

Johdanto R-ohjelmoointiin

author: Yliopistonlehtori, FT Juho Kopra #date: 8.9.2021 autosize: true



Kurssin oppimistavoitteet

- kurssilla opettelemme ohjelmoimaan R-kieellä sillä tasolla että kykenemme yksinkertaisiin tilastollisiin analyyseihin
- Opintojakson suoritettuaan opiskelija osaa:
 - selittää R-ohjelmoinnin oleelliset perusperiaatteet
 - toteuttaa tavanomaisen tutkimusaineiston analyysin R-ohjelmoimalla ml. aineiston muokkaamisen, aineiston kuvailun kuvajien ja tunnuslukujen avulla
 - hankkia tietoa käytäen internettiä ja dokumentaatiotiosivuja itsenäisen opiskelun tukena

Mitä R:llä voi tehdä?

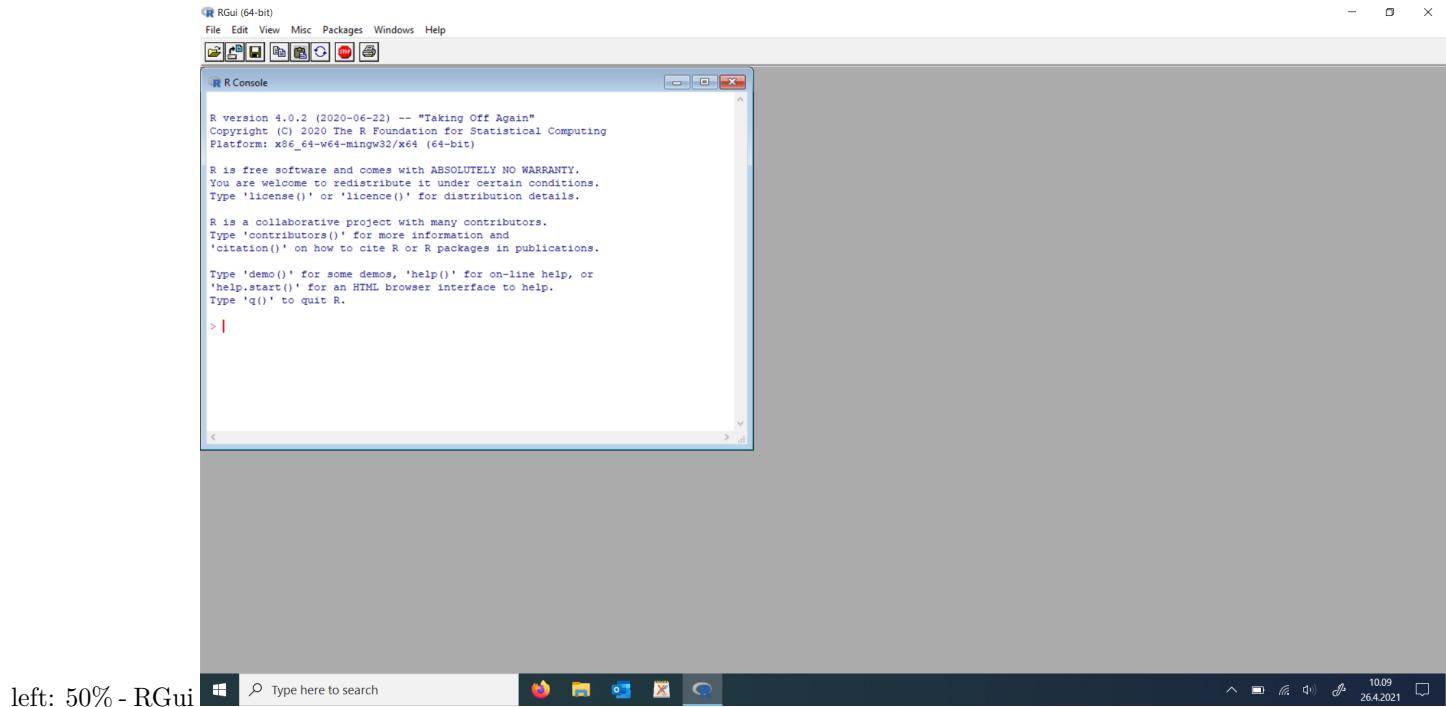
- tilastografiikkaa, kuvajaa
- laskea mitä tahansa tunnuslukuja, kuten keskiarvon ja varianssin
- muokata aineistoa
- tallentaa tulokset
- toteuttaa monimutkaisiakin tilastollisia analyysejä
- optimointia, simulointia

Mitä muuta? - R-paketteja - raportteja, kirjoja, luentokalvoja - verkkosivuja

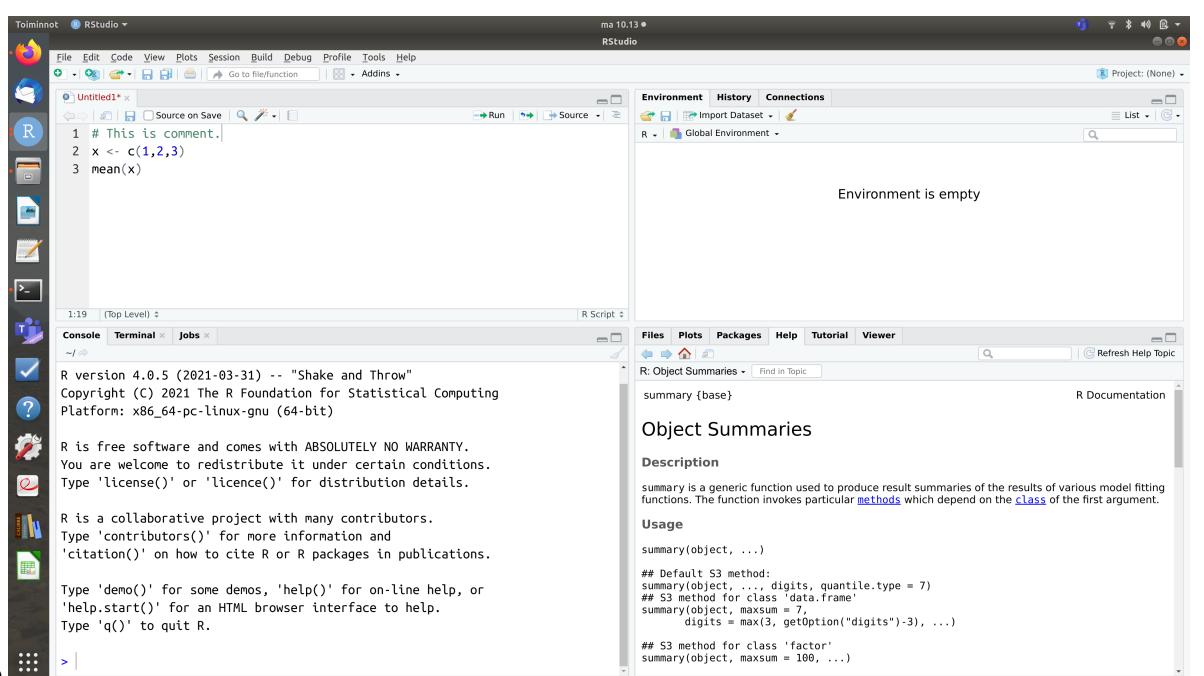
Miksi R:ää kannattaa opiskella?

- R on tehokas työkalu määrälliseen tutkimukseen
- tutkimusta tarvitaan monessa tilanteessa
 - oppinäyte, työlämä, jatko-opinnot
- tutkimusta ja data-analyysiä toteutetaan nykyään aina tietokoneen avulla
 - tietokoneella sitä tehdään jollain ohjelmalla
- R vaatii opettelua hieman muita ohjelmia enemmän, mutta taipuu todella moniin tarpeisiin!
- R-osaaminen on tarpeellista tilastotieteen ja muiden oppiaineiden kursseilla

