Lecture 4 – *R* Refresher

Learning Objectives:

1. **Become proficient in the use of the *R* language.**

    1.1 Understand the basic syntax of *R*, the organization of the workspace and environment.

    1.2 List the data types available in *R*.

    1.3 Become proficient in writing and using functions.

    1.4 Learn the basic elements of flow control.

Midterm Coding Project

- Midterm Coding project: topics need to be approved by Friday 9/17 by 5 pm.

- **Project topic should be useful to you (used for another class, used for research, etc). If you want to use another class's project, please run it by the other instructor.**

- Needs to be enough of a project in order to:
  - refactor, document, and optimize later.
  - be completed in a reasonable amount of time (not too little, not too much).
  - otherwise no limit on subject, complexity, or length.

- If you are struggling to come up with ideas, talk to me.

Downloading *R* and RStudio

Download *R*: https://www.r-project.org/

1. **Go to https://cloud.r-project.org/**
2. **Select your operating system.**
3. **Select the latest release that is "notarized and signed."**
4. **Save and open the file, follow the instructions to install.**

Download RStuido: https://rstudio.com/products/rstudio/download/

1. **Select the RStudio Desktop version.**
2. **Download, open, and follow instructions to install.**
3. **Open RStudio to get started!**

Resources for learning R

https://www.youtube.com/playlist?list=PL_5IKOC-1S0eE62DUHW2O0JEkPynZbqzP – My playlist of Intro to R Youtube Videos

http://duhi23.github.io/Analisis-de-datos/Cotton.pdf – *Learning R* book

https://bookdown.org/ndphillips/YaRrr/ – YaRrr! A Pirate's Guide to *R*

https://r4ds.had.co.nz/ – *R for Data Science* Book

http://users.metu.edu.tr/ozancan/R%20Graphics%20Cookbook.pdf – *R Graphics Cookbook*

https://adv-r.hadley.nz/ – *Advanced R* book

http://adv-r.had.co.nz/Style.html – Style guide *(Advanced R*)

About *R*

What is R?

- GNU-project language build around statistical computing and data-visualization graphics
- Integrated suite of software for handling data, running calculations, displaying graphics
- Open-access implementation of S (lots of S code runs in R).
- Can be linked with C, C++, Fortran (can be manipulated with C, C++, Java, or Python code).
- Many user-developed intermediate tools for analysis, curated and distributed by central repository (CRAN)

Why use R?

- Free and open source, good for open science, reproducibility, and accessibility
- Publication-quality graphics
- Many statistical tests and models available
- Easy to learn
- Own documentation tool

Why not use R?

- Very quirky
- Slow for a lot of applications (loops)
- Not best at handling large data sets
- Not best at matrices and linear algebra calculations

Basic Syntax in *R*

R as a calculator:

```
> 2+2
[1] 4
```

operators and orders of operations are very intuitive

Assigning a value:

```
> x <- 2
```
good anywhere

```
> 2 -> x
```
equivalent to leftward form

```
> x = 2
```
only good at top level/command line

```
> x <<- 2
```
functions, search environments for similar variable, will redefine or assigns to global environment

Creating a vector:

```
> x <- 1:5
```
vector sequence 1 to 5

```
> x <- seq(1,5)
```

```
> x <- c(1,2,3,4,5)
```

```
> y <- rep(1,5)
```
vector of 1's that is 5 long

Basic Syntax in *R*

"Everything that exists is an object; every operation is a function call."

Equivalent statements:

```
> x <- 2
> x<-2
> x <-2
> x<- 2
```

whitespace largely does not matter

Incorrect syntax:

```
> x < - 2
> x< - 2
> x < -2
```

operators/functions must remain intact

Functions:

```
> function(arg1,arg2,…)   or   > function ( arg1 ,arg2 , … )
> mean(x)
> seq(1,5,by=0.05)
> seq(1,5,b=0.05)
> seq(1,5,0.05)
> rep(c(0,1),5)
```

(1) arguments specified by exact name,

(2) partial name match, or

(3) arguments specified by position

lazy evaluation!

arguments with multiple elements must be entered as vectors or lists

Data Types

Remember: everything is an object!

Data Types of R Objects	Class homogeneous or heterogeneous?	Dimensions?
• Vectors	homogeneous	1D
• Lists	heterogeneous	1D
• Matrices	homogeneous	2D
• Arrays	homogeneous	nD
• Data Frames	heterogeneous	2D
• Factors		
• Functions		
• more!		

No scalars in R!

