Conditional Statements

Stat 133, Fall 2024

Lectures 5 & 6, 09/18/2024

1. Introduction to Conditionals in R

Conditionals statements are a critical structure in programming, as they allow for control flow based on logical conditions. In R, conditionals are used to perform various types actions depending on whether a statement evaluates as `TRUE` or `FALSE`. We will focus on the following types of conditionals:

- `if` statements
- `else` statements
- `else if` statements
- `switch` statements

2. `if`, `else if`, and `else` Statements

The simplest form of a conditional statement is the `if` statement, which executes a block of code if the condition evaluates to `TRUE`.

2.1. `if` Statements

```
if (condition) {
  # code to execute if condition is TRUE
}
```

Example:

```
x = 1
if (x > 0) {
  print("x is positive")
}
```

Output:

```
[1] "x is positive"
```

2.2. `else` and `else if` Statements

We can extend the `if` statement with `else` and `else if` to perform different actions based on multiple conditions.

```
if (condition1) {
  # code if condition1 is TRUE
} else if (condition2) {
  # code if condition2 is TRUE
} else {
  # code if all conditions are FALSE
}
```

Example:

```
x = -1
if (x > 0) {
  print("x is positive")
} else if (x == 0) {
  print("x is zero")
} else {
  print("x is negative")
}
```

Output:

```
[1] "x is negative"
```

3. The `ifelse()` Function

In addition to `if-else` structures, R provides a vectorized version called `ifelse()`, which is useful for element-wise evaluations. It takes three arguments: (1) a condition (2) a value if the condition is true and (3) a value if the condition is false.

3.1. Syntax of `ifelse()`

```
ifelse(condition, true_value, false_value)
```

Example:

```
x = c(1, -1, 0, 2)
result = ifelse(x <= 0, "non-positive", "positive")
print(result)
```

Output:

```
[1] "positive" "non-positive" "non-positive" "positive"
```

4. Nested Conditionals

We can nest conditionals to handle more complex decision-making. However, excessive nesting can lead to difficult-to-read code, so it's good practice to keep nesting at a minimum (usually there some optimizations one can make).

4.1. Example of Nested Conditionals

```
x = 1
y = 0

if (x < y) {
    print("x is less than y")
} else {
  if (x > y) {
    print("x is greater than y")
  } else {
    print("x is equal to y")
  }
}
```

Output:

```
[1] "x is greater than y"
```

5. `switch()` Statements

`switch()` is an alternative to `if-else` chains when you have multiple distinct cases. It evaluates an expression and selects an outcome based on the provided cases.

5.1. Syntax of `switch()`

```
switch(expression, case1, case2, case3, ...)
```

Example:

```
option = "D"
result = switch(option,
                "A" = "Option A selected",
                "B" = "Option B selected",
                "C" = "Option C selected",
                "Invalid option")
print(result)
```

Output:

```
[1] "Invalid option"
```

6. Logical Operators in Conditionals

Conditionals often rely on logical operators to combine or invert conditions. Common logical operators in R include:

- `&&`, `||`, `!` (for element-wise AND, OR, and negation respectively)
- `==`, `!=`, `<`, `>` (for testing equality or inequality of two logical expressions respectively)

We can use any permutation of `<` or `>` for logical expressions/conditional statements.

6.1. Example with Logical Operators

```
x = 2
y = 0
z = 4

if ((x > y && x != z) || (y <= z && y > x + 1 )) {
  print("TRUE")
} else { print("FALSE") }
```

Output:

```
[1] "TRUE"
```

7. Summary

Conditional statements such as `if`, `else`, `else if`, and `switch` are essential in control flow, and used extensively in computational processing. We should make use of vectorized operations whenever possible, as it is generally more efficient. For example, we can use the `ifelse()` function to do so. Logical operators further enhance the flexibility of conditional checks, allowing for more complex conditions. Later, we will combine these ideas (logical expressions and conditional statements) to perform even more interesting computations; particularly when iterating over certain types of structures in R.