Module 1: fundamentals

BIOSTAT 213: Introduction to Computing in the R Software Environment

Yea-Hung Chen, PhD, MS

Wednesday, January 16, 2019

Introduction

Course overview

Introductions

- within the next week, please fill out the survey on the Course Learning Environment (labeled Introductions and listed under Assignments)

Introductions

- name: Yea-Hung Chen (yea-hung.chen@ucsf.edu)
- research interests or projects:
  - studying risk behaviors for HIV transmission (with the San Francisco Department of Public Health)
  - SEARCH, a randomized community trial of the test-and-treat strategy for HIV prevention (Division of HIV, Department of Medicine, UCSF)
  - sampling hard-to-reach populations
- familiarity/experience with R:
  - since 2002!
  - in 2015, I taught an informal 6-week R workshop
  - since 2015, I've taught an R workshop for incoming students in the PhD program in Epidemiology and Translational Science
  - since 2017, I've taught an R-based introductory biostatistics course for the Master of Science in Global Health program

Introductions

- name: Adrienne Epstein (adrienne.epstein@ucsf.edu)
- research interests or projects: studying trends of malaria incidence in Uganda using spatial methods
- familiarity/experience with R: 2 years

Introductory video

- if you have not already, please watch the introductory video posted on the Course Learning Environment

Course Learning Environment

- other information available on the Course Learning Environment:
  - course overview (the syllabus)
  - dates, times, and locations
  - slides, data files, and other material
  - readings
  - assignments

Course format

- roughly 1.5 hours of lecture, interspersed with hands-on exercises
- roughly 1 hour of additional hands-on exercises

Course format

- for the lecture period, it is fine if you want to follow along, but I ask that you try to hold troubleshooting questions until the next brief exercise (marked on the guide to the right)
- other than that, please feel free to ask questions throughout
- as much as possible, I want this course to be conversational and collaborative

Course objectives

This is not a statistics class. This is not a statistics class. This is not a statistics class. This is not a statistics class. This is not a statistics class. This is not a statistics class. This is not a statistics class. This is not a statistics class. This is not a statistics class.

Course objectives

This is not an epidemiology class. This is not an epidemiology class.
This is not an epidemiology class. This is not an epidemiology class.
This is not an epidemiology class. This is not an epidemiology class.
This is not an epidemiology class. This is not an epidemiology class.
This is not an epidemiology class. This is not an epidemiology class.

Getting help

- if you have a question that you think other students could learn from, I suggest posting on the course forum, on the Course Learning Environment
- for in-depth, one-on-one help, I suggest emailing me so we can find a time to meet for office hours
- I definitely recommend reaching out early in the quarter if you feel like you need extra help
- one clarification on office hours: I can/should not provide extended consulting on your projects (sorry)

Feedback

- especially because this is a brand-new course, I am interested in feedback throughout the quarter
- you can provide this in class, in person, on the course forum, or via email (yea-hung.chen@ucsf.edu)

Thoughts on learning and using R

Fundamentals

- people often say that R has a steep learning curve
- I think part of the reason for this is that there are important fundamentals (an analogy is learning how to cook)
- so, for example, we don't even talk about importing data today!

Practice

- learning R (or any programming language) is like learning a new (human) language
- as you learn this language, you may sometimes write the "wrong" thing or not know what to write, but it is OK to stumble as you learn
- the language will become easier with practice

Practice

- I'll try to include many opportunities for practice
- I also encourage you to practice on your own
- for example, you might redo work you've done in another language
- for students who've taken biostatistics courses with TICR, for example, you might redo old Stata homework assignments

Multiple solutions

- there is almost always more than one way of doing something (and, certain solutions may be more elegant or efficient than others)
- I'll try to encourage sharing of multiple solutions
- for example, I may ask you to share your work with the class

Reviewing work

- it is useful to build in time to review and revise your code (just as you would review manuscripts you work on)
- by doing so, you may identify errors (but hopefully not introduce new errors!)
- or, you may think of more elegant or efficient solutions
- you may also deepen your understanding of what you're doing or trying to do (the science)

R and RStudio

Why use R?

