Vectors

The simplest and most common data structure in R is the vector.

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 will explore atomic vectors here before we get to lists in another lesson.

Today, you will learn how to create different kinds of vectors (numeric, logical, and character) and how to combine them into new vectors.

In previous lessons, we dealt entirely with numeric vectors, which are one type of atomic vector.

Remember that we created a vector from 1 to 20 using 1:20? Try that again.

```
1:20
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

And, that we can assign this vector to a variable, e.g., 'x'? Try that! Assign 1:20 to the variable 'x'.

```
x <- 1:20
```

There are a number of ways that we can examine our new vector.

We could just display the vector on the screen, by typing the name: 'x'. Try that.

```
x
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

But, what if we had a really, really, really *long* vector that was much too large to display on the screen? We can use several functions to look at what 'x' contains. First, we can ask if 'x' even is a vector, with the function is.vector(). Try that…

```
is.vector(x)
```

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

The response to this question function is 'TRUE', i.e., x is a vector. The alternative response is 'FALSE'. We will look at these logical statements in more detail later in the course.

Second, we can also easily examine 'x' with str(), or structure, function. Type str(x).

```
str(x)
```

```
##  int [1:20] 1 2 3 4 5 6 7 8 9 10 ...
```

This function displays a summary of the various parts of the object. In this case, we only have one part, the vector of integers. The str() function is more useful for objects with different parts, such as lists and the output of statistical tests. We will use str() more in later lessons.

OK, vectors of integers are straightforward. Hopefully you understand these now, and can make and examine them. Remember that as well as ':', you can also use seq() and rep() as we did in the last lesson.

However, we may want to work with data other than integers. Other types of atomic vectors include logical, character, integer, and complex. Now, 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'.

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

What do you think tf will look like?

1. a vector of 4 logical values
2. a single logical value

a vector of 4 logical values

Print the contents of tf now.

```
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 (i.e., not A) and is TRUE when A is FALSE and vice versa.

Do not worry if these ideas do not make sense right now. We will come back to logical expressions in a later class. On to character vectors!

Character vectors are 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.

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

BEST PRACTICE: Is to use double quotes. Single quotes (a.k.a. the apostrophe) will usually work but there are cases where using them will cause trouble. Double quotes are best for text.

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

```
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, "Swirl")
```

Take a look at the contents of my_name.

```
my_name
```

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

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 Swirl"
```

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., to 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(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"
## [12] "L-4" "M-1" "N-2" "O-3" "P-4" "Q-1" "R-2" "S-3" "T-4" "U-1" "V-2"
## [23] "W-3" "X-4" "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”.

Congratulation on completing another lesson! Now you know how to create numeric, integer, logical, and character vectors, examine their structure, and combine them together in different ways. Nice work!