Introduction to R Programming

Slide Set 4 and 5: R Programming

Maria Ptashkina

Barcelona GSE ITFD

September 2021

Introduction

- So far we've mostly focused on the features of R related to statistical computing and data work
- The next two classes are meant to introduce you to the foundational components of R programming
- A program is a set of step-by-step instructions for your computer to follow
- You can often reduce an R program into substasks so simple that each can be performed with a preexisting function
- Many of the pieces you've seen before, but we try to understand them beyond the pure application to data

Table of Contents

Control Flow

- Control flow (or flow of control) is simply the order in which we code and have our statements evaluated
- There are two primary tools of control flow: choices and loops
- Choices, like if statements, allow you to run different code depending on the input
- Loops, like for and while, allow you to repeatedly run code, typically with changing options

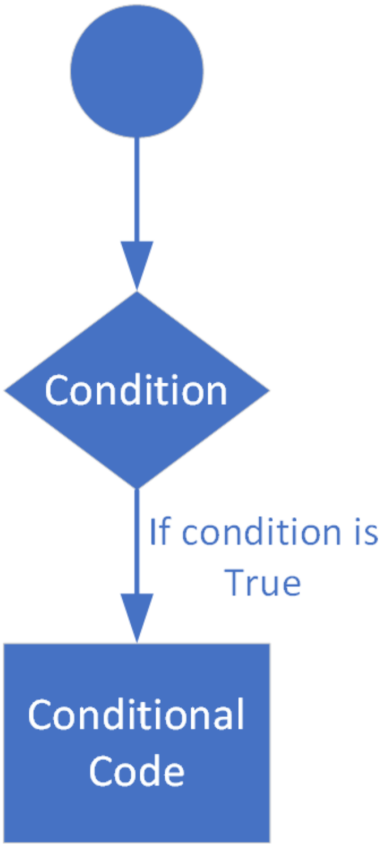
Relational Operators

- Relational operator is a construct that tests or defines some kind of relation between two entities
- You can compare numerical values, strings and logical operators
- Syntax
  - < for less than
  - > for greater than
  - <= for less than or equal to
  - >= for greater than or equal to
  - == for equal to each other
  - != not equal to each other
- For vectors and matrices R does element-wise comparison!

Logical Operators

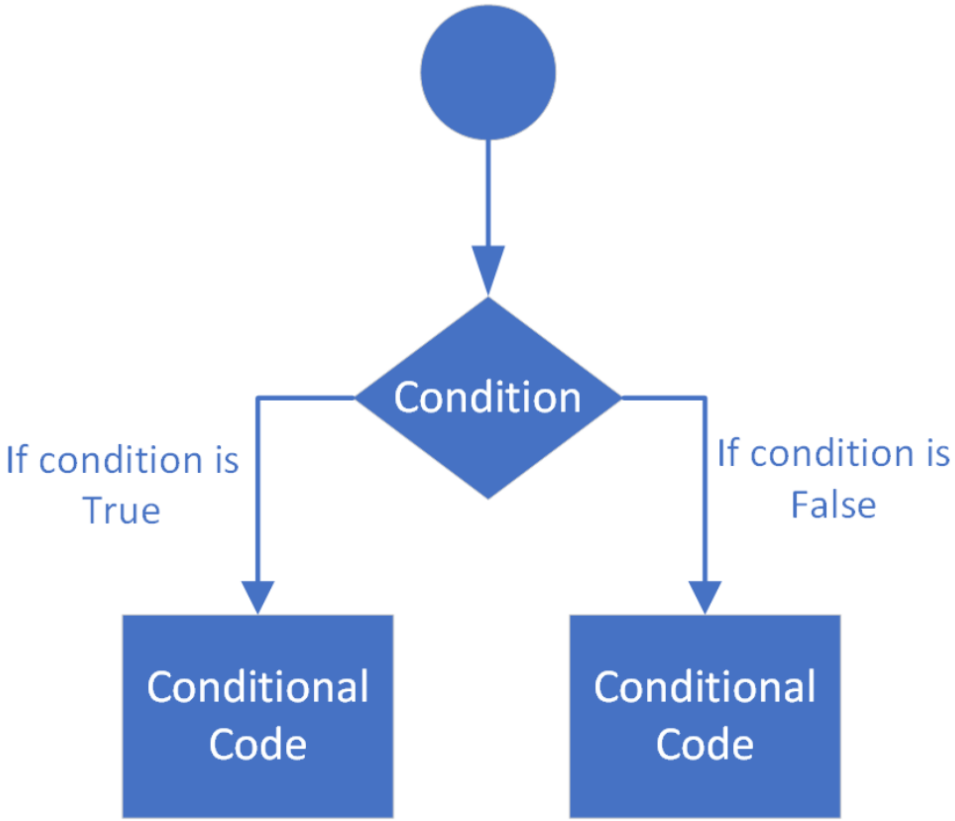
- Logical operators are used to carry out boolean operations like ‘and’ and ‘or’
- Syntax
  - ! logical NOT
  - & element-wise logical AND
  - | element-wise logacal OR
- Operators & and | perform element-wise operation producing result having length of the longer operand
- Operators && and || examines only the first element of the operands resulting into a single length logical vector

