Code

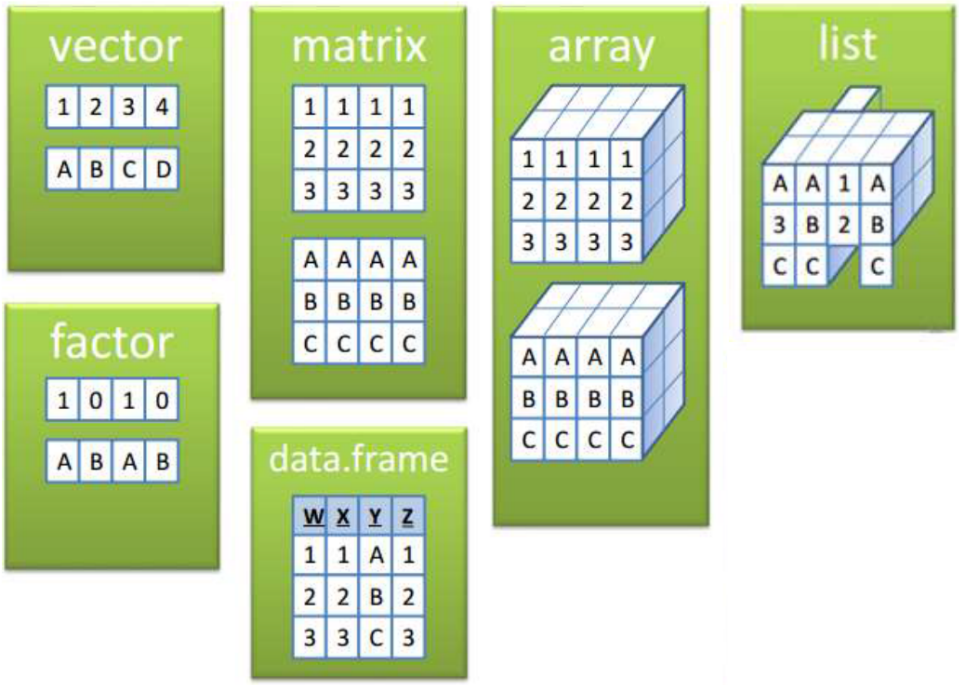
R (programming language) training course. By:Babak Babaabasi

R (programming language) training course

By: Babak Babaabasi
PhD student in Systems biomedicine

(2)R Data Structures:

R's basic data structures include the **vector, list, matrix, data frame, and factors**.



Vectors

A vector is simply a list of items that are of the same type.

To combine the list of items to a vector, use the `c()` function and separate the items by a comma.

In the example below, we create a vector variable called **fruits**, that combine strings:

Hide

```
# Vector of strings
fruits = c("banana", "apple", "orange")

# Print fruits
fruits
```

In this example, we create a vector that combines numerical values:

Hide

```
# Vector of numerical values
numbers = c(1, 2, 3)

# Print numbers
numbers
```

To create a vector with numerical values in a sequence, use the `:` operator:

Hide

```
# Vector with numerical values in a sequence
numbers = 1:10

numbers
```

You can also create numerical values with decimals in a sequence, but note that if the last element does not belong to the sequence, it is not used:

Hide

```
# Vector with numerical decimals in a sequence
numbers1 <- 1.5:6.5
numbers1

# Vector with numerical decimals in a sequence where the last element is not used
numbers2 <- 1.5:6.3
numbers2
```

In the example below, we create a vector of logical values:

Hide

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

log_values
```

Vector Length

To find out how many items a vector has, use the `length()` function

Hide

```
fruits <- c("banana", "apple", "orange")

length(fruits)
```

Sort a Vector

To sort items in a vector alphabetically or numerically, use the `sort()` function:

Hide

```
fruits <- c("banana", "apple", "orange", "mango", "lemon")
numbers <- c(13, 3, 5, 7, 20, 2)

sort(fruits)  # Sort a string
sort(numbers) # Sort numbers
```

*****Access Vectors

You can access the vector items by referring to its index number inside brackets `[]`. The first item has index 1, the second item has index 2, and so on:

Hide

