**RUBY LAB -3**

**Name: Vishwanth P**

**Register No: 21MIS1117**

**Assessment 3**

1. **Write a ruby code using the following keywords yield, lambda and procs.**

def greet

  puts "Hi!"

  yield if block\_given?

  puts "Bye!"

end

greet { puts "Hello, Vishwanth!" }

say\_hello = ->(name) { puts "Hi, #{name}!" }

say\_hello.call("Vishwanth")

multiply = ->(a, b) { a \* b }

puts multiply.call(2, 3)

def example(proc\_obj, lambda\_obj)

  puts proc\_obj.call(1, 2)

  lambda\_obj.call("Vishwanth")

  yield if block\_given?

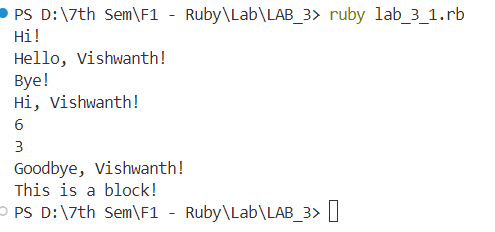
end

my\_proc = ->(x, y) { x + y }

my\_lambda = ->(name) { puts "Goodbye, #{name}!" }

example(my\_proc, my\_lambda) { puts "This is a block!" }

**Output**

****

1. **Write a ruby programming using Modules concept.**

module Vehicle

  def speed

    raise "Speed method not implemented"

  end

  def fuel

    raise "Fuel method not implemented"

  end

  def display\_info

    puts "Speed: #{speed} km/h"

    puts "Fuel: #{fuel} liters"

  end

end

class Car

  include Vehicle

  def initialize(speed, fuel)

    @speed = speed

    @fuel = fuel

  end

  def speed

    @speed

  end

  def fuel

    @fuel

  end

end

class Bike

  include Vehicle

  def initialize(speed, fuel)

    @speed = speed

    @fuel = fuel

  end

  def speed

    @speed

  end

  def fuel

    @fuel

  end

end

car = Car.new(120, 50)

puts "Car:"

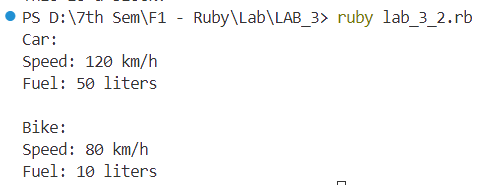
car.display\_info

bike = Bike.new(80, 10)

puts "\nBike:"

bike.display\_info

**Output**



1. **Write a ruby programming using Mixins concept**

module Readable

  def read

    puts "#{title} is being read."

  end

end

module Borrowable

  def borrow

    puts "#{title} has been borrowed."

  end

end

class Book

  include Readable

  include Borrowable

  attr\_accessor :title

  def initialize(title)

    @title = title

  end

end

class Magazine

  include Readable

  include Borrowable

  attr\_accessor :title

  def initialize(title)

    @title = title

  end

end

book = Book.new("The Great Gatsby")

magazine = Magazine.new("National Geographic")

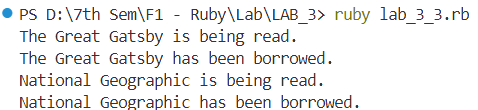
book.read

book.borrow

magazine.read

magazine.borrow

**Output**



1. **Write a ruby programming using Reflection concept.**

class Person

    attr\_accessor :name, :age

    def initialize(name, age)

      @name = name

      @age = age

    end

    def greet

      puts "Hello, my name is #{@name} and I am #{@age} years old."

    end

  end

  person = Person.new("Vishwanth", 18)

  puts "Calling greet method:"

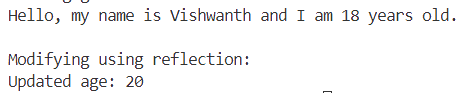
  person.send(:greet)

  puts "\nModifying using reflection:"

  person.instance\_variable\_set(:@age, 20)

  puts "Updated age: #{person.instance\_variable\_get(:@age)}"

**Output**



1. **Write a ruby programming using Meta-programming concept.**

class DynamicMethods

  def self.create\_method(name, &block)

    define\_method(name, &block)

  end

end

class Vehicle < DynamicMethods

end

vehicle = Vehicle.new

Vehicle.create\_method(:drive) do

  puts "The vehicle is driving."