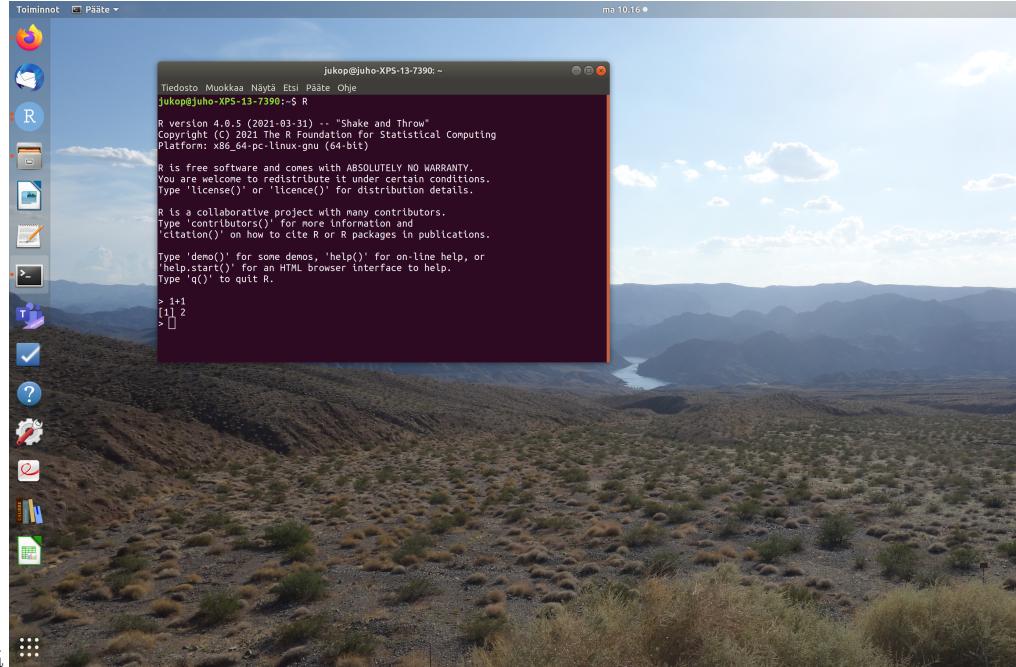
Erilaisia R-ohjelmointiympäristöjä



left: 50% - RGui

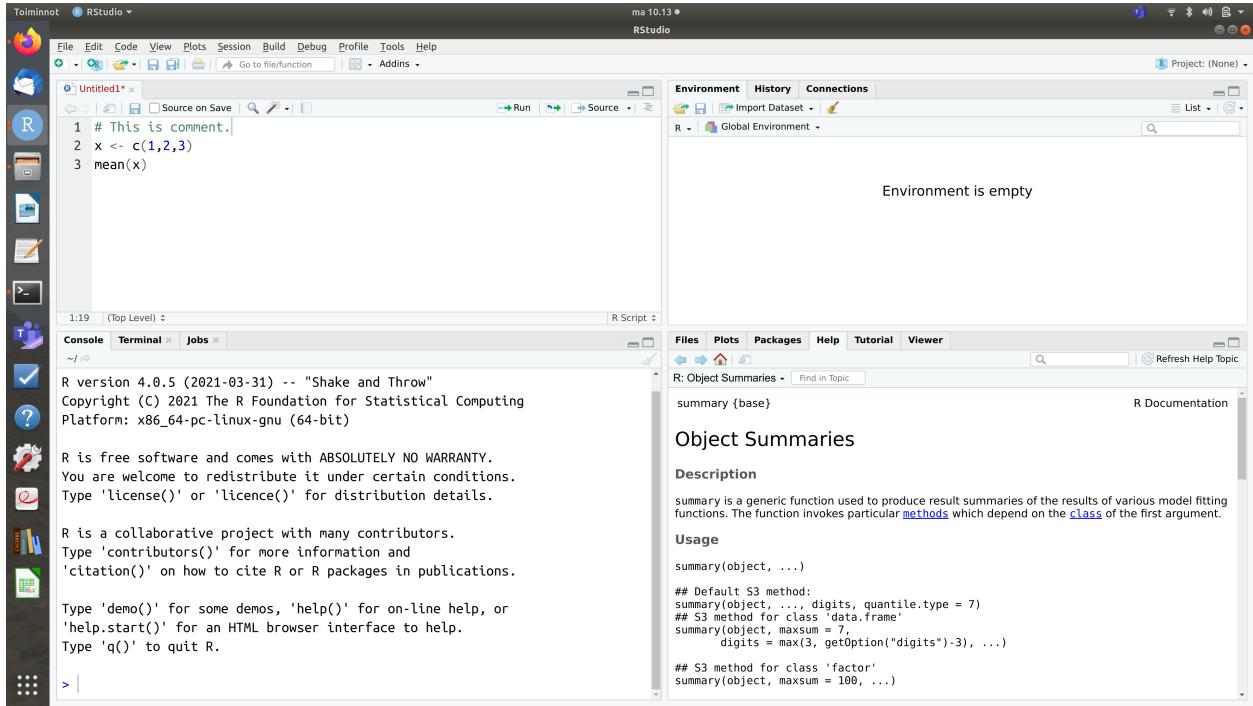


- RStudio



- R komentorivillä käynnistetynä

RStudion esittely



Laskeminen R:llä

left: 50% - R toimii laskukoneena

1+6

[1] 7

2-3

[1] -1

2*3

[1] 6

5/7

[1] 0.7142857

4^2

[1] 16

Ohjelointi on toimintaohjeiden antamista tietokoneelle:

- ohjelma muodostuu toimintaohjeista (kuten resepti on toimintaohje)
- toimintaohjeiden järjestyksellä on merkitystä!
 - esim. jos jogurttiin lisää kurkun ennen kuin on raastanut sen, niin lopputulos ei ole haluttu!

Kreikkalainen jogurtidippi (Tzatsiki)

1. Raasta puolikas kurkku
2. Purista kurkkuraasteesta ylimääräinen neste pois
3. Kuori ja silppua 1-2 valkosipulin kynttä
4. Sekoita 0.5 litraan kreikkalaista jogurttia kurkku, valkosipuli ja mausteet
5. Anna maustua jäälkaapissa 20 minuuttia
6. Tarjoile

Perusasioita ohjelmoinnista:

- R-koodi on ohjelma, joka on toteutettu R-kielellä
- koodia suoritetaan komento kerrallaan ylhäältä alkaen, alas päin edeten
- R on kieli, eli täytyy osata sen sanat ja kielioppi

Perustökalut

left: 50% - koodia kirjoitetaan koodieditorilla (RStudio) - koodia voidaan ajaa suoraan R-komentokehotteessa (console) + komentokehoteessa oleva väkänen > tarkoittaa, että komentokehote on valmis vastaanottamaan koodia (komentoja) + komentokehote tulkitsee koodin (laskutoimituksen) ja palauttaa lopputuloksen

