



Next Publishing レビュー

目次

まえがき	5
本書の想定読者	5
PowerShellについて	5
本書におけるコード部分の表記	6
推奨動作環境・動作前の下準備	7
本書サンプルコードのダウンロード	8
免責事項	8
 第1章 PowerShellの文法	9
1.1 命名規則	9
1.2 Powershellにおける予約語	10
1.3 基本的な文字列の扱い	11
1.4 コメント	11
1.5 変数	11
1.6 データ型	12
1.7 型変換(キャスト)	16
1.8 演算子	17
1.9 範囲演算子	20
1.10 ビット演算子	21
1.11 エスケープシーケンス	22
1.12 条件分岐	23
1.13 繰り返し処理	23
1.14 関数	24
1.15 スクリプトブロック	25
1.16 クラス	25
1.17 例外処理	27
1.18 数学関連の演算	28
1.19 フィルタ・パイプライン	29
1.20 リダイレクト	29

第2章 PowerShell仕様・コマンドレットの概要	32
2.1 実行ポリシー	32
2.2 コマンドレットの概要	33
2.3 自動変数	36
2.4 特殊フォルダ	36
2.5 各種ユースケースへの対応	37
第3章 基本サンプル集	45
3.1 基本サンプル1: fizzbuzz問題	45
3.2 基本サンプル2: クイックソート	46
3.3 基本サンプル3: 数当てゲーム	48
3.4 基本サンプル4: 関数積分の近似値計算	50
3.5 基本サンプル5: 円周率の近似値計算	53
第4章 応用サンプル集	60
4.1 応用サンプル1: chocolateyによるパッケージインストール	60
4.2 応用サンプル2: 複数PDFファイルの連結	61
4.3 応用サンプル3: フォルダ内画像のPDFファイル化	62
4.4 応用サンプル4: PDFファイルページの画像化	64
4.5 応用サンプル5: 直近ダウンロード済みファイルのアーカイブ処理	65
4.6 応用サンプル6: 画像のリサイズ・拡張子変換	66
4.7 応用サンプル7: 音声ファイルの連結処理	69
4.8 応用サンプル8: 音声ファイルの分割処理	71
4.9 応用サンプル9: 動画の音声データ取得・拡張子変換	73
4.10 応用サンプル10: yt-dlpによる動画ダウンロードの自動化	75
4.11 応用サンプル11: Webページ上画像ファイルの一括ダウンロード	77
4.12 応用サンプル12: 連続スクリーンショット取得・キー押下自動化	81
4.13 応用サンプル13: URL情報のQRコードへの変換	89
4.14 応用サンプル14: QRコードからのワンタイムパスワードの読み出し	91
4.15 応用サンプル15: Yahoo!ニュースヘッドライン抽出ツール(CUI)	95
4.16 応用サンプル16: Yahoo!路線検索結果表示ツール(CUI)	98
4.17 応用サンプル17: 英単語検索・辞書データ作成ツール(CUI/SQLite3使用)	103
4.18 応用サンプル18: モールス信号	108
4.19 応用サンプル19: ライフゲーム	111
4.20 応用サンプル20: テトリス	120

付録 A 巻末付録.....	136
A.1 参考文献・URL.....	136

本書を手にとっていただきありがとうございます。読者の方の中にはPowerShellの存在は知っているけれども、その中身についてはあまり知らないという方が少なくないのではないかと思います。本書ではPowerShellの文法やコマンドレット、そして筆者が描き下ろしたコードサンプル(レシピ)を紹介しており、PowerShellの力を体感してもらうことができます。

近年は数多くの新興IT企業が台頭し、そこで働くエンジニアにはMacBookが支給されることが多くなっています。それに従い、必然的にmacOSでの開発業務が主流となっています。一方でWindowsやその周辺技術はOSの中では大きなシェアを誇っているにもかかわらず、こうした新興のWeb系IT企業に所属するエンジニアからはPowerShellの存在は顧みられることが少なかったと言えるでしょう。サティア・ナデラ氏がCEOに就任する以前の、Microsoftの『悪の帝国』的なイメージもこの傾向に拍車をかけていたと思われます。

しかし、実のところ業務でmacOSしか触らないエンジニアであっても、Windows環境のPCを持っていない人は少数派ではないかと思います。特にSteamやOrigin, EpicGamesのようなプラットフォームでPCゲームを楽しむような人であればWindowsの動作環境は必ず必要となるからです。また、Windows上でしか動かないようなフリーツールも数多くあります。そうした皆さんが手元に持っているWindows環境をさらに活かす上で、PowerShellが役立つはずです。本書を通してPowerShellの潜在能力、そして.NETフレームワークの威力について理解を深めていただければ幸いです。

本書の想定読者

本書は一定程度のプログラミング経験がありbash等のシェル環境に習熟しているITエンジニアや、大学で理系分野での学習経験がある方々を読者として想定しています。すでにほかのプログラミング言語のデザインはある程度知っている方がPowerShellとはどのような書き方をすればよいかを手短かに知ることができることを目的として執筆しました。プログラミング経験が少ない人や初心者の方を想定したような書き方をとっておりませんのでご注意ください。

本書では文法やコマンドレットの説明部分については冗長に細かく説明することを極力避け、シンプルな記述にとどめました。また、すでに開発者としての経験を積んだ人が手早く言語全体の概観をつかめるような作りとすることを意識しました。コードサンプルについてはできるだけコメントを丁寧に書き、読者の皆さんが今後PowerShellを日常遣いするうえでのヒントが得られるような構成にしました。

PowerShellについて

PowerShellを使う意義とは？

PowerShellには以下のような特徴があります。

- ・ 標準のWindows環境において利用可能

- ・.NETフレームワーク機能の利用可能
- ・コマンドプロンプトよりも洗練された言語体系

PowerShellはWindows上において特別な環境を準備しなくても動作させることができます。別途インストール・セットアップの必要がないため、ほかのWindows環境へのPowerShellスクリプトの移植は容易となります。さらに.NETフレームワークの機能呼び出すことができ、シェル環境であるにもかかわらず比較的高度な処理をさせることもできます。かつてのMS-DOS時代からの技術的負債を多く抱えたコマンドプロンプトと比較しても、より洗練され一貫性を増した言語体系になっています。

あくまでシェル環境であるため、本格的なプログラミング言語の代替を果たすことはできません。ただ、.NET機能との連携に大きな強みをもっていることもあり、実際にPowerShellスクリプトを書いてみるとLinux/macOS環境で良く使われるbashと比べて多機能ぶりを実感できることでしょう。本格的なIDEを使って本格的に開発するほどではないけれど、日常づかいのためのちょっとしたツールが欲しいようなときに、PowerShellは有力な選択肢となりえます。

PowerShellのセキュリティ

ただし、その多機能ゆえにPowerShellスクリプトはハッキング・クラッキングの手段として使われることが数多くあります。Windowsの標準機能として提供されているため、悪意ある攻撃者からすれば脅威として探知されることなく攻撃するのがより容易であるからです。そのため通常のPowerShellではデフォルト設定でセキュリティポリシーが比較的厳しく設定されており、ポリシーを変更しないとスクリプトの実行ができなくなっています。

PowerShellスクリプトを使う際には、素性の分からないインターネット上の際とから取得してそのまま実行するなどは避けて、信頼できる発行元によるスクリプトかどうかを確かめることに加えて中身の処理についてもある程度知っておく必要があるでしょう。

本書におけるコード部分の表記

本書においてPowerShellにおけるコマンドラインの入力内容を示すときには先頭に"PS >"の表記を付記します。実際のPowerShell環境においてはカレントパスも同時に表示されますが、本書においてはすべて省略するものとします。

```
# これはソースコードについての解説コメントです
#
# PowerShellサンプルコードを記述します。
#
```

```
# これはPowerShellでの動作・実行結果の解説コメントです
PS > (ここにコマンドへの入力内容を書きます)
```

推奨動作環境・動作前の下準備

本書コードの検証済みバージョン

本書に記載されているコードは次の PowerShell バージョンにて、動作確認を行っています。ただし、本書に登場するサンプルコードについてはすべての PowerShell バージョンや Windows 環境での動作を保証するものではないことをご理解ください。

- ・ 検証 PowerShell バージョン
 - 5.1 (Windows10 での利用バージョン)
 - 7.4, 7.5 (Windows11 での利用バージョン)

Ver5.1 での検証作業も実施しておりますが、古いバージョンであるため環境によってはサンプルが想定通りに動作しない可能性があります。可能であれば Ver7 を利用いただくようお願いいたします。

文字コードについて

Windows において、PowerShell のバージョン 6.0 以降においては "UTF-8" を文字コードとして使用します。ただし、バージョン 5 以前は BOM 付きの UTF-8 となっていますのでご注意ください。

Windows Terminal の利用

本書においては Windows Terminal 上でサンプルコードを動作させることを推奨します。より洗練された UI・機能を持ち、PowerShell・コマンドプロンプトの実行ができます。既存の PowerShell ISE に存在する文字コード (UTF-8 の扱い) 部分の問題、日本語が上手く表示されない問題が解消されています。Windows Store から手に入れますので、試してみてください。

- ・ Microsoft Store - Windows Terminal
 - URL: <https://apps.microsoft.com/detail/9N0DX20HK701>

プロジェクトフォルダの設定

ここでは、Documents フォルダ以下に PowerShellSample という名前のフォルダを作ってこの中にプロジェクトを構成する例を挙げておきます。もちろん各自好きなのところに PowerShell 用のフォルダを作ってもらってかまいません。

```
# 新しいフォルダの作成および移動
PS > cd $env:USERPROFILE/Documents
PS > mkdir PowerShellSample
PS > cd PowerShellSample

# テンプレートに沿ってプロジェクトファイルを作成する
# Note: dotnet コマンドによるライブラリのインストールにはプロジェクトファイルが必要なため
PS > dotnet new console
```

本書サンプルコードのダウンロード

サンプルコードは筆者のGitHubリポジトリからダウンロード可能です。本書のサンプルの中で利用したいスクリプトがあればすぐに試していただくことができます。

- ・GitHub リポジトリ

- URL: <https://github.com/furaibo/PowerShellDailyRecipeBookSample>

免責事項

本書に記載された内容は、情報提供を目的としています。本書を用いた開発、製作、運用は、必ずご自身の責任と判断によって行ってください。これらの情報による開発、製作、運用の結果について、著者はいかなる責任も負いません。


```
function Add-Product { ...(定義略)... }
function Set-Product { ...(定義略)... }
function Update-UserProfile { ...(定義略)... }
function Find-EmployeeProfile { ...(定義略)... }
function Save-MedicalRecord { ...(定義略)... }
```

1.1.3 関数名として利用可能な動詞

PowerShellにおいてはコマンドレットや関数名に利用可能な動詞の例は以下の通りです。

表 1.1: 利用可能な動詞の例¹

動詞カテゴリ	動詞
Common(共通)	Get, Set, Add, Reset, Remove, Clear, Pop, Push, Open, Close, Enter, Exit, Search, Select, New 等
Communications(通信)	Connect, Disconnect, Send, Receive, Read, Write
Data(データ)	Backup, CheckPoint, Compare, Compress, Convert, Edit, Import, Export, Initialize 等
Diagnostic(診断)	Debug, Measure, Ping, Repair, Resolve, Test, Trace
LifeCycle(ライフサイクル)	Build, Complete, Confirm, Disable, Enable, Install, Invoke, Register, Start, Stop 等
Security(セキュリティ)	Block, Grant, Protect, Revoke, Unblock, Unprotect
Other(その他)	Use

1.2 Powershellにおける予約語

PowerShellでは通常のプログラミング言語と同様に予約語として扱われるキーワードがあります。

1.2.0.1 予約語一覧

予約語およびキーワード²

assembly	exit	process
base	filter	public
begin	finally	return
break	for	sequence
catch	foreach	static
class	from (*)	switch
command	function	throw
configuration	hidden	trap
continue	if	try
data	in	type
define (*)	inlinescript	until
do	interface	using
dynamicparam	module	var (*)

else	namespace	while
elseif	parallel	workflow
end	param	
enum	private	

(*) これらの利用は将来利用のために予約済みです

1.3 基本的な文字列の扱い

```
PS > Write-Host "Hello, World!"  
Hello, World!
```

```
PS > Read-Host "Input here."  
(入力待ち状態に入る)
```

1.4 コメント

1行コメント

```
# 1行コメントを書く時には'#'(シャープ)を文字列の先頭につけます。
```

複数行コメント

```
<#  
複数行コメントを使うときは、  
'<#' と '#>' とで対象文字列を囲みます。  
ただし、PowerShell v1では利用できないのでご注意ください。  
#>
```

1.5 変数

変数の宣言時には先頭に"\$"のマークを付ける必要があることに注意しましょう。また変数の型は明示的な宣言なしで初期化時に自動的に決定されます。

変数の宣言・代入

```
# 通常の変数の宣言  
$var1 = 1
```

```
# 通常の変数の宣言(型指定あり)
# Note: 以下ケースでは変数$var2をdouble型として宣言
[double]$var2 = 2

# グローバル変数の宣言
# Note: グローバル変数の定義・呼び出し時は必ず "$global:(変数名)" の表記を使う
$global:var3 = 3

# スクリプト内で有効な変数の宣言
$script:var4 = 4

# 定数の宣言
# Note: 簡略化された表記方法が無いため、以下のような宣言が必要
Set-Variable -Name var5 -Value 5 -Option Constant
```

1.6 データ型

```
# Note: GetType() 関数とその中のName プロパティを読み出すことによって変数型を確認できます。
# 整数型
PS > $a = 123
PS > $a.GetType().Name
Int32

# 整数型(long)
PS > $b = [long]1234567890
PS > $b.GetType().Name
Int64

# 浮動小数点型
PS > $c = 123.456
PS > $c.GetType().Name
Double

# decimal型
PS > $d = [decimal]1234567890
PS > $d.GetType().Name
Decimal

# 符号なし整数型
PS > $e = [uint16]12345
PS > $e.GetType().Name
UInt16

# 整数型の16進数表記
# Note: 16進表記の数値の先頭には'0x'をつける
```

```
PS > $f = 0xFF # 10進数で255
PS > $f
255
PS > $f.GetType().Name
Int32
```

```
# PowerShellにおいては論理値は$true/$falseとして表します。
PS > $true -or $false
True

PS > -not $true
False
```

```
# 文字列の定義と表示
PS > $a = "abcde-fghij"
PS > $a
"abcde-fghij"

# 文字列の一部の切り出し
# Note: Substring(開始インデックス, 取得文字数) を使用
PS > $a.SubString(1,5)
"bcde-"

# 文字列の分割
# Note: Splitを呼び出した結果は配列になることに注意
PS > $a.Split("-")
abcde
fghij

# 文字列への変数の埋め込み (その1)
PS > $b = "variable a contains ${a}"
PS > $b
variable a contains abcde-fghij

# 文字列への変数の埋め込み (その2)
PS > $b = "variable a contains {0}" -f $a
PS > $b
variable a contains abcde-fghij

# ヒアドキュメント (※複数行の文字列代入)
PS > $c = @"
>> aaa
>> bbb
>> ccc
>> "@
PS > $c
aaa
```

```
bbb
ccc
```

1.6.1 配列

```
# 1次元配列の初期化
# その1: @(...) の形式での宣言
PS > $arr1 = @(1,2,3)
PS > $arr1
1
2
3

# その2: newを使った1次元配列 (要素数3) の宣言
PS > $arr2 = [int[]]::new(3)
PS > $arr2
0
0
0

# その3: newを使った2次元配列 (要素数3x3) の宣言
PS > $arr3 = [int[][]]::new(3, 3)
PS > $arr3
(省略)
```

1.6.1.1 配列の操作・要素へのアクセス

```
# 配列の定義と要素へのアクセス
PS > $a = @("a", "b", "c")
PS > $a[0]
a

PS > $a[1..2]
b
c

# 末尾への要素の追加
PS > $a += "d"
PS > $a.Length
4

# 末尾への要素の追加
# Note: 配列に対して "+=" を使うと配列を結合 (concat) する動作になる
PS > $a += @("e", "f")
PS > $a.Length
6

# 先頭の要素の削除
```

```
# Note: 要素を削除する特別なメソッド等はないので、配列の部分を取り出して対処する
PS > $a = $a[1..($a.Length-1)]
PS > $a
b
c
d
e
f
```

```
# 配列の定義と要素へのアクセス
PS > $b = @(@"a", "b"), @"c", "d")
PS > $b[0][0]
a

PS > $b[0]
a
b

# 末尾への配列の追加
# Note: 配列への要素追加時、配列を入れ子として追加するには ', ' が必要
# @(@"a", "b"), @"c", "d"), @"e", "f") の形にする
PS > $b += ,@"e", "f"
PS > $b[2]
e
f

# 参考: 配列の前に ', ' を入れ忘れた場合
# @(@"a", "b"), @"c", "d"), @"e", "f"), "g", "h") の形になる
PS > $b += @("g", "h")
PS > $b[3]
g
PS > $b[4]
h
```

1.6.2 連想配列

```
# 順序なし連想配列の定義
# Note: 配列の定義・初期化は @{...} の形式
PS > $dict1 = @"a" = 1; "b" = 2; "c" = 3}
PS > $dict1["a"]
1
PS > $dict1
Name Value
----
a     1
b     2
c     3
```

```
# 順序あり連想配列の定義
# Note: [ordered] の表記をつけることで順序ありとして定義可能
PS > $dict2 = [ordered]@"d" = 4; "e" = 5; "f" = 6}
PS > $dict2
```

Name	Value
d	4
e	5
f	6

```
# 連想配列の要素の追加
PS > $dict1["d"] = 4
PS > $dict1
```

Name	Value
a	1
b	2
d	4
c	3

```
# 連想配列の要素の削除
PS > $dict2.Remove("d")
PS > $dict2
```

Name	Value
e	5
f	6

1.7 型変換(キャスト)

```
# 変数の定義
PS > $a = 1
PS > $b = 2.0

# 変数の型の確認
PS > $a.GetType().Name
Int32
PS > $b.GetType().Name
Double

# "[データ型]"の表記を数値や変数の前に追加することでキャストできる
PS > $a = [double]$a
PS > $b = [int]$b

# 変更後の変数の型を確認
PS > $a.GetType().Name
Double
PS > $b.GetType().Name
Int32
```


1.8 演算子

1.8.1 算術演算子

算術演算子の体系については、プログラミング言語等で一般的に使われるものと同じです。

表 1.2: 算術演算子一覧

演算子名	説明
+	加算
-	減算
*	乗算
/	除算
%	剰余計算(mod)
++	インクリメント
--	デクリメント

```
# 加算
PS > 5 + 4
9
# 減算
PS > 5 - 4
1
# 乗算
PS > 5 * 4
20
# 除算
PS > 5 / 4
1.25
# 剰余
PS > 5 % 4
1

# インクリメント
PS > $a = 5;
PS > $a++
PS > $a
6

# デクリメント
PS > $b = 5;
PS > $b--
PS > $b
4
```

1.8.2 論理・比較演算子

論理・比較演算子に関しては不等号など直感的な記号は使うことはできず、Bashと似たような体系となっています。

表 1.3: 論理演算子一覧

演算子名	説明
-and	論理積 (AND)
-or	論理和 (OR)
-xor	排他的論理和 (XOR)
-not	論理否定 (NOT)
!	論理否定 (NOT); -not と同一

表 1.4: 比較演算子一覧

演算子名	説明
-eq	等値 (==)
-ne	非等値 (!=)
-ge	以上 (>=)
-gt	大なり (>)
-le	以下 (<=)
-lt	小なり (<)
-is	同じオブジェクト型かの判定
-isnot	同じオブジェクト型でないかの判定
-contains	セット内に特定の要素を含むかどうかの判定
-notcontains	セット内に特定の要素を含まないかどうかの判定

```
# AND 演算子
PS > $true -and $false
False

# OR 演算子
PS > $true -or $false
True

# XOR 演算子
PS > $true -xor $false
True

# NOT 演算子 (-not)
PS > -not $false
True
```

```
# NOT 演算子 (!)
PS > ! $false
True
```

```
# 等値 (a == b)
PS > 5 -eq 4
False

# 等値でない (a != b)
PS > 5 -ne 4
True

# 以上 (a >= b)
PS > 5 -ge 4
True

# 大なり (a > b)
PS > 5 -gt 4
True

# 以下 (a <= b)
PS > 5 -le 4
False

# 小なり (a < b)
PS > 5 -lt 4
False

# オブジェクト型の比較
# 変数$aをint型の変数として宣言
PS > $a = 1
PS > $a -is [int]
True
PS > $a -is [double]
False
PS > $a -isnot [int]
False

# 包含するかどうかの比較
PS > 'abc', 'def' -contains 'abc'
True
PS > 'abc', 'def' -notcontains 'abc'
False
```

1.8.3 文字列比較・正規表現

```
# ワイルドカード(*)による部分一致 (like)
PS > "abcdefg" -like "abc*"
True

# ワイルドカード(*)による部分一致無し (not like)
PS > "abcdefg" -notlike "abc*"
False

# 正規表現での一致 (match)
PS > "abcdefg" -match "[a-z]+"
True

# 正規表現での不一致 (not match)
PS > "abcdefg" -notmatch "[a-z]+"
False

match 演算子で一致した文字列に関する情報は自動変数$Matchesの中に格納されます。
```

```
PS > "abc123def456" -match "[0-9]+"
True
PS > $Matches
Name Value
----
0 123
```

match演算子による正規表現は比較的手軽に利用できますが、該当する文字列が複数ある場合には対応できず、最初に一致したもののみ取得できます。該当する文字列を複数取り出したいような場合には、[Regex]::Matchesを利用する必要があります。

```
PS > $m = [Regex]::Matches("abc123def456", "[0-9]+")
PS > $m | ForEach-Object { $_.Value }
123
456
```

1.9 範囲演算子

```
# 整数値での範囲演算子の指定 (昇順)
PS > 1..5
1
2
3
4
5

# 整数値での範囲演算子の指定 (降順)
# Note: 配列要素の指定時に意図しない結果を招く場合があるので注意
PS > 0..-3
0
-1
```

```
-2
-3

# 文字での範囲演算子の指定
# Note: この表記はPowerShell ver.5 以前では利用不可
PS > "a".."e"
a
b
c
d
e
```