- it is extremely powerful
  - unlike most other statistical software, it supports using multiple data objects and various types of objects
  - it supports a wide array of methods, in large part because there are numerous (more than 12,000!) add-on packages
- it is extremely popular in academia as well as in industry
  - there are numerous books and other resources for using R
  - it is a marketable skill!
- it is free of cost

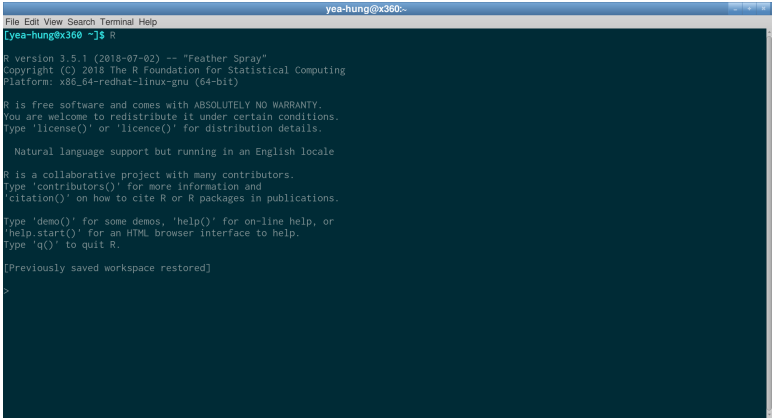
R

- Thieme's article:
  - community of developers (as opposed to within a private company)
  - user communities
  - packages
  - CRAN

R

- as mentioned in my introductory video (and apparent in Thieme's article), the term R is used to refer, interchangeably, to the programming language and the software
- what the software looks like will depend on what operating system you use
- on my computer (which uses a Linux operating system), the software does not include any graphical elements

R



R

- on your computer, the software may include additional graphical elements
- for example, on Windows, the software (sometimes referred to as Rgui) adds a script editor (more later, but this is analogous to the Do-file Editor in Stata)

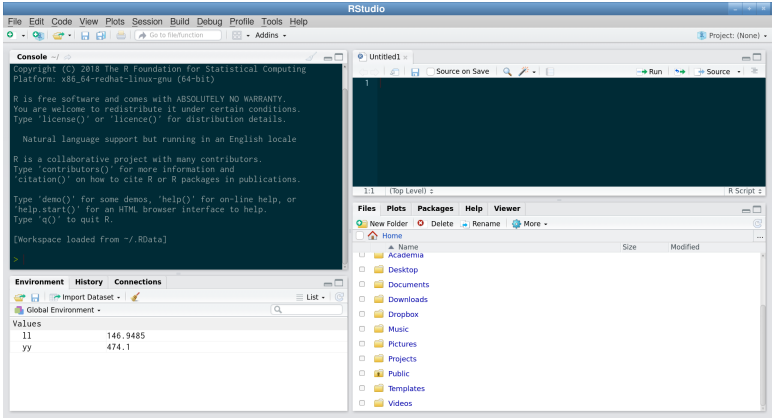
RStudio

- there are several graphical user interfaces for R, but by far the most popular is RStudio, which was first released in 2011

RStudio

- to be clear, RStudio does not allow for graphical operation of R (such as through drop-down menus)
- but, RStudio does include numerous helpful features including:
  - a tool for writing and editing code
  - a tool for browsing files on the computer
  - a tool to R's built-in documentation, with clickable links
  - a tool for viewing and exporting plots
  - a tool for browsing and viewing data objects
  - a tool for installing packages
- technical note: RStudio is an example of an integrated development environment (IDE)

RStudio



RStudio panes

- RStudio has 4 distinct sections, known as panes:
  - Console
  - Script
  - Environment, History, Connections
  - Files, Plots, Packages, Help, Viewer
- you can hide some of these panes if you wish
- you can also rearrange the panes if you wish (Tools > Global Options > Pane Layout)

The console

- the console: a space for typing instructions (code) to the computer and receiving results and messages (output) from the computer
- roughly speaking, in RStudio:
  - R is the console
  - RStudio adds the other 3 panes

The console

- at startup, the console will display a greeting, with information about the software and installation
- the greater-than sign (>) is a prompt ("it's your turn"), indicating that the console is ready to receive instructions

Why do we need to type anyway?

the console: a space for typing instructions (code) to the computer

- why do we need to type instructions?
- wouldn't it be easier to use a graphical user interface to send instructions?
- advantages of typing are:
  - it easily allows for reproducibility
  - it easily allows for sharing of work
- the advantages of typed instructions will be clearer when we discuss scripts later today (we'll focus on the console for now)