```
fruits <- c("banana", "apple", "orange")

# Access the first item (banana)
fruits[1]
```

You can also access multiple elements by referring to different index positions with the `c()` function:

Hide

```
fruits <- c("banana", "apple", "orange", "mango", "lemon")

# Access the first and third item (banana and orange)
fruits[c(1, 3)]
```

You can also use negative index numbers to access all items except the ones specified:

Hide

```
fruits <- c("banana", "apple", "orange", "mango", "lemon")

# Access all items except for the first item
fruits[c(-1)]
```

Change an Item

To change the value of a specific item, refer to the index number:

Hide

```
fruits <- c("banana", "apple", "orange", "mango", "lemon")

# Change "banana" to "pear"
fruits[1] <- "pear"

# Print fruits
fruits
```

Repeat Vectors

To repeat vectors, use the `rep()` function:

Repeat each value:

Hide

```
repeat_each <- rep(c(1,2,3), each = 3)

repeat_each
```

Repeat the sequence of the vector:

Hide

```
repeat_times <- rep(c(1,2,3), times = 3)

repeat_times
```

Repeat each value independently:

Hide

```
repeat_indepent <- rep(c(1,2,3), times = c(5,2,1))

repeat_indepent
```

Generating Sequenced Vectors

One of the examples on top, showed you how to create a vector with numerical values in a sequence with the `:` operator

Hide

```
numbers <- 1:10

numbers
```

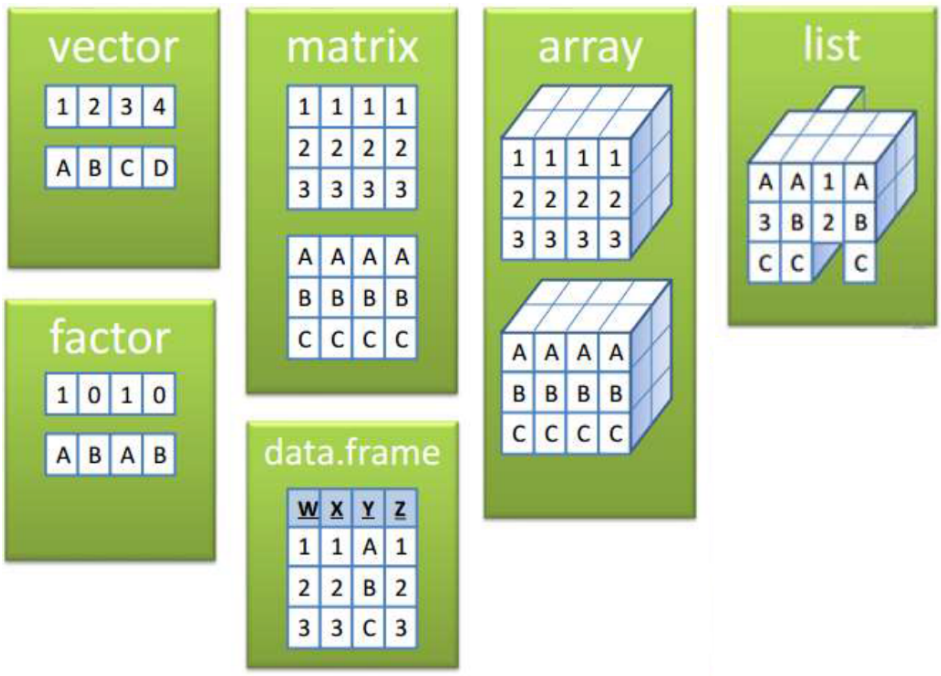
To make bigger or smaller steps in a sequence, use the `seq()` function:

Hide

```
numbers <- seq(from = 0, to = 100, by = 20)

numbers
```

Note: The `seq()` function has three parameters: `from` is where the sequence starts, `to` is where the sequence stops, and `by` is the interval of the sequence.

R Lists



A list in R can contain many different data types inside it. A list is a collection of data which is ordered and changeable.

To create a list, use the `list()` function:

Hide