1.10 ビット演算子

表 1.5: ビット演算子一覧

演算子名	説明
-shl	ビット演算・左シフト (Shift Left)
-shr	ビット演算・右シフト (Shift Right)
-band	ビット AND
-bor	ビット OR
-bxor	ビット XOR
-bnot	ビット否定

```
# ビット左シフト
# 15(2進表記:1111) を1ビット左シフトすると、30(2進表記:11110) となる。
PS > 15 -shl 1
30

# ビット右シフト
# 15(2進表記:1111) を1ビット右シフトすると、7(2進表記:111) となる。
PS > 15 -shr 1
7
```

```
# ビットAND演算
# 13(2進表記:1101) と9(2進表記: 1001) のビット毎のand演算では
# 結果は9(2進表記: 1001) となる。
PS > 13 -band 9
9

# ビットOR演算
# 8(2進表記:1000) と7(2進表記: 111) のビット毎のor演算では
# 結果は15(2進表記: 1111) となる。
PS > 8 -bor 7
```

```
# ビットXOR演算
# 13(2進表記:1101)と9(2進表記:1001)のビット毎のxor演算では
# 結果は4(2進表記:0100)となる。
PS > 13 -bxor 9
4

# ビットNOT演算
# 13(2進表記:1101)の否定を行うと、その補数である-14となる。
# 補数の説明についてはここでは省略する
PS > -bnot 13
-14
```

1.11 エスケープシーケンス

一般的にプログラミング言語では特殊文字を入力するためのエスケープシーケンスとして "\"(バックslash) が使われますが、PowerShellでは "\"" (バッククオート) が使われます。このバッククオート文字はソースコード途中で改行を入れたい場合にも利用します。

表 1.6: エスケープシーケンス一覧³

シーケンス名	説明
'0'	[Null]
'a'	アラート
'b'	バックスペース
'e'	エスケープ (PowerShell6 で追加)
'f'	フォームフィード
'n'	改行
'r'	キャリッジリターン (CR)
't'	水平タブ
'u[x]'	Unicode エスケープシーケンス (PowerShell6 で追加)
'v'	垂直タブ

```
# 改行文字 (`n)
PS > echo "これは`n改行のサンプルです"
これは
改行のサンプルです

# タブ文字 (`t)
PS > echo "これは`tタブ文字出力のサンプルです"
これは   タブ文字出力のサンプルです

# エスケープシーケンスを使ってダブルクオートを出力する例
```

```
PS > echo "バッククオートを出すには `\"このように`\" 入力します"
バッククオートを出すには "このように" 入力します
```

1.12 条件分岐

if/elseif/else 文

```
# 整数値が入った変数$aを3で割った余りについて判定
if ($a % 3 -eq 1) {
    Write-Host "3で割った余りは1です。"
} elseif ($a % 3 -eq 2) {
    Write-Host "3で割った余りは2です。"
} else {
    Write-Host "3の倍数です。"
}
```

switch 文

```
# 整数値が入った変数$aを3で割った余りについて判定
switch ($a % 3) {
    1 { Write-Host "3で割った余りは1です。" }
    2 { Write-Host "3で割った余りは2です。" }
    default { Write-Host "3の倍数です。" }
}
```

1.13 繰り返し処理

PowerShellでは利用可能な繰り返し文として5種類存在します。

for 文

```
# for(初期値;終了条件;繰り返し時処理)
for ($i = 0; $i -lt 10; $i++) {
    Write-Host $i
}
```

foreach 文

```
# foreach(変数 in 範囲演算子,配列等のデータ)
foreach ($i in 1..10) {
    Write-Host $i
}
```

while 文

```
# while(繰り返し条件) { ... }
$i = 0
while ($i -lt 10) {
    Write-Host $i
    $i++
}
```

do ~ while 文

```
# do { ... } while(繰り返し条件)
$i = 0
do {
    Write-Host $i
    $i++
} while($i -lt 10)
```

do ~ until 文

```
# do { ... } until(繰り返し終了条件)
$i = 0
do {
    Write-Host $i
    $i++
} until($i -ge 10)
```

1.14 関数

関数定義

```
# 関数の定義(その1)
function Sample1($a, $b) {
    Write-Host "Input1: ${a}"
    Write-Host "Input2: ${b}"
    return $a + $b
}

# 関数の定義(その2)
function Sample2 {
    Param([int]$a, [int]$b)
    Write-Host "Input1: ${a}"
    Write-Host "Input2: ${b}"
}
```



```
    return $a + $b
}
```

```
# 関数の呼び出し
# Note: Sample1(1,2) のような引数の渡し方はできないことに注意
PS > $n = Sample1 1 2
Input1: 1
Input2: 2
PS > $n
3
```

1.15 スクリプトブロック

関数と似た機能を持つのがスクリプトブロックです。ただし、関数とは異なりスクリプトブロックでは中括弧の外側でのパラメータ指定ができません。また、外部で定義された変数をスクリプトブロックで使う場合では同じ変数として扱われることになり、別スコープとなりませんので注意してください。

```
# スクリプトブロックの出力を変数で受ける
# Note: "&" もしくは "Invoke-Command" でスクリプトブロックの実行
PS > $a = { 1 + 1 }
PS > & $a
2
PS > Invoke-Command -ScriptBlock $a
2

# 外部変数をスクリプトブロック内で使う場合
# Note: 変数$iの値が変更されたとき、影響を受けている
PS > $i = 1
PS > $sb = { "Current i is ${$i}" }
PS > $i = 2
PS > & $sb
Current i is 2
```

1.16 クラス

クラスの定義

```
# クラス定義の一例(会社従業員の情報)
class Employee {
    # メンバ変数
    [string]$FirstName
```

```

[string]$LastName
[string]$DeptName
[string]$SectName
[datetime]$BirthDate
[datetime]$JoinDate

# イニシャライザ
[void] Init([hashtable]$Properties) {
    foreach ($Property in $Properties.Keys) {
        $this.$Property = $Properties.$Property
    }
}

# メソッド定義
# フルネームの文字列を取得する
[string] FullName() {
    return "$($this.LastName) $($this.FirstName)"
}

# 所属(部署・課名全体)を表示する
[string] DeptAndSectName() {
    return "$($this.DeptName) $($this.SectName)"
}

# 年齢を取得する
[int] Age() {
    $span = (Get-Date) - $this.BirthDate
    return [int]($span.Days / 365.25)
}

# 勤続年数を取得する
[int] WorkingYears() {
    $span = (Get-Date) - $this.JoinDate
    return [int]($span.Days / 365.25)
}
}

```

```
# クラス定義の読み込み
# Note: 前述のクラス定義をファイルとして読み込み
PS > ./sample_employee_class.ps1

# インスタンスの初期化
PS > $emp = New-Object Employee
PS > $emp.Init(@{
>> FirstName = "Taro"
>> LastName = "Tanaka"
>> DeptName = "Sales"
>> SectName = "1st Div."
>> BirthDate = "1988/01/01"
>> JoinDate = "2010/04/01"
>> })

# メソッドの実行
PS > $emp.FullName()
Taro Tanaka

PS > $emp.DeptAndSectName()
Sales 1st Div

PS > $emp.Age()
36

PS > $emp.WorkingYears()
14
```

1.17 例外処理

例外処理

```
try {  
    // 例外処理を検出するコード部分  
} catch {  
    // 例外を補足した場合の実行部分  
} finally {  
    // 例外処理の有無にかかわらず動作する部分  
}
```

1.18 数学関連の演算

```
# 円周率  
PS > [math]::pi  
3.14159265358979  
  
# 自然対数の底 (ネイピア数)  
PS > [math]::e  
2.71828182845905
```

```
# 絶対値  
PS > [math]::abs(-5)  
5  
  
# 四捨五入  
PS > [math]::round(3.5)  
4  
  
# 最大値/最小値  
PS > [math]::max(5, 4)  
5  
PS > [math]::min(5, 4)  
4  
  
# 平方根  
PS > [math]::sqrt(2)  
1.4142135623731  
  
# 累乗  
PS > [math]::pow(2, 8)  
256  
  
# 三角関数  
PS > [math]::sin([math]::pi/2)  
1
```

```
# 常用対数 (log10)
PS > [math]::log10(1000)
3

# 自然対数 (log)
PS > [math]::log([math]::e)
1

# 複素数
# Note: 以下の例は (2+i)+(3+2i) の計算と同値
PS > [Numerics.Complex]::new(2,1) + [Numerics.Complex]::new(3,2)
Real Imaginary Magnitude Phase
-----
5.00      3.00      5.83  0.54
```

1.19 フィルタ・パイプライン

PowerShellにもbashと同様のフィルタ・パイプライン処理があります。複数のコマンドをつなげて記述することで所望の処理を行うことが可能です。

```
# カレントフォルダ内の更新日時のみ表示する
# Note: 自動変数$ではパイプラインで渡された値を受け取ることができる
PS > Get-ChildItem | Select-Object{ $_.LastWriteTime }

# 指定テキストファイル内、英数字のみの行のみを抜き出して表示
# Note: %{...} の構文は foreach{...} と同様の動きをする
PS > Get-Content sample.txt | Select-String -Pattern "[0-9a-zA-Z]+" | %{
    $_.line }
```

1.20 リダイレクト

PowerShellではbash等で見られるのと同じようなリダイレクト処理が可能です。

1.20.1 標準出力のリダイレクト

標準出力のリダイレクトの方法として、2つのやり方が考えられます。

```
# ファイルへの新規書き込み
# Note: echo/cat等のコマンドがエイリアスとして設定されています。
PS > echo "aaa`n bbb" > sample01.txt
PS > cat sample01.txt
aaa
bbb
```

```
# ファイルへの追加書き込み
PS > echo "ccc`n ddd" >> sample01.txt
PS > cat sample01.txt
aaa
  bbb
ccc
  ddd
```

```
# ファイルへの新規書き込み
PS > Write-Output "aaa`n bbb" | Out-File sample01.txt
PS > cat sample01.txt
aaa
  bbb

# ファイルへの追加書き込み
PS > Write-Output "ccc`n ddd" | Out-File sample01.txt -Append
PS > cat sample01.txt
aaa
  bbb
ccc
  ddd
```

1.20.2 標準出力・ファイルへのリダイレクトの同時実行

```
# tee(Tee-Object) コマンドを利用した標準出力・ファイルの両方への出力
# Note: -Append オプションをつけて追記の動作にする
PS > echo "eee`n fff" | tee sample01.txt -Append
eee
  fff
PS > cat sample01.txt
aaa
  bbb
ccc
  ddd
eee
  fff
```

1.20.3 エラー出力のリダイレクト

```
# Note: ここでは存在しないファイルdummyを送信している
PS > Get-ChildItem dummy 2> sample02.txt
PS > cat sample02.txt
Get-ChildItem : Cannot find path '(フォルダパス)\dummy' because it does
not exist.
At line:1 char:1
```

```
+ Get-ChildItem dummy 2> sample02.txt  
(省略)
```

```
# エラー出力のファイルへの新規書き込み  
# Note: ここでは存在しないファイルdummyを送信している  
PS > Get-ChildItem dummy  
PS > Get-ChildItem dummy 2> sample02.txt  
PS > cat sample02.txt  
Get-ChildItem : Cannot find path '(フォルダパス)\dummy' because it does  
not exist.  
At line:1 char:1  
+ Get-ChildItem dummy 2> sample02.txt  
(省略)  
  
# 最新のエラー情報を$error自動変数より取得する  
PS > $error[0] | Out-File sample02.txt  
PS > cat sample02.txt  
(上記と同じエラーメッセージが表示される)
```

第2章 PowerShell仕様・コマンドレットの概要

2.1 実行ポリシー

PowerShellでは実行ポリシーが設定されており、セキュリティを確保するための権限設定の機構として機能しています。初期状態ではそのままPowerShellスクリプトの実行ができませんので、必要に応じて実行ポリシーを変更する必要があります。

2.1.1 実行ポリシーの種類

表2.1: 実行ポリシーの一覧¹

ポリシー名	説明
Restricted	コマンド実行は許可されるが、スクリプト実行は許可されない
AllSigned	スクリプト実行可能だが、すべてのスクリプト・構成ファイルが信頼された発行元によって署名されている必要がある。
RemoteSigned	デフォルトの実行ポリシー スクリプトは実行可能だが、インターネットからダウンロードしたスクリプトについては信頼できる発行元からの署名が必要となる。
Bypass	実行時のブロック処理は実行されず、警告・プロンプトの表示なし

Bypass指定はスクリプト実行時のブロック処理がない為、この設定を採用する際には各自十分に注意してください。逆に最も厳しいRestrictedを指定してしまうとPowerShellスクリプトの実行自体が許可されないことに留意してください。

2.1.2 実行ポリシーのスコープ

表2.2: 実行ポリシーのスコープ

スコープ名	説明
MachinePolicy	※グループポリシーで利用可能 コンピュータ内のすべてのユーザーに適用される
UserPolicy	※グループポリシーで利用可能 コンピュータ内の現在のユーザーのみに適用される
Process	現在のPowerShellセッションのみに適用される セッション終了すると設定は削除される
CurrentUser	現在のユーザーのみに適用される
LocalMachine	現在のコンピュータ(ローカルマシン)に適用される

2.1.3 実行ポリシーの変更手順

実行ポリシーを変更するには以下のようにコマンドを実行します。

```
# カレントユーザについてAllSignedを設定する
PS > Set-ExecutionPolicy AllSigned -Scope CurrentUser -Force

# 現在利用中のローカルマシンについてRemoteSignedを設定する
PS > Set-ExecutionPolicy RemoteSigned -Scope LocalMachine -Force
```

2.2 コマンドレットの概要

コマンドプロンプトやbashといったシェル環境では「コマンド」と呼ばれますが、PowerShellにおいて定義されたものについては『コマンドレット』と呼ばれています。

大まかな動作として伝統的なシェル環境で使われるコマンドと違いはありませんが、標準的なPowerShellコマンドレットにおいては.NET環境で実装された.dllファイルとして高速に動作します。

また、その名前が「(動詞)-(名詞)」の命名規則に従っていることが特徴です。そして、この名前が長くなりがちなコマンドレットの別名としてエイリアスが設定されており、短縮された記法・UNIX/Linuxコマンドのような使い方が利用可能になっています。

2.2.1 コマンドレット一覧

表 2.3: 主要なコマンドレット一覧

コマンドレット	処理内容
Get-ExecutionPolicy	実行ポリシー情報の取得
Set-ExecutionPolicy	実行ポリシー情報の設定
Get-WmiObject	WMI オブジェクトの取得
Get-Location	カレントパスを取得する
Set-Location	カレントパスを設定する
Get-Date	現在の時刻データを取得する
Test-Path	ファイル・フォルダのパスが存在するかの確認
Write-Host	現在のコンソール上の標準出力に文字列を表示する
Read-Host	現在のコンソール上の標準入力から入力を受け取る
Write-Output	文字列を次のオブジェクトに送る
Out-File	入力オブジェクトをファイルへ書き出す
Select-String	文字列の検索・マッチング処理を行う (grep に類似)
Where-Object	パイプラインで渡されたオブジェクトのフィルタ処理
Sort-Object	オブジェクトのソート処理
Get-ChildItem	アイテムの子アイテムを取得する。アイテム指定がない場合はカレントフォルダ内のファイル一覧が表示される
Get-ItemProperty	アイテムのプロパティを取得する
Set-ItemProperty	アイテムのプロパティを設定する
Stop-Computer	コンピュータの停止
Restart-Computer	コンピュータの再起動

2.2.2 エイリアス

表 2.4: 主要なエイリアス一覧²

エイリアス	対応コマンドレット	処理内容
%	Foreach-Object	foreach 処理の実行
?	Where-Object	絞り込み処理の実行
echo	Write-Output	文字列の表示
cat	Get-Content	ファイル内の内容の表示
clear	Clear-Host	現在のターミナル画面の表示をクリア
pwd	Get-Location	カレントディレクトリのパスを表示
cd	Set-Location	カレントディレクトリの移動
ls	Get-ChildItem	カレントディレクトリ内のファイル一覧を表示
mv	Move-Item	ファイルの移動
cp	Copy-Item	ファイルのコピー
rm	Remove-Item	ファイルの削除
mkdir	(コマンドプロンプトで定義済)	フォルダの作成
rmdir	Remove-Item	フォルダの削除
sort	Sort-Object	ソート処理の実行
wget	Invoke-WebRequest	Web リクエストの送信
curl	Invoke-WebRequest	Web リクエストの送信
ps	Get-Process	プロセス一覧の表示
kill	Stop-Process	プロセス停止のシグナル送信
tee	Tee-Object	出力先として標準出力・ファイルの両方を指定する
man	help	マニュアル情報の表示
whoami	(コマンドプロンプトで定義済)	ログインユーザー名の表示
pushd	Push-Location	ディレクトリ履歴のスタックへの指定パス追加
popd	Pop-Location	ディレクトリ履歴のスタックから指定パス取り出し

Get-Alias コマンドレットを使うことで、設定されたエイリアスから元のコマンドレットを調べることができます。

PowerShell コマンドレットのエイリアスとして設定されているものは元のコマンドレットが表示されますが、コマンドプロンプトで定義済みのコマンドは Get-Alias コマンドレット使用時にはエラーとなります。

エイリアスも通常のコマンドレットと同様に動作しますが、PowerShell の公式ドキュメントにおいては、エイリアスはコマンド入力にのみ使用し、スクリプト内では使わないことが推奨されています。

2.3 自動変数

表 2.5: 主要な自動変数一覧³

自動変数	内容
\$	セッション内で受信された最終行の最後のトークンを格納する
\$^	セッション内で受信された最終行の最初のトークンを格納する
\$?	直近のコマンドの実行状態を格納する
\$_	パイプライン オブジェクト内の現在のオブジェクトを格納する (*\$PSItem と同じ)
\$PSVersionTable	現在利用中の PowerShell 環境のバージョン情報を格納する
\$Host	現在 PowerShell を実行中のホストの情報を格納する
\$PSHome	PowerShell インストール先のパスを格納する
\$PID	プロセス識別値(PID) を格納する
\$IsMacOS	macOS 環境の場合は値が true になる
\$IsWindows	Windows 環境の場合は値が true になる
\$IsLinux	Linux 環境の場合は値が true になる
\$profile	現在のユーザーと現在のホストアプリケーションの PowerShell プロファイルのパスを含む変数
\$args	スクリプトに渡された引数の値の配列
\$Matches	"match"演算子による文字列比較の結果が格納される
\$Error	直近のエラーに関する情報が格納される
\$true	boolean 値の true として動作する
\$false	boolean 値の false として動作する
\$null	null 値をさす変数

2.4 特殊フォルダ

PowerShellでは.NETフレームワークの機能を使って、特殊フォルダパスにアクセスできます。Windowsで使われる「デスクトップ」「マイドキュメント」「マイピクチャ」といったフォルダ群がそれらに該当します。この機能のおかげで、フォルダの正確なパスを意識しなくともコーディングが可能となります。特にファイルを特定の特殊フォルダに移動・保存するような処理を実装する場合に重宝します。

表 2.6: 主要な特殊フォルダ⁴

特殊フォルダ名	日本語名称・概要
UserProfile	ユーザープロファイル
Desktop	デスクトップ
MyComputer	マイコンピュータ
MyDocuments	マイドキュメント
MyMusic	マイミュージック
MyPictures	マイピクチャ
ProgramFiles	プログラムファイル

特殊フォルダパス指定の際には、Environmentにアクセスすることになります。ただし、この場合は "\$([Environment]::GetFolderPath('...'))" の形でパス指定することになります。

```
# ユーザーのルートフォルダへ移動
PS > cd "$([Environment]::GetFolderPath('UserProfile'))"
```

```
# デスクトップへの移動
PS > cd "$([Environment]::GetFolderPath('Desktop'))"
```

```
# マイドキュメントへの移動
PS > cd "$([Environment]::GetFolderPath('MyDocuments'))"
```

```
# マイピクチャへの移動
PS > cd "$([Environment]::GetFolderPath('MyPictures'))"
```

特殊フォルダパスの取得において問題となるのが「ダウンロード」フォルダです。.NETにおいては上記の特殊フォルダ指定のやり方ではダウンロードフォルダのパスは取得できなくなっています。あまりスマートなやり方とは言えませんが、UserProfileなどと組み合わせてファイルパスを取得するしかありません。

2.5 各種ユースケースへの対応

2.5.1 PowerShellのバージョン情報の確認

自動変数\$PSVersionTableを利用します。

```
# 自動変数を利用してバージョン情報を確認
PS > $PSVersionTable
```

Name	Value
----	-----
PSVersion	5.1.19041.3693
PSEdition	Desktop
PSCompatibleVersions	{1.0, 2.0, 3.0, 4.0...}
BuildVersion	10.0.19041.3693
CLRVersion	4.0.30319.42000
WSManStackVersion	3.0
PSRemotingProtocolVersion	2.3
SerializationVersion	1.1.0.1

※各自のPowerShell実行環境によって表示内容は異なります。

2.5.2 プロセス情報の確認

1026	37	28864	17788	0.34	17376	1	
AcrobatNotificationClient							
370	17	4908	7368	1.75	1956	1	Adobe Crash Processor
610	25	9072	12936	4.33	13092	1	AdobeCollabSync
358	19	5192	7728	1.02	13796	1	AdobeCollabSync

(以下略)

※各自の PowerShell 実行環境によって表示内容は異なります。

2.5.3 OS・ハードウェア情報の確認

```
# BIOS 情報の確認
PS > Get-WmiObject Win32_BIOS

SMBIOSBIOSVersion : F2
Manufacturer       : American Megatrends Inc.
Name                : F2
SerialNumber        : Default string
Version             : ALASKA - 1072009

# プロセッサ情報の取得
PS > Get-WmiObject Win32_Processor

Caption              : AMD64 Family 23 Model 8 Stepping 2
DeviceID             : CPU0
Manufacturer         : AuthenticAMD
MaxClockSpeed        : 3700
Name                 : AMD Ryzen 7 2700X Eight-Core Processor
SocketDesignation    : AM4

# OS 情報の取得
PS > Get-WmiObject Win32_OperatingSystem

SystemDirectory      : C:\WINDOWS\system32
Organization         :
BuildNumber           : 19045
RegisteredUser        : (ユーザーのメールアドレス)
SerialNumber          : (シリアルナンバー)
Version              : 10.0.19045
```

※各自の実行マシンによって表示内容は異なります。

2.5.4 各プロパティの確認

Select-Objectによってプロパティ名を指定することにより、プロパティの絞り込みが可能です。

```
# サンプル用のファイルの作成
PS > echo "sample" > sample.txt

# Lengthのプロパティのみ抜き出す
PS > Get-ChildItem -Path sample.txt | Select-Object Length
```

```
Length
```

```
-----
```

```
18
```

2.5.5 PowerShellスクリプトの実行

bashと同じく、ピリオド(.)の後にスクリプト名を表記することでスクリプトの実行が可能です。

```
PS > ./your_ps_script.ps1
```

2.5.6 フォルダの作成

・利用コマンド

—mkdir(New-Item -ItemType Directoryのエイリアス)

```
# フォルダの作成 (その1)
```

```
PS > mkdir sample_folder
```

(※作成されたフォルダパスと指定パス以下のファイル一覧が表示される)

```
# フォルダの作成 (その2)
```

```
PS > New-Item sample_folder -ItemType Directory
```

(※作成されたフォルダパスと指定パス以下のファイル一覧が表示される)

