Microsoft

Working with Types

Learning Units

- Types
- Type Casting
- Strong Typing
- Static Members

Types

Types

Every object has a type

- Object types are **determined** based on the value

PowerShell is **non-declarative** (weakly typed)

- Variables can be **reused** to store **any** object type

Types can be converted **implicitly** or **explicitly**

- **Implicit** conversions convert the data on the **right** to the **type** on the **left**

```
PS> $Salary = '1000'
PS> $Bonus = 100

PS> $NewSalary = $Salary + $Bonus

PS> $NewSalary
1000100
```

String left

String result

```
PS> $Salary = '1000'
PS> $Bonus = 100

PS> $NewSalary = $Bonus + $Salary

PS> $NewSalary
1100
```

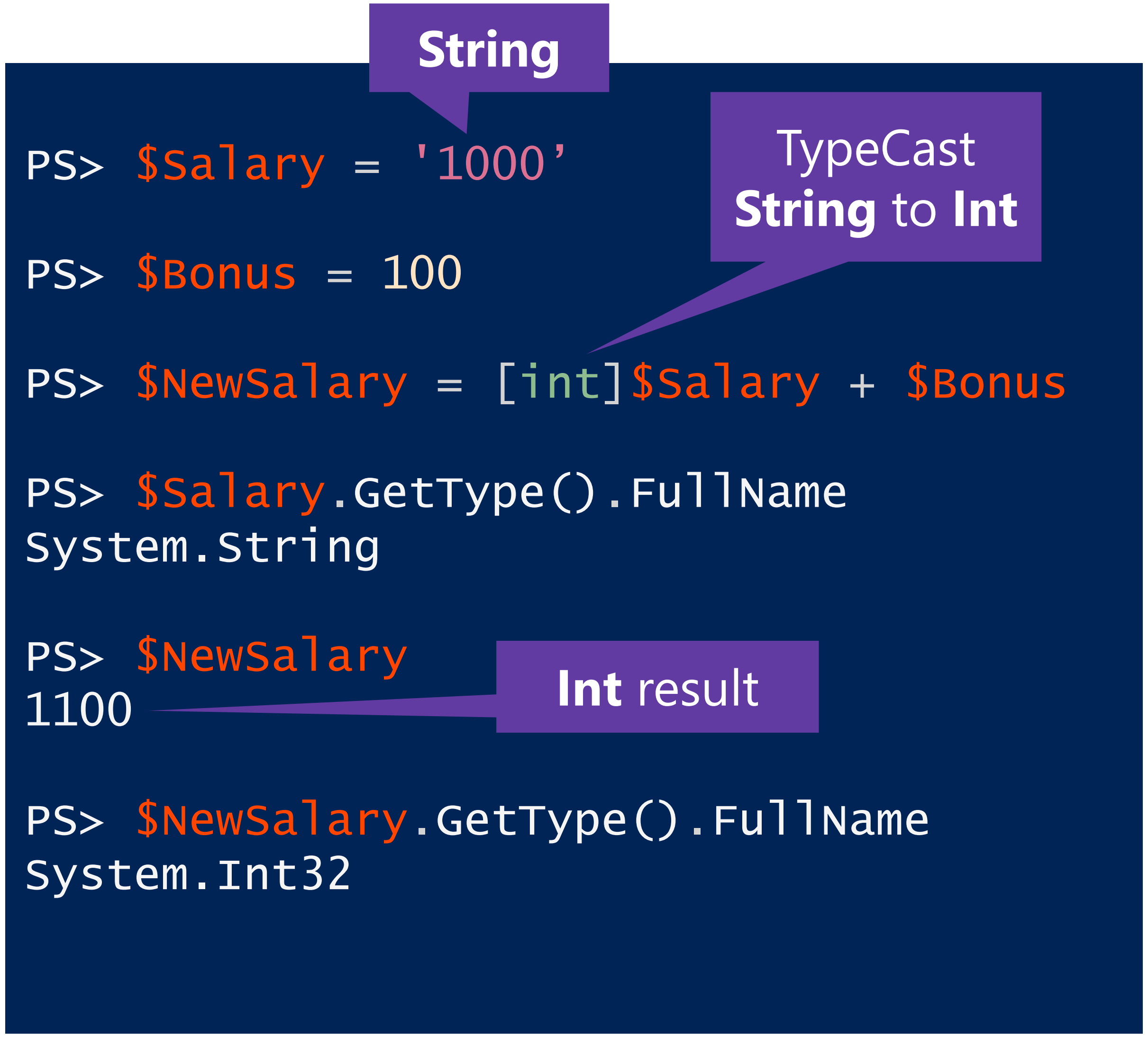
Int left

Int result

Type Casting

One-time explicit (forced) **conversion** of a value

Use **square brackets** around the **Type** Name next to the value to convert



```
PS> $Salary = '1000'

PS> $Bonus = 100

PS> $NewSalary = [int]$Salary + $Bonus

PS> $Salary.GetType().FullName
System.String

PS> $NewSalary
1100

PS> $NewSalary.GetType().FullName
System.Int32
```

Demonstration

Type conversions

Strong Typing

Strong Typing can be **applied** to a **variable**

Variable can **only** store objects of the specified **type** when **reused**

Attempts to **convert** any data stored in the variable to the **specified** type

```
PS> [Int]$Var2 = 123
PS> $Var2.GetType().FullName
System.Int

PS> $Var2 = "123"
PS> $Var2.GetType().FullName
System.Int

PS> $Var2 = "OneTwoThree"
Cannot convert value "OneTwoThree" to type "System.Int32".
Error: "Input string was not in a correct format."
```

Strong Typing Function Parameters

Any value used for that parameter will be converted to the specified type

The syntax reflects the parameter's type instead of <object>

```
Function Test-Params
{
    param ($Anything,
           [Int]$Number,
           [String]$Text)
}

PS> Get-Command Test-Params -Syntax

Test-Params [[-Anything] <Object>] [[-Number] <int>] [[-Text] <string>]
```

Type Accelerators

PowerShell is built on the .NET framework

Type Accelerators are aliases for .NET classes

PowerShell Accelerator	.NET Class Name
PSObject	System.Management.Automation.PSObject
PSCustomObject	System.Management.Automation.PSObject
Int	System.Int32
Long	System.Int64
Double	System.Double
String	System.String
DateTime	System.DateTime
PSCredential	System.Management.Automation.PSCredential
Xml	System.Xml.XmlDocument

Demonstration

Strong Typing

Questions?

Static Members and Enums

Static Members

Static members are **properties** or **methods** that **never** change, and live on the type itself

Not all .NET Framework classes can be **created** because they just exist as a **reference** libraries

- [Math]
- [Environment]

Many types contain **static members** even if they can be created, such as **[DateTime]**

Are **listed** by using the **–Static** parameter with the **Get-Member** cmdlet

```
PS> [math] | Get-Member -Static
 TypeName: System.Math

Name                MemberType Definition
----                ---------- ----------
Max                 Method     static sbyte Max(sbyte val1, sbyte val2)
Min                 Method     static sbyte Min(sbyte val1, sbyte val2)
Round               Method     static double Round(double value, int digits)
E                   Property   static double E {get;}
PI                  Property   static double PI {get;}
```

Static Members

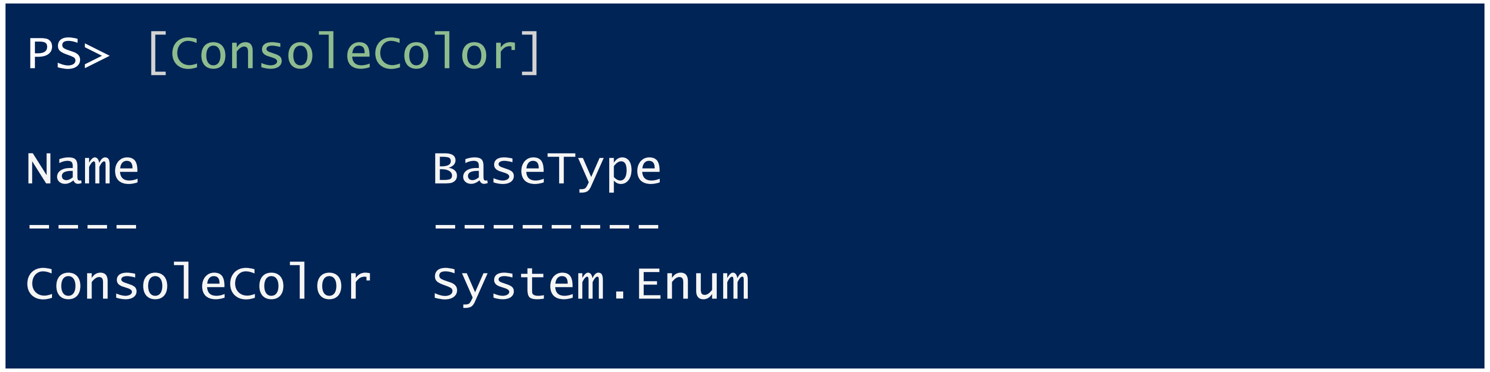
```
PS> [math]::PI
3.14159265358979

PS> [math]::Round(3.14159265358979, 3)
3.142

PS> [math]::Round([math]::PI, 3)
3.142

PS> [datetime]::now.Month
12

PS> [datetime]::DaysInMonth(2000, 2)
29

PS> [datetime]::DaysInMonth([datetime]::Now.Year, [datetime]::now.Month)
31
```

Enums

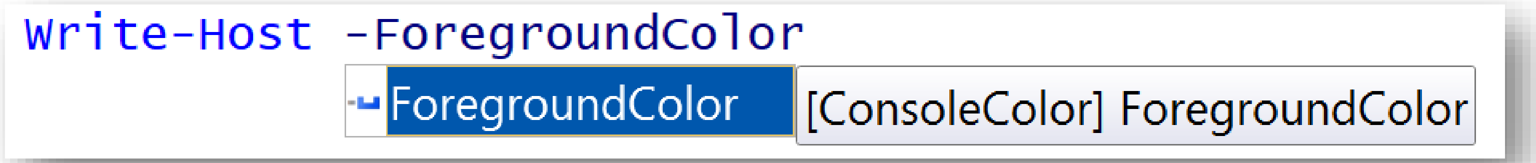
Enums are used to create reusable values with a friendly name

Access those values with static member syntax

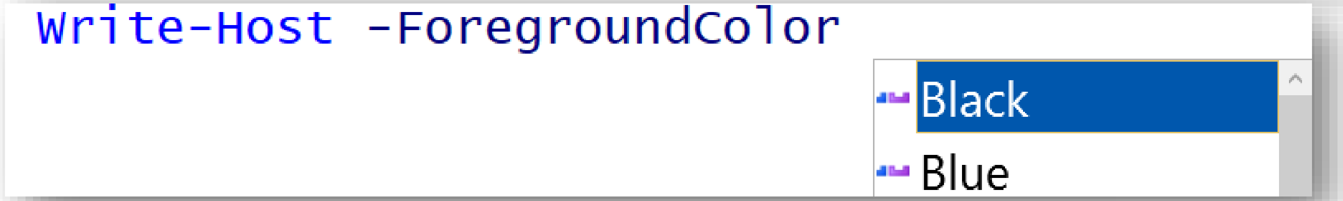
Often used for parameter input on specific cmdlets



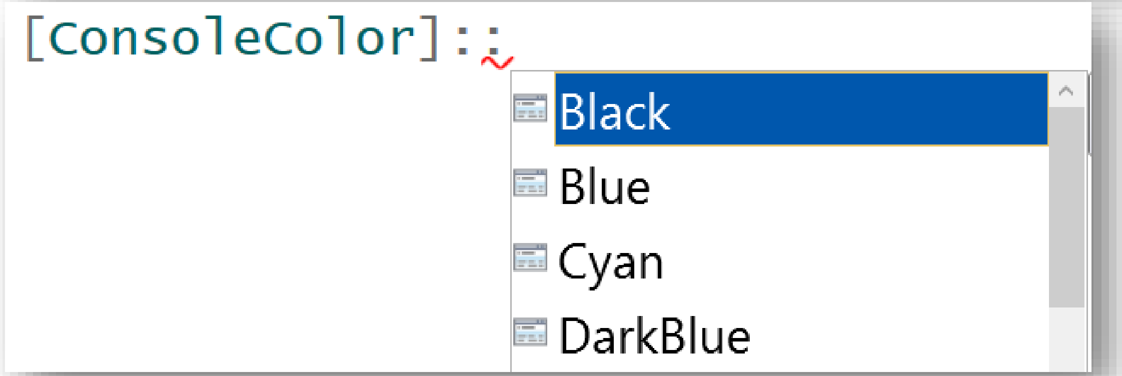
```
PS> [ConsoleColor]

Name          BaseType
----          --------
ConsoleColor  System.Enum
```



```
Write-Host -ForegroundColor
```



```
Write-Host -ForegroundColor
```



```
[ConsoleColor]::
```

Demonstration

Static Members

Questions?

Additional Info

Type Comparison

The **-is** and **-isnot** Type Comparison Operators are used to determine if an object is a **specific type** or not

```
$Num = 123
$Str = "ABC"

PS> $Num -is [int]
True

PS> $Num -is $Str.GetType()
False

PS> $Num -is [String]
False
```

```
$Num = 123
$Str = "ABC"

PS> $Num -isnot [int]
False

PS> $Num -isnot $Str.GetType()
True

PS> $Num -isnot [String]
True
```

Escape character

Backtick **removes** interpretation of **special** characters

Creates a special character inside of **expandable** strings

Escape Sequences	
`0	Null
`a	Alert
`b	Backspace
`f	Form feed
`n	New line
`r	Carriage return
`t	Horizontal tab
`v	Vertical tab

```
PS> Write-Host "`$Home is $Home"
$Home is C:\Users\admin

PS> Write-Host "Hello World" `
-ForegroundColor Green
Hello World

PS> Write-Host "Hello`nWorld" -ForegroundColor Yellow
Hello
World

PS> Write-Host "Hello`tWorld" -ForegroundColor Cyan
Hello    World
```

Parsing Modes

Parsing modes allows for **shortcuts**, by **allowing** PowerShell to **assume** a string has been typed

- `Get-Service alg` instead of `Get-Service "alg"`

Statements are broken into **tokens** and then **interpreted** in two ways

Tokens are interpreted as "**expressions**" until a command is **invoked**

Expression mode treats everything **only** as specified:

- Literal value **numbers**
- $ causes **variable** calls
- Operators, such as **arithmetic** are performed
- Strings must be **quoted**

Argument Mode is designed for parsing parameters for commands.

- All input is treated as an **expandable** string
- **Force** back to expression mode with special characters, such as **parenthesis** or **quotes**.

Parsing Modes

```
#Expression mode does the expected work
PS> 2+2
4

#Argument mode, treats everything as strings
PS> Write-Host 2+2 -ForegroundColor Green
2+2

#Expression mode forced on 2+2
PS> Write-Host (2+2) -ForegroundColor Green
4

#Expression mode needs quotes for strings
PS> Test
Test : The term 'Test' is not recognized as the name of a cmdlet

PS> "Test"
Test
```

Demonstration

-Is operator

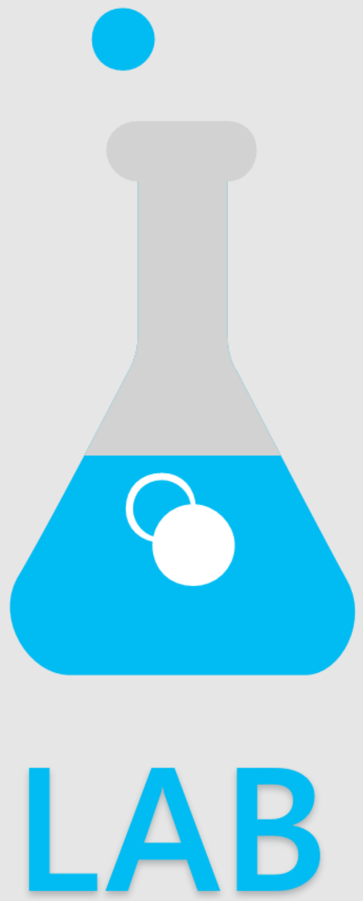
Escape character

Parsing modes

Questions?

Lab 10:
Working with Data Types

60 minutes



Microsoft