```
thislist <- list("apple", "banana", "cherry")

# Print the list
thislist
```

Change Item Value

To change the value of a specific item, refer to the index number:

Hide

```
thislist <- list("apple", "banana", "cherry")
thislist[1] <- "blackcurrant"

# Print the updated list
thislist
```

List Length

To find out how many items a list has, use the `length()` function:

Hide

```
thislist <- list("apple", "banana", "cherry")

length(thislist)
```

Check if Item Exists

To find out if a specified item is present in a list, use the `%in%` operator:

Hide

```
thislist <- list("apple", "banana", "cherry")

"apple" %in% thislist
```

Add List Items

To add an item to the end of the list, use the `append()` function:

Hide

```
thislist <- list("apple", "banana", "cherry")

append(thislist, "orange")
```

To add an item to the right of a specified index, add " `after=index number` " in the `append()` function:

Hide

```
thislist <- list("apple", "banana", "cherry")

append(thislist, "orange", after = 2)
```

Remove List Items

You can also remove list items. The following example creates a new, updated list without an "apple" item:

Hide

```
thislist <- list("apple", "banana", "cherry")

newlist <- thislist[-1]

# Print the new list
newlist
```

Range of Indexes

You can specify a range of indexes by specifying where to start and where to end the range, by using the `:` operator:

Hide

```
thislist <- list("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")

(thislist)[2:5]
```

Loop Through a List

You can loop through the list items by using a `for` loop:

Hide

```
thislist <- list("apple", "banana", "cherry")

for (x in thislist) {
  print(x)
}
```

Join Two Lists

There are several ways to join, or concatenate, two or more lists in R.

The most common way is to use the `c()` function, which combines two elements together:

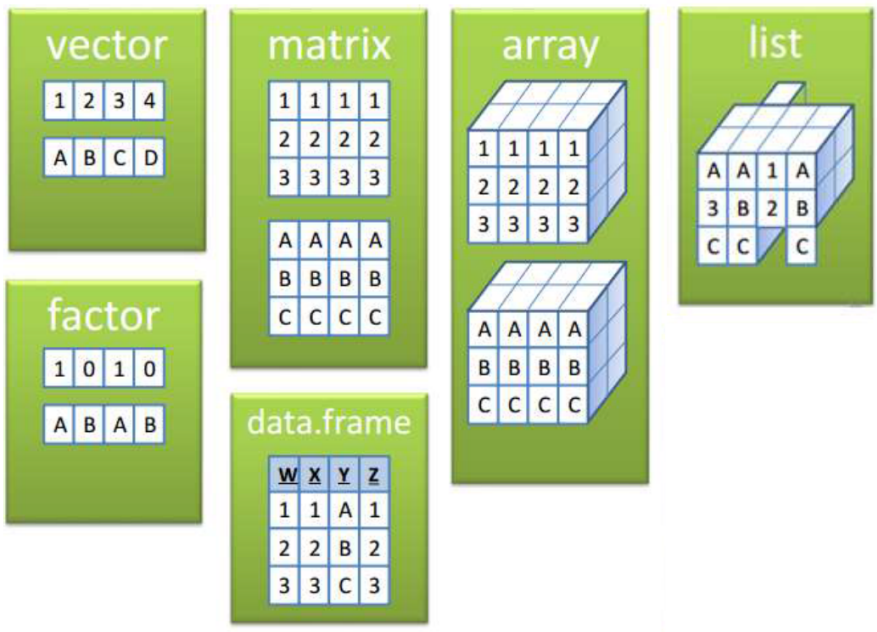
Hide

