Problem Set 3: Fundamental Objects in R

[1 pt] *You may use any functions or operators that are associated with atomic vectors and lists; you may not use any downloaded packages. Solutions that require a discussion or an explanation should be type-written in a 12-point font and submitted in class—do not include any R code. See the instructions in Problem Set 2 regarding the submission of R code.*

The following functions, which have not been discussed in class, may or may not be useful for this problem set:

`%in%`, `cut`, `ifelse`, `match`, `replicate`, `split`

1. **[8 pts]** Use vectorized operations to create the following vectors. Assign the symbol `v1` to the first vector, `v2` to the second vector, and so on.

   `v1:` `1,-2,3,-4,5,-6,...,99,-100`

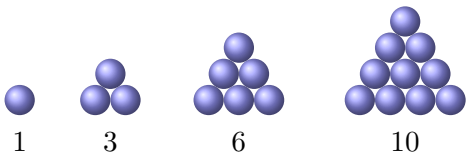
   `v2:` `1,2,2,3,3,3,4,5,5,6,6,6,...,97,98,98,99,99,99`

   `v3:` $\cos(x)$ when x is divisible by 3 and $1/\sqrt{x}$ otherwise, where $x = 1, 2, \ldots, 100$.

   `v4:` `A` appears at every index where the value of `v1` is positive; `B` appears at every index where the value of `v1` is negative and the value of `v3` is between 0 and 0.138; and `C` appears at all other indices.

   Do not write functions for this problem, just expressions.

2. A triangular number is a number that can be represented in the form of a triangular grid of identical objects. Specifically, the nth triangular number is the number of objects composing a triangle with n objects on each side:



   (a) **[3 pts]** Create a vector named `tri.num.20` that contains the first 20 triangular numbers using the `for` loop.

   (b) **[3 pts]** Repeat (a) using vectorized operations.

   (c) **[4 pts]** Use `system.time` to compare the time it takes to create a vector named `tri.num` that contains the first $100,000$ triangular numbers using (i) the `for` loop, and (ii) vectorized operations. Comment on the difference in performance, if any.

   (d) **[6 pts]** Use vectorized operations to create a vector named `tri.num.50` that contains the first 50 triangular numbers, and name the elements of the vector with the letters of the alphabet, using repeated letters when necessary:

```
> names(tri.num.50)
 [1] "a"  "b"  "c"  "d"  "e"  "f"  "g"  "h"  "i"  "j"  "k"  "l"  "m"
[14] "n"  "o"  "p"  "q"  "r"  "s"  "t"  "u"  "v"  "w"  "x"  "y"  "z"
[27] "aa" "bb" "cc" "dd" "ee" "ff" "gg" "hh" "ii" "jj" "kk" "ll" "mm"
[40] "nn" "oo" "pp" "qq" "rr" "ss" "tt" "uu" "vv" "ww" "xx"
```

(e) **[4 pts]** Use vectorized operations to select those numbers in `tri.num.50` whose names contain a vowel.

3. **[10 pts]** A perfect square is a number that is the square of an integer: $1, 4, 9, 16, 25, \ldots$. Write a function named `squareNumbers` that accepts as an argument an integer n, where n is a multiple of 10, and returns a list such that:

   - the first component contains the perfect squares in $[1, 10]$, the second component contains the perfect squares in $[11, 20]$, and so on, up to a final component that contains the perfect squares in $[n - 9, n]$;
   - the components are named "`1 to 10`", "`11 to 20`", and so on.

   Furthermore, the function prints the intervals that contain no perfect squares. Some examples of calling the function `squareNumbers`:

```
> squareNumbers(10)
> result <- squareNumbers(20)
> result
$`1 to 10`
[1] 1 4 9

$`11 to 20`
[1] 16

> squareNumbers(60)
The following interval does not contain perfect squares:
51 to 60
> squareNumbers(100)
The following intervals do not contain perfect squares:
51 to 60
71 to 80
```