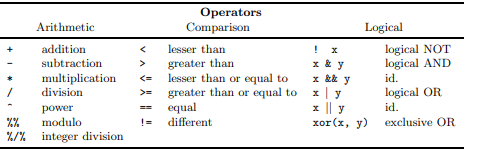
**Lecture 3: Operators, Decision making and Loops**

**3.1 Operator**:

We have seen previously that there are three main types of operators in R. The list of operators given below:

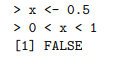


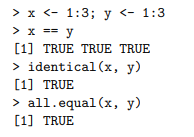
The arithmetic and comparison operators act on two elements (x + y, a < b). The arithmetic operators act not only on variables of mode numeric or complex, but also on logical variables; in this latter case, the logical values are coerced into numeric. The comparison operators may be applied to any mode: they return one or several logical values.

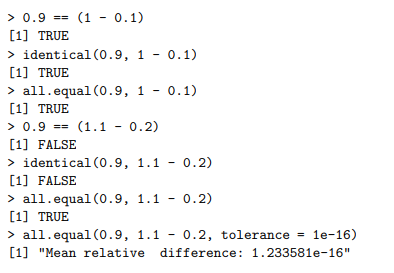
The logical operators are applied to one (!) or two objects of mode logical, and return one (or several) logical values. The operators “AND” and “OR” exist in two forms: the single one operates on each elements of the objects and returns as many logical values as comparisons done; the double one operates on the first element of the objects.

It is necessary to use the operator “AND” to specify an inequality of the type 0 < x < 1 which will be coded with: 0 < x & x < 1. The expression 0 < x < 1 is valid, but will not return the expected result: since both operators are the same, they are executed successively from left to right. The comparison 0 < x is first done and returns a logical value which is then compared to 1 (TRUE or FALSE < 1): in this situation, the logical value is implicitly coerced into numeric (1 or 0 < 1).

Some example:





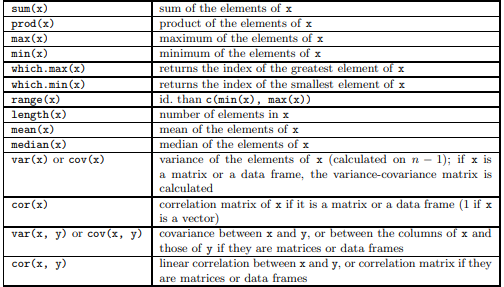


3.1. 1Arithmetics and simple functions

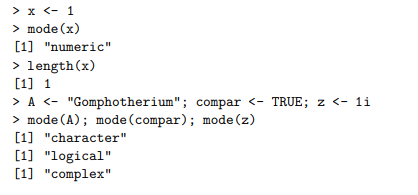
There are numerous functions in R to manipulate data. We have already seen the simplest one, c which concatenates the objects listed in parentheses. For example:



The functions available in R for manipulating data are too many to be listed here. One can find all the basic mathematical functions (log, exp, log10, log2, sin, cos, tan, asin, acos, atan, abs, sqrt, . . .), special functions (gamma, digamma, beta, besselI, . . .), as well as diverse functions useful in statistics. Some of these functions are listed in the following table.



R works with objects which are, of . , characterized by their names and their content, but also by attributes which specify the kind of data represented by an object. It is clear that the statistical analysis of this variable will not be the same in both cases: with R, the attributes of the object give the necessary information. More technically, and more generally, the action of a function on an object depends on the attributes of the latter. All objects have two intrinsic attributes: mode and length. The mode is the basic type of the elements of the object; there are four main modes: numeric, character, complex , and logical (FALSE or TRUE). Other modes exist but they do not represent data, for instance function or expression. The length is the number of elements of the object. To display the mode and the length of an object, one can use the functions mode and length, respectively.



R stores information as objects. Objects can be variables, functions, or more general structures built from those components. You assign data to objects using the assign operator <-. In the example shown in the script , we assigned some values to the variables x, y, and z. Those variables are stored as objects in R.

The assignment statement will usually have the form:

**object\_name <- expression**

The object name can have letters, numbers, underscores, and dots. All the following names are valid in R: var\_name, VarName, var.name, var2. Names can never start with a number, so 2var is not a valid name. We suggest selecting one naming style and using it consistently. In this tutorial, we will use the underscore-separated naming style, e.g. var\_name.

The expression is the information that will be stored in that object. For the variable x, for example, the information stored in that object are the numbers 2, 5, 3, and 1. Since we were assigning multiple numbers to one object, we used the combine c() function: x <- c(2, 5, 3, 1).

To display the contents of an object, type the object name in the command line. For example, type x in the command line.



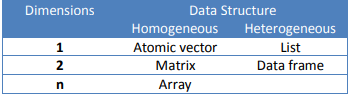
**2.2 Data frame**

In this section, we will discuss the ways that R stores and organizes data. R has five basic data structures: atomic vectors, matrices, arrays, lists, and data frames. These structures have specific requirements in terms of their dimension.

* One dimension: Atomic vectors and lists
* Two dimensions: Matrices and data frames
* N dimensions: Arrays

Data structures also have specific requirements in terms of their contents. Homogeneous structures have contents that are all of the same type. Heterogeneous structures can have contents of different types.

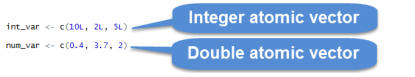
* Homogeneous: Atomic vectors, matrices, and arrays
* Heterogeneous: Lists and data frames



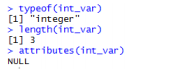
**2.2.1 Vectors**: Vectors are the basic data structure in R. There are two types of vectors: atomic vectors and lists. They have three common properties: type, length, and attribute. Type, as the name implies, describes the type of vector. Length describes the number of elements the vector contains. Attributes provide additional information, or metadata. Each of those properties can be checked using the following commands: typeof(), length(), attributes(). We will provide examples for using those commands in the following subsections.

**2.2.1.1 Atomic Vectors**: Atomic vectors can only have elements of the same type. There are four common types of atomic vectors: integer, double (often called numeric), character, and logical.

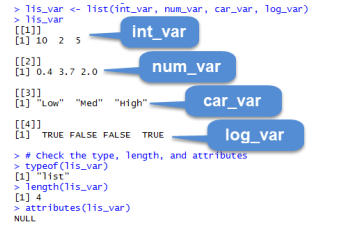
Go to the R script and run lines to create atomic vectors of each type. Notice that to create an integer atomic vector



Your console should show that the variable called int\_var is an integer atomic vector with three elements and no attributes.

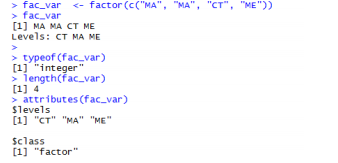


**2.2.1.2 List**: Lists, unlike atomic vectors, can have elements of any type. Go to the R script and run lines to create two lists and check their type, length, and attributes. The variable called lis\_var is a list with 4 components and no attributes. The variable called lis\_xyz is a list with 3 components and no attributes. Notice that length() now provides the number of components in a list. For atomic vectors, length() provides the number of elements in that atomic vector.



**2.2.1.2 Factors:** A factor is a vector used to store categorical variables. It is important to tell R that a variable is categorical, rather than character or numeric.

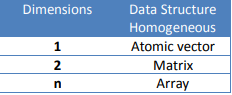
A common categorical variable is state (e.g., Massachusetts). Run lines the script to create variable fac\_var and check its type, length and attributes. The variable fac\_var is an integer vector with four elements and two attributes: class = factor and levels = CT, MA, and ME.



A factor in R is an integer atomic vector that has two attributes, class and levels. You can check the class attribute using the class() function

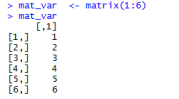


**2.2.1.3** Matrices and Arrays: Matrices and arrays are homogeneous data structures, just like atomic vectors, that can have more than one dimension, unlike vectors.

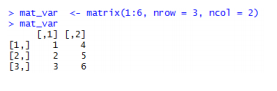


R stores matrices and arrays in a similar manner as vectors, but with the attribute called dimension. A matrix is an array that has two dimensions. Data in a matrix are organized into rows and columns. Matrices are commonly used while arrays are rare. We will focus on matrices for the rest of this section.

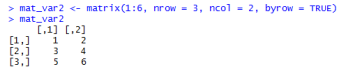
You create a matrix using the matrix() function. Run lines in the script to create a matrix. If you do not specify a number of rows and columns, R will create a matrix with one column and multiple rows. The variable mat\_var is a matrix of 1 column and 6 rows.



