Hidden Secrets of the PowerShell Masters

June 2023

MEMUG

Nathan Ziehnert

06.30.2023

Special Thanks To Our Sponsors!

PATCH MY PC

Let us handle the tedious work of packaging, testing, deploying, and troubleshooting application updates in your ConfigMgr or Intune environment. Easily extend Microsoft Endpoint Manager to deploy and update over third-party applications within your enterprise.

Save time, money, and stay secure by automating the publishing of third-party updates to your environment. Setup only takes minutes. All subscriptions include free in-house support and setup calls!

RECAST SOFTWARE

Recast Software creates tools used by hundreds of thousands of enterprise organizations worldwide, impacting millions of devices and (more importantly) the people who use them. Our mission is to be an integral part of how IT teams create highly secure and compliant environments, capable of handling technological change. We do this by integrating with existing IT infrastructure to provide deeper, more actionable insights, improved productivity, and powerful, scalable automation.

ScriptRunner®
The #1 for PowerShell Management

ScriptRunner is the #1 platform for IT infrastructure management with PowerShell. Centralizing, standardizing, automating, delegating, monitoring and controlling routine tasks frees up resources in IT operations. Administrators and DevOps teams can use and customize included script libraries or develop their own scripts.

ScriptRunner allows you to securely delegate administrative tasks to users without PowerShell knowledge or appropriate rights. ScriptRunner is used worldwide by IT teams of all sizes and industries.

MEMUG

Announcements!

MEMUG

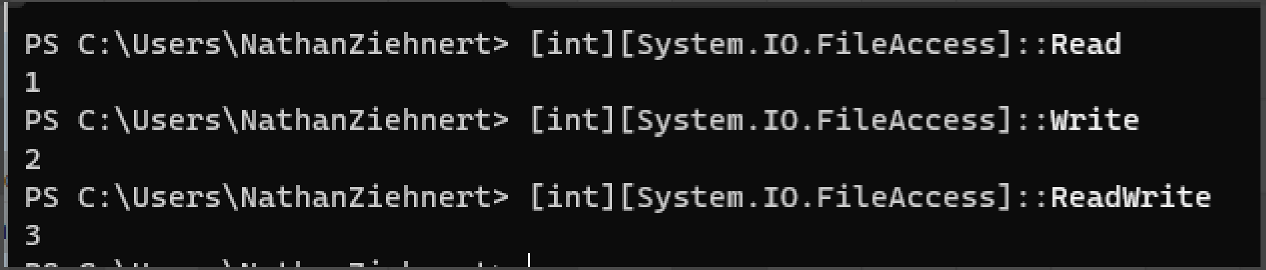
Enums

Comfortably Enum?

