# Introduction to R*

Lecture 3: Control-flow and functions

Wim R.M. Cardoen

Last updated: 10/13/2022 @ 14:45:19

## Contents

# 1 R Control-flow

## 1.1 Conditional constructs

- **if**(*condition*){

  }[**else if**(*condition*){

  }]

  [**else**{}]

The `R` language provides a **vectorized ifelse()** function.

Syntax:
**ifelse**(*cond*, *vecy*, *vecn*)
where:

- *cond* : test condition
- *vecy* : values in case of **TRUE** values
- *vecn* : values in case of **FALSE** values

### 1.1.1 Examples

```
score <- 75.0

if(score>=90.0){
  grade <- 'A'
} else if((score<90.0) && (score>=80.0)){
  grade <- 'B'
} else if((score<80.0) && (score>=70.0)){
  grade <- 'C'
} else{
  grade <- 'D'
}
cat(sprintf("Score:%4.2f -> Grade:%s\n", score, grade))
```

```
Score:75.00 -> Grade:C
```

```
x <- c(-1,2,1,-5,-7)
c
```

```
function (...)  .Primitive("c")
```

```
res <- ifelse(x>=0, x,-x)
res
```

```
[1] 1 2 1 5 7
```

## 1.2 Loop constructs

There are several loop constructs:

- **while**(*cond*){
  *body of the loop*
  }
- **for**(*obj* **in** *sequence*}){
  *body of the loop*
  }
- **repeat**
  *body of the loop*
  }

  The **repeat** loop has no condition to leave the loop:
  insert a **break** statement to leave the (infinite) loop.

The **break** statement allows one to break out of the **while**, **for** and **repeat** constructs.

The **next** statement allows to go to the next iteration.

### 1.2.1 Examples

- for loop construct

```
# Loop over all items
fruit <- c("apple", "pear", "banana", "grape")
for(item in fruit){
  cat(sprintf(" Fruit:%s\n", item))
}
```

```
 Fruit:apple
 Fruit:pear
 Fruit:banana
 Fruit:grape
```

```
# Skip all numbers which are multiples of 3
x <- sample(1:100, size=10, replace=FALSE)
x
```

```
 [1] 11 36 82 54 89 63 25 70 49 71
```

```
for(item in x){
  if(item%%3==0)
     next
  cat(sprintf(" %3d is NOT a multiple of 3\n", item))
}
```

```
  11 is NOT a multiple of 3
  82 is NOT a multiple of 3
  89 is NOT a multiple of 3
  25 is NOT a multiple of 3
  70 is NOT a multiple of 3
  49 is NOT a multiple of 3
  71 is NOT a multiple of 3
```

- while loop

```
x <- sample(1:1000, size=100, replace= FALSE)
isFound <- FALSE
i <- 1
while(!isFound){
  if(x[i]%%7==0){
      cat(sprintf(" %3d is divisible by 7\n", x[i]))
      isFound <- TRUE
  }
  else{
      cat(sprintf(" %3d is NOT divisible by 7\n", x[i]))
      i <- i + 1
  }
}
```

```
 377 is NOT divisible by 7
 619 is NOT divisible by 7
 274 is NOT divisible by 7
 355 is NOT divisible by 7
 396 is NOT divisible by 7
  73 is NOT divisible by 7
 806 is NOT divisible by 7
 641 is NOT divisible by 7
  52 is NOT divisible by 7
 434 is divisible by 7
```

- repeat loop

```
i <- 1
repeat{
  # Stop the loop as soon as you find a multiple of 7.
  if(x[i]%%7==0){
      cat(sprintf(" %3d is divisible by 7\n", x[i]))
      break
  }
  else{
      cat(sprintf(" %3d is NOT divisible by 7\n", x[i]))
      i <- i + 1
  }
}
```

```
 377 is NOT divisible by 7
 619 is NOT divisible by 7
 274 is NOT divisible by 7
 355 is NOT divisible by 7
 396 is NOT divisible by 7
  73 is NOT divisible by 7
 806 is NOT divisible by 7
 641 is NOT divisible by 7
  52 is NOT divisible by 7
```

```
434 is divisible by 7
```

## 1.3 Exercises

- Write code to find the smallest of three numbers, e.g. 21, 12, 17
- The **Fibonacci** sequence is defined by the following recurrence relation:

  $$F_n = F_{n-1} + F_{n-2}$$

  where $F_0 = F_1 = 1$.
  Calculate all Fibonacci numbers up to $F_{15}$.
- The square root of a number $n$ is equivalent to solving the following equation:

  $$x^2 - n = 0$$

  The solution to this equation can be found iteratively by using e.g. the Newton-Raphson method.

  Iteration $i + 1$ for $x$ is then given by:

  $$x_{i+1} = \frac{1}{2}(x_i + \frac{n}{x_i})$$

  Find the square root of 751 to a precision of at least 8 decimals. You can set $x_0$ to $n$ itself.

# 2 R Functions

- switch function
- lexical scoping
- simple functions
- args(), formals()
- default arg, ...
- lazy evaluation
- closure
- anonymous functions
- make your own operators
- loop functions: {l,s,m}apply, split

## 2.1 Examples

## 2.2 Exercises

- Write your own factorial function named `myfactorial(n)`.
  The factorial function, $n!$ is defined as:
  $$n! = n\,(n-1)!$$
  where $0! := 1$.
- Write your own function named `throwdie(n)` which simulates throwing a die $n$ times.
  - Assume you have a fair die.
  - Adjust the function `throwdie(n)` for the general case i.e. a non-fair die.
  - Hint: you can use `R`'s **sample()** function.
- An auto-regressive time series of type AR1 is defined as follows:
  $$x_i = \varphi\, x_{i-1} + \varepsilon_i$$
  where $\varepsilon_i \sim N(0, \sigma^2)$
  - Write the function `genAR1Series(n=1000, x0=0.0, phi=0.7)`
    which returns the AR1 time series $\{x_i\}$ for $i \in \{1, \ldots, n\}$,
    where:
    $$\begin{aligned} x_0 &= 0.0 \\ \varphi &= 0.7 \\ \varepsilon_i &\sim N(0,1)\ \ \forall\, i \in \{1, \ldots, n\} \end{aligned}$$
  - Write functions to calculate the sample autocovariance ($\gamma(h)$) and the sample autocorrelation function ($\rho(h)$) which are defined as follows:
    $$\begin{aligned} \gamma(h) &:= \frac{1}{n}\sum_{i=1}^{n-h}\Big(x_{i+h} - \overline{x}\Big)\Big(x_i - \overline{x}\Big) \\ \rho(h) &:= \frac{\gamma(h)}{\gamma(0)} \\ \text{where }\ \overline{x} &:= \frac{1}{n}\sum_{i=1}^{n} x_i \end{aligned}$$
  - Calculate the autocovariance and autocorrelation for the time series you generated previously.