2.5.7 カレントフォルダの確認・移動

・利用コマンド

—pwd(Get-Locationのエイリアス)

—cd(Set-Locationのエイリアス)

```
# カレントフォルダの確認 (その1)
```

```
PS > pwd
```

```
# カレントフォルダの確認 (その2)
```

```
PS > Get-Location
```

```
# カレントフォルダの移動 (その1)
```

```
PS > cd $Home
```

```
# カレントフォルダの移動 (その2)
```

```
PS > Set-Location $Home
```

2.5.8 カレントフォルダ以下のファイル情報取得

・利用コマンド

—ls(Set-Locationのエイリアス)

```
# カレントフォルダ内のファイルを確認
# 記法その1
PS > ls
(ファイル情報が出力される)

# 記法その2
PS > Get-ChildItem
(ファイル情報が出力される)
```

```
# カレントフォルダ内のPDF形式ファイルを取得する
# Note: ToLower() メソッドを使うことで拡張子が大文字の場合に対応
# 記法その(1)
PS > ls | ? { $_.Extension.ToLower() -eq ".pdf" }

# 記法その(2)
PS > Get-ChildItem | Where-Object { $_.Extension.ToLower() -eq ".pdf" }
```

2.5.9 フォルダ・ファイルの存在確認

・利用コマンド

—TestPath(Set-Locationのエイリアス)

```
# パスの存在確認
# Note: 変数$pathに、パスが入っているものとする
PS > Test-Path $path

# フォルダの存在確認
PS > Test-Path -Path $path -PathType Container

# ファイルの存在確認
PS > Test-Path -Path $path -PathType Leaf
```

2.5.10 ファイル・フォルダの移動

・利用コマンド

—mv(Move-Itemのエイリアス)


```
# カレントフォルダ内のファイル名称変更
# 記法その1
PS > mv sample1.txt aaa_sample.txt

# 記法その2
PS > Move-Item -Path sample.txt -Destination aaa_sample.txt
```

```
# カレントフォルダ内の.txt形式ファイルの名称に接頭辞 "aaa_" を付け加える
# 記法その1
PS > ls | $ { Move-Item -Path $_ -Destination "aaa_${$_}" }

# 記法その2
PS > Get-ChildItem | Foreach-Object { \
>> Move-Item -Path $_ -Destination "aaa_${$_}" }

# ディレクトリ内のファイルの並び・拡張子変えずに、3桁の連番の数字のファイル名に直す
# Note: たとえば "001.txt", "002.txt" ... のようなファイル名になる
PS > $count = 0
PS > Get-ChildItem | Foreach-Object { \
>> $ext = [System.IO.Path]::GetExtension($_) \
>> $newFileName = "{0:000}{1}" -f ($count+1), $ext \
>> Move-Item -Path $_ -Destination $newFileName \
>> $count++ \
>> }
```

2.5.11 ファイル・フォルダのコピー

・利用コマンド

—cp(Copy-Itemのエイリアス)

```
# 記法その1
PS > cp sample.txt sample_copy.txt

# 記法その2
PS > Copy-Item sample.txt sample_copy.txt
```

2.5.12 ファイル・フォルダの削除

・利用コマンド

—rm(Remove-Itemのエイリアス)

ファイル削除のみの場合はオプションは必要ありませんが、フォルダ削除の場合はオプション指定が必要となることに注意してください。

```
# 記法その1
PS > rm sample.txt

# 記法その2
PS > Remove-Item sample.txt
```

```
# Note:
# -Recurse ... フォルダ内のファイル・サブフォルダも削除対象とする
# -Force ... 隠しファイル・読み取り専用フォルダを削除

# 記法その1
PS > rm sample_folder -Recurse -Force

# 記法その2
PS > Remove-Item sample_folder -Recurse -Force
```

2.5.13 ファイルアクセス権限の変更

Linux/macOSのbashのようにchmodがアクセス権限変更のためのエイリアスとして設定されていてほしい所ですが、PowerShellではそのようなエイリアスはありません。icacls等のコマンドを使って権限設定をすることになります。

```
# ファイルアクセス権限の表示
# Note: フォルダ内には"sample1.txt"以下3つのファイルがある場合を考える
PS > Get-ChildItem | Select-Object Mode, Name

Mode      Name
----      -
-a----- sample1.txt
-a----- sample2.txt
-a----- sample3.txt

# 読み取りのファイルアクセス権限を設定
PS > icacls sample1.txt /grant [ユーザー名]:R
processed file: sample01.txt
Successfully processed 1 files; Failed processing 0 files

# 書き込みのファイルアクセス権限を設定
PS > icacls sample1.txt /grant [ユーザー名]:W
(省略)
```

2.5.14 テキストファイル内の文字列検索

・利用コマンド

— cat(Get-Contentのエイリアス)

```
# テキストファイルの内容
# Note: 以下のような簡単なテキストファイルが存在するものとする
PS > cat sample.txt
aaaAAA
bbbBBB
cccCCC

# 正規表現を含まない文字列のマッチング
# Note: 1行目のみマッチング
PS > cat sample.txt | Select-String -SimpleMatch "aaa"

aaaAAA

# 正規表現を含む文字列のマッチング
# Note: 1-2行目のみマッチング
PS > cat sample.txt | Select-String -Pattern "[a-b]+"
```

```
aaaAAA
bbbBBB

# 正規表現を含む文字列のマッチング (空行の削除あり)
# Note: %{...} の構文は foreach{...} と同様の動きをする
# Note: "$_.line" の記述を入れることでマッチしない行は表示させない
PS > cat sample.txt | Select-String -Pattern "[a-b]+" | %{ $_.line }
aaaAAA
bbbBBB
```

2.5.15 CSV形式ファイルの読み込み

Import-CSV コマンドレットでCSVデータを読み込んでPowerShell用のオブジェクトを取得できます。

```
# CSV ファイル内容の表示
# Note: 変数$csvFilePathにCSV ファイルへのパスが入っているものとする
PS > Get-Content $csvFilePath
A001,Sample1,50
A002,Sample2,55
A003,Sample3,60
A004,Sample4,65

# CSV から PSCustomObject への変換
```

```
# Note: -header オプション以下ヘッダ名を指定する
$csvdataArray = Import-CSV -Path $csvFilePath -Header 'id', 'name', 'score'

id      : A001
name    : Sample1
score   : 50

id      : A002
name    : Sample2
score   : 55

(以下略)
```

2.5.16 PSCustomObjectからJSON形式への変換

ConvertTo-Json コマンドレットを使って手軽にディクショナリ (PSCustomObject) を JSON 形式の文字列へと変換することができます。

```
# 順序なし連想配列の定義
# JSON への変換
PS > $dict1 = @{"a" = 1; "b" = 2; "c" = 3; "d" = @(4, 5)}
PS > $dict1 | ConvertTo-Json
{
  "a": 1,
  "b": 2,
  "d": [
    4,
    5
  ],
  "c": 3
}
```

第3章 基本サンプル集

3.1 基本サンプル 1: fizzbuzz問題

プログラマーにとっては、おなじみのfizzbuzz問題¹です。fizzbuzz問題はコーディングテストの初歩的な題材として近年よく使われますが、ここでは基本的な条件分岐のイメージをつかむために取り上げています。

3.1.0.1 fizzbuzz問題

- ・ 数字を1から順番に数え上げ、以下のルールに従って文字列を出力
 - 3の倍数でFizzを出力し、5の倍数ではBuzzを出力
 - 3の倍数かつ5の倍数である場合はFizzBuzzとして出力
 - 上記のいずれの場合にも当てはまらなければ、そのまま数値を出力
- 同じ動作をするスクリプトを2通り考えてみます。

3.1.0.2 サンプルコード

リスト3.1: 基本サンプル(1-1)

```
1: # foreachおよび範囲演算子を用いた場合
2: foreach($i in 1..100) {
3:     if ($i % 3 -eq 0 -and $i % 5 -eq 0) {
4:         Write-Host "FizzBuzz"
5:     } elseif ($i % 3 -eq 0) {
6:         Write-Host "Fizz"
7:     } elseif ($i % 5 -eq 0) {
8:         Write-Host "Buzz"
9:     } else {
10:        Write-Host $i
11:    }
12: }
```

リスト3.2: 基本サンプル(1-2)

```
1: # foreachなし、範囲演算子を用いた場合
2: 1..100 | % {
3:     if ($_ % 3 -eq 0 -and $_ % 5 -eq 0) { Write-Host "FizzBuzz" }
4:     elseif ($_ % 3 -eq 0) { Write-Host "Fizz" }
5:     elseif ($_ % 5 -eq 0) { Write-Host "Buzz" }
```

¹https://ja.wikipedia.org/wiki/Fizz_Buzz

```
6:     else { Write-Host $_ }  
7: }
```

3.1.0.3 実行結果

```
PS > .\sample01-1.ps1  
1  
2  
Fizz  
4  
Buzz  
Fizz  
7  
8  
Fizz  
Buzz  
11  
Fizz  
13  
14  
FizzBuzz  
16  
17  
...(後略)...
```

3.2 基本サンプル2: クイックソート

ソートアルゴリズムの一種である、クイックソートの実装例です。

3.2.0.1 クイックソートのアルゴリズムについて

図: クイックソートの概要 (Wikipedia)

アルゴリズム [\[編集\]](#)

クイックソートは以下の手順で行われる。

1. ピボットの選択：適当な値（**ピボット** (英語版) という）を境界値として選択する
2. 配列の分割：ピボット未満の要素を配列の先頭側に集め、ピボット未満の要素のみを含む区間とそれ以外に分割する
3. **再帰**：分割された区間に対し、再びピボットの選択と分割を行う
4. ソート終了：分割区間が整列済みなら再帰を打ち切る

・ URL:<https://ja.wikipedia.org/wiki/%E3%82%AF%E3%82%A4%E3%83%83%E3%82%AF%E3%82%BD%E3%83%BC%E3%83%88>

今回のクイックソートのサンプルにおける配列のソート手順は以下の通りとします。

- ・ 配列の要素数が1以下であればそのまま値を返す
- ・ 配列の先頭の要素をピボット(基準値)として選ぶ
- ・ ピボットよりも小さい数値、大きい数値に分けてそれぞれ配列に入れる
- ・ 分けた配列2つについて再帰的にソートを適用する

3.2.0.2 サンプルコード

リスト3.3: 基本サンプル(2)

```
1: # クイックソート処理を行う関数の定義
2: function Get-QuickSort($arr) {
3:     # 要素数が1以下ならばそのまま値を返す
4:     # Note: LengthもしくはCountで配列の長さを取得
5:     if ($arr.Length -eq 0) {
6:         return @()
7:     } elseif ($arr.Length -eq 1) {
8:         return $arr
9:     }
10:
11:     # 変数の初期化
12:     # Note: 配列の先頭の値をピボットとして選択する
13:     $pivot = $arr[0]
14:     $arr1 = @()
15:     $arr2 = @()
16:
17:     # ピボットに応じて各配列への分配を行う
18:     # Note: ピボット値よりも小さなものを$arr1, 大きなものを$arr2に格納する
19:     foreach ($item in ($arr | Select-Object -skip 1)) {
20:         if ($item -lt $pivot) {
21:             $arr1 += $item
22:         } else {
23:             $arr2 += $item
24:         }
25:     }
26:
27:     # クイックソートの適用(再帰)
28:     # Note: 配列として関数からの返り値を受け取るには'@'を追加する
29:     return @(Get-QuickSort($arr1)) + @($pivot) + @(Get-QuickSort($arr2))
30: }
31:
32: # 乱数値の配列を取得(要素数20/値域:1~100)
33: $numArray = @()
```

```

34: 0..20 | Foreach-Object {
35:     $numArray += (Get-Random -Minimum 1 -Maximum 100) }
36:
37: # 配列へのクイックソート関数の適用
38: $sortedArray = @(Get-QuickSort($numArray))
39:
40: # ソート前後の配列を表示する
41: Write-Host "Input: ${numArray}"
42: Write-Host "Output: ${sortedArray}"

```

3.2.0.3 実行結果

```

PS > .\sample02.ps1
Input: 92 93 28 64 31 46 14 13 29 30 31 33 83 37 46 57 59 27 91 63 24
Output: 13 14 24 27 28 29 30 31 31 33 37 46 46 57 59 63 64 83 91 92 93

```

3.3 基本サンプル3: 数当てゲーム

簡単な数当てゲームのサンプルです。

3.3.0.1 数当てゲームの内容

- ・ランダムで1~100の間の数字が選ばれ、それを当てる
- ・入力が答えよりも小さければ"もっと大きな数です"と表示
- ・入力が答えよりも大きければ"もっと小さな数です"と表示
- ・入力が正解であれば、正解までの試行回数を表示して終了

3.3.0.2 サンプルコード

リスト3.4: 基本サンプル(3)

```

1: # 開始時メッセージ
2: Write-Host "数当てゲーム: 1~100の間の数字を当ててください"
3:
4: # 変数の初期化
5: $count = 0
6: $ans = (Get-Random -Minimum 1 -Maximum 100)
7:
8: # 数当てゲームのループ処理
9: while($true) {
10:     # ユーザ入力を受け付ける/入力値の型チェック(入力不正ならやり直し)
11:     $rawInput = Read-Host "1~100の間で数字を入力してください"
12:     if (-not ([int]::TryParse($rawInput, [ref]$null))) {

```



```
13:         continue
14:     }
15:
16:     # 入力された数値が1～100の範囲内かチェック(入力が不正ならやりなおし)
17:     $input = [int]$rawInput
18:     if ($input -le 0 -or $input -gt 100) {
19:         continue
20:     }
21:
22:     # 試行回数のカウント
23:     $count++
24:
25:     # 数値および表示メッセージの判定
26:     if ($input -lt $ans) {
27:         Write-Host "もっと大きな数です"
28:     } elseif ($input -eq $ans) {
29:         Write-Host "正解です!"
30:         break
31:     } else {
32:         Write-Host "もっと小さな数です"
33:     }
34: }
35:
36: # 試行回数の表示
37: Write-Host "試行回数: ${count}"
```

3.3.0.3 実行結果

```
PS > .\sample03.ps1
1~100の間で数字を入力してください: 50
もっと小さな数です
1~100の間で数字を入力してください: 25
もっと大きな数です
1~100の間で数字を入力してください: 37
もっと大きな数です
1~100の間で数字を入力してください: 43
もっと大きな数です
1~100の間で数字を入力してください: 47
もっと小さな数です
1~100の間で数字を入力してください: 45
もっと小さな数です
1~100の間で数字を入力してください: 44
正解です!
試行回数: 7
```

3.4 基本サンプル4: 関数積分の近似値計算

このサンプルでは以下の関数についての積分の近似値計算をします。

3.4.0.1 対象となる数式

3.4.0.2 サンプルコード

リスト 3.5: 基本サンプル(4)

```
1: # 積分の近似値を求める関数の定義
2: function Get-Calculus {
3:     # 引数の定義
4:     # Note: 関数を渡すため、Param句を使った引数指定としている
5:     # - $func: 積分を実行する関数
6:     # - $lower: 積分区間の下端
7:     # - $upper: 積分区間の上端
8:     Param($func, $lower, $upper)
9:
10:    # 変数の初期化
11:    # Note: 積分区間の分割数は変数$divLimitで定義する
12:    $divLimit = 100000
13:    [double]$width = ([math]::abs($upper - $lower) / $divLimit)
14:    [double]$sum = 0
15:
16:    # 積分の実行
17:    foreach ($i in 0..$divLimit) {
18:        $x = $lower + $width * $i
19:        $y = Invoke-Command $func -ArgumentList $x
```

```

20:         $diff = $y * $width
21:         $sum += $diff
22:     }
23:
24:     return $sum
25: }
26:
27: # 値を計算する関数の定義
28: function Func1($x) { return [math]::sin($x) }
29: function Func2($x) { return [math]::pow($x,3) + [math]::pow($x,2) + $x + 1 }
30:
31: # 積分区間の定義
32: $lower = 0
33: $upper = 3
34:
35: # 積分の計算を実行
36: # Note: 関数を変数として渡す時は "$function:(関数名)" とする。
37: $result1 = Get-Calculus -func $function:Func1 -lower $lower -upper $upper
38: $result2 = Get-Calculus -func $function:Func2 -lower $lower -upper $upper
39:
40: # 結果の表示
41: Write-Host "Result 1: $result1"
42: Write-Host "Result 2: $result2"

```

3.4.0.3 実行結果

```

PS > .\sample04.ps1
Result 1: 1.98999461325132
Result 2: 36.7506150024748

```

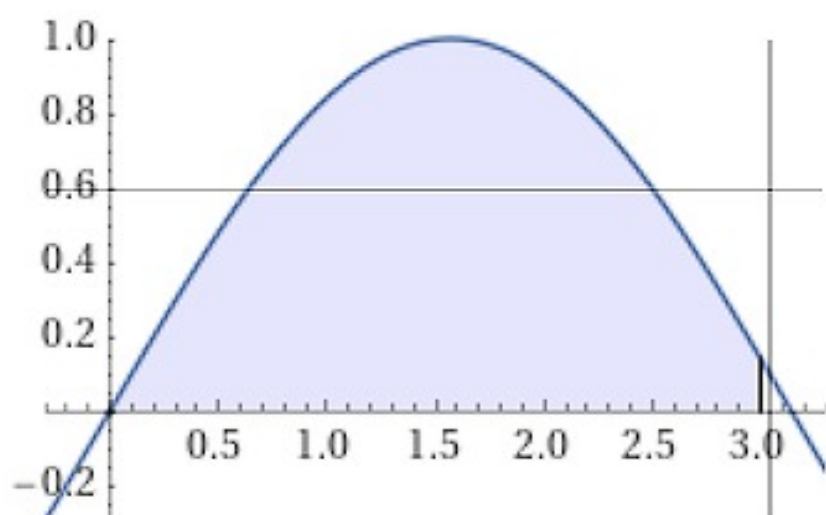
3.4.0.4 参考(実際の計算との比較)

WolframAlphaにて計算した結果は次のようになります。

定積分

$$\int_0^3 \sin(x) dx = 1 - \cos(3) \approx 1.9900$$

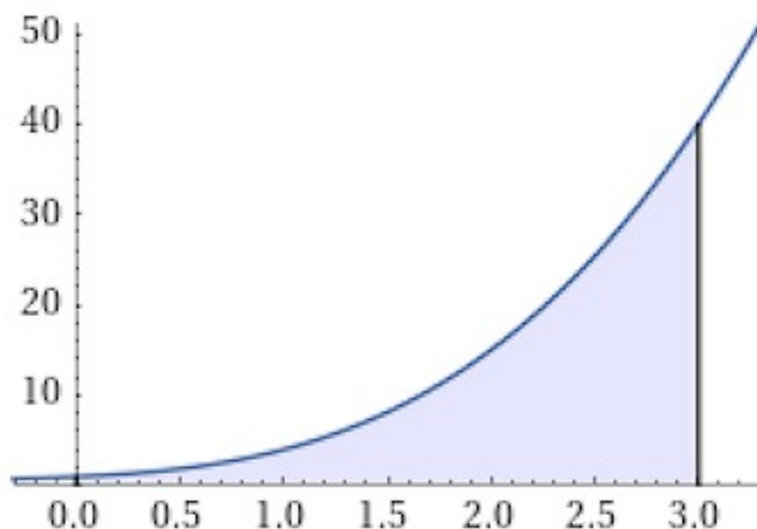
積分の視覚的表現



定積分

$$\int_0^3 (x^3 + x^2 + x + 1) dx = \frac{147}{4} = 36.75$$

積分の視覚的表現

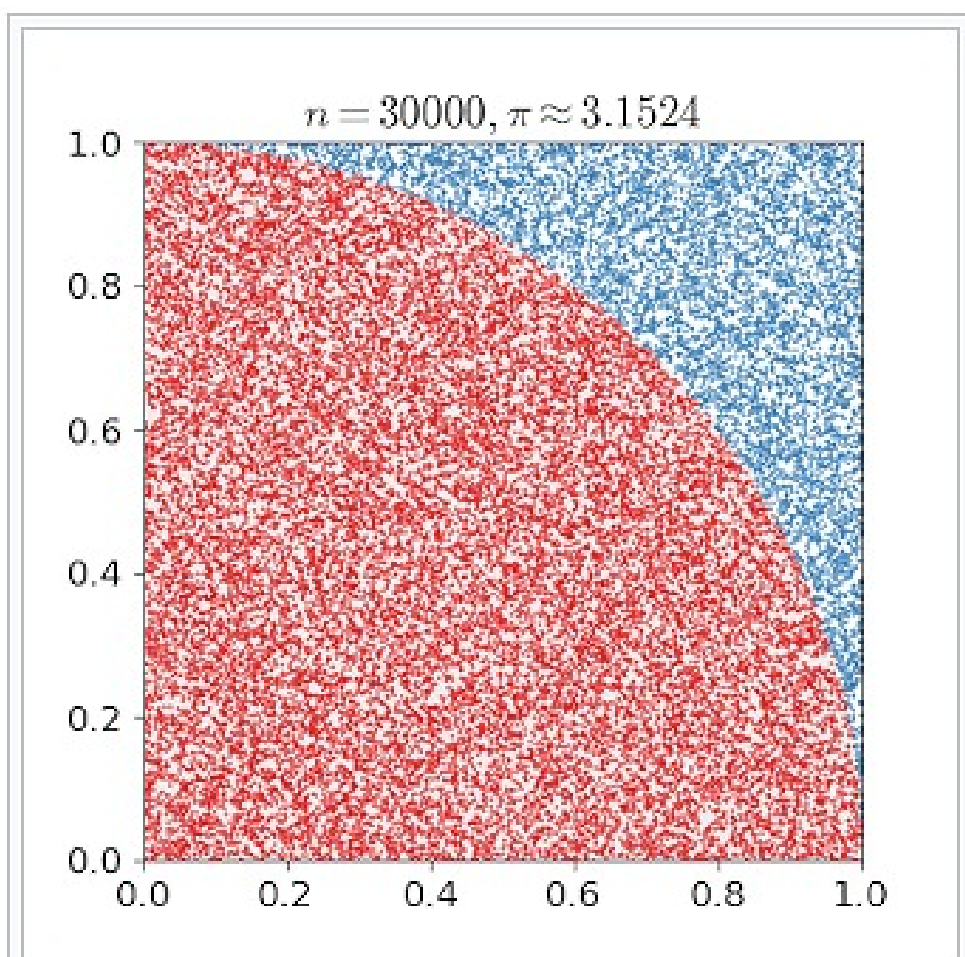


3.5 基本サンプル5: 円周率の近似値計算

3.5.1 方法(1) モンテカルロ法

モンテカルロ法²とは乱数を利用して確率や数値を近似的に求める手法です。

²<https://ja.wikipedia.org/wiki/%E3%83%A2%E3%83%B3%E3%83%86%E3%82%AB%E3%83%AB%E3%83%AD%E6%B3%95>



モンテカルロ法を円周率 π の値の 近似に適用した例。30,000点をランダムに置いたとき、 π の推定量は実際の値から0.07%以下の誤差の範囲内にあった。

このサンプルでは、長さ1の辺を持つ正方形を考えます。正方形の中に収まるようにランダムで

点を打ち、これが原点(0,0)を中心とする単位円(半径1の円)、厳密に言えば扇形ですが、この中に収まっているかどうかを調べます。前で紹介した図においては長さ1の円の正方形と、単位円(半径1の円)が描かれています。正方形の面積は1、半径1の円による扇形の面積は $\pi/4$ となります。このとき、正方形の中に点をランダムで打ったとき、それが半径1の円の内部にも入っている確率は正方形と円の面積の比と同じものになるはずであり、その確率もまた $\pi/4$ となります。

つまり、この場合において

・ $\pi/4 = (\text{単位円中の点の総数})/(\text{辺の長さ1の正方形の中の点の総数})$

の関係式が成り立ちます。この式を変形すると、

・ $\pi = 4 * (\text{単位円中の点の総数})/(\text{辺の長さ1の正方形の中の点の総数})$

ということになります。この方針でPowerShellのコードを書いてみましょう。

3.5.1.1 サンプルコード

リスト3.6: 基本サンプル(5-1)

```
1: # 変数の初期化
2: $hitCount = 0
3: $tryLimit = 100000
4:
5: # 繰り返し計算
6: foreach($_ in 1..$tryLimit) {
7:     # 乱数の生成
8:     $x = Get-Random -Minimum 0.0 -Maximum 1.0
9:     $y = Get-Random -Minimum 0.0 -Maximum 1.0
10:    $dist = [math]::sqrt($x*$x + $y*$y)
11:
12:    # 半径1の円(単位円)内にある点の数え上げ
13:    if ($dist -le 1) {
14:        $hitCount++
15:    }
16: }
17:
18: # 円周率の概算値の表示(※有効桁数10桁まで)
19: $pi_value = [decimal](4 * $hitCount / $tryLimit)
20: $text = "Piの近似値: {0:N10}" -f $pi_value
21: Write-Host $text
```

3.5.1.2 実行結果

近似値の精度は高くありませんが、円周率に近い値が得られていることがわかります。

```
PS > .\sample05-1.ps1
Piの近似値: 3.1432000000
```

(※実行結果は毎回変動します)

3.5.2 方法 (2) ガウス＝ルジャンドルのアルゴリズム

ガウス＝ルジャンドルのアルゴリズム³を使って円周率の近似値計算をする場合について考えます。特に円周率の数値を計算するうえで特に値の収束が早いアルゴリズムとして知られています。

図: ガウス＝ルジャンドルのアルゴリズムの概要 (Wikipedia)

アルゴリズム [\[編集\]](#)

これによる円周率の計算方法は以下の通りである。

初期値の設定 [\[編集\]](#)

$$a_0 = 1 \quad b_0 = \frac{1}{\sqrt{2}} \quad t_0 = \frac{1}{4} \quad p_0 = 1$$

反復式 [\[編集\]](#)

a, b が希望する精度 (桁数) になるまで以下の計算を繰り返す。小数第 n 位まで求めるとき $\log_2 n$ 回程度の反復でよい。

$$\begin{aligned} a_{n+1} &= \frac{a_n + b_n}{2} \\ b_{n+1} &= \sqrt{a_n b_n} \\ t_{n+1} &= t_n - p_n (a_n - a_{n+1})^2 \\ p_{n+1} &= 2p_n \end{aligned}$$

π の算出 [\[編集\]](#)

円周率 π は、 a, b, t を用いて以下のように近似される。

$$\pi \approx \frac{(a+b)^2}{4t}$$

3.5.2.1 サンプルコード

リスト 3.7: 基本サンプル (5-2)

```
1: # 変数の初期化
2: $a = 1
3: $b = 1 / [math]::sqrt(2)
4: $t = 1 / 4
5: $p = 1
6:
7: # 繰り返し計算
8: # Note: 計算値の精度を向上させるためdecimal型を使用
9: foreach ($iterCount in 1..5) {
10:     # 次の数値の計算
```

³<https://ja.wikipedia.org/wiki/%E3%82%AC%E3%82%A6%E3%82%B9%E3%82%BC%E3%83%AB%E3%82%B8%E3%83%A3%E3%83%B3%E3%83%89%E3%83%AB%E3%81%AE%E3%82%A2%E3%83%AB%E3%82%B4%E3%83%AA%E3%82%BA%E3%83%A0>


```

11:     $a_next = [decimal](($a + $b) / 2)
12:     $b_next = [decimal]([math]::sqrt($a * $b))
13:     $t_next = [decimal]($t - $p * [math]::pow($a - $a_next, 2))
14:     $p_next = [decimal](2 * $p)
15:
16:     # 数値の置き換え
17:     $a = $a_next
18:     $b = $b_next
19:     $t = $t_next
20:     $p = $p_next
21:
22:     # 円周率の概算値の表示(※有効桁数25桁まで)
23:     $pi_value = [decimal]([math]::pow($a + $b, 2) / (4 * $t))
24:     $text = "Piの近似値: {0:N25} ({1}回目)" -f $pi_value, $iterCount
25:     Write-Host $text
26: }

```

3.5.2.2 実行結果

decimal型の有効桁数の範囲では5回程度の試行でも円周率に近い値へと収束します

```

PS > .\sample05-2.ps1
Piの近似値: 3.1405792505221689589715153 (1回目)
Piの近似値: 3.1415926462135343190370255 (2回目)
Piの近似値: 3.1415926535897901142300206 (3回目)
Piの近似値: 3.1415926535897901144168687 (4回目)
Piの近似値: 3.1415926535897901144168687 (5回目)

```

3.5.3 方法(3) ラマヌジャンの円周率公式

インドの天才数学者と呼ばれるシュリニヴァーサ・ラマヌジャン(1887-1920)⁴が生み出した円周率公式を使う方法です。非常に収束が早いことで知られています。

ラマヌジャンは「インドの魔術師」と呼ばれたほどの天才数学者であり、前に紹介した円周率公式には証明がつけられていませんでした。あまりにも突飛な式で公式の導出根拠が長らく不明となっていました。後にモジュラー関数の考えを用いた式であることが数学的に証明されました。

4.<https://ja.wikipedia.org/wiki/%E3%82%B7%E3%83%A5%E3%83%AA%E3%83%BB%E3%83%B4%E3%82%A1%E3%83%BC%E3%82%B5%E3%83%BB%E3%83%A9%E3%83%9E%E3%83%8C%E3%82%B8%E3%83%A3%E3%83%B3>

図: ラマヌジャンの肖像画



ラマヌジャンの発見した公式等には証明が付けられていないものが少なくありませんが、現代の数学者によって証明が与えられるごとにラマヌジャンのアイデアの先進性が示され続けています。死後百年が経過した今なお彼の遺したノートから最先端の数学にも通じる、あるいはさらにその先を行くような成果が続々と発見されています。

3.5.3.1 サンプルコード

リスト3.8: 基本サンプル(5-3)

```
1: # 関数定義
2: # 階乗を求める関数
3: function Get-Factorial ($n) {
4:     if ($n -eq 0) {
5:         return 1
6:     }
7:     $fact = 1
```

```

8:     1..$n | ForEach-Object { $fact *= $_ }
9:     return $fact
10: }
11:
12: # Note: 計算値の精度を向上させるためdecimal型を使用
13: foreach ($iterCount in 1..5) {
14:     # 分母部分の計算
15:     [decimal]$sum = 0
16:     foreach ($n in 0..$iterCount) {
17:         $sum += `
18:             ((Get-Factorial (4*$n)) * (26390 * $n + 1103)) / `
19:             [math]::pow([math]::pow(4, $n) * [math]::pow(99, $n) *
(Get-Factorial $n)), 4)
20:     }
21:     [decimal]$denom = 2 * [math]::sqrt(2) * $sum / (99 * 99)
22:
23:     # 円周率の概算値の表示(※有効桁数25桁まで)
24:     $pi_value = 1 / $denom
25:     $text = "Piの近似値: {0:N25} (n={1})" -f $pi_value, $iterCount
26:     Write-Host $text
27: }

```

3.5.3.2 実行結果

処理時の誤差のために実際の円周率とずれが出てしまうものの、小数点15桁まで一致します。n=3の段階でも円周率にかなり近い近似値が得られることから、ラマヌジャンの円周率公式の精度の高さがわかります。

```

PS > .\sample05-3.ps1
Piの近似値: 3.1415926535897939864090277 (n=1)
Piの近似値: 3.1415926535897933468727709 (n=2)
Piの近似値: 3.1415926535897933468727653 (n=3)
Piの近似値: 3.1415926535897933468727653 (n=4)
Piの近似値: 3.1415926535897933468727653 (n=5)

```

第4章 応用サンプル集

4.1 応用サンプル1: chocolateyによるパッケージインストール

4.1.01 下準備

Windows用のパッケージマネージャーであるchocolateyを使って各種ソフトウェア・パッケージのインストールの自動化を行います。複数のWindows環境でインストール作業を自動化するのに役立ちます。macOSにおいてはHomebrew、Linuxではaptやrpmといったパッケージマネージャーがありますが、基本的な動作のイメージとしてはほぼ同じと考えてよいでしょう。

- ・Chocolatey公式ページ

- URL:<https://chocolatey.org/>

- ・Chocolateyパッケージ

- URL:<https://community.chocolatey.org/packages>

公式ページにあるインストール用コマンドを実行することでChocolatey本体をインストールできます。

```
PS > Set-ExecutionPolicy Bypass -Scope Process -Force; [System.Net.ServicePointManager]::SecurityProtocol = [System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object System.Net.WebClient).DownloadString('https://community.chocolatey.org/install.ps1'))
```

インストールしたいパッケージの種類によっては、進行途中で手動での入力が必要となる箇所があります。サンプルで示されたもの以外にも多くのパッケージのインストールが可能です。以下ページより利用可能なパッケージを検索できます。

4.1.02 サンプルコード(1)

以下コードを実行する際には、管理者権限でPowerShellを実行してください。管理者権限でない場合は実行前にメッセージが表示されます。

リスト4.1: 応用サンプル(1)

```
1: # nuget
2: choco install -y nuget.commandline
3: # エディタ
4: choco install -y notepadplusplus
5: # コーディング環境
6: choco install -y git vscode powershell-core
7: # CUIツール
8: choco install -y jq less curl wget
9: # アーカイバ
```

```
10: choco install -y 7zip
11: # 画像・動画ビューア
12: choco install -y mpc-be honeyview
13: # 画像・動画変換ツール
14: choco install -y ffmpeg xmedia-recode
15: # 各種チャットツール
16: choco install -y slack discord line
17: # Google IME(日本語)
18: choco install -y GoogleJapaneseInput
19: # ゲーム・音楽関連
20: choco install -y steam epicgameslauncher itunes
21: # 動画ダウンロード用ツール
22: choco install -y vividl yt-dlp
23: # Adobe Reader
24: choco install -y adobereader
25: # 画像編集用ツール
26: choco install -y gimp
```

4.1.0.3 サンプルコード(2)

以下を実行すると、これ以降の本書応用サンプルの実行時に必要となる Chocolatey パッケージをまとめてインストールできます。

```
PS > choco install -y pdftk imageMagick.app ghostscript ffmpeg sox.portable
yt-dlp zbar
```

4.2 応用サンプル2: 複数PDFファイルの連結

4.2.0.1 注意点

このサンプルの実行には"pdftk"のインストールが必要です。

```
PS > choco install -y pdftk
```

4.2.0.2 解説

このサンプルでは pdftk の機能を使って簡易的に処理を実行します。pdftk コマンドの後に続けて入力ファイルを指定することで、指定された順に結合された PDF ファイルが出力されます。PDF の内容を変更せずに結合処理を行うときには、cat(結合)のオプションを入れ、output 以下に出力先パスを指定します。

```
# 入力: sample1.pdf, sample2.pdf
# 出力: outfile.pdf
PS > pdftk sample1.pdf sample2.pdf cat output outfile.pdf
```

4.2.0.3 サンプルコード

リスト 4.2: 応用サンプル(2)

```
1: # スクリプトの引数定義
2: Param(
3:     [string[]] $inputPath, # 入力フォルダパス
4:     [string] $outputPath # 出力ファイルパス
5: )
6:
7: # 出力先パスの確認
8: if ($outputPath -eq "") {
9:     Write-Error "出力先パスを指定してください"
10:    return
11: } elseif (Test-Path $outputPath) {
12:     Write-Error "${outputPath} はすでに存在しています"
13:    return
14: }
15:
16: # 入力ファイル数の確認
17: $inputPDFs = @($inputPath)
18: Write-Host $inputPDFs
19: if ($inputPDFs.Count -le 1) {
20:     Write-Error "2つ以上のファイルパスを入力してください"
21:     return
22: }
23:
24: # PDFのマージ処理実行
25: pdftk $inputPath cat output $outputPath
26: Write-Host "ファイル出力先: ${outputPath}" -ForegroundColor Cyan
```

4.3 応用サンプル3: フォルダ内画像のPDFファイル化

4.3.0.1 下準備

このサンプルの実行には"ImageMagick"のインストールが必要です。

```
PS > choco install -y imageMagick.app
```

macOS/Linux環境ではImageMagickの機能を使うにはconvertコマンドを使いますが、Windows環境では名前のコマンド名称の重複を避けるためにmagickを指定するようになっています。

4.3.0.2 解説

ImageMagickはさまざまな画像フォーマットに対応した多機能な画像ツールです。画像の拡張子変換やPDF内部の画像を取り出すこともできます。

ただし、古くから開発が行われており、数多くのフォーマットに対応していることもあって内部に脆弱性も抱えています。すべての画像形式の変換処理に使うようなことは避け、処理対象となる画像フォーマットをある程度絞った方が無難です。

ここでは詳細について解説はしませんが、policy.xmlにより処理対象のフォーマットを指定できます。

```
# 画像からのPDF変換処理
# - 入力: 拡張子ext1,ext2,ext3のファイル全体
# - 出力: outfile.pdf、-qualityで100を指定して最高品質とする
PS > magick "*.{ext1,ext2,ext3,...}" -quality 100 outfile.pdf
```

4.3.0.3 サンプルコード

リスト4.3: 応用サンプル(3)

```
1: # スクリプトの引数定義
2: param (
3:     [string] $inputPath
4: )
5:
6: # 入力パスの存在確認
7: if (($inputPath -eq "") -or (-not (Test-Path -Path $inputPath))) {
8:     Write-Error "入力フォルダが見つかりません"
9:     exit
10: }
11:
12: # 変数定義
13: $dateString = Get-Date -Format "yyyyMMddHHmmss"
14: $saveFolderPath = [Environment]::GetFolderPath("MyPictures") + "\MyImage2PDF"
15: $outputPDFPath = "${saveFolderPath}${dateString}.pdf"
16:
17: # 保存先フォルダの作成
18: if (-not (Test-Path -Path $saveFolderPath -PathType Container)) {
19:     New-Item -Path $saveFolderPath -ItemType Directory -Force > $null
```

```

20: }
21:
22: # ImageMagickによる画像のPDF化処理
23: # Note: この例ではpng/jpeg/gif形式が対象
24: magick "${inputPath}/*.{png,jpg,jpeg,gif}" -quality 100 $outputPDFPath
25:
26: # 保存先フォルダの表示
27: Write-Host "保存先フォルダ: ${saveFolderPath}" -ForegroundColor Cyan

```

4.4 応用サンプル4: PDF ファイルページの画像化

4.4.0.1 下準備

このサンプルの実行には"ImageMagick"および"ghostscript"のインストールが必要です。

```
PS > choco install -y imageMagick.app ghostscript
```

4.4.0.2 解説

前のサンプルに続いて、ここでも ImageMagick を使用します。この場合は出力ファイル名の文字列フォーマットの指定が必要となります。

```

# 画像からの PDF からの画像取出処理
# - 入力: input.pdf
# - 出力: "(3桁の連番).jpg"で出力する。-density オプションでDPIを300として指定
PS > magick -density 300 input.pdf ".\%03d.jpg"

```

4.4.0.3 サンプルコード

リスト 4.4: 応用サンプル(4)

```

1: # スクリプトの引数定義
2: param (
3:     [string] $inputPath,
4:     [string] $outputExt=".png"
5: )
6:
7: # 変数定義
8: $dateString = Get-Date -Format "yyyyMMddHHmmss"
9: $saveFolderRoot = [Environment]::GetFolderPath("MyPictures") + "\MyPDF2Image"
10: $saveFolderPath = "${saveFolderRoot}${dateString}"
11: Write-Host $saveFolderPath

```



```

12: return
13:
14: # 対応する画像フォーマットの定義
15: # Note: png/jpg/gif形式に対応
16: $imageExtensions = @(".png", ".jpg", ".jpeg", ".gif")
17:
18: # 出力対象の画像フォーマットの確認
19: if (-not ($outputExt.ToLower() -in $imageExtensions)) {
20:     Write-Error ".png/.jpg/.jpeg/.gif のいずれかの拡張子から選んでください"
21:     exit
22: }
23:
24: # 保存先フォルダの作成
25: if (-not (Test-Path -Path $saveFolderPath -PathType Container)) {
26:     New-Item -Path $saveFolderPath -ItemType Directory -Force > $null
27: }
28:
29: # PDFページの画像化処理
30: # Note: DPI(解像度設定)は"-density"オプションで変更可能
31: magick -density 300 "$inputPath" "${saveFolderPath}\%03d${outputExt}"
32:
33: # 保存先フォルダの表示
34: Write-Host "保存先フォルダ: ${saveFolderPath}" -ForegroundColor Cyan

```

4.5 応用サンプル5: 直近ダウンロード済みファイルのアーカイブ処理

このサンプルでは1ヵ月以内に更新された比較的新しいファイルのアーカイブ処理を扱います。アーカイブ形式はzip形式としてファイルコピーおよびアーカイブ処理を自動で実行します。また入力元をダウンロードフォルダ、出力先をマイドキュメント内のArchiveフォルダとします。

4.5.0.1 サンプルコード

リスト4.5: 応用サンプル(5)

```

1: # 変数の定義
2: # Note: 一時フォルダ・圧縮ファイル名として現在時刻の文字列を使用する
3: $sourceFolderPath = "${HOME}\Downloads"
4: $saveFolderPath = [Environment]::GetFolderPath("MyDocuments") + "\Archive"
5: $dateTimeString = Get-Date -Format "yyyyMMddHHmmss"
6: $tempFolderPath = "${saveFolderPath}\${dateTimeString}"
7: $saveZipFilePath = "${saveFolderPath}\${dateTimeString}_archive.zip"
8:

```

```

9: # 保存先フォルダの作成
10: if (-not (Test-Path -Path $saveFolderPath -PathType Container)) {
11:     New-Item -Path $saveFolderPath -ItemType Directory -Force > $null
12: }
13:
14: # 一時フォルダの作成(フォルダ名は時刻による)
15: New-Item -Path $tempFolderPath -ItemType Directory -Force > $null
16:
17: # 1ヵ月以内に更新されたファイル名の取得・一時フォルダへのファイルコピー
18: # Note: フルパスを取得するには"FullName"プロパティを参照
19: Get-ChildItem -Path $sourceFolderPath |
20:     Where-Object { $_.LastWriteTime -gt (Get-Date).AddMonths(-1)} |
21:     Foreach-Object { Copy-Item $_.FullName -Destination "$tempFolderPath\$_" }
22:
23: # 一時フォルダのzip化
24: # Note: zip形式ファイルをアーカイブ先(この場合はドキュメントフォルダ)に移す
25: Compress-Archive -Path $tempFolderPath -DestinationPath $saveZipFilePath
26:
27: # 一時フォルダの削除処理
28: # Note: フォルダおよびその内部も含めて削除するには-Recurseオプションを追加
29: Remove-Item $tempFolderPath -Recurse
30:
31: # 完了メッセージ & 保存先パスの表示
32: Write-Host "アーカイブ化処理が完了しました"
33: Write-Host "保存先パス: ${saveZipFilePath}" -ForegroundColor Cyan

```

4.6 応用サンプル6: 画像のリサイズ・拡張子変換

ここでは前のサンプルで使ったImageMagick等の機能を使わずに画像のリサイズ、拡張子の変換処理を実行します。jpg/png/gif形式のような頻出の画像フォーマットであれば.NETフレームワークの標準機能だけでも画像処理を行うことが可能です。

4.6.0.1 サンプルコード

リスト4.6: 応用サンプル(6)

```

1: # スクリプトの引数定義
2: # Note: デフォルトのリサイズ後画像のサイズはXGA(1024x768)以下
3: param (
4:     [string] $inputPath,      # 入力フォルダパス
5:     [string] $outExt=".png",  # 出力時拡張子
6:     [int] $maxWidth=1024,     # リサイズ後画像の幅の最大値

```

```

7:      [int] $maxHeight=768      # リサイズ後画像の高さの最大値
8: )
9:
10: # アセンブリの読み込み
11: Add-Type -AssemblyName System.Drawing
12:
13: # 対応する画像フォーマットの定義
14: # Note: png/jpg/gif形式に対応
15: $imageExtensionFilters = @("*.png", "*.jpg", "*.jpeg", "*.gif")
16:
17: # 変数定義
18: $dateString = Get-Date -Format "yyyyMMddHHmmss"
19: $saveFolderRoot = [Environment]::GetFolderPath("MyPictures") + "\Resized"
20: $saveFolderPath = "${saveFolderRoot}\${dateString}"
21:
22: # 入力パスの存在確認
23: if (-not (Test-Path -Path $inputPath -PathType Container)) {
24:     Write-Error "入力フォルダが見つかりません"
25:     exit
26: }
27:
28: # 入力パス以下にあるファイルパスの取得
29: $filePathArray = Get-ChildItem -Recurse -Path $inputPath `
30:     -Include $imageExtensionFilters
31:
32: # 保存先フォルダの作成
33: # Note: このサンプルではマイピクチャ内に"Resized"フォルダを新規作成する
34: if (-not (Test-Path -Path $saveFolderPath -PathType Container)) {
35:     New-Item -Path $saveFolderPath -ItemType Directory -Force > $null
36: }
37:
38: # 画像のリサイズ処理実行
39: foreach($path in $filePathArray) {
40:     # 画像の読み出し
41:     $image = [System.Drawing.Image]::FromFile($path)
42:
43:     # 画像の幅・高さおよび縮小比を取得
44:     $width = $image.Width
45:     $height = $image.Height
46:     $hRatio = $maxWidth / $image.Width
47:     $vRatio = $maxHeight / $image.Height