```
list1 <- list("a", "b", "c")
list2 <- list(1,2,3)
list3 <- c(list1,list2)

list3
```

Hide

```
# list1 and list2 are uni-dimensional lists
list1 <- list (c(1:5), "hi", 0 + 5i)
list1
list2 <- list(c(6:8))
list2
# create a list_data with two lists as its elements
list_data <- list(list1, list2)
print ("The two-dimensional list is : ")
print (list_data)

cat("Length of nested list is : ",
     length (list_data))
cat("Length of first inner list is : ",
     length (list_data[[1]]))
```

R Matrices



A matrix is a two dimensional data set with columns and rows.

A column is a vertical representation of data, while a row is a horizontal representation of data.

A matrix can be created with the `matrix()` function. Specify the `nrow` and `ncol` parameters to get the amount of rows and columns:

Hide

```
# Create a matrix
thismatrix <- matrix(c(1,2,3,4,5,6), nrow = 3, ncol = 2)

# Print the matrix
thismatrix
```

You can also create a matrix with strings:

Hide

```
thismatrix <- matrix(c("apple", "banana", "cherry", "orange"), nrow = 2, ncol = 2)

thismatrix
```

Access Matrix Items

You can access the items by using `[ ]` brackets. The first number “1” in the bracket specifies the row-position, while the second number “2” specifies the column-position:

Hide

```
thismatrix <- matrix(c("apple", "banana", "cherry", "orange"), nrow = 2, ncol = 2)

thismatrix[1, 2]
```

The whole row can be accessed if you specify a comma **after** the number in the bracket:

Hide

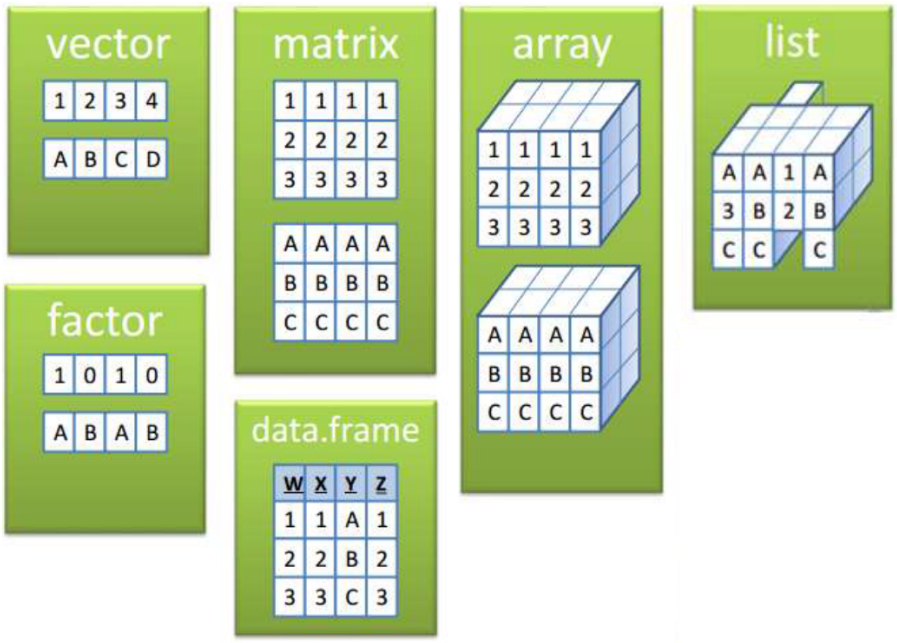
```
thismatrix <- matrix(c("apple", "banana", "cherry", "orange"), nrow = 2, ncol = 2)

thismatrix[2,]
```

The whole column can be accessed if you specify a comma **before** the number in the bracket:

Hide

```
thismatrix <- matrix(c("apple", "banana", "cherry", "orange"), nrow = 2, ncol = 2)

thismatrix[,2]
```

R Arrays



Compared to matrices, arrays can have more than two dimensions.

We can use the `array()` function to create an array, and the `dim` parameter to specify the dimensions:

Hide

```
# An array with one dimension with values ranging from 1 to 24
thisarray <- c(1:24)
thisarray

