Vectors and Loops

Kishore Hari

2022-08-11

Data Structures

So far, we have seen and played with words and numbers in R. In R's jargon, the words are called "characters" and the numbers are called "numericals". These are two of the 6 elementary data types R recognizes. The other data types are integer, logical, complex and raw. We will learn about these later in detail. Remember that these data types are also referred to as **class** of the data.

Data Structures, simply put, are collections of data having certain useful properties. A more formal definition for a data structure is *a format for efficient storage, retrieval and manipulation of data.* The files in our computers are a data structures as well, since they collect and store data (text, video, audio etc.) in a format that is easy to access and modify.

Vectors

The most basic data structures in R are vectors. A vector is simply a set of data points clubbed together. All data points in a vector must be of the same class. Thereby, the class of a vector is the class of the elements present in the vector.

Vectors are ubiquitous to R. The six elementary data types are considered as atomic vectors, i.e., vectors with a single element. Console output for an R command is often a vector as well.

The way to define a vector is as follows:

```
myVector <- c(1,2,4,2,6.5, 7) # A numerical vector
class(myVector)
```

```
## [1] "numeric"
```

```
myVector2 <- c("a", "P", "d", "Hello") # A character vector
class(myVector2)
```

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

```
#What happens if you introduce characters into a numerical vector? Find out!
```

Note that the function `c()` is used to create/define a vector.

A few useful tricks and functions:

```r
seriesVector <- 1:10 # syntax to generate a sequence of numbers
letters # vector of lowercase alphabet
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
## [20] "t" "u" "v" "w" "x" "y" "z"
```

```r
LETTERS # vector of uppercase alphabet
```

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

```r
length(seriesVector) # function that gives the number of elements in a vector.
```

```
## [1] 10
```

Indexing

Each vector has a certain number of elements, placed in an order. Therefore, each element of a vector can be accessed by using the position of that element. This position is referred to as index. Index is a concept applicable across different data structures. The syntax for accessing an element of a vector is as follows:

```r
myVector[3] # returns the third element of the vector myVector, defined previously
```

```
## [1] 4
```

```r
# What happens if you try to access the 10th element of the vector when there are only 6?
```

Loops

Often while coding, we need to repeat an instruction or a set of instructions multiple times. For example, consider calculating the sum of all numbers in a vector. To calculate the sum manually, one would go about doing what can be called progressive addition. Implementing the same in R would look something like:

```r
s <- 0 # variable containing the sum of the vector
s <- s + myVector[1]
s <- s + myVector[2]
s <- s + myVector[3]
s <- s + myVector[4]
s <- s + myVector[5]
s <- s + myVector[6]
```

Here, we repeat the addition 6 times, each time changing the index of the member of `myVector` being used. Such repetitive tasks can be concisely written using a **for loop**. A for loop takes a series of instructions and repeats them a number of times. The syntax is as follows:

```r
s <- 0
for (index in 1:6) {
    val <- myVector[index]
    s <- s + val
}
s
```

```
## [1] 22.5
```

The three aspects to pay attention are

1. index : the iterator. The value of the iterator gets updated with every iteration (loop)
2. The vector (`1:6` in this case). The length of the vector is the number of times the loop runs, while the value of the iterator in each iteration is obtained from the vector itself.
3. Commands inside the curly braces - these are the set of commands that will be repeated as many number of times as there are numbers in the vector.

It helps to think of for loop as a way to concisely write the repetition of a set of commands and nothing more.

Another way to write the same loop is :

```
s <- 0
for (val in myVector) {
    s <- s + val
}
s
```

```
## [1] 22.5
```

We will call the previous method the **index method** and this one the **value method**. Below are a few examples that we have seen in class:

```
## Print the squares of all numbers in a vector

for (val in myVector) {
    sq <- val^2
    print(sq)
}
```

```
## [1] 1
## [1] 4
## [1] 16
## [1] 4
## [1] 42.25
## [1] 49
```

```
# write the same loop using the index method.

## Calculate the product of square roots of all numbers in a vector

p <- 1
for (index in 1:length(myVector)) {
    val <- myVector[index]
    sqrtVal <- val^0.5
    p <- p * sqrtVal
}
p
```

```
## [1] 26.98148
```

```
# write the same loop using the value method
```

Excercise

1. Read the tutorialspoint links on for loops and R objects:
    - https://www.tutorialspoint.com/r/r_data_types.htm
    - https://www.tutorialspoint.com/r/r_for_loop.htm
2. Read 4 more links of your own choice about for loop.
3. Write programs to achieve the following tasks, first without using a for loop, then using a for loop.
    i. Print the first 10 letters of the alphabet. Make use of the `LETTERS` vector pre-defined in R.
    ii. Construct a right-angled traingle, using asteriscs (*). The pattern would like below:
        - ****
    iii. Calculate the factorial of a given number. See the formula give here: https://en.wikipedia.org/wiki/Factorial
    iv. A person has taken a loan from the bank worth Rs. 1,00,000, at an interest rate of 6% per annum. Assuming the interest is simple, write a program to calculate the amount owed to the bank at the end of 5 years. Write a similar program, assuming compound interest.