In programming, a **data structure** is a format for organizing and storing data. The most common data structures in the R programming language include vectors, data frames, matrices, and arrays. This reading focuses on vectors in R. Later on, you'll learn more about data frames, matrices, and arrays.

Think of a data structure as a house that contains your data. The separate parts of a house, such as bricks or boards, don't provide shelter on their own. But talented builders can use their skills and expertise, along with a blueprint, to build a functional house. Data structures are similar: Single data elements don't give you much information, but when data elements are combined into vectors, data frames, and other data structures, a talented data analyst can use them to solve a business challenge.

Just as there are rules that govern building a safe, sturdy house, there are certain rules that each type of data structure must follow. As you become more familiar with these rules, you'll become better at working with data and using it to draw conclusions in R.

There are two types of vectors in R: **atomic vectors** and **lists**.

Atomic vectors

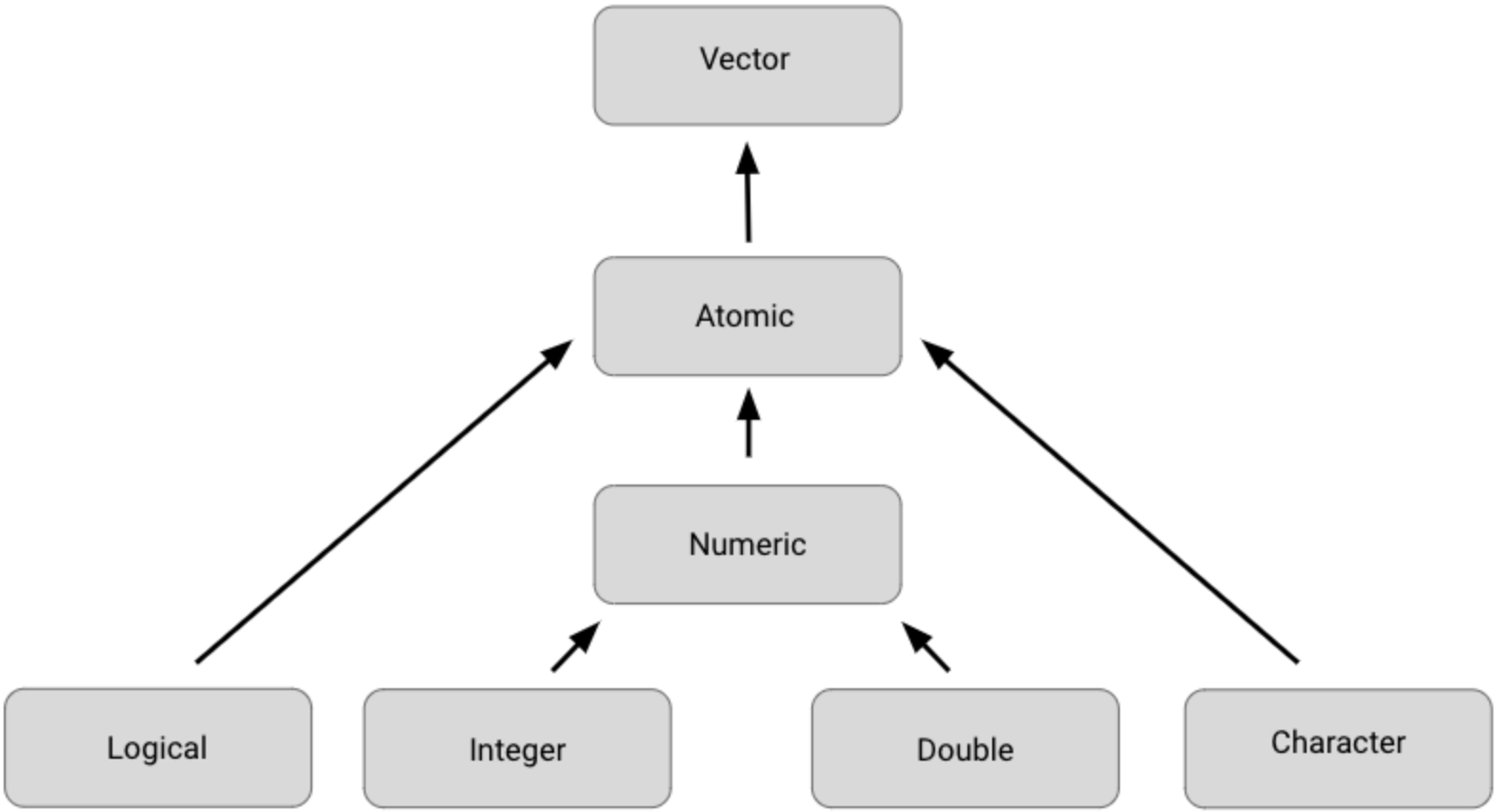
First, you'll explore the different types of atomic vectors. Then, you'll learn how to use R code to create, identify, and name the vectors. You'll also learn how to extract a subset of a vector.

