PowerShell Arrays

An **array** is a list of items stored in a single variable.

Creating Arrays

Method 1: Using @()

```
$numbers = @(1, 2, 3, 4)
```

Method 2: Without @ (PowerShell auto-creates array)

```
$names = "Ram", "Shyam", "Mohan"
```

Accessing Array Elements

Index starts at **0**.

```
$names[0]   # first element
$names[2]   # third element
```

Adding Items to Array

```
$numbers += 5
```

Array Length

```
$names.Length
```

---

5 Looping Through an Array

```
foreach ($item in $names) {
    Write-Output $item
}
```

---

6 Mixed-Type Array

Arrays can store different types:

```
$data = @(10, "hello", 3.14, $true)
```

---

7 Array of Objects

```
$students = @(
    @{name="Amit"; age=20},
    @{name="Riya"; age=22}
)
```

```
$students[0].name   # Access Amit
```

IMPORT in PowerShell (Importing Data)

Import means *bringing data into PowerShell* from a file.

You use import when you want to **read data** from:

- CSV file
- JSON file
- XML file
- Modules
- Scripts

Common Import Commands

a) Import CSV File

```
Import-Csv "C:\data.csv"
```

It reads the CSV file and converts it into table-like objects.

b) Import JSON File

```
Get-Content "info.json" | ConvertFrom-Json
```

c) Import Module

```
Import-Module ActiveDirectory
```

Loads extra PowerShell commands.

d) Import Data From File

```
Get-Content "myfile.txt"
```

Import Summary (Easy English)

- **Import = bring data or commands into PowerShell.**
- You use it when you want to **read** from files or load extra modules.

EXPORT in PowerShell (Sending Data Out)

Export means *saving data from PowerShell into a file.*

You use export when you want to **save / store** output:

- CSV file
- JSON file

- XML file
- Text file

Common Export Commands

a) Export to CSV

```
Get-Process | Export-Csv "processes.csv" -NoTypeInformation
```

b) Export to JSON

```
Get-Service | ConvertTo-Json | Out-File "services.json"
```

c) Export to Text File

```
Get-Date | Out-File "date.txt"
```

Export Summary (Easy English)

- **Export = save PowerShell output to a file.**
- You use it when you want to **store** or **share** results.

IMPORT vs EXPORT (Quick Table)

Feature	Import	Export
Meaning	Bring data *into* PowerShell	Send data *out* of PowerShell
Direction	File → PowerShell	PowerShell → File
Used For	Reading files	Saving data
Example	`Import-Csv`	`Export-Csv`

PowerShell vs CMD — Easy Comparison

Feature	PowerShell	CMD (Command Prompt)
Type	Advanced shell + scripting language	Basic command-line shell
Commands	Uses *cmdlets* like `Get-Process`	Uses simple commands like `dir`, `copy`
Objects	Outputs **objects**	Outputs **plain text**
Scripting	Full scripting language (.ps1)	Very limited (.bat)
Automation	Strong automation support	Very weak
Pipelines	Passes **objects**	Passes **text only**
Remote management	Built-in (PowerShell Remoting)	Not available
Use cases	Admin tasks, automation, cloud, servers	Simple tasks like file copy, directory listing

PowerShell Example

```
Get-Process | Where-Object {$_.CPU -gt 10}
```

CMD Example

```
tasklist
```

PowerShell vs CMD

- **PowerShell is modern, powerful, object-based, and supports automation.**
- **CMD is older, simple, text-based, and limited.**
- Use **PowerShell** for scripting, admin tasks, and automation.
- Use **CMD** only for basic commands.

Command Line Calculator in PowerShell

This calculator will:

- Ask user for **first number**
- Ask for **operator** (`+ - * /`)
- Ask for **second number**
- Perform the calculation
- Show the result

Simple Calculator Script (Basic Version)

```
# Command Line Calculator

Write-Host "----- PowerShell Calculator -----"

# Take first number
$number1 = Read-Host "Enter first number"

# Take operator
$operator = Read-Host "Enter operator (+, -, *, /)"

# Take second number
$number2 = Read-Host "Enter second number"

# Convert input to numbers
$number1 = [double]$number1
$number2 = [double]$number2

# Perform calculation
switch ($operator) {
    "+" { $result = $number1 + $number2 }
    "-" { $result = $number1 - $number2 }
    "*" { $result = $number1 * $number2 }
    "/" {
        if ($number2 -eq 0) {
            Write-Host "Error: Cannot divide by zero."
            exit
        }
```

```
        $result = $number1 / $number2
    }
    default { Write-Host "Invalid operator."; exit }
}

# Show result
Write-Host "`nResult: $number1 $operator $number2 = $result"
```

