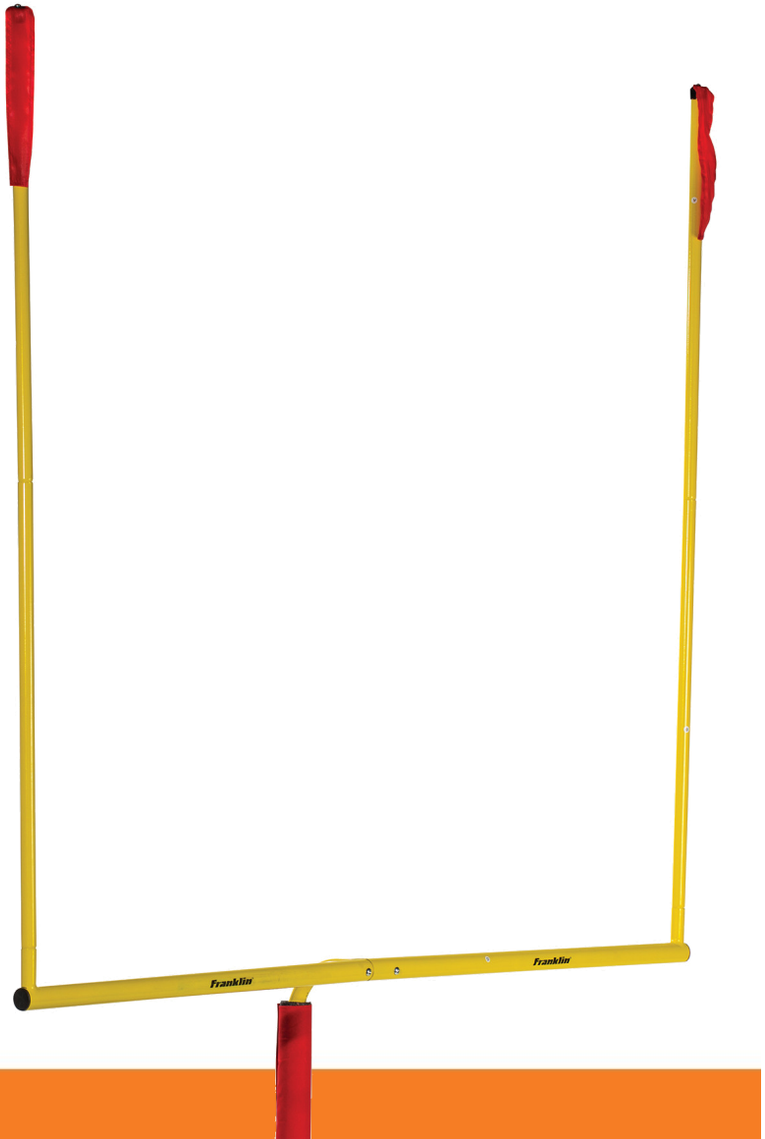
Functions

Develop Intelligence

Goals

1. List 2 flavors of function
2. Explain the advantages of strong typing
3. Define an 'advanced' function



Roadmap

1. Syntax
2. Adding Types
3. 'Advanced' Functions
4. Validation
5. Begin, Process, End

Develop
Intelligence

Syntax

Develop
Intelligence

Overview

- Named list of PowerShell statements
- Invoked like a cmdlet
- Variations
  - Plain
  - Advanced
  - Filter

Variations

This is ok

```
function Get-Identity($Self){
  return $Self
}
```

This is better

```
function Get-Identity{
  param{
    $Self
  }
  return $Self
}
```

Reasons to Prefer

- Same syntax for
  - Scripts
  - Codeblocks
  - 'Advanced Functions'

Return Values

- Explicit via
  - Keyword `return`
  - Cmdlet `Write-Output`
- Implicit via... anything else

Example

What's the output?

```
function Select-Sum($A, $B){
  $result = $A + $B
  "Result is: $result"
  return $result
}
$sum = Select-Sum 11 5
$sum.GetType()
```

Refactored

Develop Intelligence

```powershell
function Select-Sum($A, $B){
  $result = $A + $B
  Write-Information "Result is: $result"
  return $result
}
# Notice: the debug preference
$sum = Select-Sum 11 5 -Information Continue
$sum.GetType()
```

