Q1.

Code:

require 'date'

# Generate a year's worth of random temperature data

def generate\_temperature\_data

  (1..365).map { rand(15..35) }

end

# Calculate the average temperature for the year

def average\_temperature(temps)

  temps.sum / temps.size.to\_f

end

# Find the hottest and coldest days of the year

def temperature\_extremes(temps)

  hottest\_day = temps.index(temps.max) + 1

  coldest\_day = temps.index(temps.min) + 1

  [temps.max, hottest\_day, temps.min, coldest\_day]

end

# Calculate the average temperature for each month

def monthly\_average(temps)

  days\_in\_month = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]

  month\_starts = days\_in\_month.each\_with\_index.map { |days, i| days\_in\_month[0...i].sum }

  averages = []

  month\_starts.each\_with\_index do |start\_day, i|

    month\_temps = temps[start\_day, days\_in\_month[i]]

    averages << month\_temps.sum / month\_temps.size.to\_f

  end

  averages

end

# Find the length of the longest heat wave

def longest\_heat\_wave(temps)

  max\_length = 0

  current\_length = 0

  temps.each do |temp|

    if temp > 30

      current\_length += 1

      max\_length = [max\_length, current\_length].max

    else

      current\_length = 0

    end

  end

  max\_length

end

# Find the length of the longest cold spell

def longest\_cold\_spell(temps)

  max\_length = 0

  current\_length = 0

  temps.each do |temp|

    if temp < 20

      current\_length += 1

      max\_length = [max\_length, current\_length].max

    else

      current\_length = 0

    end

  end

  max\_length

end

# Find the month with the highest average temperature

def hottest\_month(temps)

  month\_avgs = monthly\_average(temps)

  hottest\_month\_index = month\_avgs.index(month\_avgs.max)

  hottest\_month\_index + 1 # months are 1-indexed

end

# Main Execution

temps = generate\_temperature\_data

puts "Average Temperature for the Year: #{average\_temperature(temps).round(2)}°C"

hottest\_temp, hottest\_day, coldest\_temp, coldest\_day = temperature\_extremes(temps)

puts "Hottest Temperature: #{hottest\_temp}°C on Day #{hottest\_day}"

puts "Coldest Temperature: #{coldest\_temp}°C on Day #{coldest\_day}"

monthly\_avgs = monthly\_average(temps)

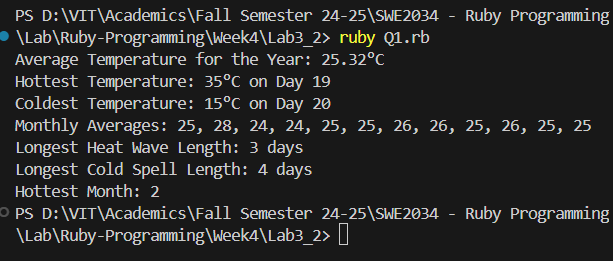
puts "Monthly Averages: #{monthly\_avgs.map { |avg| avg.round() }.join(', ')}"

puts "Longest Heat Wave Length: #{longest\_heat\_wave(temps)} days"

puts "Longest Cold Spell Length: #{longest\_cold\_spell(temps)} days"

puts "Hottest Month: #{hottest\_month(temps)}"

Output:



Q2.

Code:  
def find\_head\_number(array)

    return nil if array.length < 3

    (1...array.length - 1).each do |i|

      if array[i] > array[i - 1] && array[i] > array[i + 1]

        return array[i]

      end

    end

    nil

  end

  def find\_master\_pair(array)

    return nil if array.length < 2

    max\_sum = 0

    best\_pair = nil

    (0...(array.length - 1)).each do |i|

      ((i + 1)...array.length).each do |j|

        sum = array[i] + array[j]

        if sum > max\_sum

          max\_sum = sum

          best\_pair = [array[i], array[j]]

        end

      end

    end

    best\_pair

  end

  array = [1, 3, 2, 8, 5, 4, 10, 12, 53, 23, 25]

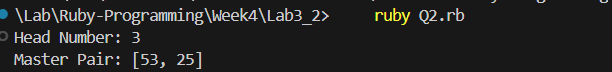
  head\_number = find\_head\_number(array)

  master\_pair = find\_master\_pair(array)

  puts "Head Number: #{head\_number.inspect}"

  puts "Master Pair: #{master\_pair.inspect}"

Output:



Q3.

