Vectors and lists

In programming, a **data structure** is a format for organizing and storing data. Data structures are important to understand because you will work with them frequently when you use R for data analysis. The most common data structures in the R programming language include:

- Vectors
- Data frames
- Matrices
- Arrays

Think of a data structure like a house that contains your data.

This reading will focus on vectors. Later on, you'll learn more about data frames, matrices, and arrays. There are two types of vectors: **atomic vectors** and **lists**. Coming up, you'll learn about the basic properties of atomic vectors and lists, and how to use R code to create them.

Atomic vectors

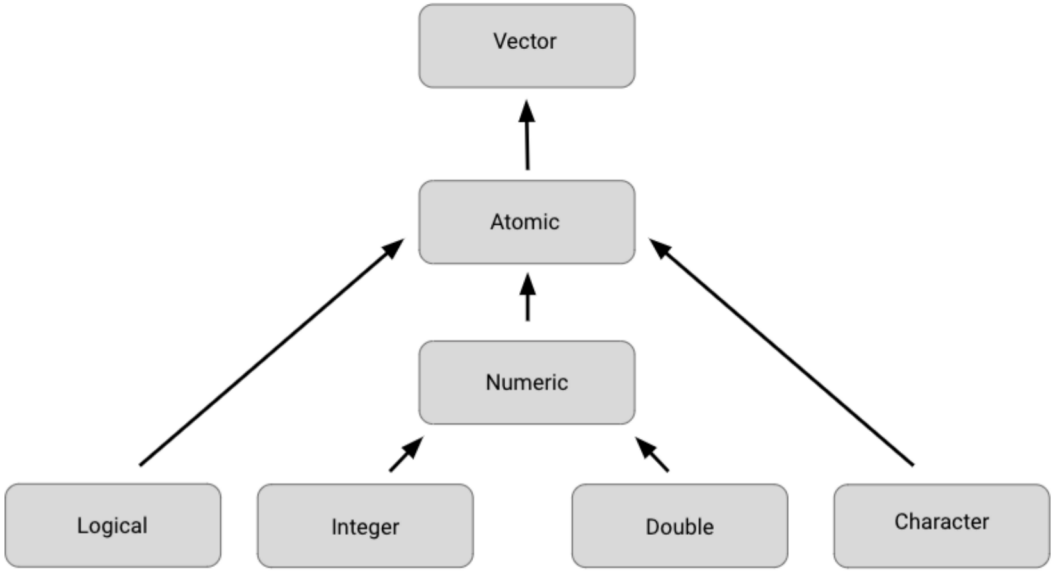
First, we will go through the different types of atomic vectors. Then, you will learn how to use R code to create, identify, and name the vectors.

Earlier, you learned that a **vector** is a group of data elements of the *same* type, stored in a sequence in R. You cannot have a vector that contains both logicals and numerics.

There are six primary types of atomic vectors: logical, integer, double, character (which contains strings), complex, and raw. The last two–complex and raw–aren't as common in data analysis, so we will focus on the first four. Together, integer and double vectors are known as numeric vectors because they both contain numbers. This table summarizes the four primary types:

Type	Description	Example
Logical	True/False	TRUE
Integer	Positive and negative whole values	3
Double	Decimal values	101.175
Character	String/character values	"Coding"

This diagram illustrates the hierarchy of relationships among these four main types of vectors:



Creating vectors

One way to create a vector is by using the **c()** function (called the "combine" function). The **c()** function in R combines multiple values into a vector. In R, this function is just the letter "c" followed by the values you want in your vector inside the parentheses, separated by a comma: **c(x, y, z, ...)**.

For example, you can use the **c()** function to store numeric data in a vector.

```
c(2.5, 48.5, 101.5)
```

To create a vector of integers using the **c()** function, you must place an **L** directly after each number.

```
c(1L, 5L, 15L)
```

You can also create a vector containing characters or logicals.

```
c(“Sara” , “Lisa” , “Anna”)
```

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

Determining the properties of vectors

Every vector you create will have two key propertes: type and length.

You can determine what type of vector you are working with by using the **typeof()** function. Place the code for the vector inside the parentheses of the function. When you run the function, R will tell you the type.