In R, a **vector** is a group of data elements of the same type, stored in a one-dimensional sequence. Vectors can only contain data of one type.

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

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

This diagram demonstrates the hierarchy of relationships among these four main vector types. It indicates that numeric vectors can be either integers or doubles and that atomic vectors can be logical, numeric, or character.



Create vectors

One way to create a vector in R is with the `c()` function. This function combines multiple values into a vector. To use it, type 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 with the `c()` function, you must place the letter `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)
```

To create a vector of a sequence of numbers, use a colon to separate the starting number and the ending number. For example, the following code creates a vector, `z`, that includes the whole numbers from `4` through `10`, and then displays the vector:

```
z <- c(4:10)
z
```

Determine vector properties

Every vector has two key properties: type and length.

Vector type

Determine a vector's type with the `typeof()` function. Place the code for the vector inside the parentheses of the function. When you run the function, R will tell you its type. For example, when you run the following code, R returns the output `"character"`:

```
typeof(c("a", "b"))
```

Similarly, if you use the **`typeof()`** function on a vector with integer values, then the output is **`"integer"`** instead:

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

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

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

In contrast, in this example, R returns a value of **`FALSE`** because the vector contains logical values, not characters.

```
y <- c(TRUE, TRUE, FALSE)
is.character(y)
```

Vector length

Determine the length of an existing vector–meaning, the number of elements it contains–with the **`length()`** function. In this example, an assignment operator assigns the vector to the variable `x`. Then, the **`length()`** function is applied to the variable. When you run the function, R tells you the length is **`3`**.

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

Name vectors

You can name elements in vectors of any type with the `names()` function. Names are useful for writing readable code and describing objects in R.

As an example, the following code assigns the variable, `x`, to a new vector with three elements. Then, it uses the `names()` function to assign a different name to each element of the vector. Finally, it displays the variable, `x`. Run the following code:

```
x <- c(1, 3, 5)
names(x) <- c("a", "b", "c")
x
```

The R output indicates that the first element of the vector is named `a`, the second `b`, and the third `c`.

Extract a subset of a vector

Sometimes, you might want to extract a particular element or subset of elements from a vector. Do this by referencing the element's position in the vector or its name (if it has one) with the extract operator, `[]`.

For example, if you want to call the second element in the vector `x`, use the code `x[2]`.

```
x <- c(1, 3, 5)
names(x) <- c("a", "b", "c")
x[2]
```

Alternatively, vector element 2 is named `b`, so you can call it with the code `x["b"]`:

```
x <- c(1, 3, 5)
names(x) <- c("a", "b", "c")
```

```
x["b"]
```

In R, the indices of elements start with `1`, so to refer to the first element in a vector you would use the code **`x[1]`**.

Remember that an atomic vector can only contain elements of the same type.

Lists

Lists are different from atomic vectors because their elements can be of any type—including characters, integers, and logical values. Lists can even contain other lists, matrices, vectors, or data frames.

Create lists

You can create a list with the **`list()`** function. Similar to the **`c()`** function, the **`list()`** function is a list of the values inside the function's parentheses: **`list(x, y, z)`**. Run the following code to 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)
```

As you learned, lists can contain other lists. Run the following code to create a list inside a list:

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

Determine a list's structure

To find out what types of elements a list contains, 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 presents the list's data structure by describing its elements and their types.

Apply the `str()` function to the first list example.

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

R presents the list's structure as containing four elements. Each element is a different data type: character (`chr`), integer (`int`), number (`num`), and logical (`logi`).

Use the `str()` function to discover the structure of another example. First, assign the list to the variable z to make it easier to input in the `str()` function. Then, run the command `str(z)` to find `z`'s structure.

```
str(z)
```

The indentation of the `$` symbols reflect the nested structure of this list. Here, there are two levels, which means that there is a list within a list.

Name list elements

List elements, like vector elements, can be named when you create them with the `list()` function:

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

Key takeaways

Data structures are formats for organizing and storing data. Vectors are a common data structure used in R. You can create two types of vectors in R: atomic vectors and lists. Both of these vectors store data elements in a one-dimensional sequence. However, lists can store multiple types of data, whereas atomic vectors can only store one type of data. To create a vector, use the combine function, `c()`. To create a list, use the list function, `list()`.

Resources for more information

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.