```

```

48:
49:     # 縮小比に応じた縮小後の画像幅・高さの決定
50:     # Note: 指定した幅・高さに収まる範囲で最大のサイズになるように縦横比率を調整
51:     if (($hRatio -lt 1) -and ($vRatio -lt 1)) {
52:         if ($hRatio -lt $vRatio) {
53:             $width = $maxWidth
54:             $height = [int]($height * $hRatio)
55:         } else {
56:             $width = [int]($width * $vRatio)
57:             $height = $maxHeight
58:         }
59:     } elseif ($hRatio -lt 1) {
60:         $width = $maxWidth
61:         $height = [int]($height * $hRatio)
62:     } elseif ($vRatio -lt 1) {
63:         $width = [int]($width * $vRatio)
64:         $height = $maxHeight
65:     }
66:
67:     # リサイズ後の画像の書き出し(ビットマップ形式)
68:     $canvas = New-Object System.Drawing.Bitmap($width, $height)
69:     $graphics = [System.Drawing.Graphics]::FromImage($canvas)
70:     $graphics.DrawImage($image, 0, 0, $width, $height)
71:
72:     # 出力時ファイル名の取得および出力パスの決定
73:     $fileName = [System.IO.Path]::GetFileNameWithoutExtension("$path") +
$outExt
74:     $outputPath = "${saveFolderPath}\${fileName}"
75:
76:     # 画像ファイルの保存
77:     $canvas.Save($outputPath)
78:
79:     # オブジェクトの破棄
80:     $graphics.Dispose()
81:     $canvas.Dispose()
82:     $image.Dispose()
83: }
84:
85: # 保存先フォルダの表示
86: Write-Host "保存先フォルダ: ${saveFolderPath}" -ForegroundColor Cyan

```

4.7 応用サンプル7: 音声ファイルの連結処理

4.7.0.1 下準備

このサンプルの実行にはffmpegのインストールが必要です。

```
PS > choco install -y ffmpeg
```

4.7.0.2 解説

このサンプルでは、動画・音声ファイルの変換やエンコーディングで良く使われるffmpegを活用します。ffmpegではさまざまなコーデック・ファイル形式に対応しています。非常に多機能ではあるのですが、反面オプション指定が複雑であるため使い方に慣れが必要です。ここでは複数の音声ファイルを結合して1つのmp3ファイルにする処理を行います。

ffmpegによるファイルの結合

fオプションを使って、-f concatと記述することで結合処理となります。また-safe 0オプションをつけると入力ファイルの絶対パス指定が可能となります(ない場合にはエラー発生)。

さらに、-iオプションで入力元ファイルを指定できるのですが、入力元ファイルのパスが書かれたテキストファイルを代わりに指定することもできます。入力ファイルが多く、入力文字列が非常に長くなる場合にはffmpeg側でエラーが発生してしまいますので、テキストファイルでパスを指定する方がエラーの発生を防げます。

```
# 複数ファイル(相対パス)で逐一指定して処理する場合
PS > ffmpeg -f concat -i ./data/input1.mp3 -i ./data/input2.mp3
./outfile.mp3
```

```
# テキストファイルでまとめて入力ファイル(絶対パス)を指定して処理する場合
PS > ffmpeg -f concat -safe 0 -i "input_path.txt" ./outfile.mp3
```

このサンプルにおいては、曲と曲の間に無音時間を追加するため、無音のmp3ファイルを作成して指定された曲の間に挟み込む形を取っています。(※ほかの方法が存在する可能性もありますが、筆者の調べのつく限りではこのアプローチ以外にうまくいく方法が見つかりませんでした。)

4.7.0.3 サンプルコード

リスト4.7: 応用サンプル(7)

```
1: # スクリプトの引数定義
2: param (
3:     [string] $inputPath,      # 入力フォルダパス
4:     [string] $outExt=".mp3",  # 出力時拡張子
5:     [int] $interval=3        # 曲間の無音時間(秒数指定)
6: )
7:
8: # 変数定義
```

```

9: $dateString = Get-Date -Format "yyyyMMddHHmmss"
10: $saveFolderPath = [Environment]::GetFolderPath("MyMusic") + "\MusicConcat"
11: $saveFilePath = "${saveFolderPath}\${dateString}\${outExt}"
12: $silentFilePath = "${saveFolderPath}\silent.mp3"
13: $tempFilePath = ".\temp_list.txt"
14:
15: # 対応する画像フォーマットの定義
16: # Note: mp3/wav/wma/aac形式に対応
17: $audioExtensions = @(".mp3", ".wav", ".wma", ".aac")
18:
19: # 入力パスの存在確認
20: if (-not (Test-Path -Path $inputPath -PathType Container)) {
21:     Write-Error "入力フォルダがありません"
22:     exit
23: }
24:
25: # 出力拡張子の確認
26: if (-not ($outExt.ToLower() -in $audioExtensions)) {
27:     Write-Error "指定された拡張子は利用できません: ${outExt}"
28:     exit
29: }
30:
31: # 保存先フォルダの作成
32: if (-not (Test-Path -Path $saveFolderPath -PathType Container)) {
33:     New-Item -Path $saveFolderPath -ItemType Directory -Force > $null
34: }
35:
36: # 無音ファイルの作成(楽曲間の無音時間用)
37: # Note: -tオプションで無音時間の秒数を指定
38: ffmpeg -f lavfi -i anullsrc=r=44100:cl=mono -t ${interval} `
39:     -aq 9 -c:a libmp3lame $silentFilePath
40:
41: # ファイルパスの取得
42: # Note: $outExtで指定された拡張子と同じファイルのみ対象
43: $filePathArray = Get-ChildItem -Recurse -Path $inputPath |
44:     Where-Object { $_.Extension.ToLower() -eq $outExt } |
45:     ForEach-Object { $_.FullName }
46:
47: # ffmpeg読み込み用のファイルパスについてのテキストを作成
48: # Note: "file '[ファイルパスの文字列]'" の形式で入力を作成する
49: $fileString = ""

```

```

50: $filePathArray | ForEach-Object {
51:     $fileString += "file '$_'\`n"
52:     $fileString += "file '${silentFilePath}'\`n"
53: }
54: Write-Output $fileString | Out-File $tempFilePath -Encoding utf8
55:
56: # ffmpegによる音声ファイルの結合処理
57: # Note: UTF-8のテキストファイルを入力として使用
58: ffmpeg -f concat -safe 0 -i $tempFilePath $saveFilePath
59:
60: # 無音ファイルおよび一時ファイルの削除
61: Remove-Item $silentFilePath
62: Remove-Item $tempFilePath
63:
64: # 完了メッセージ & 保存先パスの表示
65: Write-Host "ファイル結合処理が完了しました"
66: Write-Host "保存先パス: ${saveFilePath}" -ForegroundColor Cyan

```

4.8 応用サンプル8: 音声ファイルの分割処理

4.8.0.1 下準備

このサンプルの実行には"ffmpeg"および"sox.portable"のインストールが必要です。

```
PS > choco install -y ffmpeg sox.portable
```

また、sox コマンドでmp3処理のため、以下.dllファイルも必要です。

- libmad-0.dll
- libmp3lame-0.dll

上記の2つの.dllファイルのDL先URLは以下のとおりです。

- URL:<https://app.box.com/s/tzn5ohyh90viedu3u90w2l2pmp2bl41t>

2つの.dllファイルを取得したら、Chocolateyのライブラリフォルダ内に配置しましょう。

```
C:\ProgramData\chocolatey\lib\sox.portable\lib
```

4.8.0.2 解説

soxは音声編集・加工用のコマンドラインツールです。

silence (0 or 1) (秒数)tのようなフォーマットで指定します。silence 0で先頭の無音時間部分の削除なし、silence 1なら先頭の無音時間部分の削除が入ります。また無音時間と判定するための時間を秒数で指定するとき、数値の後に"t"を追加します。

オプションの後に無音秒数を指定し、:newfileは一意な番号を含む新しい出力ファイルの書き出

しを命令し、:restartはコマンドを繰り返すことを意味します。たとえば、0 3t 0.1% : newfile : restartのように書いたときは、無音時間を削除することなく、無音時間を区切りとして繰り返しファイルを作成することになります。

4.8.0.3 サンプルコード

リスト4.8: 応用サンプル(8)

```
1: # スクリプトの引数定義
2: param (
3:     [string] $inputPath    # 入力ファイルパス
4: )
5:
6: # ffmpeg/soxのインストール状況の確認
7: try {
8:     (Get-Command ffmpeg).Definition
9:     (Get-Command sox).Definition
10: } catch {
11:     Write-Error "ffmpeg もしくは sox が見つかりません"
12:     exit
13: }
14:
15: # 変数定義
16: $dateString = Get-Date -Format "yyyyMMddHHmmss"
17: $saveRootPath = [Environment]::GetFolderPath("MyMusic") + "\MusicSeparated"
18: $saveFolderPath = "${saveRootPath}\${dateString}"
19:
20: # 入力パスの存在確認
21: if (-not (Test-Path -Path $inputPath -PathType Leaf)) {
22:     Write-Error "入力ファイルが見つかりません"
23:     exit
24: }
25:
26: # 保存先フォルダの作成
27: if (-not (Test-Path -Path $saveFolderPath -PathType Container)) {
28:     New-Item -Path $saveFolderPath -ItemType Directory -Force > $null
29: }
30:
31: # 変数宣言
32: $fileName = [System.IO.Path]::GetFileName($inputPath)
33: $srcFilePath = "${saveFolderPath}\src.mp3"
34: $dstFilePath = "${saveFolderPath}\out.mp3"
35:
```



```

36: # 保存先フォルダへのファイルコピー(&変換処理)
37: if ([IO.Path]::GetExtension($fileName) -ne ".mp3") {
38:     # ファイルコピー
39:     Copy-Item $inputPath -Destination $srcFilePath
40: } else {
41:     # ffmpegによるmp3形式への変換
42:     ffmpeg -i $inputPath -ab 256k $srcFilePath
43: }
44:
45: # soxによるファイル分割処理
46: # Note: 音量0.1%以下を無音時間とみなし、無音時間3秒で分割する
47: sox -V3 $srcFilePath $dstFilePath `
48:     silence 0 3t 0.1% 0 3t 0.1% : newfile : restart
49:
50: # 完了メッセージ & 保存先フォルダの表示
51: Write-Host "ファイル分割処理が完了しました"
52: Write-Host "保存先フォルダ: ${saveFolderPath}" -ForegroundColor Cyan

```

4.9 応用サンプル9: 動画の音声データ取得・拡張子変換

4.9.0.1 下準備

このサンプルの実行にはffmpegのインストールが必要です。

```
PS > choco install -y ffmpeg
```

4.9.0.2 解説

煩雑なオプション指定は必要なく、単に入出力ファイルパスで拡張子を指定すればその通りに変換してくれます。

```

# mov形式ファイルをmp4形式ファイルに変換する場合
PS > ffmpeg -f concat -i ./data/input.mov ./outfile.mp4

```

4.9.0.3 サンプルコード

リスト4.9: 応用サンプル(9)

```

1: # スクリプトの引数定義
2: Param(
3:     [string]$inputPath = "",      # 入力フォルダパス
4:     [string]$outExt = "",         # 出力ファイル拡張子

```

```

5:      [switch]$getaudio = $false    # 音声データ取得のみ
6: )
7:
8: # ffmpegのインストール状況の確認
9: try {
10:     (Get-Command ffmpeg).Definition
11: } catch {
12:     Write-Error "ffmpegが見つかりません"
13:     exit
14: }
15:
16: # 入力ファイルの存在チェック
17: # Note: inputPathオプションの値がある場合はパスの存在をチェック
18: if ($inputPath -ne "" -and `
19:     (-not (Test-Path -Path $inputPath -PathType Leaf))) {
20:     Write-Error "入力ファイルが見つかりません"
21:     exit
22: }
23:
24: # 変数定義
25: $saveFolderPath = [Environment]::GetFolderPath("MyVideos") +
"\ConvertedFiles"
26: $fileName = [System.IO.Path]::GetFileNameWithoutExtension($inputPath)
27:
28: # 保存先フォルダの作成
29: if (-not (Test-Path -Path $saveFolderPath -PathType Container)) {
30:     New-Item -Path $saveFolderPath -ItemType Directory -Force > $null
31: }
32:
33: # ffmpegによるファイルの変換処理
34: try {
35:     if ($getaudio) {
36:         # 保存先パスの決定
37:         $saveFilePath = "${saveFolderPath}\${fileName}.mp3"
38:
39:         # 同じパスにファイルが存在する場合は削除
40:         if (Test-Path -Path $saveFilePath) { Remove-Item $saveFilePath }
41:
42:         # 動画からの音声データの抜き出し処理
43:         ffmpeg -i $src -q:a 0 -map a $saveFilePath
44:

```

```

45:     } else {
46:         # 保存先パスの決定
47:         $saveFilePath = "${saveFolderPath}\${fileName}${outExt}"
48:
49:         # すでに同じパスにファイルが存在する場合は削除
50:         if (Test-Path -Path $saveFilePath) { Remove-Item $saveFilePath }
51:
52:         # 動画ファイル形式の変換
53:         ffmpeg -i $src $saveFilePath
54:     }
55: } catch {
56:     Write-Error "ffmpegによるファイル変換を完了できませんでした"
57:     exit
58: }
59:
60: # 完了メッセージ & 保存先パスの表示
61: Write-Host "ファイル変換が完了しました"
62: Write-Host "保存先パス: ${saveFilePath}" -ForegroundColor Cyan

```

4.10 応用サンプル10: yt-dlpによる動画ダウンロードの自動化

4.10.0.1 下準備

このサンプルの実行にはyt-dlpの最新版のインストールが必要です。Chocolateyで最新版のyt-dlp環境を用意していただくようにお願いします。

```
PS > choco install -y yt-dlp
```

4.10.0.2 解説

yt-dlpはYouTube・ニコニコ動画等の動画サイトから動画ファイルをダウンロードできるコマンドラインツールです。-o オプションの後に出力先のフォルダを指定します。直接URL指定によるダウンロード、もしくはURLを記述したファイルを指定してダウンロード処理できます。

ここでは、動画サムネイルあり、Windows互換のファイル名に置き換えるため--embed-thumbnailと--windows-filenamesをオプションに指定します。また--console-titleオプションを付加すると、コンソール表示ウィンドウのタイトルバーに現在の処理内容を表示できます。

```

# URL直接指定でダウンロードする場合
PS > yt-dlp (ページURL) -o ./output `
>> --embed-thumbnail --windows-filenames --console-title

```

```
# 動画URLを記述したファイルを指定してまとめてダウンロードする場合
PS > yt-dlp -a ./urls.txt -o ./output `
>> --embed-thumbnail --windows-filenames --console-title
```

4.10.0.3 サンプルコード

リスト4.10: 応用サンプル(10)

```
1: # スクリプトの引数定義
2: Param(
3:     [string]$urlListPath = "",
4:     [string]$url = "",
5:     [int]$threads = 2
6: )
7:
8: # yt-dlpのインストール状況の確認
9: try {
10:     (Get-Command yt-dlp).Definition
11: } catch {
12:     Write-Error "yt-dlpが見つかりません"
13:     exit
14: }
15:
16: # 変数定義
17: $saveFolderPath = [Environment]::GetFolderPath("MyVideos") + "\DownloadFiles"
18: $fileNameFormat = "%(title)s.%(ext)s"
19: $fileSavePath = "${saveFolderPath}\${fileNameFormat}"
20:
21: # URLリスト一覧のファイルがあるかどうか確認
22: # Note: urlListPathオプションの値がある場合はパスの存在をチェック
23: if ($urlListPath -ne "" -and
24:     (-not (Test-Path -Path $urlListPath -PathType Leaf))) {
25:     Write-Error "URL指定用のファイルが見つかりません"
26:     exit
27: }
28:
29: # 保存先フォルダの作成
30: if (-not (Test-Path -Path $saveFolderPath -PathType Container)) {
31:     New-Item -Path $saveFolderPath -ItemType Directory -Force > $null
32: }
33:
34: # ダウンロード処理の実行
35: # Note: -Nオプションでマルチスレッドでのダウンロード処理が可能
```

```

36: if ($urlListPath -ne "") {
37:     yt-dlp -a $urlListPath -o $fileSavePath `
38:         -N $threads --embed-thumbnail --windows-filenames --console-title
39: } else {
40:     yt-dlp $url -o $fileSavePath `
41:         --embed-thumbnail --windows-filenames --console-title
42: }
43:
44: # 保存先フォルダの表示
45: Write-Host "保存先フォルダ: ${saveFolderPath}" -ForegroundColor Cyan

```

4.11 応用サンプル11: Webページ上画像ファイルの一括ダウンロード

4.11.0.1 解説

ここでは指定のWebページ上からHTMLを取得します。Invoke-WebRequest -Uri (HTML取得先URL)とすることでHTMLを取得できます。ただし、単にHTML文字列を取得するだけで、すでにパース済みのデータは取得できないことに注意が必要です。

かつては、ParsedHtmlを指定するとパース済みのHTMLデータが使えましたが、これは内部でInternet Explorerのエンジンを使っていました。近年のInternet Explorer廃止・Edgeへの移行に伴ってこの機能は使えなくなっています。HTMLのパース済みデータが欲しい場合、本格的にスクレイピングを行いたい場合は別途ライブラリを導入する必要があります。

このサンプル内ではaタグとimgタグの文字列から直接画像のURLを取り出すサンプルなアプローチを取っています。単にURLを取り出したいだけであればパース処理までせずとも正規表現を使った文字列処理で十分に対応可能であるためです。

具体的には、<a>タグ内のhref属性と、imgタグ内のsrc属性を取り出せばよいことになります。

```

<a href="...">(リンクテキスト)</a>


```

4.11.0.2 サンプルコード

リスト4.11: 応用サンプル(11)

```

1: # スクリプトの引数定義
2: Param(
3:     [string]$url = "",
4:     [string]$inputPath,
5:     [string]$urlListPath,
6:     [int]$minWidth=300,
7:     [int]$minHeight=300,

```

```

8:      [switch]$noDelete=$false
9: )
10:
11: # アセンブリの読み込み
12: Add-Type -AssemblyName System.Drawing
13:
14: # 対応画像フォーマット
15: $imageExtensions = @(".png", ".jpg", ".gif")
16:
17: # 変数の初期化
18: $pageUrlArray = @()
19: $saveFolderRoot = [Environment]::GetFolderPath("MyPictures") + "\MyImages"
20: $failedUrlTextPath = "${saveFolderPath}\failed_url.txt"
21:
22: # -urlオプションからのURL取得
23: if ($url -ne "") {
24:     $pageUrlArray += $url
25: }
26:
27: # 入力ファイルからのURL取得
28: # Note: 1行あたり1つのURLが書かれているものとして読み込み処理
29: if ($urlListPath -ne "" -and `
30:     (Test-Path -Path $urlListPath -PathType Leaf)) {
31:     foreach ($line in (Get-Content $inputPath)) {
32:         $pageUrlArray += $line.Trim()
33:     }
34: }
35:
36: # URL入力数のチェック
37: if ($pageUrlArray.Count -eq 0) {
38:     Write-Host "URLが入力されていません"
39:     exit
40: }
41:
42: # ダウンロード処理の実行
43: foreach ($pageUrl in $pageUrlArray) {
44:     # HTTPリクエスト処理の実行
45:     # Note: アクセス失敗したURLではメッセージの表示・ログ書き込み処理を実施
46:     $req = Invoke-WebRequest -Uri $pageUrl
47:     if ($req.StatusCode -ne 200) {
48:         Write-Host "指定されたページは利用できません"

```

```

49:         Write-Host "URL: ${pageUrl}"
50:         Write-Output $pageUrl | Out-File $failedUrlTextPath
51:         continue
52:     }
53:
54:     # 保存先パスに関する変数
55:     $dateString = Get-Date -Format "yyyyMMddHHmmss"
56:     $saveFolderPath = "${saveFolderRoot}\${dateString}"
57:     $pageInfoFilePath = "${saveFolderRoot}\${dateString}\page_info.txt"
58:
59:     # HTMLからURL文字列を取得
60:     $html = $req.Content
61:     $fetchUrls = @()
62:
63:     # 正規表現を使ったURL情報の抜き出し(HTMLパーサなし)
64:     # Note: 現在はParsedHtmlが利用できないため生HTML文字列を使う
65:     # imgタグ内のsrcの取り出し
66:     $match1 = [regex]::Matches($html, 'src="(http.+?)"' )
67:     $match1 | ForEach-Object {
68:         $tempUrl = $_.Groups[1].Value
69:         $fetchUrls += $tempUrl.Split("?")[0]
70:     }
71:
72:     # aタグ内のhrefの取り出し
73:     $match2 = [regex]::Matches($html, 'href="(http.+?)"' )
74:     $match2 | ForEach-Object {
75:         $tempUrl = $_.Groups[1].Value
76:         $fetchUrls += $tempUrl.Split("?")[0]
77:     }
78:
79:     # URLのフィルタリング・重複の除去およびソート
80:     # Note: 配列$fetchUrlsの中で、指定の拡張子をもつ画像のみ取り出す
81:     $imageUrls = $fetchUrls |
82:         Where-Object {
83:             [System.IO.Path]::GetExtension($_) -in $imageExtensions } |
84:         Get-Unique | Sort-Object
85:
86:     # ダウンロード可能な画像がない場合のメッセージ
87:     if ($imageUrls.Count -eq 0) {
88:         Write-Host "ダウンロード可能な画像がありません。"
89:         continue

```

```

90:     }
91:
92:     # 保存先フォルダの作成
93:     if (-not (Test-Path -Path $saveFolderPath -PathType Container)) {
94:         New-Item -Path $saveFolderPath -ItemType Directory -Force > $null
95:     }
96:
97:     # 画像ファイルのダウンロード処理
98:     # Note: ファイル名は番号として作成する
99:     $swc = New-Object System.Net.WebClient
100:     $count = 0
101:     foreach($link in $imageUrls) {
102:         $fileExt = ([uri]$link).Segments[-1]
103:         $fileName = "{0:03d}" -f ($count + 1)
104:         $filePath = "${saveFolderPath}\${fileName}${fileExt}"
105:         try {
106:             $swc.DownloadFile($link, $filePath)
107:         } catch {
108:             Write-Host "注意: ダウンロードに失敗しました(URL: ${link})"
109:         }
110:     }
111: }
112:
113: # 指定サイズ以下の画像の削除
114: # Note: -noDeleteオプションが指定されている場合はファイル削除処理は実行されない
115: if (-not $noDelete) {
116:     $deleteFilePathArray = @()
117:
118:     # 削除対象画像ファイルパスの取得
119:     Get-ChildItem "${saveFolderPath}\*" -Include $imageFormats |
120:     ForEach-Object {
121:         $img = [System.Drawing.Image]::FromFile($_.FullName);
122:         if (($img.Width -lt $minWidth) -or ($img.Height -lt $minHeight))
123:         {
124:             $deleteFilePathArray += $_.FullName
125:         }
126:         $img = $null
127:     }
128:
129:     # 画像ファイルの削除処理
130:     foreach($path in $deleteFilePathArray) {