Atomic Data Classes in R

- logical
- numeric
- integer
- complex
- character
- raw

Check with: `> class()`

or

`> typeof()`

Coerce with: `> as.logical()`

`> as.numeric()`

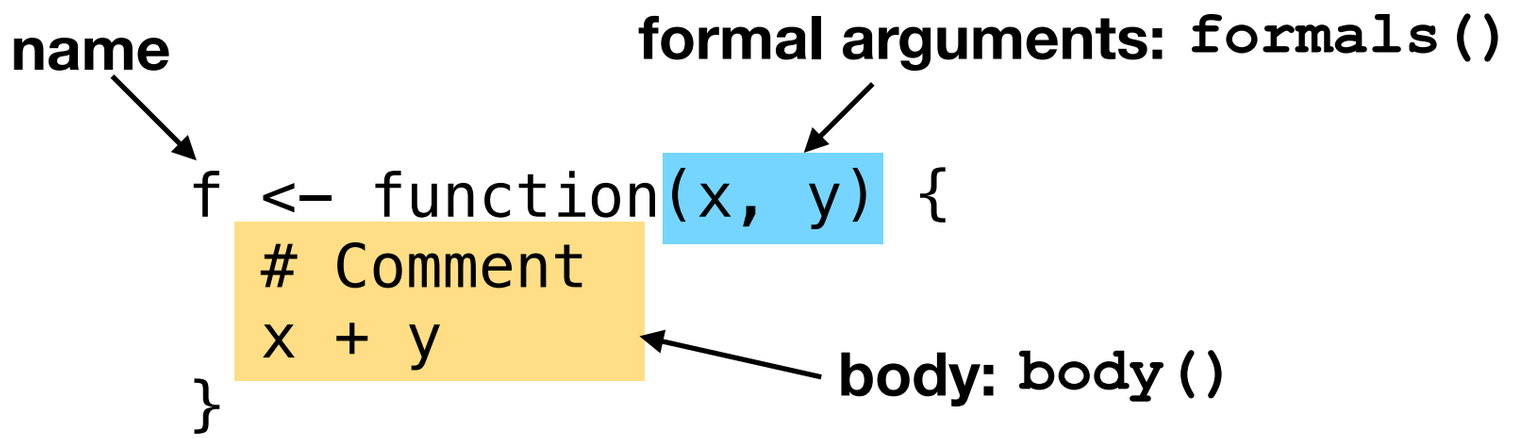
`> as.integer()`

etc.

Functions

Remember: every operation is a function call!

3 Components:



Types:

- primitive
- first-class

Returns:

- implicit: last calculation
- explicit uses **`return()`**

R Quirk: primitive functions call C code directly and don't have the 3 components listed.

Forms:

prefix:	infix:	replacement:	special:
`mean(x)`	**`x + y`**	**`names(x)<-c("a","b")`**	**`for j in`**

Flow control: conditionals

Simple conditional:

```
if(x == 1){
  # code
} else {
  # other code
}
```

vectorized:

```
ifelse(condition,”if true”,"if false”))


alpha<-c(“TRUE”,”FALSE”,”TRUE","TRUE")
ifelse(alpha,”heads”,”tails”))
```

Multiple conditions:

```
if(x == 1){
  # code
} else if(x == 2) {
  # other code
} else {
  # another block of code
}
```

```
switch(
  n,
  once=“Shame on you!”,
  twice=“Shame on me!”
)
```

Flow control: loops

Repeat:

```
repeat{print(“yay!”)}
```

clt + c to kill

```
repeat{
  print(“yay!”)
  n = n + 1
  if (n == 100) break
}
```

While:

```
while (n < 100){
  print(“yay!”)
  n = n + 1
}
```

For:

```
for (n in 1:100) print(paste("yay it’s ”,n))
```

Workspace and Environments

Examining Your Workspace:

`> ls()`	list objects in workspace
`> rm(list=ls())`	clears workspace

Environments: frame that associates data objects, powers scoping

- global environment = user workspace
- environments can be nested
- functions and packages exist in their own environments, enclosed within the environment in which they were created

How R Handles Scoping

Scoping: the act of finding the value associated with a name.

- **Dynamic scoping:** variables are set at the time of a function call, not at the time of function creation

- **Lexical scoping:** looks up values based on how functions were nested when they were created (not how they are nested when called).

- R will look up the values for objects in the current environment, and if it doesn't find it there, it will go up to the parent environment.

1. name masking
2. functions and variables
3. fresh start for every function call
4. dynamics lookup

Lexical Scoping

1. **Name masking**: an object with the same name in a parent environment will mask the value in the parent environment.

```
x <- 10
myfun <- function(){
   x <- 20
   return(x)
}
myfun()
```

What value does `myfun()` return? Is it different than x?

BEWARE!!!

   - Same rule applies to closures (functions created by other functions)

2. **Functions and variables**: finding functions works exactly the same way as finding variables (no difference between these objects).

   - However, if you are using an object in the way a function is used (following it up with `()`), it will skip non-function objects when it looks.

Lexical Scoping

3. **A Fresh Start**: every time a function is called, a new environment is created, so it always had a “fresh start”.

4. **Dynamic lookup**: R will look for values when the function is run, not when it is created.

- This means that functions can return different values if their enclosing environments are different.

- This can be a pain because you don’t pick up errors when you create functions, just when you run them.

Anything else you want to know?

Send me a list!