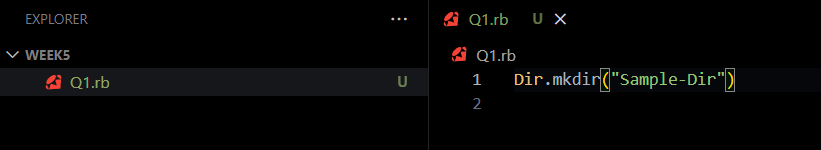
Ruby Lab Assessment 4

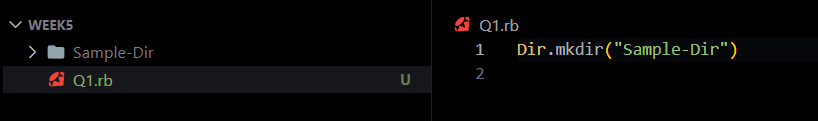
Suryakumar P 21MIS1146

1. Create a directory in Ruby

Before Creating:



After:



1. Check if a directory exists in Ruby

Code:

if Dir.exist?("Sample-Dir")

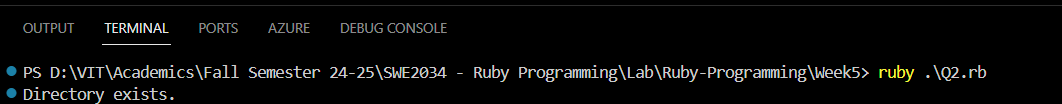
    puts "Directory exists."

  else

    puts "Directory does not exist."

  end

Output:



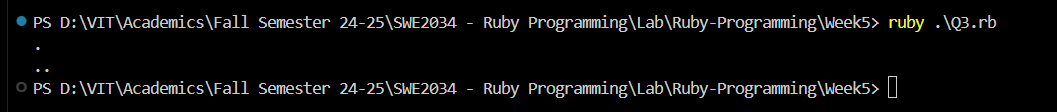
1. List files in a directory using Ruby

Code:

files = Dir.entries("Sample-Dir")

files.each { |file| puts file }

Output: (No Files inside)



1. Create a file in Ruby

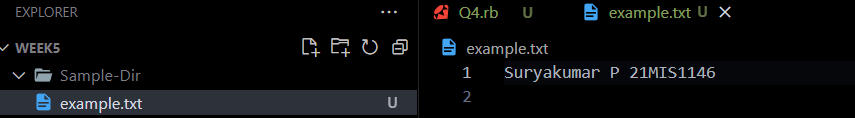
Code:

File.open("example.txt", "w") do |file|

    file.puts "Suryakumar P 21MIS1146"

  end

Output:



1. Write to a file in Ruby and use various methods like seek ( ), lineno( ), eof( ), size( )

Code:

File.open("example.txt", "w") do |file|

    file.puts "Line 1: Ruby is fun!"

    file.puts "Line 2: File handling is easy."

    file.puts "Line 3: Let's explore more features."

  end

  File.open("example.txt", "r") do |file|

    file.each\_line do |line|

      puts "#{file.lineno}: #{line}"

    end

    if file.eof?

      puts "Reached the end of the file."

    else

      puts "Still reading the file."

    end

    file.seek(0, IO::SEEK\_SET)

    puts "Moved back to the beginning of the file."

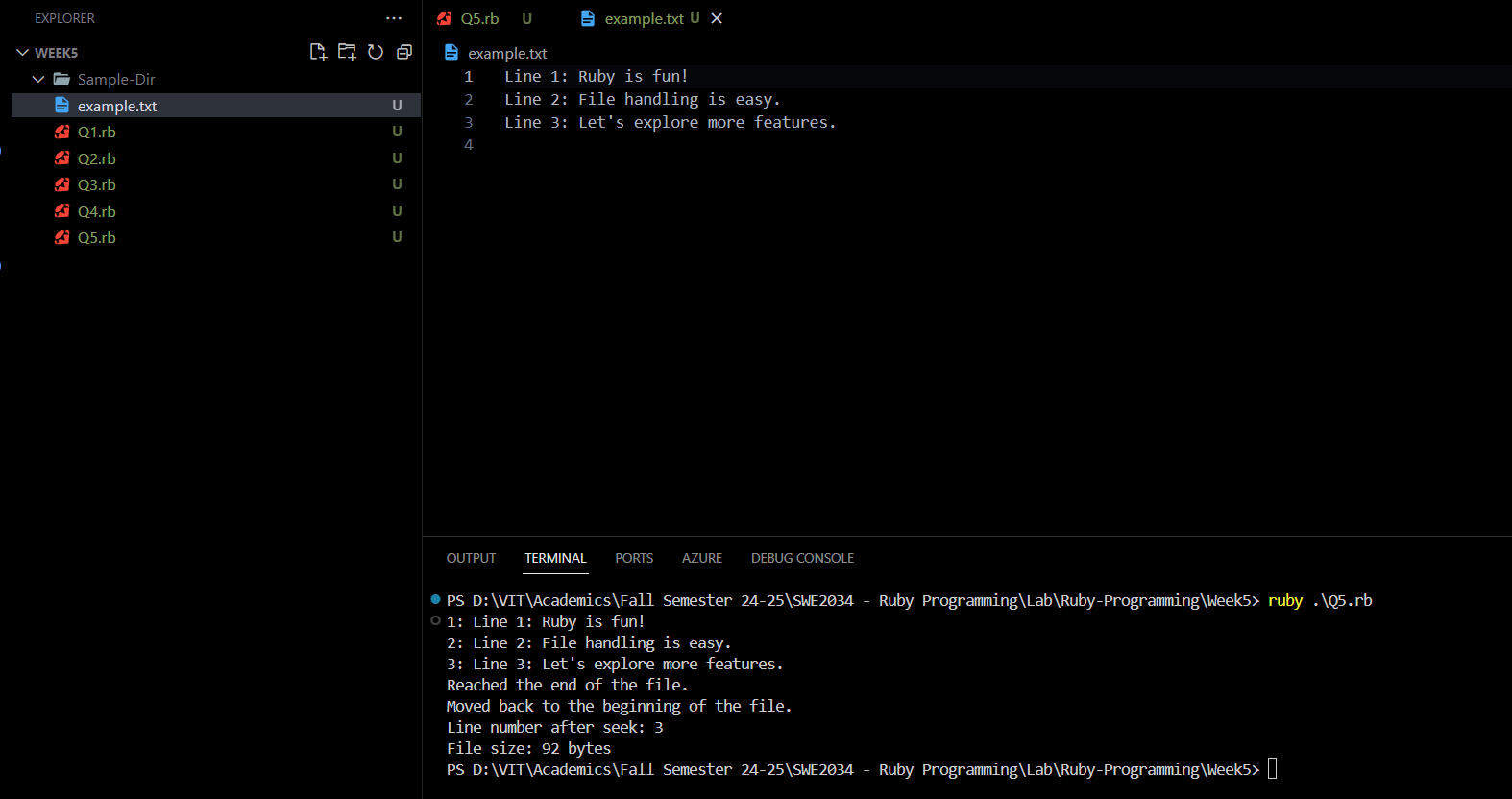
    puts "Line number after seek: #{file.lineno}"

    file\_size = File.size("example.txt")

    puts "File size: #{file\_size} bytes"

  end

Output:



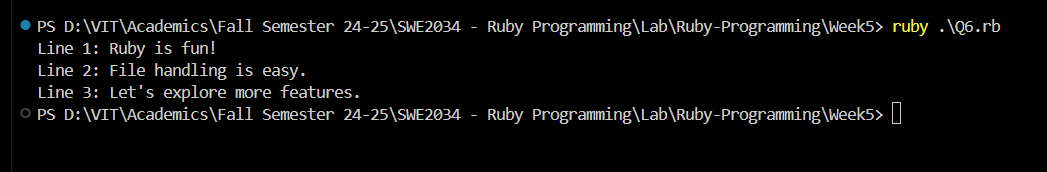
1. Read from a file in Ruby

Code:

content = File.read("example.txt")

puts content

Output:

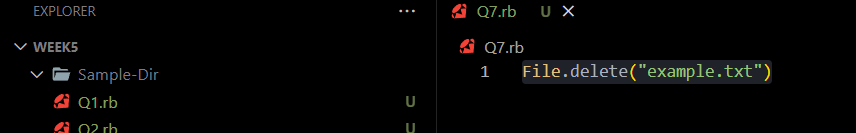


1. Delete a file in Ruby

Code:

File.delete("example.txt")

Output:



File Deleted.

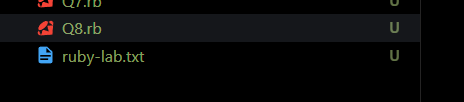
1. Rename a file in Ruby

Code:

File.rename("sample.txt", "ruby-lab.txt")

Output:  
Before:  


After:



1. Read a CSV and print specific rows and columns

Code:

require 'csv'

CSV.foreach("sample.csv", headers: true) do |row|

  if row["ID"] == "1"

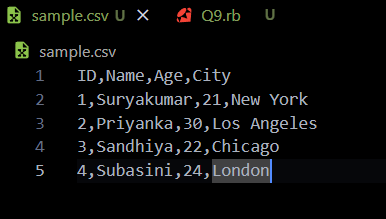
    puts "Row 1: #{row["Name"]}, #{row["Age"]}, #{row["City"]}"

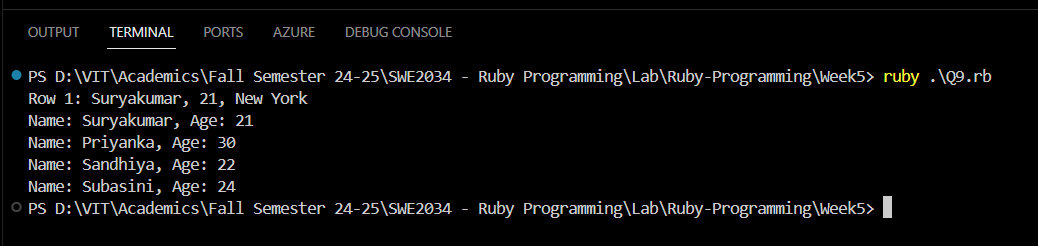
  end

  puts "Name: #{row['Name']}, Age: #{row['Age']}"

end

CSV:



Output:  


1. File Splitting and Joining: Imagine you have a large file, such as a video or a database backup, that you need to transfer or store on multiple devices. However, transferring or storing the entire file at once may not be feasible due to limitations in file size or storage capacity. In this scenario, you can use a program that splits the large file into smaller parts and joins them back together when needed. How can you implement such a program using Ruby?

Code:  
split.rb

def split\_file(file\_path, part\_size\_in\_mb)

    part\_size = part\_size\_in\_mb \* 1024 \* 1024

    file = File.open(file\_path, "rb")

    file\_size = File.size(file\_path)

    part\_number = 1

    while !file.eof?

      part\_file\_name = "#{file\_path}.part#{part\_number}"

      File.open(part\_file\_name, "wb") do |part\_file|

        part\_file.write(file.read(part\_size))

      end

      puts "Created: #{part\_file\_name}"

      part\_number += 1

    end

    file.close

  end

  split\_file("large\_file.dat", 10)

join.rb

def join\_files(output\_file, part\_file\_pattern)

    output = File.open(output\_file, "wb")

    part\_number = 1

    loop do

      part\_file\_name = part\_file\_pattern % part\_number

      break unless File.exist?(part\_file\_name)

      File.open(part\_file\_name, "rb") do |part\_file|

        output.write(part\_file.read)

      end

      puts "Joined: #{part\_file\_name}"

      part\_number += 1

    end

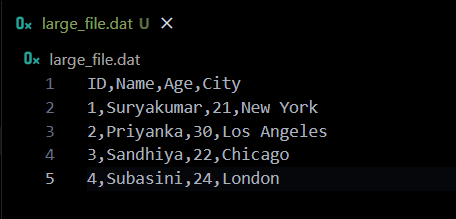
    output.close

  end

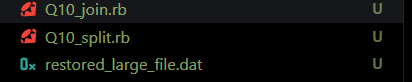
  join\_files("restored\_large\_file.dat", "large\_file.dat.part%d")

Output:

File Splitting:

File Merging:





1. File Backup and Restore: Imagine you have important files or directories on your computer that you want to protect against data loss due to hardware failure, malware, or accidental deletion. In this scenario, you can use a program that creates a backup of the files or directories and restores them later if needed. How can you implement such a program using Ruby?

Code:  
backup.rb

require 'fileutils'

def backup(source, destination)

  if File.exist?(source) || Dir.exist?(source)

    FileUtils.mkdir\_p(destination)

    FileUtils.cp\_r(source, destination)

    puts "Backup of '#{source}' created at '#{destination}'."

  else

    puts "Source '#{source}' does not exist."

  end

end

backup("important\_file.txt", "backup\_folder/important\_file\_backup")

backup("important\_directory", "backup\_folder/important\_directory\_backup")

restore.rb

require 'fileutils'

def restore(backup\_source, restore\_destination)

    if File.exist?(backup\_source) || Dir.exist?(backup\_source)

      FileUtils.cp\_r(backup\_source, restore\_destination)

      puts "Restored from '#{backup\_source}' to '#{restore\_destination}'."

    else

      puts "Backup source '#{backup\_source}' does not exist."

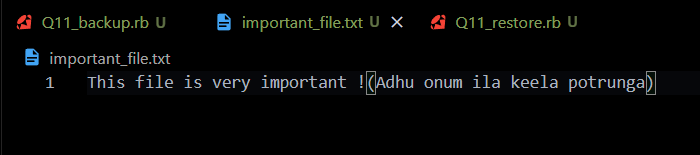
    end

  end

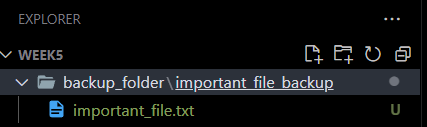
  restore("backup\_folder/important\_file\_backup", "restored\_important\_file.txt")

  restore("backup\_folder/important\_directory\_backup", "restored\_directory")

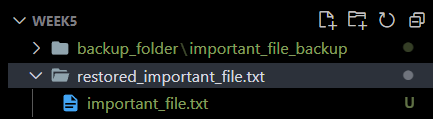
Output:

Important Text File

Folder gets created:



Restored:



1. File Renaming and Copying: Imagine you have a file that you want to rename or copy to a different location, optionally with a new name. In this scenario, you can use a program that allows you to perform these operations easily and efficiently. How can you implement such a program using Ruby?

Code:  
require 'fileutils'

def rename\_file(old\_name, new\_name)

  if File.exist?(old\_name)

    File.rename(old\_name, new\_name)

    puts "File renamed from '#{old\_name}' to '#{new\_name}'."

  else

    puts "File '#{old\_name}' does not exist."

  end

end

def copy\_file(source, destination)

  if File.exist?(source)

    FileUtils.cp(source, destination)

    puts "File '#{source}' copied to '#{destination}'."

  else

    puts "Source file '#{source}' does not exist."

  end

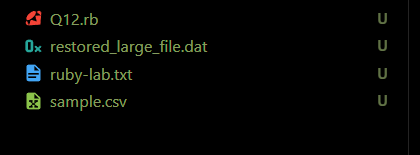
end

rename\_file("ruby-lab.txt", "new\_renamed\_file.txt")

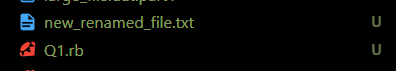
copy\_file("new\_renamed\_file.txt", "backup\_folder/copied\_file.txt")

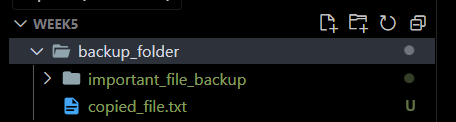
Output:

Initial Name:



Renamed file:



Copied to other location:  


1. File Searching and Filtering: Imagine you have a large collection of files on your computer and you want to find specific files based on their name, extension, size, or other criteria. In this scenario, you can use a program that searches for files matching a pattern or filters files based on certain criteria. How can you implement such a program using Ruby?

Code:

require 'find'

def search\_files(directory, name\_pattern = nil, extension = nil, min\_size = 0)

  matching\_files = []

  Find.find(directory) do |path|

    if File.file?(path)

      if name\_pattern.nil? || File.basename(path).include?(name\_pattern)

        if extension.nil? || File.extname(path) == extension

          if File.size(path) >= min\_size

            matching\_files << path

          end

        end

      end

    end

  end

  matching\_files

end

directory\_path = "backup\_folder"

txt\_files = search\_files(directory\_path, nil, '.txt')

puts "Found .txt files: #{txt\_files}"

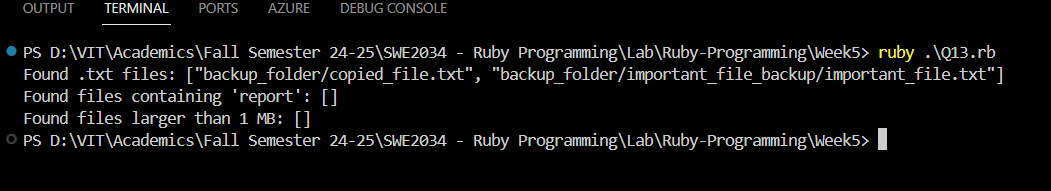
report\_files = search\_files(directory\_path, 'report')

puts "Found files containing 'report': #{report\_files}"

large\_files = search\_files(directory\_path, nil, nil, 1024 \* 1024)

puts "Found files larger than 1 MB: #{large\_files}"

Output:



1. File Sorting and Merging: Imagine you have multiple files that contain data that needs to be combined into a single file, such as log files from different servers or reports from multiple departments. In this scenario, you can use a program that sorts the files based on certain criteria and merges them into a single file. How can you implement such a program using Ruby?

Code:

require 'fileutils'

def merge\_files(file\_paths, output\_file, sort\_by = nil)

  merged\_content = []

  file\_paths.each do |file\_path|

    if File.exist?(file\_path)

      File.foreach(file\_path) do |line|

        merged\_content << line.chomp

      end

    else

      puts "File '#{file\_path}' does not exist."

    end

  end

  merged\_content.sort! { |a, b| sort\_by ? a.send(sort\_by) <=> b.send(sort\_by) : a <=> b }

  File.open(output\_file, 'w') do |file|

    merged\_content.each { |line| file.puts(line) }

  end

  puts "Merged content written to '#{output\_file}'."

end

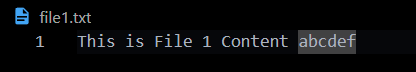
file\_list = ["file1.txt", "file2.txt", "file3.txt"]

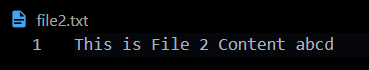
output\_file = "merged\_output.txt"

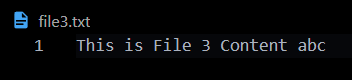
merge\_files(file\_list, output\_file)

merge\_files(file\_list, "sorted\_output.txt")

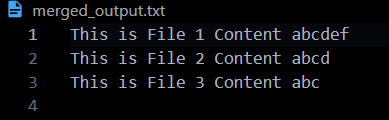
merge\_files(file\_list, "length\_sorted\_output.txt", :length)

Output:  






Merged:



Sorted:

