# Ruby Lab Assessment – 3

Suryakumar P 21MIS1146

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

Code:

*# a method that uses yield*

def greet\_with\_yield

  puts "Inside the method before the block."

  yield if block\_given?

  puts "Inside the method after the block."

end

greet\_with\_yield do

  puts "Hello from the block!"

end

puts "\n LAMBDA AND PROC \n"

*# a lambda that returns a greeting message*

greet\_lambda = lambda { |name| "Hello, #{name} from the lambda!" }

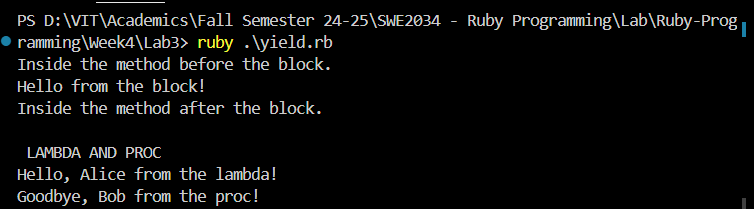
puts greet\_lambda.call("Alice")

*# a proc that returns a farewell message*

farewell\_proc = Proc.new { |name| "Goodbye, #{name} from the proc!" }

puts farewell\_proc.call("Bob")

Output:



1. Write a ruby programming using Modules concept.

Code:

module MathOperations

   def add(a, b)

     a + b

   end

   def subtract(a, b)

     a - b

   end

   def multiply(a, b)

     a \* b

   end

   def divide(a, b)

     return "Division by zero error" if b == 0

     a / b

   end

 end

 class Calculator

   include MathOperations

 end

 calc = Calculator.new

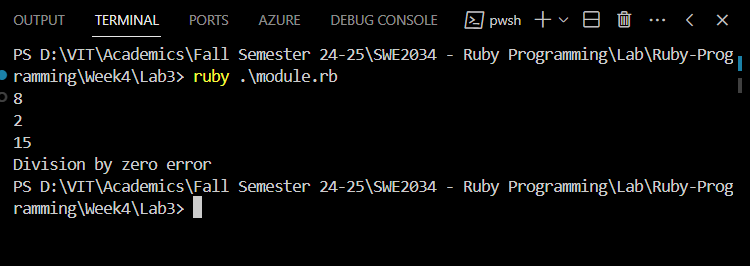
 puts calc.add(5, 3)

 puts calc.subtract(5, 3)

 puts calc.multiply(5, 3)

 puts calc.divide(5, 0)

Output:



1. Write a ruby programming using Mixins concept.

Code:

module Walkable

   def walk

     puts "#{self.class} is walking."

   end

 end

 class Human

   include Walkable

 end

 class Dog

   include Walkable

 end

 class Robot

   include Walkable

 end

 human = Human.new

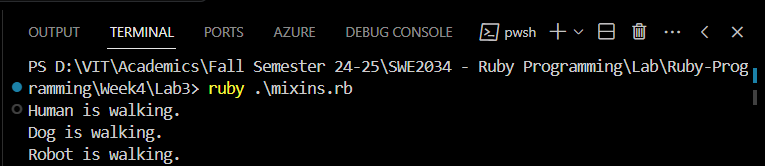
 dog = Dog.new

 robot = Robot.new

 human.walk

 dog.walk

 robot.walk

Output:  


1. Write a ruby programming using Reflection concept.

Code:

class Person

    attr\_accessor :name, :age

    def initialize(name, age)

      @name = name

      @age = age

    end

    def greet

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

    end

  end

  person = Person.new("Surya", 20)

  puts "Class: #{person.class}"

  puts "\nInstance Variables:"

  puts person.instance\_variables

  puts "\nName (using reflection): #{person.instance\_variable\_get(:@name)}"