MEMUG

Enum(eration)s

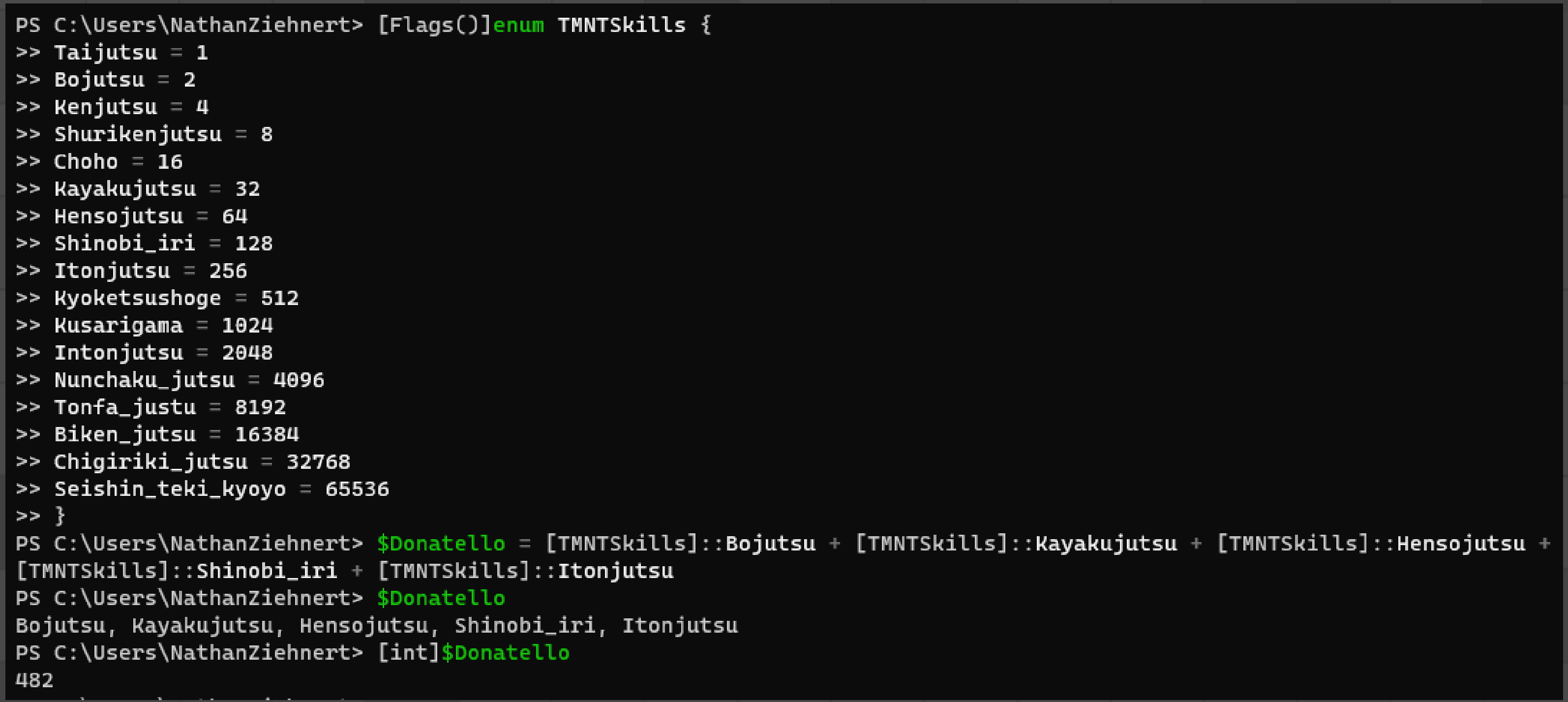
- Named labels (think strings) given an integer value
- Allow you to forget numbers and remember words
  - FileAccess Enum:
    - Read = 1
    - Write = 2
    - ReadWrite = 3



```
PS C:\Users\NathanZiehnert> [int][System.IO.FileAccess]::Read
1
PS C:\Users\NathanZiehnert> [int][System.IO.FileAccess]::Write
2
PS C:\Users\NathanZiehnert> [int][System.IO.FileAccess]::ReadWrite
3
```

Flags

- A definition to add to an enumeration
- Think of them like light switches…
- Should be Powers of 2, to work properly



```
PS C:\Users\NathanZiehnert> [Flags()]enum TMNTSkills {
>> Taijutsu = 1
>> Bojutsu = 2
>> Kenjutsu = 4
>> Shurikenjutsu = 8
>> Choho = 16
>> Kayakujutsu = 32
>> Hensojutsu = 64
>> Shinobi_iri = 128
>> Itonjutsu = 256
>> Kyoketsushoge = 512
>> Kusarigama = 1024
>> Intonjutsu = 2048
>> Nunchaku_jutsu = 4096
>> Tonfa_justu = 8192
>> Biken_jutsu = 16384
>> Chigiriki_jutsu = 32768
>> Seishin_teki_kyoyo = 65536
>> }
PS C:\Users\NathanZiehnert> $Donatello = [TMNTSkills]::Bojutsu + [TMNTSkills]::Kayakujutsu + [TMNTSkills]::Hensojutsu +
[TMNTSkills]::Shinobi_iri + [TMNTSkills]::Itonjutsu
PS C:\Users\NathanZiehnert> $Donatello
Bojutsu, Kayakujutsu, Hensojutsu, Shinobi_iri, Itonjutsu
PS C:\Users\NathanZiehnert> [int]$Donatello
482
```

