General Notes

- You will need to submit two files: a zip file and an output file (either HTML or PDF). The zip file must contain at least two files: an R script and an RMD file. All files must follow the specified naming conventions, including capitalization and underscores. Replace 123456789 with your nine-digit Bruin ID in all filenames. The files to be submitted are:

  1. *123456789_stats102a_hw1.html/pdf*: An output file, either a PDF or an HTML file depending on the output you choose to generate.
  2. *123456789_stats102a_hw1.zip*: Place all your R script, RMD, and image files into a single folder named 123456789_stats102a_hw1, then compress the folder into 123456789_stats102a_hw1.zip.
     - *123456789_stats102a_hw1.R*: An R script file containing all the functions you wrote for the homework. In your RMD file, the first line after loading libraries should source this script file.
     - *123456789_stats102a_hw1.Rmd*: A markdown file generates the output file of your submission.
     - *included image files*: You may name these files as you choose, but you must include all image files you generated for your RMD file; otherwise, it will not knit.

  If you fail to submit any of the required 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 half credit for the assignment.

- **Your RMD file must be knittable. If your RMD file does not knit in graders' computer, you will receive at most half credit for the assignment.**
  The two most common reasons RMD files fail to knit are workspace/directory structure issues and missing files. To address the first, ensure all file paths in your document are relative paths pointing to the current working directory. To resolve the second, upload all files you source or include in your RMD file to the same folder.

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

- All 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 code 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: Greatest Common Divisor (GCD)

Please write a function, gcd(x, y), which takes two integers x, y and calculates their greatest common divisor (GCD) using the Euclidean algorithm. For more details, you may refer to this Wikipedia page.

The Euclidean Algorithm Example:
Let a = 180 and b = 25

1. calculate 180/25, and get the result 7 with remainder 5, so $180 = 7 \times 25 + 5$.
2. calculate 25/5, and get the result 5 with remainder 0, so $25 = 5 \times 5 + 0$.
3. the greatest common divisor of 180 and 25 is 5.

Make use of gcd() to write a function, gcdi(v), which takes an integer vector v and return its greatest common divisor. The length of the input vector will be at least two but no more than 1000.

2: Least Common Multiple (LCM) and Sum of Fractions

- Write a function, lcm(x, y), which takes two integers x, y and calculates their least common multiple. Please check this Wikipedia page for more details.
- Write a function, add_2_frac(n1, d1, n2, d2), that takes four integers as inputs and use lcm() to calculate the sum of two fractions. The function parameters represent:

  n1 is the numerator of the first fraction,

  d1 is the denominator of the first fraction,

  n2 is the numerator of the second fraction, and

  d2 is the denominator of the second fraction.

  The function will return a generic vector that stores the numerator and denominator of the fraction's sum. For example, when executed as mysum ← add_2_frac(1, 2, 1, 3), mysum$num will be 5 and mysum$denom will be 6.

3: Prime Factorization

Please write a function get_factors() that takes a number x and returns a list object. The list object should contain a vector of unique prime factors of x and the corresponding exponents. Additionally, write one "helper function" is_prime() that returns a logical vector depending on whether or not the elements in x are prime. A good way to test this function should be:

```
x <- sample(x = 1e4, size = 1)
y <- get_factors(x)
this_works <- prod(y$primes^y$exponents) == x & all(is_prime(y$primes))
```

It is a necessary, but not sufficient, condition that this_works == TRUE for your function to work as intended. (It will largely depend on if your is_prime() function is correct.)

Homework Requirements:

a. Write algorithms (or flowchart) for gcdi(), add_2_frac(), and get_factors(). Please make sure your algorithms(or flowchart) sufficiently complete, clear, and concise enough to enable a person to accurately implement the algorithm in any programming language they are adept with using.

b. Write the functions which accurately implements the algorithms as described or requested.

c. **Each function should respond in a reasonable time.**

d. **Include the error-handling to ensure your functions work properly.**

e. Showcase the functions. You may use suggested test cases or make up some examples but no more than 5 cases per function.

f. **Keep all of the functions in a R script and make sure no R functions are written in the RMD file.**

g. The RMD file includes only algorithms (or flowchart), and examples. All the R functions should be saved in the R script.

Note:

The formatting takes 40% of your homework grade. The efficiency, accuracy, robustness, and error handling of your function account for the rest of 60%.

Sample test cases:
Here are some suggested test cases but not limited to while grading.

1. GCD
   - gcdi(c(16, 32, -58))→ 2

- gcdi(c(4789, 6123, 199))→ 1

2. Sum of Fractions
   - add_2_frac(1, 11, 1, 5)→ list(num = 16, denom = 55)
   - add_2_frac(1, 11, -1, 5)→ list(num = -6, denom = 55)
3. Prime Factorization
   - is_prime(c(9, 7))→ [False True]
   - get_factors(1920)→ list(primes = [2 3 5], exponents = [7 1 1])
   - get_factors(1.92)→ appropriate error handling