PowerShell

ANR

PowerShell Programming
Progress Bars and Functions

Preparation

Prepare a PowerShell script with extension .ps1 for each exercise in this series. Execute each script by navigating to the folder containing it. Depending on your system configuration use may need administrator privileges to do so.

Exercise 1

Implement a function counting from 1 thru 50 that displays progress in percent while walking thru the range using a progress bar. The progress bar shall be updated every 100 milliseconds. The information text shall be „Searching…“.

Exercise 2

a. Implement a function listing all files (not folders!) in the directory referenced by *$env:windir*. A progress bar shall be updated every 100 milliseconds and inform the user about the progress of the operation. The information text shall be „*Finding file*“

b. Under GNU/Linux list all folders (not files!) in the user's home directory. The progress bar shall work as described.

Exercise 3

Implement the following functions in PowerShell either directly at the command prompt or in a script. The input value can at first be a hard-coded variable in the script. Then modify your program in a way that it takes a command line argument as parameter for the function.

a. Absolute value[1] *abs*

b. Sign function[2] *sgn*

c. Factorial function[3] *fac*

d. (Bonus) Binomial coeffizient[4] *bk*

e. (Bonus) Ackermann function[5] *ack*

---

[1] https://en.wikipedia.org/wiki/Absolute_value

[2] https://en.wikipedia.org/wiki/Sign_function

[3] https://en.wikipedia.org/wiki/Factorial

[4] https://en.wikipedia.org/wiki/Binomial_coefficient

[5] https://en.wikipedia.org/wiki/Ackermann_function

Exercise 4

a. Create a ps1-file containing a function *Delete-FileTreeItems*, that deletes all files in a file system tree (for testing just list the files instead of actually deleting them). The file structure should be kept intact, i.e. no folder (not even an empty one) is deleted.

b. Add a parameter *$root* of type *string* to your function, that indicates the root oft he file system tree to be deleted.

Exercise 5

Create a file *GroupMembership.ps1* containing a function *Get-GroupMemberships*, that takes a user name as parameter and lists all (local) groups which is user is a member of.

Exercise 6

a. Create a script *MyFunctions.ps1* that prints the text *Loading my functions* when executed. After each of the following tasks test your functions with some useful input data.

b. Add a new function *Get-MyProcess* to the script that lists all processes whose name begins with ‘s‘. The output shall only contain the properties *ID*, *ProcessName* and *CPU*. The output must be sorted by property *CPU* in descending order.

c. Copy the function and add another version of it to the script. Name it *Get-MyProcess2.* Modify the function as follows:

   - The function takes a parameter *$FilterName*, that decides which processes are printed. Default value: processes whose names begin with the letter ‘s‘.
   - Let a switch parameter *$AsList* control whether the output should be formatted as a table or as a list.

d. Create a function *Get-MyProcess3*, that meets the following requirements:

   - Before the main part the current date must be printed.
   - The main part shall execute the function *Get-MyProcess2*.
   - After processing the text DONE shall be printed in black colored font on a white background.

e. Create a brief documentation for the function *Get-MyProcess2* and add it to the function definition in the script.

Exercise 7 (Bonus)

Only under GNU/Linux.

Create a function *New-LocalLinuxUser*, that adds a user account via PowerShell. The function takes two parameters of type *string* fort he user name and password respectively.

Exercise 8 (Bonus)

Only under GNU/Linux.

a. Create a function *Upgrade-System*, that updates the sources and updates all packages.

b. Add a switch parameter *-Force* to the function that forces updating of packages that have been kept back otherwise.