

Proseduurit ja funktiot



Turun yliopisto
University of Turku

Moduulit ja parametrit



Moduulin parametrit

- Viime tutoriaalin moduulit olivat hyvin tarkkaan rajattuja
 - Esim. käänny, liiku neljä askelta
- ***Parametrisoinnilla*** voidaan huomattavasti parantaa uudelleenkäytettävyyttä
- Esim. samankaltaisista operaatioista koostuvat toimenpiteet:
 - Käänny oikealle, liiku 3 askelta eteenpäin, käänny vasemmalle, 5 askelta eteen
 - Käänny oikealle, liiku 5 askelta eteenpäin, käänny vasemmalle, 2 askelta eteen
 - Käänny oikealle, liiku 1024 askelta eteenpäin, käänny vasemmalle, 65536 askelta eteen
- Voitaisiin abstrahoida moduuliin:
 - "Käänny oikealle, liiku N askelta eteenpäin, käänny vasemmalle, liiku M askelta eteenpäin"



Moduulin parametrit

- ”Käännä oikealle, liiku N askelta eteenpäin, käännä vasemmalle, liiku M askelta eteenpäin”

def kaannyJaLiiku(n, m):

 turnRight()

 for i in range(n):

 moveForward()

 turnLeft()

 for i in range(m)

 moveForward()

kaannyJaLiiku(3, 5)

kaannyJaLiiku(5, 2)

kaannyJaLiiku(1024, 65536)



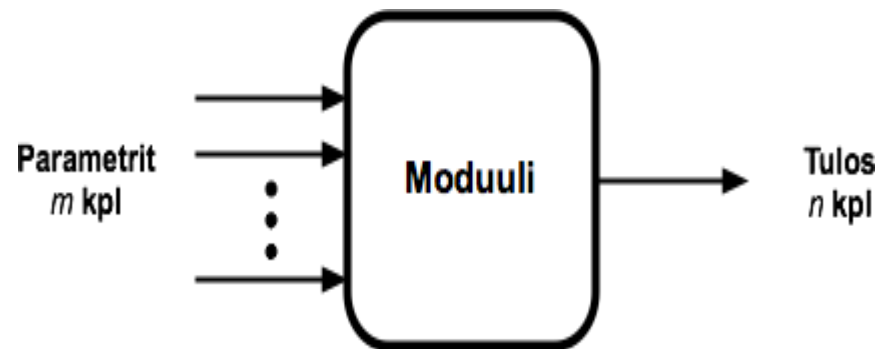
Moduulin parametrit

- Moduulin määrittelyssä olevia parametreja (edellä n) kutsutaan moduulin ***muodollisiksi parametreiksi***
 - Muodolliset parametrit ovat muuttujia
 - Ne osoittavat abstraktin toiminnan osaa, joka voi olla kutsukerroilla erilainen
- Moduulin kutsussa nämä osat kiinnitetään tapauskohtaisilla ***todellisilla parametreilla*** (edellä esim. 1024, 65536)
 - Prosessori korvaa ensin muodolliset parametrit todellisten parametrien arvoilla
 - Sitten prosessori suorittaa moduulin rungon (lohkon) eli toiminnot
 - Rungon suorittamisen jälkeen palataan sille riville, josta moduulia kutsuttiin
 - Todelliset parametrit sisältävät moduulille välitettävän tiedon



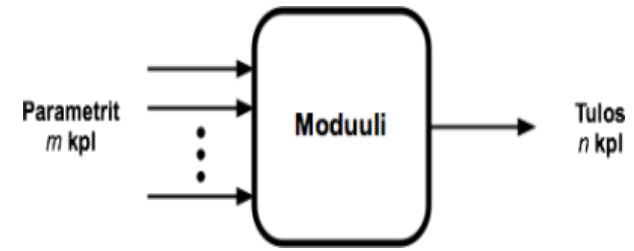
Proseduurit ja funktiot

- Matemaattisesti ajateltuna moduuli on funktio jolla on
 - m syöttöparametria
 - n tulosta





Proseduurit ja funktiot



- Tulosten määrän perusteella moduulit voidaan jakaa kahteen luokkaan tietojenkäsittelytieteen kannalta:
 - **Proseduraalinen** moduuli (eli **proseduuri**)
 - Moduulilla ei ole tulosta ($n = 0$)
 - **Funktionaalinen** moduuli (eli **funktio**)
 - Moduulilla on yksi tulos ($n = 1$)
 - Merkitys perustuu parametrien perusteella laskettuun arvoon