Enums Demo

MEMUG

Why The Masters Use This

Especially helpful for putting words to numbers

Settings/Property Flags

If you've used WMI or the PowerShell Module for ConfigMgr

Flags… flags everywhere…

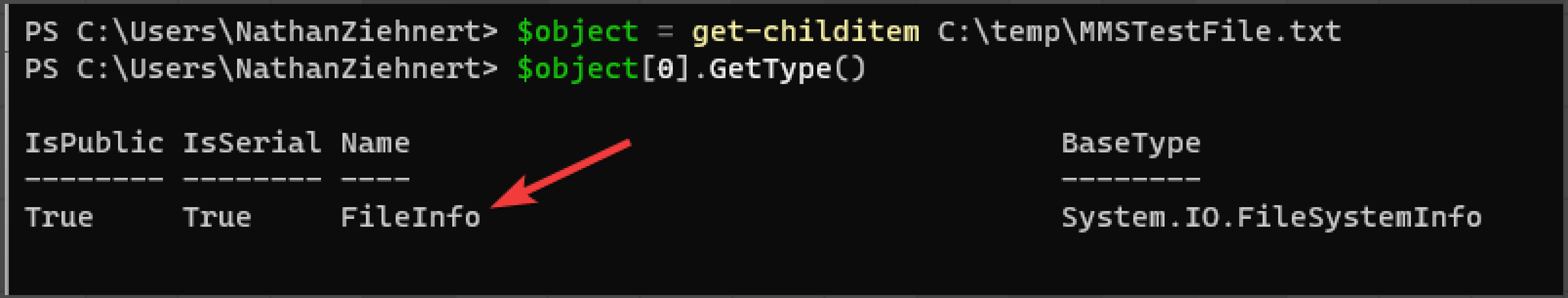
Classes

I thought we were done with school…

MEMUG

Classes

- Fundamental Concept in Object-Oriented Programming
- Object Blueprints
- You already use them! Now you can make them…
  - Example: Get-ChildItem (for files) returns a `FileInfo` class instance



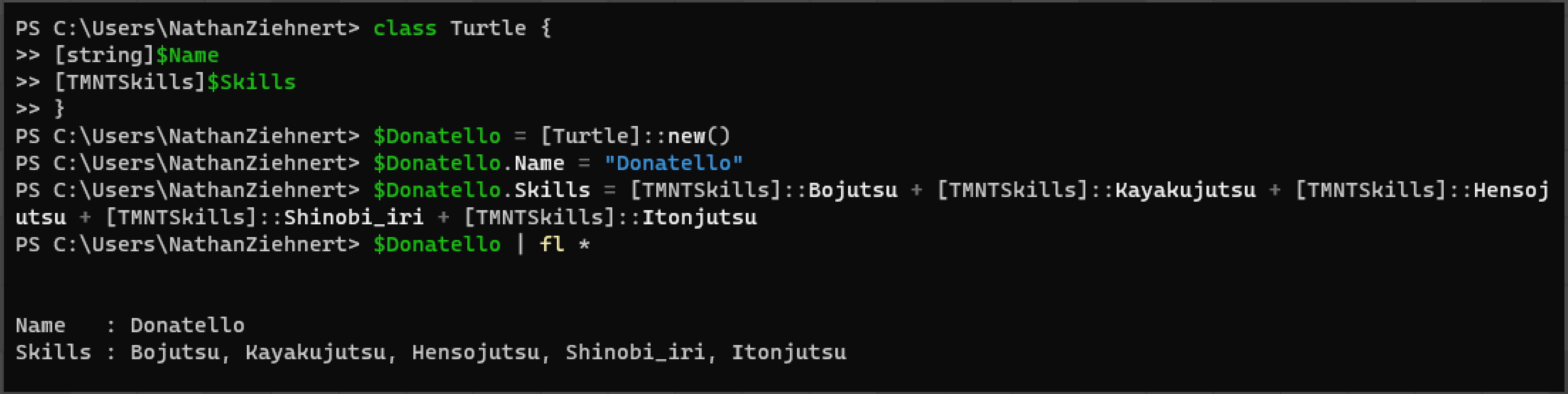
```
PS C:\Users\NathanZiehnert> $object = get-childitem C:\temp\MMSTestFile.txt
PS C:\Users\NathanZiehnert> $object[0].GetType()

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     True     FileInfo                                 System.IO.FileSystemInfo
```

- PSCustomObject is your baby step into this world…

MEMUG

Properties/Attributes

- Variables assigned to the class
- Can be primitives (string, int, etc.), enums, or even other classes
- Allows you to describe something about your object

