Control Flow: Logicals & if/then/else

Hopefully, you feel like you have an idea about programming in R. We've learned about the idea of R objects and how to investigate an R object via

- `class()`
- `typeof()`
- `str()`

We see that R commonly uses the attributes of the object to determine how to apply (or dispatch) a function. We've looked at common R objects and how to access their elements:

- Atomic vectors - `x[ ]`
- Matrices - `x[ , ]`
- Data Frames - `x[ , ]` or `x$name`
- Lists - `x[[ ]]` or `x$name`

and how to document our code by using a notebook environment such as Quarto (along with git/github for version control and sharing/collaborating).

Now we want to look at how to control the execution of our code. The three main things we'll look at here are

- `if/then/else` logic and syntax
- looping to repeatedly execute code
- vectorized functions for improved efficiency

Then we'll see how to write our own functions! We are already learning a lot of necessary material to be a data scientist. Let's go!!

Logical Statements

A logical statement is a comparison of two quantities. It will resolve as `TRUE` or `FALSE` (note the all caps).

- To compare to things in R, we can use standard operators
    - `==` equality check (although this isn't always the best choice!)
    - `!=` not equal to
    - `>=`, `>`, `<`, `<=` operators

```
#Strings must be exactly the same to be equivalent
"hi" == "hi"
```

```
[1] TRUE
```

```
"hi" == " hi"
```

```
[1] FALSE
```

```
4 >= 1
```

```
[1] TRUE
```

```
4 != 1
```

```
[1] TRUE
```

```
sqrt(3)^2  == 3
```

```
[1] FALSE
```

That last one should be true! The issue is the loss of precision with taking the square root of 3. Instead of using `==` we can use the `near()` function from the `dplyr` package. To call a function directly from a package we can use `::`

```
dplyr::near(sqrt(3)^2, 3)
```

```
[1] TRUE
```

That's more like it!

`is.` Family

In addition to the standard operators, R has a family of `is.` (read as "is dot") functions. These allow you to check a lot of things about an R object or value!

```
is.numeric("Word")
```

```
[1] FALSE
```

```
is.numeric(c(10, 12, 20))
```

```
[1] TRUE
```

```
is.character(c(10, 12, 20))
```

```
[1] FALSE
```

```
is.character(c("10", "12"))
```

```
[1] TRUE
```

```
is.na(c(1:2, NA, 3))
```

```
[1] FALSE FALSE  TRUE FALSE
```

This last one is important!

- First, note that R applies the `is.na()` function element-wise to the vector. This is *not* common behavior.
- Second, `NA` is the missing value indicator in R. When we start to read in data we need to check for missing values. More on that later.
- `NA` differs from `NULL` which is the undefined value in R

Logical Statements for Subsetting Data

Creating logical statements can be useful for subsetting data. We'll see how to do this in a streamlined fashion later, but for now, let's use `baseR` functionality to do some subsetting.

Recall the `iris` data set. This has measurements on flowers.

```
head(iris)
```

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

As R does comparisons element-wise, we can compare the `Species` column to a value. This returns a vector of `TRUE FALSE` values (a logical vector).

```
iris$Species == 'setosa'
```

```
  [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
TRUE
 [13]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
TRUE
 [25]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
TRUE
 [37]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
TRUE
 [49]  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE
 [61] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE
 [73] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE
 [85] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE
 [97] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE
[109] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE
[121] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE
[133] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE
```

```
[145] FALSE FALSE FALSE FALSE FALSE FALSE
```

