Stats 102A - Homework 1 - Output File

Daren Sathasivam

To receive full credit the functions you write must pass all tests. We may conduct further tests that are not included on this page as well.

Academic Integrity Statement

By including this statement, I, **Daren Sathasivam**, declare that all of the work in this assignment is my own original work. At no time did I look at the code of other students nor did I search for code solutions online. I understand that plagiarism on any single part of this assignment will result in a 0 for the entire assignment and that I will be referred to the dean of students.

```
source("102A_hw_01_script_Daren_Sathasivam.R")  # edit with your file name
```

Part 1: Tests for `by_type()`

Do not edit any of the code in this section. The code here will run test cases to test if your function is working properly.

```
x <- c("a", "1", "2.2", "house", "3.4", "6")
by_type(x)
```

```
## $integers
## [1] 1 6
##
## $doubles
## [1] 2.2 3.4
##
## $character
## [1] "a"     "house"
```

```
# 10 * 0.3 is 3. There are two instances of 3
# Both instances of 3 will be classified as an integer.
set.seed(1)
x <- sample(c(1:10, (1:10)*0.3, letters[1:10]))
by_type(x)
```

```
## $integers
##  [1]  4  7  1  2 10  9  5  8  6  3  3
##
## $doubles
## [1] 0.3 1.2 2.4 2.7 1.5 0.6 0.9 2.1 1.8
##
## $character
##  [1] "e" "c" "g" "j" "a" "h" "b" "f" "i" "d"
```

```r
by_type(x, sort = TRUE)
```

```
## $integers
##  [1]  1  2  3  3  4  5  6  7  8  9 10
##
## $doubles
## [1] 0.3 0.6 0.9 1.2 1.5 1.8 2.1 2.4 2.7
##
## $character
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"
```

```r
x <- 10:1
by_type(x, sort = TRUE)
```

```
## $integers
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## $doubles
## numeric(0)
##
## $character
## character(0)
```

```r
x <- logical(5)
by_type(x)
```

```
## $integers
## [1] 0 0 0 0 0
##
## $doubles
## numeric(0)
##
## $character
## character(0)
```

```r
# values 1.0, 2.0, 3.0, etc. are classified as integers.
x <- seq(0, 4, by = 0.2)
by_type(x)
```

```
## $integers
## [1] 0 1 2 3 4
##
## $doubles
##  [1] 0.2 0.4 0.6 0.8 1.2 1.4 1.6 1.8 2.2 2.4 2.6 2.8 3.2 3.4 3.6 3.8
##
## $character
## character(0)
```

```r
x <- c(seq(0, 4, by = 0.2), NA, "a", "d")
by_type(x, sort = TRUE)
```

```
## $integers
## [1] 0 1 2 3 4
##
## $doubles
##  [1] 0.2 0.4 0.6 0.8 1.2 1.4 1.6 1.8 2.2 2.4 2.6 2.8 3.2 3.4 3.6 3.8
##
```

```
## $character
## [1] "a" "d"
```

