W07 Obsługa wyjatków

Obsługa błędów

1. Try-Catch-Finally

W PowerShellu, mechanizm **try-catch-finally** jest używany do obsługi błędów w kodzie. Poniżej znajduje się opis tego, jak używać tych bloków.

```
try {
    # Kod, który może wywołać błąd
}
catch {
    # Kod wykonywany, gdy wystąpi błąd w bloku 'try'
}
finally {
    # Kod wykonywany po zakończeniu bloku 'try' i 'catch', niezależnie
od tego, czy wystąpił błąd
}
```

Try

W bloku try umieszczamy kod, który może spowodować wystąpienie błędu. Jeśli w tym bloku wystąpi błąd, wykonanie kodu przechodzi do najbliższego bloku catch.

Catch

W bloku catch umieszczamy kod, który ma zostać wykonany w przypadku wystąpienia błędu w bloku try. Możesz też złapać określone typy błędów, np.:

Finally

Blok finally zawiera kod, który zostanie wykonany po blokach try i catch, niezależnie od tego, czy wystąpił błąd, czy nie. Jest to miejsce na kod sprzątający, np. zamknięcie pliku czy połączenia z bazą danych.

```
try {
    $result = 10 / 0
}
catch [System.DivideByZeroException] {
    Write-Host "Nie można dzielić przez zero."
}
catch {
    Write-Host "Wystąpił nieznany błąd."
}
finally {
    Write-Host "Operacja zakończona."
```

W tym przykładzie, próbujemy podzielić 10 przez 0, co jest niemożliwe. Kod przechodzi do odpowiedniego bloku **catch**, wyświetlając komunikat o błędzie, a następnie do bloku **finally**, informując, że operacja została zakończona.

2. ErrorAction

Parametr **-ErrorAction** w PowerShell pozwala na określenie, jak dana komenda ma się zachować w przypadku błędu. Jest to bardzo użyteczne, gdy chcemy kontrolować błędy na poziomie pojedynczych poleceń. Oto możliwe wartości dla tego parametru:

- **Continue**: Jest to domyślne działanie. Błąd zostanie wyświetlony, ale skrypt będzie kontynuowany.
- **Stop**: Skrypt lub polecenie zostanie zatrzymane.
- **SilentlyContinue**: Błąd nie zostanie wyświetlony, a skrypt będzie kontynuowany.
- **Inquire**: PowerShell zapyta, co zrobić z błędem (kontynuować, zignorować, itp.).
- **Ignore**: Błąd zostanie zignorowany, a skrypt będzie kontynuowany (dostępne od PowerShell 3.0).

Przykład użycia **-ErrorAction**:

```
Get-Content -Path "nieistniejacy_plik.txt" -ErrorAction Stop
```

3. $ErrorActionPreference

Zmienna **$ErrorActionPreference** jest podobna do **-ErrorAction**, ale działa na poziomie całego skryptu lub sesji, a nie tylko pojedynczego polecenia. Możesz ustawić jej wartość na jedną z opcji dostępnych dla **-ErrorAction** (oprócz **Inquire**), i będzie to globalne zachowanie dla wszystkich poleceń w skrypcie/sesji, które nie mają jawnie określonego parametru **-ErrorAction**.

Przykład użycia **$ErrorActionPreference**:

```
$ErrorActionPreference = "Stop"
Get-Content -Path "nieistniejacy_plik.txt"
```

W tym przypadku, gdy błąd wystąpi, cały skrypt zostanie zatrzymany, chyba że dla konkretnego polecenia jawnie ustawimy inny parametr **-ErrorAction**.

Obie te opcje są bardzo przydatne dla kontrolowania błędów w skryptach PowerShell i pozwalają na dużą elastyczność w zarządzaniu błędami.

Blok **try-catch** w PowerShell obsłuży wyjątki tylko dla tych błędów, które są traktowane jako "terminujące" (terminating errors). Błędy terminujące są to błędy, które zatrzymują wykonanie polecenia i przechodzą do bloku **catch**, jeżeli taki istnieje.

W praktyce, nie wszystkie błędy w PowerShell są błędami terminującymi. Często standardowe błędy są "nie-terminujące" (non-terminating errors) i nie przechodzą do bloku **catch**. Możesz jednak zmienić ich charakter na terminujący, używając parametru **-ErrorAction Stop** dla danego polecenia, lub ustawiając zmienną **$ErrorActionPreference** na **Stop**.

4. $Error

$Error to automatyczna, globalna zmienna PowerShell, która przechowuje historię błędów napotkanych podczas wykonywania bieżącej sesji PowerShell. Jest to kolekcja (System.Collections.ArrayList), która przechowuje obiekty typu [System.Management.Automation.ErrorRecord]. Najnowszy błąd znajduje się zawsze na początku listy – $Error[0].

5. Kluczowe właściwości $Error

- $Error.Count – liczba błędów w historii.

- $Error[0] – najnowszy błąd (ErrorRecord).

- $Error.Clear() – usuwa historię błędów.

- $Error[0].Exception – właściwy wyjątek (np. System.IO.FileNotFoundException).

- $Error[0].Exception.Message – komunikat tekstowy błędu.

- $Error[0].FullyQualifiedErrorId – identyfikator błędu.

- $Error[0].InvocationInfo – informacje o miejscu błędu.

- $Error[0].ScriptStackTrace – stos wywołań.

- $Error[0].CategoryInfo – kategoria błędu.

- $Error[0].TargetObject – obiekt, który spowodował błąd.

- $Error[0].ToString() – pełna reprezentacja błędu jako tekst.

6. Właściwość Message

Właściwość Message jest częścią obiektu Exception, czyli uzyskujemy ją poprzez:

```
$Error[0].Exception.Message
```

Dzięki niej uzyskujemy czysty komunikat tekstowy, bez dodatkowych metadanych.