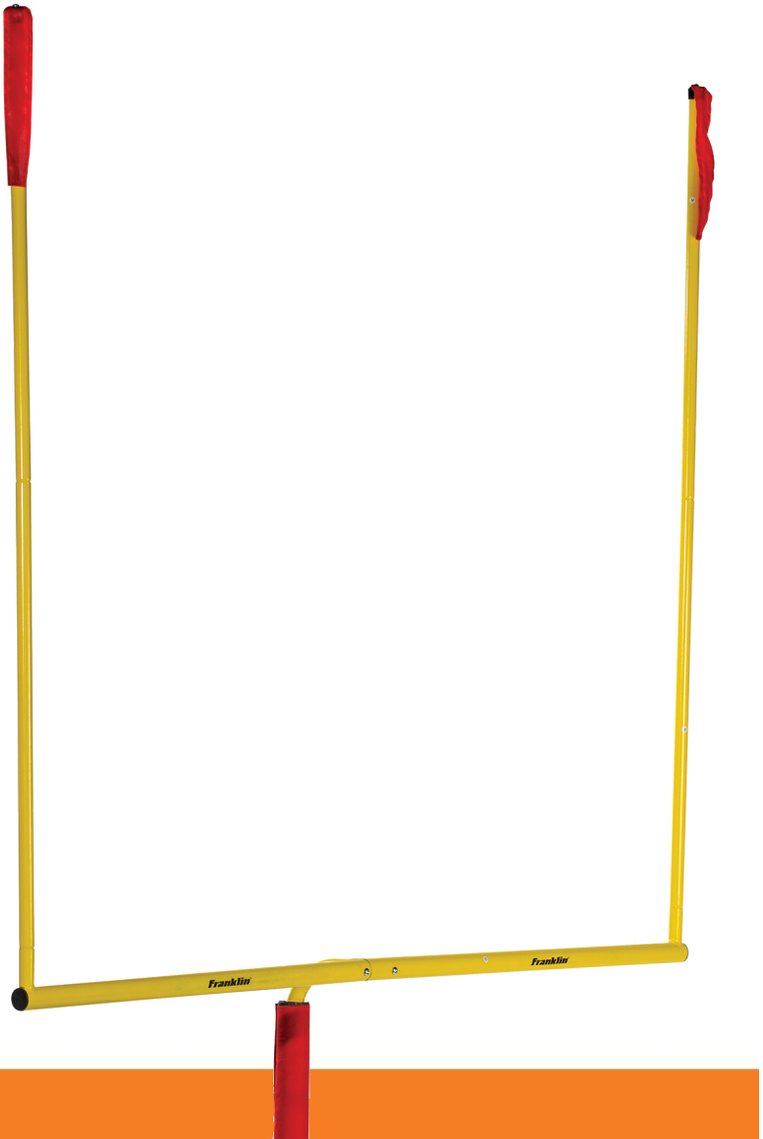
Objects

Develop Intelligence

Goals

1. List 2 benefits of using classes
2. And 2 drawbacks
3. Explain why to use an enum



Roadmap

1. What and Why?
2. Adding Class
3. Enumerations

Develop
Intelligence

What and Why?

Develop Intelligence

Complaints about Tuples

- No naming for clarity
- No help from intellisense
- No help from `Get-Help`
- No protection of invariants

```powershell
# Destructuring
$Login, $Groups = Get-UserInfo -SID 192921

# Color enum
$red = 255,0,'chicken'
$yellow = 255,275,0
$purple = 128,0,128
```

Dictionary Complaints

- No help from intellisense
- No help from `Get-Help`
- No protection of invariants

```powershell
$yellow = @{red=755; green=255; blue=0}
$purple = @{red=128; green=0; blue='spam'}
```

Develop
Intelligence

Adding Class

Develop Intelligence

Overview

- Relatively new
- Great as Data Transfer Objects (DTOs)
  - aka POCO / POJO
- Limited
  - No syntax for getter/setter
  - Not discoverable
- Heavy OO-style isn't idiomatic to PowerShell

Example

```
class User{
    [int]$id
    [string]$name
}

# Call the default constructor
$admin = [User]::new()

# Set properties
$admin.id=3881
$admin.name='administrator'
```

Better: Casting

Works

```
$admin = [User]@{
  id = 3881
  name= 'administrator'
}
```

Fail

```
$admin = [User]@{
  id = 3881
  name = 'administrator'
  handle = 'jerry'
}
```

