MTH 3220 R Notes 4

4 Higher Dimensional Arrays

4.1 Creating and Examining Arrays

- An ***array*** is like a matrix, but it can have more than two dimensions (e.g. it can have rows, columns, and layers).
- Here are some useful functions for creating and examining arrays:

```
array()           # Create an array, from a vector, with dimensions
                  # specified by the argument dim
dim()             # Returns the dimensions of an array
is.array()        # Indicates whether or not an object is an array
```

- Here's a three dimensional array. It has 3 rows, 3 columns, and 2 layers:

```
x <- array(1:18, dim = c(3, 3, 2))
x
```

```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]   10   13   16
## [2,]   11   14   17
## [3,]   12   15   18
```

```
dim(x)
```

```
## [1] 3 3 2
```

```
is.array(x)
```

```
## [1] TRUE
```

- Note that R fills the array one layer at a time, column by column within each layer.
- We can add names to the dimensions of an array (or get the names if they already exist) using:

```
dimnames()        # View the names of an array's dimensions or assign
                  # names via a list of character vectors
```

- For example, below we create an array `TempsArray` containing temperatures (°K) recorded in the northern hemisphere at four longitudes (rows), three latitudes (columns), and three months (layers), and we use `dimnames()` to add names to the three dimensions of the array:

```
TempsVec <- c(273, 239, 251, 236, 284, 260, 278, 257, 305, 299, 299, 288,
  270, 260, 258, 247, 291, 287, 279, 274, 314, 302, 299, 292, 279, 279, 273,
  277, 297, 301, 284, 288, 309, 301, 301, 292)

TempsArray <- array(TempsVec, dim = c(4, 3, 3))

dimnames(TempsArray) <- list(
      Long = c("0", "90" ,"180", "270"),
      Lat = c("75", "45", "15"),
      Month = c("Jan", "Apr", "Jul"))
```

```
TempsArray

## , , Month = Jan
##
##      Lat
## Long   75  45  15
##   0   273 284 305
##   90  239 260 299
##   180 251 278 299
##   270 236 257 288
##
## , , Month = Apr
##
##      Lat
## Long   75  45  15
##   0   270 291 314
##   90  260 287 302
##   180 258 279 299
##   270 247 274 292
##
## , , Month = Jul
##
##      Lat
## Long   75  45  15
##   0   279 297 309
##   90  279 301 301
##   180 273 284 301
##   270 277 288 292
```

4.2 Array Indexing Using [ ]

- We access array elements (rows, columns, layers, etc.) using square brackets:

```
[ , , ]    # Access array elements via their dimension indices (row,
           # column, layer, etc.) separated by commas
```

- For example, using the `TempsArray` array from above, to get the value in the 2nd row, 3rd column, and 1st layer, type:

```
TempsArray[2, 3, 1]            # Returns the value in the 2nd row, 3rd
                               # column, first layer of the array
```

```
## [1] 299
```

To get all the values in the 2nd row and 3rd column, type:

```
TempsArray[2, 3,  ]            # Returns a vector containing the values in
                               # the 2nd row and 3rd column of the array
```

```
## Jan Apr Jul
## 299 302 301
```

and to all get the values in the 1st layer, type:

```
TempsArray[, , 1]              # Returns a matrix containing the values in
                               # the 1st layer of the array
```

```
##      Lat
## Long   75  45  15
##   0   273 284 305
##   90  239 260 299
##   180 251 278 299
##   270 236 257 288
```

Section 4.2 Exercises

Exercise 1 Here's the `TempsArray` array again (as created above):

```
TempsArray

## , , Month = Jan
##
##      Lat
## Long   75  45  15
##   0   273 284 305
##   90  239 260 299
##   180 251 278 299
##   270 236 257 288
##
## , , Month = Apr
##
##      Lat
## Long   75  45  15
##   0   270 291 314
##   90  260 287 302
##   180 258 279 299
##   270 247 274 292
##
## , , Month = Jul
##
##      Lat
## Long   75  45  15
##   0   279 297 309
##   90  279 301 301
##   180 273 284 301
##   270 277 288 292
```

Guess what the results of each of the following will be, then check your answers (after creating the array using the commands given in Subsection 4.1).

a) `TempsArray[3, 1, 2]`

b) `TempsArray[, , 3]`

c) `TempsArray[1, , ]`

5 Lists

5.1 Creating and Examining Lists

- ***Lists*** are like vectors, but their elements can be *heterogeneous* (not all the same type). In fact, their elements can be *any* R *objects*. They're used to bundle objects together under one name.
- Lists are created and examined using:

```
list()            # Create a list
length()          # Returns the number of elements in a list
```

```
is.list()         # Indicates whether or not an object is a list
str()             # Describes the structure of a list
```

- Here's a simple list `y` that stores numeric, `"character"`, and `"logical"` values:

```
y <- list(2, "a", TRUE)
y

## [[1]]
## [1] 2
##
## [[2]]
## [1] "a"
##
## [[3]]
## [1] TRUE

is.list(y)

## [1] TRUE
```

Notice that the indices of the list elements are printed in double square brackets `[[ ]]`.