Mathematical operators and functions

Mathematical operators

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

- 4: the output (result) of the mathematical operation $2 + 2$
- [1]: numbering for the output (roughly analogous to line numbering you may see in, for example, a legal document)

Mathematical operators

- `(` and `)` for parentheses
- `^` for exponentiation
- `*` for multiplication
- `/` for division
- `+` for addition
- `-` for subtraction

Mathematical operators

```
> 3+5*3
[1] 18
```

- you can use more than one operator per line
- standard order of operations applies (PEMDAS)

Mathematical operators

```
> 14.26+33.14+92.90
[1] 140.3
> 100-3*14.99
[1] 55.03
> 22*0.25+34*0.10+43*0.05+182*0.01
[1] 12.87
> 22*.25+34*.1+43*.05+182*.01
[1] 12.87
> 30.75/0.25
[1] 123
```

Mathematical operators

```
> pi*5^2
[1] 78.53982
> 325*1.01^12+110*1.01^6
[1] 482.9854
> 25^3
[1] 15625
```

Mathematical functions

```
> sqrt(81)
[1] 9
```

- `sqrt()` is an example of an R function
- a function is a an instruction or a set of instructions
- in the example above:
  - `sqrt`: the name of the function
  - `81`: the number we want the square root of
  - the parentheses: a space to put the number in
- when using functions in R, it is always necessary to include both parentheses

Mathematical functions

- `sqrt()` for square roots
- `abs()` for absolute values
- `exp()` for powers of e and `log()` for natural logs
- `log10()` for base-10 logs
- `log()` with `base` argument for other bases (more on this soon)

Mathematical functions

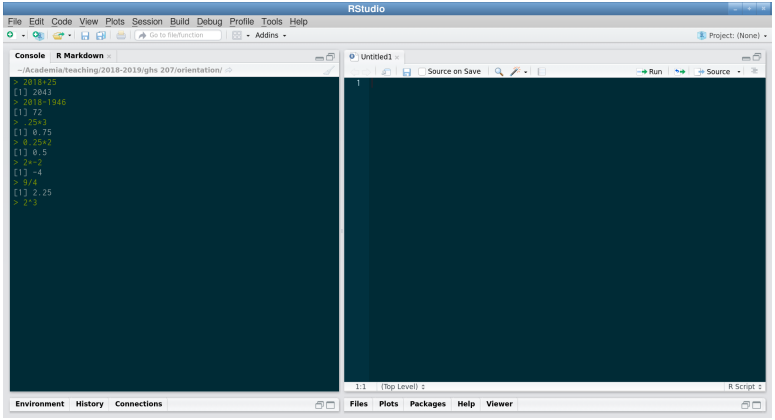
```
> sqrt(144)
[1] 12
> abs(0.32-0.38)
[1] 0.06
> exp(0.87)
[1] 2.386911
> log(exp(2))
[1] 2
> log10(100)
[1] 2
```

Rounding

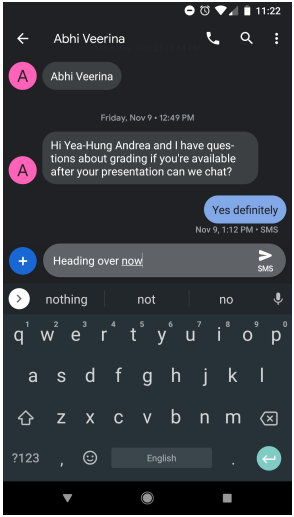
```
> round(pi,2)
[1] 3.14
> floor(pi)
[1] 3
> ceiling(pi)
[1] 4
> round(pi,3)
[1] 3.142
> round(377/120,3)
[1] 3.142
```

- I suggest avoiding rounding functions, as you may end up accidentally rounding as you go

An analogy



An analogy



An analogy

- the console is like a message thread between you and your computer
- the console displays your messages as well as messages from your computer
- the console provides a space for typing new messages
- one key distinction is that in the R console, you are typing in the R language
- another distinction is that the messages you send your computer are instructions (or clarifications)

Brief Exercise 1

Brief Exercise 1

Use the console to complete this exercise. You may work in groups of 2–3.

1. Type, `3-` in the console and press ↵ on your keyboard. Press ↵ again and again and again. Notice that the console is displaying a + sign rather than the usual > symbol (the prompt, which indicates that the computer is ready for instructions). The + sign indicates that your computer thinks you're not done with your instruction. It's occurring here because `3-` suggests an incomplete instruction to do subtraction. To complete the instruction, type any number (for example, `1.25`). You should see the prompt again.
2. Type `3-` again and press ↵ on your keyboard. Press ↵ again and again and again. Notice the + signs. This time, press the Esc key on your keyboard to cancel the instruction. You should see the prompt again.

More on the next slide...

Brief Exercise 1

3. Type, `sqrt(81` in the console and press ↵ on your keyboard. Press ↵ again and again and again. Notice the + signs. They are appearing because `\sqrt(81` suggests an incomplete instruction to find the square root of 81: you didn't close the parentheses. To complete the instruction, type `)`. You should see the prompt again.
4. Type `sqrt(81` again and press ↵ on your keyboard. Press ↵ again and again and again. Notice the + signs. This time, press the Esc key on your keyboard to cancel the instruction. You should see the prompt again.

More on the next slide...

Brief Exercise 1

5. For individuals who are 5 ft 9 in tall, the Centers for Disease Control and Prevention define "obesity" as weighing 203 lb or greater. Use R to find the definition's cutoff in kilograms (1 lb = 0.453592 kg).
6. Body mass index (BMI) is a commonly used (though controversial) measure of human size. It is defined as (mass in pounds)/(height in inches)^2 $\times$ 703. Use R to find the BMI for somebody who is 5 ft 4 in tall and weighs 130 pounds (note: there are 12 inches in a foot). Try to write this all in one line of code.

More on the next slide...

Brief Exercise 1

7. Suppose that you have \$350 in a savings account. The account accrues 0.7% per month. Assuming no withdrawals or other deposits, the mathematical expression for finding the value in the account after 18 months is: $350 \times (1 + 0.07)^{18}$. Use R to find the value in the account after 18 months.
8. With the cursor blinking at the prompt, press $\boxed{\uparrow}$ on your keyboard a few times. What do you notice? Press $\boxed{\downarrow}$ a few times. What do you notice? Continue pressing $\boxed{\downarrow}$ until nothing is displayed after the prompt.

Functions

Functions

```
> sqrt(81)
[1] 9
```

- as mentioned earlier, `sqrt()` is an example of an R function

Help

- each R function usually has a help page
- to view the help page for a function, use `help()` or `?`

```
> help(sqrt)
> ?sqrt
```

Arguments

- you'll notice a section in the help page for `sqrt()` labelled Arguments
- arguments are inputs to the functions

Arguments

- `sqrt()` has just 1 argument: the number you want the square root of
- the argument has a name, `x`:

```
> sqrt(x=81)
[1] 9
```

- however, as already shown, you can omit the `x=` for brevity:

```
> sqrt(81)
[1] 9
```

Arguments

- for arguments with more than 1 order, the order of the arguments matters
- you can omit the argument names if you use the proper order:

```
> round(pi,3)
[1] 3.142
```

- you can also specify the arguments out of order if you use the argument names:

```
> round(digits=3,x=pi)
[1] 3.142
```

Arguments

```
> log(256,2)
[1] 8
> log(x=256,base=2)
[1] 8
> log(256,base=2)
[1] 8
> log(base=2,x=256)
[1] 8
> log(2,256)
[1] 0.125
```

Writing functions

- a feature of R is the ability to write new functions
- we'll discuss this in Module 6

Object assignment

Object assignment

- you can use R to instruct your computer to save information, such as numbers:

```
> radius<-4
```

- in the example above:
  - 4: the number you want to save
  - <-: a left arrow, indicating that you want to save the number
  - radius: is an arbitrary name
- you might imagine, for example, placing (<-) a document (4) in a file folder and then labeling the folder (radius)

Object assignment

```
> radius<-4
```

- this process is known as object assignment
- radius is an object
- an object is a saved piece of information (analogy from the previous slide: like a document)
- there are many types of objects in R (more on this soon)
- in this section, we'll only discuss numeric objects
- numeric objects are saved numbers

Object assignment

```
> radius<-4
```

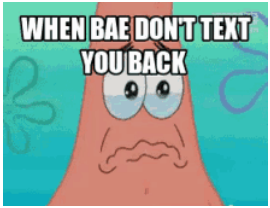
- equivalently, you can use the = sign rather than <-:

```
> another.radius=5.25
```

Object assignment

```
> radius<-4
```

- continuing the analogy that the console is like a message thread, notice that your computer will not write back to you when you assign an object (it quietly accepts the instruction)