Moduulin kutsu parametreilla

Kaikkien moduulien kutsu on muotoa:

moduulinNimi(tp1, tp2,..., tpN)

- Kutsussa olevat todelliset parametrit tp1, tp2,..., tpN ovat lausekkeita, joille voidaan laskea arvot ennen moduulin suoritusta

Moduulin kutsu parametreilla

Kaikkien moduulien kutsu on muotoa:

```
moduulinNimi(tp1, tp2,..., tpN)
```

- Parametrien lukumäärän ja järjestyksen oltava sama kuin moduulin määrittelyssä
- Kutsussa olevia todellisia parametreja kutsutaan myös argumenteiksi

Moduulin kutsu parametreilla

Kaikkien moduulien kutsu on muotoa:

```
moduulinNimi(tp1, tp2,..., tpN)
```

Esimerkkejä kutsusta:

```
asetSalasana("aoeuhtns")
```

```
tulostaViikonpaiva(22,9,2015)
```

Proseduurit

Proseduraalisen moduulin määrittely

Proseduraalisen moduulin yleinen muoto on:

```
def moduulinNimi(mp1, mp2, ..., mpN)  
    moduulin runko
```

- *mp1, mp2, ..., mpN* ovat moduulin muodolliset parametrit, jotka ovat muuttujia
- Moduulin nimen voi valita itse
 - nimen tulee kuvata moduulin tehtävä



Proseduraalisen moduulin määrittely

- Muodollisilla parametreilla ei ole arvoa ennen moduulin kutsua todellisilla parametreilla (joilla pitää olla arvo)
- Määrittelyä ei voi suorittaa sellaisenaan
- Moduuli suoritetaan vasta kutsuttaessa
- Todelliset parametrit välittyvät muodollisten parametrien arvoiksi

```
summaa(5, 3)
```

```
def summaa(n, m):  
    print (n+m)
```



Proseduraalinen moduuli - esimerkki

```
def kolmellaJaollinen(luku):  
    if luku%3 == 0:  
        print "Kolmella jaollinen"  
    else:  
        print "Ei kolmella jaollinen"
```

kolmellaJaollinen(6)



Proseduraalinen moduuli - esimerkki

```
def kolmellaJaollinen(luku):  
    if luku%3 == 0:  
        print "Kolmella jaollinen"  
    else:  
        print "Ei kolmella jaollinen"
```

Moduulin
määrittely, vielä ei
suoriteta mitään

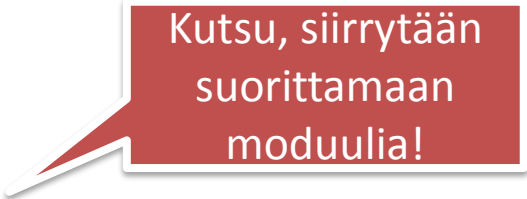
kolmellaJaollinen(6)



Proseduraalinen moduuli - esimerkki

```
def kolmellaJaollinen(luku):  
    if luku%3 == 0:  
        print "Kolmella jaollinen"  
    else:  
        print "Ei kolmella jaollinen"
```

kolmellaJaollinen(6)

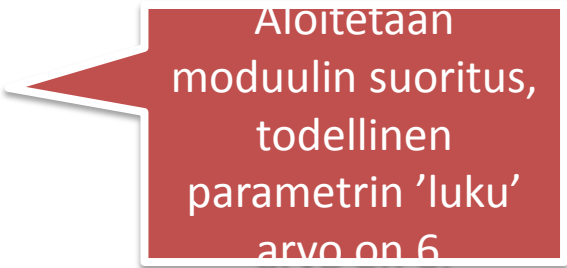


Kutsu, siirrytään
suorittamaan
moduulia!



Proseduraalinen moduuli - esimerkki

```
def kolmellaJaollinen(luku):  
    if luku%3 == 0:  
        print "Kolmella jaollinen"  
    else:  
        print "Ei kolmella jaollinen"
```



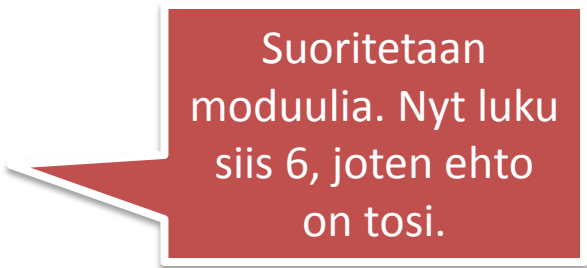
Aloitetaan
moduulin suoritus,
todellinen
parametrin 'luku'
arvo on 6

kolmellaJaollinen(6)



