# Problems from crumplab.com

Ivar Hereide Mannsåker

2022-11-01

## Easy Problems

### Problem 3

Write code that will place the numbers 1 to 100 separately into a variable using for loop. Then, again using the seq function.

```
One2Hundred <- c()
for (i in 1:100){
One2Hundred[i] <- i
}
One2seq <- seq(1,100)

print(One2Hundred)
```

```
##   [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
##  [19]  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
##  [37]  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
##  [55]  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
##  [73]  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
##  [91]  91  92  93  94  95  96  97  98  99 100
```

```
print(One2seq)
```

```
##   [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
##  [19]  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
##  [37]  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
##  [55]  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
##  [73]  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
##  [91]  91  92  93  94  95  96  97  98  99 100
```

### Problem 4

Find the sum of all the integer numbers from 1 to 100

With sum() function

```
Summate <- sum(One2Hundred)
print(Summate)
```

```
## [1] 5050
```

With a loop

```
SummateLoop <- 0L
for (i in 1:100) {
SummateLoop = SummateLoop + i
}

print(SummateLoop)
```

```
## [1] 5050
```

**Problem 5**

Write a function to find the sum of all integers between any two values.

```
SumAnyInterval <- function(IntStart,IntEnd){
  n <- 0
  for (i in IntStart:IntEnd){
    n = n+i
  }
  return(n)
}
```

Call the function:

```
SumAnyInterval(5,200)
```

```
## [1] 20090
```

**Problem 6**

List all of the odd numbers from 1 to 100.

Through the seq function:

```
seq(1,100,by=2)
```

```
##  [1]  1  3  5  7  9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
## [26] 51 53 55 57 59 61 63 65 67 69 71 73 75 77 79 81 83 85 87 89 91 93 95 97 99
```

Through the use of mod and for loop

```
Odds = c()
for (i in 1:100) {
  if (i%%2) {
    Odds = c(Odds,i)
  }
}
print(Odds)
```

```
##  [1]  1  3  5  7  9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
## [26] 51 53 55 57 59 61 63 65 67 69 71 73 75 77 79 81 83 85 87 89 91 93 95 97 99
```

**Problem 7**

List all of the prime numbers from 1 to 1000.

```
Primes <- c(2)
for (i in seq(3,1000,by=1)){
  test <- i%%Primes
  if (0%in%test){}else{
    Primes <- c(Primes,i)
  }
}
print(Primes)
```

```
##   [1]   2   3   5   7  11  13  17  19  23  29  31  37  41  43  47  53  59  61
##  [19]  67  71  73  79  83  89  97 101 103 107 109 113 127 131 137 139 149 151
##  [37] 157 163 167 173 179 181 191 193 197 199 211 223 227 229 233 239 241 251
##  [55] 257 263 269 271 277 281 283 293 307 311 313 317 331 337 347 349 353 359
##  [73] 367 373 379 383 389 397 401 409 419 421 431 433 439 443 449 457 461 463
##  [91] 467 479 487 491 499 503 509 521 523 541 547 557 563 569 571 577 587 593
## [109] 599 601 607 613 617 619 631 641 643 647 653 659 661 673 677 683 691 701
## [127] 709 719 727 733 739 743 751 757 761 769 773 787 797 809 811 821 823 827
## [145] 829 839 853 857 859 863 877 881 883 887 907 911 919 929 937 941 947 953
## [163] 967 971 977 983 991 997
```