How to Run

1. Save file as:
   calculator.ps1
2. Open PowerShell
3. Run this command:
4. `.\calculator.ps1`

Improved Calculator (Menu-Based)

```
Write-Host "====== PowerShell Calculator ======"

$number1 = Read-Host "Enter the first number"
$number2 = Read-Host "Enter the second number"

$number1 = [double]$number1
$number2 = [double]$number2

Write-Host "`nChoose operation:"
Write-Host "1. Add"
Write-Host "2. Subtract"
Write-Host "3. Multiply"
Write-Host "4. Divide"

$choice = Read-Host "Enter your choice (1-4)"

switch ($choice) {
    "1" { $result = $number1 + $number2 }
    "2" { $result = $number1 - $number2 }
    "3" { $result = $number1 * $number2 }
    "4" {
        if ($number2 -eq 0) {
            Write-Host "`nError: Division by zero not allowed."
            exit
        }
        $result = $number1 / $number2
    }
    default {
        Write-Host "`nInvalid choice!"
        exit
    }
}

Write-Host "`nResult = $result"
```

Assignment to enhance functionality

If you want a more advanced calculator, I can help you add:

Square, Square Root
Power function
Continue calculation without exiting

Project 1: System Information Report Generator

Collects system info and saves it into a file automatically

What it does

- Collects CPU, RAM, OS info
- Saves it to a `.txt` or `.html` report
- Good for IT students

Code

```
$report = @()
$report += "===== SYSTEM INFORMATION REPORT ====="
$report += "Computer Name: $(hostname)"
$report += "OS Version: $((Get-CimInstance Win32_OperatingSystem).Caption)"
$report += "CPU: $((Get-CimInstance Win32_Processor).Name)"
$report += "RAM (GB): $([math]::Round((Get-CimInstance
Win32_ComputerSystem).TotalPhysicalMemory / 1GB,2))"
$report += "Disk Space: $(Get-PSDrive C | Select-Object -ExpandProperty
Free)"

$outputFile = "C:\PowerShell_SystemReport.txt"
$report | Out-File $outputFile

Write-Host "Report created at: $outputFile"
```

Project 2: Folder Backup Script

Backup important folders automatically

What it does

- Copies a folder to a backup location
- Good practice for file system handling

Code

```
$source = "D:\Cadd Mentor\Student_Class_Code\Sandeep_C
Language_Classes\SabaShann"
$backup = "D:\Backup\MyFolder_$(Get-Date -Format yyyyMMdd)"

Copy-Item $source -Destination $backup -Recurse

Write-Host "Backup completed. Saved at $backup"
```

Project 3: User Account Creation Automation (Simulated)

Create multiple user accounts from a CSV file

CSV File (users.csv)

```
Username,FullName
user1,John Doe
user2,Mary Smith
user3,David Bill
```

PowerShell Script

```
$users = Import-Csv "C:\users.csv"

foreach ($u in $users) {
    Write-Host "Creating user: $($u.Username)"
    # Simulation (real: New-LocalUser)
}
```

Project 4: Network Connectivity Checker

Checks if websites/servers are reachable

What it does

- Pings servers
- Shows online/offline status

Code

```
$servers = @("google.com", "github.com", "bing.com")

foreach ($s in $servers) {
    if (Test-Connection -ComputerName $s -Count 1 -Quiet) {
        Write-Host "$s is Online!" -ForegroundColor Green
    } else {
        Write-Host "$s is Offline!" -ForegroundColor Red
    }
}
```

Project 5: Startup Program Manager

Shows and removes Windows startup programs

Code

```
$startup = "$env:APPDATA\Microsoft\Windows\Start Menu\Programs\Startup"

Write-Host "Startup Programs:"
Get-ChildItem $startup

# Remove unwanted startup program
# Remove-Item "$startup\app.lnk"
```

Project 6: Bulk File Renamer

Rename 100s of files automatically

**Example: add prefix "IMG_" to all files*

```
$path = "D:\Backup\MyFolder_20251124"

$i = 1
foreach ($file in Get-ChildItem $path) {
    $newName = "IMG_$i$($file.Extension)"
    Rename-Item $file.FullName $newName
    $i++
}
```

Bonus Projects

Mini Password Generator

```
$length = 10
$chars =
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!@#$%^&*"

$password = -join ((1..$length) | ForEach-Object { $chars[(Get-Random -
Minimum 0 -Maximum $chars.Length)] })

Write-Host "Generated Password: $password"
```