# An array with more than one dimension
multiarray <- array(thisarray, dim = c(4, 3, 2))
multiarray
```

How does `dim=c(4,3,2)` work? The first and second number in the bracket specifies the amount of rows and columns. The last number in the bracket specifies how many dimensions we want.

Note: Arrays can only have one data type.

Access Array Items

You can access the array elements by referring to the index position. You can use the `[]` brackets to access the desired elements from an array:

Hide

```
thisarray <- c(1:24)
multiarray <- array(thisarray, dim = c(4, 3, 2))

multiarray[2, 3, 2]
```

You can also access the whole row or column from a matrix in an array, by using the `c()` function:

Hide

```
thisarray <- c(1:24)

# Access all the items from the first row from matrix one
multiarray <- array(thisarray, dim = c(4, 3, 2))
multiarray[c(1),,1]

# Access all the items from the first column from matrix one
multiarray <- array(thisarray, dim = c(4, 3, 2))
multiarray[,c(1),1]
```

Check if an Item Exists

To find out if a specified item is present in an array, use the `%in%` operator:

Hide

```
thisarray <- c(1:24)
multiarray <- array(thisarray, dim = c(4, 3, 2))

2 %in% multiarray
```

Amount of Rows and Columns

Use the `dim()` function to find the amount of rows and columns in an array:

Hide

```
thisarray <- c(1:24)
multiarray <- array(thisarray, dim = c(4, 3, 2))

dim(multiarray)
```

Array Length

Use the `length()` function to find the dimension of an array:

Hide

```
thisarray <- c(1:24)
multiarray <- array(thisarray, dim = c(4, 3, 2))

length(multiarray)
```

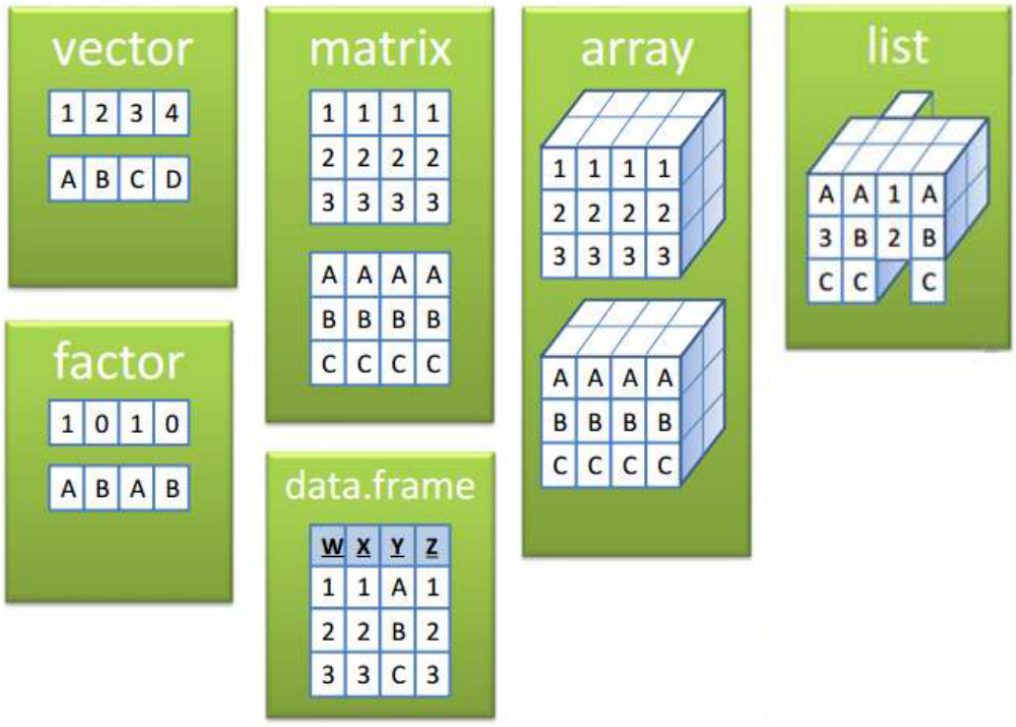
Loop Through an Array

You can loop through the array items by using a `for` loop:

Hide

