



Muistutus aikatauluista

- (Nämä eivät välttämättä koske avoimen yo:n opiskelijoita Erkki Kailan rinnakkaisella kurssilla)
- Luento 1: kotitehtävät sulkeutuvat 20.9 12:00, ennen tutoriaalia
- Tutoriaali 1 sulkeutuu 21.9 14:00
- Luento 2: kotitehtävät sulkeutuvat 22.9 14:00



Luentopalauteita

- Liikaa asiaa, liikaa opittavaa, en ymmärrä mitään!
 - Sama asia käsitellään onneksi aina kahdesti: kerran luennoilla, ja toisen kerran tutoriaaleissa
 - Tekemällä asiat tulevat paremmin haltuun
 - Lisäksi: Tutoriaalitehtävät yleensä pohjustavat kotitehtäviä!
- Yhdessä tehtävässä oli jotain ENDIF, ELSE, hommeleita ja kummallinen hymiö := ?
 - Pahoittelen! Olivat jääneet viime vuodesta
 - Jos näette näitä vielä, niin ilmoitelkaa.
 - Tehtävä on nyt korjattu
 - Pythonissa ei tarvita ENDIF –sanoja!



Luentopalauteesta

- Superesimerkki:

```
x = 3  
y = (x == 3)
```

- Sijoitetaan ensin x:n arvoksi 3
- Aloitetaan 2. rivin suorittaminen sulkeista:
 - `x == 3` on tosi, x:n arvo todellakin on 3, siis sulkulausekkeen arvo on True
- Nyt suoritetaan vielä välivaihe `y = True`
- y:n arvoksi sijoitetaan True, y on siis totuusarvoinen muuttuja, ja x oli kokonaislukuarvoinen!
- Nyt y:tä voidaan käyttää esim. if-lauseen ehtona:
if y:
 print "x on kolme"



Muuttujien nimeäminen (Pythonissa)

- Muuttujien syntaktisesti oikeat nimet riippuvat ohjelmointikielestä
- Pythonissa nimen:
 - Tulee alkaa kirjaimella (a-z) tai alaviivalla _
 - Saa sisältää kirjaimia, numeroita tai alaviivoja _
 - Pienet ja isot kirjaimet erotellaan
 - Esim **suurinarvo** on eri muuttuja kuin **suurinArvo**
- Käytäntönä on aloittaa muuttujien nimet pienellä kirjaimella
 - Seuraavat sanat kirjoitetaan isolla samassa nimessä:
 - muuttujaJonkaNimiOnTodellaPitka
 - Seuraavilla kursseilla samat käytännöt



Muuttujien nimeäminen 2

- Koska nimet voivat olla monta merkkiä, muuttujat:

$x = 1$

$y = 2$

$xy = 1$

$x3 = 3$

- ovat kaikki eri muuttujia.
- Yhden muuttujan nimi on siis tässä "x3"
- Toisaalta voidaan kirjoittaa:

$xy = x*3*y$

- jolloin muuttujaan xy sijoitetaan tulo $x*3*y$

Toisto ja modulaarisuus



Turun yliopisto
University of Turku



Päivän sisältö

- Ohjausrakenne: toisto
 - Definiitti toisto
 - Indefiniitti toisto
- Modulaarisuus

Toisto





Toisto?

- Mihin tarvitaan toistoa?

```
print "2 * 1 = 2"
```

```
print "2 * 2 = 4"
```

```
print "2 * 3 = 6"
```

```
print "2 * 4 = 8"
```

```
print "2 * 5 = 10"
```

```
·  
·  
·
```



Toisto



- Peräkkäisyys ja valinta eivät riitä algoritmeissa tarvittavan yleisyyden saavuttamiseksi
- Riittävä yleisyys saavutetaan **toiston** (*repetition*) eli **iteraation** avulla
 - Toistoa kutsutaan myös **silmuksi** (*loop*) tai **toistorakenteeksi**



Toistorakenteet

- Definiitti toisto
 - Toistokertojen määrä tunnetaan etukäteen
 - Toistokertojen määrä ei voi muuttua toistorakenteen suorituksen aikana
 - Joskus lyhennysmerkintä: korvaa “kopioi-liitä –menetelmän”
- Indefiniitti toisto
 - Dynaaminen toistorakenne, jossa toistokertojen määrä määräytyy toiston aikana
 - Voi määräytyä *toistoehdon* avulla niin, ettei määrää tiedetä ennen toiston suorituksen aloittamista
 - Lisää aidosti kielen ilmaisuvoimaisuutta
 - Huolehdittava siitä, että toisto päättyy joskus!

Definiitti toisto (“for”)

Definiitti toisto on muotoa

```
for x in lista:  
    toiminto
```

- Suoritetaan *toiminto* kerran kutakin listan L alkiota kohden
- Voidaan viitata vuorossa olevaan listan alkioon
 - Toiminto voi siis kohdistua eri objektiin eri toistokerroilla



Mikä “lista”? Esittelyssä: range

- Usein käydään läpi lukuja joltain väliltä
- Tähän voidaan käyttää **range**-funktiota
 - Funktiot käsitellään tarkemmin ensi viikolla
- Esimerkiksi
for i in range(N):
 toiminto

Tässä toiminto suoritetaan N kertaa siten, että

- Ensimmäisellä toistokerralla i:n arvo on 0, toisella kerralla 1,
 - Viimeisellä N-1, ei siis N!



Mikä “lista”? Esittelyssä: range

- Voidaan myös aloittaa jostain tietyistä luvusta ja lopettaa toiseen lukuun
for i in range(M, N):
 toiminto
- Tässä toiminto suoritetaan aloittaen arvosta M ja lopettaen N-1:een
- Esim. luvut 1,2,3,...,8,9,10
range(1,11)



Mikä “lista”? Esittelyssä: range

- Jos alku- ja loppuarvot ovat samat, silmukkaa ei suoriteta ollenkaan:

```
for i in range(5,5):  
    print “tätä ei suoriteta”
```



Askeltava toisto - esimerkki

```
summa = 0
```

```
for i in range(1,6):
```

```
    summa = summa + i
```

Muuttuja i saa siis
arvot 1, 2, 3, 4, 5 ja
suorittaa toiminnot
kullakin i:n arvolla



Askeltava toisto - esimerkki

summa = 0

for i in range(1,6):

 summa = summa + i

Muuttuja i saa siis
arvot 1, 2, 3, 4, 5 ja
suorittaa toiminnot
kullakin i:n arvolla

- 1. toisto, i = 1: sijoitetaan summaan arvo $0 + 1$



Askeltava toisto - esimerkki

summa = 0

for i in range(1,6):

 summa = summa + i

Muuttuja i saa siis
arvot 1, 2, 3, 4, 5 ja
suorittaa toiminnot
kullakin i:n arvolla

- 1. toisto, i = 1: sijoitetaan summaan arvo $0 + 1$
- 2. toisto, i = 2: sijoitetaan summaan arvo $1 + 2$



Askeltava toisto - esimerkki

summa = 0

for i in range(1,6):

 summa = summa + i

Muuttuja i saa siis
arvot 1, 2, 3, 4, 5 ja
suorittaa toiminnot
kullakin i:n arvolla

- 1. toisto, i = 1: sijoitetaan summaan arvo $0 + 1$
- 2. toisto, i = 2: sijoitetaan summaan arvo $1 + 2$
- 3. toisto, i = 3: sijoitetaan summaan arvo $3 + 3$



Askeltava toisto - esimerkki

summa = 0

for i in range(1,6):

 summa = summa + i

Muuttuja i saa siis
arvot 1, 2, 3, 4, 5 ja
suorittaa toiminnot
kullakin i:n arvolla

- 1. toisto, i = 1: sijoitetaan summaan arvo $0 + 1$
- 2. toisto, i = 2: sijoitetaan summaan arvo $1 + 2$
- 3. toisto, i = 3: sijoitetaan summaan arvo $3 + 3$
- 4. toisto, i = 4: sijoitetaan summaan arvo $6 + 4$



Askeltava toisto - esimerkki

summa = 0

for i in range(1,6):

 summa = summa + i

Muuttuja i saa siis
arvot 1, 2, 3, 4, 5 ja
suorittaa toiminnot
kullakin i:n arvolla

- 1. toisto, i = 1: sijoitetaan summaan arvo $0 + 1$
- 2. toisto, i = 2: sijoitetaan summaan arvo $1 + 2$
- 3. toisto, i = 3: sijoitetaan summaan arvo $3 + 3$
- 4. toisto, i = 4: sijoitetaan summaan arvo $6 + 4$
- 5. toisto, i = 5: sijoitetaan summaan arvo $10 + 5$
- Silmukka suoritetaan 5 kertaa i:n arvoilla 1, 2, 3, 4 ja 5



Askeltava toisto - esimerkki

```
for i in range(1,11):  
    print "2 *", i, " = ", (i * 2)
```

Tulostaa:

2 * 1 = 2

2 * 2 = 4

2 * 3 = 6

⋮

2 * 10 = 20



Askeltava definiitti toisto

- Jos askel poikkeaa yhdestä, se voidaan antaa **rangelle** kolmanneksi parametriksi
- `range(alkuarvo, loppuarvo, askel)`
- Esimerkiksi
 for i in range(1,10, 2):
 toiminto
Suorittaa toiminnon i:n arvoille 1,3,5,7,9



Edellinen esimerkki toisella tavalla

```
for i in range(2, 21, 2):  
    print "2 *", (i/2), "=", i
```

Tulostaa:

2 * 1 = 2

2 * 2 = 4

2 * 3 = 6

.