Conditional Statements: `if`



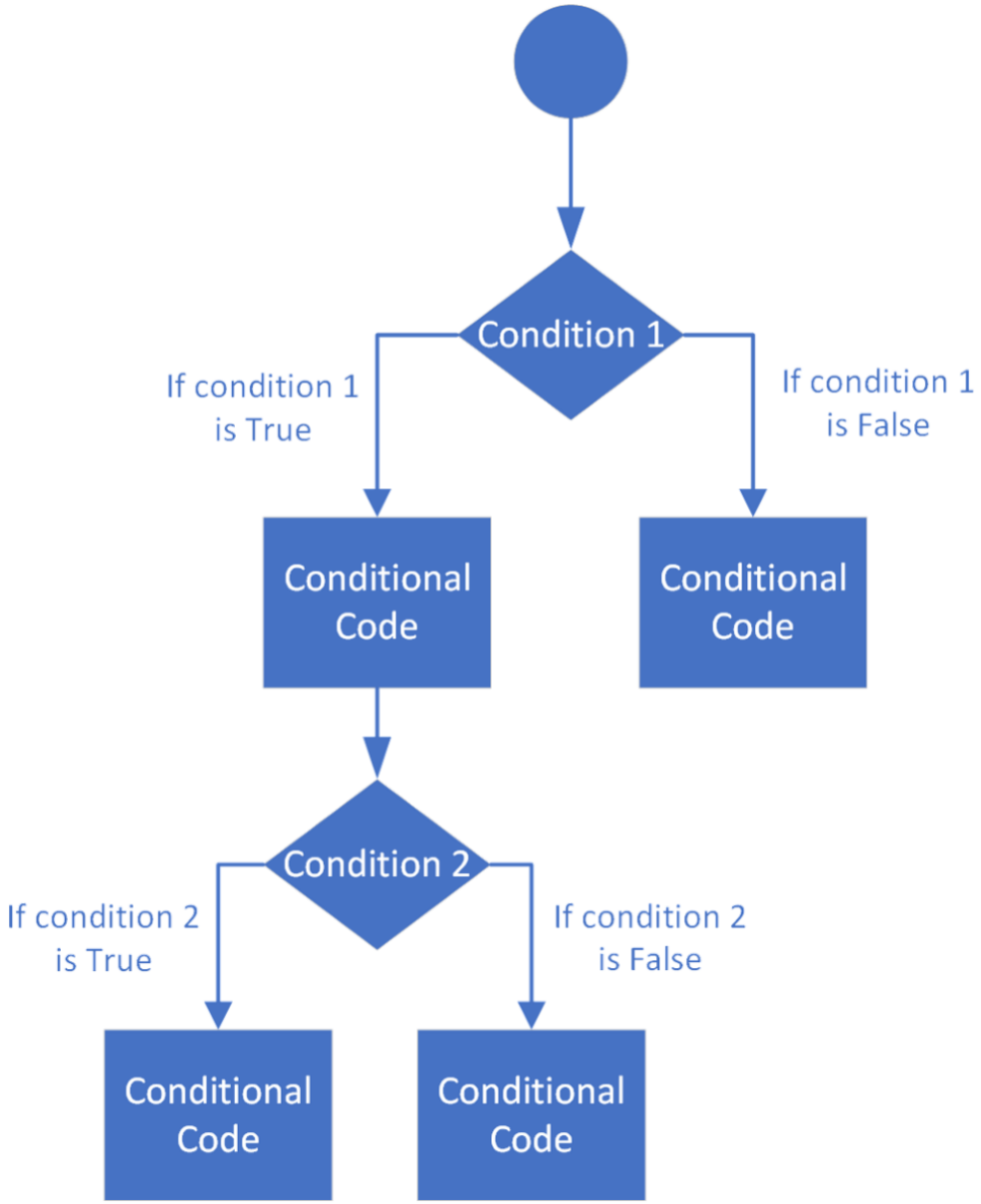

```
if (condition) {
expr
}
```

Conditional Statements: `if ... else`



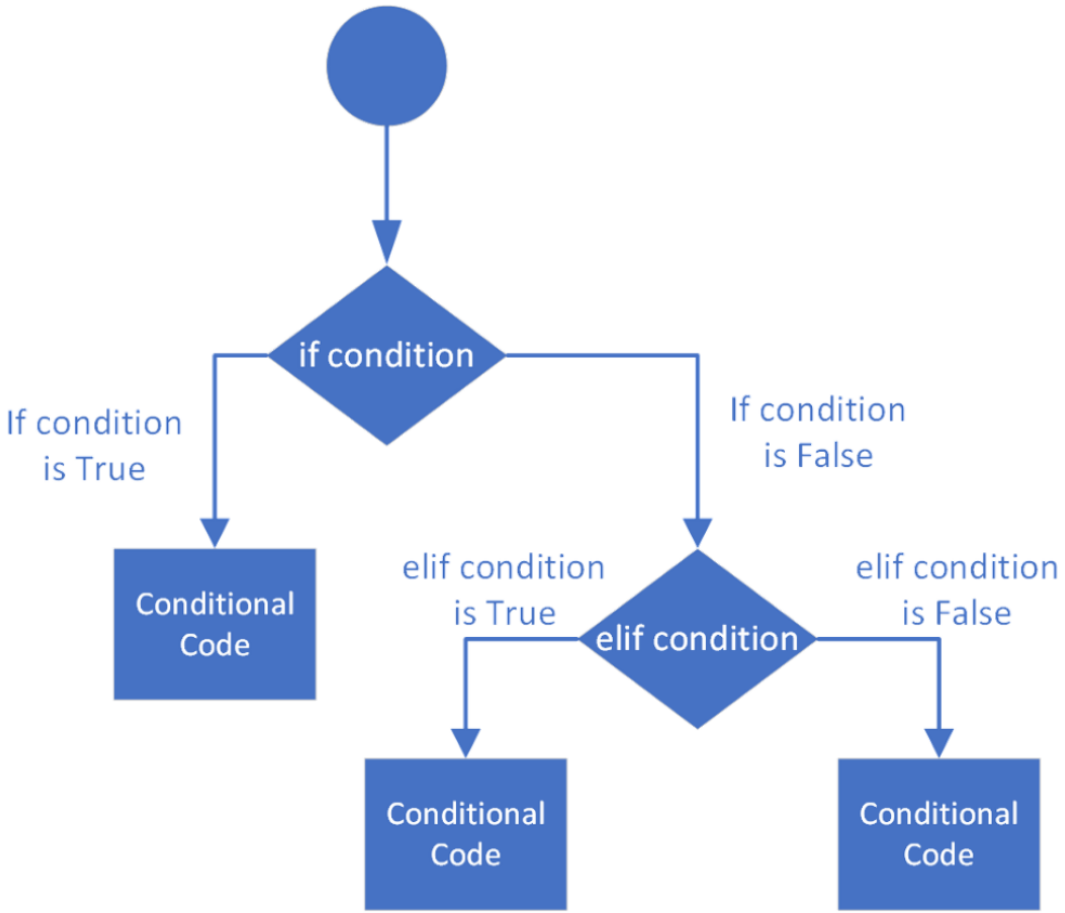

```
if (condition) {
expr1
} else {
expr2
}
```

Conditional Statements: Nested `if ... else`




```
if (conditio1n) {
expr1
if (condition2) {
expr2
}} else {
expr3
}
```

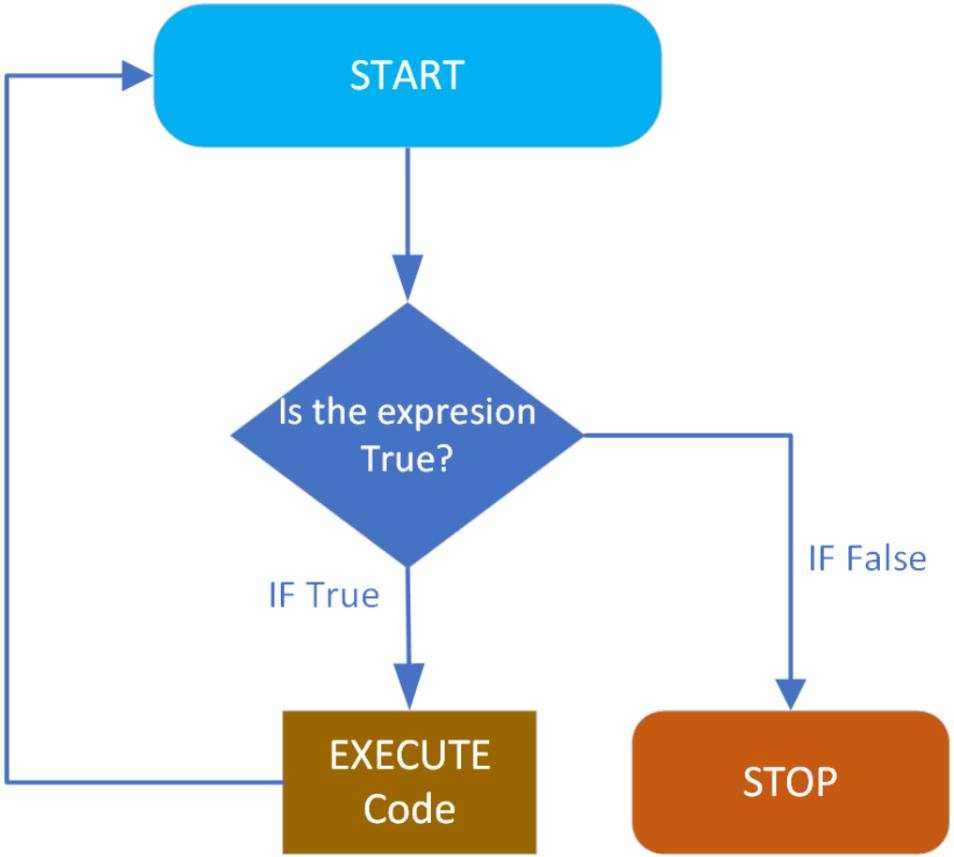
Conditional Statements: `if ... else if`




```
if (condition1) {
expr1
} else if
(condition2) {
expr2
} else if
(condition3) {
expr3
} else {
expr4
}
```

Table of Contents

while Loop

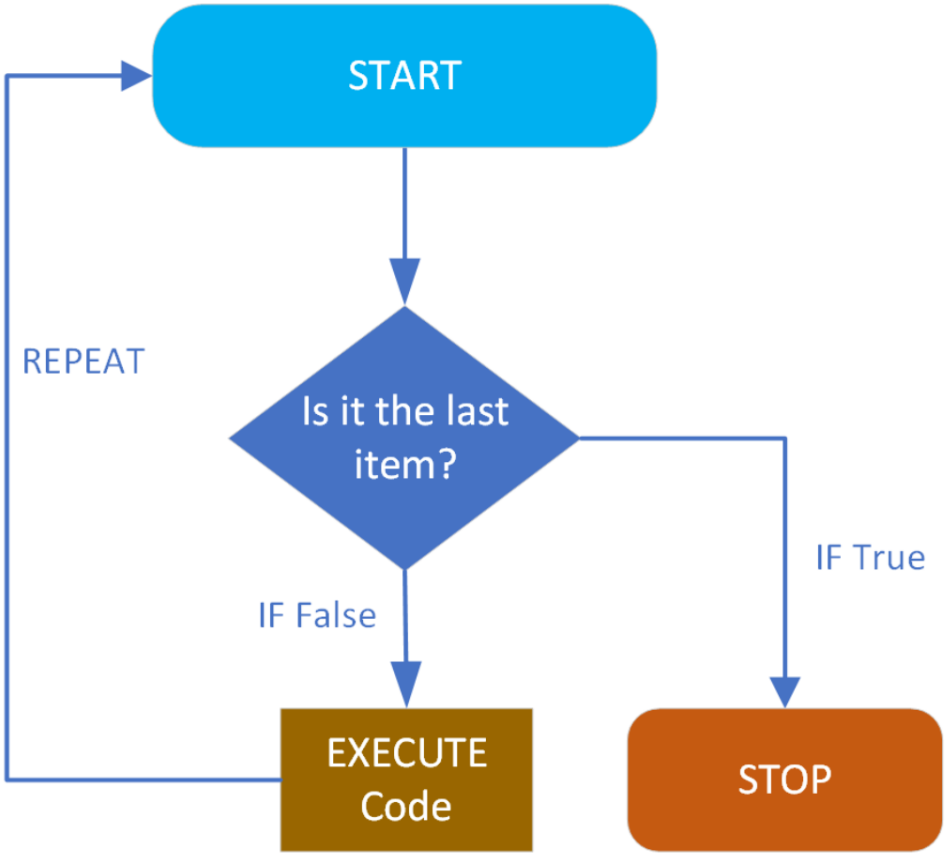



```
while (condition) {
expr
}
```

`while` Loop

- VERY IMPORTANT: In every step of the while loop change the argument in order to avoid an endless loop!
- `break` statement is a control statement: when `R` encounters it, the while loop is abandoned completely

