PowerShell → składnia języka

WCZYTYWANIE TEKSTU

`read-host`

-prompt: `read-host -prompt "Wpisz swoje imię"`

→ flaga `-prompt` nie jest konieczna!

▼ Tajne wpisywanie, np. do haseł (`-AsSecureString`)

Bardziej skomplikowane niż -s w bashu, poza * we wpisywaniu dane są przechowywane głęboko, nie w zmiennej.

```
$haslo_wpisane = Read-Host "Podaj haslo: " -AsSecureString
#użytkownik nie widzi wpisywanego hasła, tylko *
[string]$haslo_wpisane_jawnie = [System.Net.NetworkCredential]::new('', $haslo_wpisane).Password
#z uzyciem wbudowanej klasy .net uzyskujemy wartosc hasla
```

WYPISYWANIE TEKSTU

`write-host` "tekst" → wypisanie *tekst* → informacja dla użytkownika

`write-output` "tekst" → wypisanie *tekst* → wyniki do przetwarzania, można je przekierować

`write-host $zmienna` → wypisuje zmienną

ZMIENNE

`$kotlet = "Kotlety w promocji"` (spacje bez znaczenia) → dodanie wartości do zmiennej

`$kotlet` → *Kotlety w promocji* → odwołanie się do zmiennej

Domyślnie typ zmiennych jest dodawany automatycznie, ale możemy to wymusić:

`[int]$wiek=14`

Możemy też sprawdzić, jaki typ danych ma zmienna:

`$zmienna.GetType().name`

Przyporządkowanie wielu rzeczy na raz:

`$i = $j = $k = 0`

`$name, $color, $date = "Mikolaj", "????", (Get-Date).DateTime`

*samo `(get-date)` by zadziałało, ale nie będzie to obiekt, a nie tekst, `.datatime` zwraca tekst

---

DZIAŁANIA ARYTMETYCZNE (STRINGI TEŻ)

`5+5` → 10

`$x=10` → `$x % 3` → 1

`$i++` / `--`

`$x=3` → `$x *= 3` → 9

*spacje bez znaczenia, wszystko inne na logike

`"miko" + "laj"` → mikolaj

`"ha" * 3` → hahaha

PORÓWNANIA → zwracają `True` / `False`

Operator	Rozwinięcie	Opis (PL)
`-eq`	**Equal**	Sprawdza, czy wartości są identyczne (`a == b`).
`-ne`	**Not Equal**	Sprawdza, czy wartości są różne (`a ≠ b`).
`-gt`	**Greater Than**	Sprawdza, czy lewa wartość jest większa niż prawa (`a > b`).
`-lt`	**Less Than**	Sprawdza, czy lewa wartość jest mniejsza niż prawa (`a < b`).
`-ge`	**Greater or Equal**	Sprawdza, czy lewa wartość jest większa lub równa prawej (`a ≥ b`).
`-le`	**Less or Equal**	Sprawdza, czy lewa wartość jest mniejsza lub równa prawej (`a ≤ b`).

OPERATORY LOGICZNE

Operator	Opis (PL)
`-and`	Zwraca `True` , jeśli **oba** warunki są `True` .
`-or`	Zwraca `True` , jeśli **przynajmniej jeden** warunek jest `True` .
`-xor`	Zwraca `True` , jeśli **dokładnie jeden** warunek jest `True` , ale nie oba.
`-not` / `!`	Neguje wartość logiczną (`True` → `False` , `False` → `True`).

np. `5 -gt 10 -or "kot" -eq "kot"` → prawda, bo kot=kot

np. `5 -gt 10 -and "kot" -eq "kot"` → fałsz, 5 nie jest > 10

np. `-not (5 -gt 10 -and "kot" -eq "kot")` → prawda, bo `-not` / `!` zwraca przeciwność

SKRYPTY

- notatnik/PowerShell ISE (terminal + pole do skryptów jednocześnie)

Uwaga! Na początku odpalanie skryptów może nie działać, trzeba zmienić ustawienie:

`Set-ExecutionPolicy Unrestricted -Scope CurrentUser`

- odpalanie → `./nazwa_skryptu.ps1` / `ścieżka/do/nazwa_pliku.ps1`

ARGUMENTY DOMYŚLNE

`param([int]$a, [string]$b)` , `Write-Host $a` → *69*

▼ Parametr konieczny

```
param(
   [Parameter(Mandatory)]
```

```
    [string]$Plik
)
#Skrypt się nie odpala, gdy nie podano argumentu.
#moze być konieczne w używaniu help
```

`./nazwa_skryptu.ps1 69 kotlet`

HELP

▼ Skrypt z instrukcją użytku (uzupełniamy ważne rzeczy wg naszego programu)

```
<#
.SYNOPSIS
Krótki opis co robi funkcja/skrypt – maks 1-2 linie.

.DESCRIPTION
Szczegółowy opis działania – co robi, jak działa, co zwraca, co modyfikuje, kiedy używać.

.PARAMETER NazwaParametru
Opis do czego służy dany parametr. Jeden `.PARAMETER` na jeden parametr.

.INPUTS
Typy danych, które można podać przez pipeline. Jeśli brak – wpisz „None.”

.OUTPUTS
Typ danych, które funkcja/skrypt zwraca (np. `[string]`, `[int]`, `None`).

.EXAMPLE
Przykład użycia – pokazujesz komendę i ewentualnie co ona zwróci.
Dla kazdej flagi inny przyklad (a jak sie da bez to tez)

.LINK
Link do dokumentacji, innych cmdletów lub plików pokrewnych. Może być też `Get-Help Something`.
#>
```

Przykład dla pliku z jednym parametrem -plik:

```
<#
.SYNOPSIS
Determinates whether a certain file is safe.

.DESCRIPTION
1. Sends request with file hash to virusustotal and checks if NotFoundError was received.
2. If no, it determinates safety by received data.
3. If NotFoundError was received, it sends file to VirusTotal and checks again.
Attention! Sending file to VirusTotal works only on powershell 7+!

.PARAMETER Plik
Specifies path to the file we want to check

.INPUTS
  None. You can't pipe objects to Update-Month.ps1.
```

```
.OUTPUTS
  Information whether the file is safe to use.

.EXAMPLE
  PS> Path\to\file\ApiRequest-VirusTotal.ps1 -plik C:\Reports\2009\January.csv

#>

param(
    [Parameter(Mandatory)]
    [string]$Plik
)
```

Odwołanie się do tego:

`Get-Help C:\sciezka\do\pliku\plik.ps1 -Full`

TABLICE/LISTY

Tworzenie listy:

```
Sposób1:
$my_arr = 25, "Codecademy", 1, $False
$my_arr (wynik:)
25
Codecademy
1
False

Sposób2:
$colors = @("cyan", "orange", "blue")
$colors (wynik:)
cyan
orange
blue

Sposób3:
$arr_3 = @(                    # Multi-line Array
   "Uno"
   "Dos"
   "Tres"
)
*brak róznicy funkcjonalnej

Sposób4 (z gotowych danych):
$dane = "jeden-dwa-trzy"
$lista = $dane -split "-"
$lista[1] → dwa
```

Indeksowanie listy:

```
#+- jak w każdym języku
```

```
$colors = "red", "yellow", "black", "blue"
#.         0      1        2      3

PS > $colors[2]    #odwołanie się do pozycji
black

PS > $colors[1] = "brown"   #zmiana elementu
PS > $colors
red
brown
black
blue


PS > $colors[0,2]   #można wypisać więcej indeksów na raz, nawet Python tak nie potrafi
red
black


   PS > $colors[1..3]   #zakres jako indeks
brown
black
blue

PS > $colors[2..1]   #odwrotny zakres jako indeks (np. sztuczka na odwrócenie listy)
black
brown


PS > $colors[-1]   #piewszy element od końca (-2 to byłby drugi)
blue

UWAGA!
Iteracja, ForEach operuje na każdym elemencie na raz!
PS > $colors.ForEach({ "Kolorek: " + $PSItem })
Kolorek: red
Kolorek: yellow
Kolorek: black
Kolorek: blue

$colors.ForEach({ "Kolorek: " + $_ }) zadziała tak samo
( $PSIItem = $_, jest to odwołanie do bieżącego elementu w iteracji)
```

Operacje na listach:

`$lista1 + $lista2` → połączenie obu list (najpierw pierwsza)

`$lista1 * 2` → z `1,2` robi się `1,2,1,2`

`[String[]]$fruits = "apple", "banana", "kiwi"` → wymuszenie typu danych na liście (tu: string)

```
$fruits = "apple", "banana", "kiwi"

# -contains
$fruits -contains "banana"  # True
```

```
# -notcontains
$fruits -notcontains "orange"  # True

# -in
"banana" -in $fruits  # True

# -notin
"orange" -notin $fruits  # True
```

`-join "separator"` → zmienia listę na jednego stringa z danym sepratorem:

`$fruits = "apple", "banana", "kiwi"`

`$string = $fruits -join " & "`

`$string` → *apple & banana & kiwi*

`$fruits | get-random` → losowy element z listy

INSTRUKCJA WARUNKOWA (IF, switch)

IF

```
if (warunek) {
Write-Host "Warunek jest spełniony"
} elseif (warunek2){
Write-Host "Warunek2 jest spełniony"
} else {
   Write-Host "Żaden warunek nie jest spełniony"
}

#spacje/wcięcia +- bez znaczenia
```

SWITCH

`Switch` zawsze sprawdza wszystkie warunki, a w `if` .. `elseif` .. elseif nie będzie sprawdzany, gdy if był prawdziwy.

```
switch ($zmienna)
"kot" { #aby porównać to z nawiasu przy switchu do czegoś, po prostu piszemy to przed warunkiem
Write-Host $zmienna="kot"
}
{warunek} {
output
}
default { #default = else, jak nic wyżej nie zostało spełnione to to działa
output
}


$var = -2 #lub cokolwiek innego
switch ($var){
 5 {
  Write-Host "$var = 5"
 }
 {$_ -lt 0} {
  Write-Host "$var < 0"
 }
```

```
  {$PSItem -gt 0} {
    Write-Host "$_ > 0"
  }
  default {
    Write-Host "Wszystkie warunki są fałszywe"
  }
}
```

PĘTLE (FOR, FOREACH, WHILE + stopowanie)

For i Foreach

```
#for dziala tak jak w c++

for ($i=0; $i -lt 5; $i++){
Write-Host $i
}
wynik:
0
1
2
3
4

for ($i=0; $i -lt 5; $i++){Write-Host $i} → identyczna pętla, entery/spacje/wcięcia bez znaczenia
--------------------

#foreach to taka pythonowa wersja fora
$recipe = "flour", "sugar", "salt", "oil"

foreach ($ingredient in $recipe){
Write-Host "My recipe includes" $ingredient
}
wynik:
My recipe includes flour
My recipe includes sugar
My recipe includes salt
My recipe includes oil
```

WHILE

```
while (warunek) {
   output
   i++ # lub inna operacja modyfikująca licznik
}
--------------------

$i=1
while ($i -lt 6){
   write-host $($i*2)
   $i*=2
}
output:
2
```

```
4
8

$i = 1; while ($i -lt 6) { Write-Host "$($i * 2)"; $i *= 2 } # ; zamiast entera
```

DO-UNTILL, DO-WHILE

```
do {
body
} while (warunek)
#body wykonuje się, DOPÓKI warunek JEST spełniony
#różnica między while → jeżeli warunek jest spełniony od początku,
   #to while się w ogóle nie odpali, a tu jest jedna iteracja
-------------------
do {
body
} until (warunek)
#body wykonuje się, DOPÓKI warunek NIE JEST spełniony (aż będzie)
-------------------

$number=9
do {
$guess = Read-Host -prompt "Zgadnij mój numer 0-10"
} until ($guess -eq $number)

write-host "pętla się zakończyła, zgadłeś"

PS C:\Users\mikol> ./test.ps1
Zgadnij mój numer 0-10: 1
Zgadnij mój numer 0-10: 5
Zgadnij mój numer 0-10: 9
pętla się zakończyła, zgadłeś
```

`break` → zatrzymuje działanie pętli, wykonuje się kod po niej

`continue` → zatrzymuje iteracje pętli, wykonuje się następna

FUNKCJE

```
function greet { #definciowanie funkcji
  Write-Host "Hello, there!"
}

greet #odwołanie się do funkcji [bez ()]


----------------------------------------
#funkcja z parametrami
      #zamiast funkcja(parametry) definiujemy je w ciele funkcji

function ulubione {
   param($slowo, $liczba) #zdefiniowaliśmy 2 parametry wejściowe
   Write-Host "Twoje ulubione słowo to $slowo, a liczba to $liczba"
}
```

```
ulubione "kotlet" 69 #wywołanie funkcji

#output:
Twoje ulubione słowo to kotlet, a liczba to 69


-------
#a co jeżeli funkcja przyjmuje parametry, a my ich nie podamy?
      #wtedy $parametr jest pusty, ale możemy zrobić tak
      #żeby przy braku wpisanej informacji zwrócił określoną liczbę

param($slowo, $liczba)
#output bez zadeklarowania argumentów:
Twoje ulubione słowo to , a liczba to

#dodajmy default parameters
param($slowo="majonez", $liczba=1)
#output bez zadeklarowania argumentów:
Twoje ulubione słowo to majonez, a liczba to 1
```

OBSŁUGA WYJĄTKÓW (BŁĘDY)

`-ErrorAction` :

- `Continue` (domyślnie) – pokazuje błąd i kontynuuje
- `SilentlyContinue` – ignoruje błąd i nic nie pokazuje
- `Stop` – traktuje błąd jak wyjątek i przerywa wykonanie
- `Inquire` – pyta użytkownika, co robić
- `Ignore` – całkowicie ignoruje błąd (nie pokazuje nawet w `$Error`)

Przykład:

`Get-Item "C:\nie_ma.txt" -ErrorAction SilentlyContinue  # nie pokaże błędu`

`Try-Catch`

```
# Podstawowy szablon:
try {
    # kod, który może rzucić błąd
} catch {
    # obsługa błędu
}

# Przykład:
try {
    Get-Item "C:\nieistnieje.txt" -ErrorAction Stop
    Write-Host "Plik znaleziony"
} catch {
    Write-Host "Wystąpił błąd: $_"
}

# _ = aktulny błąd

#Filtrowanie wg błedu (w catch):
```

```
    if ($_ -match "NotFoundError") {
        Write-Host "Plik nieznaleziony w bazie."
```

HASHOWANIE TEKSTU

Hash pliku jest uzyskać łatwo, ale hashu tekstu już nie.

▼ Najprostsza funkcja jaką znalazłem:

```
function sha256sum {
    param([string]$text)
    $bytes = [System.Text.Encoding]::UTF8.GetBytes($text) #tekst → tablica bajtow
    $sha256 = [System.Security.Cryptography.SHA256]::Create() #tworzymy algorytm hashujacy
    $hash = $sha256.ComputeHash($bytes) #hashujemy
    return [BitConverter]::ToString($hash) -replace "-" #zmienia tablice bajtow na format szesnastkowy
}
```

API (itp)

`Invoke-RestMethod -Uri "link" -Method <get/post..>` → wysyła zapytanie do strony

`Invoke-WebRequest -Uri "link"` → pobiera dane ze strony

NBP

https://api.nbp.pl/

Wszystko dobrze opisane w tym linku, przykład:

```
$dane = Invoke-RestMethod -Uri "https://api.nbp.pl/api/exchangerates/rates/A/USD/" -Method GET
Write-Host $dane.rates.mid

#^kurs konkretnej waluty wpisanej do requsta
# *jezeli chcemy dac zmienna, to musimy to zrobic uzywając +
("https://api.nbp.pl/api/exchangerates/rates/A/" + $currency + "/")
```

WIRUSTOTAL

▼ mój klucz api

c0fc81f751e1b2de6437f94ca97d9463f2ae822d7995be5ee4e9eb4455def039

```
#wysylanie hashu do bazy wirustotal
function get{
    param($hash, $apiKey)
    Invoke-RestMethod -Uri "https://www.virustotal.com/api/v3/files/$hash" `
        -Headers @{ "x-apikey" = $apiKey } -Method Get
# ' rozdziela komende na 2 linie
#method okresla jaki to rodzaj zapytania (inne to np post (wyslanie czegos) lub delete (usunieie)
}
```

```
#dodawanie pliku do bazy wirustotal
function post{
    param($plik, $apiKey)
    $POST = Invoke-WebRequest -Uri "https://www.virustotal.com/api/v3/files" `
        -Method Post `
        -Headers @{ "x-apikey" = $apiKey } `
```

```
      -Form @{ file = Get-Item $plik }
}
```

NEWSY (sprawdza wszystkie strony/języki) → https://newsapi.org/

▼ moje api

`$api="7facd67c253649099e8092b1745feccb"`

Zapytanie:

`https://newsapi.org/v2/everything?` **`q=tesla`** `&from=2025-03-29&sortBy=publishedAt&apiKey=$api`

`q=tesla` → wszystkie newsy, które mają w sobie "tesla"

`from=2025-03-29` → newsy nie starsze niż 29.03.2025

Do filtrowania danych musimy przekonwertować *JSONA* na obiekt w PowerShellu: `ConvertFrom-Json`

▼ Przykład

```
<#
.SYNOPSIS
Pobiera artykuły z newsapi.org na podstawie podanego słowa kluczowego.

.DESCRIPTION
Pobiera artykuły z newsapi.org na podstawie podanego słowa kluczowego.
Wymaga podania słowa kluczowego oraz liczby artykułów do pobrania.

.INPUTS
None.

.OUTPUTS
String.
Zwraca tytuł i URL artykułów.

.EXAMPLE
PS> Path\to\file\find-news.ps1
Podaj słowo kluczowe do wyszukiwania wiadomości:: trump
Podaj ile maksymalnie artykółów chcesz pobrać:: 2

#>


$api="7facd67c253649099e8092b1745feccb"
$KeyWord = Read-Host "Podaj słowo kluczowe do wyszukiwania wiadomości:"
$count = Read-Host "Podaj ile maksymalnie artykółów chcesz pobrać:"

#pobranie danych ze strony newsapi.org
$response = Invoke-WebRequest -Uri ("https://newsapi.org/v2/everything?q=" + $KeyWord + "&sortBy=publish

#tworzymy obiekt powershell z jsona
$responseContent = $response.Content | ConvertFrom-Json

Write-Host "Znalezione artykuły:"
$responseContent.articles | Select-Object -First $count title, url
```

API SHODAN

```
iex (New-Object Net.WebClient).DownloadString("https://gist.githubusercontent.com/darkoperator/9378450/raw/7244d3db5c0234549a018faa41fc0a2af4f9592d/PoshShodanInstall.ps1")
```

^instalowanie

```
Set-ShodanAPIKey -APIKey 238784665352763857288393 -MasterPassword (Read-Host -AsSecureString)
```

kulfon123 ← moje haslo

- ▼ bug fix

  W powershelu 5.cos:

  ```
  PS C:\Users\mikol> Import-Module "$HOME\Documents\WindowsPowerShell\Modules\Posh-Shodan\Posh-Shoc
  PS C:\Users\mikol> Set-ShodanAPIKey -APIKey 238784665352763857288393 -MasterPassword (Read-Host -A
  *haslo*
  ```

https://developer.shodan.io/api
^lista poleceń

- ▼ Mój klucz:

  yeO2xSEUI4ZUaeww0WmHDSkc6xuhISjn