.

2 * 10 = 20

Viime esimerkissä:

```
for i in range(1,11):  
    print "2 * ", i, "=", (i * 2)
```




Askeltava definiitti toisto

- Askel voi olla myös negatiivinen

- Esimerkiksi

```
for i in range(10,0,-1):  
    print i
```

- Nyt *i* saa arvot 10, 9, 8, ..., 1
 - Loppuarvo 0 jää jälleen pois



Sisäkkäiset toistolauseet

- Kuten ehtolauseita, myös toistolauseet voivat sisältää toisia toistolauseita
- Esimerkiksi:

```
for i in range(5):  
    for j in range(5):  
        print i, j
```

Tulostaa 0 0 0 1 0 2 0 3 0 4 1 0 1 1 1 2 1 3 1 4
(eri riveille)

- Sisemmän silmukan laskuri (eli j yllä) kannattaa nimetä eri tavalla kuin ulomman



Sisäkkäiset toistolauseet

- Sisempi toistolause voi käyttää myös ulomman laskurimuuttujaa

- Esimerkiksi:

```
for i in range(3):
```

```
    for j in range(i):
```

```
        print "sisempi silmukka"
```

Indefiniitti toisto (toistojen lukumäärää ei tiedetä)

```
while ehto:  
    toiminto
```

- Silmukan *toiminnot* suoritetaan niin kauan kuin *ehto* on tosi
- Jos ehto heti epätosi, ei suoriteta kertaakaan

Indefiniitti toisto (toistojen lukumäärää ei tiedetä)

```
while ehto:  
    toiminto
```

- *Toiminnon* suorituksen jälkeen *ehto* testataan jälleen
- Jos *ehto* edelleen voimassa, suoritetaan *toiminnot* uudestaan
- Toisto loppuu, kun *ehto* on epätosi



While-toisto - esimerkki

`i = 5`

`summa = 0`

while `i < 10:`

`summa = summa + i`

`i = i + 1`



While-toisto - esimerkki

$i = 5$

$\text{summa} = 0$

while $i < 10$:

$\text{summa} = \text{summa} + i$

$i = i + 1$

- $i = 5 < 10$ – suoritetaan silmukka; summa 5; i 6



While-toisto - esimerkki

$i = 5$

$\text{summa} = 0$

while $i < 10$:

$\text{summa} = \text{summa} + i$

$i = i + 1$

- $i = 5 < 10$ – suoritetaan silmukka; summa 5; i 6
- $i = 6 < 10$ – suoritetaan silmukka; summa 11; i 7



While-toisto - esimerkki

$i = 5$

$\text{summa} = 0$

while $i < 10$:

$\text{summa} = \text{summa} + i$

$i = i + 1$

- $i = 5 < 10$ – suoritetaan silmukka; summa 5; i 6
- $i = 6 < 10$ – suoritetaan silmukka; summa 11; i 7
- $i = 7 < 10$ – suoritetaan silmukka; summa 18; i 8



While-toisto - esimerkki

$i = 5$

$\text{summa} = 0$

while $i < 10$:

$\text{summa} = \text{summa} + i$

$i = i + 1$

- $i = 5 < 10$ – suoritetaan silmukka; summa 5; i 6
- $i = 6 < 10$ – suoritetaan silmukka; summa 11; i 7
- $i = 7 < 10$ – suoritetaan silmukka; summa 18; i 8
- $i = 8 < 10$ – suoritetaan silmukka; summa 26; i 9



While-toisto - esimerkki

$i = 5$

$\text{summa} = 0$

while $i < 10$:

$\text{summa} = \text{summa} + i$

$i = i + 1$

- $i = 5 < 10$ – suoritetaan silmukka; summa 5; i 6
- $i = 6 < 10$ – suoritetaan silmukka; summa 11; i 7
- $i = 7 < 10$ – suoritetaan silmukka; summa 18; i 8
- $i = 8 < 10$ – suoritetaan silmukka; summa 26; i 9
- $i = 9 < 10$ – suoritetaan silmukka; summa 34; i 10



