# RUBY LAB -3

**Name: Vishwanth P**

**Register No: 21MIS1117**

## Assessment 3

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

```ruby
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

```
PS D:\7th Sem\F1 - Ruby\Lab\LAB_3> ruby lab_3_1.rb
Hi!
Hello, Vishwanth!
Bye!
Hi, Vishwanth!
6
3
Goodbye, Vishwanth!
This is a block!
PS D:\7th Sem\F1 - Ruby\Lab\LAB_3>
```

## 2. Write a ruby programming using Modules concept.

```ruby
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
```

```ruby
  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

```
PS D:\7th Sem\F1 - Ruby\Lab\LAB_3> ruby lab_3_2.rb
Car:
Speed: 120 km/h
Fuel: 50 liters

Bike:
Speed: 80 km/h
Fuel: 10 liters
```

## 3. Write a ruby programming using Mixins concept

```ruby
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

```
PS D:\7th Sem\F1 - Ruby\Lab\LAB_3> ruby lab_3_3.rb
The Great Gatsby is being read.
The Great Gatsby has been borrowed.
National Geographic is being read.
National Geographic has been borrowed.
```

## 4. Write a ruby programming using Reflection concept.

```ruby
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

```
Hello, my name is Vishwanth and I am 18 years old.

Modifying using reflection:
Updated age: 20
```

## 5. Write a ruby programming using Meta-programming concept.

```ruby
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

```
PS D:\7th Sem\F1 - Ruby\Lab\LAB_3> ruby lab_3_5.rb
The vehicle is driving.
Fueled up with 20 liters.
```

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

```ruby
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}"
```

```ruby
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

```
PS D:\7th Sem\F1 - Ruby\Lab\LAB_3> ruby lab_3_6.rb
Enter numbers separated by spaces:
1 4 5 6 7 8 9 41 14 23
You entered: [1, 4, 5, 6, 7, 8, 9, 41, 14, 23]

Array basics:
First element: 1
Last element: 23
Slice (2..4): [5, 6, 7]
First two elements: [1, 4]
Last element removed: 23

Array methods:
Length: 9
Reversed: [14, 41, 9, 8, 7, 6, 5, 4, 1]
Sorted: [1, 4, 5, 6, 7, 8, 9, 14, 41]
Adding 5: [1, 4, 5, 6, 7, 8, 9, 41, 14, 5]

Adding and removing elements:
After additions: [1, 7, 4, 5, 6, 7, 8, 9, 41, 14, 5, 6]
After deletion: [1, 7, 4, 6, 7, 8, 9, 41, 14, 6]

Array operations with new array:
Concatenated: [1, 7, 4, 6, 7, 8, 9, 41, 14, 6, 10, 20, 30]
Difference: [1, 7, 4, 6, 7, 8, 9, 41, 14, 6]

Linear search for 6:
Element 6 found at index 3.
```

# 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.

```ruby
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
```

```ruby
      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**

```
PS D:\7th Sem\F1 - Ruby\Lab\LAB_3> ruby lab_3_1_1.rb

Menu:
1. Add Book
2. Display Books
3. Search Books
4. Exit
Choice: 1
Title: Never Give Up
Author: Vishwa
Genre: Action
Publication Year: 2024
Book Added

Menu:
1. Add Book
2. Display Books
3. Search Books
4. Exit
Choice: 2
Choice: 2
Book 1: Never Give Up by Vishwa, Genre: Action, Year: 2024

Menu:
Choice: 2
Book 1: Never Give Up by Vishwa, Genre: Action, Year: 2024

Choice: 2
Book 1: Never Give Up by Vishwa, Genre: Action, Year: 2024
Choice: 2
Choice: 2
Choice: 2
Book 1: Never Give Up by Vishwa, Genre: Action, Year: 2024
```

```
Menu:
1. Add Book
2. Display Books
3. Search Books
4. Exit
Choice: 3
Publication Year: 2024
Book 1: Never Give Up by Vishwa, Genre: Action, Year: 2024

Menu:
1. Add Book
2. Display Books
3. Search Books
4. Exit
Choice: 4
PS D:\7th Sem\F1 - Ruby\Lab\LAB_3>
```

2. 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.

```ruby
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
```

```ruby
    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

```
PS D:\7th Sem\F1 - Ruby\Lab\LAB_3> ruby lab_3_1_2.rb

1. Add Employee
2. Display Employees
3. Find Highest Paid Employee
4. Exit
Choice: 1
Name: Vishwanth P
Department: SCOPE
Salary: 150000
Employee Added

1. Add Employee
2. Display Employees
3. Find Highest Paid Employee
4. Exit
Choice: 2

Employee 1: Vishwanth P, SCOPE, Salary: 150000

1. Add Employee
2. Display Employees
3. Find Highest Paid Employee
4. Exit
Choice: 1
Name: Dhoni
Department: SENSE
Salary: 50000
Employee Added
```

```
1. Add Employee
2. Display Employees
3. Find Highest Paid Employee
4. Exit
Choice: 2

Employee 1: Vishwanth P, SCOPE, Salary: 150000

Employee 2: Dhoni, SENSE, Salary: 50000
Choice: 3
Highest Paid Employee: Vishwanth P, SCOPE, Salary: 150000

1. Add Employee
2. Display Employees
3. Find Highest Paid Employee
4. Exit
Choice: 4
PS D:\7th Sem\F1 - Ruby\Lab\LAB_3>
```

3. 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.

```ruby
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
```

```ruby
  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

```
PS D:\7th Sem\F1 - Ruby\Lab\LAB_3> ruby lab_3_1_3.rb

1. Add Product
2. Display Products
3. Product Values
4. Exit
Choice: 1
Name: Apple
Price: 450
Quantity: 6
Product Added

1. Add Product
2. Display Products
3. Product Values
4. Exit
Choice: 1
Name: Banana
Price: 500
Quantity: 6
Product Added

1. Add Product
2. Display Products
3. Product Values
4. Exit
Choice: 2

Product 1: Apple, Price: 450, Quantity: 6

Product 2: Banana, Price: 500, Quantity: 6
```

```
1. Add Product
2. Display Products
3. Product Values
4. Exit
Choice: 3
Total Value of Product 1 (Apple): Rs.2700
Total Value of Product 2 (Banana): Rs.3000

1. Add Product
2. Display Products
3. Product Values
4. Exit
Choice: 4
PS D:\7th Sem\F1 - Ruby\Lab\LAB_3>
```

4. 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.

```ruby
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
```

```ruby
    @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**

```
PS D:\7th Sem\F1 - Ruby\Lab\LAB_3> ruby lab_3_1_4.rb

1. Add Student
2. Calculate Averages
3. Exit
Choice: 1
Student Name: Vishwanth
Number of Subjects: 2
Subject #1: Maths
Grade: 90
Subject #2: Science
Grade: 97
Student Added

1. Add Student
2. Calculate Averages
3. Exit
Choice: 2
Vishwanth's Average: 93.5

1. Add Student
2. Calculate Averages
3. Exit
Choice: 3
PS D:\7th Sem\F1 - Ruby\Lab\LAB_3>
```

5. 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.

```ruby
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
```

```ruby
  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

```
PS D:\7th Sem\F1 - Ruby\Lab\LAB_3> ruby lab_3_1_5.rb

1. Add Song
2. Shuffle and Play
3. Exit
Choice: 1
Title: Leo BGM
Artist: Aniruth
Genre: Action
Song added.

1. Add Song
2. Shuffle and Play
3. Exit
Choice: 1
Title: Mental Manadhil
Artist: AR Rahman
Genre: Love
Song added.

1. Add Song
2. Shuffle and Play
3. Exit
Choice: 2
Playing: Leo BGM by Aniruth (Genre: Action)

1. Add Song
2. Shuffle and Play
3. Exit
Choice: 3
PS D:\7th Sem\F1 - Ruby\Lab\LAB_3>
```

# 3.2 CAT – 1 Solution

## Question – 1

```ruby
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|
```

```ruby
    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

```
PS D:\7th Sem\F1 - Ruby\Lab\CAT_1> ruby Q1.rb
Average Temp: 24.964383561643835
Hottest Day: 18 (35°C)
Coldest Day: 5 (15°C)
Monthly Averages: {1=>1.3225806451612903, 2=>1.7857142857142858, 3=>1.3225806451612903, 4=>1.6666666666666667, 5=>1.3548387096774193, 6=>1.4333333333333333, 7=>1.3548387096774193, 8=>1.3225806451612903, 9=>1.2333333333333334, 10=>1.2580645161290323, 11=>1.2, 12=>1.4193548387096775}
Longest Heatwave: 4 days
Longest Cold Spell: 3 days
Hottest Month: 2
```

# Question – 2

```ruby
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: "
```

```ruby
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

```
PS D:\7th Sem\F1 - Ruby\Lab\CAT_1> ruby Q2.rb
Enter number of elements:
6
Enter elements:
50
45
89
74
15
65
Head numbers:
89
Max Pair: [89, 74]
```

# Question – 3

```ruby
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
```

```ruby
  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

```
PS D:\7th Sem\F1 - Ruby\Lab\CAT_1> ruby Q3.rb
120
16
```

# Question – 4

```ruby
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

```
PS D:\7th Sem\F1 - Ruby\Lab\CAT_1> ruby Q4.rb
true
Unbalanced at position: 5
true
3 7
```

# Question – 5

```ruby
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:"
```

```ruby
    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

```
PS D:\7th Sem\F1 - Ruby\Lab\CAT_1> ruby Q5.rb
Top 10 Most Frequent Words:
the: 4
and: 4
to: 2
from: 2
of: 2
in: 2
indian: 2
wicketkeeper: 2
as: 2
a: 2
```