4. Vector

Solutions to Swirl's R Programming Exercises

06-19-2022

Acknowledgements: R Language Concepts and code questions (with minor modifications) are used here from the swirl package. https://www.r-project.org/nosvn/pandoc/swirl.html

Vectors come in two different flavors: atomic vectors and lists. An atomic vector contains exactly one data type, whereas a list may contain multiple data types. We'll explore atomic vectors further before we get to lists.

In previous lessons, we dealt entirely with numeric vectors, which are one type of atomic vector. Other types of atomic vectors include logical, character, integer, and complex. In this lesson, we'll take a closer look at logical and character vectors.

Logical vectors can contain the values TRUE, FALSE, and NA (for 'not available').These values are generated as the result of logical 'conditions'. Let's experiment with some simple conditions.

First, create a numeric vector num_vect that contains the values 0.5, 55, -10, and 6.

```
num_vect <- c(0.5, 55, -10, 6)
```

Now, create a variable called tf that gets the result of num_vect < 1, which is read as 'num_vect is less than 1'.

Remember our lesson on vector arithmetic? The theme was that R performs many operations on an element-by-element basis. We called these 'vectorized' operations.

What do you think tf will look like?

1: a single logical value

2: a vector of 4 logical values

```
tf <- num_vect<1
tf
```

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

The statement num_vect < 1 is a condition and tf tells us whether each corresponding element of our numeric vector num_vect satisfies this condition. The first element of num_vect is 0.5, which is less than 1 and therefore the statement 0.5 < 1 is TRUE. The second element of num_vect is 55, which is greater than 1, so the statement 55 < 1 is FALSE. The same logic applies for the third and fourth elements.

Let's try another. Type num_vect >= 6 without assigning the result to a new variable.

```
num_vect>=6
```

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

====================

This time, we are asking whether each individual element of num_vect is greater than OR equal to 6. Since only 55 and 6 are greater than or equal to 6, the second and fourth elements of the result are TRUE and the first and third elements are FALSE.

The `<` and `>=` symbols in these examples are called 'logical operators'. Other logical operators include `>`, `<=`, `==` for exact equality, and `!=` for inequality.

If we have two logical expressions, A and B, we can ask whether at least one is TRUE with A | B (logical 'or' a.k.a. 'union') or whether they are both TRUE with A & B (logical 'and' a.k.a. 'intersection'). Lastly, !A is the negation of A and is TRUE when A is FALSE and vice versa.

It's a good idea to spend some time playing around with various combinations of these logical operators until you get comfortable with their use. We'll do a few examples here to get you started.

Try your best to predict the result of each of the following statements. You can use pencil and paper to work them out if it's helpful. If you get stuck, just guess and you've got a 50% chance of getting the right answer!

(3 > 5) & (4 == 4)

1: TRUE

2: FALSE

Selection: 2

(TRUE == TRUE) | (TRUE == FALSE)

Break this problem down into 2 pieces. The `|` in the middle states that AT LEAST ONE of the pieces is TRUE. Your job is to figure out if that is an accurate statement. If so, the entire condition is TRUE. If not, it's FALSE.

1: TRUE

2: FALSE

Selection: 1

((111 >= 111) | !(TRUE)) & ((4 + 1) == 5)

1: TRUE

2: FALSE

Selection: 1

Don't worry if you found these to be tricky. They're supposed to be. Working with logical statements in R takes practice, but your efforts will be rewarded in future lessons (e.g. subsetting and control structures).

Character vectors are also very common in R. Double quotes are used to distinguish character objects, as in the following example.

Create a character vector that contains the following words: "My", "name", "is". Remember to enclose each word in its own set of double quotes, so that R knows they are character strings. Store the vector in a variable called my_char.

```r
my_char <- c("My","name", "is")
```

Print the contents of my_char to see what it looks like.

```r
my_char
```

```
## [1] "My"   "name" "is"
```

Right now, my_char is a character vector of length 3. Let's say we want to join the elements of my_char together into one continuous character string (i.e. a character vector of length 1). We can do this using the paste() function.

Type paste(my_char, collapse = " ") now. Make sure there's a space between the double quotes in the `collapse` argument. You'll see why in a second.

```
paste(my_char, collapse = " ")
```

```
## [1] "My name is"
```

The `collapse` argument to the paste() function tells R that when we join together the elements of the my_char character vector, we'd like to separate them with single spaces.

It seems that we're missing something.... Ah, yes! Your name!

To add (or 'concatenate') your name to the end of my_char, use the c() function like this: c(my_char, "your_name_here"). Place your name in double quotes where I've put "your_name_here". Try it now, storing the result in a new variable called my_name.

```
my_name <- c(my_char, "Pradip")
my_name
```

```
## [1] "My"     "name"   "is"     "Pradip"
```

Now, use the paste() function once more to join the words in my_name together into a single character string. Don't forget to say collapse = " "!

```
paste(my_name, collapse= " ")
```

```
## [1] "My name is Pradip"
```

In this example, we used the paste() function to collapse the elements of a single character vector. paste() can also be used to join the elements of multiple character vectors.

In the simplest case, we can join two character vectors that are each of length 1 (i.e. join two words). Try paste("Hello", "world!", sep = " "), where the `sep` argument tells R that we want to separate the joined elements with a single space.

```
paste("Hello", "world!", sep = " ")
```

```
## [1] "Hello world!"
```

For a slightly more complicated example, we can join two vectors, each of length 3. Use paste() to join the integer vector 1:3 with the character vector c("X", "Y", "Z"). This time, use sep = " " to leave no space between the joined elements.

```
paste(c(1:3),c("X", "Y", "Z"), sep="")
```

```
## [1] "1X" "2Y" "3Z"
```

What do you think will happen if our vectors are of different length? (Hint: we talked about this in a previous lesson.)

Vector recycling! Try paste(LETTERS, 1:4, sep = “-”), where LETTERS is a predefined variable in R containing a character vector of all 26 letters in the English alphabet.

```
paste(LETTERS, 1:4, sep = "-")
```

```
##  [1] "A-1" "B-2" "C-3" "D-4" "E-1" "F-2" "G-3" "H-4" "I-1" "J-2" "K-3" "L-4"
## [13] "M-1" "N-2" "O-3" "P-4" "Q-1" "R-2" "S-3" "T-4" "U-1" "V-2" "W-3" "X-4"
## [25] "Y-1" "Z-2"
```

Since the character vector LETTERS is longer than the numeric vector 1:4, R simply recycles, or repeats, 1:4 until it matches the length of LETTERS.

Also worth noting is that the numeric vector 1:4 gets ‘coerced’ into a character vector by the paste() function.

We’ll discuss coercion in another lesson, but all it really means is that the numbers 1, 2, 3, and 4 in the output above are no longer numbers to R, but rather characters “1”, “2”, “3”, and “4”.