-
- tai koodi voidaan tallentaa skriptitedostoon (.R-päätteinen)
 - tähän palataan myöhemmin
 - R on tulkattava kieli, mikä tarkoittaa että ohjelman ei tarvitse olla valmis, jotta sitä voidaan käyttää, vaan sitä voidaan suorittaa rivi kerrallaan. Käyttäjä siis keskustelee tietokoneen kanssa R-kielen avulla.

Aritmetiikkaa R:llä

1+6

[1] 7

```
[1] 0.7142857
```

Virheitä koodissa?

left: 50% - tietokone ymmärtää vain täsmälleen oikein annettuja komentoja, joten yksikin väärä merkki koodissa aiheuttaa ongelmia! + tietokone ei osaa tulkita monimerkityksiä



- ongelma voi ilmetä joko siten että ohjelma ei tee mitä sen pitäisi tai ohjelman suorittaminen voi päättää virheilmoitukseen

3 + "four"

```
Error in 3 + "four" : non-numeric argument to binary operator
```

Miten koodia kehitetään?

1. Jos koodi on pitkä, niin suunnittele koodin osaset
2. Aloita kirjoittamalla pieni pätikä koodia.
3. Kokeile, toimiiko kirjoittamasi koodi.
4. Jos ei toimi halutulla tavalla, tai tulee virheilmoitus, selvitä vian syy.
 - Millä rivillä virhe on koodissa?
 - Mikä funktio ja muuttuja aiheuttaa virheen?
5. Korjaa vika. (Tähän saattaa mennä paljonkin aikaa.)
6. Siirry seuraavaan koodin pätikään ja jatka iterointia kohdasta 2.