Proseduraalinen moduuli - esimerkki

```
def kolmellaJaollinen(luku):  
    if luku%3 == 0:  
        print "Kolmella jaollinen"  
    else:  
        print "Ei kolmella jaollinen"
```



Suoritetaan
moduulia. Nyt luku
siis 6, joten ehto
on tosi.

kolmellaJaollinen(6)



Proseduraalinen moduuli - esimerkki

```
def kolmellaJaollinen(luku):  
    if luku%3 == 0:  
        print "Kolmella jaollinen"  
    else:  
        print "Ei kolmella jaollinen"
```

Suoritetaan
moduulia.
Tulostetaan
"Kolmella
jaollinen".

If-rakenne päättyy

kolmellaJaollinen(6)



Proseduraalinen moduuli - esimerkki

```
def kolmellaJaollinen(luku):  
    if luku%3 == 0:  
        print "Kolmella jaollinen"  
    else:  
        print "Ei kolmella jaollinen"
```

kolmellaJaollinen(6)

Moduulin
toiminnot
suoritettu,
palataan kutsun

tehneelle riville.

- Kutsulla kolmellaJaollinen(6) moduuli tulostaa siis "Kolmella jaollinen"



Moduulin runko

- Runko koostuu lauseista, joissa viitataan muodollisiin parametreihin
- Muodollisilla parametreilla on moduulia suoritettaessa todellisten parametrien arvot
- Rungon sisäiset (paikalliset) muuttujat ja niiden arvot eivät (yleensä) näy moduulin rungon ulkopuolelle
- Eri moduuleissa voidaan siis käyttää saman nimisiä muuttujia



Muuttujien näkyvyys - Esimerkki

```
def asetaLuku():
```

```
    luku = 3
```

```
asetaLuku()
```

```
print luku
```

Virhe, muuttujaa luku ei ole määritelty moduulin ulkopuolella!



Moduulin runko - miniesimerkki

```
def tulostaSumma(a, b):  
    summa = a + b  
    print summa
```

Kutsuttaessa
todellisten
parametrien arvot
moduulin sisällä:
a = 4
b = 8



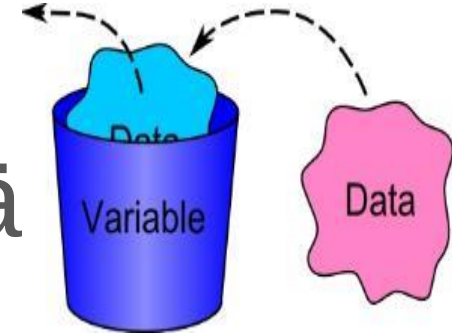
```
a = 4
```

```
c = 8
```

```
tulostaSumma(a, c)
```



Parametrien välityksestä



- Pythonissa muuttujat ovat nimiä, joihin on liitetty jokin arvo
- Arvo sijaitsee jossain muistissa, eikä usein voi muuttua (tyyppikohtaisesti)
- Esimerkkejä tällaisista tyypeistä:
 - Kokonaisluvut, liukuluvut, merkkijonot... Eli kaikki käsiteltyt
- Sen sijaan muuttujat voivat vaihtaa osoitettavaa arvoa, Esim:
 - `x=3`
 - `x=2`
- Tästä johtuen jos annamme parametrina moduuliin arvon, emme voi vaikuttaa moduulin ulkopuolella olevan muuttujan arvoon
- Myöhemmin näemme esimerkkejä tyypeistä, jotka voivat muuttua itsessään



Moduulin runko – esimerkki

```
def eiMuutaArvoja(a, b):
```

```
    a = a + 1
```

```
    b = 2 * b
```

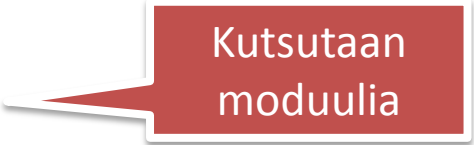
```
    print a, b
```

```
a = 4
```

```
b = 8
```

```
eiMuutaArvoja(a, b)
```

```
print a, b
```



Kutsutaan
moduulia



Moduulin runko - esimerkki

```
def eiMuutaArvoja(a, b):  
    a = a + 1  
    b = 2 * b  
    print a, b
```

```
a = 4
```

```
b = 8
```

```
eiMuutaArvoja(a, b)
```

```
print a, b
```

Alkuperäinen

moduulin
suoritus.

Todelliset
parametrit

a=4, b=8.



