R-Course: Day 4

Lukas Mödl, Matthias Becher, Erin Sprünken

Institut für Biometrie und Klinische Epidemiologie

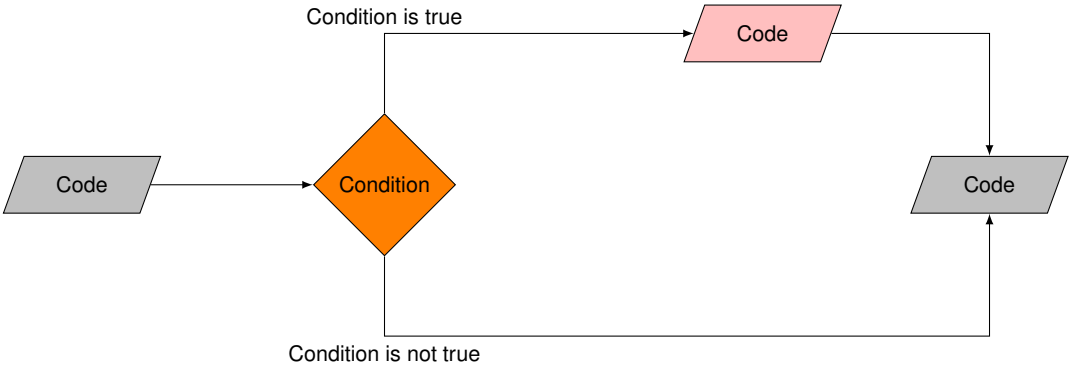
Charité - Universitätsmedizin Berlin, Berlin

erin-dirk.spruenken@charite.de

November 9, 2022

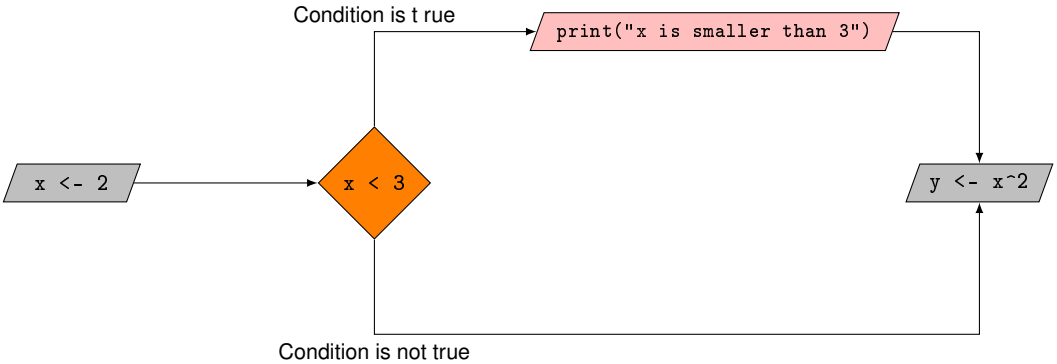
Conception if-statement

If we want to execute a block of code dependent on a certain value, we choose an if-statement. If-statements define such code-blocks, that will only be executed if a certain user-defined condition is true.



Example if-statement

To understand the idea, we want to assign the value 2 to `x`. Dependent on the value of `x` we want to print an output.



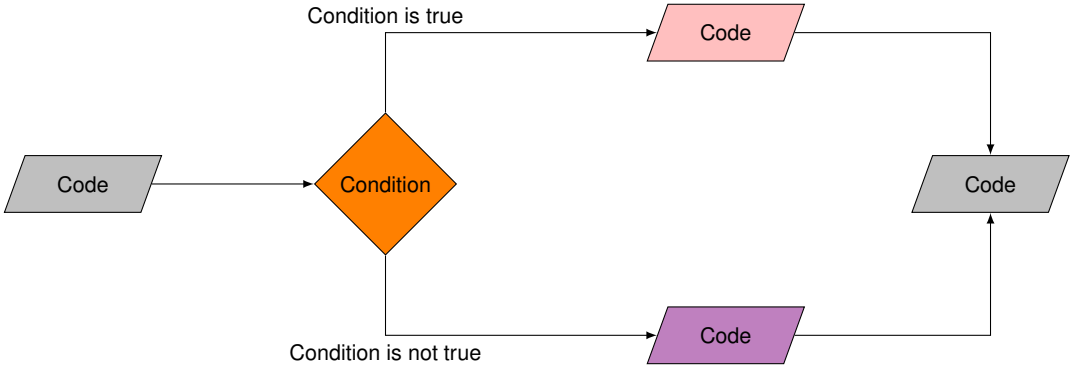
Example if-statement

To understand the idea, we want to assign the value 2 to x. Dependent on the value of x we want to print an output.

```
> x <- 2
> if(x < 3){
+     print("x is smaller than 3")
+ }
[1] "x is smaller than 3"
> y <- x^2
>
```

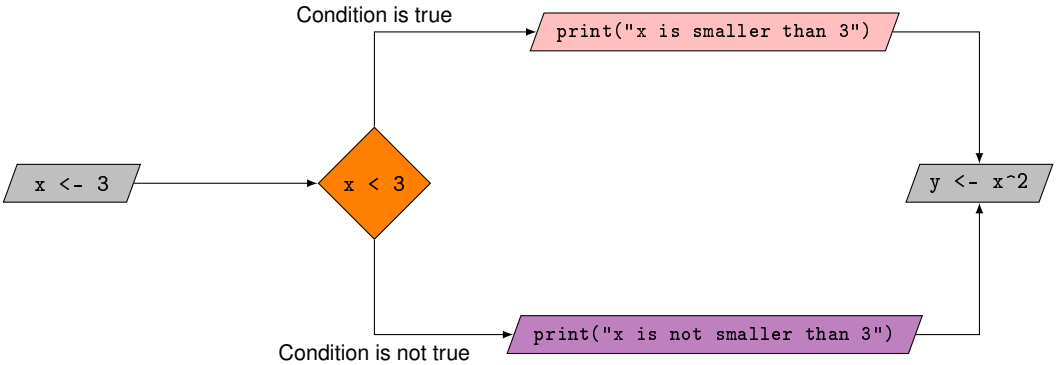
Conception if-else-statement

Sometimes, we don't want to test for only one condition but rather have an either this or that situation. In that case, we use the if-else-statement.



Example if-else-statement

To understand this concept, we will modify our former example.



Example if-else-statement

To understand this concept, we will modify our former example.

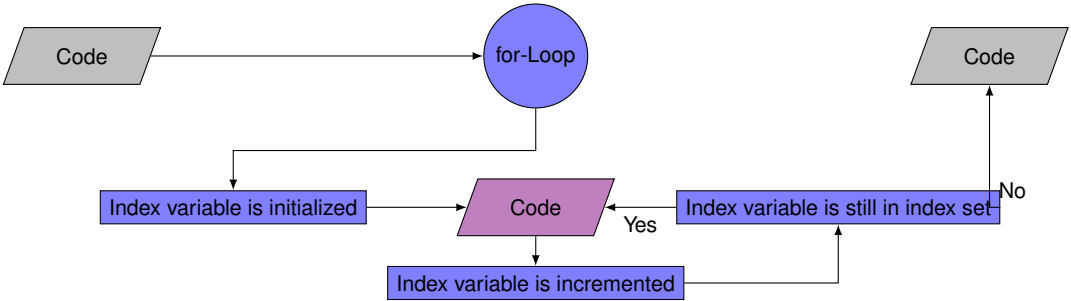
```
> x <- 3
> if(x < 3){
+     print("x is smaller than 3")
+ } else {
+     print("x is not smaller than 3")
+ }
[1] "x is not smaller than 3"
> y <- x^2
```

Remarks to the if-else-statement

- Using the brackets { and } is essential!
- Only logical statements are allowed as a condition
- Sometimes, nested statements and conditions are necessary
- Indentation is improving the readability of a code!

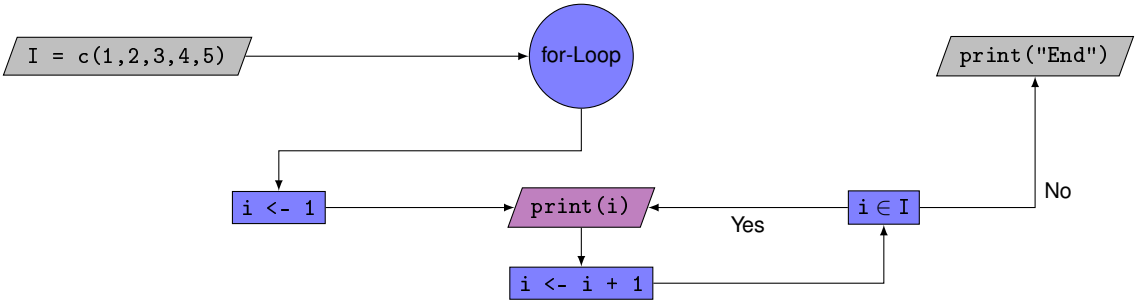
Conception for-Loop

An important loop in R is the for-loop. For-loops execute a block of code as often as a certain index variable in an index set (a vector or a list) exists. We construct this by the keyword `for()` followed by a block of code in curly brackets. The round parentheses are used to determine the index variable and index set.



Example for-Loop

This example emphasizes the for-loop. Here, `I` is the index set and `i` the index variable.



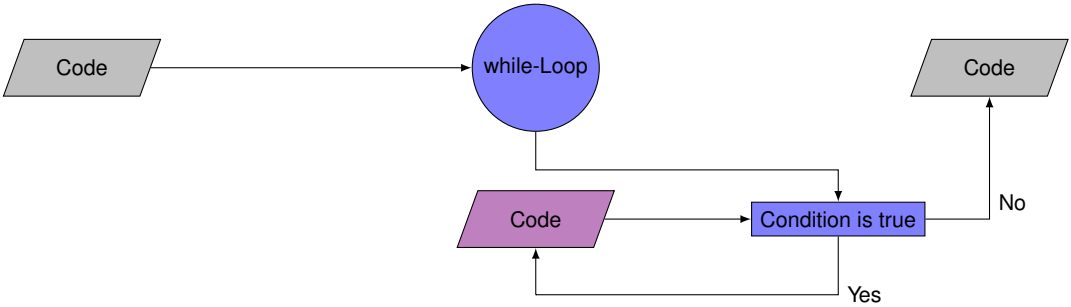
Example for-Loop

This example emphasizes the for-loop. Here, `I` is the index set and `i` the index variable.

```
> I <- c(1, 2, 3, 4, 5)
> for(i in I){
+     print(i)
+ }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
> print("End")
[1] "End"
>
```

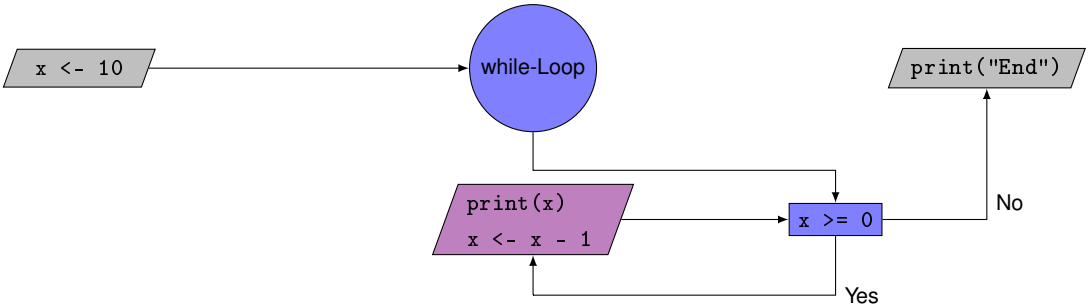
Conception while-Loop

Another elementary loop is the while-loop. Contrary to the for-loop it executes code *while* a certain condition is true. It is constructed using the keyword `while()` followed by a block of code in curly brackets. Within the round parentheses the condition is given.



Example while-Loop

This example demonstrates the while-loop. The condition here is `x >= 0`.



Example while-Loop

This example demonstrates the while-loop. The condition here is `x >= 0`.

```
> x <- 10
> while(x >= 0){
+     print(x)
+     x <- x - 1
+ }
[1] 10
[1] 9
[1] 8
[1] 7
[1] 6
[1] 5
[1] 4
[1] 3
[1] 2
[1] 1
[1] 0
> print("End")
[1] "End"
```

Remarks to for- and while-Loops

- The brackets `{` und `}` are essential!
- Beware of infinite loops!
- Any for-loop can be converted into a while loop and vice versa.
- Using `break` a loop can be terminated
- With `next` an iteration can be skipped without terminating the loop
- The `apply`-Family offers a nice alternative to loops

Apply-Family

The apply-family is a collection of functions in R that allows us to *apply* a function onto several inputs one after another. For example, on all rows (or columns) of a matrix, all elements of a list, etc. The different functions are:

- `apply()`
- `lapply()`
- `sapply()`
- `tapply()`

apply()

With `apply()` we are able to *apply* a certain function on all rows or columns of a data frame or a matrix, for example to compute all columnwise sums. The basic form is:

- `apply(data, margin, function)`

For example:

```
> apply(data, 1, sum)
[1] 10 26 42 58
> apply(data, 2, sum)
[1] 28 32 36 40
```

1	2	3	4	10
5	6	7	8	26
9	10	11	12	42
13	14	15	16	58
28	32	36	40	

lapply()

`lapply()` executes a function on each element of a data frame, a matrix, a vector or a list. The "l" in `lapply()` is meant for "list" and relates to the fact that `lapply()` always returns a list.

- `lapply(object, function)`

Zum Beispiel:

```
> lapply(c("A","B","C"), tolower)
[[1]]
[1] "a"

[[2]]
[1] "b"

[[3]]
[1] "c"
```

sapply()

`sapply()` works in the same way as `lapply()` but always tries to return a nice vector or matrix instead of a list (if possible).

- `sapply(object, function)`

Zum Beispiel:

```
> sapply(c(-1, 2, -3), abs)
[1] 1 2 3
> l <- list(a = 1:10, b = 1:20)
> sapply(l, mean)
   a    b
 5.5 10.5
```

tapply()

`tapply()` allows us to execute functions on different subgroups with respect to some factor variable:

- `tapply(object, index, function)`

Zum Beispiel:

```
> data <- data.frame(Sex = as.factor(c(rep("M",100), rep("W",100))))
> data$Height <- c(rnorm(100,180,5),rnorm(100,166,4))
> tapply(data$Height, data$Sex, mean)
       M        W
180.3354 165.3481
```

Conception of Custom Functions

So far, we only used pre-defined functions, as for example `t.test`. However, R allows us to write our own functions.

Functions have an own datatype that is initalized with the keyword `function()`, followed by a block of code in curly brackets. Within the round parentheses we can define variables freely, which are interpreted as function arguments.

Example of a function with one argument

For the mathematical function $f(x) = x^2$, the variable x is free (independent). If we want to construct a function in R that does the same, we do so as follows:

```
> f <- function(x)
+ {
+     return(x^2)
+ }
>
```

Remarks to the Definition of Custom Functions

- Using the brackets `{` und `}` is essential!
- If you specify function arguments, the user must specify them with each call of the function. An exception is, if we provide default values within the parentheses by initializing the argument directly with =
- If your function takes more than one argument, you must separate the arguments with a comma
- R tries to vectorize automatically (if posssible), meaning that if a vector or matrix is given as an argument, the function will try to apply the function on each element separately.
- If a function is not meant to return anything, then the `return()` statement can be omitted.