Part 2: Tests for `prime_factor()`

This section will print out a bunch of test cases to see if your `prime_factor()` function is working properly. Do not edit the following code which runs tests.

```
# a helper function for testing if a value x is prime.
is_prime <- function(x){
  if(!is.numeric(x)){
    stop("Input must be numeric")
  }
  if(length(x) != 1){
    stop("Input must have length of 1")
  }
  if(as.integer(x) != x){ # non-integer values are not prime
    return(FALSE)
  }
  if(x <= 1){ # negative numbers, 0, and 1 are all non-prime
    return(FALSE)
  }
  if(x == 2){ # special case for 2
    return(TRUE)
  }
  # we only need to check for prime factors less than sqrt(x)
  max <- ceiling(sqrt(x))
  for(i in 2:max){
    if(x %% i == 0){ # if the values divides evenly, it is not prime
      return(FALSE)
    }
  }
  TRUE
}
```

```
prime_factor(0)
```

```
## Error in prime_factor(0): Invalid value!
```

```
prime_factor(1)
```

```
## Error in prime_factor(1): Invalid value!
```

```
prime_factor(2.2)
```

```
## Error in prime_factor(2.2): Invalid value!
```

```
for(x in c(2:50, 2001:2050)){
  factors <- prime_factor(x)
  cat(x, "has prime factors:", factors, "\n")
  if(!all(sapply(factors, is_prime))){
    cat("Problem: not all factors found are prime.\n")
  }
  if(prod(factors) != x){
    cat("Problem: the product of factors does not equal x.\n")
  }
```

```
}
```

```
## 2 has prime factors: 2
## 3 has prime factors: 3
## 4 has prime factors: 2 2
## 5 has prime factors: 5
## 6 has prime factors: 2 3
## 7 has prime factors: 7
## 8 has prime factors: 2 2 2
## 9 has prime factors: 3 3
## 10 has prime factors: 2 5
## 11 has prime factors: 11
## 12 has prime factors: 2 2 3
## 13 has prime factors: 13
## 14 has prime factors: 2 7
## 15 has prime factors: 3 5
## 16 has prime factors: 2 2 2 2
## 17 has prime factors: 17
## 18 has prime factors: 2 3 3
## 19 has prime factors: 19
## 20 has prime factors: 2 2 5
## 21 has prime factors: 3 7
## 22 has prime factors: 2 11
## 23 has prime factors: 23
## 24 has prime factors: 2 2 2 3
## 25 has prime factors: 5 5
## 26 has prime factors: 2 13
## 27 has prime factors: 3 3 3
## 28 has prime factors: 2 2 7
## 29 has prime factors: 29
## 30 has prime factors: 2 3 5
## 31 has prime factors: 31
## 32 has prime factors: 2 2 2 2 2
## 33 has prime factors: 3 11
## 34 has prime factors: 2 17
## 35 has prime factors: 5 7
## 36 has prime factors: 2 2 3 3
## 37 has prime factors: 37
## 38 has prime factors: 2 19
## 39 has prime factors: 3 13
## 40 has prime factors: 2 2 2 5
## 41 has prime factors: 41
## 42 has prime factors: 2 3 7
## 43 has prime factors: 43
## 44 has prime factors: 2 2 11
## 45 has prime factors: 3 3 5
## 46 has prime factors: 2 23
## 47 has prime factors: 47
## 48 has prime factors: 2 2 2 2 3
## 49 has prime factors: 7 7
## 50 has prime factors: 2 5 5
## 2001 has prime factors: 3 23 29
## 2002 has prime factors: 2 7 11 13
## 2003 has prime factors: 2003
```

```
## 2004 has prime factors: 2 2 3 167
## 2005 has prime factors: 5 401
## 2006 has prime factors: 2 17 59
## 2007 has prime factors: 3 3 223
## 2008 has prime factors: 2 2 2 251
## 2009 has prime factors: 7 7 41
## 2010 has prime factors: 2 3 5 67
## 2011 has prime factors: 2011
## 2012 has prime factors: 2 2 503
## 2013 has prime factors: 3 11 61
## 2014 has prime factors: 2 19 53
## 2015 has prime factors: 5 13 31
## 2016 has prime factors: 2 2 2 2 2 3 3 7
## 2017 has prime factors: 2017
## 2018 has prime factors: 2 1009
## 2019 has prime factors: 3 673
## 2020 has prime factors: 2 2 5 101
## 2021 has prime factors: 43 47
## 2022 has prime factors: 2 3 337
## 2023 has prime factors: 7 17 17
## 2024 has prime factors: 2 2 2 11 23
## 2025 has prime factors: 3 3 3 3 5 5
## 2026 has prime factors: 2 1013
## 2027 has prime factors: 2027
## 2028 has prime factors: 2 2 3 13 13
## 2029 has prime factors: 2029
## 2030 has prime factors: 2 5 7 29
## 2031 has prime factors: 3 677
## 2032 has prime factors: 2 2 2 2 127
## 2033 has prime factors: 19 107
## 2034 has prime factors: 2 3 3 113
## 2035 has prime factors: 5 11 37
## 2036 has prime factors: 2 2 509
## 2037 has prime factors: 3 7 97
## 2038 has prime factors: 2 1019
## 2039 has prime factors: 2039
## 2040 has prime factors: 2 2 2 3 5 17
## 2041 has prime factors: 13 157
## 2042 has prime factors: 2 1021
## 2043 has prime factors: 3 3 227
## 2044 has prime factors: 2 2 7 73
## 2045 has prime factors: 5 409
## 2046 has prime factors: 2 3 11 31
## 2047 has prime factors: 23 89
## 2048 has prime factors: 2 2 2 2 2 2 2 2 2 2 2
## 2049 has prime factors: 3 683
## 2050 has prime factors: 2 5 5 41
```

Part 3: Tests for `month_convert()`

Do not edit the following code which runs tests.

```
month_names <- read.delim("month_names.txt", encoding="UTF-8", row.names=1)
```

```
x <- factor(c("March", "March", "February", "June"))
month_convert(x, "English", "Spanish")
```

```
## [1] marzo   marzo   febrero junio
## Levels: febrero junio marzo
```

```
x <- factor(c("March", "March", "February", "June", "Jaly", "Hamburger", "December"))
month_convert(x, "English", "German")
```

```
## [1] März     März     Februar  Juni     <NA>     <NA>     Dezember
## Levels: Dezember Februar Juni März
```

```
x <- factor(c("gennaio", "febbraio", "marzo", "aprile", "maggio", "giugno", "luglio",
              "agosto", "settembre", "ottobre", "novembre", "dicembre"))
month_convert(x, "Italian", "English")
```

```
##  [1] January   February  March     April     May       June      July
##  [8] August    September October   November  December
## 12 Levels: April August December February January July June March ... September
```

```
x <- factor(c("janeiro", "março", "abril", "maio", "junho", "julho", "maio", "setembro",
              "outubro", "novembro", "dezembro", "setembro", "setembro", "março"))
y <- month_convert(x, "Portuguese", "French")
print(y)
```

```
##  [1] janvier   mars      avril     mai       juin      juillet   mai
##  [8] septembre octobre   novembre  décembre  septembre septembre mars
## 10 Levels: avril décembre janvier juillet juin mai mars novembre ... septembre
```

```
y <- month_convert(y, "French", "Danish")
print(y)
```

```
##  [1] januar    marts     april     maj       juni      juli      maj
##  [8] september oktober   november  december  september september marts
## 10 Levels: april december januar juli juni maj marts november ... september
```

```
y <- month_convert(y, "Danish", "Dutch")
print(y)
```

```
##  [1] januari   maart     april     mei       juni      juli      mei
##  [8] september oktober   november  december  september september maart
## 10 Levels: april december januari juli juni maart mei november ... september
```

```
y <- month_convert(y, "Dutch", "Icelandic")
print(y)
```

```
##  [1] janúar    mars      apríl     maí       júní      júlí      maí
##  [8] september október   nóvember  desember  september september mars
## 10 Levels: apríl desember janúar júlí júní maí mars nóvember ... september
```

Part 4: Questions to Answer

Replace 'write your answer here' with your responses. Be sure your answers have been 'highlighted' using the triple hash ### which makes the text large and bold.

1. Coercion: For each of the following, explain what type of output you will receive and why R is producing that output.
   a. `c(0, TRUE)`

b. `c("F", F)`
c. `c(list(1), "b")`
d. `c(FALSE, 1L)`

a. will be of type double because 0 is type double and it will coerece the TRUE to type double.

b. Will be coerced into the character type because "F" is type character and it will coerce the logical type F to character type.

c. Will be coerced into type list because list(1) will coerce the character type "b" into list by adding the character into the list as an element.

d. Will be coerced into type integer because 1L is type integer and will coerce logical type FALSE to integer type as it becomes 0.

2. What is the difference between NULL, NA, and NaN?

All of these special values have different meanings and uses according to situations. NULL refers to an empty or nonexistent value and it is its own type. NA refers to missing or unknown values and are present for each data type. NaN refers to indeterminant forms in mathematics(such as 0/0) and is of type double.

3. What is the difference between logical(0) and NULL? Write a command (other than `logical(0)`) that will produce logical(0) as the output. Write a command (other than `NULL`) that will produce NULL as the output.

logical(0) represent an empty logical vector whereas NULL represents an absence of a value and can be used in functions to indicate the a lack of a meaningful result.

```
# For logical(0)
as.logical(numeric(0))
```

```
## logical(0)
```

```
# For NULL
as.null(numeric(0))
```

```
## NULL
```

4. A vector `c(TRUE, FALSE)` is a logical vector. Other than `TRUE` or `FALSE`, what can you insert into the vector so that it increases to a length of 3 and remains a logical vector and does not get coerced into another class?

A user can insert a value of either T or F at the end as they represent TRUE and FALSE values. Additionally, NA can also be inserted and would represent an unknown value or NULL which is considered FALSE.

```
# Ex.
c(TRUE, FALSE, T, F, NA, NULL)
```

```
## [1]  TRUE FALSE  TRUE FALSE    NA
```

5. What are the lengths of the following lists? Use bracket notation to subset them to the equivalent of `c("h","i")`. Be sure to print the result so it shows the subset.

```
l1 <- list(letters[1:5], letters[3:9] , letters[4:7])
l1
```

```
## [[1]]
## [1] "a" "b" "c" "d" "e"
##
## [[2]]
## [1] "c" "d" "e" "f" "g" "h" "i"
##
## [[3]]
## [1] "d" "e" "f" "g"
```

```
l1[[2]][c(6, 7)]
```

```
## [1] "h" "i"
```

```
l2 <- list( c(letters[1:5], letters[3:9]), letters[4:7] )
l2
```

```
## [[1]]
##  [1] "a" "b" "c" "d" "e" "c" "d" "e" "f" "g" "h" "i"
##
## [[2]]
## [1] "d" "e" "f" "g"
```

```
l2[[1]][c(11, 12)]
```

```
## [1] "h" "i"
```

The length of list 1 is 3. The length of list 2 is 2. In list 3, there are three components whereas list 2 combines the character vector resulting in two components instead of three.