Validation

- Use the same attributes as you would for function parameters

```
class Color{
    [ValidateRange(0, 255)]
    [int]$red

    [ValidateRange(0, 255)]
    [int]$green

    [ValidateRange(0, 255)]
    [int]$blue
}
```

Methods

- A functions on a a class is a *method*
- It's Totally supported
- **But** doesn't feel idiomatic

```
class Color{
   [int]$red
   [int]$green
   [int]$blue

   [void]Darken([int] $amount){
      $this.green += $amount
      $this.red += $amount
      $this.blue += $amount

   }
```

Example

Maybe don't do this...

```
[Color]$primary = @{red = 250; green = 100; blue = 31}
[Color]$accent = @{red = 190; green = 19; blue = 100}
[Color]$alert = @{red = 25; green = 128; blue = 56}

$palette = $primary, $accent, $alert |
            %{$_.Darken(10);Write-Output $_}
```

Idiomatic Alternative

Manipulate objects with cmdlets

```
function Add-Darkness{
   [CmdletBinding()]
   [OutputType([Color])]
   param(
      [Parameter(Mandatory)]
      [int] $Amount,
      [Parameter(ValueFromPipeline,Mandatory)]
      [Color] $Original
   )
   process{
    return [Color] @{
        red = $Original.red + $amount
        green = $Original.green + $amount
        blue = $Original.blue + $amount
    }
```

Example

Maybe do this

```
[Color]$primary = @{red = 250; green = 100; blue = 31}
[Color]$accent = @{red = 190; green = 19; blue = 100}
[Color]$alert = @{red = 25; green = 128; blue = 56}

$palette = $primary, $accent, $alert | Add-Darkness -Amount 10
```

ToString is the Exception

- Built in to .NET base object
- Useful for displaying DTOs as text

```
class User{
  [string] $GivenName
  [string] $Surname

  [string]ToString(){
    return "$($this.Surname), $($this.GivenName)"
  }
}
```

OO Shortcomings

Develop Intelligence

- **Ergonomics**
  - Doesn't feel like Powershell
  - Not discoverable
  - Ignored by `Import-Module`
- Always mutable
- Advice:
  - Use classes only for DTOs
  - Write a factory function

Factory Pattern

- **Best practice**: Make a factory function for instantiating custom objects

```
function Initialize-Color {
    [CmdletBinding()]
    [OutputType([Color])]
    Param (
        [int]$Red = 0,
        [int]$Green = 0,
        [int]$Blue = 0
    )
    return [Color]@{
        red = $Red
        green = $Green
        blue = $blue
    }
```

Develop
Intelligence

Enumerations

Develop Intelligence

Motivation

- What happens if we add sophistication?
- How does the user know the acceptable values for Urgency?

```
function Invoke-Cleanup{
  Param(
    [bool]$BeThorough = $false,
    [string]$Urgency = 'Normal'
  )
  Write-Output 'Cleaning up....'
}
```

Overview

- An `enum` is a strongly typed set of labels
  - Labels can be any integer
- No parsing or misspelling
- Supports flags

Refactored

```
enum CleanLevel{
    Lazy
    Normal
    Compulsive
}
enum Urgency{
    Normal
    Priority
}
function Invoke-Cleanup {
    Param(
        [CleanLevel]$Thoroughness = 'Normal',
        [Urgency]$Urgency = 'Normal'
    )


    Write-Output "Cleaning up $Thoroughness-style...."
```

Flags

- An `enum` can be a collection of bit flags
- Use for non-mutually exclusive values

```
[Flags()]enum FilePermision{
  None = 0
  Read = 1
  Write = 2
  Execute = 4
}
```

Motivation

```powershell
function New-Burger{
  Param(
    [bool]$HasPickles=$false,
    [bool]$HasOnions=$false,
    [bool]$HasMustard=$false
  )
  # ...
}
```

Refactored

Develop Intelligence

```powershell
[Flags()] enum Fixins {
  None = 0
  Pickles = 1
  Onions = 2
  Mustard = 4
}

function New-Burger{
  Param(
    [Fixins]$Fixins='None'
  )
  return "New burger with $Fixins"
}

New-Burger 'Pickles','Onions'
```

Lab: Weather

- Read instructions: $/labs/weather/README.md
- Build the applications starting with 'startingpoint'

Develop
Intelligence

Review

1. List 2 benefits of using classes
2. And 2 drawbacks
3. Explain why to use an enum