Homework 1 Solution

Disclaimer

This homework solution is for the sole benefit of students taking Stat 220 from Prof. Bastola during Spring term 2024. Dissemination of this solution to people who are not registered for this course is not permitted and will be considered grounds for Academic Dishonesty for the all individuals involved in the giving and receiving of the solution.

Problem 1: Markdown output

Consider the following code chunks (as shown in the .md or pdf), but *don't* add them to your homework write-up

````
```{r chunk1}
x <- 1:3
```

```{r chunk2}
x <- x + 1
```

```{r}
x
```
````

a.

What output is produced when all three chunks include the option `echo = FALSE`. Explain your answer in words, not by adding these options to chunks above.

answer: In the Markdown doc, none of the chunks of code will appear but we will see the output of 2, 3, 4 from the third chunk.

b.

What output is produced when all three chunks include the option `echo = FALSE` **and** `chunk2` includes the options `eval=FALSE`. Explain your answer in words, not by adding these options to chunks above.

answer: In the Markdown doc, none of the chunks of code will appear, the second code chunk will not be evaulated (so there is no change to the value of `x`) so when we see the output of the third chunk it will equal 1, 2, 3.

Problem 2: Logical vectors

Suppose we have a list of food (carrot, orange, m&m, apple, candy corn) that we characterize by color (orange or not) and candy (candy or not). Here are the data vectors describing each food:

```
> orange <- c(TRUE, TRUE, FALSE, FALSE, TRUE)
> candy <- c(FALSE, FALSE, TRUE, FALSE, TRUE)
> table(orange, candy)
```

```
       candy
orange  FALSE TRUE
  FALSE     1    1
  TRUE      2    1
```

a.

What type of food does the product of these vectors represent? (e.g. what does `x` of 0 mean? what does 1 mean?)

```
> x <- orange*candy
> x
[1] 0 0 0 0 1
```

answer: This produces a vector that indicates an food that is "orange and candy" (intersection) with a 1 (the candy corn) and all other types with a 0.

b.

What type of food does the vector `y` represent? (e.g. what does `y` of 0 mean? what does 1 mean?)

```
> y <- (1-orange)*(1-candy)
> y
[1] 0 0 0 1 0
```

answer: This produces a vector that indicates an food that is "not orange and not candy" (intersection) with a 1 (the apple) and all other types with a 0.

c.

What type of food does the vector `z` represent? (e.g. what does `z` of 0 mean? 1?)

```
> z <- orange*(1-candy)
> z
[1] 1 1 0 0 0
```

answer: This produces a vector that indicates an food that is "orange and not candy" (intersection) with a 1 (the carrot and orange) and all other types with a 0.

Problem 3: Coercion

Suppose that the follow objects are defined:

```
> obj1 <- 2:10
> obj2 <- c(-1, 1)
> obj3 <- c(TRUE, FALSE)
> obj4 <- 10
```

For **a-e** lines below, describe the value of the output that is produced and explain why that command produces the object. Try to answer these questions without using R, but you can use R to help or verify your answer. (e.g. This would be practice for an exam where you can't use R!)

```
> obj1[2:4] #a
[1] 3 4 5
> obj1[-3]  #b
[1]  2  3  5  6  7  8  9 10
> obj1 + obj2  #c
```

```
Warning in obj1 + obj2: longer object length is not a multiple of shorter
object length
[1]  1  4  3  6  5  8  7 10  9
> obj1 * obj2  #d
Warning in obj1 * obj2: longer object length is not a multiple of shorter
object length
[1]  -2   3  -4   5  -6   7  -8   9 -10
> sum(obj3)  #e
[1] 1
```

answer:

- **a.** `obj1[2:4]` will return the 2nd, 3rd and 4th entries in `obj1`. Output will look like `3,4,5`
- **b.** `obj1[-3]` will return everything except the 3rd entry in `obj`. output will look like `2,3,5,6,7,8,9,10`
- **c.** `obj1 + obj2` will start by adding the 1st and 2nd entries in each vector (2 + -1 and 3 + 1). But then we run out of entries in `obj2` so R will start recycling entries and add the 3rd and 4th entries in `obj1` to the 1st and 2nd entries in `obj2` (4 + -1 and 5 + 1). This repeats until all `obj` entries have been used. Output (with a warning) will look like `1,  4,  3,  6,  5,  8,  7, 10,  9`
- **d.** `obj1*obj2` same idea as the previous answer except with multiplication instead of addition. Output will look like `-2,   3,  -4,   5,  -6,   7,  -8,   9, -10`
- **e.** `sum(obj3)` Applying the `sum` function to a logical vector will coerce `TRUE` to 1 and `FALSE` to 0 before summing. So the sum is `1 + 0 = 1`.

