Chapter 2 Data Structures

2.1 Vectors

A vector is the simplest type of data structure in R. Vectors are particularly important as most of the functions you will write will work with vectors. Simply put, a vector is a sequence of data elements of the same basic type.

We can construct a vector using the combine `(c)` function:

```
# Here is a vector containing three numeric values 3, 5 and 7:
numbers <- c(3, 5, 7)

#Here is a vector of logical values:
logical <- c(TRUE, FALSE, TRUE, FALSE, FALSE)

#Here is a vector of character values:
fruits <- c("apple", "oranges", "banana")
```

We can also change the value in a vector:

```
fruits[2] <- "strawberries"
```

Every vector has two key properties: its type and its length:

```
typeof(fruits)
length(fruits)
```

You can combine vectors using the combine function as well:

```
results <- c(1, 2, 3)
other_results <- c(4, 5, 6)
combined_results <- c(results, other_results)
```

Vectors can be added together in an element-wise operation:

```
total_add <- results + other_results
total_div <- results / other_results
total_mod <- results %% other_results
```

Functions can be applied to each element of the vector. However, not all functions are vectorized (we will cover this later):

```
total_rou <- round(total_div, 1)

names <- c("Simmons", "Race", "Healey", "LaBarr", "Villanes")
names <- sort(names, decreasing = TRUE)
```

There are also functions for constructing useful types of vectors:

```
# Replicate - replicates a value X number of times
identity_vector <- rep(1,10)
identity_vector

# Sequence operator ":" From:To
sequence_vector <- 1:10

# Sequence function - allows sequencing by a value
sequence_by_two_vector <- seq(0,10, 2)
sequence_by_two_vector

# Sample function - take a random sample from a vector
random_vector <- sample(sequence_by_two_vector,3)

#A common trick to randomly permute all elements in a vector is sample(vector_name)
permute <- sample(numbers)
```

Finally, a vector doesn't have to be made with consistent elements, but it will force them to one type:

```
values <- c("IAA", 1, "2021", 5)
typeof(values)
```

As we mentioned earlier, the elements of a vector can only be of one type. If we want elements of different types… we will need a list.

2.2 Lists

Lists are objects which contain elements of different types like − numbers, strings, vectors and another list inside it. A list can also contain a matrix or a function as its elements. A list is created using list() function.

```
# Here are 3 vectors of different types
v1 <- c("apple", "banana")
v2 <- c("dog", "cat", "bunny", "pig", "cow", "horse")
v3 <- seq(0,10,by=2)

v1;v2;v3
```

We can put all of these vectors into a list:

```
# Here are 3 vectors of different types
l1 <- list(v1, v2, v3)

class(l1)
typeof(l1)
```

List elements can be accessed via index. Each of the components of a list is accessed via the double bracket [[x]] syntax:

```
# First element in the list
l1[[1]]

# First element in the first element (vector) of the list
l1[[1]][1]
```

List elements can be named:

```
names(l1) <- c("Fruit", "Animals", "Even_Numbers")
```

Or they can be named upon creating the list:

```
l1 <- list(Fruit=v1, Animals=v2, Even_Numbers=v3)
```

The elements in a list can be retrieved by name:

```
l1$Fruit
l1$Fruit[1]

l1$Even_Numbers
max(l1$Even_Numbers)
```

A list can even contain lists:

```
l2 <- list(Odd_Numbers=seq(1,10,by=2), list1=l1)

# First Vector
l2$Odd_Numbers
l2[[1]]

# l1 within L2
l2$list1
l2[[2]]

l2$list1$Fruit
l2[[2]][1]

l2$list1$Fruit[1]
l2[[2]][[1]][1]

l2[[2]][[1]][3] <- "blueberry"
l2$list1$Fruit
l2
```

To combine lists:

```
l4 <- list(More_Fruit=c("melon", "orange"))
l1 <- c(l1,l4)
```

Finally, if we want to update part of a list, we can just operate on and update it accordingly:

```
l1$Fruit <- c(l1$Fruit, l4$More_Fruit)
l1$Fruit
```

2.3 Arrays

Arrays are similar to vectors, except that they are multi-dimensional. An array is created using the array() function, and the dim parameter to specify the dimensions:

```
# 1 dimension - 1 row (vector)
sequence_array_1_dim <- array(1:10, dim=10)
sequence_array_1_dim

# 2 dimensions - 2 rows, 5 columns
sequence_array_2_dim <- array(1:10, dim= c(2,5))
sequence_array_2_dim

# 3 dimensions - 2 rows, 5 columns, 2 tables
sequence_array_3_dim <- array(1:20, dim = c(2,5,2))
sequence_array_3_dim

# 4 dimensions - 2 rows, 5 columns, 2 tables, 2 sets
sequence_array_4_dim <- array(1:40, dim = c(2,5,2,2))
sequence_array_4_dim
```