Naming rules

- the name should begin with a letter
- the name can include letters, numbers, ., and _

```
> not.quite.pi<-22/7
> also_not_quite_pi<-355/113
> pi.is.equal.to.3<-3
```

- (there are ways to get around the above rules)

Naming rules

"The reservation is under Frankenstein, which, yes, is actually the name of the person who created me, but can we not get into all that?" (from *The New Yorker*)

Retrieving objects

- to retrieve the value of an object, type the object name:

```
> radius
[1] 4
```

Using numeric objects

```
> pi*radius^2
[1] 50.26548
> height<-15
> width<-35
> area<-height*width
> area
[1] 525
```

Replacing objects

```
> radius
[1] 4
> radius<-35
> radius
[1] 35
```

- to continue the analogy of file folders, you can imagine taking an existing document out of the folder and replacing it with a new document
- note that your computer will still not write back to you, so you want to be careful when replacing objects!

Replacing objects

```
> pi
[1] 3.141593
> pi<-3
> pi
[1] 3
```

Replacing objects

```
> pi
[1] 3
```

- to fix this:

```
> rm(pi)
> pi
[1] 3.141593
```

- `rm()` stands for remove
- `rm(pi)` removes our assignment, `pi<-3`

Seeing all objects

- to see all objects, select the Environment tab in RStudio
- alternatively, type `ls()` (short for list):

```
> ls()
 [1] "also_not_quite_pi" "another.radius"    "area"
 [4] "height"            "image"             "link"
 [7] "not.quite.pi"      "pi.is.equal.to.3"  "radius"
[10] "width"
```

- `ls()` is a function and, as mentioned earlier, you must always include both parentheses when using R functions

Replacing objects

```
> radius<-3
> circumference<-2*pi*radius
> circumference
[1] 18.84956
> radius<-10^6
> circumference
[1] 18.84956
```

- note that objects are not "linked"
- in the example above, replacing the value of `radius` after assigning `circumference` does not affect the value of `circumference`

Removing objects

- as mentioned earlier, you can use remove objects using `rm()`
- to remove all objects, use `rm(list=ls())`

Scripts

Scripts

- it is useful to save code in a file, particularly if you expect to run that code again or share it with others
- in R, that file is called a script
- the file extension is generally `.r` or `.R`
- it is a plain-text file

Scripts

- scripts are an easy and useful way to save work
- a possible work flow is to go back and forth between the console and your script, with the script serving as a place to save work and the console serving as a place to test things out

Comments

- to add comments, use the pound symbol (hash symbol):

```
exp(2)
# this is a comment
# 2019-1776
2019-1776
```

- I've excluded prompts and output above, to emphasize that we are looking at a possible excerpt from a script

Line wrapping

- you can wrap code across lines but make sure to always put mathematical operators before the line break

```
2+
  2
2
  +2
```

RStudio's script editor

- you can write and edit scripts in any text editor
- but, RStudio does have a built-in editor, as we saw earlier
- advantages of using RStudio's editor:
  - it has syntax highlighting
  - you can run code from the editor

Brief Exercise 2

Brief Exercise 2

You may work in groups of 2–3.

1. Use RStudio to create a new script: File » New File » R Script. Alternatively, use the icon or keyboard shortcut (Ctrl + ⇧ + N).
2. Save the file: File » Save. Alternatively, use the icon or keyboard shortcut (Ctrl + S). Make sure to give the file a `.r` or `.R` extension. It doesn't really matter where you save the file, but I recommend creating a folder for this module.

More on the next slide...

Brief Exercise 2

3. Bill and Bob, identical twins, are each 6 feet 2 inches tall. In your new script, write code that will save this height (using some arbitrary name), in inches (note: there are 12 inches in a foot). You should create just one object.
4. Run the code you wrote above: place the cursor somewhere in the line and press Ctrl + ↵ (alternatively, use the icon or the menu).
5. In the console, type out the name of the object you defined above. Verify that your computer returns the value of the object.

More on the next slide...

Brief Exercise 2

6. Bill weighs 210 pounds. In your script, use the object from (3) to write code for finding Bill's BMI. BMI is defined as: $(\text{pounds})/(\text{inches})^2 \times 703$. Do not define a new object for Bill's BMI; just write out the code that will provide the value.
7. Bob weighs 180 pounds. Use the object from (3) to find Bob's BMI. Do not define a new object for Bob's BMI; just write out the code that will provide the value.
8. Run the entire script file: highlight the entire text (eg, Ctrl + A) and run the code (eg, Ctrl + ↵). Alternatively, use the icon or the menu.
9. Verify that your computer returns the two BMI values above.

