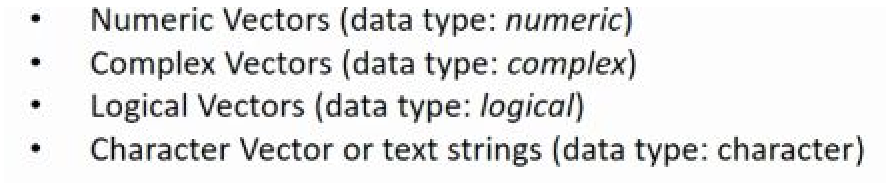
Type Data Structure:
Data Structure can be defined as the specific form of organizing and storing the data. R Programming supports five basic types of data structure namely Vector, matrix, Array, , and list.
Vector:
Vector is a sequence of data elements of the same basic type. Members of a Vector are formally called components. Whenever we are storing the numerical value in ‘a’ or ‘b’ that ‘a’ or ‘b’ nothing but a Vector, so Vector is one-dimensional array used to store collection data of the same data type.



- Numeric Vectors (data type: *numeric*)
- Complex Vectors (data type: *complex*)
- Logical Vectors (data type: *logical*)
- Character Vector or text strings (data type: character)

Similar kind of data type you can store like all the Numeric data(data type numeric) in a Vector, again you can store complex(data type complex) similarly in logical(data type logical) or character(data type character) you can store this thing in a Vector. There is six type of atomic Vector, they are Logical, Integer, Double, Complex, Character, and Raw.
Matrices:
Matrix is a collection of data elements of the same mode arranged in a two-dimensional rectangular layout. We can create a matrix containing only characters or only logical values, these are not of much use. We use matrices containing numeric elements to be used in mathematical calculations. They are accessed by two integer indices.
Three kinds of matrices are

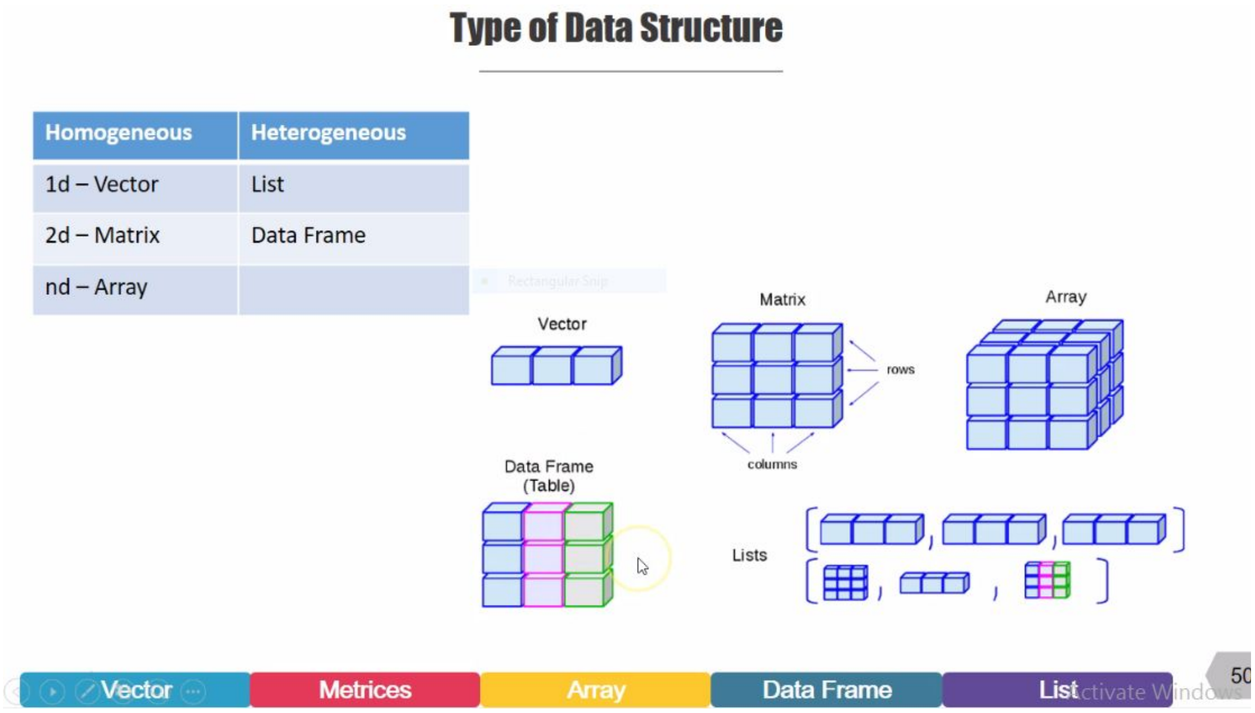
- ***Matrix Multiplication***
- ***R matrix transpose***
- ***Matrix power***

Arrays:
Similar to matrices but they can be multi-dimensional(more than two dimensions). Array function() takes vectors as input and uses the values in the dim parameter to create an array.
If you want to store age, salary, location, designation then it becomes your array. Salary or age all those things are numeric but if you store numeric as well as a character variable, character variable is nothing but a gender like male or female then you can use an array.
Data Frames:
Generalization of matrices where different columns can store in a different mode then it’s called Data Frame. It is a list of vectors of equal length. Data frame is a table or two-dimensional array like structure in which each column contains values of one variable and each row contains one set of values from each column. It can store a different kind of data types like you can store numerical with categorical and logical
Lists:
Lists are something where you can store all those other four, you can store data frame in lists, you can store an array in a list or you can store matrices and you can store a vector as well.

Lists ordered a collection of objects where the elements can be of different types. Lists contain the elements of different types like-number, strings, vectors and another list inside it. Lists are created using lists function().

Types of Data Structure:

Vector, Matrix, and Array those are in the homogeneous so they can store homogeneous data either numeric, character or logical but only different in dimension, Vector is one dimension, Matrix is two dimension and Array is multidimensional.



Type of Data Structure

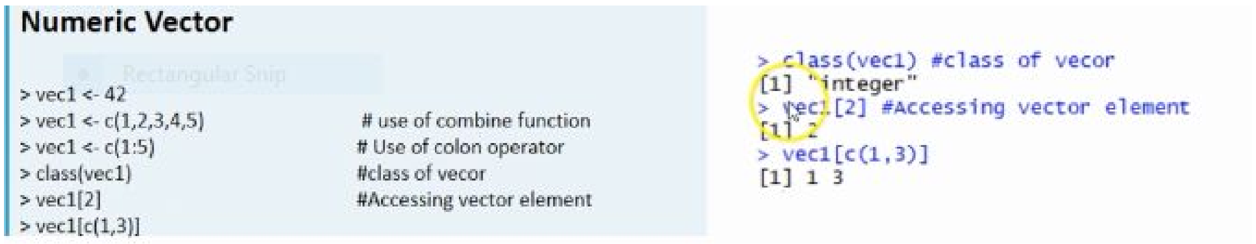
Homogeneous	Heterogeneous
1d – Vector	List
2d – Matrix	Data Frame
nd – Array	

On the other side if you talk about Data frame and lists those are in heterogeneous so they can store a different kind of data type.

VECTOR

Numeric Vector:

Now we will learn about the Vector. Suppose you want to store 42 (vec1<-42)in vector 1 then you can see the result in vector 1. Similarly if you store more than variable 1-5 (vec1<-c(1,2,3,4,5) .

