Conditional Statements

Kishore Hari

2022-08-22

Conditions

Conditions are a key aspect of programming. Often while solving problems/accomplishing tasks, some steps become conditional. For example, security personnel at roller-coasters would only let in people that are above a certain height/age, so as to conduct a low-risk ride. *Try to consciously identify all conditional steps/choices that you take in your everyday life.*

Let's take the following example: - Print all the even numbers in the vector: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10].

We know that to print each number, we can iterate/loop over the vector and print the corresponding iterator. *Try to do it using the index method and the value method.* But to print only the even numbers (numbers that are divisible by 2), in each iteration, we must check if the number is an even number, and if yes, then print the number. In other words, we must apply a *condition* on each number in the vector (the condition being if the number is even), and if the condition is true, we print the number. The general syntax for writing such conditions is as follows:

```
if (condition) { # condition should evaluate as either true or false.
    # set of instructions to R if the condition is true
}
```

Two noteworthy aspects of the above syntax:

1. The *condition* between the parenthesis is an R code that evaluates as either `TRUE` or `FALSE`, the two constants in R belonging the data type `boolean`.
2. The set of instructions enclosed by the curly braces.

The if statement can be colloquially read as: *if the condition is true, then execute the code.* In most cases, these conditions are comparative statements. In our example, the condition is that a number is even. Therefore, we want to check if the remainder we get upon dividing the number by 2 is 0. We use the relational operator == to do this:

```
for (val in 1:10) {
    if ((val %% 2) == 0) {
        print(val)
    }
}
```

```
## [1] 2
## [1] 4
## [1] 6
## [1] 8
## [1] 10
```

Let's extend the problem a little more. We now want the code to print "even" for every even number and "odd" for every odd number. We can do it in two ways:

```r
# Method 1: using two if statements
for (val in 1:10) {
    if ((val %% 2) == 0) {
        print("even")
    }
    if ((val %% 2) != 0) { # note the relational operator for inequality
        print("odd")
    }
}
```

```
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
```

```r
# Method 2: using an else statement
for (val in 1:10) {
    if ((val %% 2) == 0) {
        print("even")
    }
    else {
        print("odd")
    }
}
```

```
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
```

Note the use of `else` in the second way. `else` tells R what to do if the condition given in if is `FALSE`. While both the methods above give the same answer in this case, there are subtle differences between both methods.

1. In the first case, both if statements are evaluated when the code runs, and both conditions are checked for their truth. In the second case, if the condition in the `if` statement is satisfied, R will ignore the `else` entirely.

2. The second `if` statement in first case is only executed the when the condition given in that `if` statement is `TRUE`. Note that this does not depend on the outcome of the first `if` condition. In the second case, however, `else` is only executed if the condition within the `if` statement is `FALSE`.

To understand the difference better, see the following problem: - For each number in the vector [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], print "Low" if the number is less than 5, and "High" if it is less than 10.

```
# Method 1: using two if statements
for (val in 1:10) {
    if (val < 5) { #Notice the use of the relational operator less than
        paste0(val, " is Low") # paste0 merges multiple words into a single word.
        # Think of paste0 as the version of print for multiple arguments.
    }
    if (val < 10) {
        paste0(val, " is High")
    }
}

# Method 2: using if-else

for (val in 1:10) {
    if (val < 5) { #Notice the use of the relational operator less than
        paste0(val, " is Low")
    }
    else {
        paste0(val, "is High")
    }
}
```

Notice the difference between the outputs of the two codes. In the first case, since both if statements are evaluated separately and there are some numbers that satisfy both the conditions, we see both high and low printed for those numbers. In the second case, however, only `if` or `else` can get executed for a given number. Specifically, if the condition in the `if` statement is `FALSE`, only then does the `else` statement get executed.

In some cases, you would need to check multiple conditions at a time, with each condition exclusive of the previously stated conditions. In such cases, the `if; else if; else` sequence of conditions is used. See the below example:

- For each number in the vector [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], if the number is below 5, print "Low". If the number is between 5 and 8, print "Medium". If the number is above 8, print "High".

```
for (val in 1:10) {
    if (val < 5) { #Notice the use of the relational operator less than
        paste0(val, " is Low")
    }
    else if (val < 9) {
        paste0(val, "is Medium")
    }
    else {
        paste0(val, "is High")
    }
}
```

Excercises

1. Read the tutorialspoint pages corresponding to the conditional statements: https://www.tutorialspoint.com/r/r_if_statement.htm and https://www.tutorialspoint.com/r/r_if_else_statement.htm
2. Write a function that takes a vector and prints the number of even numbers in the vector. Call the function with the following vectors.

   i. 1, 2, 3, 5, 4, 45, 90
   ii. 3, 4, 5, 34, 4545

```
evens <- function(x) {
    numEvens <- 0
    for (i in x) {
        if (i %%2 == 0) {
            numEvens <- numEvens + 1
        }
    }
    return(numEvens)
}
evens(c(1,2,3,5,4,45,90))
```

```
## [1] 3
```

```
evens(c(3,4,5,34,4545))
```

```
## [1] 2
```

```
evensSapply <- function(x) {
    evensCount <- sapply(x, function(i) {
        i %%2 == 0
    }) %>% sum
    return(evensCount)
}
evensSapply(c(1,2,3,5,4,45,90))
```

```
## [1] 3
```

```
evensSapply(c(3,4,5,34,4545))
```

```
## [1] 2
```

```
# Note that both functions do the same thing.
```

3. Write a function that takes a number, and prints the square root of the number if the number is greater than 100, square of the number if it is less than 10, the number itself otherwise. Call the function with the numbers: `5, 10, 30, 500, 1000`

```
customPrinter <- function(x) {
    if (x > 100) {
        print(sqrt(x))
    }
    else if (x < 10) {
        print(x^2)
    }
    else {
        print(x)
    }
}
sapply(c(5, 10, 30, 500, 1000), customPrinter) # Note this as a way to call the function multiple times
```

```
## [1] 25
## [1] 10
## [1] 30
## [1] 22.36068
## [1] 31.62278
```

```
## [1] 25.00000 10.00000 30.00000 22.36068 31.62278
```

4. Your friend borrowed Rs. 1,00,000 from you and wants to return the money in installments in a unique way. Starting from Sunday as 1, he will count the number corresponding to the current day of the week, (monday is 2, tuesday is 3 and so on. let this number be D) and current week (1st week is 1, 2nd is 2 and so on. let this number be W). He will then return to you an amount equal to $W * D^2$. How long till he returns all the money to you? Make sure that the last installment is adjusted such that the total does not cross 1,00,000.

```
# Using while loop.
borrowed <- 100000
returned <- 0
numDay <- 0 # This is D in the question
numWeek <- 1 # This is W in the question
while(returned < borrowed) {
    numDay <- numDay + 1
    amt <- numWeek*numDay^2
    if (returned + amt > borrowed) {
        amt <- borrowed - returned
    }
    returned <- returned + amt
    if (numDay == 7) {
        numDay <- 0
        numWeek <- numWeek + 1
    }
}
total_num_days <- (numWeek-1)*7 + numDay # Why numWeek-1?
```