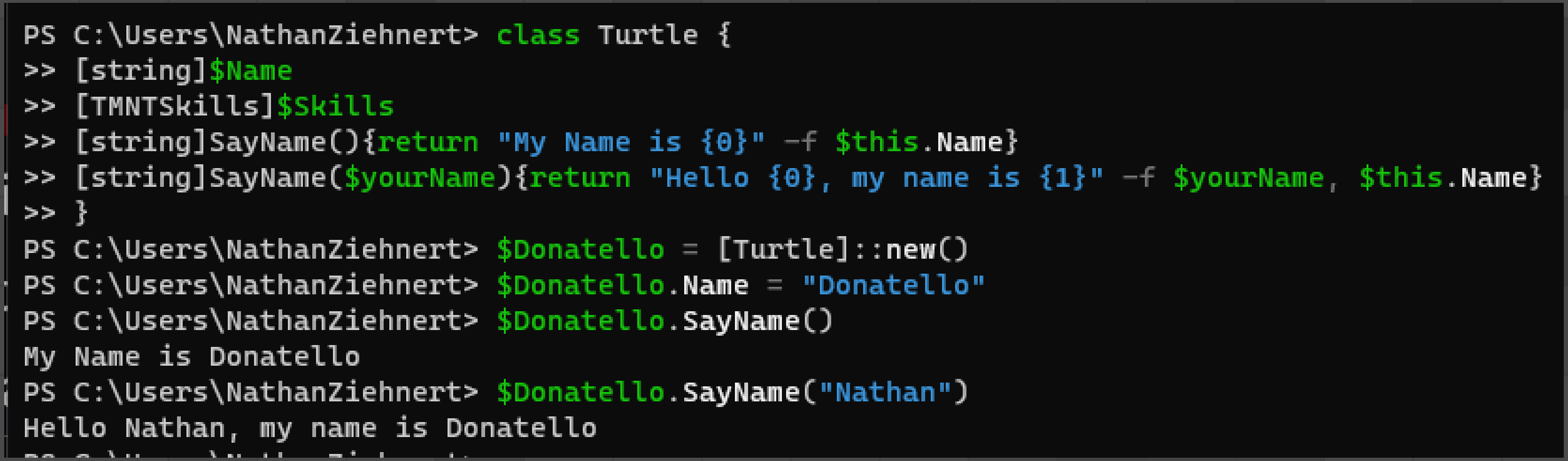


```
PS C:\Users\NathanZiehnert> class Turtle {
>> [string]$Name
>> [TMNTSkills]$Skills
>> }
PS C:\Users\NathanZiehnert> $Donatello = [Turtle]::new()
PS C:\Users\NathanZiehnert> $Donatello.Name = "Donatello"
PS C:\Users\NathanZiehnert> $Donatello.Skills = [TMNTSkills]::Bojutsu + [TMNTSkills]::Kayakujutsu + [TMNTSkills]::Hensoj
utsu + [TMNTSkills]::Shinobi_iri + [TMNTSkills]::Itonjutsu
PS C:\Users\NathanZiehnert> $Donatello | fl *


Name   : Donatello
Skills : Bojutsu, Kayakujutsu, Hensojutsu, Shinobi_iri, Itonjutsu
```

MEMUG

Methods

- They're like functions… sort of
- Can take parameters, can return data…
- Supports "Overloading"

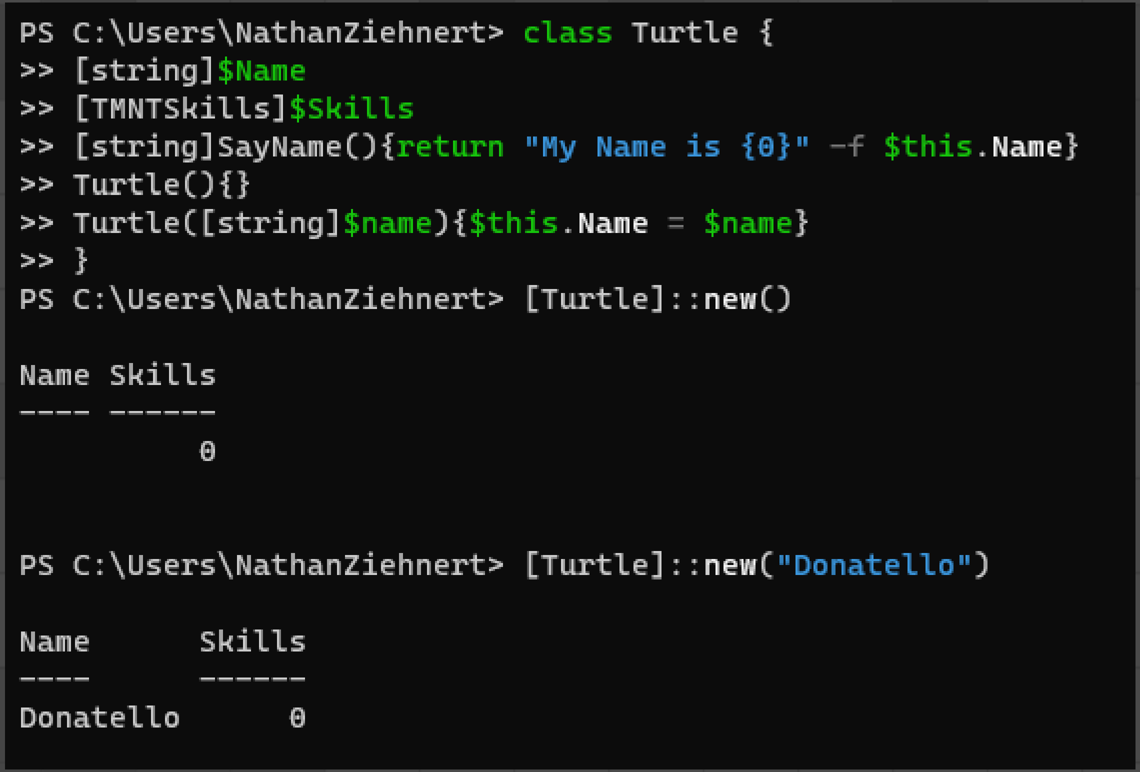


```
PS C:\Users\NathanZiehnert> class Turtle {
>> [string]$Name
>> [TMNTSkills]$Skills
>> [string]SayName(){return "My Name is {0}" -f $this.Name}
>> [string]SayName($yourName){return "Hello {0}, my name is {1}" -f $yourName, $this.Name}
>> }
PS C:\Users\NathanZiehnert> $Donatello = [Turtle]::new()
PS C:\Users\NathanZiehnert> $Donatello.Name = "Donatello"
PS C:\Users\NathanZiehnert> $Donatello.SayName()
My Name is Donatello
PS C:\Users\NathanZiehnert> $Donatello.SayName("Nathan")
Hello Nathan, my name is Donatello
```

