

Loops and Flow Control

Programming in R for Data Science

Anders Stockmarr, Kasper Kristensen, Anders Nielsen

Loops in R

- ▶ In **R** the **for loop** is used to perform a task for each element in a set.

- ▶ Example:

- ▶ Given a set of integers:

```
> 1:3
```

```
[1] 1 2 3
```

- ▶ Let a variable i run through the set and print $i + i$:

```
> for(i in 1:3) {  
+   cat(i, "+", i, "=", i+i, "\n")  
+ }
```

```
1 + 1 = 2
```

```
2 + 2 = 4
```

```
3 + 3 = 6
```

- ▶ The sequence to loop through is not limited to a vector of integers. For example, we could let a variable run through a **list**, **data.frame** or **matrix**.

- ▶ Example:

- ▶ `> d <- data.frame(a = 1:2, b=2:3)`

- ▶ Print column sums of d :

```
> for(x in d) {  
+   cat("Column sum:", sum(x), "\n")  
+ }
```

```
Column sum: 3
```

```
Column sum: 5
```

Flow Control: if and if else statements

- ▶ `if(cond) expr:`

```
> for(i in 1:3){  
+   if (i==2) cat("This index is even:", "\n")  
+   cat(i, "\n")  
+ }
```

1

This index is even:

2

3

- ▶ `if(cond) cond.expr else alt.expr:`

```
> for(i in 1:3){  
+   if (i==2) cat("The index is 2", "\n") else  
+               cat("The index is not 2", "\n")  
+ }
```

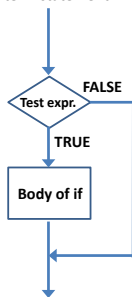
The index is not 2

The index is 2

The index is not 2

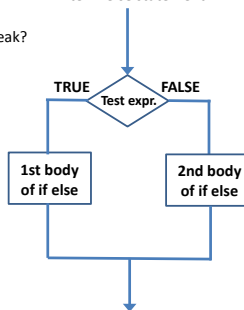
Flow charts for if and if else statements

Enter if statement



Enter if else statement

Break?



While and repeat loops

The **while** loop:

- ▶ `while(cond) expr`

The **repeat** loop:

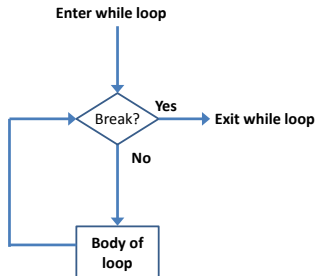
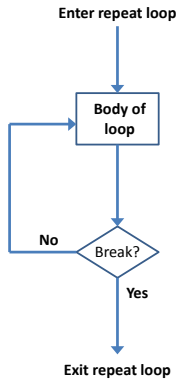
- ▶ `repeat expr`

The repeat loop has to be exited manually.

Flow controllers:

- ▶ **next:** Halts the execution of the current iteration of the loop and advances to the next
- ▶ **break:** Exits the loop

Flow charts for repeat and while loops



While loop example

Storing of machine parts

```
> k<-0 # number of big parts (>2)
> y<-abs(rnorm(1000)) # simulated part size
> i<-0 # index of parts
> # loop:
> while(k<3 & i<1000){
+   i<-i+1
+   temp<-y[i]
+   k<-k+(temp>2)
+ }
> rm(temp)
> i
[1] 42
```

Repeat loop example

Selection of persons without blue or yellow eyes

```
> eye.colors<-c("brown","blue","green","yellow","grey")
> eyecolor<-data.frame(personId=1:100,color=
+                       sample(eye.colors,100,rep=T))
> i<-0
> list.of.ids<-numeric(0) # patient ID list
> #loop:
> repeat {
+   i<-i+1
+   if(eyecolor$color[i]=="yellow" |
+       eyecolor$color[i]=="blue") next
+   list.of.ids<-c(list.of.ids,eyecolor$personId[i])
+   if(i==100 | length(list.of.ids)==20) break
+ }
> list.of.ids

[1]  5  6  7  9 10 11 12 14 15 18 19 20 21 22 23 24 25 28
[19] 29 30
```


Loops and run times

When the amount of data is large, loops can significantly increase run time:

```
> y<-matrix(rnorm(1000000),nrow=1000)
> z<-0*y
> time1<-as.numeric(Sys.time())
> #loop:
> for(i in 1:1000){
+   for(j in 1:1000){
+     z[i,j]<-y[i,j]^2
+   }
+ }
> time2<-as.numeric(Sys.time())
> # using object form in R:
> z<-y^2
> time3<-as.numeric(Sys.time())
> # run time increase factor:
> (time2-time1)/(time3-time2)

[1] Inf
```

lapply() and sapply()

- ▶ Functions which apply functions are to vectors, matrices, arrays or lists.
- ▶ The lapply() and sapply() function apply functions to the input.
- ▶ The lapply() and sapply() are similar, except that:
- ▶ The output from lapply() is a list;
- ▶ The output from sapply() is a vector or a matrix.

lapply() and sapply examples

A simple example: Sum the columns of a data frame.

```
> my.data<-data.frame(data1=rnorm(10),data2=rnorm(10),data3=rnorm(10))
```

```
> lapply(my.data,sum)
```

```
$data1
```

```
[1] 3.189681
```

```
$data2
```

```
[1] 2.711566
```

```
$data3
```

```
[1] -3.247268
```

```
> sapply(my.data,sum)
```

data1	data2	data3
3.189681	2.711566	-3.247268

lapply() and sapply() examples

A less simple example: Applying lapply() and sapply() to lists.

```
> A<-matrix(1:9,nrow=3);B<-matrix(1:16,nrow=4);C=matrix(1:8,nrow=4)
> my.list<-list(A=A,B=B,C=C); my.list
```

\$A

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

\$B

	[,1]	[,2]	[,3]	[,4]
[1,]	1	5	9	13
[2,]	2	6	10	14
[3,]	3	7	11	15
[4,]	4	8	12	16

\$C

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

lapply() and sapply() examples

Now let's suppose that we want to extract the second column of each of these matrices. We can do this with the square bracket function:

```
> lapply(my.list, "[", ,2)
```

```
$A
```

```
[1] 4 5 6
```

```
$B
```

```
[1] 5 6 7 8
```

```
$C
```

```
[1] 5 6 7 8
```

sapply() cannot do this job!