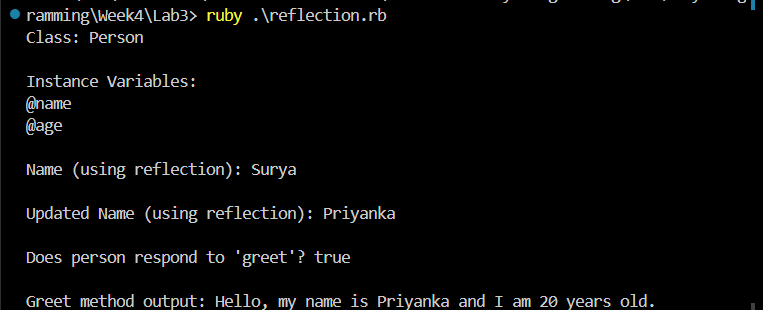
  person.instance\_variable\_set(:@name, "Priyanka")

  puts "\nUpdated Name (using reflection): #{person.name}"

  puts "\nDoes person respond to 'greet'? #{person.respond\_to?(:greet)}"

  puts "\nGreet method output: #{person.greet}"

Output:



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

Code:

class DynamicAttributes

    def initialize(attributes = {})

      attributes.each do |key, value|

        self.class.send(:define\_method, key) do

          instance\_variable\_get("@#{key}")

        end

        self.class.send(:define\_method, "#{key}=") do |val|

          instance\_variable\_set("@#{key}", val)

        end

        instance\_variable\_set("@#{key}", value)

      end

    end

  end

  person = DynamicAttributes.new(name: "Surya", age: 20)

  puts person.name

  puts person.age

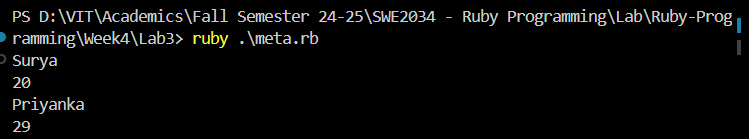
  person.name = "Priyanka"

  person.age = 29

  puts person.name

  puts person.age

Output:



6. Create an array a=[1,2,3,4,5,6], and perform the following:

* Different ways to access the array elements
* Five different methods associated with array.
* Different ways to add and delete an element of an array.
* Introduce two new arrays and perform intersection, concatenation, difference.
* Perform a binary search using array a.

Code:

*# Create an array*

a = [1, 2, 3, 4, 5, 6]

*# a. Different ways to access the array elements*

puts "Accessing elements:"

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

puts "Third element: #{a[2]}"

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

puts "Range of elements (index 1 to 3): #{a[1..3]}"

puts "Using fetch method: #{a.fetch(2)}"

*# b. Five different methods associated with array*

puts "\nArray methods:"

puts "Length of array: #{a.length}"

puts "Reversed array: #{a.reverse}"

puts "Array includes 3? #{a.include?(3)}"

puts "Joined array into string: #{a.join('-')}"

puts "Sum of all elements: #{a.sum}"

*# c. Different ways to add and delete an element of an array*

puts "\nAdding and deleting elements:"

a.push(7)          *# Adding an element to the end*

puts "Array after push(7): #{a}"

a << 8             *# Another way to add an element to the end*

puts "Array after << 8: #{a}"

a.unshift(0)       *# Adding an element to the beginning*

puts "Array after unshift(0): #{a}"

a.insert(3, 10)    *# Adding an element at index 3*

puts "Array after insert(3, 10): #{a}"

a.pop              *# Removing the last element*

puts "Array after pop: #{a}"

a.shift            *# Removing the first element*

puts "Array after shift: #{a}"

a.delete\_at(2)     *# Removing an element at index 2*

puts "Array after delete\_at(2): #{a}"

a.delete(10)       *# Removing a specific element by value*

puts "Array after delete(10): #{a}"

*# d. Introduce two new arrays and perform intersection, concatenation, difference*

b = [4, 5, 6, 7, 8]

c = [1, 2, 9, 10]

puts "\nArray b: #{b}"

puts "Array c: #{c}"

intersection = a & b

puts "Intersection of a and b: #{intersection}"

concatenation = a + c

puts "Concatenation of a and c: #{concatenation}"

difference = a - b

puts "Difference between a and b: #{difference}"