Mistä ohjelmakoodi muodostuu?

- komennoista, jotka muodostuvat:
- muuttujista (muuttujien nimet ja tyypit)
- alkeistietotypeistä ja tietotypeistä
- funktioista
- operaattoreista (tekevät jonkin toiminnon, kuten sijoitusoperaattori tai vertailuoperaattori)
- suluista ja rivinvaihdoista
- kommenteista

```
x <- c(1,2,3)
mean(x) # lasketaan keskiarvo
```

```
[1] 2
```

Muuttujat

- muuttujan nimeäminen

```
cats <- 3  
dogs <- 4  
cats + dogs
```

```
[1] 7
```

```
paste("My friend has", cats + dogs, "animals.")
```

```
[1] "My friend has 7 animals."
```

- komento voi mennä myös usealle riville
 - tällöin komentokehote ilmoittaa keskeneräisyydestä +-merkillä

```
paste("My friend has", cats + dogs, "animals."  
)
```

```
[1] "My friend has 7 animals."
```

Muuttujat

left: 50% - muuttujien tyypit

```
class(cats)
```

```
[1] "numeric"
```

```
# boolean variable  
var_bool <- TRUE  
var_bool
```

```
[1] TRUE
```

```
class(var_bool)
```

```
[1] "logical"
```

```
var_char <- "this is text"  
class(var_char)
```

```
[1] "character"
```

Vektorit

- tämä on sopiva kohta opettaa vektorit ja aloittaa Rcourse-paketin osio 1

Muuttujan muuttaminen faktoriaksi

- muuttujien mitta-asteikko voi olla myös kategorinen, jota sanotaan joskus faktoriaksi (factor)

```

# numeric variable
var_num <- c(42.1,11.2,31)
class(var_num)

[1] "numeric"

# factor variable (categorical)
var_fact <- factor(c(0,1,1,0,0))
class(var_fact)

[1] "factor"

```

- muuttuja kannattaa muuttaa faktoriksi vasta kun on pakko

Funktiot

- olemme jo käyttäneet joitain funktioita (class, factor, mean)
- käytetään summary-funktioita eri typpisiin muuttuihin

```

summary(var_num)

Min. 1st Qu. Median Mean 3rd Qu. Max.
11.20 21.10 31.00 28.10 36.55 42.10

```

```

summary(var_fact)

0 1
3 2

```

- muuttujan tyyppi voi siis vaikuttaa siihen, mitä funktio tekee!

Lisää faktoreista

```

animals <- factor(c(0,1,1,0,0),labels=c("cat","dog"))
class(animals)

[1] "factor"

summary(animals)

cat dog
3 2

animals <- factor(c("cat","dog","dog","cat","cat"))
class(animals)

[1] "factor"

summary(animals)

cat dog
3 2

```

Funktiot

left: 50% - osa ohjelmointikielen toiminnallisuksista on funktioiden muodossa - funktioilla on lähtökohtaisesti sama idea kuin funktioilla matematiikassa + R:ssä on valmiina lukuisia eri funktioita

```
exp(3) # eksponenttifunktio exp()
```

```
[1] 20.08554
```

- yleensä funktio ottaa joitain tietoja **syötteenä** (input), joita sanotaan myös argumenteiksi

-
- funktio myös yleensä palauttaa joitain arvoja **palautusarvona** (output)
 - funktioita käytettäessä on tarpeen ymmärtää mitä kyseinen funktio tekee
 - ei kuitenkaan tarvitse tietää miten ko. funktio on toteutettu
 - R:ssä funktion argumentteja ei ole pakko aina mainita, mikä usein vähentää kirjoittamista
 - epäselvissä tilanteissa argumentit kannattaa kirjoittaa, se selkeyttää koodia

Funktiot ja tiedonhaku

- usein on tarpeen selvitää funktion toiminnallisuus joko ohjesivun avulla tai internetistä
 - ohjesivun saat R:ssä auki kirjoittamalla kysymysmerkin haluamasi funktion eteen

```
?mean
```

