Intro to R - 3. Functions, Loops and Apply
OIT/SMU Libraries Data Science Workshop Series

Michael Hahsler

OIT, SMU
Functions

Conditions and Loops

apply, lapply, sapply, ...

Exercises
Section 1

Functions
R offers a wide range of useful functions. Here are some examples:

- **Statistics:** min, max, mean, median, quantile, sd, var, cor, round
- **Sorting:** sort, order, rev, rank
- **Random numbers:** runif, rnorm,...

```r
x <- runif(10)
x
## [1] 0.1137 0.6223 0.6093 0.6234 0.8609 0.6403 0.0095 0.2326 0.6661
## [10] 0.5143
summary(x)
## Min. 1st Qu. Median Mean 3rd Qu. Max.    
## 0.01  0.30  0.62  0.49  0.64  0.86
sort(x)
## [1] 0.0095 0.1137 0.2326 0.5143 0.6093 0.6223 0.6234 0.6403 0.6661
## [10] 0.8609
sort(x, decreasing = TRUE)
## [1] 0.8609 0.6661 0.6403 0.6234 0.6223 0.6093 0.5143 0.2326 0.1137
## [10] 0.0095
```
### User defined functions

R is a functional programming language. Functions are objects of mode “function’’.

```r
# defining a function
inc <- function(x) { x + 1 }
inc
## function(x) { x + 1 }

mode(inc)
## [1] "function"

# calling the function
inc(5)
## [1] 6

# functions that use only vectorized operators (e.g., +)
# are automatically vectorized.
inc(1:10)
## [1] 2 3 4 5 6 7 8 9 10 11
```

**Note**

Functions return the value of the last expression or can be specified via `return(value)`. 
Named arguments and default values

```r
inc <- function(x, by = 1) { x + by }

inc(5)  # using a default value
## [1] 6
inc(1:5, 10)  # using argument order
## [1] 11 12 13 14 15
inc(1:5, by = 10)  # using argument order + names
## [1] 11 12 13 14 15
inc(by = 10, x = 1:5)  # using argument names
## [1] 11 12 13 14 15
inc(matrix(1:4, nrow = 2), 10)
## [,1] [,2]
## [1,] 11 13
## [2,] 12 14
```

**Advanced Knowledge**

Since functions are regular (first class) objects they can be passed on as arguments and returned by functions.
**Remember:** Objects have a class that can be seen when calling

```r
x <- data.frame(a = 1:2, b = c("A", "B"))
class(x)
```

```r
## [1] "data.frame"
```

Many functions in R (e.g., `print`, `plot`) look at the supplied object and then choose automatically an appropriate behavior. These functions are called **generic** functions. The **implementations** have the object type in the name after a dot.

**Example:** `print` is generic and calls for a `data.frame` the implementation `print.data.frame`.

```r
print(x)
```

```r
## a b
## 1 1 A
## 2 2 B
```

You can find the manual page using `?print.data.frame`

**Hint**

When you type a function name in RStudio, it shows you all the implemented methods in the auto-completion context menu.
Section 2

Conditions and Loops
The if Statement

```r
x <- 12
if (x > 10) {      # result of condition needs length 1
  print("x is > 10")
} else {
  print("x is <= 10")
}
## [1] "x is > 10"
```

Using vectors to make decisions

```r
x <- c(12, 16, 3)
if (all(x > 10)) print("All values in x are >10")
if (any(x > 10)) print("There is at least one value >10")
## [1] "There is at least one value >10"
```

```r
c(FALSE, TRUE, TRUE) | c(FALSE, TRUE, FALSE) # element-wise OR (see &)
## [1] FALSE  TRUE  TRUE

c(FALSE, TRUE, TRUE) || c(FALSE, TRUE, FALSE) # only eval. the 1. elements
## [1] FALSE
```
The for loop

```r
# calculate the sum of the integers 1 to 5
x <- 0
for (i in seq_len(5)) {
  x <- x + i
}
x
## [1] 15

# we can also use lists (R recycles values!)
l <- list(a = 2, b = 1:2, c = 4)
x <- 0; for(i in l) { x <- x + i }
x
## [1] 7 8
```
The while loop

```r
# calculate the sum of the integers 1 to 5
x <- 0
i <- 1
while (i <= 5) { x <- x + i; i <- i + 1 }
x
## [1] 15

# in R we would rather use a vectorized function
sum(1:5)
## [1] 15
```

**Note**

Loops are not frequently used in R since most problems can be solved more efficiently using functions and vectorization.
Section 3

apply, lapply, sapply, ...
apply – apply functions to a matrix

```r
m <- matrix(1:9, nrow = 3)
m
## [,1] [,2] [,3]
## [1,] 1 4 7
## [2,] 2 5 8
## [3,] 3 6 9
apply(m, MARGIN = 1, sum) # apply sum to rows
## [1] 12 15 18
apply(m, MARGIN = 2, sum) # apply sum to cols
## [1] 6 15 24
# same as
rowSums(m); colSums(m)
## [1] 12 15 18
## [1] 6 15 24
```
lapply/sapply – apply functions to each element in a list

```r
l <- list(1:3, 6, 7:3)
# apply rev to all elements
lapply(l, FUN = function(x) { rev(x) })

## [[1]]
## [1] 3 2 1
##
## [[2]]
## [1] 6
##
## [[3]]
## [1] 3 4 5 6 7

# apply automatically "simplifies" the result. Here into a vector.
sapply(l, length)

## [1] 3 1 5
```
Section 4

Exercises
Exercises

1. Create \( x \) by \( x \leftarrow \text{runif}(100) \). Write a function with the name `avg_gt` with two formal arguments: a vector \( x \) and a value \( gt \). The function computes the average of the values greater than \( gt \) in \( x \). Write a version with a loop and if and one version without loops and if statements.

2. Create a list with 5 numeric vectors (lengths and values of your choice). Sort all vectors in the list. Hint: see `sort()`.

3. Write a function that computes the smallest value in each column of a given matrix. Create a random \( 5 \times 5 \) matrix to test the function.