```
thisarray <- c(1:24)
multiarray <- array(thisarray, dim = c(4, 3, 2))

for(x in multiarray){
  print(x)
}
```

R Data Frames



Data Frames are data displayed in a format as a table.

Data Frames can have different types of data inside it. While the first column can be `character`, the second and third can be `numeric` or `logical`. However, each column should have the same type of data.

Use the `data.frame()` function to create a data frame:

Hide

```
# Create a data frame
Data_Frame <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)

# Print the data frame
Data_Frame
```

Summarize the Data

Use the `summary()` function to summarize the data from a Data Frame:

Hide

```
Data_Frame <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)

Data_Frame

summary(Data_Frame)
```

Access Items

We can use single brackets `[ ]`, double brackets `[[ ]]` or `$` to access columns from a data frame:

Hide

```
Data_Frame <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)

Data_Frame[1]

Data_Frame[["Training"]]

Data_Frame$Training
```

Add Rows

Use the `rbind()` function to add new rows in a Data Frame:

Hide

```
Data_Frame <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)

# Add a new row
New_row_DF <- rbind(Data_Frame, c("Strength", 110, 110))

# Print the new row
New_row_DF
```

Add Columns

Use the `cbind()` function to add new columns in a Data Frame:

Hide

```
Data_Frame <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)

# Add a new column
New_col_DF <- cbind(Data_Frame, Steps = c(1000, 6000, 2000))

# Print the new column
New_col_DF
```

Remove Rows and Columns

Use the `c()` function to remove rows and columns in a Data Frame:

Hide

```
Data_Frame <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)

# Remove the first row and column
Data_Frame_New <- Data_Frame[-c(1), -c(1)]

# Print the new data frame
Data_Frame_New
```

Amount of Rows and Columns

Use the `dim()` function to find the amount of rows and columns in a Data Frame:

Hide

```
Data_Frame <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)

dim(Data_Frame)
```

You can also use the `ncol()` function to find the number of columns and `nrow()` to find the number of rows:

Hide

```
Data_Frame <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)

ncol(Data_Frame)
nrow(Data_Frame)
```

Data Frame Length

Use the `length()` function to find the number of columns in a Data Frame (similar to `ncol()`):

Hide

```
Data_Frame <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)

length(Data_Frame)
```

Combining Data Frames

Use the `rbind()` function to combine two or more data frames in R vertically:

Hide

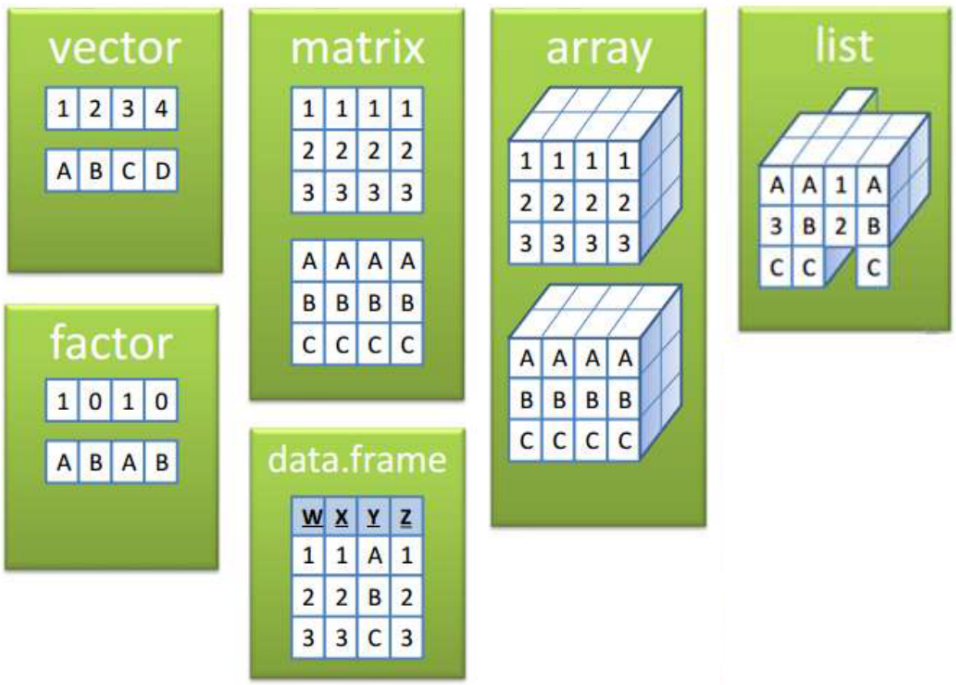
```
Data_Frame1 <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)