While-toisto - esimerkki

$i = 5$

$\text{summa} = 0$

while $i < 10$:

$\text{summa} = \text{summa} + i$

$i = i + 1$

- $i = 5 < 10$ – suoritetaan silmukka; summa 5; i 6
- $i = 6 < 10$ – suoritetaan silmukka; summa 11; i 7
- $i = 7 < 10$ – suoritetaan silmukka; summa 18; i 8
- $i = 8 < 10$ – suoritetaan silmukka; summa 26; i 9
- $i = 9 < 10$ – suoritetaan silmukka; summa 34; i 10
- $i = 10$ ei ole < 10 , lopetetaan



While-toisto – ehdon muotoilu

- Mikä tässä on vikana?

i = 5

summa = 0

while i < 10:

 summa = summa + i



While-toisto– ehdon muotoilu

- Toiston rungossa on tärkeä tehdä operaatioita jotka lopulta muuttavat ehdon epätodeksi
- Ehdon muotoon kiinnitettävä huomiota
- Yleinen virhe: ehto joka pysyy totena
- ViLLEssä tällaiset virheet ilmenevät aikakatkaisuvirheenä (Operation timed out)

i = 5

summa = 0

while i < 10:

summa = summa + i



While-toisto – ehdon muotoilu

- Esimerkki:

$i = 1$

$\text{summa} = 0$

while $i \neq 10$:

$\text{summa} = \text{summa} + i$

$i = i + 2$

- Nyt i käy läpi parittomia arvoja 1, 3, 5, 7, 9, 11
- Ehto ei siis ikinä tule epätodeksi
- Korjaus: ehto muotoon $i < 10$



Indefiniitti vs definiitti toisto

- Definiitti toisto voidaan aina korvata indefiniitillä toistolla
 - Päinvastoin ei päde!
- **Indefiniitti toisto riittäisi kaikkien toistojen esittämiseen**
- Selvyyden vuoksi valitaan ongelmaan parhaiten sopiva toistomuoto
 - Jos tarvitaan silmukkamuuttujaa, definiitti yleensä soveltuu paremmin



Indefiniitti vs definiitti toisto

- Definiitti toiston korvaus indefiniitillä toistolla esimerkki:

Definiitti toisto	Indefiniitti toisto
<pre>summa = 0 for i in range(1,6): summa = summa + i</pre>	<pre>summa = 0 i = 1 while i < 6: summa = summa + i i = i + 1</pre>



Yhteenveto kontrollirakenteista

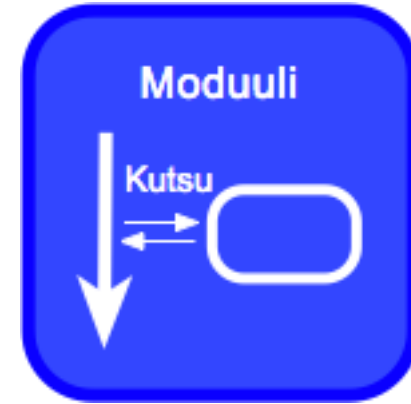
Ohjausrakenne	Selitys
	• Iteratiivisten ohjelmien perusrakenne • Koodirivit suoritetaan ylhäältä alas algoritmiin kirjoitetussa järjestyksessä • Tästä järjestyksestä voidaan poiketa muilla rakenteilla
	• Kontrollirakenne jolla valitaan kahden toiminnon välillä • Ehdon ollessa tosi suoritetaan if-lohkon koodi • Ehdon ollessa epätosi suoritetaan else-lohkon koodi
	• Mahdollistaa toiminnon suorittamisen monta kertaa • Eri tyyppejä sen mukaan tiedetäänkö toistojen määrä • Rakenteet for, while

Modulaarisuus



Moduuli

- **Moduuli** on itsenäinen algoritmikomponentti
 - Moduuli eli metodi (method)
eli rutiini (routine)
eli alirutiini (subroutine)
eli aliohjelma (subprogram)
- Moduuli ratkaisee yhden osaongelman
- Moduuli voidaan suunnitella käyttöyhteydestä riippumattomasti
- Moduulia voidaan käyttää missä tahansa algoritmissa jossa esiintyy ko. Osaongelma





Moduulin kutsu ja modulaarisuus

- Moduulia käyttävän algoritmin sanotaan ***kutsuvan*** moduulia
- Useista moduuleista koostuvaa algoritmia sanotaan ***modulaariseksi***
- Modulaarisessa algoritmissa jokainen ei-triviaali moduuli koostuu pienemmistä moduuleista

