Week 3 Exercises

Anthony V. Razzano, DHA

2024-10-10

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

```
library(stringr)


name_in="Sheets, Dave"

reorder_name<-function(last_first){
  # Split the input string by the comma
  split_name <- str_split(last_first, ",")[[1]]
  # Trim whitespace and create the desired format
  return(paste(trimws(split_name[2]), trimws(split_name[1])))
}

# Test the function
reorder_name(name_in)  # "Dave Sheets"
```

```
## [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

```r
x=c(1,3,5,7,9,11,13)

powers_df<-function(x) {
  # Create a data frame with the required calculations
  data.frame(
    x = x,
    x_squared = x^2,
    sqrt_x = sqrt(x)
  )
}

# Test the function
powers_df(x)
```

```
##    x x_squared   sqrt_x
## 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.) Write in a function that takes in a value x and returns

y= 0.3x if x<0
y=0.5x if x>=0

This is a variant on a relu function as used in some neural networks.

```r
relu_variant <- function(x) {
  # Return the calculated value based on the condition
  if (x < 0) {
    return(0.3 * x)
  } else {
    return(0.5 * x)
  }
}

# Test the function
relu_variant(-5)  # -1.5
```

```
## [1] -1.5
```

```r
relu_variant(4)   # 2
```

```
## [1] 2
```

4.) Write a function that takes in a value x and returns the first power of two greater than x (use a While loop)

```r
next_power_of_two <- function(x) {
  # Start with 1 (2^0) and keep doubling until we find a power of two greater than x
  power <- 1
  while (power <= x) {
    power <- power * 2
  }
  return(power)
}

# Test the function
next_power_of_two(5)  # 8
```

```
## [1] 8
```

5) 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: [0,1] Explanation: Because nums[0] + nums[1] == 9, we return [0, 1]. Example 2:

Input: nums = [3,2,4], target = 6 Output: [1,2] Example 3:

Input: nums = [3,3], target = 6 Output: [0,1]

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

```r
two_sum <- function(nums_vector, target) {
  # Loop through each element in the vector
  for (i in seq_along(nums_vector)) {
    for (j in seq_along(nums_vector)) {
      # Check if the sum equals the target
      if (i != j && nums_vector[i] + nums_vector[j] == target) {
        return(c(i - 1, j - 1))  # return 0-based index
      }
    }
  }
  return(NULL)  # Return NULL if no solution found
}

# Test code
```

```
nums_vector <- c(5, 7, 12, 34, 6, 10, 8, 9)
target <- 13

z = two_sum(nums_vector, target)
print(z)  # Expected answers: [1] 1 7 or [1] 2 5
```

```
## [1] 0 6
```

6.) Write one piece of code that will use a regex command to extract a phone number written in the form

```
456-123-2329
```

The sentences to use are located below

use the str_extract function from stringr

use the same regex search pattern from each

-What does \d match to? or alternatively [[:digit:]]

-How do you specify a specific number of repeated characters

```
library(stringr)

# Sentences to search
a = "Please call me at 456-123-2329, asap"
b = "Hey, we have a code 234 on machine a-234-12, call me at 678-321-98766"
c = "On 12-23-2022, Joe over at 122 Turnpike, dialled 912-835-4756, tell me by 9:02 pm Wed"

# Regex to extract phone numbers
phone_numbers <- str_extract(c(a, b, c), "\\d{3}-\\d{3}-\\d{4}")

# Print extracted phone numbers
print(phone_numbers)
```

```
## [1] "456-123-2329" "678-321-9876" "912-835-4756"
```

```
# \\d matches any digit; [[:digit:]] is an alternative to \\d
# To specify a specific number of repeated characters, use {n}, where n is the number of repetitions
```

7.) For lines below, extract the domains (ie the part of the address after @)

```
# Email addresses
d = "jimmy.halibut@gmail.com"
e = "His address is: c.brown@hopeles.org, do write him"
f = "h.potter@hogwarts.edu is bouncing back on me, I wonder why?"

# Extract domains using regex
domains <- str_extract(c(d, e, f), "(?<=@)[^ ]+")
print(domains)  # Output the domains
```

```
## [1] "gmail.com"    "hopeles.org," "hogwarts.edu"
```