**Problem 8**

Generate 100 random numbers

```
runif(100)
```

```
##   [1] 0.570292515 0.717720031 0.497294861 0.935893837 0.250305377 0.512273038
##   [7] 0.786536646 0.259388552 0.298290362 0.568724837 0.100926543 0.202022632
##  [13] 0.606435544 0.110513885 0.346049161 0.662016249 0.951981557 0.029790296
##  [19] 0.661551528 0.682697909 0.636323901 0.676700539 0.705626856 0.005487765
##  [25] 0.932037224 0.153616306 0.901631762 0.050050779 0.514894658 0.129638350
##  [31] 0.942543373 0.227458170 0.312526791 0.881212118 0.865482855 0.252787050
##  [37] 0.594979968 0.312965965 0.729261518 0.594496123 0.140295335 0.047460064
##  [43] 0.250029218 0.578618705 0.069445519 0.013843450 0.335222026 0.291545094
##  [49] 0.036808332 0.895339380 0.058347163 0.994925191 0.041287297 0.567784655
##  [55] 0.324329273 0.490912999 0.396170197 0.822635567 0.401043116 0.227246478
##  [61] 0.291539039 0.252226281 0.529919320 0.774624525 0.291089231 0.889356188
##  [67] 0.821883474 0.020494811 0.145012773 0.027994898 0.191491048 0.194473787
##  [73] 0.262607111 0.201539285 0.674569150 0.123474427 0.056606415 0.652323854
##  [79] 0.798489466 0.787579105 0.472648302 0.451557581 0.527609397 0.313721141
##  [85] 0.892440590 0.107838234 0.679286016 0.612756801 0.780633525 0.669396641
##  [91] 0.461357322 0.516054146 0.142529747 0.529862024 0.760236494 0.795626782
##  [97] 0.929788017 0.680171496 0.977270794 0.174657755
```

**Problem 9**

Generate 100 random numbers within a specific range

```
runif(100,2,5)
```

```
##   [1] 2.429392 4.609112 4.506052 2.148219 3.825453 3.718369 2.819764 2.504533
##   [9] 4.082852 4.540669 3.373546 3.212316 4.825419 4.820710 2.891696 4.556973
##  [17] 2.487125 4.849869 4.775912 3.462972 4.053844 3.909301 3.447561 3.043543
##  [25] 2.441888 4.267473 4.373124 3.479563 2.466089 4.353735 2.716649 3.578171
##  [33] 2.546044 2.674138 4.196625 4.965297 4.424432 3.409932 4.383091 2.571378
##  [41] 3.479214 4.904592 3.543828 2.218241 4.931139 2.308317 4.360783 3.841666
##  [49] 4.264416 3.896597 3.954894 4.865204 2.395601 4.182400 3.835234 3.144666
##  [57] 4.411236 4.813918 4.847552 4.356585 2.575804 3.766502 2.574003 2.226523
##  [65] 4.674526 4.408918 4.596921 4.422150 4.894650 2.976679 3.638125 2.568875
##  [73] 4.695544 3.291422 2.473690 3.445536 4.287661 3.095167 2.413502 2.578097
##  [81] 3.318465 3.759376 2.567481 2.464305 3.252983 3.874364 3.864701 3.806749
##  [89] 2.338129 4.640823 3.030392 3.349801 3.416738 4.461460 3.129433 3.245686
##  [97] 2.515249 4.544124 2.731149 2.598898
```

### Problem 10

Write your own functions to give descriptive statistics for a vector variable storing multiple numbers. Write functions for the following without using R intrinsics: mean, mode, median, range, standard deviation

- It's ok to use sum() and length()
- be creative and see if you can find multiple solutions.

```r
DescriptiveStats <- function(array){
  array2 = sort(array)
  len = length(array2)
  mean <- sum(array2)/len
  if (length(array2)%%2){
    median = array2[(len/2)+.5]
  }else{
    median = (array2[len/2] + array2[(len/2+1)])/2
  }
  current <- 0
  for (i in array2) {
    new <- length(which(array2==i))
    if (new>current) {
      current <- new
      mode <- array2[i]
    }
  }
  std <- sqrt(sum((array2 - mean)^2)/(len-1))

  return(c(mean,median,mode,std))

}

DescriptiveStats(c(1,2,3,4,5))
```

```
## [1] 3.000000 3.000000 1.000000 1.581139
```