*# e. Perform a binary search using array a*

*# Ensure the array is sorted*

a.sort!

puts "\nSorted array a: #{a}"

*# Binary search for the element '4'*

def binary\_search(arr, target)

  low = 0

  high = arr.length - 1

  while low <= high

    mid = (low + high) / 2

    if arr[mid] == target

      return "Element #{target} found at index #{mid}"

    elsif arr[mid] < target

      low = mid + 1

    else

      high = mid - 1

    end

  end

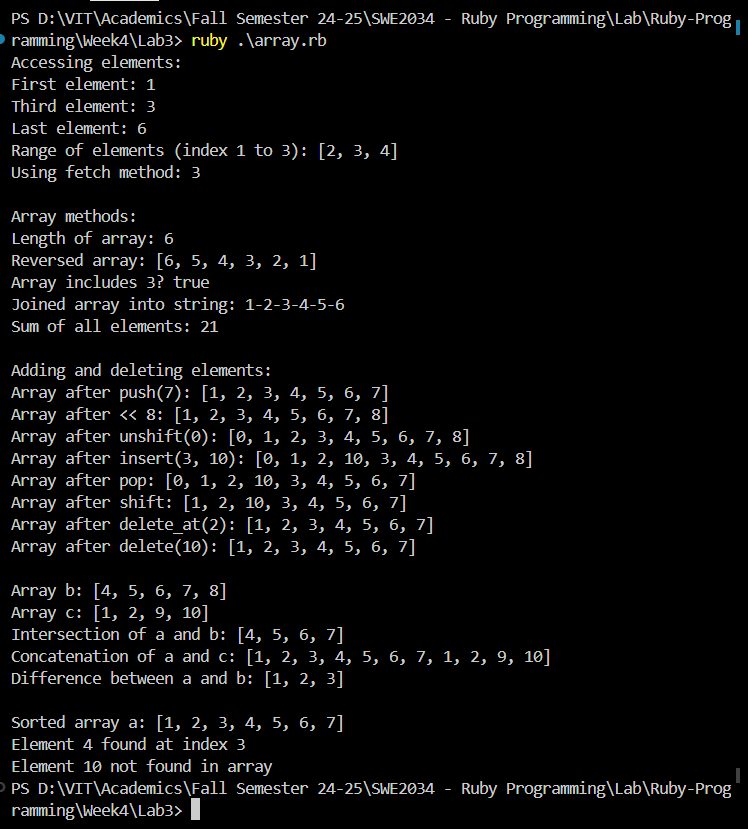
  return "Element #{target} not found in array"

end

puts binary\_search(a, 4)

puts binary\_search(a, 10)

Output:



3.1

#Library Catalog:  
#Code:

library\_catalog = {}

def add\_book\_to\_catalog(catalog)

  puts "Enter the book's title:"

  title = gets.chomp

  puts "Enter the author's name:"

  author = gets.chomp

  puts "Enter the genre:"

  genre = gets.chomp

  puts "Enter the publication year:"

  year = gets.chomp.to\_i

  id = catalog.size + 1

  catalog[id] = { title: title, author: author, genre: genre, year: year }

end

def books\_published\_after(catalog, year)

  catalog.select { |id, book| book[:year] > year }

end

loop do

  add\_book\_to\_catalog(library\_catalog)

  puts "Do you want to add another book? (yes/no)"

  answer = gets.chomp.downcase

  break if answer != 'yes'

end

puts "Enter the year to search for books published after:"

search\_year = gets.chomp.to\_i

searched\_books = books\_published\_after(library\_catalog, search\_year)

