Statistical Programming Languages – Day 5

Uwe Ziegenhagen

Institut für Statistik and Ökonometrie
Humboldt-Universität zu Berlin
http://www.uweziegenhagen.de

Agenda for Today

Programming ***R***

- Functions
- Loops
- Conditions
- Packages

Functions

```
myfun <- function(x){
return(x*x)
}
myfun(2)
```

```
myfun <- function(x){
x*x
}
myfun(2)
```

If no return() is given, the object last created is returned.

Functions

```
myfun <- function(x, a){
r <- a*sin(x)
return(r)
}
myfun(pi/2,2)
```

```
myfun1 <- function(x, a){ a*sin(x) } #short version
myfun1(pi/2,2)
```

Functions

```
myfun2 <- function(x, a=1){ # opt. parameter with default=1
a*sin(x)
}
myfun2(pi/2,2)
myfun2(pi/2)
```

optional = the function can also be run without specification

Functions

```
myfun3 <- function(x, a=NULL){ # opt. parameter without default
if (!is.null(a)){
    a*sin(x)
}else{
    cos(x)
    }
}
myfun3(pi/2,2)
myfun3(pi/2)
```

Functions

```
myfun4 <- function(x, a=1){
 r1 <- a*sin(x)
 r2 <- a*cos(x)
return(r1,r2)
# return two results
# don't use this, use a list
}
myfun4(pi/2)
```

Functions

```
myfun5 <- function(x, a=1){
 r1 <- a*sin(x)
 r2 <- a*cos(x)
return(list(r1,r2)) # one list as result
}
myfun5(pi/2)
```

Functions

many different calls on this page

```r
myfun6 <- function(x, a=1, b=2){
r1 <- a*sin(x)
r2 <- b*cos(x)
return(list(r1,r2))
}
myfun6(pi/2)    # a=1, b=2 (defaults)
myfun6(pi/2,1,2)# a=1, b=2 (given explicitly)
myfun6(pi/2,2)  # a=2, b=2 (only a given)
myfun6(pi/2,a=2)# a=2, b=2 (only a given)
myfun6(pi/2,b=3)# a=1, b=2 (only b given)
```

Using Sets

```
a <- 1:3
b <- 2:6
a
b
a %in% b
b %in% a
a <- c("A","B"); b <- LETTERS[2:6]
a
b
a %in% b
b %in% a
```

`LETTERS` is a special predefined variable, containing 'A' to 'Z', (`letters`='a' to 'z')

Loops and Conditions - IF

```
# simple if
x<-1
if (x==2){ print("x=2") }
# if-else
x<-1
if (x==2){
  print("x=2")
} else {
  print("x!=2")}
```

Loops and Conditions - FOR

```
for (i in 1:4){ print(i) }
for (i in letters[1:4]){ print(i) }

a<-numeric(400) # generate empty a of length 400
for (i in 1:400){ a[i]=i } # fill a with 1:400
# takes much longer than a<-1:400
```

Loops and Conditions - WHILE

```
i<-0
while(i<4){
    i <- i+1
    print(i)
}
```

Do NOT forget the increase of the counter variable *i*!

Loops and Conditions - REPEAT

```
i <- 0;
repeat{
 i <- i+1;
 print(i);
 if (i==4) break
 }
```

If no break is given, loops runs forever!

Loops and Conditions - IFELSE

`ifelse` allows shorter if-else syntax

```
ifelse(boolean check, if-case, else-case)
```

```
x <- c(6:-4)
sqrt(x) # gives warning
sqrt(ifelse(x >= 0, x, NA))# no warning
```

Loops and Conditions - SWITCH

switch avoids the use of nested if-else constructs, however it looks tricky. If `type` is numeric, switch uses the *i*-th item.

```r
rootsquare <- function(x, type) {
  switch(type,
        square = x*x,
        root = sqrt(x))
}
rootsquare(10,1)
rootsquare(10,2)
```

Loops and Conditions - SWITCH

If type is a string, R matches the string.

```r
rootsquare <- function(x, type) {
  switch(type,
        square = x*x,
        root = sqrt(x))
}
rootsquare(10,"square") # ok
rootsquare(10,"root")   # ok
rootsquare(10,"ROOT")   # returns NULL
```

Loops and Conditions - SWITCH

another example for switch

```
centre <- function(x, type) {
  switch(type,
         mean = mean(x),
         median = median(x),
         trimmed = mean(x, trim = .1))
}
x <- rcauchy(10)
centre(x, "mean")
centre(x, "median")
centre(x, "trimmed")
```

Loops and Conditions

system.time() measures the time necessary to run a command.

```
> x <- c(1:50000)
> system.time(for (i in 1:50000){x[i]<-rnorm(1)})
       User      System verstrichen
       0.67        0.01        0.69
> system.time(x<-rnorm(50000))
       User      System verstrichen
       0.01        0.00        0.01
```

⇒ use matrix functions

The `apply()` commands

these commands allow functions to be run on matrices.

`apply()`	Function used on matrix
`tapply()`	table grouped by factors
`lapply()`	on lists and vectors; returns a list
`sapply()`	like `lapply()`, returns vector/matrix
`mapply()`	multivariate `sapply()`

apply()

```
apply(data, margin, function)
```

```
> matrix(1:10,nrow=2)->a
> apply(a,1,mean)
[1] 5 6
> apply(a,2,mean)
[1] 1.5 3.5 5.5 7.5 9.5
```

lapply()

```
> matrix(2:11,nrow=2)->a
> matrix(1:10,nrow=2)->b
> c= list(a,b)
> lapply(c,mean)
[[1]]
[1] 6.5

[[2]]
[1] 5.5
```

sapply()

```
> matrix(2:11,nrow=2)->a
> matrix(1:10,nrow=2)->b
> c= list(a,b)
> sapply(c,mean)
[1] 6.5 5.5
```

mapply()

```
> mapply(rep, pi, 3:1)
[[1]]
[1] 3.141593 3.141593 3.141593

[[2]]
[1] 3.141593 3.141593

[[3]]
[1] 3.141593
```

tapply()

requires variable, Factor and function

```
> data(iris)
> attach iris
> tapply(Sepal.Width, Species, mean)
    setosa versicolor  virginica
     3.428      2.770      2.974
```

User Input

```
a<-scan()
menu(c("abc","def"),title="Enter value")
menu(c("abc","def"),graphics=TRUE, title="Enter value")
```

GUIs with Tcl/Tk

examples taken from http://bioinf.wehi.edu.au/~wettenhall/RTclTkExamples/mb.html

```
require(tcltk)
ReturnVal <- tkmessageBox(title="Greetings from R TclTk",
message="Hello, world!",icon="info",type="ok")
```

GUIs with Tcl/Tk - MessageBoxes

```r
require(tcltk)
tkmessageBox(message="Do you want to save before quitting?",
icon="question",type="yesnocancel",default="yes")
```

Tips & Hints

Some useful stuff:

`system()` runs OS commands

`xtable()` generates LaTeX code (package **xtable**)

`latex()` generates LaTeX code (package **Hmisc**)

`eval()` evaluate string as command

`parse()` evaluate string as command