Alternative Preference Variables

- Examples:
  - `$InformationPreference`
  - `$DebugPreference`
  - `$VerbosePreference`
- Options
  - `SilentlyContinue`, `Continue`, `Stop`, `Ignore`…

Example


```powershell
function Select-Sum($A, $B){
  # ...
}

$InformationPreference = Continue
$sum = Select-Sum 11 5
$sum.GetType()
```

Develop
Intelligence

Adding Types

Develop Intelligence

Motivation

- For developers, not many guardrails
- For consumers, it's hard to tell what this is doing
  - What it returns
  - What it expects

```
function Select-Sum($A,$B){
  $result = $A + $B
  return $result
}
$sum = Select-Sum 11 'chicken' 44
```

Refactored

```powershell
function Select-Sum{
  [OutputType([int])]
  Param(
    [int]$A,
    [int]$B
  )
  $result = $A + $B
  Write-Information "Result is: $result"
  return $result
}
```

About `OutputType`

- Metadata only
- Not enforced
- **BUT** still a good idea

Example

```powershell
function Get-Pi{
  [OutputType([float])]
  Param()
  return "Apple"
}
"Expected type: "
(Get-Command Get-Pi).OutputType.Name

"`nActual type: "
(Get-Pi).GetType().Name
```

Develop
Intelligence

'Advanced' Functions

Develop Intelligence

Defined

Advanced functions allow you create cmdlets that are written as a PowerShell function. Advanced functions make it easier to create cmdlets without having to write and compile a binary cmdlet.

- **Motivation** having a function that works just like built-in cmdlets
  - `ShouldProcess`
  - `Whatif`

Overview

Not Advanced

```powershell
function Get-Self{
  Param($Self)
  return $Self
}
```

Totally Advanced

```powershell
function Get-Self{
  [CmdletBinding()]
  Param($Self)
  return $Self
}
```

Ex: SupportsShouldProcess

Develop Intelligence

```powershell
function Remove-File {
  [CmdletBinding(
    SupportsShouldProcess,
    ConfirmImpact = 'High'
  )]
  param(
     [Parameter(Mandatory)]
     [string]$path
  )
  if ($PSCmdlet.ShouldProcess($path)) {
    Remove-Item $path
  }
}
```

Develop
Intelligence

Validation

Develop Intelligence

Overview

- Powershell comes with attributes for checking parameters
- Most useful:
  - ValidateCount
  - ValidateRange
  - ValidateScript
  - ValidateSet

Example

```powershell
function Write-Update{
  param(
    [Parameter(Mandatory)]
    [ValidateLength(1, 100)]
    [string] $Message,

    [Parameter(Mandatory)]
    [ValidateSet('Low', 'Medium', 'High')]
    [string] $Severity,

    [Parameter(Mandatory)]
    [ValidateScript( { Test-Path $_ } )]
    [string] $Path
  )


  Write-Host "$Severity - $Message - $Path"
```

Develop
Intelligence

Begin, Process, End

Develop Intelligence

Motivation

Develop Intelligence

- Normally a function is **stateless**
- Working on a pipeline can require keeping state around

```
1  function Select-Sum([int[]]$Xs) {
2    $sum = 0
3    foreach($x in $Xs){$sum += $x}
4    return $sum
5  }
6  ## Wrong answers
7  $s1 = Get-Process | % WorkingSet | Select-Sum
8  $s2 = Get-Process | % WorkingSet | %{ Select-Sum $_ }
9
10 ## Slightly ugly


11 $sets = Get-Process | % WorkingSet
```

Piping into Functions

- Optionally, a function can have 3 blocks
  - **Begin** runs before the first thing piped in
  - **Process** for each item
  - **End** after the last item

```
function Save-Users{
  begin { Write-Host "Opening DB connection"  }

  process { Write-Host "Saving $_" }

  end{ Write-Host 'Closing connection' }



}
```

Select-Sum Refactored

Develop Intelligence

```powershell
function Select-Sum {
  begin { $sum = 0 }
  process { $sum += $_ }
  end {return $sum}
}

Get-Process | % WorkingSet | Select-Sum
```

Develop
Intelligence

Review

1. List 2 flavors of function
2. Explain the advantages of strong typing
3. Define an 'advanced' function