MEMUG

Constructors

- Set / Validate objects on creation
- Same name as class
- Zero or more allowed… i.e. supports overloading

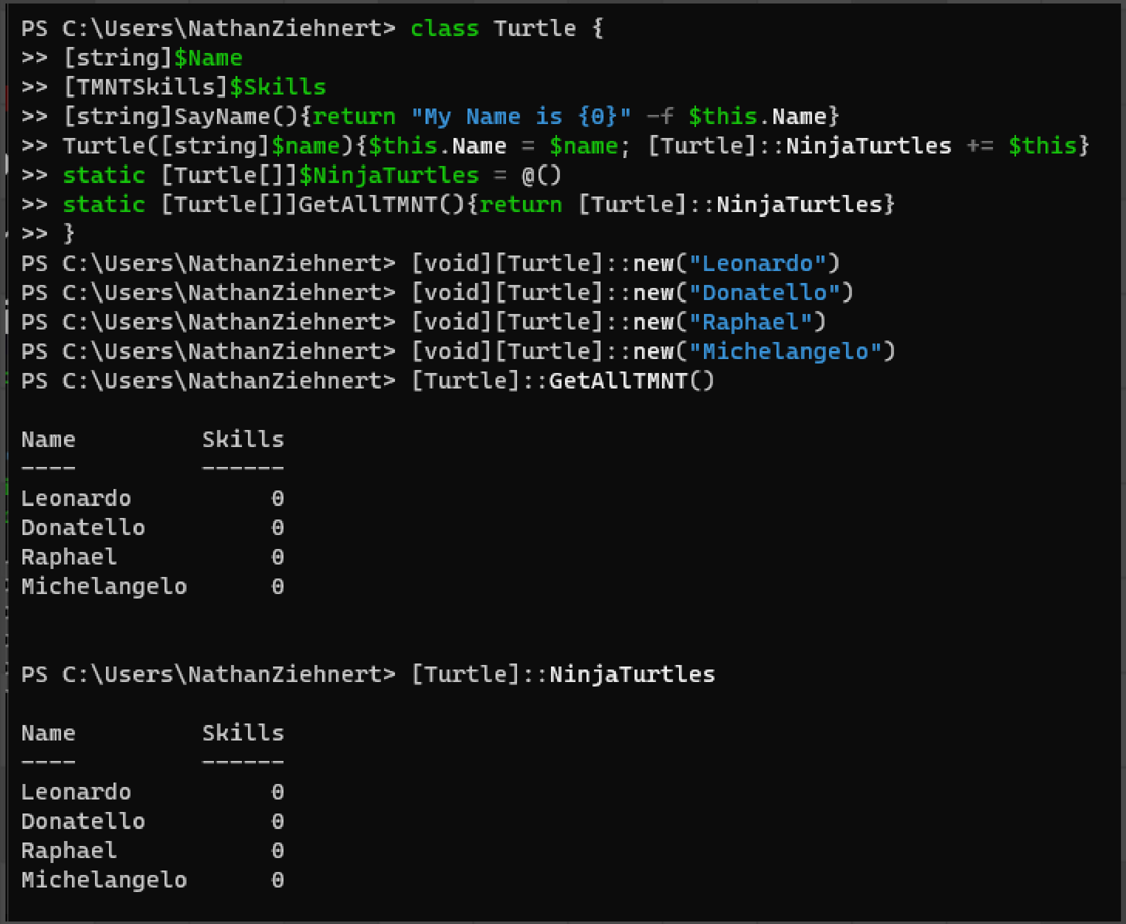


```
PS C:\Users\NathanZiehnert> class Turtle {
>> [string]$Name
>> [TMNTSkills]$Skills
>> [string]SayName(){return "My Name is {0}" -f $this.Name}
>> Turtle(){}
>> Turtle([string]$name){$this.Name = $name}
>> }
PS C:\Users\NathanZiehnert> [Turtle]::new()

Name Skills
---- ------
          0


PS C:\Users\NathanZiehnert> [Turtle]::new("Donatello")

Name      Skills
----      ------
Donatello      0
```

