Loops, conditional statements and functions in R

Quantitative Applications for Data Analysis

Alexey Fedoseev

January 23, 2024

Lists

We already know about vectors, a structure that can hold multiple values of the *same* data type.

Another built-in data type in R that allows to hold multiple values of *different* data types is called lists.

Lists are the R objects which contain elements of different types like - numbers, strings, vectors and another list inside it. A list can also contain a matrix or a function as its elements. List is created using the `list` command:

```
> my.list <- list(34, 183.5, "Blue", TRUE, c(11,8,3))
```

Lists

```
> my.list
[[1]]
[1] 34

[[2]]
[1] 12.56

[[3]]
[1] "Blue"

[[4]]
[1] TRUE

[[5]]
[1] 11  8  3
```

Named lists

If you would like to give some meaning to your values in a list, you can create a *named list*:

```
> my.named.list <- list(score = 183.5, color = "Blue", is.cold = TRUE)
```

Alternatively, you can use the `names` command:

```
> my.named.list <- list(183.5, "Blue", TRUE)
> names(my.named.list) <- c("Score", "Color", "is.cold")
```

Named lists

```
> my.named.list
$Score
[1] 183.5

$Color
[1] "Blue"

$is.cold
[1] TRUE
```

Accessing elements of a list

- Elements of (unnamed) lists are indexed by [[. . .]]:

```
> my.list[[1]]
[1] 34
> my.list[[3]]
[1] "Blue"
```

- Elements of named lists are extracted using the $ operator:

```
> my.named.list$Score
[1] 183.5
> my.named.list$is.cold
[1] TRUE
```

Repetitive tasks

Computers are very good at repeating commands!

You want to congratulate your friends, but you have so many of them!

```
> cat("Happy Birthday Erik!\n")
Happy Birthday Erik!
> cat("Happy Birthday John!\n")
Happy Birthday John!
> cat("Happy Birthday Alice!\n")
Happy Birthday Alice!
> cat("Happy Birthday Mike!\n")
Happy Birthday Mike!
```

These are only 4 friends out of hundreds that you have. If you want to change your congratulation message you would rather *not* change it hundred times.

Loops

Loops give you control over repetitive tasks. Loop means: ‘for’ each friend name ‘in’ friends vector repeat the commands inside the *code block* surrounded by the curly braces {}.

```
> friends <- c("Erik", "John", "Alice", "Mike")
> for (friend_name in friends) {
+   cat("Happy Birthday ", friend_name, '!\n', sep = "")
+ }
Happy Birthday Erik!
Happy Birthday John!
Happy Birthday Alice!
Happy Birthday Mike!
```

To change your congratulation message now you need to modify one line only! The additional parameter `sep = ""` for the command `cat` indicates that we want to concatenate our strings using *separator* empty string (`""`). It means that R with paste an empty string `""` between every string in our command. Necessarily we need to provide spaces at the ends of our strings ourselves.

Conditional statements

What if you have a very special friend? ‘If’ it is a special friend be specific, ‘else’ be general.

```
> friends <- c("Erik", "John", "Alice", "Mike")
> for (friend_name in friends) {
+   cat("Happy Birthday ", friend_name, '! ', sep = "")
+   if (friend_name == "Erik") {
+     cat("I wish you to stay healthy and feel great!\n")
+   }
+   else
+     cat("Best wishes!\n")
+ }
Happy Birthday Erik! I wish you to stay healthy and feel great!
Happy Birthday John! Best wishes!
Happy Birthday Alice! Best wishes!
Happy Birthday Mike! Best wishes!
```

You do not have to specify curly braces {} if your *code block* consists of only one command.

Loops

You want to wish your friends something amazing! Create another vector with your wishes and use the same index to go through both vectors.

```
> friends <- c("Erik",        "John",        "Alice",          "Mike")
> wishes <- c("feel great", "stay happy", "feel beautiful", "stay healthy")
> friends.indices <- seq_along(friends) # same as 1:length(friends)
> friends.indices
[1] 1 2 3 4
> for (index in friends.indices) { # for every 'element' in a 'vector'
+   cat("Happy Birthday ", friends[index], '! Wishing you to ',
+       wishes[index], '!\n', sep = "")
+ }
Happy Birthday Erik! Wishing you to feel great!
Happy Birthday John! Wishing you to stay happy!
Happy Birthday Alice! Wishing you to feel beautiful!
Happy Birthday Mike! Wishing you to stay healthy!
```

if...else statement

```
> ingredients <- c("sugar", "milk", "water", "flour", "salt", "vanilla")
> for (ingredient in ingredients) {
+   if (ingredient %in% c("sugar", "flour", "salt"))
+     cat(ingredient, "is measured in tablespoons\n")
+   else if (ingredient %in% c("water", "milk"))
+          cat(ingredient, "is measured in cups\n")
+        else if (ingredient %in% c("apple", "banana"))
+               cat(ingredient, "is measured in pounds (lb)\n")
+             else cat(ingredient, "is measured in gramms\n")
+ }
sugar is measured in tablespoons
milk is measured in cups
water is measured in cups
flour is measured in tablespoons
salt is measured in tablespoons
vanilla is measured in gramms
```

User input

What if you want to have more friends? You can create one.

```
> your_name <- readline(prompt = "Enter your name: ")
Enter your name: Alice
> cat("Hi ", your_name, '!\n', sep="")
Hi Alice!
```

Command `readline` allows you to ‘interact’ with R prompt, meaning you can type in your name and press `Enter` to let R know that you finished typing.

Conditional loop

'While' loop can repeat the task `while` certain condition is `TRUE`. Change of the condition to `FALSE` ends the loop. Use it when you do not how many times the task needs to be performed.

```
> user_input <- ""
> while (user_input != "stop") {
+   user_input <- readline(prompt = "\nWhat should I do? ")
+   cat(your_name, " wants me to ", user_input, ".", sep="")
+   if (user_input == "stop") cat(" Bye.\n")
+ }

What should I do? walk
Alice wants me to walk.
What should I do? dance
Alice wants me to dance.
What should I do? stop
Alice wants me to stop. Bye.
```

Breaking out of loops

To give you more control over your loops, R has a way to break out of a loop or skip an iteration.

As your code becomes more sophisticated you may find yourself looking to stop your loop earlier than normally anticipated:

```
> desired.friends <- 1000
> actual.friends <- 77
> friends.number <- 0
> while (friends.number < desired.friends) {
+   friends.number <- friends.number + 1
+ }
> cat("My desired number of friends is", friends.number, "\n")
My desired number of friends is 1000
```

Breaking out of loops

```
> desired.friends <- 1000
> actual.friends <- 77
> friends.number <- 0
> while (friends.number < desired.friends) {
+   if (friends.number >= actual.friends)
+     break
+   friends.number <- friends.number + 1
+ }
> cat("My actual number of friends is", friends.number, "\n")
My actual number of friends is 77
```

Skipping iterations

If you would like to skip an iteration, you can use the `next` command:

```
> friends <- c("Erik", "John", "Alice", "Mike")
> for (friend_name in friends) {
+   if (friend_name == "John")
+     next # John is not here today
+   cat("Hi ", friend_name, "!\n", sep = "")
+ }
Hi Erik!
Hi Alice!
Hi Mike!
```