Q1.

Code:

```ruby
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
```

```ruby
      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)
```

```ruby
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:

```
PS D:\VIT\Academics\Fall Semester 24-25\SWE2034 - Ruby Programming
\Lab\Ruby-Programming\Week4\Lab3_2> ruby Q1.rb
Average Temperature for the Year: 25.32°C
Hottest Temperature: 35°C on Day 19
Coldest Temperature: 15°C on Day 20
Monthly Averages: 25, 28, 24, 24, 25, 25, 26, 26, 25, 26, 25, 25
Longest Heat Wave Length: 3 days
Longest Cold Spell Length: 4 days
Hottest Month: 2
PS D:\VIT\Academics\Fall Semester 24-25\SWE2034 - Ruby Programming
\Lab\Ruby-Programming\Week4\Lab3_2>
```

Q2.

Code:

```ruby
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
```

```ruby
  (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:

```
\Lab\Ruby-Programming\Week4\Lab3_2>     ruby Q2.rb
Head Number: 3
Master Pair: [53, 25]
```

Q3.

Code:

```ruby
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

```
PS D:\VIT\Academics\Fall Semester 24-25\SWE2034 - Ruby Programming
\Lab\Ruby-Programming\Week4\Lab3_2> ruby Q3.rb
Result of factorial(5): 120
Result of factorial(10): 3628800
Error: Factorial is only defined for non-negative integers.
Error: Factorial is only defined for non-negative integers.
true
false
```

Q4.

Code:

```ruby
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
```

```ruby
  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:

```
PS D:\VIT\Academics\Fall Semester 24-25\SWE2034 - Ruby Programming
\Lab\Ruby-Programming\Week4\Lab3_2> ruby Q4.rb
11
true
true
Mismatch at position 3
```

Q5.

TextFile:

song_lyrics.txt

```
1   Don't want to be a fool for you
2   Just another player in your game for two
3   You may hate me but it ain't no lie
4   Baby bye bye bye
5   Bye bye
```

Code:

```ruby
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:

```
PS D:\VIT\Academics\Fall Semester 24-25\SWE2034 - Ruby Programming
\Lab\Ruby-Programming\Week4\Lab3_2> ruby Q5.rb
don't: 1
want: 1
to: 1
be: 1
a: 1
fool: 1
for: 2
you: 2
just: 1
another: 1
player: 1
in: 1
your: 1
game: 1
two: 1
may: 1
hate: 1
me: 1
but: 1
it: 1
ain't: 1
no: 1
lie: 1
baby: 1
bye: 5

Most frequently used word: 'bye' appears 5 times
```