Moduulin runko - esimerkki

```
def eiMuutaArvoja(a, b):
```

```
    a = a + 1
```

```
    b = 2 * b
```

```
    print a, b
```

Muutetaan

paikallisen
muuttujan a
arvoksi 5.

Moduulin
ulkopuolisen a:n

arvo ei muutu

```
a = 4
```

```
b = 8
```

```
eiMuutaArvoja(a, b)
```

```
print a, b
```



Moduulin runko - esimerkki

```
def eiMuutaArvoja(a, b):
```

```
    a = a + 1
```

```
    b = 2 * b
```

```
    print a, b
```

Muutetaan

paikallisen
muuttujan b
arvoksi 16.

Moduulin
ulkopuolisen b:n

arvo ei muutu

```
a = 4
```

```
b = 8
```

```
eiMuutaArvoja(a, b)
```

```
print a, b
```



Moduulin runko - esimerkki

```
def eiMuutaArvoja(a, b):
```

```
    a = a + 1
```

```
    b = 2 * b
```

```
    print a, b
```



Tulostetaan 5 16.

```
a = 4
```

```
b = 8
```

```
eiMuutaArvoja(a, b)
```

```
print a, b
```



Moduulin runko - esimerkki

```
def eiMuutaArvoja(a, b):
```

```
    a = a + 1
```

```
    b = 2 * b
```

```
    print a, b
```



Suoritus palaa
kutsuneelle
riville.

```
a = 4
```

```
b = 8
```

```
eiMuutaArvoja(a, b)
```

```
print a, b
```



Moduulin runko - esimerkki

```
def eiMuutaArvoja(a, b):
```

```
    a = a + 1
```

```
    b = 2 * b
```

```
    print a, b
```

```
a = 4
```

```
b = 8
```

```
eiMuutaArvoja(a, b)
```

```
print a, b
```

Tulostetaan 4 8.

Moduulin
suoritus ei siis
muuttanut
näiden
muuttujien

arvoja.



Moduulin alkuehto

- Kaikkia moduuleja ei ole suunniteltu toimimaan kaikilla arvoilla
- Moduulille olisi hyvä kirjoittaa ***alkuehto*** kommenttina ennen määrittelyä
- Alkuehto kertoo ne olosuhteet, joissa moduuli toimii oikein
 - Yleensä tämä tarkoittaa hyväksytyjä parametrien arvoja
 - Ei yleensä vaikuta moduulin suoritukseen, vaan on ohje moduulin käyttäjälle
 - Moduuli suoritetaan myös virheellisillä parametrien arvoilla
- Esim.
alkuehto: $n \geq 0$
def kertoma(n)
...

Funktiot

Fuktionaalisen moduulin määrittely

Pythonissa funktiot määritellään samoin kuin proseduurit

```
def moduulinNimi(mp1, mp2,...,mpN):  
    moduulin runko
```

- Funktion ainut ero on, että se palauttaa arvon moduulin rungossa



Funktion palautusarvo

- Funktion rungossa on oltava yksi (tai useampia) return-lauseita jolla määritellään moduulin palauttama arvo
- Return-lauseen yleinen muoto on:
return lauseke
- return-lause:
 - Päättää moduulin suorituksen
 - Antaa moduulin kutsulle lausekkeen määräämän arvon ja
 - Palauttaa kontrollin kutsukohtaan



Funktion palautusarvo - esimerkki

```
def summa(a, b):  
    return a + b
```

```
print summa(2, 5)
```



Moduulin
määrittely,
ei vielä
suoriteta.



Funktion palautusarvo - esimerkki

```
def summa(a, b):  
    return a + b
```

```
print summa(2, 5)
```



Funktiota
kutsutaan
parametreilla
2 ja 5.



Funktion palautusarvo - esimerkki

```
def summa(a, b):  
    return a + b
```

```
print summa(2, 5)
```

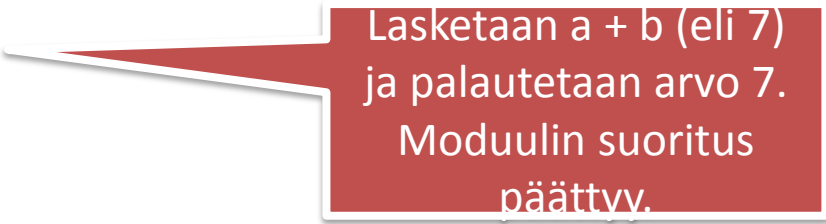
Aloitetaan
funktion
suoritus,
todelliset
parametrit 2
ja 5.



