Module 1: fundamentals

BIOSTAT 213: Introduction to Computing in the R Software Environment

Yea-Hung Chen, PhD, MS

Monday, September 23, 2019

Introduction

Course overview

Introductions

- name: Yea-Hung Chen, PhD, MS (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!
  - since 2015, I've taught a half-day 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: Pingyang Liu, PhD (pingyang.liu@ucsf.edu)

Collaborative Learning Environment

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

Course format

- roughly 1 hour of lecture
- roughly 1 hour of 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 hands-on exercise
- 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 strongly encourage posting on the course forum, on the Collaborative 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

Thoughts on learning and using R

Fundamentals

- people often say that R is difficult to learn
- I think part of the reason for this is that there are important fundamentals
- an analogy is learning how to cook: in a professional cooking school, you might spend weeks practicing knife skills or other fundamentals before you actually begin to cook

Practice

- learning R 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 existing work, rewriting it in R
- for students who've taken biostatistics courses with TICR, 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)

What and why

What is statistical software?

- statistical software is software for conducting statistical analysis
- popular examples include: R, Stata, SAS, and IBM SPSS
- each software program often has its own programming language
- the language allows you to write instructions to your computer

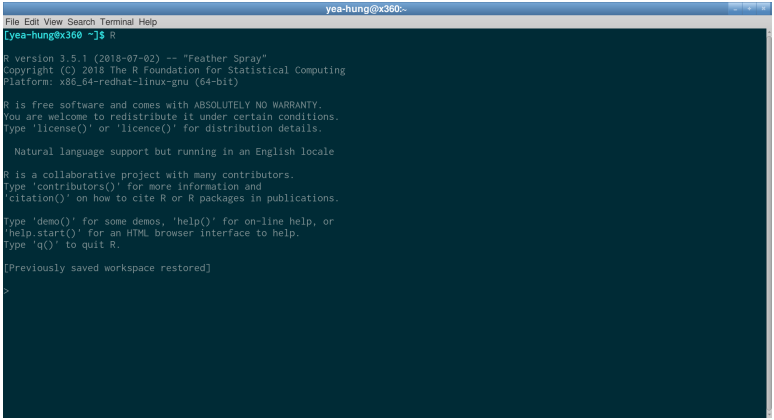
What is R?

- R is a popular statistical software program
- the term also refers to the programming language

What is R?

- R is open source, meaning that the computer code behind it is publicly available online
- development (planning and improvement, etc) of R does not occur within any one private company, but rather within a community of individual contributors, from a variety of organizations
- there is no official technical support, but there are various online forums and other resources
- depending on your operating system, R includes little/no graphical interface (there are few windows, menus, or icons)

What is R?



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 very complex data manipulation
  - 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

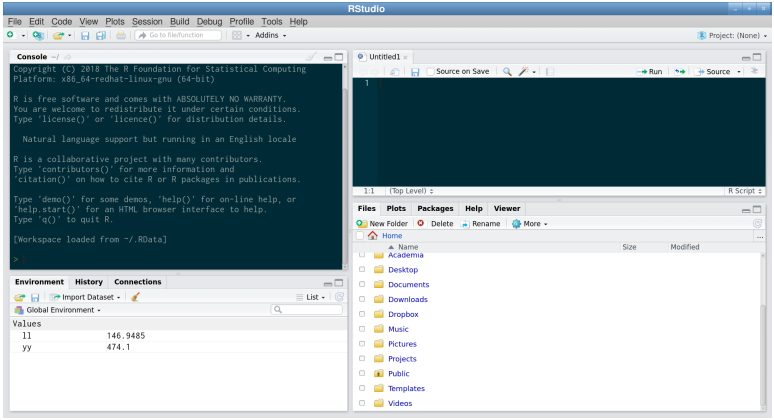
What is RStudio?

- RStudio is an add-on interface (windows, menus, and icons) for R
- it was first released in 2011

What is RStudio?

- to be clear, RStudio does not allow for graphical operation of R (such as through menus/icons, as in IBM SPSS)
- 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

What is RStudio?



What is RStudio?

- 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?

- why do we need to type instructions?
- wouldn't it be easier to use menus and icons to send instructions (as in IBM SPSS)?
- the 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

```
> 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
```

Mathematical operators

```
> pi*5^2
[1] 78.53982
> 325*1.01^12
[1] 366.2181
```

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

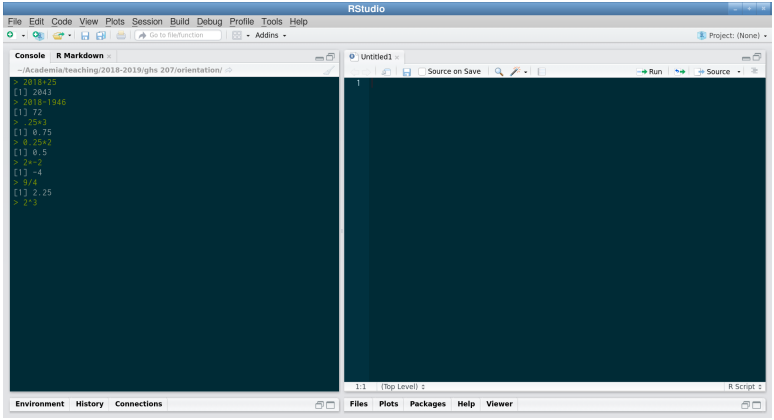
Mathematical functions

- `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 the `base` argument for other bases (more on this soon)

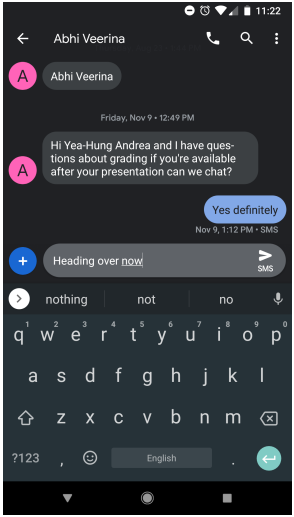
Mathematical functions

```
> 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
```

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)

Functions

Functions

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

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

Help pages for functions

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

```
> help(sqrt)
```

- equivalently, use ?:

```
> ?sqrt
```

Function arguments

- you'll notice a section in the help page for `sqrt()` labelled Arguments
- arguments are inputs to the functions
- you may think of them as options or settings
- a cooking analogy: if baking is the function, then the temperature setting might be an argument to the function
- what's another possible argument to baking, if we view baking as a function?

Function arguments

- the help page for `sqrt()` indicates that `sqrt()` has just 1 argument: the number you want the square root of
- the argument has a name, `x`
- you can optionally include the name when you use the function:

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

- however, as already shown, you can omit the argument name for brevity:

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

Function arguments

- let's look at the help page for the `log()` function:

```
> help(log)
```

Function arguments

- `log()` has two arguments:
  - `x`: the number we want to take the log of
  - `base`: the base of the logarithm
- notice that the help page indicates that the `base` argument has a default value, the natural number e
- this is consistent with what I've told you: by default, `log()` will give you the natural log

Function arguments

- for functions with more than 1 argument, such as `log()`, use commas to separate the arguments:

```
> log(x=64,base=2)
[1] 6
```

Function arguments

- you can omit the argument names if you use the proper order:

```
> log(64,2)
[1] 6
```

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

```
> log(base=2,x=64)
[1] 6
```

Function 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
```

Object assignment

Object assignment

- you can use R to instruct your computer to save information, such as numbers, words ("characters"), or entire data sets
- an analogy is placing some food ingredients in a jar (pasta, for example), and then labeling the jar
- in R terminology, the jar of ingredients is known as an object
- the process of putting the ingredients in a jar—and labeling the jar—is known as object assignment

Object assignment

nicolescadeaux.wordpress.com

Object assignment is like putting food ingredients (pasta, for example) in a jar, and then labeling the jar.

Object retrieval

preservecompany.com

The objects you save (the food ingredients) will be available to you later, and easily identifiable if you've used useful object names (labels on the jars).

Object assignment

- to assign an object (put some food ingredients in a jar), use <- (it's supposed to look like a left arrow):

```
> radius<-4
```

- in the example above:
  - 4: the number you want to save
  - <-: a left arrow, indicating that you want to save the number (put something in a jar)
  - radius: an arbitrary name

Object assignment

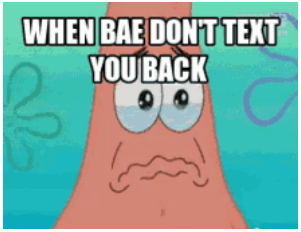
```
> radius<-4
```

- continuing the jar analogy: you can imagine that you are placing (<-) some food ingredient (4) in a jar and then labeling the jar with some arbitrary (but hopefully useful) name (radius)

Object assignment

```
> radius<-4
```

- notice that your computer will not write back to you when you assign an object (it quietly accepts the instruction)



Object names

- in general:
  - 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
```

- otherwise, the names are up to you to decide
- there are ways to get around the above rules sometimes

Object names

"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*)

Object names

- object names in R are case sensitive
- the following defines two different objects (two different jars):

```
> radius<-4
> Radius<-900
```

Object types

```
> radius<-4
```

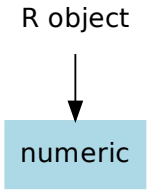
- radius is an object
- there are many types of objects in R
- continuing the jar analogy: there are main types of food ingredients you might put in a jar
- `radius` is an example of a numeric object

Object types

```
> class(radius)
[1] "numeric"
```

- the `class()` function shows the object type
- other object types include: character, logical, factor, data frame...

Object types

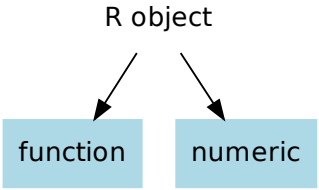


Object types

- we've also seen another object type:

```
> class(sqrt)
[1] "function"
```

Object types



Retrieving objects

- to retrieve the value of an object (take everything out of the jar), type the object name:

```
> radius
[1] 4
> Radius
[1] 900
```

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 jar analogy: you can imagine taking food ingredients out of the jar and replacing it with new ingredients
- note that your computer will still not write back to you, so you want to be careful when replacing objects!

Removing objects

- to remove an object, use `rm()` (short for remove):

```
> rm(area)
```

Listing all objects

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

```
> ls()
[1] "also_not_quite_pi" "height"
[3] "image"             "link"
[5] "not.quite.pi"      "radius"
[7] "Radius"            "width"
```

- `ls()` is a function and, as mentioned earlier, you must always include both parentheses when using R functions, even if there is nothing inside the parentheses

Removing all objects

- to remove all objects, use `rm(list=ls())`
- `list` is the name of one of the arguments to the function `rm()` and `ls()` is the value of the argument we are providing

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
- you can think of this as writing down a set of instructions for your computer, which you can revisit later
- or, using a food analogy, you can think of a script file as being like a recipe

Scripts

- the file extension for an R script is generally `.r` or `.R`
- it is a plain-text file

Scripts

- scripts are an easy and useful way to save work (again: it's like a written set of instructions, or a recipe)
- 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):

```
sqrt(25)
# 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
- this:

```
102*.01+122*.05+425*.1+
  220*.25
```

- ... is not the same as this:

```
102*.01+122*.05+425*.1
  +220*.25
```

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
  - it will try to fill things in for you sometimes
  - you can run code from the editor

RStudio's script editor

- to run code from RStudio's script editor:
  1. select (highlight) the code by holding down the mouse button and dragging (or using ⇧ and the arrow keys on your keyboard) (note that ⇧ represents the shift key, not the up key!)
  2. select Run at the top of the window (equivalently, press Ctrl + ↵)

Vectors

Vectors