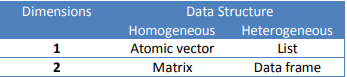
If you want that matrix to have a different number of columns and rows, you have to use the nrow and ncol attributes. Run lines in the script. Now the variable mat\_var has 3 rows and 2 columns.



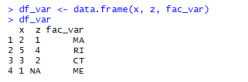
Matrices are constructed column wise. If you want to construct a matrix row wise, set the argument byrow = TRUE



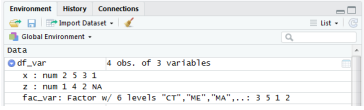
**2.2.1.4 Data Frames:** A data frame is the most common way of storing data in R. A data frame shares properties with both lists and matrices. Like a matrix, a data frame has 2 dimensions and data are organized into rows and columns. Each column represents a variable and each row represents an observation. You can also think of each column as a vector. All columns must have the same length, but they can be of different data types. A data frame is therefore a list of equal-length vectors and can thus be heterogeneous.



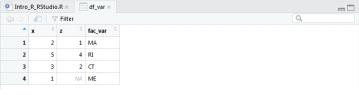
You create a data frame using the data.frame() function. Run lines 267 to 272 in the script to create a data frame that includes the atomic vectors x and z, and the factor fac\_var.

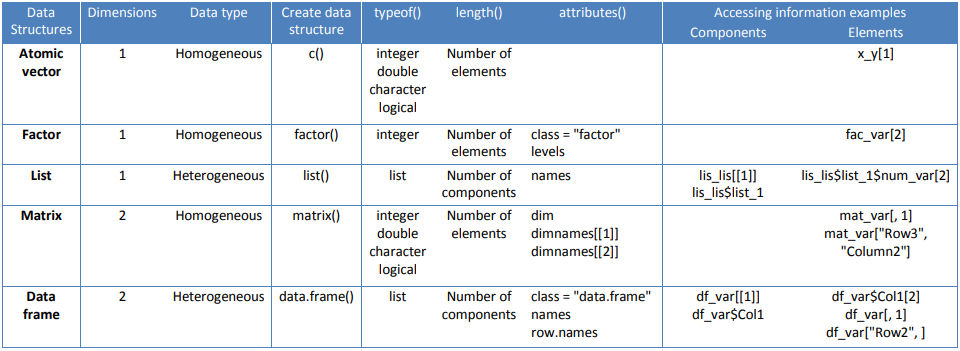


Each vector is a column of the data frame. Every column of a data frame must have the same length, so we used three vectors of the same length (4 elements). Notice that not all vectors included are of the same type, x and z are double vectors while fac\_var is a factor. The data frame maintains the data structure of each column. You can check the structure of each column by going to the environment pane and finding the variable under Data. If you click on the blue arrow next to a data frame, you can see the columns of that data frame. Notice that the fac\_var column is still saved as a factor and the x and z variables are numeric (or double).

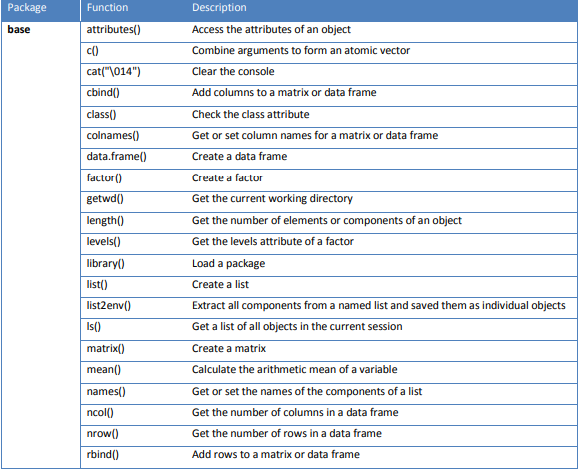


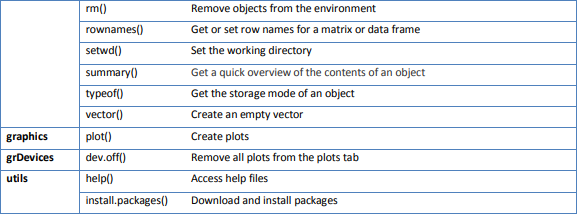
If you click on the name of the data frame, a window will open in the Source pane that will show you the contents of that data frame.



**Summary of Differences between Data Structures**

**List of Functions used commonly:**





**References:**

* A Gentle Introduction to R, Tufts University Data Lab Workshop, Kyle Monahan