

# R - Vectors

Vectors are the most basic R data objects and there are six types of atomic vectors. They are logical, integer, double, complex, character and raw.

## Vector Creation

### Single Element Vector

Even when you write just one value in R, it becomes a vector of length 1 and belongs to one of the above vector types.

```
# Atomic vector of type character.  
print("abc");
```

```
# Atomic vector of type double.  
print(12.5)
```

```
# Atomic vector of type integer.  
print(63L)
```

```
# Atomic vector of type logical.  
print(TRUE)
```

```
# Atomic vector of type complex.  
print(2+3i)
```

```
# Atomic vector of type raw.  
print(charToRaw('hello'))
```

When we execute the above code, it produces the following result –

```
[1] "abc"  
[1] 12.5  
[1] 63  
[1] TRUE  
[1] 2+3i  
[1] 68 65 6c 6c 6f
```

### Multiple Elements Vector

#### Using colon operator with numeric data

```
# Creating a sequence from 5 to 13.  
v <- 5:13  
print(v)
```

```
# Creating a sequence from 6.6 to 12.6.
v <- 6.6:12.6
print(v)
```

```
# If the final element specified does not belong to the
sequence then it is discarded.
v <- 3.8:11.4
print(v)
```

When we execute the above code, it produces the following result –

```
[1] 5 6 7 8 9 10 11 12 13
[1] 6.6 7.6 8.6 9.6 10.6 11.6 12.6
[1] 3.8 4.8 5.8 6.8 7.8 8.8 9.8 10.8
```

## Using sequence (Seq.) operator

```
# Create vector with elements from 5 to 9 incrementing by
0.4.
print(seq(5, 9, by = 0.4))
```

When we execute the above code, it produces the following result –

```
[1] 5.0 5.4 5.8 6.2 6.6 7.0 7.4 7.8 8.2 8.6 9.0
```

## Using the c() function

The non-character values are coerced to character type if one of the elements is a character.

```
# The logical and numeric values are converted to
characters.
s <- c('apple', 'red', 5, TRUE)
print(s)
```

When we execute the above code, it produces the following result –

```
[1] "apple" "red"    "5"      "TRUE"
```

## Accessing Vector Elements

Elements of a Vector are accessed using indexing. The **[ ]** brackets are used for indexing. Indexing starts with position 1. Giving a negative value in the index drops that element from result. **TRUE**, **FALSE** or **0** and **1** can also be used for indexing.

```
# Accessing vector elements using position.
t <- c("Sun", "Mon", "Tue", "Wed", "Thurs", "Fri", "Sat")
u <- t[c(2, 3, 6)]
```

```
print(u)
```

```
# Accessing vector elements using logical indexing.  
v <- t[c(TRUE,FALSE,FALSE,FALSE,FALSE,TRUE,FALSE)]  
print(v)
```

```
# Accessing vector elements using negative indexing.  
x <- t[c(-2,-5)]  
print(x)
```

```
# Accessing vector elements using 0/1 indexing.  
y <- t[c(0,0,0,0,0,0,1)]  
print(y)
```

When we execute the above code, it produces the following result –

```
[1] "Mon" "Tue" "Fri"  
[1] "Sun" "Fri"  
[1] "Sun" "Tue" "Wed" "Fri" "Sat"  
[1] "Sun"
```

## Vector Manipulation

### Vector arithmetic

Two vectors of same length can be added, subtracted, multiplied or divided giving the result as a vector output.

```
# Create two vectors.  
v1 <- c(3,8,4,5,0,11)  
v2 <- c(4,11,0,8,1,2)
```

```
# Vector addition.  
add.result <- v1+v2  
print(add.result)
```

```
# Vector subtraction.  
sub.result <- v1-v2  
print(sub.result)
```

```
# Vector multiplication.  
multi.result <- v1*v2  
print(multi.result)
```

```
# Vector division.  
divi.result <- v1/v2  
print(divi.result)
```

When we execute the above code, it produces the following result –

```
[1]  7 19  4 13  1 13
[1] -1 -3  4 -3 -1  9
[1] 12 88  0 40  0 22
[1] 0.7500000 0.7272727      Inf 0.6250000 0.0000000
5.5000000
```

## Vector Element Recycling

If we apply arithmetic operations to two vectors of unequal length, then the elements of the shorter vector are recycled to complete the operations.

```
v1 <- c(3,8,4,5,0,11)
v2 <- c(4,11)
# v2 becomes c(4,11,4,11,4,11)
```

```
add.result <- v1+v2
print(add.result)
```

```
sub.result <- v1-v2
print(sub.result)
```

When we execute the above code, it produces the following result –

```
[1]  7 19  8 16  4 22
[1] -1 -3  0 -6 -4  0
```

## Vector Element Sorting

Elements in a vector can be sorted using the **sort()** function.

```
v <- c(3,8,4,5,0,11, -9, 304)
```

```
# Sort the elements of the vector.
sort.result <- sort(v)
print(sort.result)
```

```
# Sort the elements in the reverse order.
revsort.result <- sort(v, decreasing = TRUE)
print(revsort.result)
```

```
# Sorting character vectors.
v <- c("Red", "Blue", "yellow", "violet")
sort.result <- sort(v)
print(sort.result)
```

```
# Sorting character vectors in reverse order.  
revsort.result <- sort(v, decreasing = TRUE)  
print(revsort.result)
```

When we execute the above code, it produces the following result -

```
[1] -9  0  3  4  5  8 11 304  
[1] 304 11  8  5  4  3  0 -9  
[1] "Blue" "Red" "violet" "yellow"  
[1] "yellow" "violet" "Red" "Blue"
```