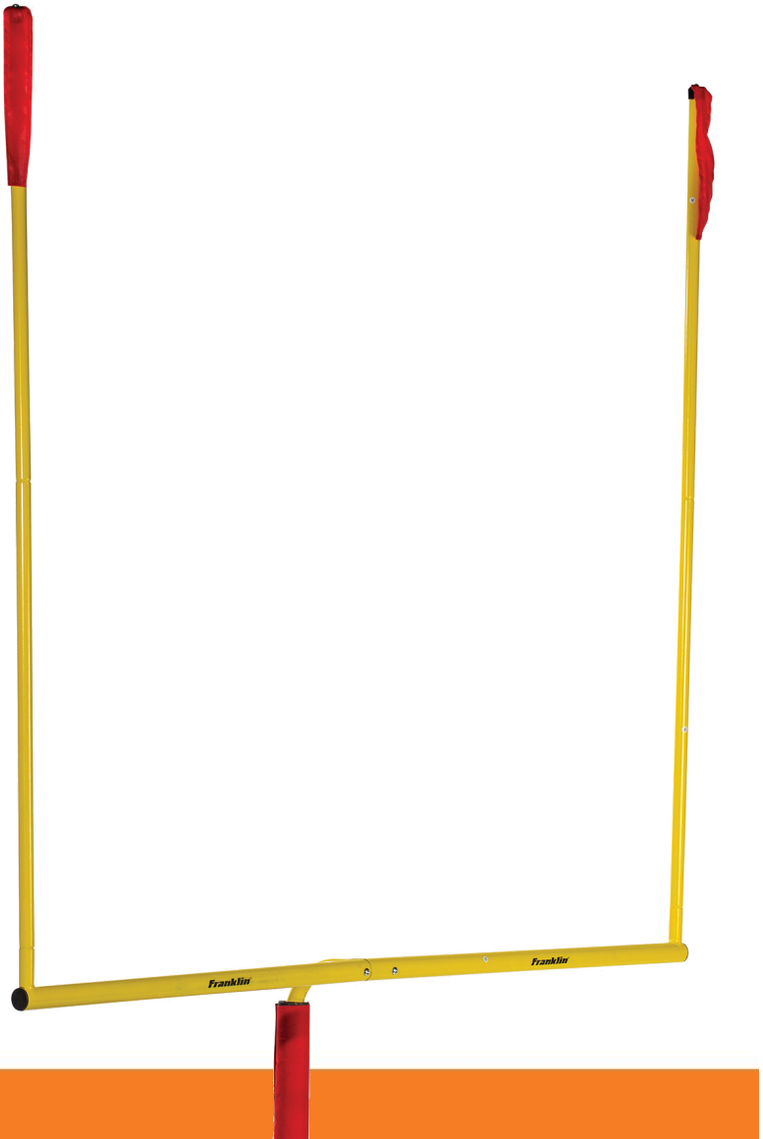
Expressions

Goals

1. Discuss two ways to invoke a function
2. List 3 different comparison operators
3. Name two different loop constructs



Roadmap

1. Variables
2. Comments
3. Functions
4. Strings
5. Conditions
6. Iteration

Develop
Intelligence

Variables

Develop Intelligence

PowerShell the Language

- Dynamically typed (mostly)
- Imperative (mostly)
- Inspired by
  - C#, JavaScript, etc
  - Python
  - Shell

Variables

- Associate names with values
- Pop into existence on assignment *
- Rules:
  - Prefixed with **$**
  - Only letters, numbers, and underscores
  - Don't start with a number

```
$path = 'c:/Windows'

$documents = Get-ChildItem -Recurse |
                ? Name -like '*.md'
```

Sloppy Variables

- PowerShell defaults to 'sloppy mode'
- Every variable already exists
  - Even before assignment
  - The value is `$null`
- Stay tuned for `Set-StrictMode`

```
"I like this commercial product: $product. It's great!"
```

Typing

- Delimit with brackets
- Add safety and clarity
- Use with
  - .NET type name
  - Accelerators

```
1 # .NET typing
2 [System.IO.DirectoryInfo]$src = Get-Item ./src/
3
4 # Accelerator
5 [int]$slotsAvailable = 12
6 $slotsAvailable = 'none' #Bork!
```

Variable Cmdlets

- Cmdlets manage variables
  - e.g. `Get-Variable`, `Set-Variable`, `New-Variable`
- Useful for
  - Finding out what's available
  - Creating private or constant variables

```
# Find out what's out there
Get-Variable

# Make sure it doesn't get clobbered
New-Variable -Name PI -Value 3.14 -Option Constant
Write-Host $PI
```

Why Read-Only?

- Prevent accidental re-assignment
  - If you can't change it, you can't screw it up
- Two options:
  - **`Constant`** never get changed
  - **`ReadOnly`** can be reassigned with `--Force`

Develop
Intelligence

Comments

Develop
Intelligence

Line Comments

- Used for annotation
- Don't affect any behavior
- Demarcated with **#**
- Use judiciously

```
# Below is the id of the user
$userID = 12
```

Block Comments

- Demarcated with **<# / #>**
- Useful for help system

```
function Get-SphereVolume($radius){
  <# Figures out the volume of a sphere with the
  specified radius.
  #>
  $pi = 3.14
  return 4/3*$pi*[Math]::Pow($radius, 3)
}
```

Example

```
<#
.SYNOPSIS
  Gets pi
.OUTPUTS
  Pi
.Description
  General notes....
#>
function Get-Pi {
  return 3.14
}

Get-Help Get-Pi
```

Develop
Intelligence

Functions

Develop Intelligence

Calling Functions

- Pass arguments (sometimes)
- Get something back (sometimes)
- Syntax
  - Whitespace-delimited arguments
  - No parens

```
1 $now = Get-Date
2 $number = Get-Random -Maximum 10
```

Calling .NET Functions

Develop Intelligence

- Syntax
  - Parens
  - Comma-delimited arguments

```
1 $x = [System.Math]::Sqrt(5)
2 $y = [System.Math]::Pow(2, 16)
```

Defining Functions

- Define with keyword **`function`**
- Optional: exit via keyword **`return`**
- Use the form **Verb-Noun**

```
function Get-Hello{
  return 'Hello World'
}
```

About Verbs

Develop Intelligence

- Use only verbs that already exist
- Verbs come with semantics-- e.g.
  - `New-*` implies a state-change
  - `Get-*` should be safe

```
# Here's the list of approved verbs:
Get-Verb | Sort-Object -Property Verb
```

Examples

Explicit `return`

```
function Get-Sum($A,$B){
    return $A + $B
}
```

Multiple values (implicit)

```
function Get-Lyrics(){
    "Got to be good-lookin"
    "Cause it's so hard to see"
}
```

Develop
Intelligence

Strings

Develop Intelligence

Flavors

- Quotation marks
  - Single-tick means literal
  - Double-tick means interpolated
- Newlines are ok

```
$word = 'thing'
$sameWord = "thing"

$lines = '
I am the very model
Of a modern major general
'
```

Quotes in Strings

Develop Intelligence

- Mix up single ticks with double-ticks
- Escape by doubling-up

```
# No escape needed
$motto = "Don't take stuff."
$slogan = 'Always "remember" to steal.'

# Ticks in strings
$advice = 'Don''t forget the milk.'
$words = "It's impolite to say ""Shut up""."
```

Interpolated

- Double-tick strings
- Expands variables, expressions

```
$givenName = 'joe'
$surname = 'bloggs'
Write-Host "Paging $givenName $surname!"

$pi = 3.14
Write-Host "Pi is: $pi"
Write-Host "Pi^2 is: $($pi*$pi)"

Write-Host "Notepad is at $((Get-Process notepad).Path)"
```

Special Characters

- Use the *grave accent* `
- Newline

```
Write-Host "Processing...`n`nDone!"
```

- Tabs

```
Write-Host "Here are things:"
Write-Host "`tNinja"
Write-Host "`tGoat"
Write-Host "`tTree"
```

Develop
Intelligence

Conditions

Develop Intelligence

Overview

- Syntax:
  - Condition in parens
  - Codeblock in braces
- Keywords: `if`, `switch`

```powershell
if ((Get-Date).DayOfWeek -eq 'Sunday'){
  Write-Host "Tomorrow is trash day."
}
```

Adding Branches

```powershell
$color = '#FFFF00'

if ($color -eq '#FFFF00'){
  Write-Host 'yellow'
}
elseif ($color -eq '#FF0000'){
  Write-Host 'red'
}
else {
  Write-Host 'unknown'
}
```

Equality is Referential

Develop Intelligence

- Equality operator does not compare structure

```
$a = @{id=1}
$b = @{id=1}
$a -eq $b
```

- Goofy for collections

```
$xs = 1,2,3
$ys = 1,2,3
[bool]$areSame = $xs -eq $ys
```

Comparison Operators

- Least intuitive part of powershell
- Everything has a dash
- Examples
  - Equals: `-eq`
  - Greater than: `-gt`
  - Less than: `-lt`

Selected Operators

Operation	In C#	In Powershell
Equals	`==`	`-eq`
Not equals	`!=`	`-ne`
Greater than	`>`	`-gt`
Greater than or equal	`>=`	`-ge`
Less than	`<`	`-lt`
Less than or equal	`<=`	`-le`

Develop
Intelligence

Iteration

Develop Intelligence

Loop Flavors

1. `for`
2. `foreach`
3. `while`
4. `do..while` / `do..until`

foreach

```
1 $names='Rob','Mary','David','Jenny','Chris','Imogen'
2 foreach ($name in $names){
3     "Name of: " + $name
4 }
```

for

- Works just like C family
- Doesn't feel very much like PowerShell

```powershell
# C-flavored
for ($i = 0; $i -lt 10; $i++){
    "Digit: " + $i
}
# Idiomatic pwsh
0..9 | ForEach-Object {"Digit: " + $_}
```

while

- Executes as long as a condition is true

```
1 $i = 0
2 while ($i -lt 3){
3   Write-Host $i
4   $i = $i + 1
5 }
```

Kicking Out

- Quit function with `return`
- Quit loop with `break`

```
$i = 0
while ($true){
  Write-Host $i
  $i++
  if($i -gt 3){
    break
  }
}
Write-Host 'Done'
```

Lab: FizzBuzz

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

Develop
Intelligence

Review

1. Discuss two ways to invoke a function
2. List 3 different comparison operators
3. Name two different loop constructs