Coding Lab: Basic Syntax and Operators in R

Ari Anisfeld, Angela Li

10/1/2019

Why are we here?

The purpose of this Coding Lab is to:

1. introduce you to basic programming concepts
2. prepare you to do data analysis in R for the future (additional courses, your next job)
3. give you the coding skills necessary to complete your Stats I homework

We will do things that produce errors and then talk about how to read the error and fix it.

Survey time

- How many people have R and RStudio installed on their computers?
- How many people know how to install packages and load libraries in R?
- How many people have experience with a programming language of any type?

Some tips for computational reasoning

1. Make small tweaks to the code to see if you understand how the code works - be hands-on!
2. Be lazy: usually someone else has had your problem and developed a solution.
3. Find help: Check the documentation with ?

Variable assignment

We use <- for assigning variables in R.

```
my_number <- 4
my_number
```

```
## [1] 4
```

The reference is a pointer to the object and can be set to a different value. What will the following expressions return?

```
my_number + 3
```

```
my_number * 3
```

```r
my_number + 3
```

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

```r
my_number * 3
```

```
## [1] 12
```

Exercise: variable assignment

Based on the previous exercise, what value will this code return? What's the value of `my_number` after we run both lines of code?

```
my_number <- 2
(12 ^ my_number) + 1
```

Data types

R has several types which distinguish how R stores data internally. Some of the most common ones are logical, numeric, and character types.

```
type_logical <- FALSE
type_logical <- TRUE

type_double <- 1.0
type_integer <- 1L

type_character <- "abbreviated as chr, also known as a stri
```

Type issues

Sometimes you'll run into problems because the data is not of the type you expect. For example, you may import data where something that look likes a number is a string.

```
a <- "1"
b <- 2
```

What will happen when we try to run the following code?

```
a + b
```

```
Error in a + b : non-numeric argument to binary
operator
```

To avoid errors, we can check for the type of the data we have with `typeof()`:

```
typeof(a)
```

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

```
typeof(b)
```

```
## [1] "double"
```

We can also use the `class()` function

```
class(a)
```

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

```
class(b)
```

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

The error we got when we tried `a + b` was because `a` is a character. We can reassign types on the fly:

```
as.integer(a) + 3
```

```
## [1] 4
```

```
as.numeric(a) + 3
```

```
## [1] 4
```

Some type reassignment is done by R automatically:

```
paste0(a, as.character(b))
```

```
## [1] "12"
```

```
paste0(a, b)
```

```
## [1] "12"
```

Exercise: type coercion

Operators

R is also a calculator! We can do math with numbers, using the following symbols:

```
4 + 4
```

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

```
4 - 4
```

```
## [1] 0
```

```
4 * 4
```

```
## [1] 16
```

```
4 / 4
```

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

```
4 ^ 4
```

```
## [1] 256
```

Operators pt. 2

We can also compare things.

```
4 < 4
```

```
## [1] FALSE
```

```
4 >= 4
```

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

```
4 == 4
```

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

```
4 != 4
```

```
## [1] FALSE
```

Operators. . . applied to strings?

What do you think is being compared in this case?

```
"four" == "four"
```

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

```
"four" == 4
```

```
## [1] FALSE
```

Exercise: comparing a string and a number

What do you think will happen when you run the following code?

```
"4" == 4
```

Data structures

Usually we want to keep track of more than one thing at time, say a list of numbers. Computer scientists refer to these as data structures. The main workhorse data structure in R is the vector.

```
my_numbers <- c(1, 2, 3, 4, 5, 6)

my_numbers
```

```
## [1] 1 2 3 4 5 6
```

Adding to vectors

We can do math with vectors!

```
my_numbers + my_numbers
```

```
## [1]  2  4  6  8 10 12
```

```
my_numbers + 6
```

```
## [1]  7  8  9 10 11 12
```

How would the judges rank this lesson right now?

```
my_numbers / c(.1, .2, .3, .4, .5, .6)
```

```
## [1] 10 10 10 10 10 10
```

What will the vector a look like? Remember, `my_numbers` is a list of numbers from 1 to 6.

```
a <- my_numbers + c(1, 2)
```

```
a
```

```
## [1] 2 4 4 6 6 8
```

Using functions with vectors

Some functions act directly on a vector in a pleasing way:

```
sum(a)
```

```
## [1] 30
```

```
length(a)
```

```
## [1] 6
```

Example: Using vectors to calculate a fraction

Work up to this nice representation of the sum of powers of one-half.

```
numerator <- rep(1, 10)
denominator <- 2 ^ c(0:10)

sum(numerator/denominator)
```

```
## Warning in numerator/denominator: longer object length i
## shorter object length
```

```
## [1] 1.999023
```

What we learned

Today we discussed basic syntax and operators in R. You now know how to:

- Assign data to a variable for future reference.
- Distinguish between types
- Working with atomic vectors, the basic data structure of R

Up next week: data frames (aka, rectangular data)!