General Notes

- You will submit a minimum of three files, the core files must conform to the following naming conventions (including capitalization and underscores). 123456789 is a placeholder, please replace these nine digits with your nine-digit Bruin ID. The files you must submit are:

  1. *123456789_stats102a_hw1.R*: An R script file containing all of the **functions** and **constructors** you wrote for the homework. The first line of your .Rmd file, after loading libraries, should be sourcing this script file.
  2. *123456789_stats102a_hw1.Rmd*: Your markdown file which generates the output file of your submission.
  3. *123456789_stats102a_hw1.html/pdf*: Your output file, either a PDF or an HTML file depending on the output you choose to generate.
  4. *included image files*: You may name these what you choose, but you must include all of the image files you generated for your structured flowcharts, otherwise your file will not knit.

  If you fail to submit any of the required core files you will receive ZERO points for the assignment. If you submit any files which do not conform to the specified naming convention, you will receive (at most) half credit for the assignment.

- Your .Rmd file must knit. If your .Rmd file does not knit you will receive (at most) half credit for the assignment.
  The two most common reason files fail to knit are because of workspace/directory structure issues and because of missing include files. To remedy the first, ensure all of the file paths in your document are relative paths pointing at the current working directory. To remedy the second, simply make sure you upload any and all files you source or include in your .Rmd file.

- Your coding should adhere to the tidyverse style guide: https://style.tidyverse.org/.

- All pseudocode or flowcharts should be done on separate sheets of paper, but be included, inline as images, in your final markdown document.

- Any functions/classes you write should have the corresponding comments as the following format.

```
my_function <- function(x, y, ...){
#A short description of the function
#Args:
#x: Variable type and dimension
#y: Variable type and dimension
#Return:
#Variable type and dimension
Your codes begin here
}
```

NOTE: *Everything* you need to do this assignment is here, in your class notes, or was covered in discussion or lecture.

- **DO NOT** look for solutions online.
- **DO NOT** collaborate with anyone inside (or outside) of this class.
- Work **INDEPENDENTLY** on this assignment.
- **EVERYTHING** you submit **MUST** be 100% your, original, work product. Any student suspected of plagiarizing, in whole or in part, any portion of this assignment, will be **immediately** referred to the Dean of Student's office without warning.

1: Dealing with Large Numbers

To calculate with large floating point numbers we define objects called (p, q) number with a list as its base type. The list should have four components. The first one is either the integer +1 or the integer −1. It gives the sign of the number. The second and third are p and q. And the fourth component is a vector of p + q + 1 integers between zero and nine. For example,

```
x <- structure(list(sign = 1, p = 3, q = 4, nums = 1:8), class = "pqnumber")
```

(a) Write a constructor function, an appropriate predicate function, appropriate coercion functions, and a useful print() method.

- The constructor takes the four arguments: **sign**, **p**, **q**, and **nums**. Then check if the arguments satisfy all requirements for a (p, q) number. If not, stop with an error message. If yes, return a (p, q) number object.
- A predicate is a function that returns a single **TRUE** or **FALSE**, like is.data.frame(), or is.factor(). Your predicate function should be is_pqnumber() and should behave as expected.
- A useful print() method should allow users to print a (p, q) number object with the decimal representation defined as follows:

$$x = \sum_{s=-p}^{q} x_s \times 10^s$$

Thus p = 3 and q = 4 with nums = 1:8 has the decimal value

$$0.001 + 0.002 + 0.3 + 4 + 50 + 600 + 7000 + 80000 = 87654.321$$

- Please create three (p, q) number objects for the following numbers to demonstrateis_pqnumber(), print():
  (a) sign = 1, p = 3, q = 4, nums = 1:8
  (b) sign = 1, p = 6, q = 0, nums = c(3,9,5,1,4,1,3)
  (c) sign = -1, p = 5, q = 1, nums = c(2,8,2,8,1,7,2)

- A coercion function forces an object to belong to a class, such as as.factor() or as_tibble(). You will create a generic coercion function **as_pqnumber()** which will accept a numeric or integer argument x and return the appropriate (p, q) number object. For example, given the decimal number 3.14 with p = 3 and q = 4, the function will return a (p, q) number with num = c(0, 4, 1, 3, 0, 0, 0, 0). You should also create a generic as_numeric() function and a method to handle a (p, q) number. You may use the provided example to showcase your functions.

(b) Addition and Subtraction

Write an addition function and a subtraction function. Suppose we have two positive (p, q) number objects x and y. Write a function to calculate the sum of x and y. Clearly its decimal representation is

$$x + y = \sum_{s=-p}^{q} (x_s + y_s) \times 10^s$$

but this cannot be used directly because we can have $x_s + y_s > 9$.

So we need a *carry-over* algorithm that moves the extra digits in the appropriate way. Same as one would do when adding two large numbers on paper. Also we need a special provision for overflow, because the sum of two (p, q) number objects can be too big to be a (p, q) number.

A subtraction algorithm should have a *carry-over* algorithm that *borrows* 10 in the same way as you would do a subtraction with pencil-and-paper.

Your functions should work for both positive and negative (p, q) numbers.

(c) Multiplication

Write a multiplication function which can multiply two pqnumber objects. Think about how you would multiply two large numbers by hand and implement that algorithm in R for two pqnumber objects.

You may demonstrate your three functions using the examples provided in (a).

Note: Please attach the flowcharts or pseudo code for your addition function, subtraction function, and multiplication function. Also, the cases provided here are only for your test. We will use different arguments and objects to try your functions while grading. Therefore, try your best to make your functions robust.

2: Root Finding

Use Newton's method to compute the square root by using only simple arithmetic operations.

(a) Please write a function that computes the square root of an arbitrary positive number. The function should take four arguments, the last three have default values:

1. a a non-negative number to compute the square root of.
2. tol determines when you consider two iterates to be equal.
3. iter_max gives the maximum number of iterations.
4. verbose determines if you want to display intermediate results.

(b) Give an example of your function by computing the square root of 5.

(c) How would the program change to compute an arbitrary, positive integer, root? Please use calculus to determine an appropriate iterative equation for approximating this arbitrary root.

(d) Write an R function like the one for square root which calculates the arbitrary root of a number.

(e) Give an example of your function by computing the fifth root of 7.

(f) Let $e_k = |x_k - \sqrt[5]{7}|$, where k indicates the k_{th} iteration. Please print from e_1 to e_4, and specify your findings.