For example:

```
typeof(c(“a” , “b”))
```

```
#> [1] "character"
```

Notice that the output of the typeof function in this example is the word “**character**”. Similarly, if you use the typeof function on a vector with integer values, then the output will include the word “**integer**” instead:

```
typeof(c(1L , 3L))
```

```
#> [1] "integer"
```

You can determine the length of an existing vector–meaning the number of elements it contains–by using the **length()** function. In this example, we use an assignment operator to assign the vector to the variable **x**. Then, we apply the **length()** function to the variable. When we run the function, R tells us the length is “**3**”.

```
x <- c(33.5, 57.75, 120.05)

length(x)

#> [1] 3
```

You can also check if a vector is a specific type by using an **is** function: **is.logical(), is.double(), is.integer(), is.character()**. In this example, R returns a value of **TRUE** because the vector contains integers.

```
x <- c(2L, 5L, 11L)

is.integer(x)

#> [1] TRUE
```

In this example, R returns a value of **FALSE** because the vector does *not* contain characters, rather it contains logicals.

```
y <- c(TRUE, TRUE, FALSE)

is.character(y)

#> [1] FALSE
```

Naming vectors

All types of vectors can be named. Names are useful for writing readable code and describing objects in R. You can name the elements of a vector with the **names()** function. As an example, let's assign the variable x to a new vector with three elements.

```
x <- c(1, 3, 5)
```

You can use the names() function to assign a different name to each element of the vector.

```
names(x) <- c("a", "b", "c")
```

Now, when you run the code, R shows that the first element of the vector is named **a**, the second **b**, and the third c.

```
x
```

```
#> a b c
```

```
#> 1 3 5
```

Remember that an atomic vector can only contain elements of the same type. If you want to store elements of different types in the same data structure, you can use a list.

Creating Lists

Lists are different from atomic vectors because their elements can be of any type—like dates, data frames, vectors, matrices, and more. Lists can even contain other lists.

You can create a list with the **list()** function. Similar to the **c()** function, the **list()** function is just **list** followed by the values you want in your list inside parentheses: **list(x, y, z, ...)**. In this example, we create a list that contains four different kinds of elements: character (**"a"**), integer (**1L**), double (**1.5**), and logical (**TRUE**).

```
list("a", 1L, 1.5, TRUE)
```

Like we already mentioned, lists can contain other lists. If you want, you can even store a list inside a list inside a list—and so on.

```
list(list(list(1 , 3, 5)))
```

Determining the structure of lists

If you want to find out what types of elements a list contains, you can use the **str()** function. To do so, place the code for the list inside the parentheses of the function. When you run the function, R will display the data structure of the list by describing its elements and their types.

Let's apply the **str()** function to our first example of a list.

```
str(list("a", 1L, 1.5, TRUE))
```

We run the function, then R tells us that the list contains four elements, and that the elements consist of four different types: character (**chr**), integer (**int**), number (**num**), and logical (**logi**).

```
#> List of 4
#>  $ : chr "a"
#>  $ : int 1
#>  $ : num 1.5
#>  $ : logi TRUE
```

Let's use the str() function to discover the structure of our second example. First, let's assign the list to the variable z to make it easier to input in the str() function.

```
z <- list(list(list(1 , 3, 5)))
```

Let's run the function.

```
str(z)
```

```
#> List of 1
```

```
#>  $ :List of 1
```

```
#>   ..$ :List of 3
```

```
#>   .. ..$ : num 1
```

```
#>   .. ..$ : num 3
```

```
#>   .. ..$ : num 5
```

The indentation of the **$** symbols reflect the nested structure of this list. Here, there are three levels (so there is a list within a list within a list).

Naming lists

Lists, like vectors, can be named. You can name the elements of a list when you first create it with the list() function:

```
list("Chicago" = 1, "New York" = 2, "Los Angeles" = 3)
```

```
$Chicago
```

```
[1] 1
```

```
$`New York`
```

```
[1] 2
```

```
$`Los Angeles`
```

```
[1] 3
```

Additional resource

To learn more about vectors and lists, check out R for Data Science, Chapter 20: Vectors. R for Data Science is a classic resource for learning how to use R for data science and data analysis. It covers everything from cleaning to visualizing to communicating your data. If you want to get more details about the topic of vectors and lists, this chapter is a great place to start.