Assignment Date : 09-08-2025

Name: Lalit joshi

Email : [joshilalit2275@gmail.com](mailto:joshilalit2275@gmail.com)

TOPIC 1: Input, Output & Formatting in PowerShell

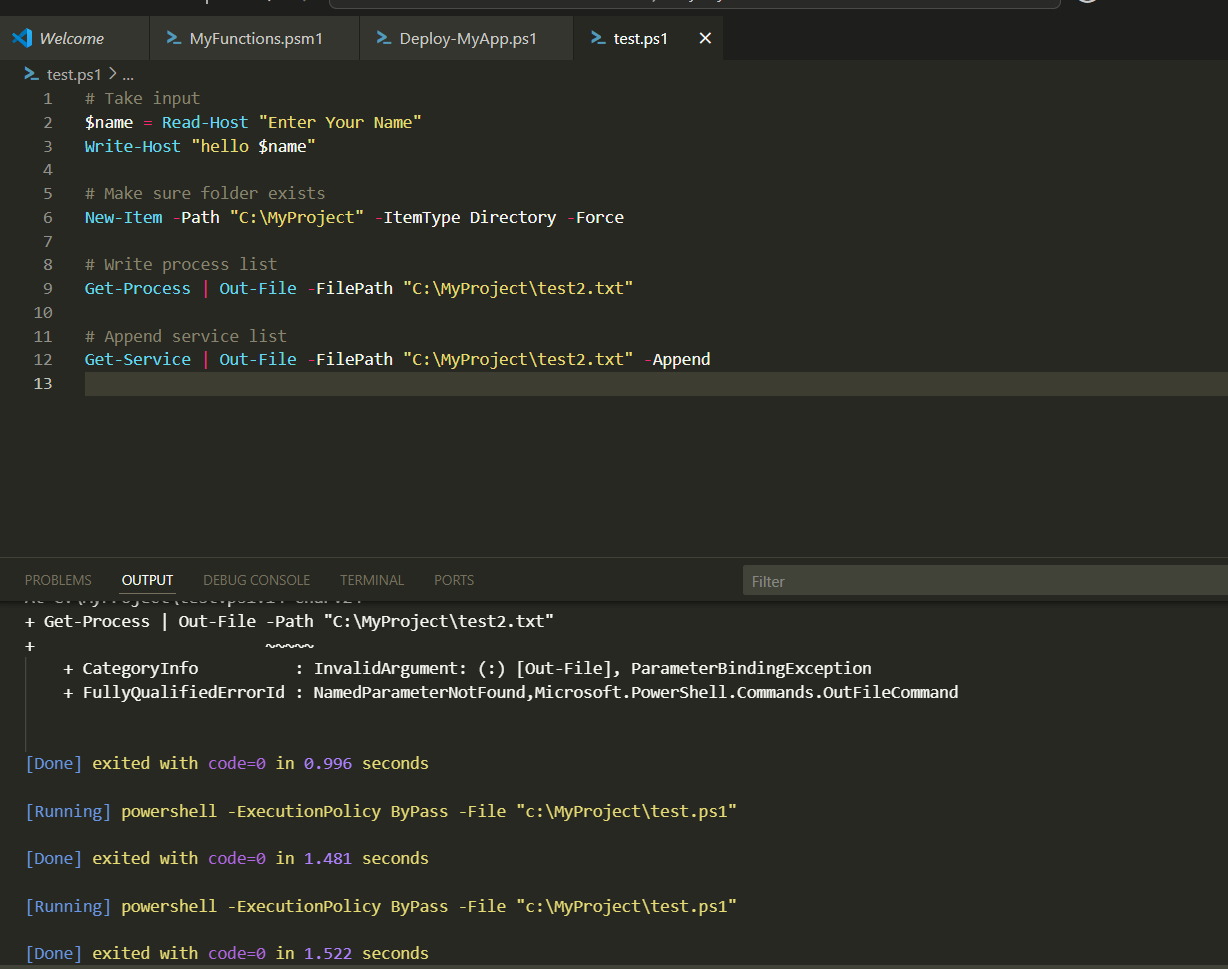
**1. Input**

* PowerShell works with **objects**, not plain text.
* **Sources of Input**:
  1. **Interactive** → Read-Host, GUI prompts.
  2. **Pipeline Input** → Data passed from previous command using |.
  3. **Script Parameters** → param() block or -ParameterName in cmdlets.
  4. **Files** → Get-Content to read data.
* Use $\_ to refer to the **current pipeline object**.

