Worksheet 2

Data Types

Lab worksheet: Data types

Data types

This brings up data types. Data types describe what kind of data can be held by a variable.

Numeric. So far you've mostly used numeric data types which represent real numbers.

Logical. In the last exercise you created logical variables which can have the value `TRUE` or `FALSE` (equivalently `T` or `F`, or alternatively `1` or `0`).

1. Use logical variables to
   a. store the value of each logical expression from worksheet 1, exercise 10 a-d in variables L1, L2, L3, and L4
   b. find the number of those variables which are true using addition in R

String. Another useful data type is a string, which is a string of characters. In R strings are annotated in quotes (either single or double will do). For example,

```
> greeting = "hello"
```

2. Create the following strings
   a. Reset `greeting` to some other way to say hello
   b. a string named `class` which holds the value FGMM
   c. a string named `student` set to be your first name
   d. a string named `leader` set to be empty

Comparing data types. Note that strings and numerics are not equivalent. Define these variables in your R session

```
> sixWord = "six"
> sixNumber = "6.0"
> sixNumeric = 6.0
```

Check the values of these variables by typing their names into the console. Note that the presence of quotation marks indicates that this is a string, rather than a numeric.

3. Now use logical operators to see if each pair is equivalent
   a. `sixWord` and `sixNumber`
   b. `sixWord` and `sixNumeric`
   c. `sixNumber` and `sixNumberic`

However, you can convert from one type to another using the functions `as.numeric()`, `as.character()`, `as.logical()`, etc. For example,

```
> as.character(sixNumeric)
[1] "6"
```

4. Convert `sixNumber` into a numeric and use a logical operator to compare the result to `sixNumeric`.
5. But there is a limit. Try to convert `sixWord` to a numeric.

Data types (multi-element)

Vector. Often, you want to store many related values under a single name. There are many data types which can do this in R. We'll start with vectors. A vector is an indexed list of variables. Vectors can be created with the `c` function (`c` stands for concatenate), for example

```
> fibSeq = c(1, 1, 2, 3, 5, 8, 13)
[1]  1  1  2  3  5  8 13
```

Again, you can use the console to see the values stored in vectors.

6. Create the following vectors
   a. a vector called `circleFigs` that contains the values of `pi`, `r`, and `circumference`
   b. a vector called `mostlyTrue` that contains 3 true logical variables and one false
   c. a vector named `class` which contains the first names of at least five students in the class (get to know your neighbors!)

Tip, to make a vector of consecutive whole numbers, you input the endpoints separated by a colon. For example,

```
> oneToTen = 1:10
> oneToTen
 [1]  1  2  3  4  5  6  7  8  9 10
```

Vectors are indexed, which means that the individual elements are each labeled with sequentially increasing numbers. The following schematic shows how indexing works for the `fibSeq` vector we created earlier.

`fibSeq`

index	1	2	3	4	5	6	7
value	1	1	2	3	5	8	13

Individual values in vectors can be retrieved using square brackets and the index number. For example, using the console you can get the fourth number in `fibseq` with

```
> fibSeq[4]
[1] 3
```

7. Use the vectors you created to
   a. check the value of the third element of `circleFigs`
   b. save the first element in class to the variable `firstStudent`

To get a subset of a vector, you can specify the indicies you're interested in. For example, to get the last three elements of `fibseq`

```
> fibSeq[5:7]
[1]  5  8 13
```

8. Use the vectors you created to
   a. check the first two values of `mostlyTrue`
   b. check the middle five values of `fibSeq`

To get the length of a vector, you use the function `length()` (sometimes things make sense). For example,

```
> length(fibSeq)
[1] 7
```

9. Use the length function to
   a. check the length of `mostlyTrue`
   b. save the length of `class` to the variable `nStudents`

Matrix. Sometimes you might have a whole table of data that you want to save. Again, there are a number of ways to do this in R. First we'll consider matrices, which are indexed like vectors, but are two-dimensional.

To create a matrix, you use the function matrix and specify the values, number of rows, and number of columns. For example

```
> fibMat = matrix(data=fibSeq, nrow=3, ncol=2)
Warning message:
In matrix(data = fibSeq, nrow = 3, ncol = 2) :
  data length [7] is not a sub-multiple or multiple of the
  number of rows [3]
```

Ahl, a warning message! But first, check the value of `fibmat` (use the console) and notice that you made a 3x2 matrix (remember we talk about matrix dimensions with rows first always (rows then columns, like RC cola for those inclined)) with the `fibseq` values entered one row at a time.

So let's see what this matrix looks like

```
> fibMat
     [,1] [,2]
[1,]    1    3
[2,]    1    5
[3,]    2    8
```

To access those values, you use square brackets with the row and column indexes. For example, to get the element in the third row, first column, you type in

```
> fibMat[3,1]
[1] 2
```

10. Use the matrix you created to
    a. save the bottom right element in the variable `lastFib`
    b. check if the first element of the first is equal to the first element of the second row
    c. check if the first element of the second row is equal to the second element of the second row

Now back to that warning message. Warning and error messages can be startling, but are very useful! Warning messages alert you to something that may be a problem, but didn't make your last command impossible. Error messages tell you what happened that made your command impossible.

11. What is this warning message telling us?
12. How would you go about changing our `fibMat` declaration so that it doesn't produce an error message?

And returning to matrices, to find out the dimensions of a matrix you can use the function `dim()` (similar to how we used `length()` for a vector).

13. Use the `dim()` function to check the dimensions of `fibMat`.

If you want to get a 'slice' of a matrix, you can use an indefinite index to specify a number of values. Indefinite indexes are just blank indexes, indicating all values. For example, to get the first row of `fibMat`, we can use

```
> fibMat[1,]
[1] 1 3
```

14. Using `fibMat`,
    a. access the second column of the matrix
    b. multiply the middle row of the matrix by 6

Arrays. Another way to store tables of numbers in R is in arrays. Arrays are a lot like vectors and matrices, but they can have any number of dimensions. To create an array you use the function `array()`. This time, you look up how to use the function `array()` by typing `?array` into the console to see the manual page.

15. Create the 3x2x4array `fibArr` that contains the first 24 numbers in the Fibonacci sequence. Hint: those are 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10966, 17751, 28717, 46468

Data frames. R provides data frames as another useful way to store data of different types. Typically, data frames are tables with named rows and columns containing heterogenous types of data (for examples, strings, numerics, and logical variables). For example, consider the following data about the Fibonacci sequence

number	odd	flower
1	TRUE	callalilly
1	TRUE	callalilly
2	FALSE	euphorbia
3	TRUE	trillium
5	TRUE	columbine
8	FALSE	bloodroot
13	TRUE	blackEyedSusan

We can store this in a data frame as

```
> FibDF = data.frame(number = c(1, 1, 2, 3, 5, 8, 13), odd =
c(TRUE, TRUE, FALSE, TRUE, TRUE, FALSE, TRUE), flower =
c("callaLilly", "callaLilly", "euporbia", "trillium",
"columbine", "bloodroot", "blackEyedSusan"))
```

Then attributes from the dataframe can be accessed either using coordinates as in matricies and arrays, or by identifying the attribute by name. For example, to get the flower names

```
> FibDF$flower
```

16. Practice skills with data frames by
    a. Create your own dataframe `studentDF` with the first names, zip code, and birth month of at least five students in the class.
    b. Referencing the data frame manual page (`?data.frame`), create a new version of your data.frame where the row names are set to student last names.
    c. Access the alphabetically first student entry.
    d. Save the zip codes in a new variable called `zips`.

Lists.

To make things more complex, R has another data type for series of things: lists. We won't be using lists in class much, but it's useful to know what they look like in case you run into one. The elements of lists can be mixed types. For example

```
> mixedList = list(c(1,1,2,3,5,8), 42, c("Oakland", "Hayward",
"SF"), TRUE)
> mixedList
[[1]]
[1] 1 1 2 3 5 8

[[2]]
[1] 42

[[3]]
[1] "Oakland" "Hayward" "SF"

[[4]]
[1] TRUE
```

Lists are much like vectors, however the elements of the list are accessed with double square brackets. For example, to access the list of cities in `mixedList`

```
> mixedList[[3]]
[1] "Oakland" "Hayward" "SF"
```