Similar to vectors, functions can be applied to arrays:

```
dim(sequence_array_2_dim)
length(sequence_array_2_dim)
sum(sequence_array_2_dim)
max(sequence_array_2_dim)
```

2.4 Matrices

Matrices are objects in which the elements are arranged in a two-dimensional rectangular layout. A matrix is created using the matrix() function:

```
# A matrix is a 2-demensional (row x column) array
A <- matrix(1:25, 5, 5)

B <- matrix(1:25, 5, 5, byrow=TRUE) #if you want to fill down the row instead of across
the columns

C <- matrix(1:5, 1, 5)

D <- matrix(1:5, 5, 1)
```

Matrices are convenient because we can do linear algebra with them:

```
# Element-wise operators
A + B
A - B
A * B
A / B

# Matrix Multiplication
C %*% D

# Transpose
C
t(C)
```

We can also name the matrix columns and and rows:

```
colnames(A) <- c("Column_1", "Column_2", "Column_3", "Column_4", "Column_5")
A

colnames(A)[4] <- "Changed_It"
A

rownames(A)  <- c("Row_1", "Row_2", "Row_3", "Row_4", "Row_5")
A
```

We can also use some helpful functions for matrices:

```
colSums(A)
rowSums(A)
sum(A)
dim(A)
length(A)
```

And we can extract information from a matrix using similar index notation (row, column):

```
A[2,1] # Single element

A[,1] # All rows in the first column
A[,c(1,2)] # All rows for columns 1 and 2

A[1,] # All columns in the first row
A[c(1,2),] # All columns for rows 1 and 2

A["Row_1","Column_2"]
```

2.5 Data Frames

Data Frames are data displayed in a format as a table. A data frame is essentially a 2 dimensional matrix (row x column). You can think of it like an excel table or relational database table. Data Frames can have different types of data inside it, and we use the data.frame() function to create a data frame:

```r
# Create a data frame called dt1 from a set of vectors:
dt1 <- data.frame (
  names =  c("Simmons", "Race", "Healey", "LaBarr", "Villanes"),
  class = c("Time Series", "Linear Algebra", "Visualization", "Finance", "Programming"),
  female = c(1,1,0,0,1)
)
```

To view the data frame:

```r
# View the table
View(dt1)

# Head of the table
head(dt1, 3)
```

To change the column names:

```r
colnames(dt1)
colnames(dt1) <- c("Last_Name", "Class_Taught", "Female?")
colnames(dt1)
colnames(dt1)[2] <- "Classes_Taught"
colnames(dt1)
```

To reference a column by name:

```r
dt1$Last_Name
dt1$Classes_Taught
```

Data frame indexes - similar to vectors and arrays. We can perform the following actions:

```r
# Retrieve a row
# leaving the row or column blank means "everything"
# the negative operator "-" means "everything but"
dt1[1,]
dt1[2:5,]
dt1[-1,]

# Retrieve a column
dt1[,1]
dt1[,-3]
dt1[,c(1:2)]
dt1[,"Last_Name"]
dt1[,c("Last_Name","Female?")]

# Retrieve an element
dt1[5,2]

# Reassign an element
dt1[5,2] <- "R Programming"
dt1[5,]

#Find elements that meet a condition
dt1$`Female?`==1
dt1$Last_Name=='Race'

# Filter by an value
dt1[dt1$`Female?`==1,]
dt1[dt1$Last_Name=='Race',]

# Add a new row - rbind (row-bind) function. The cbind function is its column version.
dt1 <- rbind(dt1, c("Larsen", "Teaching Assistant", 0))
dt1

# Add a new column
dt1$First_Name <- c("Susan", "Shaina", "Christopher", "Aric", "Andrea", "Nicholas")
```

```r
# Change the order of columns
dt1 <- dt1[,c(4,1:3)]

# Make columns based off other columns
dt1$First_Name_Length <- nchar(dt1$First_Name)
dt1$Last_Name_Length <- nchar(dt1$Last_Name)
dt1$Total_Name_Length <- dt1$First_Name_Length + dt1$Last_Name_Length + 1
dt1$Full_Name <- paste(dt1$First_Name, dt1$Last_Name, sep=" ")
dt1

# Remove some intermediate step columns
dt1 <- dt1[,-c(5,6)]
```

2.6 Working with different types

We can examine the class of each object using the class function:

```r
total_add
class(total_add)

names
class(names)

l2
class(l2)

sequence_array_1_dim
class(sequence_array_1_dim)

A
class(A)

dt1
class(dt1)

dt1$First_Name
class(dt1$First_Name)
```

Finally, we can also do some conversions between them using the “as” functions:

```r
new_array <- as.array(total_add)
class(new_array)

old_vector <- as.vector(new_array)
class(old_vector)

dfA <- as.data.frame(A)
class(dfA)
```