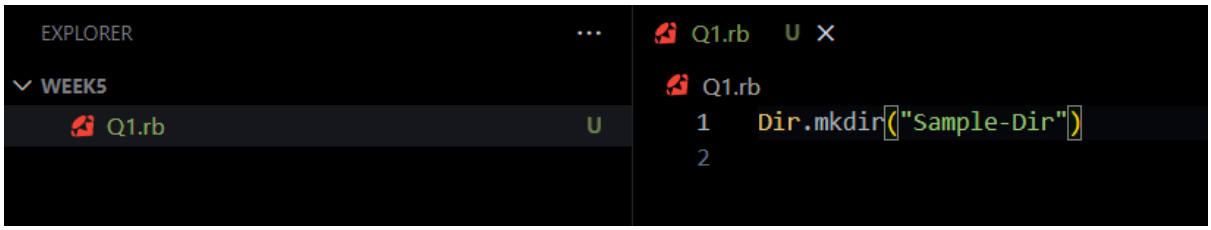
Ruby Lab Assessment 4

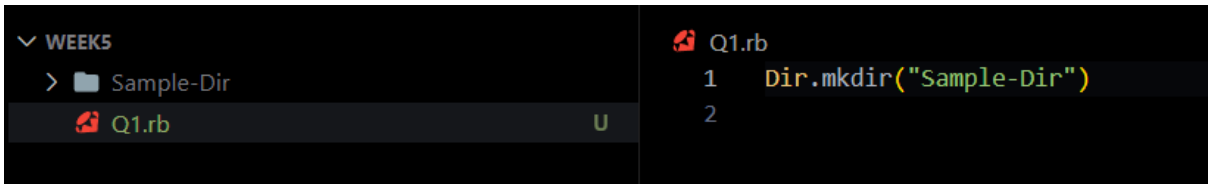
Suryakumar P 21MIS1146

1. Create a directory in Ruby

Before Creating:



After:

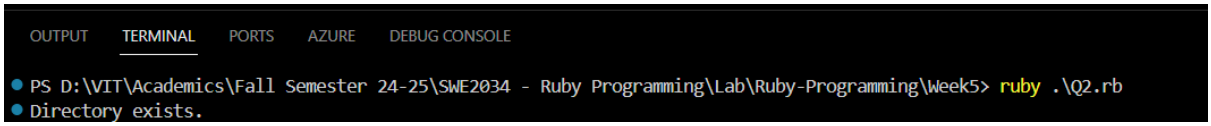


2. Check if a directory exists in Ruby

Code:

```
if Dir.exist?("Sample-Dir")
    puts "Directory exists."
  else
    puts "Directory does not exist."
  end
```

Output:

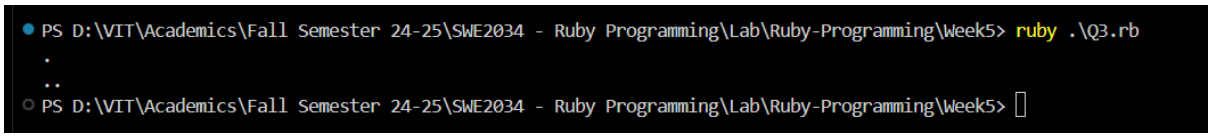


3. List files in a directory using Ruby

Code:

```
files = Dir.entries("Sample-Dir")
files.each { |file| puts file }
```

Output: (No Files inside)

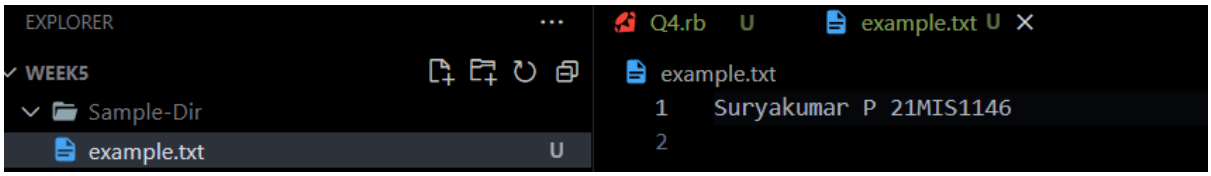


4. Create a file in Ruby

Code:

```
File.open("example.txt", "w") do |file|
    file.puts "Suryakumar P 21MIS1146"
  end
```

Output:

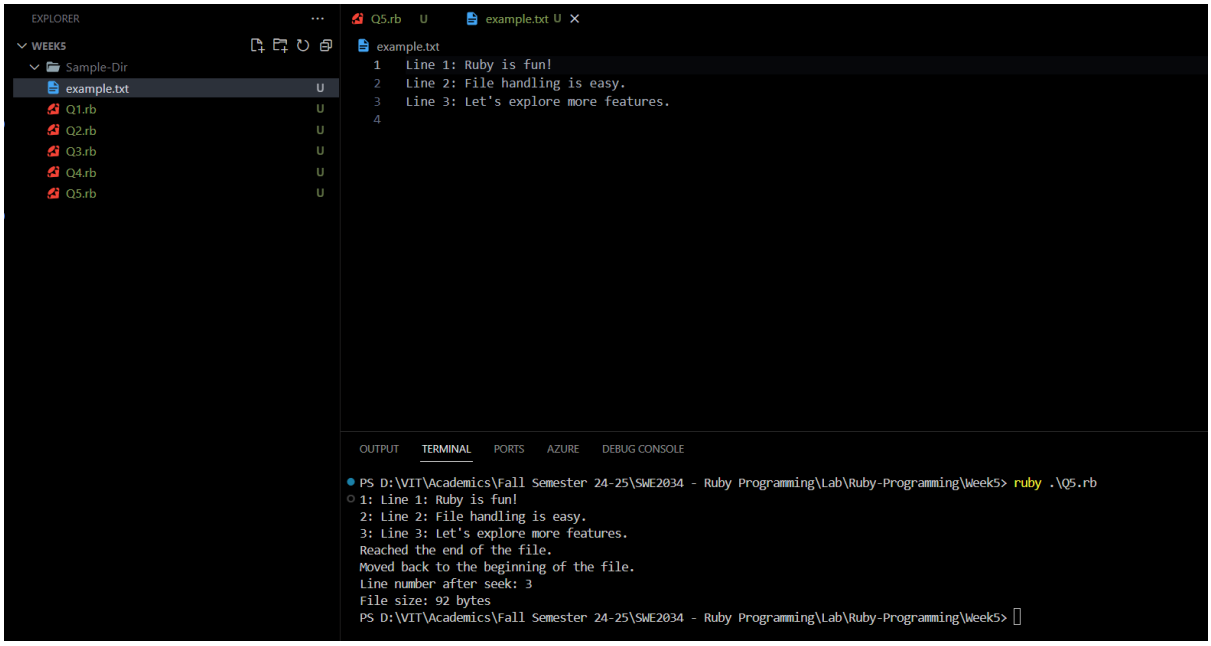


5. Write to a file in Ruby and use various methods like seek ( ), lineno( ), eof( ), size( )

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:



6. Read from a file in Ruby

Code:

```ruby
content = File.read("example.txt")
puts content
```

Output:

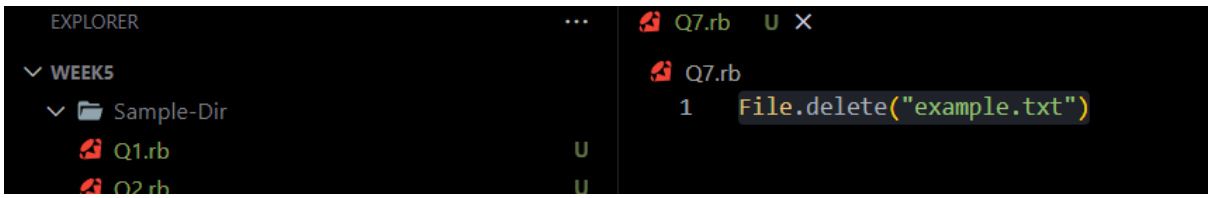
```
PS D:\VIT\Academics\Fall Semester 24-25\SWE2034 - Ruby Programming\Lab\Ruby-Programming\Week5> ruby .\Q6.rb
Line 1: Ruby is fun!
Line 2: File handling is easy.
Line 3: Let's explore more features.
PS D:\VIT\Academics\Fall Semester 24-25\SWE2034 - Ruby Programming\Lab\Ruby-Programming\Week5>
```

7. Delete a file in Ruby

Code:

```ruby
File.delete("example.txt")
```

Output:



File Deleted.

8. Rename a file in Ruby

Code:

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

Output:

Before:

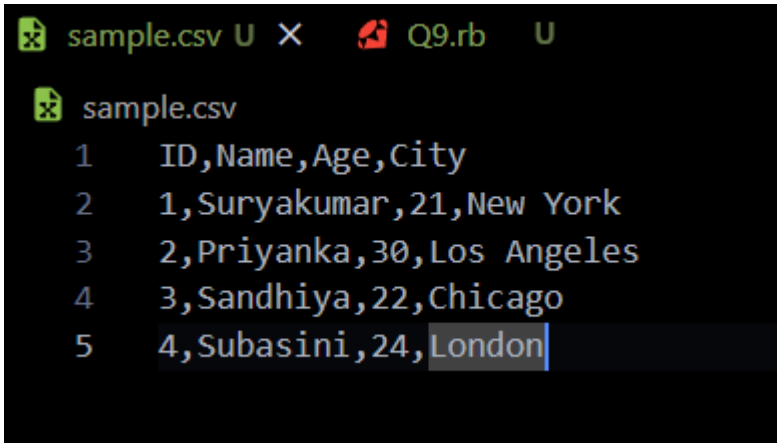
After:

9. 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:

```
OUTPUT    TERMINAL    PORTS    AZURE    DEBUG CONSOLE

PS D:\VIT\Academics\Fall Semester 24-25\SWE2034 - Ruby Programming\Lab\Ruby-Programming\Week5> ruby .\Q9.rb
Row 1: Suryakumar, 21, New York
Name: Suryakumar, Age: 21
Name: Priyanka, Age: 30
Name: Sandhiya, Age: 22
Name: Subasini, Age: 24
PS D:\VIT\Academics\Fall Semester 24-25\SWE2034 - Ruby Programming\Lab\Ruby-Programming\Week5>
```

10. 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

```ruby
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

```ruby
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:

large_file.dat U
large_file.dat.part1 U

large_file.dat U

large_file.dat

```
ID,Name,Age,City
1,Suryakumar,21,New York
2,Priyanka,30,Los Angeles
3,Sandhiya,22,Chicago
4,Subasini,24,London
```

File Merging:

Q10_join.rb U
Q10_split.rb U
restored_large_file.dat U

large_file.dat U restored_large_file.dat U

restored_large_file.dat

```
ID,Name,Age,City
1,Suryakumar,21,New York
2,Priyanka,30,Los Angeles
3,Sandhiya,22,Chicago
4,Subasini,24,London
```

11. 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

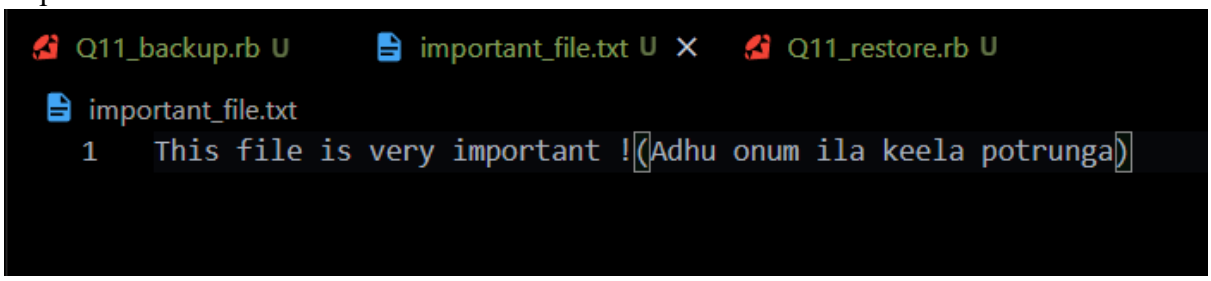
```
require 'fileutils'

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

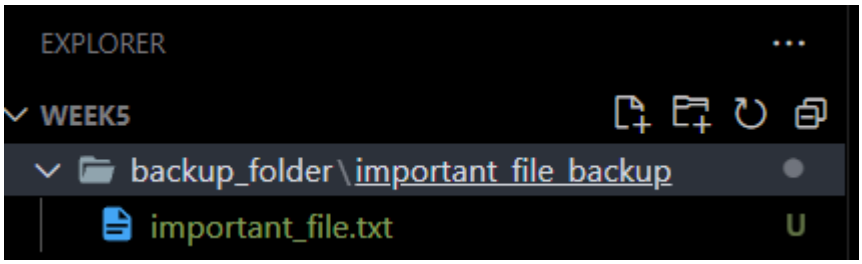
```
require 'fileutils'

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:

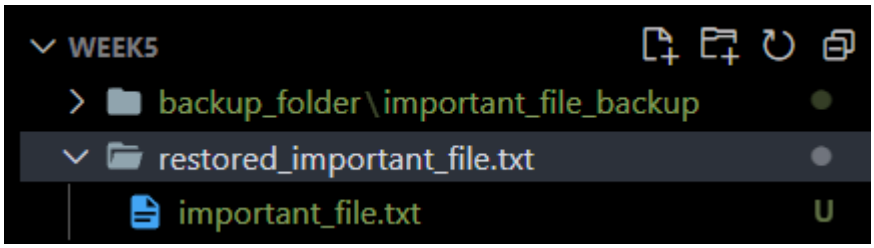
Important Text File



Folder gets created:



Restored:



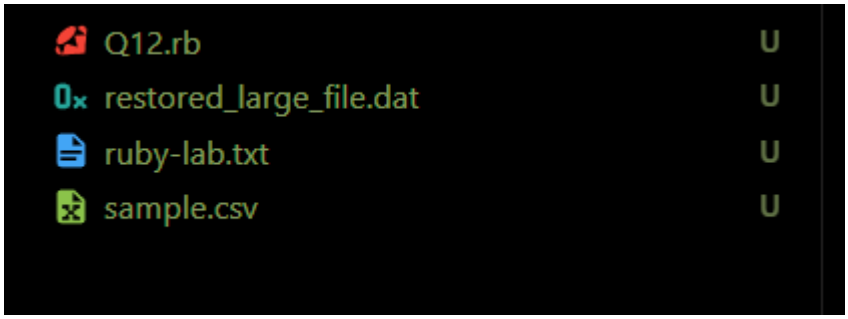
12. File Renaming and Copying: Imagine you have a file that you want to rename or copy to a different location, optionally with a new name. In this scenario, you can use a program that allows you to perform these operations easily and efficiently. How can you implement such a program using Ruby?

```
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)
```

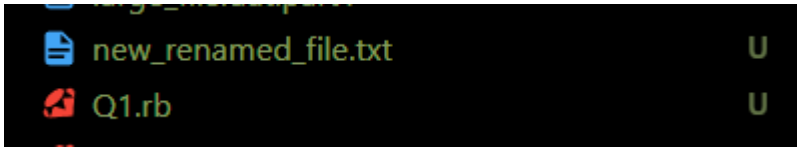
```
    puts "File '#{source}' copied to '#{destination}'."
  else
    puts "Source file '#{source}' does not exist."
  end
end

rename_file("ruby-lab.txt", "new_renamed_file.txt")
copy_file("new_renamed_file.txt", "backup_folder/copied_file.txt")
```

Output:

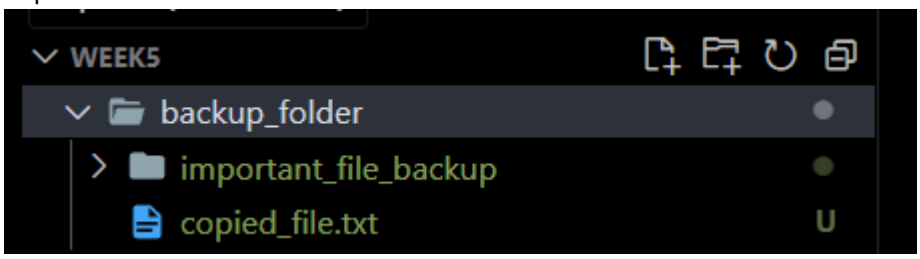
Initial Name:



Renamed file:



Copied to other location:



13. 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:

```ruby
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:

```
OUTPUT    TERMINAL    PORTS    AZURE    DEBUG CONSOLE

PS D:\VIT\Academics\Fall Semester 24-25\SWE2034 - Ruby Programming\Lab\Ruby-Programming\Week5> ruby .\Q13.rb
Found .txt files: ["backup_folder/copied_file.txt", "backup_folder/important_file_backup/important_file.txt"]
Found files containing 'report': []
Found files larger than 1 MB: []
PS D:\VIT\Academics\Fall Semester 24-25\SWE2034 - Ruby Programming\Lab\Ruby-Programming\Week5>
```

14. 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:

file1.txt

```
1    This is File 1 Content abcdef
```

file2.txt

```
1    This is File 2 Content abcd
```

file3.txt

```
This is File 3 Content abc
```

Merged:

merged_output.txt

```
This is File 1 Content abcdef
This is File 2 Content abcd
This is File 3 Content abc
```

Sorted:

length_sorted_output.txt

```
This is File 3 Content abc
This is File 2 Content abcd
This is File 1 Content abcdef
```