```
> radius
[1] 35
```

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

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

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

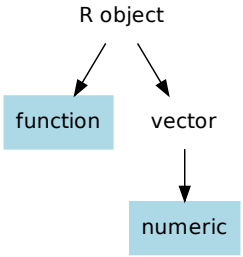
Vectors

- to check whether an object is a vector, use `is.vector()`:

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

- `radius` is a vector and, more specifically, it is a numeric object

Vectors



A numeric object is a type of vector. We'll see other types of vectors next week.

Vectors

- note that even a single number is a vector:

```
> Radius
[1] 900
> is.vector(Radius)
[1] TRUE
```

Math with vectors

```
> 2*pi*radius
[1]  12.56637  94.24778  31.41593  50.26548  75.39822
[6] 100.53096
> pi*radius^2
[1]  12.56637 706.85835  78.53982 201.06193 452.38934
[6] 804.24772
```

Math with vectors

```
> temp<-c(72,85,58,90)
> (temp-32)*5/9
[1] 22.22222 29.44444 14.44444 32.22222
```

Math with vectors

```
> exam1<-c(92,85,67,82)
> exam2<-c(88,80,72,85)
> (exam1+exam2)/2
[1] 90.0 82.5 69.5 83.5
```

- here, exam1 and exam2 are the same length, so the math works pairwise

Math with vectors

```
> exam1<-c(92,85,67)
> exam2<-c(88,80,72,85)
> (exam1+exam2)/2
Warning in exam1 + exam2: longer object length is not a
multiple of shorter object length
[1] 90.0 82.5 69.5 88.5
```

- here, exam1 and exam2 are not the same length, so R attempts to recycle the shorter vector
- I discourage such usage, as it's confusing to think about

Quitting

Quitting

- when quitting RStudio (or R), you will be prompted to indicate whether you want to save the workspace image
- select Save if you want the objects (jars) to be available to you later
- note that to see the objects later, you will need to return to the same working directory (more on this next week!)

Quitting

- you might also be prompted to save your script
- you should select Save if that prompt appears (unless you really don't want to save changes to your script)

Saving work

- we'll discuss this in greater detail next week, but in general, my recommendation is that you rely on scripts to save your work, rather than saved objects (jars)

Exercise

Exercise

You may work in groups of 2–3.

Use the console to complete the next few questions.

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...

Exercise

3. Type, `sqrt(81` in the console. RStudio will automatically try to fill in the closing parenthesis, but you should manually delete it. 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` in the console. RStudio will automatically try to fill in the closing parenthesis, but you should manually delete it. 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...

Exercise

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...

Exercise

7. Suppose that you have \$350 in a savings account. The account accrues 0.7% per month. Assuming no withdrawals or deposits, the mathematical expression for finding the value in the account after 18 months is: $350 \times (1 + 0.007)^{18}$. 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.

More on the next slide...

Exercise

Work in a script for the remaining questions.

9. Use RStudio to create a new script: File » New File » R Script. Alternatively, use the icon or keyboard shortcut (Ctrl + ⇧ + N).
10. 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 class or day.

More on the next slide...

Exercise

11. 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.
12. Run the code you wrote above: place the cursor somewhere in the line and press Ctrl + ↵ (alternatively, use the icon or the menu).
13. In the console, type the name of the object you defined above, and press ↵. Verify that your computer returns the value of the object.

More on the next slide...

Exercise

14. Bill weighs 210 pounds. In your script, use the height object from above 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 the code that will provide the value.
15. Bob weighs 180 pounds. Use the height object from above to find Bob's BMI. Do not define a new object for Bob's BMI; just write the code that will provide the value.
16. Run the code above, and verify that your computer returns two BMI values.

More on the next slide...

Exercise

17. Create an object that stores the following temperatures, in Fahrenheit: 72, 58, 63, and 91.
18. Type out an expression for the temperatures in Celsius, using the following formula: $(\text{Fahreneheit} - 32) \times 5/9$. Run the code, and verify that the conversions show up in the console.