6. What will `c(4:7) * c(2:4)` produce? Briefly, why?

The above multiplication can be broken down to two integer vectors multiplying by each other. (4, 5, 6, 7) * (2, 3, 4) will result in `8 15 24 14` due to recycling because of differing lengths. The 2 will be used to multiply with 7 resulting in 14. In addition to the output, a warning message of the recycling will be displayed.

7. Take a look at the following code chunks. What are some of the differences between `cat()` and `print()`?

```
cat(5 + 6)
```

```
## 11
```

```
print(5 + 6)
```

```
## [1] 11
```

```
x8 <- cat(5 + 6)
```

```
## 11
```

```
y8 <- print(5 + 6)
```

```
## [1] 11
```

```
x8
```

```
## NULL
```

```
y8
```

```
## [1] 11
```

```r
cat(letters[1:3], letters[24:26])
```

```
## a b c x y z
```

```r
print(letters[1:3], letters[24:26]) # Why are we getting the following error?
```

```
## Warning in print.default(letters[1:3], letters[24:26]): NAs introduced by
## coercion
```

```
## Error in print.default(letters[1:3], letters[24:26]): invalid printing digits -2147483648
```

```r
# Error in print.default(letters[1:3], letters[24:26]) : invalid 'digits' argument
cat(l1)
```

```
## Error in cat(l1): argument 1 (type 'list') cannot be handled by 'cat'
```

```r
print(l1)
```

```
## [[1]]
## [1] "a" "b" "c" "d" "e"
##
## [[2]]
## [1] "c" "d" "e" "f" "g" "h" "i"
##
## [[3]]
## [1] "d" "e" "f" "g"
```

The cat() function concatenates and produces an output without extra information. print() is designed for displaying objects and provides an organized and informational output. x8 returns NULL due to cat() ability to print but not store values whereas y8 returns the previously outputted value because of its ability to print objects printed previously. print() results in an error due to its inabillity to handle multiple arguments whereas cat() is designed for concantenation and will output the multiple arguments placed within the function. Lastly, cat(l1) results in an error because it is not designed to handle additional information that lists contain whereas print() will output the list with it retaining information and values.

8. What happens to a factor when you reverse its levels?

```r
f1 <- factor(c("A","A","B","C","D","A","C"))
f1
```

```
## [1] A A B C D A C
## Levels: A B C D
```

```r
levels(f1) <- rev(levels(f1))
f1
```

```
## [1] D D C B A D B
## Levels: D C B A
```

When the levels are reversed, the original vector does not change. However, the corresponding levels are reversed and take the place of the level beforehand such as "D" corresponding to where the values of "A" were and so forth. "A"(originally 1) becomes "D"(new 1) which is then represented in the factor. So the actual values of the factor are not modified.

9. How do f2 and f3 differ from the unmodified f1?

```r
f1 <- factor(c("A","A","B","C","D","A","C"))
f1
```

```
## [1] A A B C D A C
## Levels: A B C D
```

```
f2 <- factor(rev(c("A","A","B","C","D","A","C")))
f2
```

```
## [1] C A D C B A A
## Levels: A B C D
```

```
f3 <- factor(c("A","A","B","C","D","A","C"), levels = rev(c("A","B","C","D")))
f3
```

```
## [1] A A B C D A C
## Levels: D C B A
```

All three contain the same elements in the vector. f2 reverses the actual factor so the levels remains the same and the mapping of the values remains the same. f3 reverses the levels but not the factor which then results in the elements "A" in the factor mapping to the level "D"

10. What attributes does a data frame possess?

A data fram can possess multiple attributes consisting of names, row.names, col.names, class, dimensions, and comments. Attributes can be obtained through the use of the attributes() function.

11. What does as.matrix() do when applied to a data frame with columns of different types? Create a simple data.frame with two columns: one numeric and one string. Use as.matrix and show the results.

When the as.matrix() function is applied to a data frame with different column types, the function attempts to coerce the elements into a common data type. In the case of a numeric data type and and a string type, the matrix will be coerced into the character data type.

```
df <- data.frame(
  num_col = c(1, 2, 3),
  string_col = c("A", "B", "C")
)
print(df)
```

```
##   num_col string_col
## 1       1          A
## 2       2          B
## 3       3          C
```

```
print(as.matrix(df)) # Coerced into character data type
```

```
##      num_col string_col
## [1,] "1"     "A"
## [2,] "2"     "B"
## [3,] "3"     "C"
```