Funktion palautusarvo - esimerkki

```
def summa(a, b):  
    return a + b
```

```
print summa(2, 5)
```



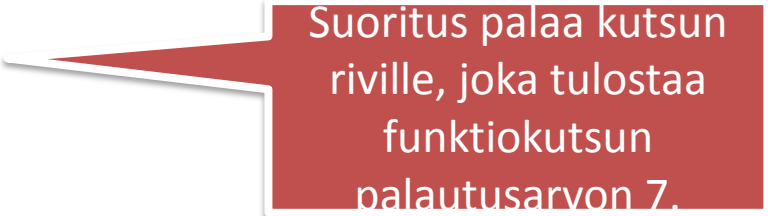
Lasketaan $a + b$ (eli 7)
ja palautetaan arvo 7.
Moduulin suoritus
päättyy.



Funktion palautusarvo - esimerkki

```
def summa(a, b):  
    return a + b
```

```
print summa(2, 5)
```



Suoritus palaa kutsun
riville, joka tulostaa
funktio kutsun
palautusarvon 7.



Funktion palautusarvo - esimerkki

```
def kolmellaJaollinen(luku):
```

```
    if luku%3 == 0:
```

```
        return True
```

```
    else:
```

```
        return False
```

- Funktio palauttaa True, mikäli parametri luku on kolmella jaollinen
- Muuten funktio palauttaa arvon False



Funktion palautusarvo

- Muissa ohjelmointikielissä kirjoitetaan usein palautusarvon tyyppi moduulin määrittelyyn
 - Esim. Javassa `int` jos moduuli palauttaa kokonaisluvun
- Pythonissa ei tarvitse määrittelyssä erottaa proseduuria ja funktiota
- Itse asiassa Pythonissa proseduurit ovat myös funktioita, ja proseduureja ei ole
 - Ero on muissa ohjelmointikielissä tärkeämpi, ja siksi osana kurssia
- Pythonissa proseduuri, jonka suoritus päättyy ilman `return`-lausetta, palauttaa arvon `None` automaattisesti
- Monissa muissa kielissä funktiosta ei voi poistua ilman `return`-lausetta



Funktion palautusarvon käyttö - esimerkkejä

- Funktion palautusarvo voidaan sijoittaa muuttujaan
jaollinen = kolmellaJaollinen(n)
- Funktion totuusarvoista palautusarvoa voidaan käyttää
lausekkeessa
`if kolmellaJaollinen(n):`
 `print "Kolmella jaollinen"`
- Funktiokutsu on siis *lauseke*!



Funktion palautusarvon käyttö

- Funktion palautusarvoa voidaan käyttää moduulikutsuissa
`print max(a, max(b, c))`
- Tässä ensin suoritettaisiin kutsu `max(b, c)`
- Sitten tämä arvo on a:n lisäksi toinen todellinen parametri
- Lopuksi tulostetaan toisen kutsun palautusarvo



Tulostaminen ja palauttaminen ovat eri asioita!

- Yleinen virhe: tulostaminen ja palauttaminen sekoitetaan
- **print**-käsky tulostaa jotain ruudulle
- **return**-lause *palauttaa* arvon funktion kutsujalle

```
def tulostaKolme():  
    print 3
```

```
def palautaKolme():  
    return 3
```

```
a = palautaKolme()
```

```
a = tulostaKolme()
```

a:n arvo on tämän
rivin jälkeen 3

Ja tämän rivin
jälkeen None,
mutta ruudulle
tulostettiin 3



Milloin sulkeita () tarvitaan?

- Sulkeita tarvitaan funktion/proseduurin parametrien ympärillä määriteltäessä:

```
def range(a, b, c):
```

- Ja kun funktiota kutsutaan:

```
n = onkoParillinen(24)
```

```
for i in range(a, b, c):
```

- Huomaa, että print ja return eivät ole funktioita, siksi niihin ei tarvita sulkeita, mutta range on



Modulaarisuuden edut

- Moduulit selkeyttävät algoritmeja ja helpottavat niiden ymmärtämistä
 - Muiden (ja alkuperäisen kirjoittajan) helpompi muuttaa myöhemmin
 - Helpottaa virheiden etsimistä ja oikeellisuuden toteamista
- Käytettäessä moduulia tarvitsee ainoastaan tietää mitä se tekee, ei miten se tehtävänsä täyttää
- Kerran hyvin suunniteltu ja toteutettu moduuli on käytettävissä muuallakin
 - Lähes kaikki ohjelmointikielet sisältävät laajan kirjastot moduuleja