Basic Data Types and Structures

```
# Not that we'll need them here, but I like to routinely load the
# tidyverse and mosaic packages.
library(tidyverse)
library(mosaic)

# To begin, we'll cover the following five data types: logical
# (TRUE or FALSE), numeric (double), integer, complex, and
# character (string).  Data types associated with dates and
# times are covered under the lubridate package.
# Here are some examples.
answer <- TRUE
answer

## [1] TRUE

# One can check the type of a value by using the class() command.
class(answer)

## [1] "logical"

pi <- 3.14159
pi

## [1] 3.14159

class(pi)

## [1] "numeric"

# Let's assign the value 7 to the variable lucky. What type
# does it have?
lucky <- 7
lucky

## [1] 7

class(lucky)

## [1] "numeric"

# If we want to get an integer, we need to put an "L" after the number.
n <- 7L
n

## [1] 7

class(n)

## [1] "integer"
```

```r
z <- 2 + 3i
z
```

```
## [1] 2+3i
```

```r
class(z)
```

```
## [1] "complex"
```

```r
name <- "Betty"
name
```

```
## [1] "Betty"
```

```r
class(name)
```

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

```r
# Actually, each of the above values is a vector of length 1.
# In general, a vector is a sequence of values all of the same type.
# A vector can be created using the c() function or seq() function.
# Here are some examples:
days_of_week <- c("Sun","Mon","Tue","Wed","Thu","Fri","Sat")
days_of_week
```

```
## [1] "Sun" "Mon" "Tue" "Wed" "Thu" "Fri" "Sat"
```

```r
class(days_of_week)
```

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

```r
# If you haven't already, look under the "Environment" tab
# to see all of the variables we've created thus far.
#
# The length() function can be used to compute the length of a vector;
# for example:
length(days_of_week)
```

```
## [1] 7
```

```r
odds <- seq(1, 9, by = 2)
odds
```

```
## [1] 1 3 5 7 9
```

```r
class(odds)
```

```
## [1] "numeric"
```

```r
# An example using the ":" operator:
digits <- 0:9
digits
```

```
##  [1] 0 1 2 3 4 5 6 7 8 9
```

```r
some_primes <- c(11L,13L,17L,19L)
some_primes
```

```
## [1] 11 13 17 19
```

```r
class(some_primes)
```

```
## [1] "integer"
```

```r
fourth_roots_of_unity <- c(1,0 + 1i,-1,0 - 1i)
fourth_roots_of_unity
```

```
## [1]  1+0i  0+1i -1+0i  0-1i
```

```r
class(fourth_roots_of_unity)
```

```
## [1] "complex"
```

```r
# To access the jth element of the vector v, use v[j].
congruent_to_one_mod_three <- seq(1,22,by = 3)
congruent_to_one_mod_three
```

```
## [1]  1  4  7 10 13 16 19 22
```

```r
congruent_to_one_mod_three[5]
```

```
## [1] 13
```

```r
# Next, we consider some of R's data structures.
# Lists: A list is similar to a vector, except it can be
# heterogeneous, meaning that its elements can have different types.
# The list() command can be used to create a list.
my_list <- list(FALSE,2.718,69L,"blue")
my_list
```

```
## [[1]]
## [1] FALSE
##
## [[2]]
## [1] 2.718
##
## [[3]]
## [1] 69
##
## [[4]]
## [1] "blue"
```

```r
# As the above output suggests, an element of a list may be a
# list itself.  For instance, here's a weird example:
weird_list <- list(1, list(2,3), list(4,list(5)))
weird_list
```

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

```
# Elements of a list are accessed just like vector elements.
my_list[4]
```

```
## [[1]]
## [1] "blue"
```

```
weird_list[2]
```

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

```
# Alternately, the elements of a list may be named, and then
# accessed via their names.
names(my_list) <- c("first","second","third","fourth")
my_list
```

```
## $first
## [1] FALSE
##
## $second
## [1] 2.718
##
## $third
## [1] 69
##
## $fourth
## [1] "blue"
```

```
my_list$second
```

```
## [1] 2.718
```

```
# Matrices: A matrix contains elements of the same type arranged into
# rows and columns.  A matrix is created using the matrix() command,
# giving the vector of elements, the number of rows, and the number
# of columns.  (Also, the rows and columns may be named, but we'll
# skip that feature.)  Here, we create a 3 by 4 matrix:
my_matrix <- matrix(c(7,1,-5,2,-1,-2,-8,2,-9,-1,6,9),3,4)
my_matrix
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    7    2   -8   -1
## [2,]    1   -1    2    6
## [3,]   -5   -2   -9    9
```

```
# Note that the elements were arranged by column, rather than by row.
# The default value of the "byrow" parameter is FALSE.
# If we want the elements to be arranged by row, we set byrow equal
# to TRUE.
my_matrix <- matrix(c(7,1,-5,2,-1,-2,-8,2,-9,-1,6,9),3,4,
                    byrow = TRUE)
my_matrix
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    7    1   -5    2
## [2,]   -1   -2   -8    2
## [3,]   -9   -1    6    9
```

```
# Elements of a matrix are accessed in the obvious way.
my_matrix[2,3]
```

```
## [1] -8
```

```
# R also allows for arrays with any number of dimensions.
#
# Data Frames: The primary structure that we deal with in data science
# is the data frame.  A data frame is similar to a matrix, with the
# columns being named.  Often, a data frame is automatically created
# when data is imported.  A data frame can also be constructed by first
# creating the columns, and then using the tibble() command to
# organize the columns into a data frame.  Here is an example:
name <- c("Curly","Joe","Moe","Shemp")
siblings <- c(6L,4L,2L,3L)
height <- c(74.5,70,59.8,62)
gender <- c("M","M","F","M")
my_data_frame <- tibble(name,siblings,height,gender)
my_data_frame
```

```
## # A tibble: 4 × 4
##   name  siblings height gender
```

```
##   <chr>    <int>  <dbl> <chr>
## 1 Curly        6   74.5 M
## 2 Joe          4   70   M
## 3 Moe          2   59.8 F
## 4 Shemp        3   62   M

# Notice that the type for each column is indicated.
# Extra Credit: Since there are a limited number of possible values
# for the gender variable, we might want to make it a "factor".
# We can do this as follows:
my_data_revised <- my_data_frame %>%
  mutate(gender = as.factor(gender))
my_data_revised

## # A tibble: 4 × 4
##   name  siblings height gender
##   <chr>    <int>  <dbl> <fct>
## 1 Curly        6   74.5 M
## 2 Joe          4   70   M
## 3 Moe          2   59.8 F
## 4 Shemp        3   62   M

# A subsequent tutorial is devoted to data frames.
```