Lesson one: Values

Brian S. Evans, Ph.D.

Introduction

Many early R users find the process of learning R a grueling uphill climb. I believe that one of the biggest impediments to learning R stems from studying applications of the program without taking the appropriate time to learn the language itself. Developing a flexible R skillset, and thus being able to apply what you learn in this course, requires us to better understand R as a language and develop new ways of thinking about data.

In this lesson we will begin our exploration of R as a language by learning the primary types of values that users interact with. A **value** is any single unit of data. A value can be described by its "**class**", which describes what kind of value it is, and "**type**" which describes how the value is stored in R. The most commonly used classes of values are:

- **Numbers**: Numbers can be double (e.g., 1.123) or integer (e.g., 1, 2, 3)
- **Characters**: Words or symbols (e.g., "hello")
- **Factors**: Symbols or words assigned to integer values (e.g., "hello" = 1, "world" = 2)
- **Logical**: Integer values of 0 and 1 assigned to FALSE and TRUE

Please complete the sections of this lesson in order. Even if some elements of this lesson are very easy for you, it is important that each step is completed. Additionally, I ask that the steps are completed with the methodology shown – the goals of this introductory material is to provide early R users with foundational skills while ensuring that advanced users understand some of the nuances of the R language.

A review

The function "combine", `c()`, can be used to combine multiple values into a single object. For example, we can use to combine the numbers 5, 8, and 13 into an object you would type `c(5, 8, 13)`.

You can store the object in memory by assigning a name to the object using the assignment operator `<-`. For example, assign the name "bunnies" to the object by typing `bunnies <- c(5, 8, 13)`.

To print the R object in your Console panel, you can simply type the name of the object. For example, we can view the values of the object named “bunnies” by typing `bunnies` in the R console.

Please complete the following steps to generate an R object and store that object in your system’s memory:

- Create an object that contains the numbers 1, 1, 2, and 3
- Store the object in your system’s memory by assigning the name “numberObject”
- Print the R object

The function `mean()` can be used to calculate the average of number values within an object. We will use this function, in conjunction with the `numberObject`, to review non-nested and nested code structures.

Calculate the `mean` of the values stored in the object `numberObject`:

Because `numberObject` is simply the name assigned to the object `c(1, 1, 2, 3)`, we can write the above using a nested coding structure by substituting `numberObject` with `c(1, 1, 2, 3)`.

Use a nested coding structure to calculate the `mean` of the values `c(1, 1, 2, 3)`:

Numbers

There are two basic kinds of numbers in R:

- **Numeric, double**: For our purposes, this can be thought of simply as a decimal number
- **Integer**: whole numbers

The functions `class()` and `typeof()` are used to to determine the **class** and **type** of a value, respectively.

Use the functions `class` and `typeof` to explore the attributes of `numberObject`.

If you specify a range of whole numbers, using the notation `min:max`, you can generate integer values.

Please use a nested coding structure to complete the following:

- Generate an ordered sequence of whole numbers from 1 to 3 using the `:` operator and use `class()` and `typeof()` to explore object.
- Use `c()` to generate an ordered sequence of whole numbers from 1 to 3. Explore the object using `class()` and `typeof()`.
- How do the two objects differ?

We can coerce double numeric values into integer numeric values using `as.integer()`. This is only safe, however, if the number that you are converting to an integer is a whole number!

Use `as.integer()` to convert `numberObject` from numeric double to integer values. Use the `class()` and `typeof` functions to describe the object.

- Complete this process using an object assignment coding structure.
- Complete this process using a nested coding structure.

Similarly, we can use the function `as.numeric()` to convert a value to a double numeric value.

Use a nested coding structure and `as.numeric()` to convert a set of integers to double numeric values.

Describe the object using a nested coding structure and the `class()` and `typeof()` functions.

If you've clicked on the hints, you may have noticed that there is a specific, and maybe strange, format to my code. The format I use represents best management practices in coding. You can enhance the readability of your code by:

- Assigning a name to an object in its own line
- Writing each step of a nested function on a separate line

Characters

A **character** or “string” value is a symbol or set of symbols from a given alphabet (note: also includes numbers and punctuation). The type and class of character values are both “character”.

Whenever you are working with characters, you specify the character values within single or double quotation marks. For example, “hello world” would be written in R as `'hello world'` or `"hello world"`.

Please use a nested coding structure to complete the following:

- Create an object that contains the months, “March”, “February”, “January”, and “January”.
- Explore the **class** and **type** of the object using a nested coding structure .

Factors

A **factor** is a value that includes the following information:

- **Integer value**: Numeric integer value associated with factor level. For example, you may construct a factor of female and male treatment groups. In this instance, each treatment group would be assigned to a different integer value.
- **Levels**: Character values that are associated with the integer value. Continuing with the example above, the levels would be female and male.
- **Labels**: Characters to assign to each factor level. This relates to the character values that you see when you print or plot your data. For example, you may want to visualize female as “F” and males as “M”.

Why would you use factors? You would use factors in instances in which data can be grouped into one of a few (or several) values. For example, banded birds may be grouped into females and males. Factors also may be useful if you are interested in specifying the order that values are displayed. For example, the days of the week, in alphabetical order, are Friday, Monday, Saturday, Sunday, Thursday, Tuesday, and Wednesday. To specify the days of the week in date order, you can change the object into a factor (as below).

Factors are generated using the function `factor()`. The first argument to provide to `factor()` is a set of character values that you would like to structure as factor variables.

Create a factor that contains the words, “one”, “one”, “two”, and “three”. Assign the name `exampleFactor` to the resultant object.

Explore the **class** and **type** of the object using object assignment.

Notice that the type of object is integer??? Remember that this lets us know how R is storing the object internally. Any guesses as to why R would store the object as a integer?

The function `levels()` provides us with information on the levels of a factor. Use `levels()` to view the factor levels of `exampleFactor`.

The object `exampleFactor`, as written, might cause us some trouble because levels are ordered alphabetically by default. To see why this might be a problem, let’s use a new function, `plot()` to represent the object graphically. The first argument of `plot()` is the data that you evaluating (see `?plot`). When a factor object is plotted, the default output is the count of records (y-axis) at each factor level.

Use the `plot()` function to observe why this might be an issue.

To address this, levels can be manually set, as `levels` can be used as one of the arguments for the `factor` function (see `?factor`). To do so, provide the set of unique characters (`c(...)`) in the order that you would like them to be arranged. Recall that arguments of functions are separated by a comma (e.g., `myFunction(arg1 = x, arg2 = y)`).

Create a factor that contains the words, “one”, “one”, “two”, and “three”. Use the `levels` argument of `factor()` to provide levels in sequential number order. Do not assign a name to the resultant object.

Using a nested coding structure, plot the releveled factor:

Likewise, labels can be manually set if those that are automatically assigned are not ideal. Labels can be set manually using the `labels` argument of the `factor` function. To do so, provide the set of unique characters (`c(...)`) in the order of the factor `labels` . Recall that arguments of functions are separated by a comma (e.g., `myFunction(arg1 = x, arg2 = y)`)

 Create a factor that contains the words, “one”, “one”, “two”, and “three”. Use the `levels` argument of `factor()` to provide levels in sequential number order. Do not assign a name to the resultant object. Use the `labels` argument of the `factor` function to capitalize the factor labels.

Using a nested coding structure, plot the releveled factor with the nicer-looking labels:

 Factors are very useful (especially for statisticians!), but can also be a bit of a pain to deal with. If you are working with a factor that has an untenable number of levels, consider simplifying the factor or using character values instead. At one time, evaluating characters required much more memory to process, but this is no longer the case. I recommend avoiding factors unless they are necessary for the application that you are working on.

Logical values

R reserves the words TRUE and FALSE as logical constants. These constants are mapped to integer values:

- **FALSE**: 0
- **TRUE**: 1

 To better understand logical values, take a moment to run and explore the output of the code below.

```
FALSE

TRUE

as.numeric(FALSE)

as.numeric(TRUE)

mean(
  c(FALSE, TRUE, TRUE)
  )

FALSE + TRUE

FALSE + TRUE + TRUE
```

Logical values can be obtained by evaluating objects with **logical operators**. Logical operators in R include those displayed in the table below. We will explore logical operators in-depth in a future lesson.

Operator	Usage	Meaning
==	x == y	x is equal to y
!=	x != y	x is NOT equal to y
!	!(x)	not x
|	x | y	x OR y
&	x & y	x AND y
%in%	x %in% yz	x is in the vector c(y,z)
<	x < y	x is less than y
<=	x <= y	x is less than or equal to y
>	x > y	x is greater than y
>=	x >= y	x is greater than or equal to y

Use the logical operator `==` to test whether each set of values is equivalent ($\equiv$). For example, the first set of expressions would be written as `3 == 3`.

$3 \equiv 3$

$3 \equiv 4$

$3 \equiv 2 + 1$

$3 \equiv 3 + 1$

$2 + 2 \equiv 3 + 1$

$(3 \equiv 3) + (3 \equiv 2 + 1)$

Term review and glossary

Functions and operators

- `as.integer` Convert an object from double numeric to an integer
- `as.numeric` Convert an object from integer to double numeric
- `c` Combine objects
- `class` Determine the class of an object
- `factor` Make a factor object
- `levels` Determine the levels of a factor object
- `mean` Take the average of a set of values (note, use na.rm = TRUE in the presence of NA's)
- `plot` Plot an object, used in this exercise to plot a factor
- `typeof` Determine how R stores an object in your global environment
- `<-` Assign a name to an object in R (assignment operator)
- `?` Get help for a given function

Logical operators

Operator	Usage	Meaning
==	x == y	x is equal to y
!=	x != y	x is NOT equal to y

Operator	Usage	Meaning
|	x | y	x OR y
&	x & y	x AND y
<	x %in% yz	x is in the vector c(y,z)
<=	x < y	x is less than y
%in%	x <= y	x is less than or equal to y
>	x > y	x is greater than y
>=	x >= y	x is greater than or equal to y

Vocabulary

- **Character**: a symbol or set of symbols from a given alphabet (note: also includes numbers and punctuation)
- **Factor**: a value in which characters are mapped to integer levels and labels
- **Integer**: a whole number
- **Labels (of a factor)**: character values associated with a factor level when printing or plotting factors
- **Levels (of a factor)**: character values mapped to integer values
- **Logical operator**: Operators that are used to test logical statements (e.g., 3 > 2)
- **Logical value**: TRUE or FALSE
- **Numeric, double**: For our purposes, a decimal number
- **Type**: how a value is stored in R (e.g., double or integer)
- **Value**: any single unit of data

End of lesson!