Assignment 1

Alex Johnson

9/12/2021

Contents

0.1 sum_mults(nums, n)

Write a function sum_mults(nums, n) that returns the sum of all multiples of values in the vector nums less than n. For example, sum_mults(c(3,5), 30) should return 195. Assume that the elements of nums are positive integers.

```
nums <- c(3, 5)
n <- 30

sum_mults <- function(nums, n){
    if(n < 1)
        return(0)
    sum <- 0
    for(i in 1:n-1){
        for(j in nums){
            if(i %% j == 0){
                sum = sum + i
                break
            }
        }
    }
    return(sum)
```

```
}

sum_mults(nums, n)
```

```
## [1] 195
```

```
# l <- 1:(n-1)
# nums <- c(3,5)
# sum(l[sapply(l, function(x) any(x %% nums == 0))])
```

0.2 collatz_len(n)

Given any positive integer n, define:

$$f(n) := \begin{cases} n/2, & n \text{ even} \\ 3n+1, & n \text{ odd.} \end{cases} \tag{1}$$

The Collatz conjecture states:

$$a_i := \begin{cases} n, & i = 1 \\ f(a_{i-1}), & i > 1 \end{cases} \tag{2}$$

Write a function collatz_len(n) that determines the first i for which $a_i = 1$ for a given n. For example, when n = 17, the sequence a_i begins:

$$17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1, \ldots$$

and $a_{13} = 1$. Thus, collatz_len(17) should return 13.

```
n <- 17

collatz_len <- function(n){
  n1 <- n
  index <- 1
  while(n1 != 1){
    if(n1 %% 2 == 0)
      n1 <- n1/2
    else
      n1 <- 3*n1+1
    index <- index + 1
  }
  return(index)
}

collatz_len(n)
```

```
## [1] 13
```

0.3 reverse(v)

Write a function reverse(v) that reverses the vector v. So reverse(c(1,2,3)) should return c(3,2,1). reverse should return NULL, not NA, when v is c().

```
v <- 1:3

reverse <- function(v){
  if(length(v) == 0) # If the list is empty, return NULL
    return(NULL)
  if(length(v))
    res <- v[length(v):1]
  else
    res <- v
  return(res)
}

reverse(v)
```

```
## [1] 3 2 1
```

0.4 drop(v, n)

Write a function drop(v, n) that drops every n^{th} element from the vector v. drop(c(1,2,3,4,5), 2) should return c(1,3,5). drop should return NULL when n 1.

```
v <- 1:5
n <- 2

drop <- function(v, n){
  if(n <= 1)
    return(NULL)
  v[seq(1, length(v), n)]
}

drop(v, n)
```

```
## [1] 1 3 5
```

0.5 intersect_3(v, w, x)

Write a function intersect_3(v, w, x) that returns a vector of the elements that appear in each of the vectors v, w, and x. intersect_3(c(1,2,3,1), c(1,1,3,2), c(3,1,9,1)) should return c(1,3).

```
v <- c(1, 2, 3, 1)
w <- c(1, 1, 3, 2)
x <- c(3, 1, 9, 1)

intersect_3 <- function(v, w, x){
  res <- intersect(v, w)
  res <- intersect(res, x)
```

```
  return(res)
}

intersect_3(v, w, x)
```

```
## [1] 1 3
```

0.6 filter_vec(v, p)

Write a function filter_vec(v, p) that returns a vector containing all the elements of v for which the predicate function p returns TRUE. For example:

\begin{center} p <- function(x){ return(x>3) } l <- 1:6 m <- filter_vec(l, p) \end{center}

results in m being equal to the vector c(4,5,6). Make sure that filter_vec returns NULL when p is FALSE for all elements of v.

```
v <- 1:6
p <- function(x){ return(x>3) }

filter_vec <- function(v, p){
  if(all(p(v) == FALSE))
    return(NULL)
  v[p(v)]
}

filter_vec(v, p)
```

```
## [1] 4 5 6
```

0.7 n_fibs(n)

Write a function n_fibs(n) that creates a vector of the first n Fibonacci numbers where the first and second Fibonacci numbers are 1 (n_fibs(2) returns c(1,1)).

```
n <- 6
v <- vector()

n_fibs <- function(n){
  v[[1]] <- 1
  v[[2]] <- 1
  if(n <= 2)
    return(v)
  for(i in 3:n)
    v[i] <- v[i-1] + v[i-2]
  return(v)
}

n_fibs(n)
```

```
## [1] 1 1 2 3 5 8
```

0.8 shift(v, n)

Write a function shift(v, n) that shifts the elements of a vector n places to the right. If n is negative, the function should shift the vector to the left. For example, shift(c(1,2,3,4), 2) should return c(3,4,1,2) while shift(c(1,2,3,4), -3) should return c(4,1,2,3).

```
v <- 1:4
n <- 2 # Does not function properly with negatives...

shift <- function(v, n){
  n <- n %% length(v)
  l <- c(v[(n+1):length(v)], v[1:n])
    return(l)
}

shift(v, n)
```

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

0.9 rem_consec_dups(v)

Write a function rem_consec_dups(v) that removes consecutive duplicates from the vector v. rem_consec_dups(c(1,1,1,2,3,3,1,2,2)) should return c(1,2,3,1,2). Do not use the built-in function rle in your solution.

```
v <- c(1, 1, 1, 2, 3, 3, 1, 2, 2)

rem_consec_dups <- function(v){
  lastEntry <- v[-length(v)] # v without last entry
  firstEntry <- v[-1]        # v without first entry
  v[c(TRUE, !lastEntry == firstEntry)] # Compare each entry with its next entry and index.
}

rem_consec_dups(v)
```

```
## [1] 1 2 3 1 2
```

0.10 n_even_fibs(n)

Write a function n_even_fibs(n) that creates a list of the first n even Fibonacci numbers. The name of each value should be its position in the Fibonacci sequence as a string. For example, n_even_fibs(5) should create a list with the structure

\begin{center} List of 5 $ 3 : num 2 $ 6 : num 8 $ 9 : num 34 $ 12: num 144 $ 15: num 610 \end{center}

```
n <- 5
v <- vector()
temp <- vector()

n_even_fibs <- function(n){
  n <- n*3
  v[[1]] <- 1
```

```
  v[[2]] <- 1
  if(n <= 2)
    return(v)
  for(i in 3:n)
    v[i] <- v[i-1] + v[i-2]
  cat("List of", n/3)
  cat("\n")
   for(j in 1:n){
    if(v[j] %% 2 == 0){ # Giant pile of spaghetti printing methodology
      cat("  $ ")       # because I don't know what I'm doing...
      cat(j)
      if(j %% 10 == j)
        cat(" : num ")
      else
        cat(": num ")
      cat(v[j])
      cat("\n")
    }
  }
}
n_even_fibs(n)
```

```
## List of 5
##   $ 3 : num 2
##   $ 6 : num 8
##   $ 9 : num 34
##   $ 12: num 144
##   $ 15: num 610
```