MEMUG

Statics

- Attribute for properties and methods
- Exists as part of the class, not an instance of the class
- Shared across all instances of the class

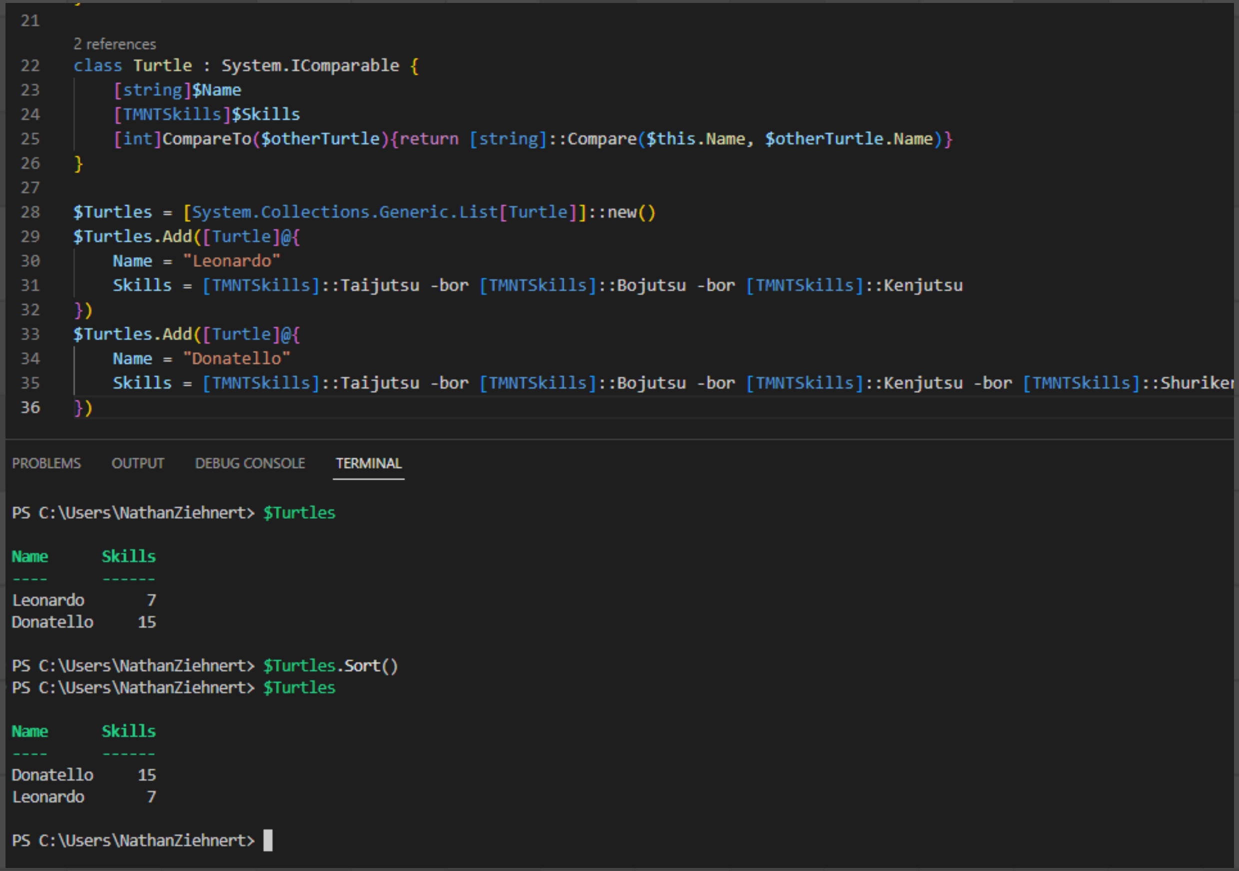


```
PS C:\Users\NathanZiehnert> class Turtle {
>> [string]$Name
>> [TMNTSkills]$Skills
>> [string]SayName(){return "My Name is {0}" -f $this.Name}
>> Turtle([string]$name){$this.Name = $name; [Turtle]::NinjaTurtles += $this}
>> static [Turtle[]]$NinjaTurtles = @()
>> static [Turtle[]]GetAllTMNT(){return [Turtle]::NinjaTurtles}
>> }
PS C:\Users\NathanZiehnert> [void][Turtle]::new("Leonardo")
PS C:\Users\NathanZiehnert> [void][Turtle]::new("Donatello")
PS C:\Users\NathanZiehnert> [void][Turtle]::new("Raphael")
PS C:\Users\NathanZiehnert> [void][Turtle]::new("Michelangelo")
PS C:\Users\NathanZiehnert> [Turtle]::GetAllTMNT()

Name         Skills
----         ------
Leonardo          0
Donatello         0
Raphael           0
Michelangelo      0


PS C:\Users\NathanZiehnert> [Turtle]::NinjaTurtles

Name         Skills
----         ------
Leonardo          0
Donatello         0
Raphael           0
Michelangelo      0
```

MEMUG

Inheritance (Base / Interface)

- Extending classes
  - Inherit Base properties/methods
  - Fulfill a contract (Interface)
- You can implement .NET Interfaces (like IComparable)



```
2 references
class Turtle : System.IComparable {
    [string]$Name
    [TMNTSkills]$Skills
    [int]CompareTo($otherTurtle){return [string]::Compare($this.Name, $otherTurtle.Name)}
}

$Turtles = [System.Collections.Generic.List[Turtle]]::new()
$Turtles.Add([Turtle]@{
    Name = "Leonardo"
    Skills = [TMNTSkills]::Taijutsu -bor [TMNTSkills]::Bojutsu -bor [TMNTSkills]::Kenjutsu
})
$Turtles.Add([Turtle]@{
    Name = "Donatello"
    Skills = [TMNTSkills]::Taijutsu -bor [TMNTSkills]::Bojutsu -bor [TMNTSkills]::Kenjutsu -bor [TMNTSkills]::Shurike
})
```

```
PS C:\Users\NathanZiehnert> $Turtles

Name      Skills
----      ------
Leonardo       7
Donatello     15

PS C:\Users\NathanZiehnert> $Turtles.Sort()
PS C:\Users\NathanZiehnert> $Turtles

Name      Skills
----      ------
Donatello     15
Leonardo       7

PS C:\Users\NathanZiehnert>
```

MEMUG

Classes Demo

MEMUG

Why The Masters Use This

- Organize your data better!
- Fun with Statics (Methods and Attributes)
- Implementing helpful .NET Interfaces (IComparable)

MEMUG

.NET Namespaces Classes and Methods

More classes?

MEMUG

Docs, Docs, Docs!

- .NET
  https://learn.microsoft.com/en-us/dotnet/api/?view=net-7.0
- .NET Framework
  https://learn.microsoft.com/en-us/dotnet/api/?view=netframework-4.8.1

MEMUG

.NET 7 vs .NET Framework

- Is there a difference? Yes.
- Will it affect you? Probably not.
    - Windows specific items…
    - dotnet can likely load .NET Framework libraries*… the reverse is not true…

MEMUG

Nuget Packages

- When you want something special…
  it actually probably exists in .NET already…
- For those rare exceptions where it doesn’t, there’s Nuget.org
- How To: Reference It
  `Add-Type -Path("c:\path\to\thingy.dll")`
- Cross-compatibility (pswh/PowerShell)?

MEMUG

How To Use Them!

- The same way you use PowerShell classes!
- Instance Constructors
  `[Namespace.Namespace.Namespace.Class]::new()`
- Static Methods
  `[Namespace.Namespace.Namespace.Class]::method()`
- Fields/Constants/Enum
  `[Namespace.Namespace.Namespace.Class]::Field`

MEMUG

Using Namespace

- Tired:

```
$list = [System.Collections.Generic.List[int]]::new()
```

- Wired:

```
using namespace System.Collections.Generic
$list = [List[int]]::new()
$list2 = [List[string]]::new()
```

MEMUG

Crawl (Math)

- [System.Math] https://learn.microsoft.com/en-us/dotnet/api/system.math.sign?view=net-7.0
- Fields
  - E
  - Pi
  - Tau
- Methods
  - Round
  - Floor/Ceiling
  - Min/Max

MEMUG

Math Examples

MEMUG

Walk (Array Replacements)

- [System.Collections.XXXXXX]
  https://learn.microsoft.com/dotnet/api/system.collections?view=net-7.0
  - Queue (FIFO)
  - Stack (LIFO)
  - ArrayList (NO! BAD! NO!)
- [System.Collections.Generic.XXXXXX[_____]]
  - Queue (FIFO)
  - Stack (LIFO)
  - List

MEMUG

Collection Examples

MEMUG

Run (System.IO.File)

- For very large files… you might see a performance increase
- If you need to read bytes instead of strings
  - [System.IO.File]::ReadAllBytes("c:\path\to\file.exe")
- Also works with text, although maybe no measurable difference
  - [System.IO.File]::ReadAllLines("c:\path\to\text.txt")

System.IO.File Example

MEMUG

Why The Masters Use This

- Unlock some serious new potential
- Some things are better than what is “built-in” (List vs. Array)
- Some things might be faster than what is “built-in” (System.File.IO)

P/Invoke

Pee What?

MEMUG

P/invoke – Turtle Power!

- What is it?
  - Calling code that is stored in a DLL
  - “Unmanaged” code
- What can it do?
  - Everything!
- Why?
  - Need to use a third-party DLL
  - Need to use a WinAPI
- Quick Deep Dive…

MEMUG

P/Invoke Demo

MEMUG

Why The Masters Use This

- Use libraries from other programming languages
- Supported Control over the OS
- Performance

MEMUG

Advanced Troubleshooting

MEMUG

Try/Catch/Finally

- Try
  - Run this block of code, and see if there is an error
- Catch
  - If a terminating error is encountered, run this script block to handle the error
- Finally
  - Runs after try/catch to cleanup anything that might have been created during try/catch

- You can “catch” specific errors – so you could handle different exceptions differently

MEMUG

Try/Catch/Finally

```
Try {
  ThisDoesntExistAndWillFail
  New-Item C:\Test.txt -Force
}
Catch {
  Write-Output "Huh... Guess that function doesn’t exist"
}
Finally {
  if(Test-Path C:\Test.txt) {
    Remove-Item C:\Test.txt -Force
  }
}
```

MEMUG

Try/Catch/Finally Demo

MEMUG

Wait-Debugger

- Sort of like a nested prompt
- But actually a Debugger!
    - Step in/over/out
    - Continue/quit

MEMUG

Trap

- A script block to run when encountering unhandled terminating errors
- Like "catch" you can handle SPECIFIC errors
- FIFW – First In, First Win…
- Nice to combine with Wait-Debugger

MEMUG

Trap

```
Trap {
  Write-Host "!!!!!" -Foreground Red
  Write-Host "Error Encountered" -Foreground Red
  Write-Host $_ -Foreground Red
  Write-Host "!!!!!" -Foreground Red
  Wait-Debugger
}
ThisFunctionDoesntExist
```

Trap Demo

MEMUG

Why The Masters Use This

- Nobody’s code is perfect
- Troubleshooting scripts without an ISE
- Handling !expected! errors

Questions?

MEMUG

Special Thanks To Our Sponsors!

Let us handle the tedious work of packaging, testing, deploying, and troubleshooting application updates in your ConfigMgr or Intune environment. Easily extend Microsoft Endpoint Manager to deploy and update over third-party applications within your enterprise.

Save time, money, and stay secure by automating the publishing of third-party updates to your environment. Setup only takes minutes. All subscriptions include free in-house support and setup calls!

PATCH MY PC

Recast Software creates tools used by hundreds of thousands of enterprise organizations worldwide, impacting millions of devices and (more importantly) the people who use them. Our mission is to be an integral part of how IT teams create highly secure and compliant environments, capable of handling technological change. We do this by integrating with existing IT infrastructure to provide deeper, more actionable insights, improved productivity, and powerful, scalable automation.

RECAST SOFTWARE

ScriptRunner®

The #1 for PowerShell Management

ScriptRunner is the #1 platform for IT infrastructure management with PowerShell. Centralizing, standardizing, automating, delegating, monitoring and controlling routine tasks frees up resources in IT operations. Administrators and DevOps teams can use and customize included script libraries or develop their own scripts.

ScriptRunner allows you to securely delegate administrative tasks to users without PowerShell knowledge or appropriate rights. ScriptRunner is used worldwide by IT teams of all sizes and industries.

MEMUG