- monet perusfunktioista on vuosikymmeniä vanhoja, jonka takia dokumentaatio ei ole aina kovin helppolukuista (pitää silti paikkansa)
 - Dokumentaation lopussa oleva Examples kannattaa suorittaa ja pohtia ymmärtääkö miten funktio toimii
- tavallisimpiin tarpeisiin on myös R Cheat Sheets, jotka ovat hyödyllisiä, esim. base R cheat sheet lataa tästä

Funktiot ja tiedonhaku

- jos et tiedä funktion nimeä, niin kannattaa etsiä englanninkielisellä Google-haulla sopivaa funktiota
 - esim. jos et tiedä miten varianssi lasketaan R:ssä, niin mene Googleen ja laita hakuun: how to calculate variance in R
 - mikäli etsit apua jonkin R-paketin käyttöön kannattaa lisätä hakusanaksi vignette

Funktiot

- jotkut funktiot tarvitsevat useita argumentteja

```
seq(1,3,0.5)
```

```
[1] 1.0 1.5 2.0 2.5 3.0
```

```
seq(from=1,to=3,by=0.5) # sama, mutta argumentit nimetty
```

```
[1] 1.0 1.5 2.0 2.5 3.0
```

```
seq(from=1,to=3,length.out=6) # by-argumenttia ei käytetä, joten on pakko nimetä length.out-argumentti
```

```
[1] 1.0 1.4 1.8 2.2 2.6 3.0
```

Funktiot

- funktion palauttamat tiedot voi tallentaa suoraan muuttujaan

```
k <- seq(1,3,0.5)
```

```
k
```

```
[1] 1.0 1.5 2.0 2.5 3.0
```

Muuttujat argumentteina

left: 50% - muistissa olevia muuttuja voi käyttää funktion argumentteina (operaattorina yhtäsuuruusmerkki “=”)

```
numbers <- c(1,2,3,4)  
mean(x=numbers)
```

```
[1] 2.5
```

- tai muuttujan voi luoda suoraan sijoittaan se funktion argumentiksi

```
mean(x=c(1,2,3,4))
```

```
[1] 2.5
```

-
- edelliset funktiokutsut ovat täsmälleen sama kuin

```
mean(numbers)
```

```
[1] 2.5
```

- ja (eli argumenttia ei ole pakko nimetää)

```
mean(c(1,2,3,4))
```

```
[1] 2.5
```

Operaattorit

- sijoitusoperaattorit <-, «-, =

```
x <- 5.2  
y <<- 7.44 #harvinainen, mutta Rcourse -paketti pyytää tätä joskus  
x2 = 5.3 # toimii myös, mutta ei suositella
```

- sijoitusoperaattoria käytettäessä arvoa ei palauteta

-
- jos haluaa katsoa mitä tuli sijoitetuksi, on muuttuja itse tulostettava

```
x
```

```
[1] 5.2
```

```
y
```

```
[1] 7.44
```

```
x2
```

```
[1] 5.3
```

Operaattorit

left: 50% - vertailuoperaattorit <=,<,>=, !=, >, >=

```
3 <= 4 # onko 3 pienempi tai yhtä suuri kuin 4 ?
```

```
[1] TRUE
```

```
5 < 4 # onko 5 pienempi kuin 4 ?
```

```
[1] FALSE
```

```
3.01 == 3 # onko 3.01 yhtä suuri kuin 3 ?
```

```
[1] FALSE
```

```
3 != 4 # onko 3 eri suuri kuin 4?
```

```
[1] TRUE
```

```
5 > 4 # onko 5 pienempi kuin 4 ?
```

```
[1] TRUE
```

```
3 >= 4 # onko 3 suurempi tai yhtä suuri kuin 4 ?
```

```
[1] FALSE
```

- Huom! Sijoitusoperaattoria “=” ei voi käyttää vertailuun. Käytä sen sijaan kahta yhtäsuuruusmerkkiä “==”

Kommentit

- Kommentti on koodissa oleva seliteteksti, joka ei vaikuta koodin toimintaan

```
# Kommentin tarkoitus on kertoa, mitä koodi tekee.
```