- Here's a list `x` whose elements are vectors and a matrix. Notice that we assign names (`x1`, `x2`, and `x3`) to the list elements when creating the list:

```
x <- list(x1 = c(1, 2, 3),
          x2 = c("a", "b", "c"),
          x3 = matrix(1:4, nrow = 2, ncol = 2))
x

## $x1
## [1] 1 2 3
##
## $x2
## [1] "a" "b" "c"
##
## $x3
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

When list elements have names (like `x1`, `x2`, and `x3` above), R prints the names preceded by a dollar sign `$`.

- `length()` tells how many elements are contained in a list:

```
length(x)

## [1] 3
```

- `str()` tells us the "structure" of the list:

```
str(x)

## List of 3
##  $ x1: num [1:3] 1 2 3
##  $ x2: chr [1:3] "a" "b" "c"
##  $ x3: int [1:2, 1:2] 1 2 3 4
```

The output above says that `x` is a list with three elements: a numeric vector `x1`, a `"character"` vector `x2`, and a `"integer"` matrix `x3`.

Section 5.1 Exercises

Exercise 2 The command below creates a list named `Employees` containing three elements:

- `Name`, a `"character"` vector containing the names of three employees
- `Salary`, a numeric vector containing their salaries
- `Union`, a `"logical"` vector indicating whether or not they belong to a union.

```
Employees <- list(
       Name = c("Joe", "Tom", "Ann"),
       Salary = c(50000, 47000, 65000),
       Union = c(FALSE, FALSE, TRUE))
```

a) After creating the `Employees` list, type:

```
Employees
```

and report the results.

b) Use `str()` to look at the structure of the list. Report the results.

c) Use `length()` to find the number of elements in the list. Report the result.

5.2 General List Operations

5.2.1 Accessing List Elements

- We access elements of a list using double square brackets `[[ ]]`, the dollar sign `$`, or single square brackets `[ ]`:

```
[[ ]]          # Access a list element via its index or name
$              # Access a list element via its name
[ ]            # Access a list element via its index or name, re-
               # turning a list
```

- We almost always use either `[[ ]]` or `$`. The main distinctions are:
  - `[[ ]]` and `$` return the actual *object* from the list. For example, if the list element is a vector, they return a vector.

  - `[ ]` doesn't return the actual *object* from the list, but instead returns a *list* containing that object.
  - `[[ ]]` can be used with numerical index values or names of list elements.
  - `$` can only be used with names of list elements.
  - `[ ]` can extract more than one list element, but `[[ ]]` and `$` extract only a single element.

- Here we use double square brackets `[[ ]]` to extract the second element from the list `x` created above:

```
x[[2]]                          # We could also use x[["x2"]]

## [1] "a" "b" "c"
```

  Note that a `"character"` vector is returned. We could also have used `x[["x2"]]`.

- Here we use the dollar sign operator `$` to extract the second element from `x`:

```
x$x2

## [1] "a" "b" "c"
```

  A `"character"` vector is returned here too.

- Now watch what happens when we use single brackets `[ ]` to extract the second element from `x`:

```
x[2]

## $x2
## [1] "a" "b" "c"
```

  This time a *list* with one element (the vector `x2`) is returned. This is usually *not* what we want.

5.2.2 Adding and Deleting List Elements

- The operators `[[ ]]` and `$` (and also `[ ]`) can be used to add or delete a list element.
- Below, we add the value 16 as the fourth element of `x` (created earlier) and give it the name `x4`:

```
x$x4 <- 16                      # We could also use x[["x4"]] <- 16
x

## $x1
## [1] 1 2 3
##
## $x2
## [1] "a" "b" "c"
##
## $x3
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
##
## $x4
## [1] 16
```

  Above, we could also have used `x[["x4"]] <- 16`.

- To delete a list element, we set its value to `NULL`:

```
x$x4 <- NULL
x

## $x1
## [1] 1 2 3
##
## $x2
## [1] "a" "b" "c"
##
## $x3
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

Section 5.2 Exercises

Exercise 3 Recreate the `Employees` list from Exercise 2.

a) What kind of object (vector or list) will returned by:

```
Employees[[2]]
```

What kind of object (vector or list) will returned by the following command:

```
Employees$Salary
```

What kind of object (vector or list) will returned by:

```
Employees[2]
```

b) Guess what the result of the following command will be, then check your answer:

```
Employees[[2]][2]
```

Guess what the result of the following command will be, then check your answer:

```
Employees$Salary[2]
```

c) Guess what the following command does, then check your answer:

```
Employees$YearsExperience <- c(2, 4, 9)
```

Could we have accomplished the same thing by typing the following?:

```
Employees[["YearsExperience"]] <- c(2, 4, 9)
```

5.3 Named List Elements

- We can get or add list element names using:

```
names()       # Examine or change the names of list elements
```

- As an example, consider again the list `x`:

```
x

## $x1
## [1] 1 2 3
##
## $x2
## [1] "a" "b" "c"
##
## $x3
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

To get the list element names, we type:

```
names(x)

## [1] "x1" "x2" "x3"
```

and to change them to, say, `y1`, `y2`, and `y3`, we type:

```
names(x) <- c("y1", "y2", "y3")
names(x)

## [1] "y1" "y2" "y3"
```

- To remove the names, we'd type:

```
names(x) <- NULL
```

Section 5.3 Exercises

Exercise 4 Here's a simple list `x`:

```
z <- list(z1 = c(1, 2), z2 = c("a", "b"), z3 = 12)
```

a) What will the following command return? Check your answer.

```
names(z)
```

b) What does the following command do? Make sure to check your answer.

```
names(z) <- c("y1", "y2", "y3")
```