### Problem 11

Count the number of characters in a string variable

```r
String <- "Count the number of characters in this string"

CharSplit<- strsplit(String,"")
Count <- length(CharSplit[[1]])

print(Count)
```

```
## [1] 45
```

### Problem 12

Count the number of words in a string variable

```r
String = "Count the number of words in this string"
WordSplit <- strsplit(String," ")
```

```
Count <- length(WordSplit[[1]])
print(Count)
```

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

### Problem 13

Count the number of sentences in a string variable.

```
String = "Count the sentences in this String. Use the strsplit function."

SentenceSplit = strsplit(String,".",fixed=TRUE)
Count <- length(SentenceSplit[[1]])
print(Count)
```

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

Had to change the fixed option to TRUE. Most likely because of regex.

### Problem 14

Count the number of times a specific character occurs in a string variable

```
CountAccurance <- function(char,string){
  split <- strsplit(string,"")
  split <- unlist(split)
  a <- data.frame(table(split))
  index <- which(a$split== char)
  return(a$Freq[index])
}

CountAccurance("c",String)
```

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

### Problem 15

Do a logical test to see if one word is found within the text of another string variable.

```
TestWord <- "hello"
TestSentence <- "Is hello contained in this sentence?"

grepl(TestWord,TestSentence)
```

```
## [1] TRUE
```

### Problem 16

Put the current computer time in milliseconds into a variable

```
print(as.numeric(Sys.time())*1000, digits = 10)
```

```
## [1] 1667557498712
```

### Problem 17

Measure how long a piece of code takes to run by measuring the time before the code is run, and after the code is run, and taking the difference to find the total time

```
t1 = as.numeric(Sys.time())*1000
print(DescriptiveStats(c(Primes)))
```

```
## [1] 453.1369 436.0000   3.0000 298.1924
```

```
t2 = as.numeric(Sys.time())*1000
print(t2-t1)
```

```
## [1] 8.689941
```

### Problem 18

Read a .txt file or .csv file into a variable

```
# data = read.csv("track_points.csv")
```

### Problem 19

Output the contents of a variable to a .txt file

```
# write.csv(data, file = "test.txt")
```

### Problem 20

Create a variable that stores a 20x20 matrix of random numbers

```
m <- matrix(runif(25), nrow=5)
print(m)
```

```
##           [,1]      [,2]      [,3]        [,4]       [,5]
## [1,] 0.3479302 0.2468107 0.7191295 0.108848188 0.33854835
## [2,] 0.4894181 0.6003259 0.4835545 0.669990584 0.42811305
## [3,] 0.7032132 0.6122268 0.7956494 0.599236371 0.06594126
## [4,] 0.1481174 0.9731860 0.6691159 0.493684301 0.22276346
## [5,] 0.9526958 0.3034572 0.6549028 0.002319674 0.85542782
```

### Problem 21

Output any matrix to a txt file using commas or tabs to separate column values, and new lines to separate row values

```
# write.csv(m,"RandomMatrix.txt")
```