end

Vehicle.create\_method(:fuel\_up) do |amount|

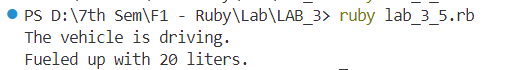
  puts "Fueled up with #{amount} liters."

end

vehicle.drive

vehicle.fuel\_up(20)

**Output**



1. **Create an array a=[1,2,3,4,5,6], and perform the following:**
   1. **Different ways to access the array elements**
   2. **Five different methods associated with array.**
   3. **Different ways to add and delete an element of an array.**
   4. **Introduce two new arrays and perform intersection, concatenation, difference.**
   5. **Perform a binary search using array a.**

def get\_numbers\_from\_user

  puts "Enter numbers separated by spaces:"

  gets.chomp.split.map(&:to\_i)

end

numbers = get\_numbers\_from\_user

puts "You entered: #{numbers}"

puts "\nArray basics:"

puts "First element: #{numbers[0]}"

puts "Last element: #{numbers[-1]}"

puts "Slice (2..4): #{numbers[2..4]}"

puts "First two elements: #{numbers.first(2)}"

puts "Last element removed: #{numbers.pop}"

puts "\nArray methods:"

puts "Length: #{numbers.length}"

puts "Reversed: #{numbers.reverse}"

puts "Sorted: #{numbers.sort}"

puts "Adding 5: #{numbers.push(5)}"

puts "\nAdding and removing elements:"

numbers.push(6)

numbers.insert(1, 7)

puts "After additions: #{numbers}"

numbers.delete(5)

puts "After deletion: #{numbers}"

new\_array = [10, 20, 30]

puts "\nArray operations with new array:"

puts "Concatenated: #{numbers + new\_array}"

puts "Difference: #{numbers - new\_array}"

def linear\_search(array, target)

  array.index(target) || "Not found"

end

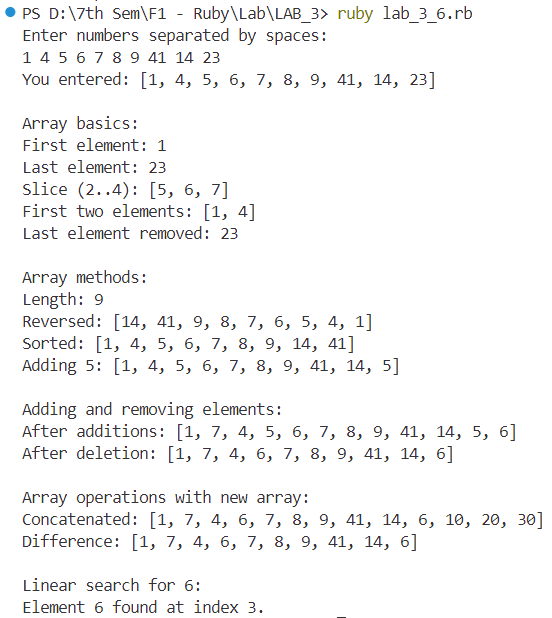
target = 6

index = linear\_search(numbers, target)

puts "\nLinear search for #{target}:"

puts "Element #{target} found at index #{index}." if index.is\_a?(Integer)

**Output**



**Assessment 3.1**

1. Scenario: Managing a Library Catalog Question: You are building a library catalog system in Ruby. Each book has multiple attributes such as title, author, genre, and publication year. Design a hash structure to store information about multiple books and implement a method to search for books published after a specific year.

class LibraryManagement

  def initialize

    @books = []

  end

  def add\_book

    print "Title: "

    title = gets.chomp

    print "Author: "

    author = gets.chomp

    print "Genre: "

    genre = gets.chomp

    print "Publication Year: "

    pub\_yr = gets.chomp

    @books << { title: title, author: author, genre: genre, pub\_yr: pub\_yr }

    puts "Book Added"

  end

  def display\_books

    if @books.empty?

      puts "No books available."

    else

      @books.each\_with\_index do |book, index|

        puts "Book #{index + 1}: #{book[:title]} by #{book[:author]}, Genre: #{book[:genre]}, Year: #{book[:pub\_yr]}"

      end

    end

  end

  def search\_book

    print "Publication Year: "

    year = gets.chomp

    result = @books.select { |book| book[:pub\_yr] == year }

    if result.empty?

      puts "No books from #{year}."

    else

      result.each\_with\_index do |book, index|

        puts "Book #{index + 1}: #{book[:title]} by #{book[:author]}, Genre: #{book[:genre]}, Year: #{book[:pub\_yr]}"

      end

    end

  end

end

library = LibraryManagement.new

loop do

  puts "\nMenu:"

  puts "1. Add Book"

  puts "2. Display Books"

  puts "3. Search Books"

  puts "4. Exit"

  print "Choice: "

  choice = gets.chomp.to\_i

  case choice

  when 1 then library.add\_book

  when 2 then library.display\_books

  when 3 then library.search\_book

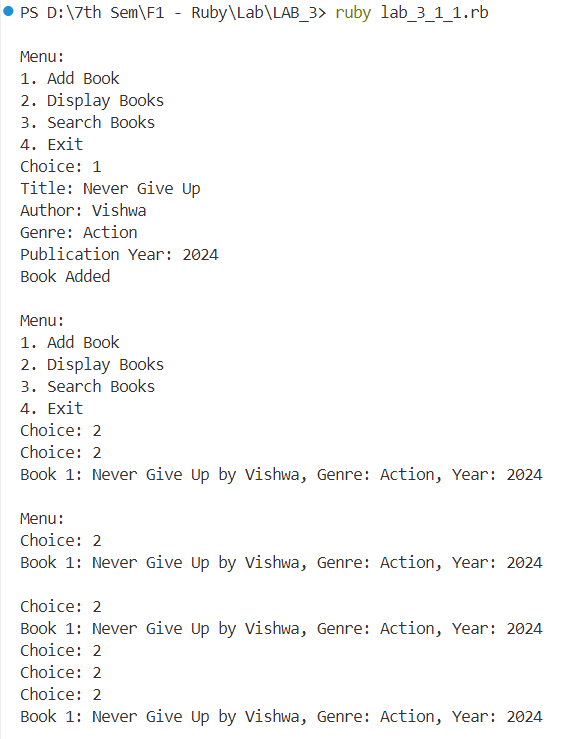
  when 4 then break

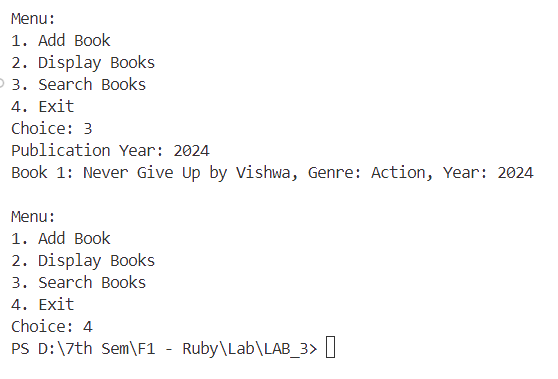
  else puts "Invalid choice."

  end

end

**Output**





1. Scenario: Employee Database Question: You are developing an employee management system. Design a hash structure to store information about employees, including their names, departments, and salaries. Implement a method to find the highest-paid employee and display their details.

class EmployeeManagement

  def initialize

    @employees = []

  end

  def add\_emp

    print "Name: "

    name = gets.chomp

    print "Department: "

    dept = gets.chomp

    print "Salary: "

    sal = gets.chomp.to\_i

    @employees << { name: name, dept: dept, sal: sal }

    puts "Employee Added"

  end

  def display\_emp

    if @employees.empty?

      puts "No employees found."

    else

      @employees.each\_with\_index do |emp, index|

        puts "\nEmployee #{index + 1}: #{emp[:name]}, #{emp[:dept]}, Salary: #{emp[:sal]}"

      end

    end

  end

  def high\_paid

    highest = @employees.max\_by { |emp| emp[:sal] }

    if highest

      puts "Highest Paid Employee: #{highest[:name]}, #{highest[:dept]}, Salary: #{highest[:sal]}"

    else

      puts "No records found."

    end

  end

end

employee\_mgmt = EmployeeManagement.new

loop do

  puts "\n1. Add Employee"

  puts "2. Display Employees"

  puts "3. Find Highest Paid Employee"

  puts "4. Exit"

  print "Choice: "

  choice = gets.chomp.to\_i

  case choice

  when 1 then employee\_mgmt.add\_emp

  when 2 then employee\_mgmt.display\_emp

  when 3 then employee\_mgmt.high\_paid

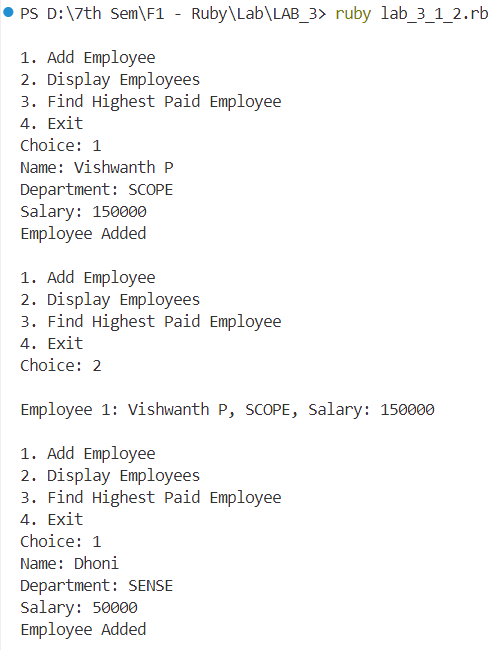
  when 4 then break

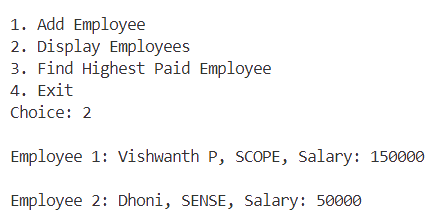
  else puts "Invalid choice."

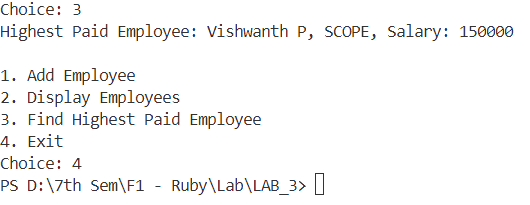
  end

end

**Output**







1. Scenario: Online Marketplace Question: You are creating an online marketplace where sellers can list their products. Design a hash structure to store information about products, including their names, prices, and quantities. Implement a method to calculate the total value of all products in the marketplace.

class MarketPlace

  def initialize

    @products = []

  end

  def add\_product

    print "Name: "

    name = gets.chomp

    print "Price: "

    price = gets.chomp.to\_i

    print "Quantity: "

    qty = gets.chomp.to\_i

    @products << { name: name, price: price, qty: qty }

    puts "Product Added"

  end

  def display\_product

    if @products.empty?

      puts "No products available."

    else

      @products.each\_with\_index do |prd, index|

        puts "\nProduct #{index + 1}: #{prd[:name]}, Price: #{prd[:price]}, Quantity: #{prd[:qty]}"

      end

    end

  end

  def tot\_value

    @products.each\_with\_index do |prd, index|

      puts "Total Value of Product #{index + 1} (#{prd[:name]}): Rs.#{prd[:price] \* prd[:qty]}"

    end

  end

end

marketplace = MarketPlace.new

loop do

  puts "\n1. Add Product"

  puts "2. Display Products"

  puts "3. Product Values"

  puts "4. Exit"

  print "Choice: "

  choice = gets.chomp.to\_i

  case choice

  when 1 then marketplace.add\_product

  when 2 then marketplace.display\_product

  when 3 then marketplace.tot\_value

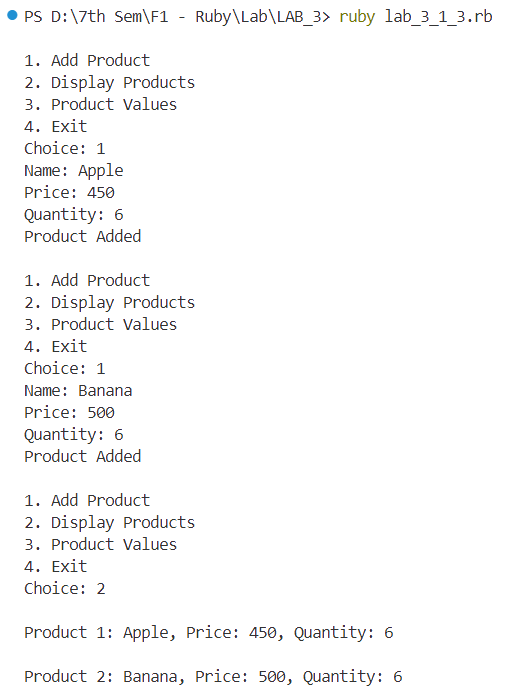
  when 4 then break

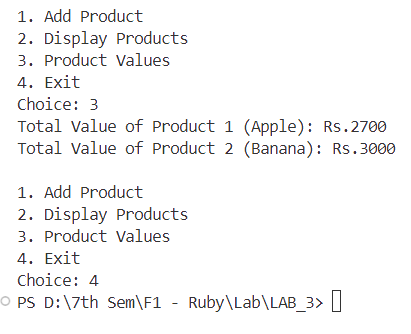
  else puts "Invalid choice."

  end

end

**Output**





1. Scenario: Student Grades Question: You are building a system to manage student grades. Design a hash structure to store information about multiple students, including their names and an array of grades for different subjects. Implement a method to calculate the average grade for each student and display the results.

class GradeManagement

  def initialize

    @students = {}

  end

  def add\_student

    print "Student Name: "

    name = gets.chomp

    subjects = []

    grades = []

    print "Number of Subjects: "

    num\_subjects = gets.chomp.to\_i

    num\_subjects.times do |i|

      print "Subject ##{i + 1}: "

      subjects << gets.chomp

      print "Grade: "

      grades << gets.chomp.to\_i

    end

    @students[name] = { subjects: subjects, grades: grades }

    puts "Student Added"

  end

  def calculate\_averages

    if @students.empty?

      puts "No records found."

    else

      @students.each do |name, details|

        avg = details[:grades].sum.to\_f / details[:grades].size

        puts "#{name}'s Average: #{avg.round(2)}"

      end

    end

  end

end

grade\_management = GradeManagement.new

loop do

  puts "\n1. Add Student"

  puts "2. Calculate Averages"

  puts "3. Exit"

  print "Choice: "

  choice = gets.chomp.to\_i

  case choice

  when 1 then grade\_management.add\_student

  when 2 then grade\_management.calculate\_averages

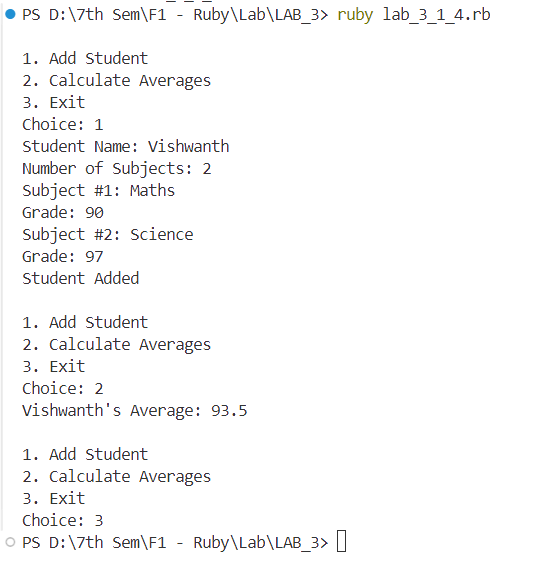
  when 3 then break

  else puts "Invalid choice."

  end

end

**Output**



1. Scenario: Music Playlist Question: You are developing a music playlist application. Design a hash structure to store information about songs, including their titles, artists, and genres. Implement a method to shuffle the playlist randomly and play the songs in a random order.

class PlaylistManagement

  def initialize

    @playlist = []

    @shuffled\_playlist = []

  end

  def add\_song

    print "Title: "

    title = gets.chomp

    print "Artist: "

    artist = gets.chomp

    print "Genre: "

    genre = gets.chomp

    @playlist << { title: title, artist: artist, genre: genre }

    @shuffled\_playlist.clear

    puts "Song added."

  end

  def shuffle\_and\_play

    if @playlist.empty?

      puts "The playlist is empty."

    else

      @shuffled\_playlist = @playlist.shuffle if @shuffled\_playlist.empty?

      song = @shuffled\_playlist.shift

      puts "Playing: #{song[:title]} by #{song[:artist]} (Genre: #{song[:genre]})"

    end

  end

end

playlist\_manager = PlaylistManagement.new

loop do

  puts "\n1. Add Song"

  puts "2. Shuffle and Play"

  puts "3. Exit"

  print "Choice: "

  choice = gets.chomp.to\_i

  case choice

  when 1 then playlist\_manager.add\_song

  when 2 then playlist\_manager.shuffle\_and\_play

  when 3 then break

  else puts "Invalid choice."

  end

end

**Output**



**3.2 CAT – 1 Solution**

**Question – 1**

def generate\_temp\_data

  temps = {}

  (1..365).each do |day|

    temps[day] = rand(15..35)

  end

  temps

end

def calc\_avg\_temp(temps)

  total\_temp = temps.values.sum

  total\_temp.to\_f / temps.size

end

def find\_temp\_extremes(temps)

  hot\_day, hot\_temp = temps.max\_by { |\_day, temp| temp }

  cold\_day, cold\_temp = temps.min\_by { |\_day, temp| temp }

  { hot\_day: hot\_day, hot\_temp: hot\_temp, cold\_day: cold\_day, cold\_temp: cold\_temp }

end

def calc\_monthly\_avg(temps)

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

  monthly\_avg = {}

  day = 1

  days\_in\_month.each\_with\_index do |days, idx|

    month\_temps = temps.slice(day, days).values

    monthly\_avg[idx + 1] = month\_temps.sum.to\_f / days

    day += days

  end

  monthly\_avg

end

def find\_long\_heatwave(temps)

  heatwave = 0

  max\_wave = 0

  temps.each\_value do |temp|

    if temp > 30

      heatwave += 1

    else

      max\_wave = [max\_wave, heatwave].max

      heatwave = 0

    end

  end

  max\_wave

end

def find\_long\_cold\_spell(temps)

  cold\_spell = 0

  max\_spell = 0

  temps.each\_value do |temp|

    if temp < 20

      cold\_spell += 1

    else

      max\_spell = [max\_spell, cold\_spell].max

      cold\_spell = 0

    end

  end

  max\_spell

end

def find\_hot\_month(monthly\_avg)

  monthly\_avg.max\_by { |\_month, avg\_temp| avg\_temp }.first

end

temps = generate\_temp\_data

avg\_temp = calc\_avg\_temp(temps)

extremes = find\_temp\_extremes(temps)

monthly\_avg = calc\_monthly\_avg(temps)

long\_heatwave = find\_long\_heatwave(temps)

long\_cold\_spell = find\_long\_cold\_spell(temps)

hot\_month = find\_hot\_month(monthly\_avg)

puts "Average Temp: #{avg\_temp}"

puts "Hottest Day: #{extremes[:hot\_day]} (#{extremes[:hot\_temp]}°C)"

puts "Coldest Day: #{extremes[:cold\_day]} (#{extremes[:cold\_temp]}°C)"

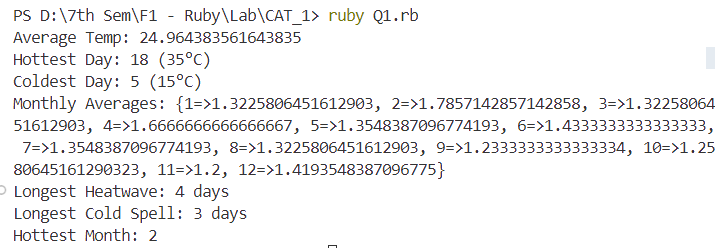
puts "Monthly Averages: #{monthly\_avg}"

puts "Longest Heatwave: #{long\_heatwave} days"

puts "Longest Cold Spell: #{long\_cold\_spell} days"

puts "Hottest Month: #{hot\_month}"

Output



**Question – 2**

class NumAnalyzer

  def find\_heads(nums, n)

    puts "Head numbers: "

    (1...n - 1).each do |i|

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

        puts nums[i]

      end

    end

  end

  def find\_max\_pair(nums, n)

    max\_pair = [nums[0], nums[1]]

    max\_sum = nums[0] + nums[1]

    (0...n - 1).each do |i|

      (i + 1...n).each do |j|

        cur\_sum = nums[i] + nums[j]

        if cur\_sum > max\_sum

          max\_sum = cur\_sum

          max\_pair = [nums[i], nums[j]]

        end

      end

    end

    max\_pair

  end

end

puts "Enter number of elements: "

n = gets.chomp.to\_i

nums = []

puts "Enter elements: "

n.times do

  nums << gets.chomp.to\_i

end

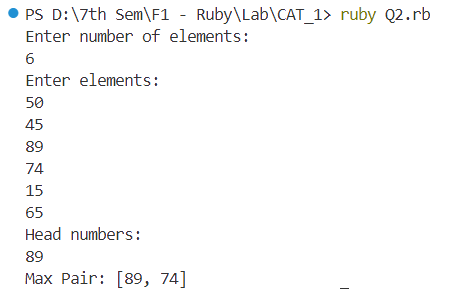
analyzer = NumAnalyzer.new

analyzer.find\_heads(nums, n)

max\_pair = analyzer.find\_max\_pair(nums, n)

puts "Max Pair: #{max\_pair}"

Output



**Question – 3**

class DynamicDispatcher

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

    if method\_name.to\_s.start\_with?("calculate")

      operation = method\_name.to\_s.split("calculate")[1]

      perform\_calculation(operation, args)

    else

      super

    end

  end

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

    method\_name.to\_s.start\_with?("calculate") || super

  end

  private

  def perform\_calculation(operation, args)

    case operation

    when "Factorial"

      puts factorial(args[0])

    when "Square"

      puts square(args[0])

    else

      puts "Unsupported operation: #{operation}"

    end

  end

  def factorial(n)

    return 1 if n == 0

    n \* factorial(n - 1)

  end

  def square(n)

    n \* n

  end

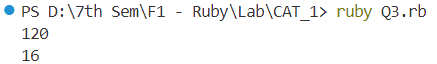
end

dispatcher = DynamicDispatcher.new

dispatcher.calculateFactorial(5)

dispatcher.calculateSquare(4)

Output



**Question – 4**

class BracketChecker

  def balanced\_parentheses(str)

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

    stack = []

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

      if pairs.key?(char)

        stack.push([char, idx])

      elsif pairs.value?(char)

        return "Unbalanced at position: #{idx + 1}" if stack.empty? || pairs[stack.last[0]] != char

        stack.pop

      end

    end

    stack.empty? ? true : "Unbalanced at position: #{stack.last[1] + 1}"

  end

  def evaluate\_expression(str)

    return balanced\_parentheses(str) unless balanced\_parentheses(str) == true

    exprs = str.scan(/\([^)]+\)/)

    exprs.each do |expr|

      result = eval(expr[1..-2])

      str.sub!(expr, result.to\_s)

    end

    str

  end

end

checker = BracketChecker.new

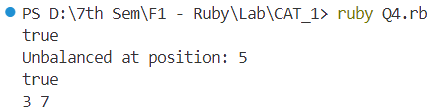
puts checker.balanced\_parentheses("(11)")

puts checker.balanced\_parentheses("|(|)])")

puts checker.balanced\_parentheses("<(1+2) (3+4)>")

puts checker.evaluate\_expression("(1+2) (3+4)")

Output



**Question – 5**

require 'fileutils'

class WordCounter

  def initialize(filename)

    @filename = filename

    @word\_counts = {}

  end

  def analyze\_word\_usage

    read\_file

    count\_words

    output\_results

  end

  private

  def read\_file

    File.open(@filename, 'r') do |file|

      @text = file.read

    end

  end

  def count\_words

    words = @text.downcase.gsub(/[^a-z\s]/, '').split

    words.each do |word|

      @word\_counts[word] ||= 0

      @word\_counts[word] += 1

    end

  end

  def output\_results

    sorted\_words = @word\_counts.sort\_by { |\_word, count| -count }

    puts "Top 10 Most Frequent Words:"

    sorted\_words[0..9].each do |word, count|

      puts "#{word}: #{count}"

    end

  end

end

filename = 'sample.txt'

word\_counter = WordCounter.new(filename)

word\_counter.analyze\_word\_usage

sample.txt

Mahendra Singh Dhoni (born 7 July 1981) is an Indian professional cricketer who plays as a right-handed batter and a wicket-keeper. Widely regarded as one of the most prolific wicket-keeper batsmen and captains, he represented the Indian cricket team and was the captain of the side in limited overs formats from 2007 to 2017 and in test cricket from 2008 to 2014.

Output