```



```

130:         $errorCount = 0
131:         while($errorCount -lt 3) {
132:             try {
133:                 # Note: ErrorAction指定によりエラー時の挙動を制御する
134:                 Remove-Item $path -ErrorAction:Stop
135:                 break
136:             } catch {
137:                 Write-Host "注意: 削除に失敗しました(Path: ${path})"
138:                 $errorCount++
139:                 Start-Sleep 3
140:                 continue
141:             }
142:         }
143:     }
144: }
145:
146: # ダウンロード先ページ情報の出力
147: Write-Output "URL: ${inputUrl}" | Out-File $pageInfoFilePath
148:
149: # 保存先フォルダの表示
150: Write-Host "保存先フォルダ: ${saveFolderPath}" -ForegroundColor Cyan

```

4.12 応用サンプル12: 連続スクリーンショット取得・キー押下自動化

このサンプルではマウスクリックによってスクリーンショット範囲を指定し、その指定範囲についてのスクリーンショットを連続して取得します。

スクリーンショットの取得間隔は適宜変更でき、スクリーンショットを撮る直前のキー操作も自動で行えるようにします。特定のページやアプリケーション上のスクリーンショットを手動での操作なしに連続で取りたい場合に使えます。

基本的にはディスプレイ上で見えているものと同じ画像を取ることができますが、著作権保護があるようなアプリケーションの場合、今回のサンプルでスクリーンショットを取得しようとしても真っ黒な画像になってしまう場合があります。

4.12.0.1 使用時の注意

- ・ディスプレイのDPI設定・画面解像度が100%でない場合には意図した座標のスクリーンショットが取得できない場合があるため、ご注意ください。
- ・サンプルコードにはキー送信の機能が含まれます。これにより『このスクリプトには、悪質なコンテンツが含まれているため、ウィルス対策ソフトウェアによりブロックされています』という表示がされてコードを実行できないことがあります。"更新とセキュリティ→Windows セ

セキュリティ→ウィルスと脅威の防止"と進み、リアルタイム保護をいったん無効にすることでスクリプトの実行が可能となります。

4.12.0.2 サンプルコード

リスト4.12: 応用サンプル(12)

```
1: # スクリプトの引数定義
2: Param([switch]$reuse)
3:
4: # アセンブリの読み込み
5: Add-Type -AssemblyName System.Windows.Forms
6: Add-Type -AssemblyName System.Drawing
7:
8: # マウスイベントの定義
9: $signature = @'
10: [DllImport("user32.dll")]
11: public static extern void mouse_event(long dwFlags, long dx, long dy, long
cButtons, long dwExtraInfo);
12: '@
13: $global:mouseEvent = Add-Type -MemberDefinition $signature -Name
"Win32MouseEvent" -Namespace Win32Functions -PassThru
14:
15: # キャプチャ関連設定のクラス(json形式用)
16: class CaptureConfig {
17:     [int] $rectX
18:     [int] $rectY
19:     [int] $rectWidth
20:     [int] $rectHeight
21:     [int] $waitSecond
22:     [string] $imageExt
23:     [string] $autoPressKey
24:     [int] $autoClickPosX
25:     [int] $autoClickPosY
26: }
27:
28: # グローバル変数定義
29: $global:pageCountLimit = 500
30: $global:waitSecondDefault = 1
31: $global:waitSecondLimit = 20
32: $global:imageFormatDefault = [System.Drawing.Imaging.ImageFormat]::Jpeg
33: $global:imageExtDefault = ".jpg"
34: $global:autoPressKeyDefault = "{Right}"
35:
```

```

36: # 関数の定義
37: # 拡張子文字列からの画像フォーマットの取得
38: function Get-ImageFormatFromExt($selectExt) {
39:     switch ($selectExt.ToLower()) {
40:         ".jpg" { return [System.Drawing.Imaging.ImageFormat]::Jpeg }
41:         ".png" { return [System.Drawing.Imaging.ImageFormat]::Png }
42:         ".bmp" { return [System.Drawing.Imaging.ImageFormat]::Bmp }
43:         default { return $global:imageFormatDefault }
44:     }
45: }
46:
47: # 画像形式の選択処理
48: function Get-ImageFormatAndExt() {
49:     $imageExt = $global:imageExtDefault
50:     $inputString = Read-Host -Prompt `
51:         "画像形式を選択してください ... 1: jpg, 2: png, 3: bmp (デフォルト: jpg)"
52:     $selectNumber = [int]$inputString
53:
54:     switch ($selectNumber) {
55:         1 { $imageExt = ".jpg" }
56:         2 { $imageExt = ".png" }
57:         3 { $imageExt = ".bmp" }
58:         default { $imageExt = $global:imageExtDefault }
59:     }
60:
61:     $imageFormat = Get-ImageFormatFromExt $imageExt
62:     return $imageFormat, $imageExt
63: }
64:
65: # ページ数の入力処理
66: function Get-PageCount() {
67:     $pageCount = 0
68:     $inputString = Read-Host -Prompt "ページ数を入力してください"
69:     $pageCount = [int]$inputString
70:
71:     if($pageCount -gt $global:pageCountLimit) {
72:         Write-Host "指定可能なページ数は上限${pageCountLimit}です"
73:         $pageCount = $global:pageCountLimit
74:     }
75:
76:     return $pageCount

```

```

77: }
78:
79: # マウスクリックによる矩形領域の取得
80: function Get-DragRectArea() {
81:     Write-Host "マウスをドラッグして矩形領域を選択してください(左クリックで開始)"
82:     while ([System.Windows.Forms.Control]::MouseButtons -ne 'Left') {
83:         Start-Sleep 0.5 }
84:     $p1 = [System.Windows.Forms.Control]::MousePosition
85:     Write-Host "Pressed"
86:
87:     while ([System.Windows.Forms.Control]::MouseButtons -ne 'None') {
88:         Start-Sleep 0.5 }
89:     $p2 = [System.Windows.Forms.Control]::MousePosition
90:     Write-Host "Released"
91:
92:     $rect = [System.Drawing.Rectangle]::FromLTRB(
93:         [Math]::Min($p1.X, $p2.X), [Math]::Min($p1.Y, $p2.Y),
94:         [Math]::Max($p1.X, $p2.X), [Math]::Max($p1.Y, $p2.Y))
95:
96:     return $rect
97: }
98:
99: # マウスクリック座標の座標取得
100: function Get-ClickPos() {
101:     Write-Host "自動でマウスクリックする座標を指定して下さい(左クリック)"
102:     while ([System.Windows.Forms.Control]::MouseButtons -ne 'Left') {
103:         Start-Sleep 0.5 }
104:     $pos = [System.Windows.Forms.Control]::MousePosition
105:     Write-Host "Pressed"
106:     return $pos
107: }
108:
109: # キャプチャ間隔秒数の取得
110: function Get-WaitSecond() {
111:     $inputString = Read-Host -Prompt `
112:         "キャプチャ間隔(秒数)を入力してください (デフォルト: 1秒)"
113:     if ($inputString -eq "") { return $global:waitSecondDefault }
114:
115:     $waitSecond = [int]$inputString
116:     if($waitSecond -gt $waitSecondLimit) {
117:         Write-Host "指定可能なキャプチャ間隔は上限${waitSecondLimit}秒です"

```

```

118:         $waitSecond = $global:waitSecondLimit
119:     } elseif ($waitSecond -le 0) {
120:         $waitSecond = $global:waitSecondDefault
121:     }
122:
123:     return $waitSecond
124: }
125:
126: # 自動押下キーの取得
127: function Get-PressKey() {
128:     $pressKey = $global:autoPressKeyDefault
129:     $inputString = Read-Host -Prompt `
130:         "自動押下キーを選択してください ... 1: Right(→), 2: Left(←), 3: Enter,
131:         4: Click (デフォルト: Right)"
132:     $selectNumber = [int]$inputString
133:     switch ($selectNumber) {
134:         1 { $pressKey = "{Right}" }
135:         2 { $pressKey = "{Left}" }
136:         3 { $pressKey = "{Enter}" }
137:         4 { $pressKey = "{Click}" }
138:     }
139:
140:     return $pressKey
141: }
142:
143: # スクリーンショットの取得・保存処理
144: function Get-Screenshot($rect, $saveFilePath, $imageFormat) {
145:     $bitmap = New-Object System.Drawing.Bitmap($rect.Width, $rect.Height)
146:     $graphics = [System.Drawing.Graphics]::FromImage($bitmap)
147:     $graphics.CopyFromScreen($rect.Left, $rect.Top, 0, 0, $bitmap.Size)
148:     $bitmap.Save($saveFilePath, $imageFormat)
149:     $graphics.Dispose()
150:     $bitmap.Dispose()
151: }
152:
153: # 変数宣言
154: # ファイルパス関連の変数
155: $dateString = Get-Date -Format "yyyyMMddHHmmss"
156: $saveFolderRoot = [Environment]::GetFolderPath("MyPictures") +
    "\MyScreenshots"

```

```

157: $saveFolderPath = "${saveFolderRoot}\${dateString}"
158: $captureConfigFilePath = "${saveFolderRoot}\capture_config.json"
159:
160: # キャプチャ設定関連の変数
161: $conf = New-Object CaptureConfig
162: $configExists = $reuse -and (Test-Path -Path $captureConfigFilePath)
163: $pageCount = 0
164: $imageFormat = $global:imageFormatDefault
165: $captureRect = [System.Drawing.Rectangle]::FromLTRB(0, 0, 0, 0)
166:
167: # スクリーンショット取得設定が存在する場合は読み込む
168: if ($configExists) {
169:     # jsonの読み込み処理
170:     $json = Get-Content -Path $captureConfigFilePath | ConvertFrom-Json
171:
172:     # jsonから読み込んだ情報によって設定情報を初期化
173:     $conf.rectX = $json.rectX
174:     $conf.rectY = $json.rectY
175:     $conf.rectWidth = $json.rectWidth
176:     $conf.rectHeight = $json.rectHeight
177:     $conf.waitSecond = $json.waitSecond
178:     $conf.imageExt = $json.imageExt
179:     $conf.autoPressKey = $json.autoPressKey
180:     $conf.autoClickPosX = $json.autoClickPosX
181:     $conf.autoClickPosY = $json.autoClickPosY
182:     $imageFormat = Get-ImageFormatFromExt $conf.imageExt
183:     $captureRect = [System.Drawing.Rectangle]::FromLTRB(
184:         $conf.rectX, $conf.rectY,
185:         $conf.rectX + $conf.rectWidth,
186:         $conf.rectY + $conf.rectHeight)
187:
188:     Write-Host "`n前回のスクリーンショット取得設定を再利用します`n"
189:
190:     # 取得ページ数の入力
191:     do {
192:         $pageCount = Get-PageCount
193:         $response = Read-Host -Prompt `
194:             "保存ページ数: ${pageCount} ... OK?(y/n)"
195:     } until ($response -eq 'y')
196:
197: } else {

```

```

198:      # 画像フォーマットの取得
199:      $imageFormat, $conf.imageExt = Get-ImageFormatAndExt
200:
201:      # 取得ページ数の入力
202:      do {
203:          $pageCount = Get-PageCount
204:          $response = Read-Host -Prompt `
205:              "保存ページ数: ${pageCount} ... OK?(y/n)"
206:      } until ($response -eq 'y')
207:
208:      # スクリーンショット対象となる矩形領域の決定
209:      do {
210:          $captureRect = Get-DragRectArea
211:          $conf.rectX, $conf.rectY = $captureRect.X, $captureRect.Y
212:          $conf.rectWidth, $conf.rectHeight = `
213:              $captureRect.Width, $captureRect.Height
214:          $response = Read-Host -Prompt `
215:              "指定矩形領域: ${captureRect} ... OK?(y/n)"
216:      } until ($response -eq 'y')
217:
218:      # スクリーンショット取得間隔の決定
219:      $conf.waitSecond = 1
220:      do {
221:          $conf.waitSecond = Get-WaitSecond
222:          $response = Read-Host -Prompt `
223:              "キャプチャ間隔: ${($conf.waitSecond)} sec ... OK?(y/n)"
224:      } until ($response -eq 'y')
225:
226:      # スクリーンショット取得完了後の対象入力キーの決定
227:      $conf.autoPressKey = "{Right}"
228:      do {
229:          $conf.autoPressKey = Get-PressKey
230:          $response = Read-Host -Prompt `
231:              "自動押下キー: ${($conf.autoPressKey)} ... OK?(y/n)"
232:      } until ($response -eq 'y')
233:
234:      # 入力キーとしてクリック指定の場合はクリック座標も取得
235:      if ($conf.autoPressKey -eq '{Click}') {
236:          do {
237:              $pos = Get-ClickPos
238:              $conf.autoClickPosX = $pos.X

```

```

239:             $conf.autoClickPosY = $pos.Y
240:             $response = Read-Host -Prompt `
241:                 "クリック座標: $($pos.X), $($pos.Y)) ... OK?(y/n)"
242:         } until ($response -eq 'y')
243:     }
244: }
245:
246: # キャプチャ開始前のメッセージ表示・スリープ時間
247: Write-Host "`n対象となるウィンドウをアクティブ状態にしてください。"
248: Write-Host "スクリーンショット開始後は画面操作を行わないでください。"
249: Write-Host "10秒後にスクリーンショット取得を開始します..."
250: Start-Sleep 10
251:
252: # 保存先フォルダの作成
253: if (-not (Test-Path -Path $saveFolderPath)) {
254:     New-Item -Path $saveFolderPath -ItemType Directory -Force > $null
255: }
256:
257: # 設定情報の出力
258: Write-Host "`n[各種設定情報]"
259: Write-Host "- 保存先フォルダ: ${saveFolderPath}"
260: Write-Host "- 画像の保存形式: $($conf.imageExt)"
261: Write-Host "- 保存ページ数: ${pageCount} ページ"
262: Write-Host "- 指定矩形領域: ${captureRect}"
263: Write-Host "- キャプチャ間隔: $($conf.waitSecond) 秒"
264: Write-Host "- 自動押下キー: $($conf.autoPressKey)"
265:
266: # スクリーンショットの取得処理実行
267: for ($i=1; $i -le $pageCount; $i++){
268:     # ファイル保存先パスの指定
269:     $saveFilePath = "{0}\{1:000}{2}" -f $saveFolderPath, $i, $conf.imageExt
270:
271:     # スクリーンショット取得
272:     Get-Screenshot $captureRect $saveFilePath $imageFormat
273:     if ($i % 10 -eq 0) { Write-Host "${i}ページ取得完了..." }
274:
275:     # キー押下処理の実行
276:     if ($conf.autoPressKey -eq "{Click}") {
277:         # マウスカーソル移動および左クリックイベントの発火
278:         [System.Windows.Forms.Cursor]::Position = New-Object
System.Drawing.Point(

```



```

279:         $conf.autoClickPosX, $conf.autoClickPosY)
280:         $global:mouseEvent::mouse_event(0x0002, 0, 0, 0, 0);
281:         $global:mouseEvent::mouse_event(0x0004, 0, 0, 0, 0);
282:     } else {
283:         # キーボード内のキー押下の実行
284:         [System.Windows.Forms.SendKeys]::SendWait($conf.autoPressKey)
285:     }
286:
287:     # 指定秒数分のスリープ
288:     Start-Sleep $conf.waitSecond
289: }
290:
291: # 現在のキャプチャ設定情報の書き出し(json形式)
292: $utf8NoBom = New-Object System.Text.UTF8Encoding $false
293: [System.IO.File]::WriteAllLines($captureConfigFilePath, `
294:     (ConvertTo-Json $conf), $utf8NoBom)
295:
296: # 完了メッセージの表示 & # 保存先フォルダの表示
297: Write-Host "スクリーンショット取得処理を完了しました"
298: Write-Host "保存先フォルダ: ${saveFolderPath}" -ForegroundColor Cyan

```

4.13 応用サンプル13: URL情報のQRコードへの変換

4.13.0.1 下準備

このサンプルの実行には、.NET用の"QrCodeGenerator"のライブラリのインストールが必要です。

```
PS > dotnet add package Net.Codecrete.QrCodeGenerator
```

4.13.0.2 解説

文字列からのQRコードの生成については完全に"QrCodeGenerator"に任せる形を取っています。ここでは入力したURLをQRコード化してその画像を出力するサンプルを掲載しています。

4.13.0.3 サンプルコード

リスト4.13: 応用サンプル(13)

```

1: # スクリプトの引数定義
2: Param(
3:     [string]$url = "",
4:     [int]$width = 15
5: )

```

```

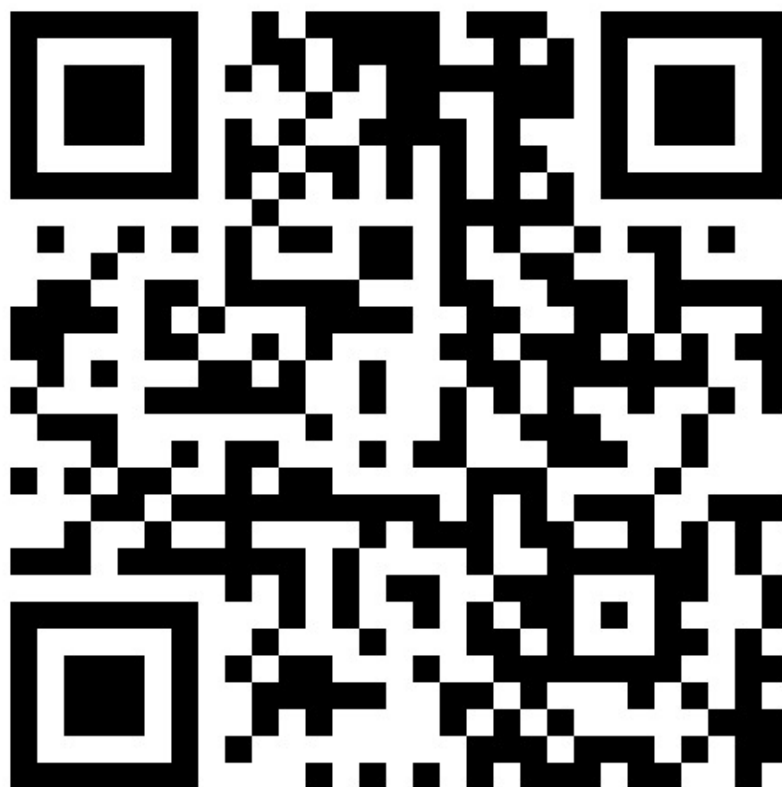
6:
7: # QRコード画像保存先フォルダ・ファイルパスの初期化
8: $dateString = Get-Date -Format "yyyyMMddHHmmss"
9: $fileName = "\${dateString}.png"
10: $saveFolderPath = `
11:     [Environment]::GetFolderPath("MyPictures") + "\MyQRcodes"
12: $saveFilePath = "${saveFolderPath}\${fileName}"
13:
14: # URLアクセスのチェック
15: $req = Invoke-WebRequest -Uri $url
16: if ($req.StatusCode -ne 200) {
17:     Write-Error "アクセスできません`nステータスコード: ${req.StatusCode}"
18:     exit
19: }
20:
21: # 保存先フォルダの作成
22: if (-not (Test-Path -Path $saveFolderPath -PathType Container)) {
23:     New-Item -Path $saveFolderPath -ItemType Directory -Force > $null
24: }
25:
26: # QRcode画像の生成・保存
27: try {
28:     New-PSOneQRCodeURI -URI $url -Width $width -OutPath $saveFilePath
29:     Write-Host "QRコード画像の生成が完了しました"
30:     Write-Host "QRコード画像パス: ${saveFilePath}"
31: } catch {
32:     Write-Error "新しいQRコード生成に失敗しました"
33: }

```

4.13.0.4 実行結果

```
PS > .\sample_advanced13_qrcode_url.ps1 -url https://yahoo.co.jp
```

図: QRコードの出力ファイル(Yahoo!トップ)



4.14 応用サンプル14: QRコードからのワンタイムパスワードの読み出し

4.14.0.1 下準備

このサンプルの実行には"Zbar"のインストールが必要です。

```
PS > choco install -y zbar
```

4.14.0.2 解説

ここではワンタイムパスワード生成用のQRコードから、ワンタイムパスワードの6ケタを出力するサンプルを挙げます。

普段中身を意識することはほとんどないワンタイムパスワードですが、バーコード・QRコード解読用ライブラリとPowerShellを併用してワンタイムパスワードをコマンドライン上で確認する処理を作ることができます。

秘密鍵について

QRコードから文字列を読み出すと以下のような構造になっています。

```
otpauth://totp/[issuer]:[accountname]?secret=[secret]
```

計算に必要なのが秘密鍵(secret)の文字列です。これが分かれば、RFCで定義されたワンタイムパスワード計算の手順に従って最終的な6桁の出力値を求めることができます。

まず秘密鍵はBase32文字列として設定されており、これを変換ルールにのっとって1文字ずつ5ビットの2進数に変換し、これらをすべて文字列として結合します。結合して得た文字列をハッシュ関数用の鍵としてセットしておきます。

ハッシュ関数に与える入力値

ハッシュ関数用の鍵をセットできたら、次は入力値を与えます。具体的には現在時刻のUNIX時間での経過秒数を30で割った値をハッシュ関数に与える入力とします(つまり30秒が経過するたびにパスワードが変化することを意味します)。

ハッシュ値への操作とワンタイムパスワード値

秘密鍵とUNIX時間から得た入力値をHMAC-SHA1関数に与えます。すると、出力としてハッシュ値(20バイト/160ビットの文字列)を得ます。このHMAC-SHA1関数から得られたハッシュ値について以下操作をします。

- ・ 末尾1文字(1バイト/8ビット)を取り出す
- ・ 末尾1文字のうち下位4ビットを整数に変換し、オフセット値として使う
- ・ ハッシュ値全体から、(オフセット値)～(オフセット値+3)の部分について4バイト/32ビット分だけ文字列を切り出す
- ・ 4バイト/32ビットのうち上位1ビットを捨て、下位31ビットを取り出したのち整数値に変換する
- ・ 得られた整数値を1,000,000で割った剰余をパスワードとする(6桁にならない場合は先頭を0埋めする)