Moduulin määrittely

Yksinkertaisen moduulin määrittely on muotoa:

```
def moduulinNimi():  
    moduulin runko
```

- *moduulinNimi* on moduulille annettu nimi
 - Nimen tulee kuvata moduulin tehtävää
 - Käytetään moduulia kutsuttaessa
- Moduulin runko on tehtävän toteuttavat toiminnot

Moduulin kutsu

```
def moduulinNimi():  
    moduulin runko
```

- Moduulia kutsutaan sen nimeä käyttäen
- Esimerkiksi yo. moduulin kutsu:
 moduulinNimi()
- Monimutkaisempia määrittelyjä ja kutsuja
 ensi viikolla

Moduulin käyttö - esimerkki

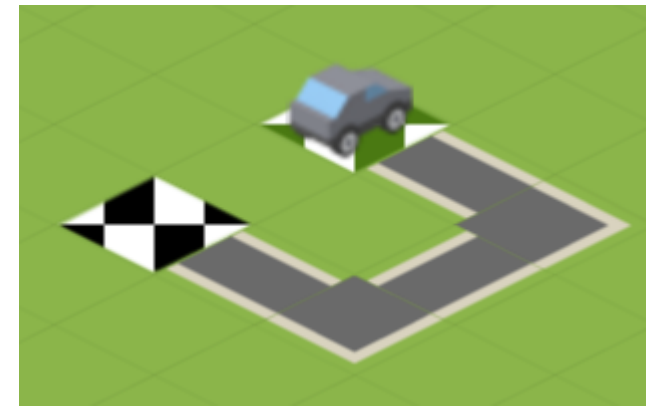
- Tehtävä: kuljeta auto maaliin teitä pitkin



- Käytössä moduulit `turnRight()`, `turnLeft()`, `moveForward()`, `moveBackward()`



Moduulin käyttö - esimerkki



- Käytössä moduulit `turnRight()`, `turnLeft()`, `moveForward()`, `moveBackward()`

- Naiivi tapa:

`turnRight()`

`moveForward()`

`moveForward()`

`turnRight()`

`moveForward()`

`moveForward()`

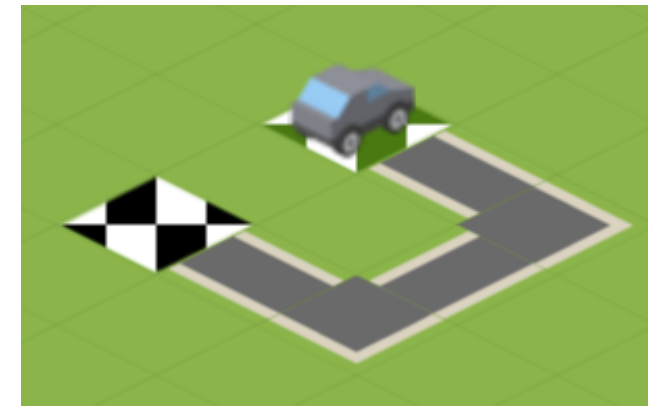
`turnRight()`

`moveForward()`

`moveForward()`



Moduulin käyttö - esimerkki



- Käytössä moduulit `turnRight()`, `turnLeft()`, `moveForward()`, `moveBackward()`
- Naiivi tapa:

```
turnRight()  
moveForward()  
moveForward()
```

```
turnRight()  
moveForward()  
moveForward()
```

```
turnRight()  
moveForward()  
moveForward()
```

Moduuliin:
`def oikeaJaEteen():`
 `turnRight()`
 `moveForward()`
 `moveForward()`

```
oikeaJaEteen()  
oikeaJaEteen()  
oikeaJaEteen()
```

Moduulin käyttö - esimerkki

- Miten algoritmi voidaan korjata kuvan uuteen tilanteeseen?

```
# Ei toimi!  
def oikeaJaEteen():  
    turnRight()  
    moveForward()  
    moveForward()  
  
oikeaJaEteen()  
oikeaJaEteen()  
oikeaJaEteen()
```





Moduulin käyttö - esimerkki

- V: korjataan vain moduulia

```
# Toimii!  
def oikeaJaEteen():  
    turnRight()  
    moveForward()  
    moveForward()  
    moveForward()
```

```
oikeaJaEteen()  
oikeaJaEteen()  
oikeaJaEteen()
```





Moduulin käyttö - esimerkki

- V: Vähennetään vielä turhaa "copy-pastea" käyttämällä silmukoita

```
# Toimii!  
def oikeaJaEteen():  
    turnRight()  
    for i in range(3):  
        moveForward()  
  
for i in range(3):  
    oikeaJaEteen()
```

