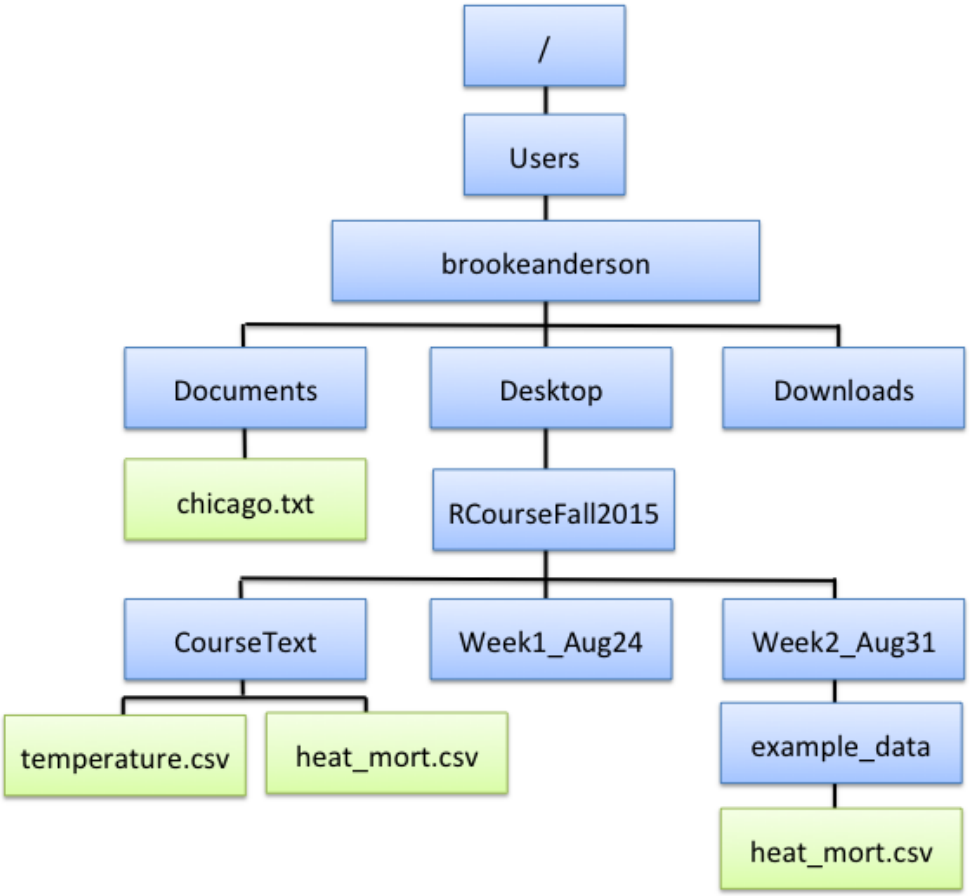
Quiz 2

Name:

1.

Your computer has the following file structure, with `"/Users/brookeanderson` as the home directory:



When you run:

```
getwd()
```

you get:

```
"/Users/brookeanderson/Desktop/RCourseFall2015/Week2_Aug31/example_data"
```

- On the figure, label your current working directory and your home directory (you must get both right for credit for this question).

2.

Suppose you are working on a computer with the same directory structure as in the previous question and are working within the same working directory. You want to read in the file `heat_mort.csv` in the `CourseText` directory. Which of the following commands is the only one that would **not** do that? (You may assume that the file can be read in using all the defaults for `read.csv()`, so you don't need any options in that command.)

a. `read.csv("~/Desktop/RCourseFall2015/CourseText/heat_mort.csv")`
b. `read.csv("/Users/brookeanderson/Desktop/RCourseFall2015/CourseText/heat_mort.csv")`
c. `read.csv("heat_mort.csv")`
d. `read.csv("../../CourseText/heat_mort.csv")`

Answer c. Explanation: `read.csv("heat_mort.csv")` would read in the "heat_mort.csv" file in your current working directory (`/Users/brookeanderson/Desktop/RCourseFall2015/Week2_Aug31/example_data`). This is not the file you want. All other answers would appropriately navigate from the root directory (Answer b.), from your home directory (Answer a.), or, using a relative pathname, move up and over from the current working directory to find the file you want (Answer d.). Note from this question (and the previous one) that you're allowed to have files with the same names in different directories on you computer. This makes it especially important to correctly specify a file pathname, so you're sure you get the right file.

3.

For the previous question, which of the responses uses an absolute pathname?

a. Answer a.
b. Answer b.
c. Answer c.
d. Answer d.

Answer b. (Answer a. is also acceptable) Explanation: Answer b. is a clear example of an absolute pathname, with directions all the way from the root directory down to the file you want. Answer a. is also an absolute pathname (it would work the same regardless of your current working directory), it just abbreviates the part getting from the root to the home directory using the `~` abbreviation. Answers c. and d. are both relative pathnames: they will give you a different result depending on where you start (i.e., what directory you're currently working in).

4.

