Session 1

Nick Hathaway; nicholas.hathaway@umassmed.edu

Contents

R Basics

R is like most programming languages and operates by storing by data in `variabes/objects` and then operating on these objects with `functions`.

Variables/objects

Objects are able to store a very wide variety of data, some of the common types are explained below. To store data in R you can use two separate syntaxes, `<-` or `=`. The normal R convention is to use `<-` and there are some real subtle differences between using `<-` and `=` but for the majority of time you can get by by using either one.

```
x = 10
print(x)
```

```
[1] 10
```

```
#same as
y <- 10
print(y)
```

```
[1] 10
```

Functions

Functions are used on objects that have stored data to either output new data or to simply print information about the data stored in that object (e.g. above the function `print()` will print to the screen the data stored in the object that is given to print). You can store the output of a function into a new variable.

```
x = 9
sqrt(x)
```

```
[1] 3
```

```
y = sqrt(x)
print(y)
```

```
[1] 3
```

When people talk about functions, they will refer to the objects being given to the functions as the function's arguments. Depending on the function, it can take several different arguments. To find out more about a function and the functions it takes, you can use the `help()` function and give it the name of function or you can use `?` and then name of the function. If you don't know the name of the function exactly or want to do a keyword search you can `??` and then a keyword. You can also just go to the bottom right window in RStudio and go to the help tab.

```
help(print)
```

or

```
?print
```

or to search for any topic with the word `print`

```
??print
```

Also Google is your best friend as well.

There are a large number of different functions in R that you will become accustomed to as you use R more. Also each library you load will have more functions as well. Also the arguments to functions have names and rather than giving arguments simply in the order in which they are listed in the function's definition you can give an argument by using `arg=value` syntax. For example take the `seq()` function, which is a function for creating different ranges. First call `help(seq)` to see the definition of `seq()`, whose definition looks like below.

```
seq(from = 1, to = 1, by = ((to - from)/(length.out - 1)),
    length.out = NULL, along.with = NULL, ...)
```

So using the seq function you can create a range from `2` to `4` by typing

```
print(seq(2,4))
```

```
[1] 2 3 4
```

or you can give the arguments by naming them

```
print(seq(from = 2, to = 4))
```

```
[1] 2 3 4
```

When naming the function arguments, order no longer matters

```
print(seq(to = 4, from = 2))
```

```
[1] 2 3 4
```

When naming the arguments you don't have to name all of them

```
print(seq(2,4, length.out = 6))
```

```
[1] 2.0 2.4 2.8 3.2 3.6 4.0
```

Other Resources

The markers of RStudio have a series of free webinars you can watch here, https://www.rstudio.com/resources/webinars/. RStudio also has several links to cheatsheets if you go to Help -> Cheatsheets.

Data Types

Every object in R will have a different data type. To determine the type of any object just use the `class()` function.

```
x <- 10
print(class(x))
```

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

```
y <- "Example"
print(class(y))
```

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

Depending on the type of data, functions will have different behaviors

character

character type data is anything that can be represented by a string of letters/characters like a name or categorical data.

```
name = "Nick"

loc = "Goff Auditorium"

condition = "Control"
```

numeric

numeric type data is anything that be represented by, well, numbers like measurements or data values

```
speed = 10

time = 60

fraction = 0.5
```

factor

factor type data is basically character data that has been encoded into numeric values underneath but is represented by characters, this mostly used on categorical data that needs to be converted into numbers in order for certain modeling/statistics functions to work. More on factors latter.

logic/Boolean

Boolean refers to a type of data is simply either TRUE or FALSE and is normally used in conjunction with logic.

```
x = 10 > 11
print(x)
```

```
[1] FALSE
```

```
y = 12 > 11
print(y)
```

```
[1] TRUE
```

The Boolean values can use be set directly using `TRUE` and `FALSE` and `T` and `F` are also short hand for these

```
aTrueValue = TRUE
print(aTrueValue)
```

```
[1] TRUE
```

```
alsoTrueValue = T
print(aTrueValue)
```

```
[1] TRUE
```

```
aFalseValue = FALSE
print(aFalseValue)
```

```
[1] FALSE
```

```
alsoFalseValue = F
print(alsoFalseValue)
```

```
[1] FALSE
```

Logic Tests

The majority of the available logical tests are below.

operator	meaning
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	exactly equal to
!=	not equal to

vectors

Vectors are just several of the same data type hold together in one container, the most common way to create a vector is to use the concatenate function, which in R is just called `c()` for short.

```
#numeric values
speeds = c(10.5, 11, 13, 14, 10)
print(speeds)
```

```
[1] 10.5 11.0 13.0 14.0 10.0
```

```
print(class(speeds))
```

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

```
times = c(20,30,20,30,40,50)
print(times)
```

```
[1] 20 30 20 30 40 50
```

```
print(class(times))
```

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

```
#charater values

names = c("Nick", "Jake", "Michael", "Elisa")
print(names)
```

```
[1] "Nick"    "Jake"    "Michael" "Elisa"
```

```r
print(class(names))
```

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

```r
#Boolean/logical values

logics = c(10 < 11, 12 < 11, 10 > 9)
print(logics)
```

```
[1]  TRUE FALSE  TRUE
```

```r
print(class(logics))
```

```
[1] "logical"
```

You can also create integer ranges using the colon : symbol

```r
range1 = 1:5
print(range1)
```

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

```r
range2 = 10:20
print(range2)
```

```
 [1] 10 11 12 13 14 15 16 17 18 19 20
```

```r
#you can also reverse the direction
rev_range2 = 20:10
print(rev_range2)
```

```
 [1] 20 19 18 17 16 15 14 13 12 11 10
```

Note: R will force everything in a container to be the same type if it can so be careful to not to actually mix types if you don't mean to.

```r
#Accidental converion to character rather than numberic vector
numbers = c(1,2,3,4,5,"6")
print(numbers)
```

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

```r
print(class(numbers))
```

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

```r
#actual numeric vector
actualNumbers = c(1,2,3,4,5,6)
print(actualNumbers)
```

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

```r
print(class(actualNumbers))
```

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

Math operators

Several of the math operators available in R are as follows, `+`, `-`, `*`, `/`, and `^`.

```r
3 + 4
```

```
[1] 7
```

```r
4 - 2
```

```
[1] 2
```

```r
2*120
```

```
[1] 240
```

```r
1e3/2
```

```
[1] 500
```

```r
3^2
```

```
[1] 9
```

These can also be applied to vectors of numbers all at once as well

```r
c(1,2,3,4) * 2
```

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

```r
c(1,2,3,4) / 4
```

```
[1] 0.25 0.50 0.75 1.00
```

```
c(1,2,3,4) + 1
```

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

```
c(1,2,4,5) ^ 2
```

```
[1]  1  4 16 25
```

And of course these can also be applied to the stored numbers in variables

```
x = 2

x * 2
```

```
[1] 4
```

```
y = c(2,4,6,8)

# divide all the numbers in the vector y by 2
y/2
```

```
[1] 1 2 3 4
```

```
#multiple all the values in the vector y by the value stored in x
y * x
```

```
[1]  4  8 12 16
```

Common Basic Stat Functions

There are also a multitude of statistical functions available in R including some of the basics functions like `max`,`min`,`mean`, and `sum`. These are mostly performed on vectors.

```
y = c(2,4,6,8)

mean(y)
```

```
[1] 5
```

```
min(y)
```

```
[1] 2
```

```
mean(y)
```

```
[1] 5
```

```r
sum(y)
```

```
[1] 20
```

Converting between data types

There are sometimes a need to convert between data types, the most common being converting character to numeric types and factor to character types. There are several functions available for this which all start with `as.`

Characters to numeric

```r
x = c("1", "2", "3", "4")
print(x)
```

```
[1] "1" "2" "3" "4"
```

```r
print(class(x))
```

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

```r
x = as.numeric(x)
print(x)
```

```
[1] 1 2 3 4
```

```r
print(class(x))
```

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

Characters to factors

```r
y = c("group1", "group2", "group3", "group1", "group2", "group3")
print(y)
```

```
[1] "group1" "group2" "group3" "group1" "group2" "group3"
```

```r
print(class(y))
```

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

```r
y = as.factor(y)
print(y)
```

```
[1] group1 group2 group3 group1 group2 group3
Levels: group1 group2 group3
```

```r
print(class(y))
```

```
[1] "factor"
```

```r
#levels is function specifically for factors that prints out the different levels stored for the factor
print(levels(y))
```

```
[1] "group1" "group2" "group3"
```

Factors to characters

```r
z = factor(c("1", "2", "4", "5"))
print(z)
```

```
[1] 1 2 4 5
Levels: 1 2 4 5
```

```r
print(class(z))
```

```
[1] "factor"
```

```r
z = as.character(z)
print(z)
```

```
[1] "1" "2" "4" "5"
```

```r
print(class(z))
```

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

Factors to numeric

Now it is very important when converting from factors to numeric to first convert to character type, this is because factors are special data type that underneath are actually numbers that have names associated with these numbers and so the `as.numeric` function actually converts factors to these underlying numbers and not to the numeric equivalent of the character

```r
z = factor(c("1", "2", "4", "5"))
print(z)
```

```
[1] 1 2 4 5
Levels: 1 2 4 5
```

```r
print(class(z))
```

```
[1] "factor"
```

```
#not quite what you would expect!
z = as.numeric(z)
print(z)
```

```
[1] 1 2 3 4
```

```
print(class(z))
```

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

```
z = factor(c("1", "2", "4", "5"))
print(z)
```

```
[1] 1 2 4 5
Levels: 1 2 4 5
```

```
print(class(z))
```

```
[1] "factor"
```

```
z = as.numeric(as.character(z))
print(z)
```

```
[1] 1 2 4 5
```

```
print(class(z))
```

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

Part 1 Excercises

1. Like in algebra, parentheses can be used to specify the order of operations. What then would you expect to be the result of the following expressions, knowing the order of operations from exercise 1? (Try to predict the answer before typing the code into R.)

```
1 + 3*3
```

```
[1] 10
```

```
(1 + 3)*3
```

```
[1] 12
```

```
2^4/2 + 2
```

```
[1] 10
```

```r
2^4/(2 + 2)
```

```
[1] 4
```

```r
(5 + 2*10/(1 + 4))/3
```

```
[1] 3
```

2. Predict the vector and its class resulting from the following expressions:

```r
c(1, 3, 5)
```

```
[1] 1 3 5
```

```r
c("a", "b")
```

```
[1] "a" "b"
```

```r
c(TRUE, TRUE, TRUE, FALSE)
```

```
[1]  TRUE  TRUE  TRUE FALSE
```

```r
c(1, TRUE, 10)
```

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

```r
c("a", FALSE, 100, "dog")
```

```
[1] "a"     "FALSE" "100"   "dog"
```

```r
c(as.numeric(TRUE), "fish", 2, "fish")
```

```
[1] "1"    "fish" "2"    "fish"
```

```r
c(6, 7, as.numeric(FALSE), as.numeric("hello"))
```

```
[1]  6  7  0 NA
```

```r
as.logical(c(1, 0, 10, -100))
```

```
[1]  TRUE FALSE  TRUE  TRUE
```

```r
as.logical(c("TRUE", "false", "T", "F", "True", "red"))
```

```
[1]  TRUE FALSE  TRUE FALSE  TRUE    NA
```

```r
as.numeric(as.logical(c(10, 5, 0, 1, 0, 100)))
```

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

3. Predict the result of the following expressions:

```r
1 > 3
```

```
[1] FALSE
```

```r
14 >= 2*7
```

```
[1] TRUE
```

```r
"1" > "3"
```

```
[1] FALSE
```

```r
as.logical(10) > 2
```

```
[1] FALSE
```

```r
0 == FALSE
```

```
[1] TRUE
```

```r
0 == as.character(FALSE)
```

```
[1] FALSE
```

```r
0 == as.character(as.numeric(FALSE))
```

```
[1] TRUE
```

```r
as.character(0) == 0
```

```
[1] TRUE
```

```r
TRUE == 1^0
```

```
[1] TRUE
```

```r
as.numeric(TRUE) == as.character(1^0)
```

```
[1] TRUE
```

```r
as.numeric("one") == 1
```

```
[1] NA
```

```r
# These are some "bonus" concepts. How does R compare character values?
# Make some predictions, then run the code and see if you can figure out
# the rules for yourself. Then write your own expressions to test the rules!
"a" < "b"
```

```
[1] TRUE
```

```r
"a" < "1"
```

```
[1] FALSE
```

```r
"a2" > "a1"
```

```
[1] TRUE
```

```r
"aaa" > "aa"
```

```
[1] TRUE
```

```r
"a" > "A"
```

```
[1] TRUE
```

```r
as.character(as.numeric(TRUE)) > FALSE
```

```
[1] FALSE
```