- Olemme nähneet useita kommentteja, mm. edellisellä kalvolla
 - Kommentti alkaa #-merkillä ja seliteteksti tulee sen oikealle puolelle
 - ennen #-merkkiä voi olla suoritettavaa koodia

```
x <- 1:10 # Create a vector with values from one to ten.
```

- Hyvä kommentti on yksiselitteinen ja selittää koodin toimintaa tarpeellisella tarkkuudella
 - jos arvelet, että joku ei-suomea puhuva henkilö voisi tarkastella koodiasi, niin kommentit ja muuttujien nimet kannattaa olla englanniksi

Kommentit

- R:ssä on vain yksi kommenttimerkki, joka on risuaita (#)
 - joissain muissa ohjelmointikielissä on useita kommenttimerkkejä
- Yksi hyvä tapa on laittaa skriptin alkuun kommentti, mitä koodi yleisellä tasolla tekee ja kuka sen on tehnyt

```
# R-programming: examples to students, Juho Kopra, University of Eastern Finland
```

- Lisäksi kannattaa kommentoida ainakin hankalasti ymmärrettävät kohdat koodista.
- RStudiossa voi kommentimerkeillä lohkoa koodia osasiin

```
#-----  
# new section of the code starts
```

Alkeistietotyypit ja tietotyypit

- alkeistietotyypit sisältävät ohjelmointikielen pienimmät osaset, kuten vaikkapa reaaliluvut tai totuusarvot
- tietotyypit ovat alkeistietotyyppiestä johdettuja objekteja, joiden avulla ohjelman voi rakentaa
- tietotyypeihin voidaan säälöä alkeistietotyyppien sisältämää dataa
- tietotyypit säälövät tietoa ja eri tietotyypejä voidaan käyttää eri tavoin

Alkeistietotyypit

- R:ssä alkeistietotyypejä ovat **character**, **numeric**, **logical**, **integer**, **complex**
 - Tyypejä integer ja complex tarvitaan vain harvoin, joten niitä ei käsitellä tällä kurssilla tarkemmin.
- Alkeistietotyypit voivat toimia muiden tietotyyppien elementteinä (esim. vektoreiden alkioina).
- ottamalla käyttöön R-paketteja, voidaan saada lisää alkeistietotyypejä (ja tietotyypejä) käyttöön

Hieman lisää alkeistietotyyppiestä

left: 50% - **character** eli merkkijono

```
var_char <- "this is text"  
class(var_char)
```

```
[1] "character"
```

- **numeric**

```
var_num <- c(42.1,11.2,31)  
class(var_num)
```

```
[1] "numeric"
```

- **logical**

```
var_num <- c(TRUE,FALSE)  
class(var_num)
```

```
[1] "logical"
```

-
- **integer**: ei yleensä tarpeellinen

```
var_num <- c(0L,1L,5L)  
class(var_num)
```

```
[1] "integer"
```

```
5L %% 2L
```

```
[1] 2
```

- **complex**: kompleksilukujen laskemiseen. Emme käytä.

Alkeistietotyyppien erikoisarvot

- NA eli **puuttuva arvo**
 - käytetään aineistossa, kun lukuarvoa ei tunneta

```

x <- c(1, 2, NA)
is.na(x)
[1] FALSE FALSE TRUE
mean(x)
[1] NA
mean(x, na.rm=TRUE)
[1] 1.5
– toimii kaikille alkeistietotyypeille
* Joskus harvoin tarvitsee määritellä NA-arvon tyyppi, sen voi tehdä näin: NA_real_;
NA_character_; NA_complex_

```

- Inf ja -Inf eli **ääretön ja miinus ääretön** esim. 1/0
 - vain numeeriselle tietotyypille

```

x <- c(1, 2, Inf)
is.finite(x)

[1] TRUE TRUE FALSE

```

- NaN eli “**Not a Number**” 0/0
 - nolla per nolla ei ole sallittua matematiikassa
 - R käsittelee tämän ongelman siten, että 0/0 tuottaa NaN-arvon
- voidaan käsitellä funktoilla is.na(), is.nan(), is.finite()
- NULL on **määrittelemätön arvo**. Harvoin tarpeellinen.

```

x <- NULL
is.null(x)

[1] TRUE