4.14.0.3 サンプルコード

リスト4.14: 応用サンプル(14)

```
1: # スクリプトの引数定義
2: Param(
3:     [string] $inputPath,
4:     [switch] $showSecret=$false
5: )
6:
7: # 関数の定義
8: function Convert-Base32ToByte($secret) {
```

```

 9:     $base32Chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ234567"
10:     $bits = ""
11:     $output = @()
12:
13:     # Base32のRFC定義にもとづくビット値の計算
14:     foreach ($char in $secret.ToUpper().ToCharArray()) {
15:         $bits += [Convert]::ToString(
16:             $base32Chars.IndexOf($char), 2).PadLeft(5, '0')
17:     }
18:
19:     # 8ビットのチャンクをバイト値に変換、最後のビットは無視する
20:     for ($i = 0; $i -le ($bits.Length - 8); $i += 8) {
21:         $output += [Byte][Convert]::ToInt32(
22:             $bits.Substring($i, 8), 2)
23:     }
24:     return $output
25: }
26:
27: function Get-OTPCodeInfo($secret) {
28:     # Unix時間の計算・ハッシュ関数へ渡す値の準備
29:     $now = Get-Date
30:     $numberOfSeconds = ([datetimeoffset]$now).ToUnixTimeSeconds()
31:     $numberOfIntervals = [Convert]::ToInt64([Math]::Floor($numberOfSeconds /
32: 30))
33:     $remainingSec = 30 - ($numberOfSeconds % 30)
34:     $validDeadLine = $now.AddSeconds($remainingSec)
35:
36:     # Unix時間由来のバイト列取得
37:     $intervalByteArray = [BitConverter]::GetBytes($numberOfIntervals)
38:     [Array]::Reverse($intervalByteArray)
39:
40:     # HMAC-SHA1によるハッシュ処理
41:     $objHMACSHA1 = New-Object -TypeName System.Security.Cryptography.HMACSHA1
42:     $objHMACSHA1.key = Convert-Base32ToByte($secret)
43:     $hashByteArray = $objHMACSHA1.ComputeHash($intervalByteArray)
44:
45:     # ハッシュ値周りの計算処理
46:     $offset = $hashByteArray[-1] -band 0xf
47:     $otpBits = (($hashByteArray[$offset] -band 0x7f) -shl 24) -bor
48:         (($hashByteArray[$offset+1] -band 0xff) -shl 16) -bor
49:         (($hashByteArray[$offset+2] -band 0xff) -shl 8) -bor

```

```

49:             (($hashByteArray[$offset+3] -band 0xff))
50:
51:     # 下位6桁をコードとして取得
52:     $otpInt = $otpBits % 1000000
53:     $otpCode = $otpInt.ToString().PadLeft(6, '0')
54:
55:     # 返却値
56:     $info = @{
57:         RemainingSec = $remainingSec;
58:         ValidDeadline = $validDeadLine;
59:         OTPCode = $otpCode;
60:     }
61:
62:     return $info
63: }
64:
65: # 入力パスの存在確認
66: if (-not (Test-Path -Path $inputPath -PathType Leaf)) {
67:     Write-Error "入力ファイルが見つかりません"
68:     exit
69: }
70:
71: # QRコードからの文字列読み込み
72: # Note: zbarimg内部でQRコードの読込処理を行ってくれる
73: $zbarResult = (zbarimg $inputPath --quiet)
74: if ("ERROR" -match $zbarResult) {
75:     Write-Error "読み込みに失敗しました。ファイル形式を確認してください"
76:     exit
77: }
78:
79: # 秘密鍵の文字列データの取得
80: # Note: 取得した文字列中の "secret=" 以下に秘密鍵が格納されている
81: $match = [regex]::Matches($zbarResult, "QR-Code:.*secret\=(.+?)\&.*")
82: $secret = $match.Groups[1].Value
83:
84: # ワンタイムパスワードの取得
85: $info = Get-OTPCodeInfo $secret
86:
87: # ワンタイムパスワードおよび有効期限情報の出力
88: Write-Host "[ワンタイムパスワード情報]"
89: if ($showSecret) { Write-Host "- 秘密鍵: ${secret}" }

```

```
90: Write-Host "- OTPCode: $($info["OTPCode"])"
91: Write-Host "- 有効期限: $($info["ValidDeadline"])"
92: Write-Host "- 残り時間: $($info["RemainingSec"]) sec"
```

4.14.0.4 実行結果

```
# ワンタイムパスワードの表示例
PS > .\sample_advanced14.ps1 -inputPath .\files\sample_qrcode.png
[ワンタイムパスワード情報]
- OTPCode: 821073
- 有効期限: 03/12/2024 20:02:30
- 残り時間: 30 sec

# ワンタイムパスワードと秘密鍵を同時に表示する例
PS > .\sample_advanced14.ps1 -inputPath .\files\sample_qrcode.png -showSecret
[ワンタイムパスワード情報]
- 秘密鍵: (秘密鍵の文字列が表示されます)
- OTPCode: 260306
- 有効期限: 03/12/2024 20:03:00
- 残り時間: 26 sec
```

(※inputPathにワンタイムパスワード生成用の有効なQRコードの画像ファイルを指定してください)

4.15 応用サンプル15: Yahoo!ニュースヘッドライン抽出ツール(CUI)

4.15.0.1 解説

Yahoo!トップページに表示されるニュースヘッドラインを取得するサンプルです。

・URL:<https://www.yahoo.co.jp/>

主要 | 経済 | エンタメ | スポーツ | 国内 | 国際 | IT | 科学 | 地域

3/10(日) 9:01更新

- 米向け装備部品増産へ ウ支援支え **NEW**
- 3/11追悼式 沿岸自治体で半数割る **NEW** 193
- ガザ物資投下 また市民に直撃か 790
- 死亡ひき逃げ疑い ダンプの男逮捕 **NEW** 389
- 半導体に沸く熊本 地元企業の悩み 441
- スノボ元代表 藤森由香さんが出産 207
- 宮城野部屋 力士ら全員を転籍へ 857
- 世界ふしぎ発見 最終回は3時間SP **NEW** 605

もっと見る トピックス一覧



「春麗」設置

3/10(日) 6:10
奈良新聞デジタル

Yahoo!トップページ内では、ニュースヘッドラインの表示部分についてはtabpanelTopics1のIDをもつdivタグが設定されており、このタグ内の文字列を処理することで文字列を取り出すことができます。このサンプルでは更新日時・ニュース内容・記事URLを取得してシェル上に内容を出力します。

4.15.0.2 サンプルコード

リスト4.15: 応用サンプル(15)

```
1: # URLの指定
2: $yahooTopUrl = "https://www.yahoo.co.jp/"
3:
4: # リクエスト処理の実行
5: $req = Invoke-WebRequest -Uri $yahooTopUrl
6: if ($req.StatusCode -ne 200) {
7:     Write-Error "Yahoo!ページにアクセスできません"
8:     exit
9: }
10:
11: # HTMLテキストの取得
12: $html = $req.Content
13:
```



```

14: # 正規表現を使った文字列の抜き出し
15: # Note: IE廃止以後はParsedHtmlが利用できないため生のHTML文字列から直接文字列を取り出す
16: # ヘッダー部分のテキスト取得
17: $match1 = [regex]::Matches($html,
18:     '<section id="tabpanelTopics1".*>(.*?)</section>')
19: $topicText = $match1.Groups[1].Value
20:
21: # 更新日時の取得
22: $match2 = [regex]::Matches($topicText,
23:     '([0-9]{1,2}/[0-9]{1,2}.+[0-9]{1,2}:[0-9]{2}更新)')
24: $updateTimeString = $match2.Groups[1].Value
25:
26: # ulタグ内の文字列の取得
27: $match3 = [regex]::Matches($topicText, '<ul>(.*?)</ul>')
28: $ulInnerText = $match3.Groups[1].Value
29:
30: # ulタグ内の文字列からさらにURL/タイトルの文字列を抜き出す
31: $outputLineArray = @()
32: $match4 = [regex]::Matches($ulInnerText,
33:     '<a class=.*? href="(.*?)".*?>.*?<h1 class.*?><span.*?>(.*?)</span></h1>')
34: $match4 | ForEach-Object {
35:     $url = $_.Groups[1].Value
36:     $title = $_.Groups[2].Value
37:     $outputLineArray += "- ${title}`nURL: ${url}"
38: }
39:
40: # 取得した文字列の出力
41: Write-Host "Yahoo!ヘッダー ( $updateTimeString )"
42: $outputLineArray | ForEach-Object { Write-Host $_ }

```

4.15.0.3 実行結果

```

PS > .\sample_advanced15.ps1
Yahoo!Japanヘッダー (2/25(日) 15:32 更新)
- イスラエル、6週間休戦案で合意か
URL: https://news.yahoo.co.jp/pickup/6492728
- マリウポリ占領「ロシア化」進む
URL: https://news.yahoo.co.jp/pickup/6492715
- 東大寺の参道に車 2人はね1人死亡
URL: https://news.yahoo.co.jp/pickup/6492727
- 国公立大2次試験開始 23万人志願
URL: https://news.yahoo.co.jp/pickup/6492729
- 異例 インフルエンザ2回目ピーク

```

URL: <https://news.yahoo.co.jp/pickup/6492725>
- 物件内覧できず 断られる高齢者
URL: <https://news.yahoo.co.jp/pickup/6492723>
- 国学院大・平林 初マラソン衝撃V
URL: <https://news.yahoo.co.jp/pickup/6492721>
- あばれる君 妻と無言バトンタッチ
URL: <https://news.yahoo.co.jp/pickup/6492732>

4.16 応用サンプル16: Yahoo!路線検索結果表示ツール(CUI)

4.16.01 解説

Yahoo!路線検索の検索結果を取得するサンプルです。

・URL:<https://transit.yahoo.co.jp/>

このサンプルコードではより簡易に結果を取得できるように、路線情報の印刷用ページをもとに路線検索の結果を取得します。

図: Yahoo!路線検索結果(印刷用ページ)

YAHOO!
JAPAN

路線情報

印刷する

東京→渋谷 2024年03月10日(日)10:51出発

10:52発→11:14着 22分(乗車18分) 乗換: 1回 7.7km
IC優先: 209円 定期券 通勤: 1か月 7,830円 / 3か月 22,320円 / 6か月 42,290円

10:52 発 東京 乗車位置: [6両] 中 後
4駅 東京メトロ丸ノ内線 荻窪行 [発] 1番線 → [着] 2番線 209円

11:01着
11:05発 赤坂見附
4駅 東京メトロ銀座線 渋谷行 [発] 1番線 → [着] 1番線

11:14 着 渋谷

URL 指定時は、以下の形式によってHTMLが取得できます。

目的指定のみ・日時指定なし(現在時刻での検索)

[https://transit.yahoo.co.jp/search/print?from=\(出発駅名\)&to=\(到着駅名\)](https://transit.yahoo.co.jp/search/print?from=(出発駅名)&to=(到着駅名))

時刻指定あり

[https://transit.yahoo.co.jp/search/print?from=\(出発駅名\)&to=\(到着駅名\)&y=\(年\)&m=\(月\)&d=\(日\)&h=\(時間\)&m1=\(分/十の位\)&m2=\(分/一の位\)](https://transit.yahoo.co.jp/search/print?from=(出発駅名)&to=(到着駅名)&y=(年)&m=(月)&d=(日)&h=(時間)&m1=(分/十の位)&m2=(分/一の位))

ページのレイアウトはシンプルですが、路線検索の結果を得るには十分であり、HTML内部の構

造を把握するのも容易です。また、fromとtoのパラメータを指定してURLアクセスすればよいため実装も簡素になります(もちろん中身のHTMLを読み解く必要はありますが)。さらに、日時・時刻指定をしたい場合には以下のパラメータをセットします。

- ・ y ... 西暦年
- ・ m ... 月(2桁)
- ・ d ... 日(2桁)
- ・ h ... 時間(2桁)
- ・ m1 ... 分(十の位/上1桁)
- ・ m2 ... 分(一の位/下1桁)

Yahoo!路線検索での路線情報の二次利用(スクレイピング等を使ってサービスとして外部提供等)はできませんので、ご注意ください。このサンプルでは個人の情報取得用途として使うことを想定しています。

4.16.0.2 サンプルコード

リスト4.16: 応用サンプル(16)

```
1: # スクリプトの引数定義
2: Param(
3:     [string]$from,
4:     [string]$to,
5:     [string]$date,
6:     [string]$time,
7:     [int]$h=-1,
8:     [int]$m=-1
9: )
10:
11: # 入力パラメータのチェック
12: if ($from -eq "") {
13:     Write-Error "'from'パラメータを入力してください"
14:     exit
15: } elseif ($to -eq "") {
16:     Write-Error "'to'パラメータを入力してください"
17:     exit=0
18: }
19:
20: # URLの指定
21: $url = "https://transit.yahoo.co.jp/search/print?"
22:
23: # 出発・到着先のパラメータ付加
24: $url += "from=${from}&to=${to}"
25:
```

```

26: # 日時のパラメータ付加
27: $dt = Get-Date
28: if ($date -ne "") {
29:     try {
30:         $dt = [DateTime]$date
31:     } catch {
32:         Write-Error "日付の入力フォーマットに誤りがあります"
33:         exit
34:     }
35: }
36: $url += "&y=${$dt.Year}"
37: $url += "&m=${$dt.Month.ToString('00')}"
38: $url += "&d=${$dt.Day.ToString('00')}"
39:
40: # 時刻のパラメータ付加(時間)
41: if ($time -ne "") {
42:     $timeArr = $time.Split(":")
43:     $h = [int]$timeArr[0]
44:     $m = [int]$timeArr[1]
45:
46:     if (($h -ge 0) -and ($h -lt 24)) {
47:         $url += "&hh=${h}"
48:     } else {
49:         Write-Error "時刻のフォーマットに誤りがあります"
50:         exit
51:     }
52:
53:     # 時刻のパラメータ付加(分)
54:     if (($m -ge 0) -and ($m -lt 60)) {
55:         $m1 = $m / 10
56:         $m2 = $m % 10
57:         $url += "&m1=${m1}&m2=${m2}"
58:     } else {
59:         Write-Error "時刻のフォーマットに誤りがあります"
60:     }
61: }
62:
63: # リクエスト処理の実行
64: $req = Invoke-WebRequest -Uri $url
65: if ($req.StatusCode -ne 200) {
66:     Write-Error "Yahoo!ページにアクセスできません"

```

```

67:     exit
68: }
69:
70: # HTMLテキストの取得
71: $html = $req.Content
72:
73: # 経路検索エラー判定
74: if ($html -match "経路検索ができませんでした") {
75:     Write-Host "経路検索エラーが発生しています。"
76:     Write-Host "条件を変更して再度検索してください"
77:     exit
78: }
79:
80: # 正規表現を使った文字列の抜き出し
81: # Note: IE廃止以後はParsedHtmlが利用できないため生のHTML文字列から直接文字列を取り出す
82: # 経路の概要部分のテキスト取得
83: $match1 = [regex]::Matches($html,
84:     '<div class="routeSummary"><ul class="summary">(.*?)</ul>')
85: $routeSummaryText = $match1.Groups[1].Value
86:
87: # 時刻情報
88: $match1_1 = [regex]::Matches($routeSummaryText,
89:     '<li class="time">(.*?)</li>')
90: $timeString = ($match1_1.Groups[1].Value -replace '<.*?>', '')
91:
92: # 運賃情報の取得
93: $match1_2 = [regex]::Matches($routeSummaryText,
94:     '<li class="fare">(.*?)</li>')
95: $fareString = ($match1_2.Groups[1].Value -replace '<.*?>', '')
96:
97: # 経路の詳細部分のテキスト取得
98: $match2 = [regex]::Matches($html,
99:     '<div class="routeDetail">(.*?)</div>')
100: $routeDetailText = $match2.Groups[1].Value
101:
102: # 駅情報の取得
103: $stationInfoArray = @()
104: $match2_1 = [regex]::Matches($routeDetailText,
105:     '<div class="station">.*?<dl><dt>(.*?)</dt></dl>.*?</div>')
106: $match2_1 | ForEach-Object {
107:     $stationInfoArray += $_.Groups[1].Value }

```

```

108:
109: # 経路情報の取得
110: $fareInfoArray = @()
111: $match2_2 = [regex]::Matches($routeDetailText,
112:     '<div class="access"><ul class="info">(.*?)</ul></div>')
113: $match2_2 | ForEach-Object {
114:     $tempString = $_.Groups[1].Value.Trim()
115:     $tempString = ($tempString -replace '\[line\]', '')
116:     $tempString = ($tempString -replace '<.+?>', '')
117:     $fareInfoArray += $tempString
118: }
119:
120: # 路線検索結果の出力
121: Write-Host $timeString
122: Write-Host $FareString
123: foreach($i in 0..($stationInfoArray.Length-2)) {
124:     Write-Host "[ $($stationInfoArray[$i]) ]"
125:     Write-Host " ↓ $($fareInfoArray[$i])"
126: }
127: Write-Host "[ $($stationInfoArray[-1]) ]"

```

4.16.0.3 実行結果

```

# 経路検索例(1) ... 東京→横浜
PS > .\sample_advanced16.ps1 -from 東京 -to 横浜
14:27 発→14:51 着 24 分 (乗車 24 分)
IC 優先: 483 円
[ 東京 ]
↓ J R 東海道本線熱海行 [ 発 ] 10 番線 → [ 着 ] 6 番線 4 駅
[ 横浜 ]

# 経路検索例(2) ... 大洗(茨城)→新宿
PS > .\sample_advanced16.ps1 -from 大洗 -to 新宿
15:05 発→17:56 着 2 時間 51 分 (乗車 2 時間 37 分)
IC 優先: 2,640 円
[ 大洗 ]
↓ 鹿島臨海鉄道大洗鹿島線 (当駅始発) 水戸行 [ 発 ] 情報なし → [ 着 ] 8 番線 3 駅
[ 水戸 ]
↓ J R 常磐線上野行 [ 発 ] 5 番線 → [ 着 ] 3 番線 23 駅
[ 日暮里 ]
↓ J R 山手線内回り池袋・新宿方面 [ 発 ] 11 番線 → [ 着 ] 14 番線 10 駅
[ 新宿 ]

# 経路検索例(3) ... 八王子→秋葉原(日時・時刻指定あり)
PS > .\sample_advanced16.ps1 -from 八王子 -to 秋葉原 -date 2024/4/1 -time 11:30

```

```
11:34 発→12:26 着 52 分 (乗車 49 分)
IC 優先 : 824 円
[ 八王子 ]
↓ J R 中央線中央特快東京行 [ 発 ] 2 番線 → [ 着 ] 4 番線 9 駅
[ 御茶ノ水 ]
↓ J R 総武線千葉行 [ 発 ] 3 番線 → [ 着 ] 6 番線
[ 秋葉原 ]
```

4.17 応用サンプル 17: 英単語検索・辞書データ作成ツール(CUI/SQLite3 使用)

4.17.0.1 下準備

このサンプルの実行には.NET 用の"SQLite"のライブラリのインストールが必要です。

```
PS > dotnet add package System.Data.SQLite.Core
```

4.17.0.2 解説

オンライン和英辞書の「英辞郎 on the Web」より英単語データを取得します。

・英辞郎 on the Web

—URL:<https://eow.alc.co.jp/>

以下のようにqパラメータに検索したい英単語を入力して検索ページを表示します。

```
https://eow.alc.co.jp/search?q=\(検索対象の英単語\)
```

前に上げたサンプルと同様にページのHTML構造からうまく英単語情報を抽出します。ただし、このサンプルでは単に英単語の意味を取得して表示するだけではなく、SQLiteを使って簡易データベースへの単語情報の登録も行うこととします。

SQLite ファイル内の単語情報テーブルの中身を検索し、登録済みでない単語の情報は英辞郎のページから取得の上SQLiteのデータベース上に登録します。登録済みである場合にはテーブルの中身のテキストデータを取り出して表示します。このようなアプローチにより、不要なWeb リクエスト回数を減らしています。

4.17.0.3 サンプルコード

リスト 4.17: 応用サンプル (17)

```
1: # スクリプトの引数定義
2: Param(
3:     [string]$word = ""
4: )
5:
```

```

6: # 関数定義
7: # SELECT文用
8: function Get-SQLCommand($conn, $sql) {
9:     $sqlcmd = $conn.CreateCommand()
10:    $sqlcmd.CommandText = $sql
11:    return $sqlcmd
12: }
13:
14: # INSERT/DELETE/UPDATE文用
15: # Note: ExecuteNonQuery実行時の変更行数表示をOut-Nullで捨てる
16: function Get-SQLExecuteNonQuery($conn, $sql, $returnCount=$false) {
17:     $sqlcmd = $conn.CreateCommand()
18:     $sqlcmd.CommandText = $sql
19:     if ($returnCount) {
20:         return $sqlcmd.ExecuteNonQuery()
21:     } else {
22:         $sqlcmd.ExecuteNonQuery() | Out-Null
23:     }
24: }
25:
26: # 単語の意味の取得
27: function Get-NewWordMeaning($word) {
28:     # 英辞郎からのスクレイピング処理
29:     $url = "https://eow.alc.co.jp/search?q=${word}"
30:
31:     # リクエスト処理の実行
32:     $req = Invoke-WebRequest -Uri $url
33:     if ($req.StatusCode -ne 200) {
34:         Write-Error "指定のWebページにアクセスできません"
35:         exit
36:     }
37:
38:     # HTML取得
39:     $html = $req.Content
40:
41:     # 意味情報の抜き出し
42:     $meaningText = ""
43:     $match1 = [regex]::Matches($html,
44:         '<span class="wordclass">(.*?)</span><ol>(.*?)</ol>')
45:     foreach($m1 in $match1) {
46:         $wordClass = $m1.Groups[1].Value

```



```

47:         $meaningSection = $m1.Groups[2].Value
48:
49:         # liタグ以下から情報の取得
50:         $meaningText += "${wordClass}`n"
51:         $match2 = [regex]::Matches(
52:             $meaningSection, '<li>(.*?)</li>')
53:
54:         # liタグが存在するかどうかで分岐
55:         if ($match2) {
56:             $count = 1
57:             foreach($m2 in $match2) {
58:                 $tempStr = $m2.Groups[1].Value
59:                 $tempStr = $tempStr -replace '<span.+?>.+?</span>', ''
60:                 $tempStr = $tempStr -replace '<br />.+$', ''
61:                 $meaningText += "${count}. ${tempStr}`n"
62:                 $count++
63:             }
64:         } else {
65:             $tempStr = $meaningSection
66:             $tempStr = $tempStr -replace '<span.+?>.+?</span>', ''
67:             $tempStr = $tempStr -replace '<br />.+$', ''
68:             $meaningText += "${tempStr}`n"
69:         }
70:     }
71:     return $meaningText
72: }
73:
74: # 変数宣言
75: $isNewDatabase = $false
76: $isUpdateHistoryRequired = $false
77: $saveFolderPath = [Environment]::GetFolderPath("MyDocuments")
78: $sqliteFilePath = "${saveFolderPath}\wordbook.sqlite"
79:
80: # 入力パラメータのチェック
81: if ($word -eq "") {
82:     Write-Host "-wordオプションで検索したい英単語を指定してください"
83:     exit
84: }
85:
86: # 入力パスの存在確認
87: if (-not (Test-Path -Path $sqliteFilePath -PathType Leaf)) {