You want to read into R some data that's available online as a flat csv file. The web address for this data is `http://www.website.com/FileUWant.csv`. (You may assume that the file can be read in using all the defaults for `read.csv()`, so you don't need any options in that command.) What is the simplest way to read this data directly into R and save it as the object `my_data`? Assume you don't have any packages beyond the base R packages installed and would like to avoid installing a new package, if possible.

a. `my_data <- source_data("http://www.website.com/FileUWant.csv")`
b. `my_data <- read.csv("http://www.website.com/FileUWant.csv")`
c. `my_data <- read.csv("../../FileUWant.csv")`
d. `my_data <- load("http://www.website.com/FileUWant.csv")`

Answer b Explanation: If you want to read data from online that is on a page that starts with `http://` and ends with `.csv`, you can read it just like data on your computer, just use the web address as the file name. This is the easiest way to read the data in and does not require installing

(or figuring out how to use) and additional packages. If the data were on a secure site (`https://` at the start of the web address), you could not get `read.csv()` to work and instead would need to install and load the `repmis` package and use its `source_data()` function (Answer a.). Note that this is what you would need to do to read a csv file from GitHub or from a Dropbox public folder, since those both start web addresses with `https://`. While the code in Answer a. would work, it is not the simplest way to do this and would require installing a new package, and so is not the correct answer. Answers c. and d. would not work at all. Answer c. would look for the file in the directory two up from your current working directory, would not find it there (since it's online), and would fail. Answer d. is how you would load a `.RData` file. `load()` cannot be used to read in a csv file.

5.

You have the following data, saved as an object called `my_df`:

```
##                    date temperature
## 1 Thursday, 30 Dec 1999    24.24672
## 2   Friday, 31 Dec 1999    33.09140
## 3 Saturday, 01 Jan 2000    34.07629
## 4   Sunday, 02 Jan 2000    33.92724
```

When you try `class` for the `date` variable, you get the following:

```
class(my_df$date)
```

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

You are trying to get the `date` variable into a Date class, so it is saved as a date object and, if you print it out, it looks like:

```
## [1] "1999-12-30" "1999-12-31" "2000-01-01" "2000-01-02"
```

How would you convert the `date` variable to have the Date class?

a. `my_df$date <- as.Date(as.factor(my_df$date), format = "%Y-%m-%d")`
b. `my_df$date <- as.Date(my_df$date, format = "%A, %d %b %Y")`
c. `as.Date(my_df$date)`
d. `my_df$date <- as.Date(my_df$date, format = "%Y-%m-%d")`

Answer b. Explanation: Everyone had a hard time with this one. When you convert a vector into the `Date` class, you need to tell R what the format **currently** looks like, not what you would like it to look like when you're done. Otherwise, R has no idea where to find different elements of the date in the character vector you're giving it. The `date` column currently looks like this: "Thursday, 30 Dec 1999". You need to tell R where to look for what parts of the date in that format. To do that, you take each element (e.g., month, year, weekday), use its "code", and otherwise keep the format (spaces, commas, etc.) the same. So, here, the format the `date` is currently in is: "%A, %d %b %Y". It is hard to keep track of all of the abbreviations, but this is the only `format` answer that includes all the commas and spaces, so even if you don't know the abbreviations, you should be able to figure out the correct answer from that. Answer a. is especially wrong because it converts the column `date` to a factor before it tries to convert to a date. A lot of problems can crop up trying to convert a factor directly to a date. Often, you'll actually go the other way to avoid those problems, and convert something in a factor class to a character class before you try to use `as.Date()`.

6.

What is the name for a file in which you store R code, which can be fully run using the "Source" button, and which is usually saved with the extension `.R`?

a. A csv file
b. A package file
c. An R console
d. An R script

> **Answer d.** Explanation: A csv file (Answer a.) is a flat file stored on your computer that contains data you may want to read into R using `read.csv`. It is usually saved with the extension `.csv`. The R console (Answer c.) is the area in RStudio where you can enter commands one at a time, after the `>` prompt (bottom left pane for most people).

7.

You have an object called `nepali` that is a dataframe that starts like this:

```
head(nepali)
```

```
##       id sex   wt    ht mage lit died alive age
## 1 120011   1 12.8  91.2   35   0    2     5  41
## 2 120011   1 12.8  93.9   35   0    2     5  45
## 3 120011   1 13.1  95.2   35   0    2     5  49
## 4 120011   1 13.8  96.9   35   0    2     5  53
## 5 120011   1   NA    NA   35   0    2     5  57
## 6 120012   2 14.9 103.9   35   0    2     5  57
```

Which of the following is the only command that would **not** help you figure out whether the variable `sex` is currently stored as a factor?

a. `as.factor(nepali$sex)`
b. `is.factor(nepali$sex)`
c. `str(nepali)`
d. `class(nepali$sex)`

> **Answer a.** Explanation: `as.factor()` would try to convert `nepali$sex` into a factor, not check whether it was one. `is.factor()` (Answer b.) will give you a TRUE / FALSE answer for whether the vector is stored as a factor. `str()` (Answer c.) would give you a lot of information about `nepali`, including the class of all of the columns in that dataframe, including the `sex` column. `class()` (Answer d.) tells you the class of a vector, which would let you figure out if it was stored as a factor.