## Harder Problems

### Problem 22

Fizzbuzz

```
testvalues <- c(3,5)
outputstring <- c("Fizz","Buzz")

for (i in 1:100) {
output = ""

test <- !i%%testvalues

if (TRUE%in%test) {
```

```
output <- paste(outputstring[test],collapse = "")
}else {
  output <- as.character(i)
}

print(output)
}
```

```
## [1] "1"
## [1] "2"
## [1] "Fizz"
## [1] "4"
## [1] "Buzz"
## [1] "Fizz"
## [1] "7"
## [1] "8"
## [1] "Fizz"
## [1] "Buzz"
## [1] "11"
## [1] "Fizz"
## [1] "13"
## [1] "14"
## [1] "FizzBuzz"
## [1] "16"
## [1] "17"
## [1] "Fizz"
## [1] "19"
## [1] "Buzz"
## [1] "Fizz"
## [1] "22"
## [1] "23"
## [1] "Fizz"
## [1] "Buzz"
## [1] "26"
## [1] "Fizz"
## [1] "28"
## [1] "29"
## [1] "FizzBuzz"
## [1] "31"
## [1] "32"
## [1] "Fizz"
## [1] "34"
## [1] "Buzz"
## [1] "Fizz"
## [1] "37"
## [1] "38"
## [1] "Fizz"
## [1] "Buzz"
## [1] "41"
## [1] "Fizz"
## [1] "43"
## [1] "44"
## [1] "FizzBuzz"
## [1] "46"
## [1] "47"
## [1] "Fizz"
## [1] "49"
## [1] "Buzz"
## [1] "Fizz"
## [1] "52"
## [1] "53"
## [1] "Fizz"
## [1] "Buzz"
## [1] "56"
## [1] "Fizz"
## [1] "58"
## [1] "59"
## [1] "FizzBuzz"
## [1] "61"
## [1] "62"
## [1] "Fizz"
## [1] "64"
## [1] "Buzz"
## [1] "Fizz"
## [1] "67"
## [1] "68"
## [1] "Fizz"
## [1] "Buzz"
## [1] "71"
## [1] "Fizz"
## [1] "73"
## [1] "74"
## [1] "FizzBuzz"
## [1] "76"
## [1] "77"
## [1] "Fizz"
## [1] "79"
## [1] "Buzz"
## [1] "Fizz"
## [1] "82"
## [1] "83"
## [1] "Fizz"
## [1] "Buzz"
## [1] "86"
## [1] "Fizz"
## [1] "88"
## [1] "89"
## [1] "FizzBuzz"
## [1] "91"
## [1] "92"
## [1] "Fizz"
## [1] "94"
## [1] "Buzz"
## [1] "Fizz"
## [1] "97"
## [1] "98"
## [1] "Fizz"
## [1] "Buzz"
```

other solution to fizzbuzz:

```
fizzbuzz <- 1:100

mod3 <- !fizzbuzz%%3
mod5 <- !fizzbuzz%%5
mod35 <- mod3 & mod5
fizzbuzz[mod3] <- "Fizz"
fizzbuzz[mod5] <- "Buzz"
fizzbuzz[mod35] <- "FizzBuzz"

print(fizzbuzz)
```

```
##   [1] "1"        "2"        "Fizz"     "4"        "Buzz"     "Fizz"
##   [7] "7"        "8"        "Fizz"     "Buzz"     "11"       "Fizz"
##  [13] "13"       "14"       "FizzBuzz" "16"       "17"       "Fizz"
##  [19] "19"       "Buzz"     "Fizz"     "22"       "23"       "Fizz"
##  [25] "Buzz"     "26"       "Fizz"     "28"       "29"       "FizzBuzz"
##  [31] "31"       "32"       "Fizz"     "34"       "Buzz"     "Fizz"
##  [37] "37"       "38"       "Fizz"     "Buzz"     "41"       "Fizz"
##  [43] "43"       "44"       "FizzBuzz" "46"       "47"       "Fizz"
##  [49] "49"       "Buzz"     "Fizz"     "52"       "53"       "Fizz"
##  [55] "Buzz"     "56"       "Fizz"     "58"       "59"       "FizzBuzz"
##  [61] "61"       "62"       "Fizz"     "64"       "Buzz"     "Fizz"
##  [67] "67"       "68"       "Fizz"     "Buzz"     "71"       "Fizz"
##  [73] "73"       "74"       "FizzBuzz" "76"       "77"       "Fizz"
##  [79] "79"       "Buzz"     "Fizz"     "82"       "83"       "Fizz"
##  [85] "Buzz"     "86"       "Fizz"     "88"       "89"       "FizzBuzz"
##  [91] "91"       "92"       "Fizz"     "94"       "Buzz"     "Fizz"
##  [97] "97"       "98"       "Fizz"     "Buzz"
```

Less code, but same test three times and harder to adapt new values or words.