Data_Frame2 <- data.frame (
  Training = c("Stamina", "Stamina", "Strength"),
  Pulse = c(140, 150, 160),
  Duration = c(30, 30, 20)
)

New_Data_Frame <- rbind(Data_Frame1, Data_Frame2)
New_Data_Frame
```

And use the `cbind()` function to combine two or more data frames in R horizontally:

Hide

```
Data_Frame3 <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)

Data_Frame4 <- data.frame (
  Steps = c(3000, 6000, 2000),
  Calories = c(300, 400, 300)
)

New_Data_Frame1 <- cbind(Data_Frame3, Data_Frame4)
New_Data_Frame1
```

R Factors



Factors are used to categorize data. Examples of factors are:

- Demography: Male/Female
- Music: Rock, Pop, Classic, Jazz
- Training: Strength, Stamina

To create a factor, use the `factor()` function and add a vector as argument:

Hide

```
# Create a factor
music_genre <- factor(c("Jazz", "Rock", "Classic", "Classic", "Pop", "Jazz", "Rock", "Jazz"))

# Print the factor
music_genre
```

You can see from the example above that that the factor has four levels (categories): Classic, Jazz, Pop and Rock.

To only print the levels, use the `levels()` function:

Hide

```
music_genre <- factor(c("Jazz", "Rock", "Classic", "Classic", "Pop", "Jazz", "Rock", "Jazz"))

levels(music_genre)
```

You can also set the levels, by adding the `levels` argument inside the `factor()` function:

Hide

```
music_genre <- factor(c("Jazz", "Rock", "Classic", "Classic", "Pop", "Jazz", "Rock", "Jazz"),
levels = c("Classic", "Jazz", "Pop", "Rock", "Other"))

levels(music_genre)
```

Factor Length

Use the `length()` function to find out how many items there are in the factor:

Hide

```
music_genre <- factor(c("Jazz", "Rock", "Classic", "Classic", "Pop", "Jazz", "Rock", "Jazz"))

length(music_genre)
```

Access Factors

To access the items in a factor, refer to the index number, using `[]` brackets:

Hide

```
music_genre <- factor(c("Jazz", "Rock", "Classic", "Classic", "Pop", "Jazz", "Rock", "Jazz"))

music_genre[3]
```

Change Item Value

To change the value of a specific item, refer to the index number:

Hide

```
music_genre <- factor(c("Jazz", "Rock", "Classic", "Classic", "Pop", "Jazz", "Rock", "Jazz"))

music_genre[3] <- "Pop"

music_genre[3]
```

Note that you cannot change the value of a specific item if it is not already specified in the factor. The following example will produce an error:

Hide

```
music_genre <- factor(c("Jazz", "Rock", "Classic", "Classic", "Pop", "Jazz", "Rock", "Jazz"))

music_genre[3] <- "Opera"

music_genre[3]
```

However, if you have already specified it inside the `levels` argument, it will work:

Hide

```
music_genre <- factor(c("Jazz", "Rock", "Classic", "Classic", "Pop", "Jazz", "Rock", "Jazz"),
levels = c("Classic", "Jazz", "Pop", "Rock", "Opera"))

music_genre[3] <- "Opera"

music_genre[3]
```

END