# Intro to R: Week 2

## Topics Covered: Matrices, Data Frames, Lists, and Indexing

Last week we talked about how to store data in vectors, which are one-dimensional objects. With more complex data, we can use matrices, data frames, and lists to store information in multiple dimensions.

- A **matrix** is a collection of elements of the same data type (numeric, character, or logical) arranged in a fixed number of rows and columns. An array is an n-dimensional matrix.
- A **data frame** has variables as columns and observations as rows and can contain elements of multiple data types
- A **list** can contain different kinds of objects (data frames, vectors, matrices, etc.) with different dimensions

**Task 1:** Use the `matrix()` function to create a matrix of the numbers 1 to 100 in ten rows and ten columns. Can you get the numbers to read from left to right instead of from top to bottom?

```
# Matrices are two-dimensional and store one type of data (e.g., numeric)

data <- 1:100

mat <- matrix(data, nrow=10, ncol=10) # by default, R fills in the columns top to bottom

?matrix() # look at the arguments for this function.  By default, byrow=FALSE

mat <- matrix(data, nrow=10, ncol=10, byrow=TRUE)
```

Elements within matrices can be accessed using square brackets in two ways:

- Element-wise indexing: `mat[e]` returns the eth element within the matrix (just like a vector!)
- Row-and-column indexing: `mat[r, c]` returns the element in the rth row and the cth column

**Task 2:** Extract the number 50 from your matrix using both row-and-column and element-wise indexing

```
# First, figure out which element is equal to 50

e <- which(mat==50)

mat[e] # element-wise indexing

mat[5, 10] # row-and-column indexing
```

**Task 3:** Use the functions `colSums()` and `rbind()` to add a row to the bottom of your matrix that contains the mean of the numbers in each column

```
mat <- rbind(mat, colSums(mat))
```

Matrices are great, but most of the time, biologists collect data of several different types, so data frames are more appropriate.

**Task 4:** Use the `data.frame()` function to store the following data (stolen from Sandin et al. 2008). What happens if you store the data in a matrix instead? Why might this be a problem?

| Island | Inhabited | Coral Cover (%) | Fish Biomass (mT ha^-1) |
|---|---|---|---|
| Kingman | No | 44 | 5 |
| Palmyra | No | 20 | 3 |
| Tabuaeran | Yes | 20 | 2 |
| Kiritimati | Yes | 15 | 1 |

```r
islands <- c("Kingman", "Palmyra", "Tabuaeran", "Kiritimati")
inhabited <- c(FALSE, FALSE, TRUE, TRUE)
coral <- c(44, 20, 20, 15)
fish <- c(5, 3, 2, 1)

island.df <- data.frame(islands, inhabited, coral, fish)

island.mat <- cbind(islands, inhabited, coral, fish) # all variables are characters

# it's problematic to have numerical data stored as characters because functions
# like sum() and mean() can't evaluate non-numerical data
```

In addition to element-wise and row-and-column indexing, we can also use column and row names to index within matrices and data frames. `island.mat[,"coral"]` and `island.df[,"coral"]` will return only the coral columns of these objects. Additionally, data frames can be indexed using the `$` to access named columns, so `island.df$coral` also returns the coral column of this data frame.

**Task 5:** What is the average coral cover of islands that are uninhabited?

```r
i <- which(island.df$inhabited==FALSE) # first find the indexes of inhabited islands

c.i.mean <- mean(island.df$coral[i]) # find the mean of % cover on uninhabited islands
```

**Task 6:** Your advisor asks to see only the data from inhabited islands. Use the `subset()` function to extract this information from your data frame.

```r
inhabited.data <- subset(island.df, inhabited==TRUE)
```

**Task 7:** Now imagine that you want to know whether fish biomass is greater on inhabited or uninhabited islands.

*Step 7.1* Calculate the mean fish biomass on inhabited and uninhabited islands

```r
fish.u <- mean(island.df$fish[which(island.df$inhabited==FALSE)])

fish.i <- mean(island.df$fish[which(island.df$inhabited==TRUE)])
```

*Step 7.2* Use an `if()` statement with a logical test on the mean fish biomass calculated above to print a message indicating whether fish biomass is greater on inhabited or uninhabited islands. Your code should have the following structure:

```
if (logical test goes here) {thing to do if logical test is true} else {
                              thing to do if logical test is false
}
```

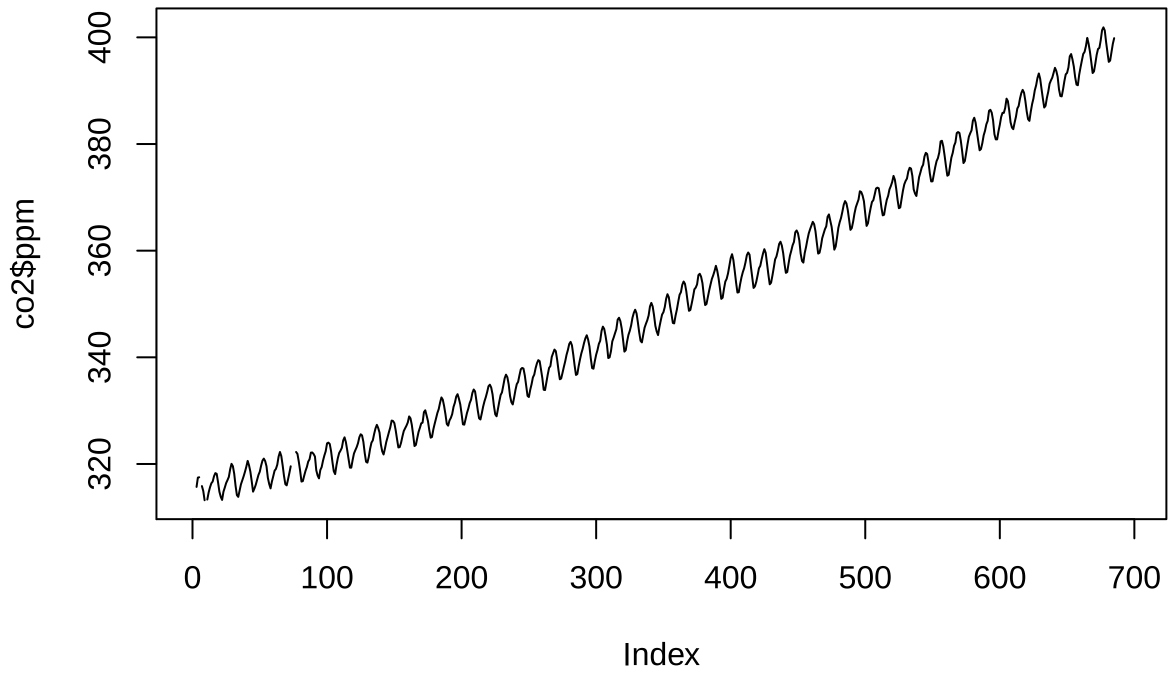
```
if (fish.u > fish.i) {print("Fish biomass on uninhabited islands is greater")} else {
                       print("Fish biomass on inhabited islands is greater")
}
```

**Task 8:** Use the function `read.csv()` to read in the file 'monthly_mlo.csv'.
What kind of data object is it?

```
setwd("~/Desktop/IntroR/Week 2") # Make sure your working directory is set to the file loc

co2 <- read.csv("monthly_mlo.csv")

str(co2)
```

**Task 9:** Not surprisingly, this data set is not perfect. Replace any missing values with NA and use the `plot()` function to create a line graph of CO2 values over time.

```
# Looks like missing months were entered as -99.99

bad <- which(co2$ppm == -99.99)

co2$ppm[bad] <- NA

# There is an easier way to do this . . .

co2 <- read.csv("monthly_mlo.csv", header=TRUE, na.strings=-99.99)

plot(co2$ppm, type="l")
```



Sometimes during research projects we collect many different types of data that aren't necessarily directly linked. For example, perhaps you collect CTD data AND count organisms collected in mid-water trawls AND have logs of whales detected in sonobuoy recordings all from the same location. These data can't be combined into a single data frame, but it is useful to have them all in the same place. This is where **lists** come in handy. Think of them as R databases that can be stored as single objects (.RData files) and can contain a bunch of different information.

**Task 10:** Use the `list()` function to create a list that contains the whale survey data from Week 1, the islands data set we created earlier today, and the Mauna Loa CO2 data. Use the `str()` function to examine the structure of the list.

```r
species <- c("Fin", "Blue", "Humpback", "Minke", "Gray")

whales <- c(5, 2, 14, 0, 1)

names(whales) <- species

data.list <- list("whales"=whales, "islands"=island.df, "co2"=co2)

str(data.list)
```

Lists use a combination of double and single square brackets to access nested elements of a list. The `$` can also be used to access elements of a list. From the output of `str()` you can see how to access elements using `$`.

**Task 11:** There are at least nine different ways to extract the coral column of the islands data set from this list. Can you find three of them?

```r
data.list$islands$coral

data.list$islands[3]

data.list$islands["coral"]

data.list[["islands"]][3]

data.list[["islands"]]["coral"]

data.list[["islands"]]$coral

data.list[[2]][3]

data.list[[2]]$coral

data.list[[2]]["coral"]
```