Intro R- Lists

In R a list is a data structure that contains an ordered sequence of objects that can be of different types. Unlike vectors and matrices, lists can contain all sorts of objects including numeric, logical and character vectors or matrices and even other lists. Construct a list by passing each object you wish the list to hold to the list() function:

```
# Create a new list containing a vector, a character string and a matrix

new_list <- list( c(1,2,3), "Life is Study", matrix(seq(1,6,1),2,3))

print(new_list)
```

```
## [[1]]
## [1] 1 2 3
##
## [[2]]
## [1] "Life is Study"
##
## [[3]]
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

You can give the objects in a list specific names upon creation:

```
# Create the same list but give the objects names

new_list <- list(vector= c(1,2,3),                   # *
                 char_string = "Life is Study",
                 my_matrix = matrix(seq(1,6,1),2,3))

print(new_list)
```

```
## $vector
## [1] 1 2 3
##
## $char_string
## [1] "Life is Study"
##
## $my_matrix
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

*Note: it is sometimes cleaner to break up long bits of code across multiple lines. You can access a list's object names or assign new names with the names() function:

```
names(new_list)  # Access names
```

```
## [1] "vector"      "char_string" "my_matrix"
```

```
names(new_list) <- c("obj1","obj2","obj3") # Assign new names

names(new_list)  # Access names
```

```
## [1] "obj1" "obj2" "obj3"
```

List Indexing

You can access the contents of lists several different ways. First, you can take slices of a list using the same single square bracket index notation you use with vectors:

```
new_list[1:2]         # Take a slice of the first 2 objects
```

```
## $obj1
## [1] 1 2 3
##
## $obj2
## [1] "Life is Study"
```

```
new_list[c(1,3)]      # Take a slice of the first and third objects
```

```
## $obj1
## [1] 1 2 3
##
## $obj3
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

```
new_list[2]           # Take a slice that contains only the second object
```

```
## $obj2
## [1] "Life is Study"
```

```
typeof(new_list[2])   # Check the type of the single-object slice
```

```
## [1] "list"
```

Note that list slices returned when you access a list with single brackets give you new, smaller lists, even if the slices only contain one object. To access a specific member of a list directly, wrap the object's index in double square brackets:

```
new_list[[2]]          # Access the second list object
```

```
## [1] "Life is Study"
```

```
typeof(new_list[[2]])  # Check the object's type
```

```
## [1] "character"
```

If you gave names to your list objects, you can access them directly using those names either by passing the names as characters wrapped in double square brackets or by using $ followed by the object name:

```
new_list[["obj3"]]   # Access an object directly by name with square brackets
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

```
new_list$obj3   # Access an object directly by name using $
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

After directly accessing a list object, you can perform indexing and other operations on the object as usual. For instance, after accessing a matrix in a list you can then index the matrix:

```
new_list[["obj3"]][2,2] # Get obj3 and then get the value in row 2 column 2
```

```
## [1] 4
```

List Summaries and Alterations

Since lists can hold objects of different types, math operations and many of the other functions that work on vectors and matrices do not work on lists. The most common list functions are those that give basic information about the list, such as length, str and summary:

```
length(new_list)     # Return the length of a list
```

```
## [1] 3
```

```
str(new_list)        # Get an overview of the list's structure
```

```
## List of 3
##  $ obj1: num [1:3] 1 2 3
##  $ obj2: chr "Life is Study"
##  $ obj3: num [1:2, 1:3] 1 2 3 4 5 6
```

```
summary(new_list)    # Get an summary of the list's contents
```

```
##      Length Class  Mode
## obj1 3      -none- numeric
## obj2 1      -none- character
## obj3 6      -none- numeric
```

*Note: In RStudio the environment pane in the upper right corner provides a summary of stored objects. Certain objects like lists have an arrow next to them that you can click on to expand and view additional information about the object's contents. To add a new object to a list, you can give the list a new name with specified value using the $ notation:

```
new_list$obj4 <- "Additional object"

str(new_list)
```

```
## List of 4
##  $ obj1: num [1:3] 1 2 3
##  $ obj2: chr "Life is Study"
##  $ obj3: num [1:2, 1:3] 1 2 3 4 5 6
##  $ obj4: chr "Additional object"
```

```
new_list[["obj5"]] <- "The bracket notation also works"

str(new_list)
```

```
## List of 5
##  $ obj1: num [1:3] 1 2 3
##  $ obj2: chr "Life is Study"
##  $ obj3: num [1:2, 1:3] 1 2 3 4 5 6
##  $ obj4: chr "Additional object"
##  $ obj5: chr "The bracket notation also works"
```

If you don't want to assign a name, you can add an object using the numeric index notation:

```
new_list[[6]] <- "this object has no name!"

str(new_list)
```

```
## List of 6
##  $ obj1: num [1:3] 1 2 3
##  $ obj2: chr "Life is Study"
##  $ obj3: num [1:2, 1:3] 1 2 3 4 5 6
##  $ obj4: chr "Additional object"
##  $ obj5: chr "The bracket notation also works"
##  $     : chr "this object has no name!"
```

The c() function can be used to join two lists together:

```
second_list <- list("combine me", "with new_list")
combined_list <- c(new_list, second_list)

str(combined_list)
```

```
## List of 8
##  $ obj1: num [1:3] 1 2 3
##  $ obj2: chr "Life is Study"
##  $ obj3: num [1:2, 1:3] 1 2 3 4 5 6
##  $ obj4: chr "Additional object"
##  $ obj5: chr "The bracket notation also works"
##  $     : chr "this object has no name!"
##  $     : chr "combine me"
##  $     : chr "with new_list"
```

Finally, you can remove list items by assigning them a value of NULL:

```
combined_list[[8]] = NULL   # remove item 8

combined_list$obj1 = NULL   # remove item the with the name obj1

str(combined_list)
```

```
## List of 6
##  $ obj2: chr "Life is Study"
##  $ obj3: num [1:2, 1:3] 1 2 3 4 5 6
##  $ obj4: chr "Additional object"
##  $ obj5: chr "The bracket notation also works"
##  $     : chr "this object has no name!"
##  $     : chr "combine me"
```

```
combined_list[3:6] = NULL   # remove items 3 through 6

str(combined_list)
```

```
## List of 2
##  $ obj2: chr "Life is Study"
##  $ obj3: num [1:2, 1:3] 1 2 3 4 5 6
```

In practice, lists are often created as a result of running functions on data so you probably won't spend a lot of time creating them yourself. When you work with data in R, it is more often in the form of a data frames, which are the subject of the next lesson. http://hamelg.blogspot.com/