**2. Output**

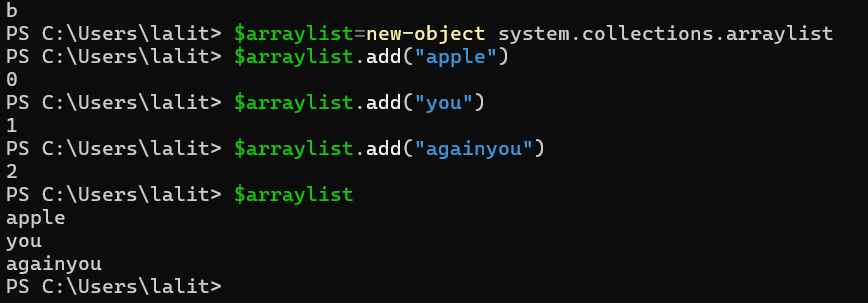
* Every command outputs **objects** into the pipeline.
* If another command follows, objects pass **as objects**.
* If it’s the last command, PowerShell **formats and displays** it as text.
* Output destinations:
  + **Screen** (default)
  + **File** (Out-File, Export-Csv, Set-Content)
  + **Variable** (assignment =)

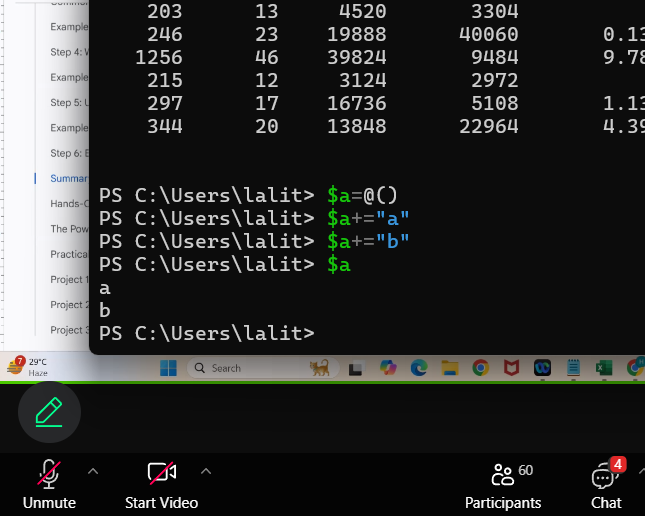
**3. Formatting**

* **Purpose**: Shape data for display, not for processing.
* Main cmdlets:
  + Format-Table → Table view.
  + Format-List → Property list view.
  + Format-Wide → Only one property in wide columns.
* **Rule**: Apply formatting only at the end of the pipeline — after processing/filtering.
* Once formatted, data becomes **text only**, not objects.

Topic 2: Objects, Arrays, Variables in PowerShell

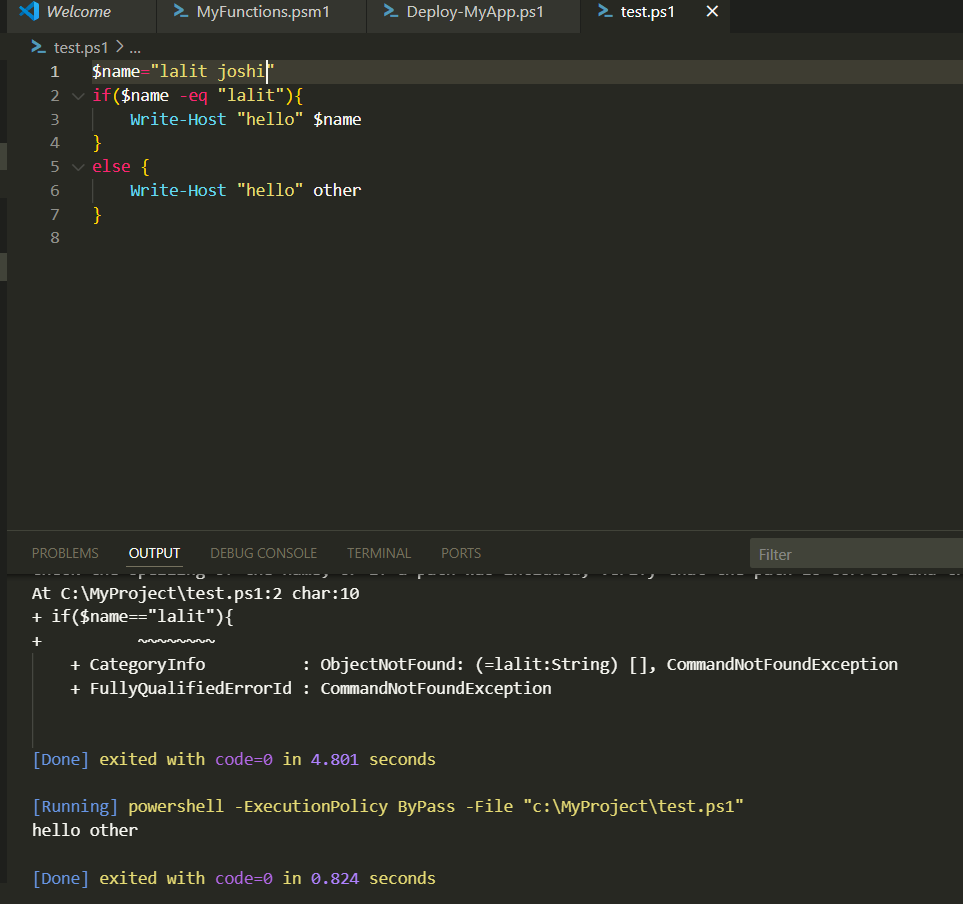
* **Objects**:  
  Everything in PowerShell is an object. Objects have properties (data) and methods (actions). They can be passed through the pipeline between commands.
* **Arrays**:  
  A collection of multiple items stored in a single variable. Can contain any type of data, including mixed types. Supports indexing and has a fixed order.
* **Variables**:  
  Named storage for data, prefixed with $. Can store any type of object, array, or value. Variables exist in memory until removed or session ends.





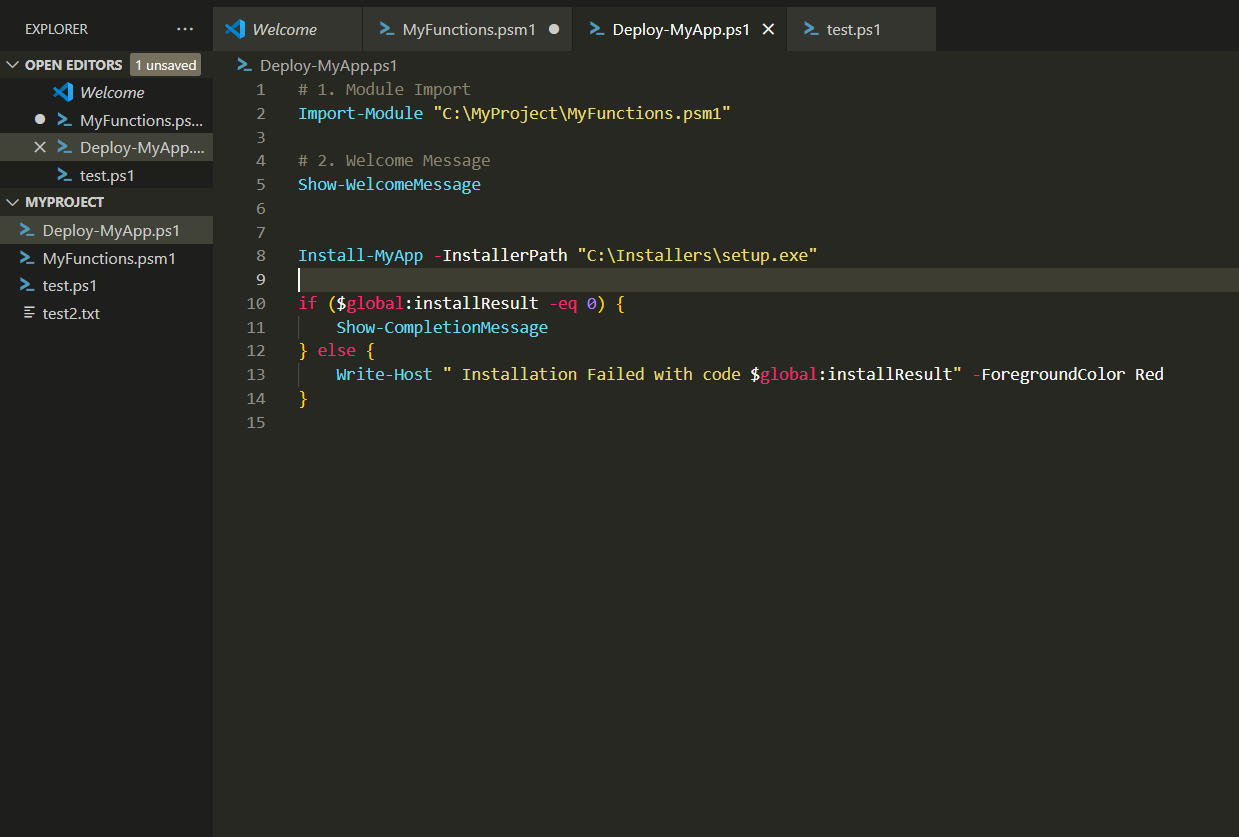
Topic 3: Scripting Constructs in PowerShell

Scripting constructs in PowerShell are the fundamental building blocks used to control the flow, structure, and logic of scripts. They include conditional statements like if, elseif, else, and switch for decision-making based on conditions, and loops such as for, foreach, while, do...while, and do...until for repeating tasks. Functions allow grouping reusable code blocks with optional parameters, while script blocks (enclosed in {}) can hold multiple commands or expressions. Error handling is managed through try, catch, and finally blocks, and loop control keywords like break and continue adjust execution flow. The pipeline enables passing objects from one command to another, creating efficient and readable command sequences. Together, these constructs form the foundation for writing structured, maintainable, and flexible PowerShell scripts.



Topic 4 : Modularization

Modularization in PowerShell is the practice of breaking large scripts into smaller, reusable, and maintainable units called **modules**. A module can contain functions, variables, aliases, and other resources that can be imported and used in different scripts. This approach improves code organization, reusability, and readability while reducing duplication. PowerShell supports different types of modules such as **script modules** (.psm1 files written in PowerShell), **binary modules** (.dll files compiled from languages like C#), and **manifest modules** (.psd1 files containing metadata about the

module). Modules can be loaded automatically from standard module paths or manually using Import-Module, and their commands can be discovered using Get-Command or Get-Module.