Numeric objects

Object types

- as mentioned before, there are various types (classes) of objects in R
- we've been focusing on numeric objects
- we've also seen another object type: functions

```
> class(sqrt)
[1] "function"
> class(sum)
[1] "function"
> class(pi)
[1] "numeric"
> class(15)
[1] "numeric"
```

Numeric objects

- so far, we've only been considering single numeric values at a time
- but, a numeric object can actually store more than 1 value

```
> radius<-c(2,15,5,8,12,16)
> radius
[1]  2 15  5  8 12 16
```

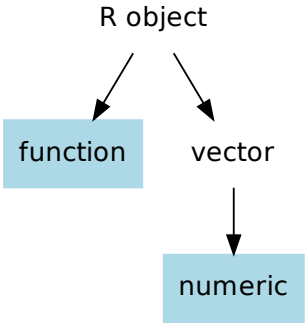
- `c()` is a function and combines values together into a "set," known in R as a vector (analogous to a column in a spreadsheet)

Numeric objects

```
> class(radius)
[1] "numeric"
> is.vector(radius)
[1] TRUE
```

- `radius` is an example of a numeric object
- it is also a vector (we'll see other examples of vectors later)

Numeric objects



Math with numeric objects

```
> 2*pi*radius
[1]  12.56637  94.24778  31.41593  50.26548  75.39822 100.53096
> pi*radius^2
[1]  12.56637 706.85835  78.53982 201.06193 452.38934 804.24772
```

Math with numeric objects

```
> temp<-c(72,85,58,90)
> (temp-32)*5/9
[1] 22.22222 29.44444 14.44444 32.22222
> logor<-c(0.182,-0.356,1.163)
> exp(logor)
[1] 1.1996142 0.7004726 3.1995174
```

Math with numeric objects

```
> exam1<-c(92,85,67,82)
> exam2<-c(88,80,72,85)
> (exam1+exam2)/2
[1] 90.0 82.5 69.5 83.5
> price<-c(100,80,90)
> discount<-c(.25,.1,1/3)
> price*(1-discount)
[1] 75 72 60
```

Sequences

```
> 1:4
[1] 1 2 3 4
> 10:2
[1] 10  9  8  7  6  5  4  3  2
> -1:3
[1] -1  0  1  2  3
```

Sequences

```
> seq(1,4,1)
[1] 1 2 3 4
> seq(0,6,0.5)
 [1] 0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0
> seq(from=0,to=12,by=3)
[1]  0  3  6  9 12
> seq(2,6.1,0.5)
[1] 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0
> seq(2,by=3,length.out=5)
[1]  2  5  8 11 14
```

Repetition

```
> rep(-1,7)
[1] -1 -1 -1 -1 -1 -1 -1
> rep(c(1,3),2)
[1] 1 3 1 3
> rep(c(1,3),times=2)
[1] 1 3 1 3
> rep(c(1,3),each=2)
[1] 1 1 3 3
```

Lengths and sums

```
> length(radius)
[1] 6
> sum(radius)
[1] 58
```

Sorting

```
> sort(radius)
[1]  2  5  8 12 15 16
> radius
[1]  2 15  5  8 12 16
> order(radius)
[1] 1 3 4 5 2 6
> sort(radius,decreasing=TRUE)
[1] 16 15 12  8  5  2
```

Sets

```
> duplicated(c(5,2,2,5,2,5))
[1] FALSE FALSE  TRUE  TRUE  TRUE  TRUE
> unique(c(5,2,2,5,2,5))
[1] 5 2
> is.element(5,1:6)
[1] TRUE
> is.element(1:6,5)
[1] FALSE FALSE FALSE FALSE  TRUE FALSE
```

Brief Exercise 3

Brief Exercise 3

Type your code in the script you created in the 2nd exercise, possibly adding comments. You may work in groups of 2–3.

1. A popular anecdote is that Gauss (or Einstein, or some other mathematical genius), was "punished" in class and asked to sum every integer between 1 and 100. As the anecdote goes, Gauss realized that he could do this via the following formula: $(100 \times 101)/2$. Compute the sum in R, without using this formula. Compute the sum in R again, this time using the formula. Verify that the two answers match.
2. Create an object that stores the following years of birth: 1952, 1946, 1979, and 1981.
3. Use (2) to compute the approximate age of each represented individual.

More on the next slide...

Brief Exercise 3

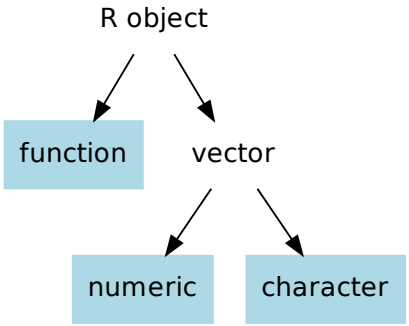
4. Create an object that stores the following weights, in pounds: 141, 185, 226, and 196.
5. Create an object that stores the following heights, in inches: 68, 72, 70, 72, and 63.
6. Use the above to compute the BMI of each represented individual in inches. BMI is defined as: $(\text{pounds})/(\text{inches})^2 \times 703$.

Character objects

Character objects

- character objects store text
- like numeric objects, character objects are vectors

Character objects



Character objects

- you must always use quotation marks in your definition
- you can use single or double quotes, but must end with whatever you start with (for example, if you start with single, end with single)

```
> smile<-":)"
> smile
[1] ":)"
> last.name<-c('Washington','Adams','Jefferson','Madison')
> last.name
[1] "Washington" "Adams"      "Jefferson"  "Madison"
> numeric.no<-c('3.2','4.4','-1.2')
> class(numeric.no)
[1] "character"
```

Character objects

```
> note1<-"it's ok to put single quotes inside double quotes"
> note2<-'yea-hung said, "you can also put double
+   quotes inside single quotes"'
```

- the + above is not part of the code (it's part of the output, indicating line wrapping)

Character manipulation

```
> last.name
[1] "Washington" "Adams"      "Jefferson"  "Madison"
> tolower(last.name)
[1] "washington" "adams"      "jefferson"  "madison"
> toupper(last.name)
[1] "WASHINGTON" "ADAMS"      "JEFFERSON"  "MADISON"
> paste('hello','world')
[1] "hello world"
> paste('res',1:5,sep='')
[1] "res1" "res2" "res3" "res4" "res5"
> paste(last.name,collapse='.')
[1] "Washington.Adams.Jefferson.Madison"
```

Character manipulation

```
> last.name
[1] "Washington" "Adams"      "Jefferson"  "Madison"
> nchar(last.name)
[1] 10  5  9  7
> substr(last.name,start=1,stop=3)
[1] "Was" "Ada" "Jef" "Mad"
> substr(last.name,1,5)
[1] "Washi" "Adams" "Jeffe" "Madis"
```

Character manipulation

- we'll discuss more advanced methods for character manipulation (regular expressions) in Module 6

Character to numeric

```
> numeric.no
[1] "3.2"  "4.4"  "-1.2"
> class(numeric.no)
[1] "character"
> numeric.yes<-as.numeric(numeric.no)
> numeric.yes
[1]  3.2  4.4 -1.2
> class(numeric.yes)
[1] "numeric"
> as.numeric(last.name)
Warning: NAs introduced by coercion
[1] NA NA NA NA
```

Numeric to character

```
> phone<-c(4154762300,4154762310)
> phone<-as.character(phone)
> phone
[1] "4154762300" "4154762310"
> class(phone)
[1] "character"
> phone<-paste(substr(phone,1,3),substr(phone,4,6),
+              substr(phone,7,10),sep='-')
> phone
[1] "415-476-2300" "415-476-2310"
```

Brief Exercise 4

Brief Exercise 4

Type your code in the script you created in the 2nd exercise, possibly adding comments. You may work in groups of 2–3.

1. Create an object that stores the first name of every member of your family (you can define this however you wish) or the main characters of a favorite television show or movie, listing the individuals in an arbitrary order.
2. Redefine the object, without typing out the names again, so that the names are sorted in alphabetical order.
3. Verify that the names are now sorted in alphabetical order.

Quitting

Quitting

- when quitting RStudio, you will be prompted to indicate whether you want to save the workspace image
- it's fine to select either Save or Don't Save for now (we'll discuss workspace images further next week)
- you might also be prompted to save your script, if you haven't saved your script since the most recent update
- you should select Save if that prompt appears (unless you really don't want to save changes to your script)