for Loop



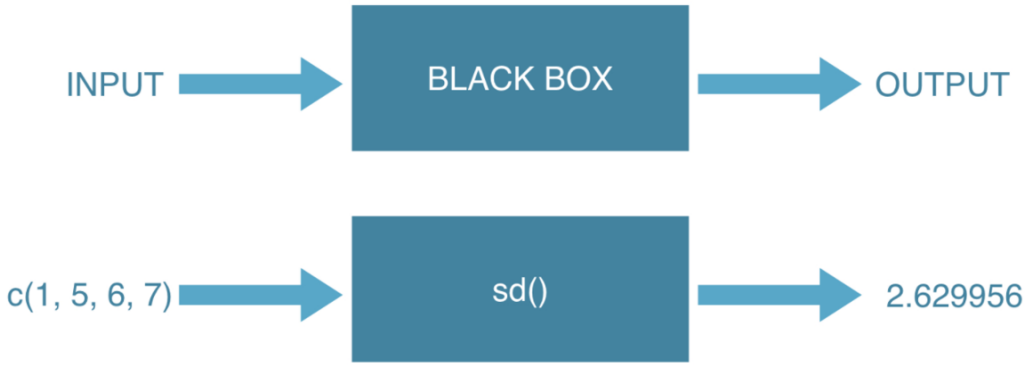

```
for (i in vector) {
expr
}
```

for Loop

- You can tell R to explicitly go through every element of the loop
- You can equivalently loop over a list
- To loop over a matrix make a nested loop to go through every row and every column
- Loops in simulations: very handy when one iteration depends on the value of a previous iteration (we will see later in an example)

Table of Contents

Using Functions



- Argument matching: by position or by name
- Remember about `na.rm` argument and how `R` treats missing values!
- Useful trick: `args()` function

Writing Functions

- Most of the time you will have a function already written and packaged for you in `CRAN`
- But you can also write your own functions as `R` objects

```
my_fun <- function(arg1, arg2) {
body
}
```

- You can write functions that does not require an input
- You can define default argument values in your own R functions
- Note! Variables that are defined inside a function are not accessible outside that function
- `R` passes arguments by value, i.e. an `R` function cannot change the variable that you input to that function

Table of Contents

lapply

- In R there is a very useful set of functional applications under the apply "family"
- It's a convenient substitute for writing for loops
- lapply takes a *vector* or *list* X, and applies the function FUN to each of its members
- If FUN requires additional arguments, you pass them after you've specified X and FUN
- The output of lapply() is a *list*, the same length as X, where each element is the result of applying FUN on the corresponding element

Anonymous Functions

- Defining functions to use them only once is kind of overkill
- So `R` allows you to use so-called anonymous functions
- Anonymous function is a function that you did not assign to any object
- You can use it directly inside `lapply()`

sapply

- The usage and logic is exactly the same as in `lapply`
- It tries to simplify the resulting list to an array
- If `sapply` can't simplify because the dimensions don't match, then it returns a list!

vapply

- `vapply()` again has a similar logic, but requires a bit more syntax:

```
vapply(X, FUN, FUN.VALUE, ..., USE.NAMES = TRUE)
```

- The `FUN.VALUE` argument expects a template for the return argument of the function `FUN`
- `vapply()` can be considered a more robust version of `sapply()`, because you explicitly restrict the output of the function you want to apply

Useful Functions and Utilities

- Everything you would need to do you will find by googling it / searching in the `StackExchange`
- There is no need to memorize any functions
- But there are certain functions you will end up memorizing because they are used very often when working with data
- Functions of the type `is.*()` and `as.*()`
- Mathematical utilities: `abs()`, `sum()`, `mean()`, etc.
- Data utilities: `seq()`, `rep()`, `sort`, `str`, `unlist`

Examples

1. Random walk simulation
   - Draw random steps directions
   - Write a function to simulate 200 steps
   - Plot the resulting series
   - Do 40 replications
2. Handling missing values
   - Write a function to substitute missing values
   - Apply this function to an array
   - What if missing values are encoded differently?
   - Many arguments and closures

Additional

- Regular expressions are sequences of characters
- You can
  - See if a pattern exists
  - Identify and extract the pattern
  - Replace the pattern
- `R` has advanced tools to deal with times and dates

References and Resources

- Intermediate R ▸ Tutorial
- Python for beginners - Control Flow ▸ Tutorial
- Control flow and repeating operations ▸ Tutorial
- Functional programming ▸ Tutorial
- How to Use If-Else Statements and Loops in R ▸ Tutorial
- Creating Functions ▸ Tutorial