8.

Assume that you find out that `sex` is stored as a factor, with levels "1" = male and "2" = female, and that `wt` gives the child's weight in kilograms. How could you create a subset of this dataframe, called `nepali_f`, with only observations with female children who weigh more than 12 kg?

a. `nepali_f <- subset(nepali, sex == "2" & wt >= 12)`
b. `nepali_f <- subset(nepali, sex == 2 | wt > 12)`

c. `nepali_f <- subset(nepali, sex == "2" & wt > 12)`
d. `nepali_f <- subset(nepali, sex == "female" & wt == 12)`

Answer c. Explanation: You want to put a logical statement that checks if sex is female (`sex == "2"`, note that R could also gets this right if you forgot the quotation marks, but it's safer to have them) and (`&`) that the weight is over 12 kg (`wt > 12`). Therefore, the correct answer is Answer c. Answer a. would pull girls whose weight was **equal to** or greater than 12. Answer b. would pull any observation where either the sex was female or (`|`) weight was above 12. Answer d. would pull observations where weight equalled `==` 12 and also uses the wrong level label for females, based on the data we're using.

9.

You want to pull out the vector on childrens' weight from the `nepali` object above and save it as the object `weight`. Which of the following commands would **not** achieve this?

a. `weight <- nepali[ , "wt"]`
b. `weight <- nepali$wt`
c. `weight <- nepali[1:nrow(nepali), "wt"]`
d. `weight <- nepali[ , wt]`

Answer d. Explanation: Answer d. omits quotation marks on `wt` in the indexing. By forgetting those, R will not look for a column named "wt", but instead will look for an object *outside of* ***`nepali`*** named `wt`, and then will use the values in that to index the dataframe. This seems very picky in terms of a quiz question, but this error can come up a lot and be very frustrating, so it's useful to understand these kinds of details. The code in Answer c. would work because it's choosing all rows in the dataframe (`1:nrow(nepali)` is indexing every row in the original data), although the faster way to run this line of code would be `weight <- nepali[ , "wt"]`. Answers a. and b. are both very straightforward ways to index out this column from the dataframe.

10.

Which of the following is the only reason that is **not** a valid reason why you might sometimes want to use a relative rather than absolute pathname in your R code?

a. Relative pathnames are usually shorter than absolute pathnames
b. The same relative pathname would work if you share your project directories with another researcher, who runs the code on a different computer
c. Using relative pathnames can help get you in the habit of creating well-structured project directories
d. The same relative pathname would work from any working directory

Answer d. Explanation: Relative pathnames depend on where you start, and so work differently depending on what working directory you're starting from. Therefore, the same relative pathway that works in one working directory will almost always fail if you try to run it from a different working directory. Relative pathnames are almost always shorter than absolute pathnames, so Answer a. is incorrect. Relative pathnames don't depend on the structure of directories near the root directory of the computer, so they could with them can be more portable among different computers, so Answer b. is incorrect. For example, in the graphic for question 1, imagine you saved the `RCourseFall2015` directory and all its contents to a zip file and shared it with a collaborator. Code using the relative pathname (e.g., Answer d. in question 2) would still work, no matter where in his file structure your collaborator saved the directory. The same is not true for absolute pathnames. Answer c. is correct because starting to use relative pathnames can help you think about how to structure subdirectories for projects you work on (e.g., one subdirectory for `Data`, one for `Presentations`, one for `Figures`).

11.

You have a file in your current working directory called `ClassInfo.txt` that starts like this:

```
## Class dates and topics
Class, Date
"Preliminaries", "Aug. 24, 2015"
"Getting and cleaning data 1", "Aug. 31, 2015"
"Exploring data 1", "Sept. 14, 2015"
```

Which of these is the only command that would **not** work to read this file into R?

a. `read.delim("ClassInfo.txt", skip = 1, header = TRUE, sep = ",")`
b. `read.table("ClassInfo.txt", skip = 1, header = TRUE, sep = ",")`
c. `read.csv2("ClassInfo.txt", skip = 1, header = TRUE)`
d. `read.csv("ClassInfo.txt", skip = 1, header = TRUE)`

Answer c Explanation: The default for `read.csv2()` is to assume that values in different columns are separated by a semi-colon, not a comma. The default for `read.csv` (Answer d.) is that commas are separators, so that works without any `sep` options. `read.delim()` (Answer a.) and `read.table()` (Answer b.) have different defaults for the separator, but you can override the default by using `sep = ","`. Remember that all of these functions are in the `read.table()` family. They all do the exact same thing underneath, they just have different defaults for things like `sep =`, so if you aren't using the one with the proper default value for that for your data, you'll just need to specify it as an option. This fact can be convenient– instead of learning every different one, you could always use `read.table()` and just make sure you use the `sep` option to specify your separator.