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
```

Looping Through an Array

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

Mixed-Type Array

Arrays can store different types:

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

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"
```

GUI Based Password Generator

```
Add-Type -AssemblyName System.Windows.Forms
Add-Type -AssemblyName System.Drawing

# --- Window ---
$form = New-Object System.Windows.Forms.Form
$form.Text = "Password Generator"
$form.Size = New-Object System.Drawing.Size(400, 250)
$form.StartPosition = "CenterScreen"

# --- Label for Length ---
$label = New-Object System.Windows.Forms.Label
$label.Text = "Password Length:"
$label.Location = New-Object System.Drawing.Point(20, 20)
$label.AutoSize = $true
$form.Controls.Add($label)

# --- Numeric Input (Length) ---
$lengthBox = New-Object System.Windows.Forms.NumericUpDown
$lengthBox.Location = New-Object System.Drawing.Point(150, 18)
$lengthBox.Minimum = 4
$lengthBox.Maximum = 50
$lengthBox.Value = 12
$form.Controls.Add($lengthBox)

# --- Textbox to show password ---
$outputBox = New-Object System.Windows.Forms.TextBox
$outputBox.Location = New-Object System.Drawing.Point(20, 70)
$outputBox.Size = New-Object System.Drawing.Size(340, 30)
$outputBox.ReadOnly = $true
$form.Controls.Add($outputBox)

# --- Generate Button ---
```

```
$generateButton = New-Object System.Windows.Forms.Button
$generateButton.Text = "Generate Password"
$generateButton.Location = New-Object System.Drawing.Point(20, 120)
$generateButton.Size = New-Object System.Drawing.Size(150, 30)
$form.Controls.Add($generateButton)

# --- Copy Button ---
$copyButton = New-Object System.Windows.Forms.Button
$copyButton.Text = "Copy"
$copyButton.Location = New-Object System.Drawing.Point(210, 120)
$copyButton.Size = New-Object System.Drawing.Size(80, 30)
$form.Controls.Add($copyButton)

# --- Password Generation Logic ---
$generateButton.Add_Click({
    $length = [int]$lengthBox.Value
    $chars =
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!@#$%^&*"
    $password = -join ((1..$length) | ForEach-Object {
        $chars[(Get-Random -Minimum 0 -Maximum $chars.Length)]
    })
    $outputBox.Text = $password
})

# --- Copy to clipboard ---
$copyButton.Add_Click({
    Set-Clipboard -Value $outputBox.Text
    [System.Windows.Forms.MessageBox]::Show("Password copied to
clipboard!")
})

# Show window
$form.ShowDialog()
```

1. Process and Thread Management

Process

A **process** is a program that is running in the system.
It has:

- Its own memory
- Its own resources (files, registers, variables)
- Independent execution

Example: Running Chrome, VLC, MS Word — all are separate processes.

Thread

A **thread** is a smaller part of a process.

- Threads share the same memory of the process.
- Multiple threads can run inside one process.

Example:
Chrome tabs → Each tab is a thread
Word auto-save → Runs in a separate thread

Process States

1. **New** – created
2. **Ready** – waiting for CPU
3. **Running** – using the CPU
4. **Waiting/Blocked** – waiting for I/O
5. **Terminated** – finished

Context Switching

The CPU switches from one process/thread to another.
It saves the previous process state and loads the next one.
More context switching → CPU overhead increases.

CPU Scheduling (Basic)

CPU scheduling decides **which process will run next on the CPU**.

Goals:

- Maximize CPU usage
- Reduce waiting time
- Improve response time
- Increase throughput (tasks completed per second)

Basic Scheduling Algorithms

1. **FCFS (First Come First Serve)**
    - The process that comes first runs first
    - Simple but slow for long processes
2. **SJF (Shortest Job First)**
    - Smallest job first
    - Fast but needs job time estimation
3. **Priority Scheduling**
    - Each process gets a priority
    - Higher priority process runs first
4. **Round Robin (RR)**
    - Each process gets fixed time (Time Quantum)
    - Good for multitasking systems

Memory Management & Virtual Memory

Memory Management

OS allocates and manages main memory (RAM).
It must:

- Load processes into RAM
- Prevent processes from using others' memory
- Optimize usage of available memory

Paging

Memory is divided into **fixed-size blocks**:

- **Frames** (in RAM)
- **Pages** (in processes)

No external fragmentation.

Segmentation

Memory divided based on **logical sections**:

- Code
- Data
- Stack

Reduces memory wastage.

Virtual Memory

Virtual memory allows programs to run even when RAM is full.
It uses **disk space** (swap file) as extra memory.

Key mechanism:

- **Demand Paging** — load pages when needed
- **Page Fault** — if a page is not in RAM, bring it from disk

File System & Permissions

File System

OS stores and organizes data in files and directories.

Responsibilities:

- File creation, deletion, update
- Directory structure
- Access permissions
- Disk space management

Common file systems:

- NTFS (Windows)
- ext4 (Linux)
- HFS+ / APFS (Mac)

File Permissions

Used to control who can access the file.

Linux/Unix has 3 permissions:

- **r (read)**
- **w (write)**
- **x (execute)**

3 types of users:

- **Owner**
- **Group**
- **Others**

Example: `rwxr-x--x`

I/O Management

I/O devices include:

- Keyboard, mouse
- Printer, scanner
- USB, Hard disk
- Network devices

Role of OS in I/O

- Communicate with hardware
- Manage device drivers
- Buffer management
- Interrupt handling

Device Drivers

Software that tells the OS *how to work with a device*.

Interrupt

A signal from a device → CPU stops current work → handles I/O → resumes work.

Virtualization & Hypervisors

Virtualization

Virtualization allows one physical machine to run multiple virtual machines (VMs).

Each VM behaves like a real computer.

Benefits:

- Resource efficiency
- Scalability
- Isolation
- Easy backup and recovery
- Reduced hardware cost

Hypervisor

Software that creates and manages Virtual Machines.

Two types:

Type 1 (Bare Metal)

Runs directly on hardware
Examples:

- VMware ESXi
- Microsoft Hyper-V
- Xen Server

Type 2 (Hosted)

Runs on an operating system
Examples:

- VMware Workstation
- VirtualBox

Resource Allocation in OS

Resource = CPU, memory, storage, network, I/O devices.

OS must distribute these resources fairly and efficiently.

Goals of Resource Allocation

- Avoid conflicts

- Maximize efficiency
- Prevent starvation
- Prevent deadlocks

Deadlock

When two or more processes wait for resources held by each other → system halts.
Deadlock conditions (must all be true):

1. Mutual Exclusion
2. Hold and Wait
3. No Preemption
4. Circular Wait

Resource Allocation Techniques

- Priority-based allocation
- Fair share scheduling
- Banker's Algorithm (avoids deadlock)
- Dynamic memory allocation

IP Addressing (IPv4, IPv6, Subnetting)

What is an IP Address?

An **IP Address** (Internet Protocol Address) is a unique number assigned to each device in a network.
It helps devices **identify**, **communicate**, and **send/receive data**.

Example:
`192.168.1.10`

IPv4 (Internet Protocol Version 4)

Features

- 32-bit address
- Written in dotted decimal format
- Example: `192.168.0.1`
- Total possible addresses: **4.3 billion**

Structure

IPv4 = 4 Octets → each 0–255
Example: `192.168.1.25`

Types of IPv4 Addresses

1. **Public IP** – Used on the Internet.
2. **Private IP** – Used inside local networks.
   Common private ranges:
   - 10.0.0.0 – 10.255.255.255
   - 172.16.0.0 – 172.31.255.255
   - 192.168.0.0 – 192.168.255.255
3. **Loopback IP** – `127.0.0.1`
4. **Broadcast IP** – For broadcasting to all devices.

IPv6 (Internet Protocol Version 6)

Features

- 128-bit address
- Written in hexadecimal
- Example:
  `2001:0db8:85a3:0000:0000:8a2e:0370:7334`
- Supports **3.4×10^{38}** addresses
- Designed to replace IPv4
- Better security & speed

Shortened IPv6 Example

`2001:db8:85a3::8a2e:370:7334`

Subnetting

What is Subnetting?

Subnetting divides a large network into **smaller sub-networks** to improve:

- Security
- Performance
- IP management

Key Terms

- **Subnet Mask**: Identifies network & host portion.
  Example: `255.255.255.0`
- **CIDR Notation**: `/24`, `/16`, `/8`

Example:
`192.168.1.0/24`
→ 256 total IPs
→ 254 usable IPs

Why Subnetting?

- Reduce network traffic

- Better control over routing
- Efficient usage of IP addresses

Protocols (HTTP, HTTPS, TCP/IP, DNS, DHCP, SSH, FTP)

HTTP (HyperText Transfer Protocol)

- Used for **web browsing**
- Transfers webpages (HTML, CSS, JS)
- Works on **Port 80**
- Not encrypted → data can be seen by attackers

HTTPS (HTTP Secure)

- Secure version of HTTP
- Uses **SSL/TLS** encryption
- Works on **Port 443**
- Protects login info, passwords, payments
- Used by all modern websites

TCP/IP (Transmission Control Protocol / Internet Protocol)

TCP/IP is the **foundation of the internet**.

TCP

- Ensures reliable data delivery
- Breaks data into packets
- Re-sends lost packets
- Connection-oriented

IP

- Routes the packets to the correct destination
- Connectionless

Together, TCP/IP enables communication between any two devices globally.

DNS (Domain Name System)

DNS = "Phonebook of the Internet"

It converts **domain names → IP addresses**

Example:
`www.google.com` → `142.250.193.78`

Why DNS?

Humans remember names, not numbers.

Works on **Port 53**.

DHCP (Dynamic Host Configuration Protocol)

DHCP automatically assigns:

- IP Address
- Subnet Mask
- Default Gateway
- DNS Server

Works on **Port 67/68**.

Why DHCP?

Without it, you would manually set IPs for every device → very time-consuming.

SSH (Secure Shell)

Used for **secure remote login** to servers.
Example: Logging into a Linux server from your laptop.

Features:

- Encrypted communication
- Secure file transfer (SCP, SFTP)

Works on **Port 22**.

FTP (File Transfer Protocol)

Used for **uploading/downloading files** between client and server.
Works on **Port 20 & 21**.

Versions

- **FTP** – Not encrypted
- **FTPS** – Encrypted (uses SSL)
- **SFTP** – Secure FTP (uses SSH)