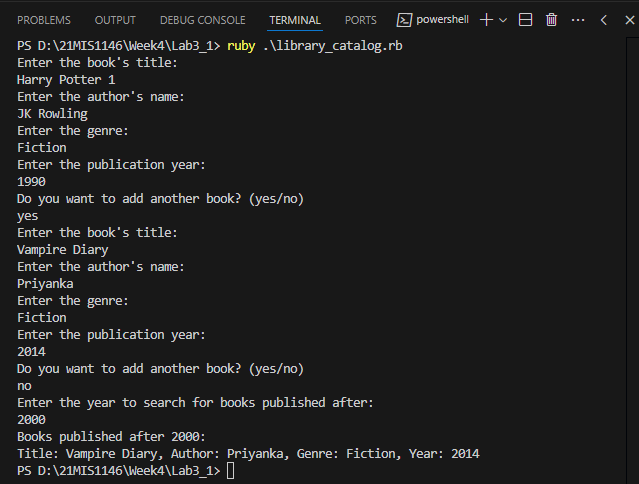
puts "Books published after #{search\_year}:"

searched\_books.each do |id, book|

  puts "Title: #{book[:title]}, Author: #{book[:author]}, Genre: #{book[:genre]}, Year: #{book[:year]}"

end

Output:



Employee Database Management:

Code:

employee\_database = {}

def add\_employee\_to\_database(database)

  puts "Enter the employee's name:"

  name = gets.chomp

  puts "Enter the employee's department:"

  department = gets.chomp

  puts "Enter the employee's salary:"

  salary = gets.chomp.to\_f

  id = database.size + 1

  database[id] = { name: name, department: department, salary: salary }

end

def highest\_paid\_employee(database)

  database.max\_by { |id, employee| employee[:salary] }

end

loop do

  add\_employee\_to\_database(employee\_database)

  puts "Do you want to add another employee? (yes/no)"

  answer = gets.chomp.downcase

  break if answer != 'yes'

end

highest\_paid = highest\_paid\_employee(employee\_database)

if highest\_paid

  id, details = highest\_paid

  puts "The highest-paid employee is:"

  puts "Name: #{details[:name]}"

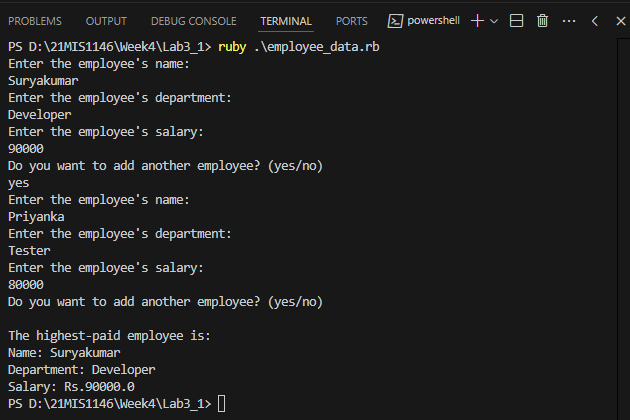
  puts "Department: #{details[:department]}"

  puts "Salary: Rs.#{details[:salary]}"

else

  puts "No employees found in the database."

end

Output:  


Market Place

Code:

marketplace = {}

def add\_product\_to\_marketplace(marketplace)

  puts "Enter the product name:"

  name = gets.chomp

  puts "Enter the product price:"

  price = gets.chomp.to\_f

  puts "Enter the product quantity:"

  quantity = gets.chomp.to\_i

  id = marketplace.size + 1

  marketplace[id] = { name: name, price: price, quantity: quantity }

end

def total\_value\_of\_products(marketplace)

  marketplace.sum { |id, product| product[:price] \* product[:quantity] }

end

loop do

  add\_product\_to\_marketplace(marketplace)

  puts "Do you want to add another product? (yes/no)"

  answer = gets.chomp.downcase

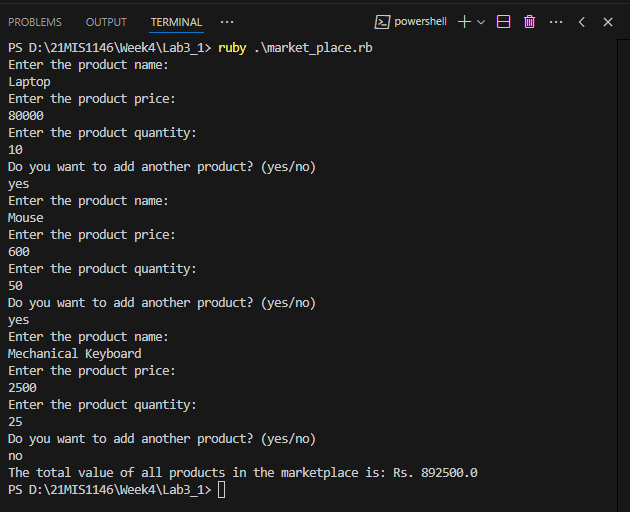
  break if answer != 'yes'

end

total\_value = total\_value\_of\_products(marketplace)

puts "The total value of all products in the marketplace is: Rs. #{total\_value}"

Output:



Student Grade:

Code:  
def grade\_to\_points(grade)

  case grade

  when 'S' then 10

  when 'A' then 9

  when 'B' then 8

  when 'C' then 7

  when 'D' then 6

  when 'F' then 0

  else 0

  end

end

def calculate\_cgpa(students\_grades)

  students\_grades.each do |student, grades|

    total\_points = grades.map { |grade| grade\_to\_points(grade) }.sum

    cgpa = total\_points.to\_f / grades.size

    puts "#{student}'s CGPA: #{cgpa.round(2)}"

  end

end

def get\_student\_grades

  students\_grades = {}

  puts "Enter the number of students:"

  number\_of\_students = gets.chomp.to\_i

  number\_of\_students.times do

    puts "Enter the student's name:"

    student\_name = gets.chomp

    puts "Enter the grades for #{student\_name} (separated by spaces):"

    grades = gets.chomp.split

    students\_grades[student\_name] = grades

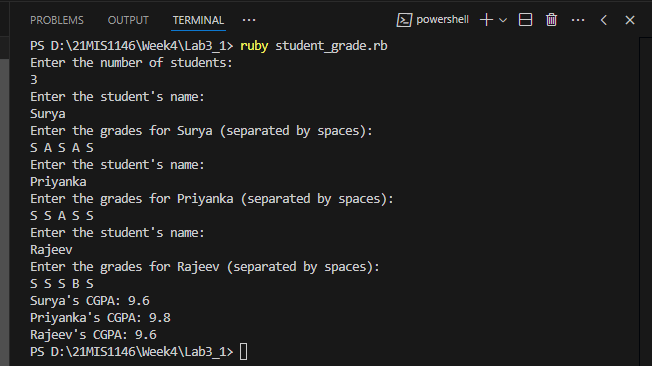
  end

  students\_grades

end

students\_grades = get\_student\_grades

calculate\_cgpa(students\_grades)

Output:  


Music Player:

Code:

playlist = {}

def add\_song\_to\_playlist(playlist)

  puts "Enter the song title:"

  title = gets.chomp

  puts "Enter the artist name:"

  artist = gets.chomp

  puts "Enter the genre:"

  genre = gets.chomp

  id = playlist.size + 1

  playlist[id] = { title: title, artist: artist, genre: genre }

end

def shuffle\_and\_play(playlist)

  shuffled\_playlist = playlist.keys.shuffle

  puts "Playing songs in random order:"

  shuffled\_playlist.each do |id|

    song = playlist[id]

    puts "Now playing: '#{song[:title]}' by #{song[:artist]} [#{song[:genre]}]"

  end

end

loop do

  add\_song\_to\_playlist(playlist)

  puts "Do you want to add another song? (yes/no)"

  answer = gets.chomp.downcase

  break if answer != 'yes'

end

shuffle\_and\_play(playlist)

Output:

