

Instrukcja try/catch – info

Zobaczysz jak samodzielnie obsługiwać błędy

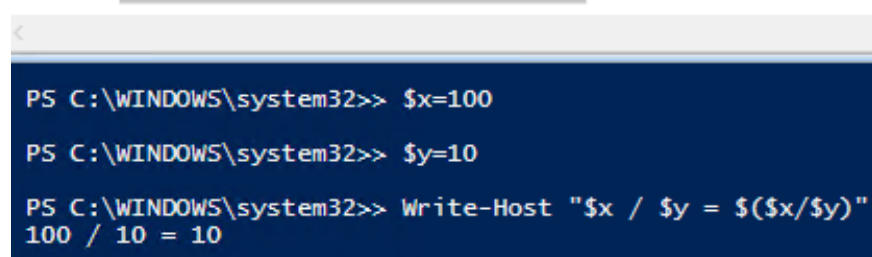
Poznasz znaczenie zmiennej \$Enor

Dowiesz się jak i po co zapisywać błędy do pliku

Zacznijmy od strony programistycznej

Przeanalizujmy poniższy przypadek

```
3 Write-Host "$x / $y = $($x/$y)"
```

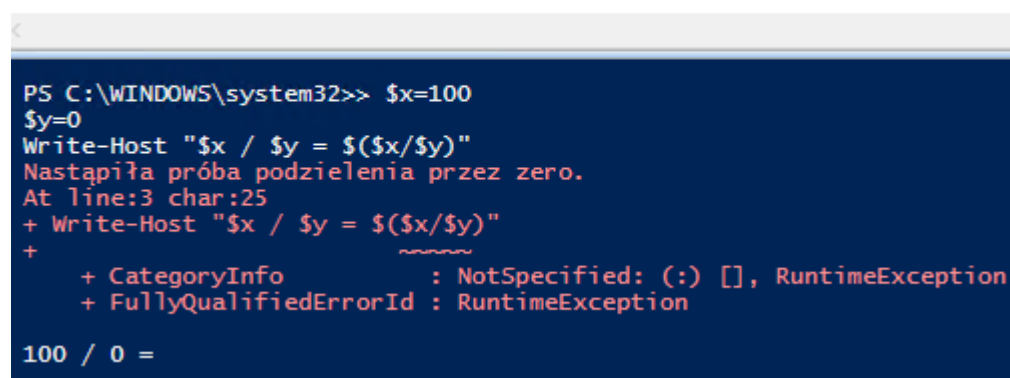


```
PS C:\WINDOWS\system32>> $x=100
PS C:\WINDOWS\system32>> $y=10
PS C:\WINDOWS\system32>> Write-Host "$x / $y = $($x/$y)"
100 / 10 = 10
```

Mamy wyniki dzielenia

Co, gdy wartość zmiennej się zmieni

```
1 $x=100
2 $y=0
3 Write-Host "$x / $y = $($x/$y)"
```



```
PS C:\WINDOWS\system32>> $x=100
$y=0
Write-Host "$x / $y = $($x/$y)"
Nastąpiła próba podzielenia przez zero.
At line:3 char:25
+ Write-Host "$x / $y = $($x/$y)"
+ ~~~~~
+ CategoryInfo          : NotSpecified: (:) [], RuntimeException
+ FullyQualifiedErrorId : RuntimeException

100 / 0 =
```

Może w pewnych okolicznościach chcielibyśmy ukryć ten błąd

Do obsłużenia błędu własnymi metodami można użyć instrukcji try/catch

```
4 try
5 {
6     Write-Host "$x / $y = $($x/$y)"
7 }
8 catch
9 {
10    Write-Host "Przepraszam - Wystąpił błąd"
11 }
12
```

Zasada działania bloku try/catch jest taka, że:

w bloku **try** próbujemy wykonać pewne czynności zdając sobie sprawę, że przy niektórych z nich może dojść do błędów

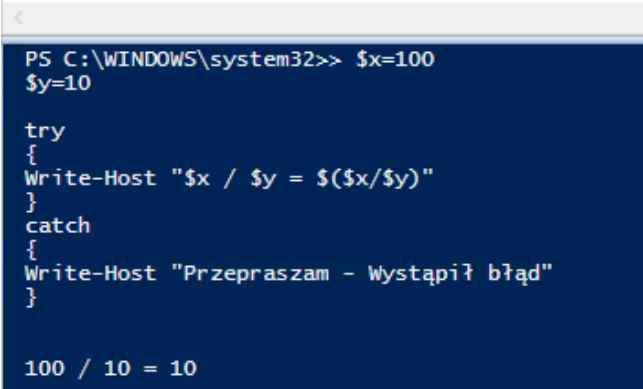
a

w bloku **catch** określamy co należy zrobić, jeżeli do takiego błędu dojdzie

Powyższa instrukcja spróbuje wykonać dzielenie x przez y ale gdyby miało dojść do błędu wyświetli informacje "Przepraszam - Wystąpił błąd"

Sprawdzam wynik przy prawidłowych wartościach

```
1 $x=100
2 $y=10
3
4 try
5 {
6 Write-Host "$x / $y = $($x/$y)"
7 }
8 catch
9 {
10 Write-Host "Przepraszam - Wystąpił błąd"
11 }
12
```



The screenshot shows a PowerShell console window with the following content:

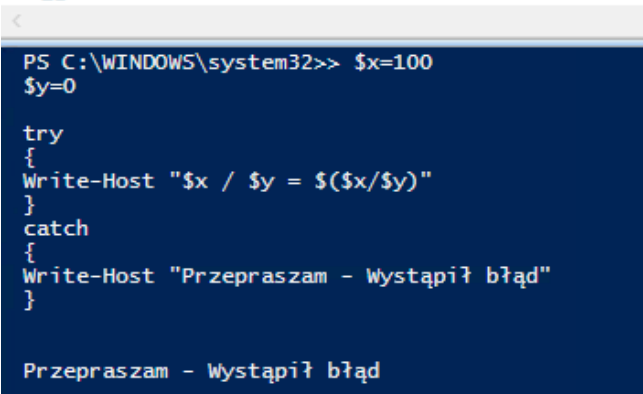
```
PS C:\WINDOWS\system32>> $x=100
$y=10

try
{
Write-Host "$x / $y = $($x/$y)"
}
catch
{
Write-Host "Przepraszam - Wystąpił błąd"
}

100 / 10 = 10
```

Sprawdzam wynik przy nieprawidłowej wartości \$y

```
1 $x=100
2 $y=0
3
4 try
5 {
6 Write-Host "$x / $y = $($x/$y)"
7 }
8 catch
9 {
10 Write-Host "Przepraszam - Wystąpił błąd"
11 }
12
```



The screenshot shows a PowerShell console window with the following content:

```
PS C:\WINDOWS\system32>> $x=100
$y=0

try
{
Write-Host "$x / $y = $($x/$y)"
}
catch
{
Write-Host "Przepraszam - Wystąpił błąd"
}

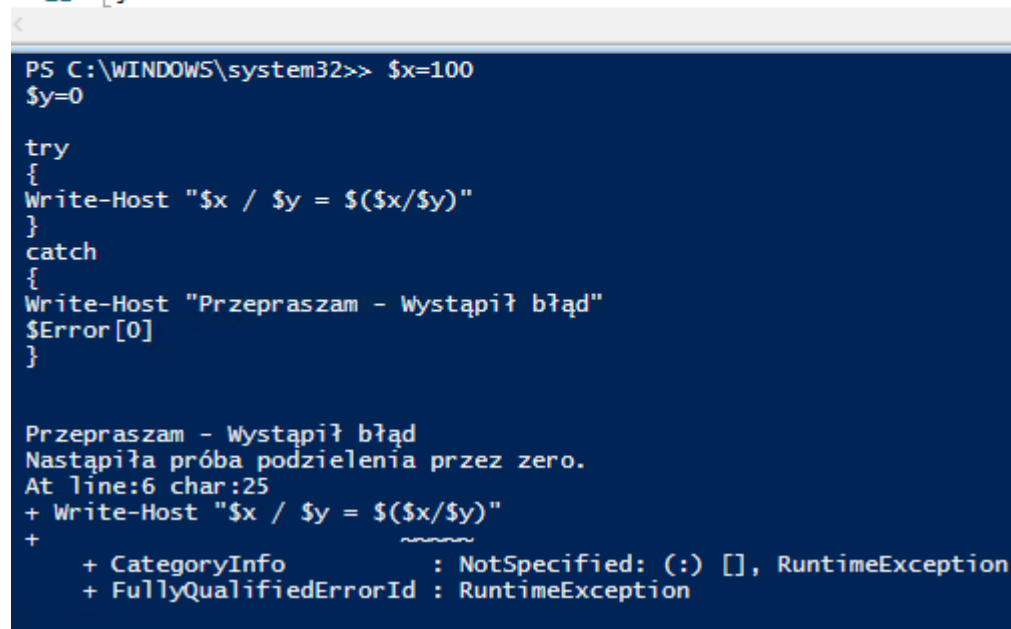
Przepraszam - Wystąpił błąd
```

Zauważ, że nie jest to standardowy komunikat o błędzie wyświetlany czerwoną czcionką tylko nasza własna obsługa tego błędu

Programista PowerShell ma do dyspozycji zmienną `$Error[0]` która jest tablicą zapamiętującą wszystkie błędy do jakich doszło w sesji, ostatni błąd jest pierwszym elementem tej listy.

Pamiętaj że pierwszy element PowerShell nosi numer 0, `$Error[0]` odpowie na pytanie do jakiego błędu doszło ostatnio

```
1 $x=100
2 $y=0
3
4 try
5 {
6 Write-Host "$x / $y = $($x/$y)"
7 }
8 catch
9 {
10 Write-Host "Przepraszam - Wystąpił błąd"
11 $Error[0]
12 }
```



Mamy nasz komunikat, ale również dokładne wskazanie, gdzie doszło do błędu.

Wykorzystamy teraz zmienną `$Error[0]` w bardziej praktyczny sposób.

Powstanie centralne miejsce, w którym będziemy gromadzić błędy z całego skryptu.

```
"-----" | Out-File C:\Users\errors.txt -Append
Get-Date | Out-File C:\Users\errors.txt -Append
$Error[0] | Out-File C:\Users\errors.txt -Append
```

Jeżeli dojdzie do błędu, instrukcja Out-File dopisze kreskę oddzielającą komunikaty o błędach, datę kiedy doszło do błędu i komunikat błędu, czyli wartość zmiennej `$Error[0]`

```

1  $x=100
2  $y=0
3
4  try
5  {
6  Write-Host "$x / $y = $($x/$y)"
7  }
8  catch
9  {
10 Write-Host "Przepraszam - Wystąpił błąd"
11 $Error[0]
12 "-----" | Out-File C:\Users\errors.txt -Append
13 Get-Date | Out-File C:\Users\errors.txt -Append
14 $Error[0] | Out-File C:\Users\errors.txt -Append
15 }

```

Trzy razy uruchamiam nasze polecenia i mam nadzieję, że każde uruchomienie wprowadzi informacje do pliku C:\Users\errors.txt

Sprawdzamy

```

PS C:\WINDOWS\system32>> Get-Content C:\Users\errors.txt
-----

czwartek, 30 września 2021 22:56:08 ●

Nastąpiła próba podzielenia przez zero.
At line:6 char:25
+ Write-Host "$x / $y = $($x/$y)"
+ ~~~~~
+ CategoryInfo          : NotSpecified: (:) [], RuntimeException
+ FullyQualifiedErrorId : RuntimeException

-----

czwartek, 30 września 2021 22:56:09 ● ●

Nastąpiła próba podzielenia przez zero.
At line:6 char:25
+ Write-Host "$x / $y = $($x/$y)"
+ ~~~~~
+ CategoryInfo          : NotSpecified: (:) [], RuntimeException
+ FullyQualifiedErrorId : RuntimeException

-----

czwartek, 30 września 2021 22:56:10 ● ● ●

Nastąpiła próba podzielenia przez zero.
At line:6 char:25
+ Write-Host "$x / $y = $($x/$y)"

```

Zgadza się

W instrukcji try / catch / finally blok finally jest opcjonalny.

Zobaczyłeś jak samodzielnie obsługiwać błędy

Poznałeś znaczenie zmiennej \$Error

Wiesz się jak i po co zapisywać błędy do pliku