If we index an object with a logical vector, it returns the values where a `TRUE` occurred and doesn't return values where a `FALSE` occurred!

```
iris[iris$Species == "setosa", ]
```

```
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1           5.1         3.5          1.4         0.2  setosa
2           4.9         3.0          1.4         0.2  setosa
3           4.7         3.2          1.3         0.2  setosa
4           4.6         3.1          1.5         0.2  setosa
5           5.0         3.6          1.4         0.2  setosa
6           5.4         3.9          1.7         0.4  setosa
7           4.6         3.4          1.4         0.3  setosa
8           5.0         3.4          1.5         0.2  setosa
9           4.4         2.9          1.4         0.2  setosa
10          4.9         3.1          1.5         0.1  setosa
11          5.4         3.7          1.5         0.2  setosa
12          4.8         3.4          1.6         0.2  setosa
13          4.8         3.0          1.4         0.1  setosa
14          4.3         3.0          1.1         0.1  setosa
15          5.8         4.0          1.2         0.2  setosa
16          5.7         4.4          1.5         0.4  setosa
17          5.4         3.9          1.3         0.4  setosa
18          5.1         3.5          1.4         0.3  setosa
19          5.7         3.8          1.7         0.3  setosa
20          5.1         3.8          1.5         0.3  setosa
21          5.4         3.4          1.7         0.2  setosa
22          5.1         3.7          1.5         0.4  setosa
23          4.6         3.6          1.0         0.2  setosa
24          5.1         3.3          1.7         0.5  setosa
25          4.8         3.4          1.9         0.2  setosa
26          5.0         3.0          1.6         0.2  setosa
27          5.0         3.4          1.6         0.4  setosa
28          5.2         3.5          1.5         0.2  setosa
29          5.2         3.4          1.4         0.2  setosa
30          4.7         3.2          1.6         0.2  setosa
31          4.8         3.1          1.6         0.2  setosa
32          5.4         3.4          1.5         0.4  setosa
33          5.2         4.1          1.5         0.1  setosa
34          5.5         4.2          1.4         0.2  setosa
35          4.9         3.1          1.5         0.2  setosa
36          5.0         3.2          1.2         0.2  setosa
37          5.5         3.5          1.3         0.2  setosa
38          4.9         3.6          1.4         0.1  setosa
39          4.4         3.0          1.3         0.2  setosa
40          5.1         3.4          1.5         0.2  setosa
41          5.0         3.5          1.3         0.3  setosa
42          4.5         2.3          1.3         0.3  setosa
43          4.4         3.2          1.3         0.2  setosa
44          5.0         3.5          1.6         0.6  setosa
45          5.1         3.8          1.9         0.4  setosa
46          4.8         3.0          1.4         0.3  setosa
```

```
47          5.1         3.8          1.6         0.2  setosa
48          4.6         3.2          1.4         0.2  setosa
49          5.3         3.7          1.5         0.2  setosa
50          5.0         3.3          1.4         0.2  setosa
```

Compound Logical Statements

Of course there are times we want to check whether two conditions are both TRUE or at least one of the conditions is TRUE. The **Logical Operators** below help us with that:

- `&` 'and'
- `|` 'or'

Operator	A,B true	A true, B false	A,B false
`&`	`A & B = TRUE`	`A & B = FALSE`	`A & B = FALSE`
`|`	`A | B = TRUE`	`A | B = TRUE`	`A | B = FALSE`

Note! `&&` and `||` are alternatives that look at only first comparison when given a vector of comparisons. This is used a lot in writing functions but is generally not what you want to use.

In subsetting data, this let's us do a lot more!

- Only pull out large petal setosa flowers

```
(iris$Petal.Length > 1.5) & (iris$Species == "setosa")
```

```
  [1] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
TRUE
 [13] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE
TRUE
 [25]  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE
FALSE
 [37] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE  TRUE
FALSE
 [49] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE
 [61] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE
 [73] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE
 [85] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE
 [97] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE
[109] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE
[121] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE
[133] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```
FALSE
[145] FALSE FALSE FALSE FALSE FALSE FALSE
```

```
iris[(iris$Petal.Length > 1.5) & (iris$Species == "setosa"), ]
```

```
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
6           5.4         3.9          1.7         0.4  setosa
12          4.8         3.4          1.6         0.2  setosa
19          5.7         3.8          1.7         0.3  setosa
21          5.4         3.4          1.7         0.2  setosa
24          5.1         3.3          1.7         0.5  setosa
25          4.8         3.4          1.9         0.2  setosa
26          5.0         3.0          1.6         0.2  setosa
27          5.0         3.4          1.6         0.4  setosa
30          4.7         3.2          1.6         0.2  setosa
31          4.8         3.1          1.6         0.2  setosa
44          5.0         3.5          1.6         0.6  setosa
45          5.1         3.8          1.9         0.4  setosa
47          5.1         3.8          1.6         0.2  setosa
```

The parentheses are not required but are useful to keep things straight. For example, we might want only long petal or skinny petal, setosa flowers.

```
iris[((iris$Petal.Length > 1.5) | (iris$Petal.Width < 0.4)) & (iris
```

```
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1           5.1         3.5          1.4         0.2  setosa
2           4.9         3.0          1.4         0.2  setosa
3           4.7         3.2          1.3         0.2  setosa
4           4.6         3.1          1.5         0.2  setosa
5           5.0         3.6          1.4         0.2  setosa
6           5.4         3.9          1.7         0.4  setosa
7           4.6         3.4          1.4         0.3  setosa
8           5.0         3.4          1.5         0.2  setosa
9           4.4         2.9          1.4         0.2  setosa
10          4.9         3.1          1.5         0.1  setosa
11          5.4         3.7          1.5         0.2  setosa
12          4.8         3.4          1.6         0.2  setosa
13          4.8         3.0          1.4         0.1  setosa
14          4.3         3.0          1.1         0.1  setosa
15          5.8         4.0          1.2         0.2  setosa
18          5.1         3.5          1.4         0.3  setosa
19          5.7         3.8          1.7         0.3  setosa
20          5.1         3.8          1.5         0.3  setosa
21          5.4         3.4          1.7         0.2  setosa
23          4.6         3.6          1.0         0.2  setosa
24          5.1         3.3          1.7         0.5  setosa
25          4.8         3.4          1.9         0.2  setosa
26          5.0         3.0          1.6         0.2  setosa
27          5.0         3.4          1.6         0.4  setosa
28          5.2         3.5          1.5         0.2  setosa
29          5.2         3.4          1.4         0.2  setosa
30          4.7         3.2          1.6         0.2  setosa
31          4.8         3.1          1.6         0.2  setosa
```

