenien tuhansien rivien mittaisia.
- Jos koko ohjelma muodostuu tuolloin yhdestä pääohjelmasta, on sen rakenteen ja toiminnan hahmottaminen vaikeaa.
- Lisäksi ohjelmissa tehdään usein sama asia monta kertaa (mutta ei heti peräkkäin niin, että voitaisiin käyttää toistokäskyä). On työlästä kirjoittaa sama koodi monta kertaa.
- Ratkaisu: käytetään funktioita.
- Funktio on ohjelmakoodin osa, jolle on annettu oma nimi.
- Funktion nimeä käyttämällä voidaan *kutsua* funktiota eli pyytää funktiota suoritettavaksi muualta ohjelmasta.

Funktioiden käytön etuja

- Ohjelmakoodi selkiytyy.
- Saman asian tekevä koodi joudutaan kirjoittamaan vain kerran.
- Ohjelman testaus helpottuu.
- Ohjelman kirjoittaminen ryhmätyönä helpottuu.
- Ohjelman osia on helpompi käyttää uudelleen toisissa ohjelmissa.

Esimerkki: kolmioiden tulostus

- Halutaan kirjoittaa ohjelma, joka tulostaa seuraavan kuvion.

```
  *
 ***
*****
  *
 ***
*****
  *
 ***
*****
```

- Kuvio muodostuu kolmiosta, joka piirretään 3 kertaa. Kirjoitetaan funktio yhden kolmion tulostamiseen.

Esimerkin koodi

```
def tulosta_kolmio():
    print "  *  "
    print " *** "
    print "*****"


def main():
    tulosta_kolmio()
    tulosta_kolmio()
    tulosta_kolmio()


main()
```

Toinen versio: kutsut toistokäskyn sisällä

```
def tulosta_kolmio():
    print "  *  "
    print " *** "
    print "*****"


def main():
    KERRAT = 3
    for i in range(KERRAT):
        tulosta_kolmio()


main()
```

Parametrit

- Halutaan kirjoittaa ohjelma, jolle annetaan kuljettu matka ja aika (minuutit ja sekunnit erikseen) ja joka tulostaa sitten nopeuden kilometriä kohti.
- Kilometrinopeuden laskeminen sopii hyvin omaksi funktioksi.
- Tarvitaan kuitenkin jokin tapa kertoa funktiolle, mikä on kuljettu matka ja aika.
- Tämä tieto voidaan välittää *parametrien* avulla.
- Parametri on funktion otsikossa sulkujen sisällä annettu nimi, jota voi käyttää funktion sisällä kuin mitä tahansa muuttujaa.
- Kun funktiota kutsutaan, määrätään parametrille tuleva alkuarvo.

Kilometrinopeus, koodi

```
def laske_kilometrinopeus(matka, minuutit, sekunnit):
    aika_sekunteina = minuutit * 60 + sekunnit
    km_sekunnit = int(aika_sekunteina / matka)
    km_min = km_sekunnit / 60
    km_sek = km_sekunnit % 60
    print "Nopeus on %d min %d s/km" % (km_min, km_sek)
```

Kilometrinopeus, koodi jatkuu

```
def main():
    rivi = raw_input("Anna matka kilometreina.\n")
    kilometrit = float(rivi)
    rivi = raw_input("Anna ajan minuutit.\n")
    min = int(rivi)
    rivi = raw_input("Anna ajan sekunnit.\n")
    sek = int(rivi)
    laske_kilometrinopeus(kilometrit, min, sek)


main()
```

Vielä parametreista

- Parametrit saavat funktion kutsussa annetut alkuarvot samassa järjestyksessä kuin parametrit ovat funktion otsikossa.
- Funktion kutsussa parametrina arvo voidaan antaa minä tahansa lausekkeena, jonka arvo voidaan laskea, esimerkiksi:
  - suoraan lukuarvo
  - muuttuja
  - monimutkaisempi lauseke
- Esimerkkejä

```
laske_kilometrinopeus(12.0, 55, 15)
laske_kilometrinopeus(kilometrit, min, sek)
laske_kilometrinopeus(2 * kilometrit, min, sek - 15)
```

Toinen esimerkki: sijoituksen arvokehitys

- Muutetaan edellisellä luennolla esitettyä sijoituksen arvokehityksen laskevaa ohjelmaa niin, että sijoituksen arvon laskeminen tehdään omassa funktiossa.
- Käyttäjä antaa ohjelmalle sijoitettavan pääoman, oletetun vuosituoton ja sijoitusajan.
- Ohjelma laskee ja tulostaa sijoituksen arvon kunkin vuoden lopussa sekä vuoden aikana kertyneen koron.
- Funktiota käyttämällä on helppo laskea arvokehitys usealle eri tuotto-odotukselle.

Arvokehitys, koodi

```
def laske_arvon_kehittyminen(paaoma, aika, tuottopros):
    print "vuosi  vuosikorko  paaoma"
    print "                   vuoden lopussa"
    for i in range(1, aika+1):
        korko = tuottopros / 100.0 * paaoma
        paaoma = paaoma + korko
        print "%4d. %10.2f %12.2f" % (i, korko, paaoma)
```

Arvokehitys, koodi jatkuu

```
def main():
    print "Ohjelma laskee sijoituksen arvon kehittymisen."
    vastaus = raw_input("Anna sijoitettava paaoma.\n")
    summa = float(vastaus)
    vastaus = raw_input("Anna sijoitusaika.\n")
    vuodet = int(vastaus)
    vastaus = raw_input("Anna 1. vuosituotto (%).\n")
    tuotto1 = float(vastaus)
    laske_arvon_kehittyminen(summa, vuodet, tuotto1)
    vastaus = raw_input("Anna 2. vuosituotto (%).\n")
    tuotto2 = float(vastaus)
    laske_arvon_kehittyminen(summa, vuodet, tuotto2)
    vastaus = raw_input("Anna 3. vuosituotto (%).\n")
    tuotto3 = float(vastaus)
    laske_arvon_kehittyminen(summa, vuodet, tuotto3)

main()
```

Arvon palauttavat funktiot

- Usein on tarve saada tieto funktion laskemasta arvosta muualle ohjelmaan.
- Esimerkiksi edellisen luennon murtoluvun sievennys -esimerkissä suurimman yhteisen tekijän laskeminen sopisi hyvin omaksi funktioksi.
- Jotta funktiossa laskettua syt:iä voitaisiin käyttää sievennyksessä, pitää se saada jotenkin tietoon funktion ulkopuolelle.
- Funktio voi välittää tiedon laskemastaan arvosta *palauttamalla* tämän arvon.
- Arvon voi palauttaa `return`-käskyllä. Sen suoritus aina päättää funktion suorituksen.
- Palautetun arvon voi ottaa talteen siellä, missä funktiota kutsuttiin.

Arvon palauttaminen

- Arvo palautetaan `return`-käskyllä:

  `return lauseke`

  `lauseke` voi olla vakio, muuttujan nimi tai monimutkaisempi lauseke.
- Palautettu arvo voidaan ottaa sijoituskäskyllä talteen siellä, missä funktiota kutsuttiin:

  `muuttuja = funktio(parametrit)`
- Palautetun arvon voi myös tulostaa suoraan esimerkiksi `print`-käskyssä:

  `print "Tulos on", funktio(parametrit)`
- Palautettua arvoa voi myös käyttää hyväksi suoraan toisen lausekkeen arvoa laskettaessa:

  `uusi_tulos = 2 * funktio(parametrit) - 5`

Murtoluvun sieventäminen, koodi

```
def laske_syt(luku1, luku2):
    luku1 = abs(luku1)
    luku2 = abs(luku2)
    while (luku1 != luku2):
        if luku1 > luku2:
            luku1 = luku1 - luku2
        else:
            luku2 = luku2 - luku1
    return luku1
```

Murtoluvun sieventäminen, koodi jatkuu

```
def main():
    print "Ohjelma sieventaa antamasi murtoluvun."
    syote = raw_input("Anna osoittaja.\n")
    osoittaja = int(syote)
    syote = raw_input("Anna nimittaja.\n")
    nimittaja = int(syote)
    if osoittaja == 0 or nimittaja == 0:
        print "Ohjelma ei pysty sieventamaan lukua."
    else:
        syt = laske_syt(osoittaja, nimittaja)
        osoittaja = osoittaja / syt
        nimittaja = nimittaja / syt
        print "Sievennettyna: ", osoittaja, "/", nimittaja


main()
```

Funktiokutsu toisen funktion sisällä

- Funktiota voi kutsua myös toisen funktion sisältä.
- Muutetaan edellistä esimerkkiä niin, että myös sievennyksestä tehdään oma funktio, jonka sisältä kutsutaan funktiota `laske_syt`
- Pääohjelmassa kutsutaan sievennyksen tekevää funktiota. Ohjelmaa on kuitenkin muutettu niin, että käyttäjä voi sieventää ohjelman samalla suorituskerralla useita lukuja.

Sieventäminen, koodi

```
def laske_syt(luku1, luku2):
    luku1 = abs(luku1)
    luku2 = abs(luku2)
    while (luku1 != luku2):
        if luku1 > luku2:
            luku1 = luku1 - luku2
        else:
            luku2 = luku2 - luku1
    return luku1
```

Sieventäminen, koodi jatkuu

```
def sievenna():
    syote = raw_input("Anna osoittaja.\n")
    osoittaja = int(syote)
    syote = raw_input("Anna nimittaja.\n")
    nimittaja = int(syote)
    if osoittaja == 0 or nimittaja == 0:
        print "Ohjelma ei pysty sieventamaan lukua."
    else:
        syt = laske_syt(osoittaja, nimittaja)
        osoittaja = osoittaja / syt
        nimittaja = nimittaja / syt
        print "Sievennettyna: ", osoittaja, "/", nimittaja
```

Sieventäminen, koodi jatkuu

```
def main():
    print "Ohjelma sieventaa antamasi murtoluvun."
    jatko = "K"
    while jatko != "e" and jatko != "E":
        sievenna()
        jatko = raw_input("Haluatko jatkaa (K/E)?\n")


main()
```

Esimerkki: ympyrän pinta-ala

- Seuraava esimerkkiohjelma laskee ympyröiden pinta-aloja, kunnes käyttäjä antaa negatiivisen säteen.
- Ympyrän pinta-alan laskeminen on kirjoitettu omaan funktioon.
- Laskemisessa tarvitaan piin arvoa. Se on Pythonissa valmiina olevassa `math`-modulissa vakiona `math.pi`. Jotta moduli saataisiin ohjelman käyttöön, on tiedoston alkuun kirjoitettava

```
import math
```

Ympyrän pinta-ala, koodi

```
import math

def laske_ala(r):
    return math.pi * r * r

def main():
    print "Ohjelma laskee ympyroiden pinta-aloja."
    print "Lopeta antamalla negatiivinen sade."
    syote = raw_input("Anna sade.\n")
    sade = float(syote)
    while sade >= 0.0:
        ala = laske_ala(sade)
        print "Pinta-ala on %.4f." % (ala)
        syote = raw_input("Anna sade.\n")
        sade = float(syote)

main()
```