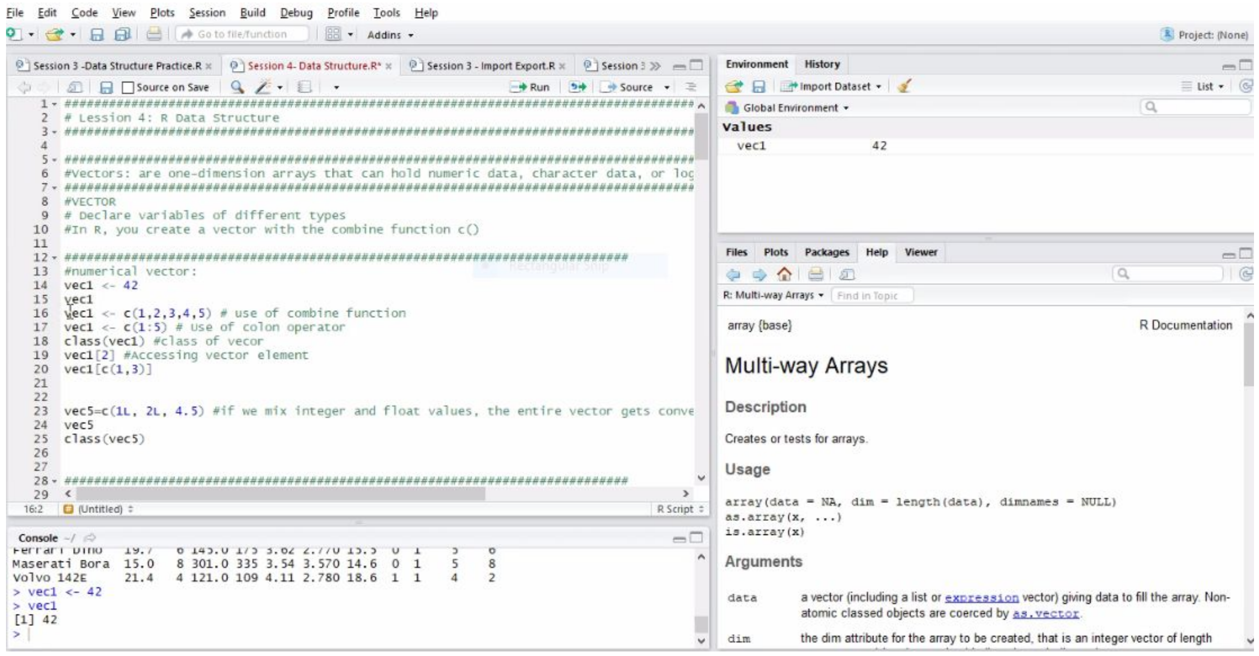


Numeric Vector

```
> vec1 <- 42
> vec1 <- c(1,2,3,4,5)          # use of combine function
> vec1 <- c(1:5)                # Use of colon operator
> class(vec1)                   #class of vecor
> vec1[2]                       #Accessing vector element
> vec1[c(1,3)]
```

```
> class(vec1) #class of vecor
[1] "integer"
> vec1[2] #Accessing vector element
[1] 2
> vec1[c(1,3)]
[1] 1 3
```

Here we show you both the way either you can use combine function and use “,” and you can store 1,2,3,4,5 (vec1<-c(1,2,3,4,5) or you can store 1-5 (vec1<-c(1:5) the only advantage is suppose you want to store it 1,7,9,3,5 then also you can use the combine function and you can store it.



```r
###############################################################################
# Lession 4: R Data Structure
###############################################################################

###############################################################################
#Vectors: are one-dimension arrays that can hold numeric data, character data, or log
###############################################################################
#VECTOR
# Declare variables of different types
#In R, you create a vector with the combine function c()

##########################################################################
#numerical vector:
vec1 <- 42
vec1
vec1 <- c(1,2,3,4,5) # use of combine function
vec1 <- c(1:5) # Use of colon operator
class(vec1) #class of vecor
vec1[2] #Accessing vector element
vec1[c(1,3)]


vec5=c(1L, 2L, 4.5) #if we mix integer and float values, the entire vector gets conve
vec5
class(vec5)


##########################################################################
```

```
Ferrari Dino   19.7   6 145.0 175 3.62 2.770 15.5  0  1   5   6
Maserati Bora  15.0   8 301.0 335 3.54 3.570 14.6  0  1   5   8
Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.6  1  1   4   2
> vec1 <- 42
> vec1
[1] 42
>
```

Values
vec1   42

array {base}   R Documentation

Multi-way Arrays

Description

Creates or tests for arrays.

Usage

```
array(data = NA, dim = length(data), dimnames = NULL)
as.array(x, ...)
is.array(x)
```

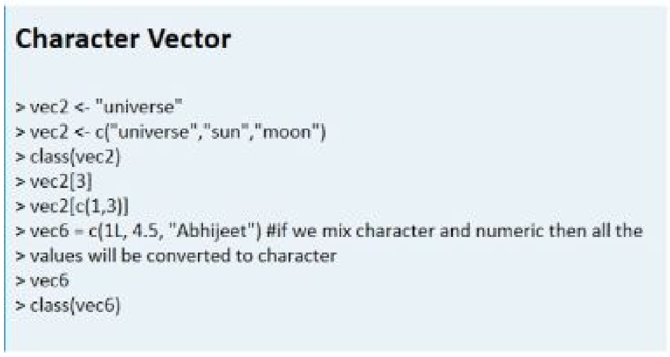
Arguments

data	a vector (including a list or expression vector) giving data to fill the array. Non-atomic classed objects are coerced by as.vector.
dim	the dim attribute for the array to be created, that is an integer vector of length

If you see the class of the vector 1 then it would be an integer. Likewise, if you want to access a particular suppose you have like 1,2,3,4,5 but you only access the second element of your vector then you would get probably 2 since 1-5 in store. Likewise, you can also access 1 and 3 from that vector

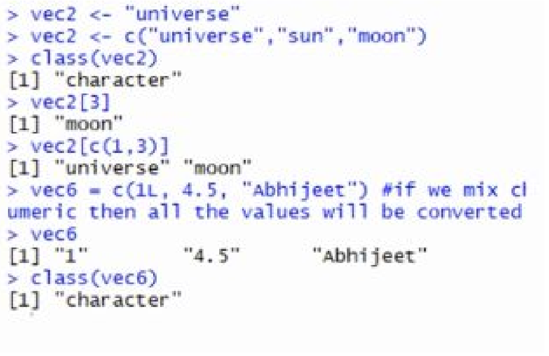
Character Vector:

Within double coat, you just pass the value which you want to store. Suppose in Vec2 we want to in “universe” (vec2<-”Universe”) but if you store more than 1 character variable.



Character Vector

```
> vec2 <- "universe"
> vec2 <- c("universe","sun","moon")
> class(vec2)
> vec2[3]
> vec2[c(1,3)]
> vec6 = c(1L, 4.5, "Abhijeet") #if we mix character and numeric then all the
> values will be converted to character
> vec6
> class(vec6)
```

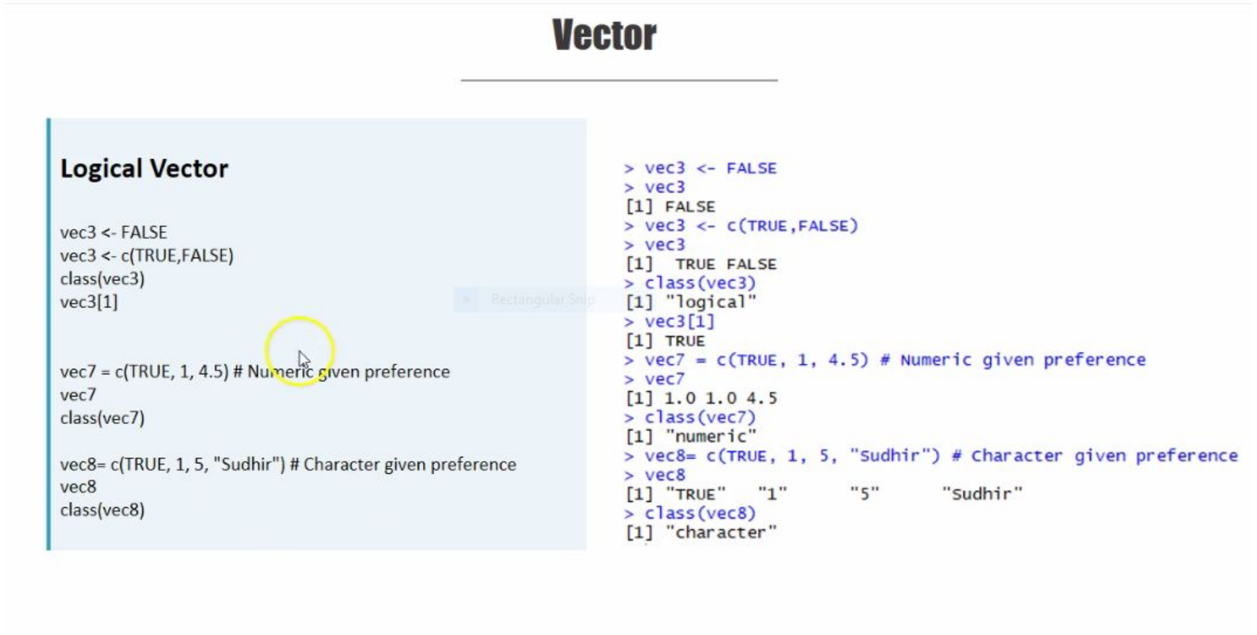


```
> vec2 <- "universe"
> vec2 <- c("universe","sun","moon")
> class(vec2)
[1] "character"
> vec2[3]
[1] "moon"
> vec2[c(1,3)]
[1] "universe" "moon"
> vec6 = c(1L, 4.5, "Abhijeet") #if we mix cl
umeric then all the values will be converted
> vec6
[1] "1"         "4.5"       "Abhijeet"
> class(vec6)
[1] "character"
```

Then you have to use the combine function (vec2<-c(“Universe”,”sun”,”moon”) and if you see the class of it then you definitely get the class as character.

If you mix the character and numeric then all the values will be converted to character.

Logical Vector:
You can store TRUE and FAlSE for the logical function. Suppose we are using vector 3. Here you store FALSE for vec3(Vec3<-FALSE). In vec3 if you store more than one variable which is TRUE and FALSE (Vec3<-c(TRUE, FALSE) and finally if you class of it, it would be logical.



Vector

Logical Vector

```
vec3 <- FALSE
vec3 <- c(TRUE,FALSE)
class(vec3)
vec3[1]


vec7 = c(TRUE, 1, 4.5) # Numeric given preference
vec7
class(vec7)

vec8= c(TRUE, 1, 5, "Sudhir") # Character given preference
vec8
class(vec8)
```

```
> vec3 <- FALSE
> vec3
[1] FALSE
> vec3 <- c(TRUE,FALSE)
> vec3
[1]  TRUE FALSE
> class(vec3)
[1] "logical"
> vec3[1]
[1] TRUE
> vec7 = c(TRUE, 1, 4.5) # Numeric given preference
> vec7
[1] 1.0 1.0 4.5
> class(vec7)
[1] "numeric"
> vec8= c(TRUE, 1, 5, "Sudhir") # Character given preference
> vec8
[1] "TRUE"   "1"      "5"      "Sudhir"
> class(vec8)
[1] "character"
```

If you mix the logical vector function with numeric vector function then the numeric function gets the preferences and If you mix character with logical function and numeric function then all value converted to character function so character is given the preferences.