Chapter 9: Lists

Section 9.1: Introduction to lists

Lists allow users to store multiple elements (like vectors and matrices) under a single object. You can use the `list` function to create a list:

```
l1 <- list(c(1, 2, 3), c("a", "b", "c"))
l1
## [[1]]
## [1] 1 2 3
##
## [[2]]
## [1] "a" "b" "c"
```

Notice the vectors that make up the above list are different classes. Lists allow users to group elements of different classes. Each element in a list can also have a name. List names are accessed by the `names` function, and are assigned in the same manner row and column names are assigned in a matrix.

```
names(l1)
## NULL
names(l1) <- c("vector1", "vector2")
l1
## $vector1
## [1] 1 2 3
##
## $vector2
## [1] "a" "b" "c"
```

It is often easier and safer to declare the list names when creating the list object.

```
l2 <- list(vec = c(1, 3, 5, 7, 9),
       mat = matrix(data = c(1, 2, 3), nrow = 3))
l2
## $vec
## [1] 1 3 5 7 9
##
## $mat
##      [,1]
## [1,]    1
## [2,]    2
## [3,]    3
names(l2)
## [1] "vec" "mat"
```

Above the list has two elements, named "vec" and "mat," a vector and matrix, resepcively.

Section 9.2: Quick Introduction to Lists

In general, most of the objects you would interact with as a user would tend to be a vector; e.g numeric vector, logical vector. These objects can only take in a single type of variable (a numeric vector can only have numbers inside it).

A list would be able to store any type variable in it, making it to the generic object that can store any type of variables we would need.

Example of initializing a list

```
exampleList1 <- list('a', 'b')
exampleList2 <- list(1, 2)
exampleList3 <- list('a', 1, 2)
```

In order to understand the data that was defined in the list, we can use the str function.

```
str(exampleList1)
str(exampleList2)
str(exampleList3)
```

Subsetting of lists distinguishes between extracting a slice of the list, i.e. obtaining a list containing a subset of the elements in the original list, and extracting a single element. Using the [operator commonly used for vectors produces a new list.

```
# Returns List
exampleList3[1]
exampleList3[1:2]
```

To obtain a single element use [[instead.

```
# Returns Character
exampleList3[[1]]
```

List entries may be named:

```
exampleList4 <- list(
    num = 1:3,
    numeric = 0.5,
    char = c('a', 'b')
)
```

The entries in named lists can be accessed by their name instead of their index.

```
exampleList4[['char']]
```

Alternatively the $ operator can be used to access named elements.

```
exampleList4$num
```

This has the advantage that it is faster to type and may be easier to read but it is important to be aware of a potential pitfall. The $ operator uses partial matching to identify matching list elements and may produce unexpected results.

```
exampleList5 <- exampleList4[2:3]

exampleList4$num
# c(1, 2, 3)

exampleList5$num
# 0.5

exampleList5[['num']]
# NULL
```

Lists can be particularly useful because they can store objects of different lengths and of various classes.

```
## Numeric vector
exampleVector1 <- c(12, 13, 14)
## Character vector
exampleVector2 <- c("a", "b", "c", "d", "e", "f")
## Matrix
exampleMatrix1 <- matrix(rnorm(4), ncol = 2, nrow = 2)
## List
exampleList3 <- list('a', 1, 2)

exampleList6 <- list(
    num = exampleVector1,
    char = exampleVector2,
    mat = exampleMatrix1,
    list = exampleList3
)
exampleList6
#$num
#[1] 12 13 14
#
#$char
#[1] "a" "b" "c" "d" "e" "f"
#
#$mat
#           [,1]        [,2]
#[1,] 0.5013050 -1.88801542
#[2,] 0.4295266  0.09751379
#
#$list
#$list[[1]]
#[1] "a"
#
#$list[[2]]
#[1] 1
#
#$list[[3]]
#[1] 2
```

Section 9.3: Serialization: using lists to pass information

There exist cases in which it is necessary to put data of different types together. In Azure ML for example, it is necessary to pass information from a R script module to another one exclusively throught dataframes. Suppose we have a dataframe and a number:

```
> df
       name height        team fun_index title age         desc Y
1    Andrea    195       Lazio        97     6  33   eccellente 1
2      Paja    165  Fiorentina        87     6  31       deciso 1
3      Roro    190       Lazio        65     6  28       strano 0
4    Gioele     70       Lazio       100     0   2    simpatico 1
5     Cacio    170    Juventus        81     3  33         duro 0
6     Edola    171       Lazio        72     5  32     svampito 1
7    Salami    175       Inter        75     3  30  doppiopasso 1
8    Braugo    180       Inter        79     5  32          gjn 0
9     Benna    158    Juventus        80     6  28     esaurito 0
10   Riggio    182       Lazio        92     5  31     certezza 1
11 Giordano    185        Roma        79     5  29        buono 1
```

```
> number <- "42"
```

We can access to this information:

```
> paste(df$name[4],"is a",df3$team[4], "supporter." )
[1] "Gioele is a  Lazio supporter."
> paste("The answer to THE question is", number )
[1] "The answer to THE question is 42"
```

In order to put different types of data in a dataframe we have to use the list object and the serialization. In particular we have to put the data in a generic list and then put the list in a particular dataframe:

```
l <- list(df,number)
dataframe_container <- data.frame(out2 = as.integer(serialize(l, connection=NULL)))
```

Once we have stored the information in the dataframe, we need to deserialize it in order to use it:

```
#----- unserialize ----------------------------------------+
unser_obj <- unserialize(as.raw(dataframe_container$out2))
#----- taking back the elements---------------------------+
df_mod        <- unser_obj[1][[1]]
number_mod    <- unser_obj[2][[1]]
```

Then, we can verify that the data are transfered correctly:

```
> paste(df_mod$name[4],"is a",df_mod$team[4], "supporter." )
[1] "Gioele is a  Lazio supporter."
> paste("The answer to THE question is", number_mod )
[1] "The answer to THE question is 42"
```