MTH 2520 R Notes 8

8 R Programming Structures

8.1 `if`, `else`, and `ifelse()` Statements

8.1.1 Conditional Execution Using `if` and `else`

- If we want R to execute a statement only if a certain condition is met, we can use:

```
if          # Used to execute a statement only if a specified con-
            # dition is met
else        # Used with if to specify an alternative statement to
            # be executed if the condition isn't met
```

- Here's a simple example of an `if` statement:

```
> y <- 5

> if (y > 0) {
+   x <- 1
+ }

> x

[1] 1
```

- The general form of an `if` statement is:

```
> if (cond) {
    statement1
    statement2
        .
        .
        .
    statementq
  }
```

where `cond` is a `"logical"` expression (i.e. it evaluates to `TRUE` or `FALSE`), and `statements 1` through `q` are only executed if `cond` is `TRUE`. If `cond` is `FALSE`, nothing happens.

- If there's only one `statement`, we don't need to use the curly brackets `{ }` as long as the entire `if` statement is one line, like this:

```
> if (cond) statement1
```

- Here's an example of an `if-else` statement:

```
> y <- -6

> if (y > 0) {
+   x <- 1
+ } else {
+   x <- 2
+ }

> x

[1] 2
```

- The general form of an `if-else` statement is:

```
> if (cond) {
    statement1
          .
          .
          .
    statementp
  } else {
    statement1
          .
          .
          .
    statementq
  }
```

  If `cond` is `TRUE`, the first set of `statements` (`1` through `p`) are executed. If it's `FALSE`, the second set (`1` through `q`) are executed.

  `else` has to be on the same line as the first closing curly bracket `}`. Otherwise, R will think it's just an `if` statement without an `else`.

- If there's only one `statement` in each set of `statements`, we can leave out the curly brackets `{ }` as long as the entire `if-else` command is on one line, like this:

```
> if (cond) statement1 else statement2
```

- We can assign values to variables using `if` and `if-else` statements. For example, typing:

```
> x <- if (y > 0) 1 else 2
```

does the same thing as:

```
> if (y > 0) x <- 1 else x <- 2
```

- In the absence of `else`, the value `NULL` is assigned if the condition is `FALSE`. So

  ```
  x <- if (y > 0) 1
  ```

  is equivalent to

  ```
  x <- if (y > 0) 1 else NULL
  ```

8.1.2 Vectorized if-else: The `ifelse()` Function

- The `if-else` statements that use `if` and `else` aren't *vectorized*. More specifically, if `cond` is a `"logical"` *vector*, only the first element is used.
- The `ifelse()` function is a vectorized version of the `if-else` statement:

  ```
  ifelse()          # Takes a "logical" vector, and returns a vector of
                    # equal length with values depending on whether the
                    # corresponding "logical" value is TRUE or FALSE.
  ```

- `ifelse()` takes arguments `test`, a `"logical"` vector (usually expressed as a condition to be met), `yes`, the return value when `test` is `TRUE`, and `no`, the return value when `test` is `FALSE`.
- Here's a simple example:

  ```
  > ifelse(test = c(FALSE, TRUE, FALSE), yes = "a", no = "b")

  [1] "b" "a" "b"
  ```

  Above, the returned vector contains elements `"a"` (the value of `yes`) or `"b"` (the value of `no`) depending on whether the corresponding element of `test` is `TRUE` or `FALSE`.

- Here we classify peoples' heights as `"short"` or `"tall"`:

  ```
  > height <- c(69, 71, 67, 66, 72, 71, 61, 65, 73, 70, 68, 74)

  > ifelse(height > 69, yes = "tall", no = "short")

   [1] "short" "tall"  "short" "short" "tall"  "tall"  "short" "short" "tall"  "tall"  "short"
  ```

Section 8.1 Exercises

Exercise 1 The function `print()` will print a value in the R console. Guess what will be printed in each of the following sets of commands, then check your answers.

a)
```
> y <- 1
> if (y > 0) print("hello")
```

b)
```
> y <- -1
> if (y > 0) print("hello")
```

c)
```
> y <- -1
> if (y > 0) print("hello") else print("goodbye")
```

Exercise 2 Guess what the value of `z` will be after each of the following sets of commands, then check your answers.

a)
```
> x <- 3
> if (x > 2) {
    y <- 2 * x
    z <- 3 * y
  }
```

b)
```
> x <- 1
> if (x > 2) {
    y <- 2 * x
    z <- 3 * y
  }
```

c)
```
> x <- 1
> if (x > 2) {
    y <- 2 * x
    z <- 3 * y
  } else {
    y <- 2 / x
    z <- 3 / y
  }
```

Exercise 3 Guess what the value of `x` will be after each of the following sets of commands, then check your answers.

a)
```
> y <- 6
> x <- if (y > 0) 1 else 2
```

b)
```
> y <- -6
> x <- if (y > 0) 1 else 2
```

c)
```
> y <- 6
> x <- if (y > 0) 1
```

d)
```
> y <- -6
> x <- if (y > 0) 1
```

Exercise 4 The `if-else` statements that use `if` and `else` aren't *vectorized*. More specifically, if the condition, `cond`, is a `"logical"` *vector*, only the first element is used. Guess what `x` will be after the following command, then check your answer.

```
> x <- if (c(FALSE, TRUE, TRUE)) 5 else 10
```

Exercise 5 Statements that use the `ifelse()` function *are* vectorized. Guess what `x` will be after the following command, then check your answer.

```
> x <- ifelse(c(FALSE, TRUE, TRUE), yes = 5, no = 10)
```

Exercise 6 The `if-else` statements that use `if` and `else` aren't *vectorized*, but ones that use the `ifelse()` function are. Here are two vectors `x` and `y`:

```
> x <- c(1, 2, 3)
> y <- c(0, 2, 4)
```

Guess what `z` will be after each of the following commands, then check your answers.

a)
```
> z <- if (x == y) "Equal" else "Not Equal"
```

b)
```
> z <- ifelse(test = x == y, yes = "Equal", no = "Not Equal")
```

Exercise 7 Here's a `"character"` vector containing responses to a survey question:

```
> response <- c("agree", "agree", "disagree", "agree", "disagree",
                "disagree", "disagree", "disagree", "agree", "disagree")
```

Write a command (or commands) involving `ifelse()` that converts `response` to a vector of `1`'s and `0`'s depending on whether the `response` is `"agree"` or `"disagree"`. You should end up with this:

```
 [1] 1 1 0 1 0 0 0 0 1 0
```

8.2 The Logical Operations "And", "Or", and "Not"

8.2.1 Logical Operations and Compound Logical Expressions

- ***Logical operators*** (or ***Boolean operators***) correspond to "and", "or", and "not", and are written in R as:

```
!               # "Not"
&               # "And"
|               # "Or"
```

- These operate on `"logical"` (`TRUE` or `FALSE`) expressions and return `TRUE` or `FALSE` values. They're listed above in order of operator precedence (highest to lowest).

8.2.2 Logical Operations on Scalar Logical Expressions

- `&` returns `TRUE` if both expressions are `TRUE`, and it returns `FALSE` if at least one expression is `FALSE`:

```
> TRUE & TRUE

[1] TRUE

> TRUE & FALSE

[1] FALSE
```

- `|` returns `TRUE` if at least one of the expressions is `TRUE`, and it returns `FALSE` if both expressions are `FALSE`:

```
> FALSE | TRUE

[1] TRUE

> FALSE | FALSE

[1] FALSE
```

- The negation operator, `!`, returns the "opposite" of a logical expression:

```
> !TRUE

[1] FALSE

> !FALSE

[1] TRUE
```

- As an example, to test whether a variable `x` lies *between* two numbers (60 and 70), we type:

```
> x <- 63
> x > 60 & x < 70
```

```
[1] TRUE
```

and to test whether it lies *outside* the range (60 to 70), we type:

```
> x < 60 | x > 70
```

```
[1] FALSE
```

- Here's an example using `&` in an `if-else` statement:

```
> x <- 3
> y <- 5

> if (x < 10 & y < 10) {
+   print("Both less than 10")
+ } else {
+   print("Not both less than 10")
+ }

[1] "Both less than 10"
```

- The logical operators `&`, `|`, and `!` operate *elementwise* on `"logical"` vectors. For example:

```
> c(TRUE, FALSE, TRUE) & c(TRUE, TRUE, FALSE)

[1]  TRUE FALSE FALSE
```

- As another example, here are two vectors, `Syst` and `Diast`, containing systolic and diastolic blood pressures for five people:

```
> Syst

[1] 110 119 111 113 128

> Diast

[1] 70 74 88 74 83
```

A blood pressure considered normal if the systolic level is less than 120 *and* the diastolic level is less than 80. To identify the people with normal blood pressures, we can type:

```
> Syst < 120 & Diast < 80

[1]  TRUE  TRUE FALSE  TRUE FALSE
```

or we can use `which()`:

```
> which(Syst < 120 & Diast < 80)
```

```
[1] 1 2 4
```

- In the next example, we use `&` in square brackets `[ ]` to extract rows from a data frame `bp` containing the blood pressures from above:

```
> bp

  Name Syst Diast
1  Joe  110    70
2 Katy  119    74
3 Bill  111    88
4  Kim  113    74
5 Mark  128    83
```

To extract the rows corresponding to people whose blood pressures are normal (systolic less than 120 and diastolic less than 80), we type:

```
> bp[bp$Syst < 120 & bp$Diast < 80, ]

  Name Syst Diast
1  Joe  110    70
2 Katy  119    74
4  Kim  113    74
```

- `&`, `|`, and `!` are useful in `ifelse()` statements. (Recall that `ifelse()` operates elementwise on vectors.). For example, consider again the blood pressure vectors (from above):

```
> Syst

[1] 110 119 111 113 128

> Diast

[1] 70 74 88 74 83
```

We can use `ifelse()` to create a `"character"` vector `status` that indicates whether a person's blood pressure is "Normal" (systolic less than 120 and diastolic less than 80) or "Abnormal":

```
> status <- ifelse(test = Syst < 120 & Diast < 80,
+                  yes = "Normal",
+                  no = "Abnormal")

> status

[1] "Normal"   "Normal"   "Abnormal" "Normal"   "Abnormal"
```

- Pay attention to the operator precedence for `&`, `|`, and `!`. It can be found by typing:

```
> ? Syntax
```

Parentheses can be used to control the order of operations.

Section 8.2 Exercises

Exercise 8 Here are two variables `x` and `y`:

```
> x <- 4
> y <- 7
```

Guess what the result of each of the following will be, then check your answers.

a) `> x > 2 & y == 7`

b) `> x < 0 | y == 7`

c) `> !(x < 0)`

Exercise 9 Recall that the operator precedence of the logical operators, from highest to lowest, is `!`, `&`, and `|`. The order of operations can be controlled using parentheses. Guess what the result of each of the following commands will be, then check your answers.

a) `> 10 < 20 & 15 < 16 | 9 == 10`

b) `> (10 < 20 & 15 < 16) | 9 == 10`

c) `> 4 < 3 & (5 < 6 | 8 < 9)`

d) `> (4 < 3 & 5 < 6) | 8 < 9`

Exercise 10 Guess what the result of each of the following will be, then check your answers.

a) `> c(FALSE, TRUE, FALSE) & c(TRUE, TRUE, FALSE)`

b) `> c(FALSE, TRUE, FALSE) | c(TRUE, TRUE, FALSE)`

c) `> !c(FALSE, TRUE, FALSE)`

Exercise 11 Recall that `is.na()` will identify missing values (`NA`s) in a vector. Here's a vector `x`:

```
> x <- c(1, 2, NA, 6, 3, NA, 5)
```

(See what happens when you type `is.na(x)`).

a) Guess what the following command will return, then check your answer:

```
> !is.na(x)
```

b) Write a command (or commands) involving `is.na()`, the operator `!`, and square brackets `[ ]` that extracts all the non-missing values from `x`.

Exercise 12 Here's a vector `x`:

```
> x <- c(1, 2, 6, 3, 5)
```

Guess what the result of each of the following commands will be, then check your answers.

a) `> !(x > 4)`

b) `> x > 4 & x < 6`

c) `> !(x > 4 & x < 6)`

d) `> x > 4 | x < 6`

Exercise 13 Here are two vectors, `Gender` and `Age`:

```
> Gender <- c("m", "f", "m", "f", "f")
> Age <- c(27, 34, 55, 21, 43)
```

Guess what the following command will return, then check your answers.

a) `> Gender == "m" & Age > 40`

b) `> Gender == "m" | Age > 40`

Exercise 14 Here's a data frame `x`:

```
> x <- data.frame(Gender = c("m", "f", "m", "f", "f"),
                  Age = c(27, 34, 55, 21, 43),
                  Height = c(60, 58, 65, 55, 59),
                  Weight = c(160, 129, 174, 170, 130))
> x
```

```
  Gender Age Height Weight
1      m  27     60    160
2      f  34     58    129
3      m  55     65    174
4      f  21     55    170
5      f  43     59    130
```

a) Write a command involving one or more of the logical operators (`&`, `|`, and `!`) and square brackets `[ ]` that extracts the rows of `x` corresponding to males who are over the age of 40. You should end up with this:

```
  Gender Age Height Weight
3      m  55     65    174
```

b) Write a command involving one or more of the logical operators (`&`, `|`, and `!`) and square brackets `[ ]` that extracts the rows of `x` corresponding to people who are *either* male *or* over the age of 40. You should end up with this:

```
  Gender Age Height Weight
1      m  27     60    160
3      m  55     65    174
5      f  43     59    130
```