33	5.2	4.1	1.5	0.1	setosa
34	5.5	4.2	1.4	0.2	setosa
35	4.9	3.1	1.5	0.2	setosa
36	5.0	3.2	1.2	0.2	setosa
37	5.5	3.5	1.3	0.2	setosa
38	4.9	3.6	1.4	0.1	setosa
39	4.4	3.0	1.3	0.2	setosa
40	5.1	3.4	1.5	0.2	setosa
41	5.0	3.5	1.3	0.3	setosa
42	4.5	2.3	1.3	0.3	setosa
43	4.4	3.2	1.3	0.2	setosa
44	5.0	3.5	1.6	0.6	setosa
45	5.1	3.8	1.9	0.4	setosa
46	4.8	3.0	1.4	0.3	setosa
47	5.1	3.8	1.6	0.2	setosa
48	4.6	3.2	1.4	0.2	setosa
49	5.3	3.7	1.5	0.2	setosa
50	5.0	3.3	1.4	0.2	setosa

Quick Aside on Implicit Coercion

R attempts to coerce data into usable form when necessary. Unfortunately, it doesn't always let us know it is doing so. This means we need to be careful and understand how R works.

Recall the behavior of combining elements together into an atomic vector. R coerces to the more flexible data type.

```
#coerce numeric to string
c("hi", 10)
```

```
[1] "hi" "10"
```

In this way, R will treat `TRUE` as a 1 and `FALSE` as a 0 when math is done.

```
#coerce TRUE/FALSE to numeric
c(TRUE, FALSE) + 0
```

```
[1] 1 0
```

```
c(TRUE, FALSE) + 10
```

```
[1] 11 10
```

```
as.numeric(c(TRUE, FALSE, TRUE))
```

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

```
mean(c(TRUE, FALSE, TRUE))
```

```
[1] 0.6666667
```

The order of coercion (from least flexible to most)

- logical
- integer
- double
- character.

Conditional Execution via `if/then/else`

We often want to execute statements conditionally. For instance, we might want to create a variable that takes on different values depending on whether or not some condition is met.

- `if then else` syntax

```
if (condition) {
  then execute code
}

#if then else
if (condition) {
  execute this code
} else {
  execute this code
}
```

```
#Or more if statements
if (condition) {
  execute this code
} else if (condition2) {
  execute this code
} else if (condition3) {
  execute this code
} else {
  #if no conditions met
  execute this code
}
```

Note! You should keep the `{` on the lines as you see here. There are some occassions where something like this would work:

```
if (condition)
  {
  execute this code
} else
  {
  execute this code
}
```

but it generally won't! So just mind the positioning.

As an example of using `if/then/else` consider the built-in data set `airquality`. This data has daily air quality measurements in New York from May (Day 1) to September (Day 153) in 1973.

```
head(airquality)
```

```
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
5    NA      NA 14.3   56     5   5
6    28      NA 14.9   66     5   6
```

We may want to code a wind category variable as follows:

- high wind days (wind $\geq$ 15mph)
- windy days (10mph $\leq$ wind < 15mph)
- lightwind days (6mph $\leq$ wind < 10mph)
- calm days (wind $\leq$ 6mph)

For a given value we can do a check and assign a new value.

```
airquality$Wind[1]
```

```
[1] 7.4
```

```
if(airquality$Wind[1] >= 15) {
  "High Wind"
} else if (airquality$Wind[1] >= 10){
  "Windy"
} else if (airquality$Wind[1] >= 6) {
  "Light Wind"
} else if (airquality$Wind[1] >= 0) {
  "Calm"
} else {
  "Error"
}
```

```
[1] "Light Wind"
```

Unfortunately, to apply this to each observation requires a loop or the use of a vectorized function. We'll cover those shortly!

Quick R video

Please pop this video out and watch it in the full panopto player!

11 - if/then/else

[Auto-generated transcript. Edits may have been applied for clarity.]

Recap

- Logical comparisons resolve as `TRUE` or `FALSE`
- Compound logical operators are `&` (and) and `|` (or)
- `if/then/else` logic to conditionally execute code