Problem 4: Data frames

Install the R package `dslabs` if needed (see Irizarry 1.5), then load the library and the `murders` data that is used in chapter 2 of Irizarry.

```
> library(dslabs)
> data(murders)
```

a.

Use the accessor `$` to extract the state abbreviations and assign them to the object `a`. What is the class of this object?

answer: character

```
> a <- murders$abb
> class(a)
[1] "character"
```

b.

Now use the square brackets to extract the state abbreviations and assign them to the object `b`. Use the `identical` function to determine if `a` and `b` are the same.

answer: they are the same

```
> names(murders)
[1] "state"      "abb"        "region"     "population" "total"
> b <- murders[,2]  # 2nd column
> identical(a,b)
[1] TRUE
```

c.

What class of object is `murders[1:2,1:4]`?

answer: data frame

```
> murders[1:2,1:4]
    state abb region population
1 Alabama  AL  South    4779736
2  Alaska  AK   West     710231
> class(murders[1:2,1:4])
[1] "data.frame"
```

d.

Explain why all entries in `as.matrix(murders[1:2,1:4])` are character entries.

answer: the variables in the data frame `murders[1:2,1:4]` are character, factor and numeric types. A matrix can only have one type of entry, so all entries are coerced into the most complex type which is character.

Problem 5: Lists

Consider the list below.

```
> mylist <- list(x1="sally", x2=42, x3=FALSE, x4=1:5)
```

Show how to produce the following output in **one command**:

a.

`"sally"` (atomic character vector of length 1)

answer: The book uses `mylist[["x1"]]` to get `"sally"`. Possible other ways include `mylist$x1`, `mylist[[1]]` and `mylist[1][[1]]`. The last command (which I wouldn't suggest using) returns a list with "x1" as its only entry, you then access the into in the first entry with `[[1]]`.

b.

`42` (atomic numeric vector of length 1)

answer: The book uses `mylist$x2` to get 42. Other ways include `mylist[[2]]`, `mylist[["x2"]]` and `mylist[2][[1]]`

c.

the 3rd and 4th entries in `x4` (atomic numeric vector of length 2)

answer: Possible ways include `mylist$x4[3:4]`, `mylist[[4]][3:4]`, `mylist[["x4"]][3:4]`

d.

the length of `x4`

answer: The book uses `length(mylist[["x4"]])`. Possible other ways include `length(mylist$x4)` and `length(mylist[[4]])`

Code to check work:

```
> # 1
> mylist$x1
[1] "sally"
> mylist[1][[1]]
[1] "sally"
> mylist[[1]]
[1] "sally"
> # 2
> mylist[[2]]
[1] 42
> mylist[["x2"]]
[1] 42
> mylist[2][[1]]
[1] 42
> # 3
> mylist$x4[3:4]
[1] 3 4
> mylist[[4]][3:4]
[1] 3 4
> mylist[["x4"]][3:4]
[1] 3 4
> # 4
> length(mylist$x4)
[1] 5
> length(mylist[[4]])
[1] 5
```

Problem 6: More lists

Use the same list as problem 5. Describe the class of object that is produced with each of the following commands:

a.

`mylist[1]`

answer: `mylist[1]` returns the first entry (which is a character vector of length 1) in `mylist` in list form (it preserved the original class of `mylist`).

b.

`mylist[[1]]`

answer: `mylist[1]` returns an object that equals the first entry in `mylist`, so the type of object that it returns is a character vector of length 1.

c.

`unlist(mylist)`

answer: `unlist(mylist)` returns a vector of all values stored in `mylist`. Because there is one character entry, the type of vector will be character.

Code to check work:

```
> str(mylist[1])
List of 1
 $ x1: chr "sally"
> str(mylist[[1]])
 chr "sally"
> str(unlist(mylist))
 Named chr [1:8] "sally" "42" "FALSE" "1" "2" "3" "4" "5"
 - attr(*, "names")= chr [1:8] "x1" "x2" "x3" "x41" ...
```