**R\_DataTypes:**

Researchers work mostly with dataframes . With previous knowledge you can built dataframes in R Also, import dataframes into R.

primitive (or: atomic) data types in R are:

* numeric (integer, double, complex)
* character
* logical
* function

out of these, vectors, arrays, lists can be built.

**Numeric:** Decimal values are called **numerics**in R. It is the default computational data type. If we assign a decimal value to a variable x as follows, x will be of numeric type.

> x = 10.5       # assign a decimal value   
> x              # print the value of x   
[1] 10.5   
> class(x)       # print the class name of x   
[1] "numeric"

**Integer:** In order to create an **integer**variable in R, we invoke the as.integer function. We can be assured that y is indeed an integer by applying the is.integer function.

> y = as.integer(3)   
> y              # print the value of y   
[1] 3   
> class(y)       # print the class name of y   
[1] "integer"   
> is.integer(y)  # is y an integer?   
[1] TRUE

**Complex:** A **complex**value in R is defined via the pure imaginary value *i*.

> z = 1 + 2i     # create a complex number   
> z              # print the value of z   
[1] 1+2i   
> class(z)       # print the class name of z   
[1] "complex"

The following gives an error as −1 is not a complex value.

> sqrt(−1)       # square root of −1   
[1] NaN   
Warning message:   
In sqrt(−1) : NaNs produced

**Logical:** A **logical**value is often created via comparison between variables.

> x = 1; y = 2   # sample values   
> z = x > y      # is x larger than y?   
> z              # print the logical value   
[1] FALSE   
> class(z)       # print the class name of z   
[1] "logical"

Standard logical operations are "&" (and), "|" (or), and "!" (negation).

> u = TRUE; v = FALSE   
> u & v          # u AND v   
[1] FALSE   
> u | v          # u OR v   
[1] TRUE   
> !u             # negation of u   
[1] FALSE

**Character:** A **character**object is used to represent string values in R. We convert objects into character values with the as.character() function:

> x = as.character(3.14)   
> x              # print the character string   
[1] "3.14"   
> class(x)       # print the class name of x   
[1] "character"

Two character values can be concatenated with the paste function.

> fname = "Joe"; lname ="Smith"   
> paste(fname, lname)   
[1] "Joe Smith"

However, it is often more convenient to create a readable string with the sprintf function, which has a C language syntax.

> sprintf("%s has %d dollars", "Sam", 100)   
[1] "Sam has 100 dollars"

To extract a substring, we apply the substr function. Here is an example showing how to extract the substring between the third and twelfth positions in a string.

> substr("Mary has a little lamb.", start=3, stop=12)   
[1] "ry has a l"

And to replace the first occurrence of the word "little" by another word "big" in the string, we apply the sub function.

> sub("little", "big", "Mary has a little lamb.")   
[1] "Mary has a big lamb."