Code:

class FactorialDispatcher

    def method\_missing(method\_name, \*args)

      if method\_name == :factorial

        handle\_factorial(\*args)

      else

        super

      end

    end

    def respond\_to\_missing?(method\_name, include\_private = false)

      method\_name == :factorial || super

    end

    private

    def handle\_factorial(n)

      if n.is\_a?(Integer) && n >= 0

        result = factorial(n)

        puts "Result of factorial(#{n}): #{result}"

      else

        puts "Error: Factorial is only defined for non-negative integers."

      end

    end

    def factorial(n)

      (1..n).inject(:\*) || 1

    end

  end

  dispatcher = FactorialDispatcher.new

  dispatcher.factorial(5)

  dispatcher.factorial(10)

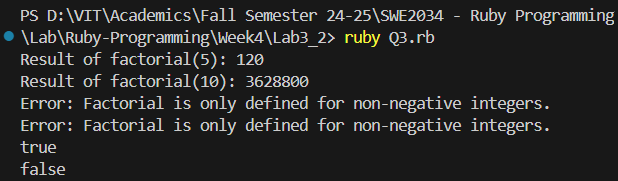
  dispatcher.factorial(-1)

  dispatcher.factorial('a')

  puts dispatcher.respond\_to?(:factorial)

  puts dispatcher.respond\_to?(:non\_existent)

Output



Q4.

Code:

def evaluate\_expression(expression)

    begin

      eval(expression)

    rescue StandardError => e

      return nil

    end

  end

  def balanced\_parentheses?(str)

    stack = []

    brackets = { '(' => ')', '{' => '}', '[' => ']', '<' => '>' }

    positions = []

    expression = ""

    str.each\_char.with\_index do |char, index|

      if brackets.keys.include?(char)

        stack.push(char)

        positions.push(index)

      elsif brackets.values.include?(char)

        last\_open = stack.pop

        if last\_open.nil? || brackets[last\_open] != char

          return "Mismatch at position #{index + 1}"

        end

        positions.pop

      else

        expression << char unless char.match?(/[()\[\]{}<>]/)

      end

    end

    if stack.empty?

      result = evaluate\_expression(expression.strip)

      return result.nil? ? true : result

    else

      return "Mismatch at position #{positions.last + 1}"

    end

  end

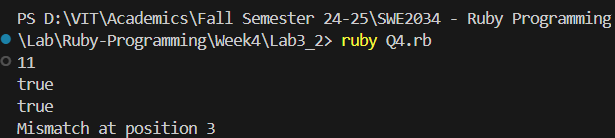
  puts balanced\_parentheses?("(1+2)\*{3+4}")

  puts balanced\_parentheses?("a ( b c ) d")

  puts balanced\_parentheses?("{[<>]}")

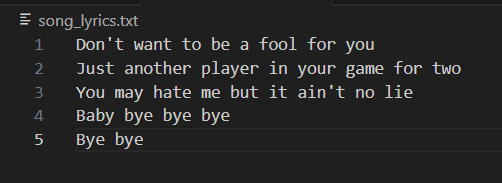
  puts balanced\_parentheses?("{[>]}")

Output:



Q5.

TextFile:



Code:

def word\_frequency(file\_path)

    word\_count = Hash.new(0)

    text = File.read(file\_path)

    words = text.downcase.scan(/\b[\w']+\b/)

    words.each { |word| word\_count[word] += 1 }

    word\_count.each { |word, count| puts "#{word}: #{count}" }

    most\_frequent\_word = word\_count.max\_by { |\_, count| count }

    puts "\nMost frequently used word: '#{most\_frequent\_word[0]}' appears #{most\_frequent\_word[1]} times"

  end

  word\_frequency('song\_lyrics.txt')

Output:

