Week 3 Exercises

Amanda Frithsen

July 20, 2024

Please complete all exercises below. You may use any library that we have covered in class UP TO THIS POINT.

For each function, show that it works, by using the provided data as a test and by feeding in some test data that you create to test your function

Add comments to your function to explain what each line is doing

1.) Write a function that takes in a string with a person's name in the form

"Sheets, Dave"

and returns a string of the form

"Dave Sheets"

Note:

-assume no middle initial ever -remove the comma -be sure there is white space between the first and last name

You will probably want to use stringr

```r
#import stringr package
library(stringr)

name_in="Sheets, Dave"

reorder_name<-function(last_first)
{
  # the function takes an input in the form last, first and returns first last

  # if else to ensure that the input is in a string in the form "last, first"
  if((is.character(last_first))&(grepl("*, *", last_first)))
  {

    # split name into a 1x2 matrix of substrings
    split_name <- str_split_fixed(last_first, ", ", n = 2)

    # join substrings from matrix in desired order
    name_out <- str_c(split_name[1, 2], " ", split_name[1, 1])
    # return output of string in the form first last
    return(name_out)

    # run error text explaining incorrect input
  } else
```

```
    {
    stop("ERROR: Input must be a string written in the form 'last name, first name'.")
    }
}

reorder_name(name_in)
```

```
## [1] "Dave Sheets"
```

2.) Write a function that takes in a string of values x, and returns a data frame with three columns, x, x^2 and the square root of x

```
x <- c(1,3,5,7,9,11,13)

powers_df<-function(x)
{
  # the function takes an input of a list of values and returns a data frame with three columnns - the

  #if else to ensure that input is a list of numeric or integer values
  if((class(x) =="numeric") | (class(x) == "integer"))
  {
    # create data frame with values, squares, and square roots
    powers <- data.frame(value = x, squares = x^2, roots = x^(1/2))

    # return output of data frame created
  return(powers)
  }

  #run error if incorrect type of input
  else
    {
    stop("ERROR: Input must be a numeric vector.")
    }
}

powers_df(x)
```

```
##   value squares    roots
## 1     1       1 1.000000
## 2     3       9 1.732051
## 3     5      25 2.236068
## 4     7      49 2.645751
## 5     9      81 3.000000
## 6    11     121 3.316625
## 7    13     169 3.605551
```

3) Two Sum - Write a function named two_sum()

Given a vector of integers nums and an integer target, return indices of the two numbers such that they add up to target.

You may assume that each input would have exactly one solution, and you may not use the same element twice.

You can return the answer in any order.

Example 1:

Input: nums = [2,7,11,15], target = 9 Output: [1, 2] Explanation: Because nums[1] + nums[2] == 9, we return [1, 2].

Example 2:

Input: nums = [3,2,4], target = 6 Output: [2, 3]

Example 3:

Input: nums = [3,3], target = 6 Output: [1, 2]

Constraints:

2 <= nums.length <= 104 –109 <= nums[i] <= 109 –109 <= target <= 109 Only one valid answer exists.

Note: For the first problem I want you to use a brute force approach (loop inside a loop)

The brute force approach is simple. Loop through each element x and find if there is another value that equals to target – x

Use the function seq_along to iterate

```
two_sum <- function(nums_vector,target)
{
# this function returns a matrix with each row representing a pair of indices; the indices indicate whi

# sum_indices is an empty vector that will "collect" the pairs with a sum of the target as the function
    sum_indices <- c()

#if else to ensure that the nums_vector and target are integer values
  if (!is.integer(nums_vector)|!is.integer(target))
      {
    cat("Please be sure the inputs are integers")
    return(NULL)
      }
  else
  {
# loop through each element in the nums_vector
          for (j in seq_along(nums_vector))
            {
# loop through each element in the nums_vector again to see which (if any) sums with the element from t
            for (k in seq_along(nums_vector))
              {
# check if the target sum is achieved; if TRUE, the indices of these values will be stored in the sum_i
              if (nums_vector[j] == target - nums_vector[k])
                {
                sum_indices <- c(sum_indices, j, k)
                }
              }
            }

# return the sum indices in a matrix with two columns to ensure that each row represents a possible pai
          return(matrix(sum_indices, ncol = 2, byrow = TRUE))
  }
}
```

```
# Test code
nums_vector <- as.integer(c(5,7,12,34,6,10,8,9))
target <- as.integer(13)

z=two_sum(nums_vector,target)
print(z)
```

```
##      [,1] [,2]
## [1,]    1    7
## [2,]    2    5
## [3,]    5    2
## [4,]    7    1
```

```
nums_vector_2 <- c(1L, 3L, 5L, 6L, 9L, 10L)
target_2 <- 11L

y=two_sum(nums_vector_2, target_2)
print(y)
```

```
##      [,1] [,2]
## [1,]    1    6
## [2,]    3    4
## [3,]    4    3
## [4,]    6    1
```

```
#expected answers
#[1] 1 7
#[1] 2 5
#[1] 5 2
```

Okay tough problem coming here! Look carefully at Jeremiah's examples of using a hash to simplify a loop by using a hash.

4) Now write the same function using hash tables.

Loop the array once to make a hash map of the value to its index. Then loop again to find if the value of target-current value is in the map.

The keys of your hash table should be each of the numbers in the nums_vector minus the target.

A simple implementation uses two iterations. In the first iteration, we add each element's value as a key and its index (array position) as a value to the hash table. Then, in the second iteration, we check if each element's complement (target – nums_vector[i]) exists in the hash table. If it does exist, we return current element's index and its complement's index. Beware that the complement must not be nums_vector[i] itself!

```
require(hash)
```

```
## Loading required package: hash
```

```
## hash-2.2.6.3 provided by Decision Patterns
```

```r
two_sum2 <- function(nums_vector, target)
{
# Create an empty hash table
  h <- hash()
# sum_indices is an empty vector that will "collect" the pairs with a sum of the target as the function
  sum_indices <- c()

# Add each element's value as a key to the hash table with its index as the value
  for(j in seq_along(nums_vector))
  {
    h[as.character(nums_vector[j])] <- j   # Note: key must be character value
  }

# check to see if the element's complement exists in the hash table
  for(k in seq_along(nums_vector))
    {
       complement <- target - nums_vector[k]
       if (is.element(as.character(complement), keys(h)))
       {
# if element's complement does exist, add the index of the element and the index of the element's compl
         complement_index <- h[[as.character(complement)]]
         sum_indices = rbind(sum_indices, c(k, complement_index))
       }
    }
# return the indices of each pair that has a sum of the target
  return(sum_indices)
}


# Test code
nums_vector <- as.integer(c(5,7,12,34,6,10,8,9))
target <- as.integer(15)

two_sum2(nums_vector,target)
```

```
##      [,1] [,2]
## [1,]    1    6
## [2,]    2    7
## [3,]    5    8
## [4,]    6    1
## [5,]    7    2
## [6,]    8    5
```

```r
#expected answers
#[1] 1 6
#[1] 2 7
#[1] 5 8
#[1] 6 1
#[1] 7 2
#[1] 8 5
```