```

Tietotyypit

- R:n perustietotyypit ovat **vector**, **data.frame**, **factor**, **list**, **matrix** ja **array**
 - näistä keskeisimmät ovat vector, data.frame ja factor
 - * näitä tarvitaan lähes jokaisessa data-analyysissä
 - myös list ja matrix tietotyypeillä on oma käytönsä
 - array on harvinainen tietotyppi

Vektori

- **vector** eli vektori, voidaan ajatella lukujonoksi
 - tallennetut luvut pysyvät oletusarvoisesti siinä järjestyksessä kuin ne on syötetty
 - elementtien on oltava samaa alkeistietotyppiä
 - elementeillä voi olla jokin nimi

```

values <- c(1.1, 3.1, 2.5)
text <- c("cat", "dog", "animal")
named_vector <- c(first=1, second=2)
named_vector

first second

```

1 2

Datakehikko

- **data.frame** datakehikko pitää sisällään aineiston, jossa voi olla eri tyyppisiä muuttuja
 - muuttujien on oltava vektoreita ja niissä on oltava yhtä monta elementtiä
 - muuttujilla on aina jokin nimi

```
dat <- data.frame(values = c(1.1,3.1,2.5), text = c("cat","dog","animal"))
dat
```

```
values   text
1      1.1    cat
2      3.1    dog
3      2.5 animal
```

Hieman lisää tietotyypeistä

- **factor** faktori-tietotyyppiin tutustuimme jo aiemmin
 - faktori muodostetaan numeerisen tai tekstimuotoisen vektorin pohjalta
 - teknisesti faktori on tietokoneen muistissa numeerinen vektori, jonka lukuarvoille on selitetekstit (label)

Lista

left: 50% - **list** lista on luettelo, joka sallii eri tyyppisten objektien säilömisen + listan osioilla voi olla nimi, mutta ei ole pakko

```
1 <- list("a"=c(1,2),"b"=dat)
1
```

```
$a
[1] 1 2
```

```
$b
values   text
1      1.1    cat
2      3.1    dog
3      2.5 animal
```

-
- listan alkioihin voi viitata kaksoishakasuluilla

```
1[[2]]
```

- yksinkertainen hakasulku palauttaa listan, jossa valitut listan objektit

```
1[2]
```

```
$b
values   text
1      1.1    cat
2      3.1    dog
3      2.5 animal
```

Matriisi

- **matrix** matriisi on taulukko, joka sisältää vain yhtä alkeistietotyyppiä
 - numeerisilla matriiseilla ja vektoreilla voidaan tehdä lineaarialgebraa, kuten matriisitulo (sisätulo/pistetulo), transpoosi, matriisin kääntäminen, jälki ja ominaisarvot.
 - matriisi muodostetaan vektorista, siten että oletusarvoisesti matriisi täytetään sarake kerrallaan alkaen vasemman puolisesta sarakkeesta
 - matriisiin voi täyttää riveittäin argumentilla **byrow=TRUE**

```
vec <- c(1,2,3,4,5,6)
matrix(vec,nrow=2,ncol=3)
```

```
[,1] [,2] [,3]
[1,]    1     3     5
[2,]    2     4     6
```

Array

- **array** on moniulotteinen taulukko, joka on matriisin yleistys.
 - tarvitaan tilanteissa, joissa aineisto halutaan esittää kolme- tai useampiulotteisena taulukkona

```
ar <- array(data=1:27,dim=c(3,3,3))
ar # uses three indices
```

```
, , 1
[,1] [,2] [,3]
[1,]    1     4     7
[2,]    2     5     8
[3,]    3     6     9
```

```
, , 2
[,1] [,2] [,3]
[1,]   10    13    16
[2,]   11    14    17
[3,]   12    15    18
```

```
, , 3
[,1] [,2] [,3]
[1,]   19    22    25
[2,]   20    23    26
[3,]   21    24    27
```

```
ar[, , 1]
```

```
[,1] [,2] [,3]
[1,]    1     4     7
[2,]    2     5     8
[3,]    3     6     9
```