```

```

88:     $isNewDatabase = $true
89: }
90:
91: # SQLiteの接続処理
92: $conn = New-Object -TypeName System.Data.SQLite.SQLiteConnection
93: $conn.ConnectionString = "Data Source = ${sqliteFilePath}"
94: try {
95:     $conn.Open()
96: } catch {
97:     Write-Error "SQLiteファイルのオープンに失敗しました"
98:     exit
99: }
100:
101: # 新規テーブル作成時の処理
102: if ($isNewDatabase) {
103:     # 単語登録用テーブルのSQL
104:     $sqlCreateTable1 = "CREATE TABLE words (
105:         ID INTEGER PRIMARY KEY AUTOINCREMENT,
106:         word TEXT,
107:         meaning TEXT,
108:         created_at DATETIME DEFAULT CURRENT_TIMESTAMP
109:     );"
110:
111:     # 単語検索履歴登録用テーブルのSQL
112:     $sqlCreateTable2 = "CREATE TABLE word_search_histories (
113:         ID INTEGER PRIMARY KEY AUTOINCREMENT,
114:         word TEXT,
115:         created_at DATETIME DEFAULT CURRENT_TIMESTAMP
116:     );"
117:
118:     # テーブル作成用SQLクエリの実行
119:     Get-SQLExecuteNonQuery $conn $sqlCreateTable1
120:     Get-SQLExecuteNonQuery $conn $sqlCreateTable2
121: }
122:
123: # 完全一致検索の準備
124: $searchExactResult = @()
125: $sqlSearchExactWord = "SELECT * FROM words WHERE word='${word}';"
126: $sqlcmd1 = Get-SQLCommand $conn $sqlSearchExactWord
127: $reader1 = $sqlcmd1.ExecuteReader()
128:

```

```

129: # 完全一致検索の実行
130: while ($reader1.HasRows) {
131:     if ($reader1.Read()) {
132:         $tempStr = "[$($reader1["word"])]`n$($reader1["meaning"])"
133:         $searchExactResult += $tempStr
134:     }
135: }
136:
137: # 完全一致検索の結果に応じて次の処理を決定
138: if ($searchExactResult.Length -eq 0) {
139:     # 部分一致検索の準備
140:     $searchLikeResult = @()
141:     $sqlSearchLikeWord = "
142:         SELECT * FROM words WHERE word LIKE '${word}%'";
143:     $sqlcmd2 = Get-SQLCommand $conn $sqlSearchLikeWord
144:     $reader2 = $sqlcmd2.ExecuteReader()
145:
146:     # 部分一致検索の実行
147:     while ($reader2.HasRows) {
148:         if ($reader2.Read()) {
149:             $tempStr = "$($reader2["word"]) - $($reader2["meaning"])"
150:             $searchLikeResult += $tempStr
151:         }
152:     }
153:
154:     # 部分一致検索結果がなかった場合の処理
155:     if ($searchLikeResult.Length -eq 0) {
156:         # 新出単語のスクレイピング処理
157:         $meaning = Get-NewWordMeaning $word
158:
159:         # 単語の登録処理
160:         $sqlInsertNewWord = "
161:             INSERT INTO words(word, meaning)
162:             VALUES ('${word}', '${meaning}');"
163:         Get-SQLExecuteNonQuery $conn $sqlInsertNewWord
164:
165:         # 単語検索履歴登録フラグの設定
166:         $isUpdateHistoryRequired = $true
167:     }
168: } elseif ($searchExactResult.Length -ge 1) {
169:     # 完全一致検索結果の表示

```

```

170:     foreach ($tempStr in $searchExactResult) {
171:         Write-Host $tempStr
172:     }
173:     # 単語検索履歴登録フラグの設定
174:     $isUpdateHistoryRequired = $true
175: }
176:
177: # 単語検索履歴の登録処理
178: if ($isUpdateHistoryRequired) {
179:     $sqlInsertNewHistory = "
180:         INSERT INTO word_search_histories (word)
181:         VALUES ('${word}');"
182:     Get-SQLExecuteNonQuery $conn $sqlInsertNewHistory
183: }

```

4.17.0.4 実行結果

```

PS > .\sample_advanced17_english_wordbook.ps1 -word tribute
[tribute]
【名】
1. 〔感謝や尊敬のしるしの〕 贈り物、記念品
2. 〔感謝や尊敬を表す〕 言葉、賛辞
3. 〔称賛すべきものの〕 あかし、証拠
4. 〔他国の君主への〕 貢ぎ物（税）◆ある君主が別の君主に服従を示すために贈られる品物や金、またはそれを調達するために課する税金。
5. 〔中世の家臣の君主への〕 税金（の義務）
【映画】
マイ・ハート／マイ・ラブ◆米1980年
【雑誌名】
ア・ロイヤル・トリビュート

```

4.18 応用サンプル18: モールス信号

4.18.0.1 解説

- ・モールス符号 (Wikipedia)

—URL:<https://ja.wikipedia.org/wiki/%E3%83%A2%E3%83%BC%E3%83%AB%E3%82%B9%E7%AC%A6%E5%8F%B7>

有名なモールス信号を出力するサンプルです。モールス信号はかつて電信のための主な手段として用いられ、陸上の通信で、20世紀前半までは電報等の文字通信で多く使われました。その後も利用は続きましたが、IT技術が発達して電信の手段が増えたことにより、近年では一部の用途を除いて使用されることはほとんどなくなっています。

このサンプルでは英数字・一部記号をモールス信号の文字列として表現し、モールス信号の音声

を再生します。モールス信号では出現頻度に従って、文字ごとにあらかじめ符号が割り当てられていますので、ハッシュ・辞書データを事前に準備しておけば、対応を取るのは容易です。

図: モールス信号(符号)表¹

A	.-	N	-. .	1	. ---	?	. . - - . .
B	- . . .	O	---	2	. - - -	!	- . - . - -
C	- . - .	P	. - - .	3	. . - -	.	. - . - . -
D	- . .	Q	- - - .	4	. . . -	,	- - . . - -
E	.	R	. - .	5	;	- . - - . .
F	. . - .	S	. . .	6	-	:	- - - . . .
G	- - - .	T	-	7	- - . . .	+	. - . - .
H	U	. . -	8	- - - . .	-	- . . . -
I	. .	V	. . . -	9	- - - . .	/	- . - . -
J	. - - - -	W	. - - -	0	- - - - -	=	- . . - -
K	- . -	X	- . . -				
L	. - . .	Y	- . - - -				
M	- - -	Z	- - . .				

4.18.0.2 サンプルコード

リスト4.18: 応用サンプル(18)

```

1: # ビープ音の設定
2: # Note: ミリ秒表記
3: $beepUnitHertz = 800
4: $beepUnitTime = 200
5:
6: # モールス信号の変換用連想配列
7: $morseCodeDict = @{
8:     # アルファベット
9:     "A" = "-. "; "B" = "-... "; "C" = "-.-. "; "D" = "-.. ";
10:    "E" = ". "; "F" = ".-. "; "G" = "--. "; "H" = ".... ";
11:    "I" = ".. "; "J" = ".--- "; "K" = "-.- "; "L" = "-.. ";
12:    "M" = "-- "; "N" = "-. "; "O" = "--- "; "P" = ".-.- ";
13:    "Q" = "--.- "; "R" = ".-. "; "S" = "... "; "T" = "- ";
14:    "U" = "..- "; "V" = "...- "; "W" = "-.- "; "X" = "-.-.- ";
15:    "Y" = "-.-.- "; "Z" = "--.. ";
16:    # 数字
17:    "1" = ".---- "; "2" = "..--- "; "3" = "...-- ";
18:    "4" = "....- "; "5" = "..... "; "6" = "-.... ";

```

```

19:     "7" = "--..."; "8" = "---.."; "9" = "----."; "0" = "-----"
20:     # 記号
21:     "." = "-.-.-"; "," = "--.-.-"; ":" = "---..."; "?" = ".-.-.-";
22:     "!" = ".-----"; "-" = "-....-"; "(" = "-.-.-"; ")" = "-.-.-.-";
23:     "/" = "-.-.-"; "=" = "-...-"; "+" = "-.-.-"; "'" = "-.-.-.-";
24:     '@' = "-.-.-.-";
25:     # スペース
26:     " " = "~"
27: }
28:
29: # 変換対象の文字列の入力
30: $rawInputString = Read-Host "input string"
31: $inputString = $rawInputString.ToUpper()
32:
33: # 入力文字列の正当性チェック
34: $exitFlag = $false
35: foreach ($char in $inputString.ToCharArray()) {
36:     if (-not $morseCodeDict.Contains([string]$char)) {
37:         Write-Error "不正な文字があります: '${char}'"
38:         $exitFlag = $true
39:     }
40: }
41: if ($exitFlag) { exit }
42:
43: # 入力文字列のモールス信号への変換
44: $morseCodeArray = @()
45: $morseCodeString = ""
46: foreach ($word in $inputString.Split()) {
47:     $tmpCodeArray = $()
48:     foreach ($char in $word.ToCharArray()) {
49:         $code = $morseCodeDict[[string]$char]
50:         $tmpCodeArray += $code
51:         $tmpCodeArray += " "
52:     }
53:     $morseCodeArray += , $tmpCodeArray
54:     $morseCodeString += $tmpCodeArray -join " "
55: }
56:
57: # モールス信号文字列の出力
58: Write-Host $morseCodeString
59:

```

```

60: # ビープ音でのモールス信号出力
61: foreach($tmpCodeArray in $morseCodeArray) {
62:     foreach($codeChunk in $tmpCodeArray) {
63:         foreach ($char in $codeChunk.ToCharArray()) {
64:             $code = [string]$char
65:             if ($code -eq ".") {
66:                 # 短いビープ音を出力("トン"の音)
67:                 [System.Console]::Beep(
68:                     $beepUnitHertz, $beepUnitTime)
69:             } elseif ($code -eq "-") {
70:                 # 長いビープ音を出力("ツー"の音)
71:                 [System.Console]::Beep(
72:                     $beepUnitHertz, $beepUnitTime * 3)
73:             }
74:
75:             # 信号スペース間の無音時間
76:             Start-Sleep -M $beepUnitTime
77:         }
78:     }
79:
80:     # 単語間スペースの無音時間
81:     Start-Sleep -M ($beepUnitTime * 6)
82: }

```

4.18.0.3 実行結果

実行時にはモールス信号相当のビープ音が鳴ります。

```

# モールス信号出力例 (1)
PS > .\sample_advanced18.ps1
文字列を入力してください: I am a cat.
.. .- -- .- -. .- - .-.-

# モールス信号出力例 (2)
PS > .\sample_advanced18.ps1
文字列を入力してください: You got me mad now.
-.- --- .. --. --- - - . -- .- .. .- --- .-.-

```

4.19 応用サンプル 19: ライフゲーム

4.19.0.1 解説

ライフゲーム (Conway's Game of Life[1]) は1970年にイギリスの数学者ジョン・ホートン・コンウェイ (John Horton Conway) が考案した数理モデルである。単純なルールから複雑な結果が生成され、パズルやミニスケープの要素を持っている。生命の誕生、進化、淘汰などのプロセスを連想させるパターンも存在し、シミュレーションゲームと分類される場合がある。

生物集団においては、過疎でも過密でも個体の生存に適さないという個体群生態学的な側面を背景に持つ。セル・オートマトンのもっともよく知られた例でもある。

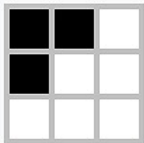
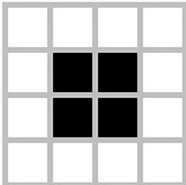
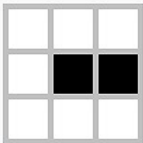
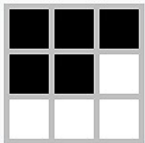
・ ライフゲーム (Wikipedia)

— URL:<https://ja.wikipedia.org/wiki/%E3%83%A9%E3%82%A4%E3%83%95%E3%82%B2%E3%83%BC%E3%83%A0>

ライフゲームはセルオートマトンの事例として代表的なものです。ライフゲームは4つのシンプルなルールから成り立っています。このルールに従って次のセル状態を決定します。一見単純なルールに見えますが、驚くほど多様なパターンが現れてきます。またライフゲームにおいては周期をもつ数多くのパターンがあることも知られています。

- ・ 誕生 ... 死んでいるセルに隣接する生きたセルがちょうど3つあれば、次の世代が誕生する。
- ・ 生存 ... 生きているセルに隣接する生きたセルが2つか3つならば、次の世代でも生存する。
- ・ 過疎 ... 生きているセルに隣接する生きたセルが1つ以下ならば、過疎により死滅する。
- ・ 過密 ... 生きているセルに隣接する生きたセルが4つ以上ならば、過密により死滅する。

ライフゲームの基本ルール

誕生	生存（維持）	死（過疎）	死（過密）
			

ライフゲームにおける物体の例 (Wikipedia より)

図: 固定物体の例

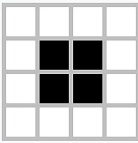
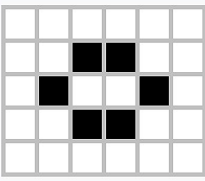
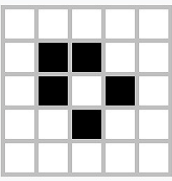
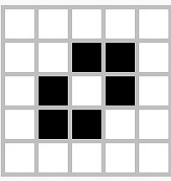
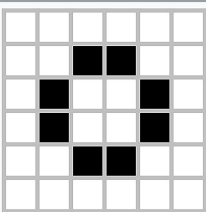
ブロック	蜂の巣	ボート	船	池
				

図: 振動子(周期2)の例

ブリンカー	ヒキガエル	ビーコン	時計
			

図: 振動子(周期3以上)の例

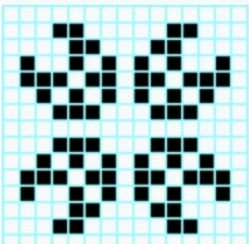
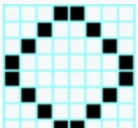
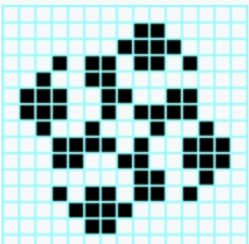
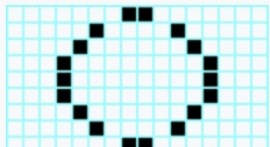
パルサー	八角形	銀河	ペンタデカスロン
			

図: 移動物体の例

グライダー	軽量級宇宙船	中量級宇宙船	重量級宇宙船
			

こうした物体を用いてさらに複雑なパターンや巨大物体を生成するような試みもたびたび行われています。特にグライダー銃の組み合わせで特定の処理を実行させるなど、ある種プログラミングとも呼べるようなしくみを使うこともあります。ライフゲームを使って疑似的に計算機として動かすような場合、チューリング完全であることが知られています。興味がある方はGoogleで検索してみてください。

ライフゲームの紹介・解説動画

- ・ライフゲームの世界1【複雑系】

—URL:<https://www.nicovideo.jp/watch/sm19347846>

- ・「Life Universe」で無限に再帰するライフゲームの世界を漂ってみた

4.19.0.2 サンプルコード

リスト4.19: 応用サンプル(19)

```
1: # スクリプトの引数定義
2: Param(
3:     [string]$inputPath,
4:     [switch]$random,
5:     [int]$width=20,
6:     [int]$height=20,
7:     [int]$limit=100
8: )
9:
10: # ランダムな盤面データの作成
11: function Init-BoardRandom($wsize, $hsize) {
12:     $board = [int[] []]::New($hsize, $wsize)
13:
14:     # Note: 3分の1程度が生きたセルとして生成されるようにする
15:     foreach ($i in 0..($hsize-1)) {
16:         foreach ($j in 0..($wsize-1)) {
17:             $value = (Get-Random -Minimum 1 -Maximum 100)
18:             if (($value % 3) -eq 0) {
19:                 $board[$i][$j] = 1
20:             } else {
21:                 $board[$i][$j] = 0
22:             }
23:         }
24:     }
25:
26:     return $board
27: }
28:
29: # 入力ファイルからの盤面データの作成
30: function Init-BoardFromFile($wsize, $hsize, $inputPath) {
31:     $board = [int[] []]::New($hsize, $wsize)
32:     $i, $j = 0, 0
33:
34:     # Note: 生存しているセルは"o"として表記されているものとする
35:     foreach ($line in (Get-Content $inputPath)) {
36:         $charArray = $line.ToCharArray()
37:         $j = 0
38:         foreach ($c in $charArray) {
```

```

39:         if ($c -eq "o") {
40:             $board[$i][$j] = 1
41:         } else {
42:             $board[$i][$j] = 0
43:         }
44:         $j++
45:     }
46:     $i++
47: }
48:
49: return $board
50: }
51:
52: # 次の盤面データの取得
53: function Get-NextBoard($board, $wsize, $hsize) {
54:     $nextBoard = @()
55:
56:     # 次のセル状態についての配列データを作成する
57:     foreach ($i in 0..($hsize-1)) {
58:         $row = @()
59:
60:         foreach ($j in 0..($wsize-1)) {
61:             # セルの周囲マスのインデックス範囲を計算
62:             # Note: セルが盤面端にある場合に不正なインデックス指定を防ぐ
63:             $iStartIndex, $iEndIndex = `
64:                 [math]::max(0, $i-1), [math]::min($hsize-1, $i+1)
65:             $jStartIndex, $jEndIndex = `
66:                 [math]::max(0, $j-1), [math]::min($wsize-1, $j+1)
67:
68:             # セルの周囲8マスの生存しているセルの数を計算する
69:             $aliveCellCount = 0
70:             foreach($x in $iStartIndex..$iEndIndex) {
71:                 foreach($y in $jStartIndex..$jEndIndex) {
72:                     if (($x -ne $i) -or ($y -ne $j)) {
73:                         $aliveCellCount += $board[$x][$y]
74:                     }
75:                 }
76:             }
77:
78:             # 次の各セルの状態を決定
79:             $status = $board[$i][$j]

```

```

80:         if ($status -eq 0) {
81:             switch ($aliveCellCount) {
82:                 3 { $row += 1 }
83:                 default { $row += 0 }
84:             }
85:         } else {
86:             switch ($aliveCellCount) {
87:                 2 { $row += 1 }
88:                 3 { $row += 1 }
89:                 default { $row += 0 }
90:             }
91:         }
92:     }
93:
94:     $nextBoard += ,$row
95: }
96:
97: return $nextBoard
98: }
99:
100: # 盤面の状態を出力する
101: # Note: 生存しているセルは"o"、そうでなければ "-"で表示する
102: function Show-BoardStatus($turn, $board) {
103:     # 画面上へカーソルを戻すANSIエスケープシーケンス追加
104:     # Note: Clear-Hostを使わずに画面描画するための処置
105:     $printStr = "$([char]0x1B)[0d"
106:
107:     # 経過ターンの表示
108:     $printStr += "Turn: ${turn}`n"
109:
110:     # 盤面状態の表示
111:     foreach ($row in $board) {
112:         foreach ($cell in $row) {
113:             if ($cell -eq 1) {
114:                 $printStr += "o"
115:             } else {
116:                 $printStr += "-"
117:             }
118:         }
119:         $printStr += "`n"
120:     }

```

```

121:
122:     Write-Host $printStr
123: }
124:
125: # 盤面のサイズの確認
126: # Note: ここでは盤面の幅は5以上100以下であるものとする
127: $isValidBoard = $true
128: if (($width -lt 5) -or ($width -gt 100)) {
129:     Write-Error "盤面の幅は5～100の範囲で入力してください"
130:     $isValidBoard = $false
131: }
132: if (($height -lt 5) -or ($height -gt 100)) {
133:     Write-Error "盤面の高さは5～100の範囲で入力してください"
134:     $isValidBoard = $false
135: }
136: if (-not $isValidBoard) { return }
137:
138: # 盤面データの初期化
139: $board = @()
140: if ($random) {
141:     # ランダムに盤面データを初期化
142:     $board = Init-BoardRandom $width $height
143: } else {
144:     # 入力ファイルのパスの確認
145:     if ($inputPath -eq "" -or (-not (Test-Path $inputPath))) {
146:         Write-Error "入力ファイルが見つかりません"
147:         exit
148:     }
149:
150:     # 入力ファイルにもとづいて盤面データを初期化
151:     $board = Init-BoardFromInputFile $width $height $inputPath
152: }
153:
154: # ライフゲームの実行
155: Clear-Host
156: foreach($turn in 0..$limit) {
157:     Show-BoardStatus $turn $board
158:     Start-Sleep 1
159:     $board = Get-NextBoard $board $width $height
160: }
161:

```

4.19.0.3 実行結果

初期値ランダムでの実行時

初期値をランダムで決定する際の実行結果です。入力がランダムで与えられるため、出現パターンによっては一定時間が経過した後に固定物体を残して動きが完全になくなるような場合もあります。

```
PS > .\sample_advanced19.ps1 -width 10 -height 10 -random
Turn: 3
-o-ooo----
o-----
-o---o--o-
--oo-o-o-o
--o-----oo
-ooo-----
o-ooo-----
oo--oo-----
-ooo-o-----
--ooo-----
```

※実行ごとに結果は異なります。

初期値ファイルを指定して実行開始した場合

次は特定の周期を持ったパターンを初期値として与えるときの実行結果です。

サンプル 1: 八角形 (8x8 マス; 周期 3)

```
# 入力ファイルの内容
PS > cat sample01_octagon.txt
---oo---
--o--o--
-o----o-
o-----o
o-----o
-o----o-
--o--o--
---oo---

# 実行結果
PS > .\sample_advanced19.ps1 -width 8 -height 8 -inputPath
.\sample01_octagon.txt
Turn: 3
-----
--o--o--
-o-oo-o-
--o--o--
--o--o--
-o-oo-o-
--o--o--
-----
```

(※時間経過ごとに画面が再描画されます)

サンプル2: 銀河(15x15マス; 周期8)

```
# 入力ファイルの内容
PS > cat sample02_galaxy.txt
-----
-----
-----
---00-000000---
---00-000000---
---00-----
---00----00---
---00----00---
---00----00---
-----00---
---000000-00---
---000000-00---
-----
-----
-----

# 実行結果
PS > .\sample_advanced19.ps1 -width 15 -height 15 -inputPath
.\sample02_galaxy.txt
.\sample02_galaxy.txt
Turn: 3
-----
-----0-0-----
-----0-----
---00-----
-0-----00-0---
---0---0-0---
---000-0---0---
-0-----0---
--0---0-000---
--0-0---0---
---0-00-----0-
-----00-----
-----0-----
----0-0-----
-----

(※時間経過ごとに画面が再描画されます)
```

サンプル3: 銀河(15x15マス; 周期8)

```
# 入力ファイルの内容
PS > cat sample03_pentadecathlon.txt
-----
-----
-----
-----o-----o-----
---oo-oooo-oo---
-----o-----o-----
-----
-----
-----

# 実行結果
PS > .\sample_advanced19.ps1 -width 16 -height 9 -inputPath
.\sample03_pentadecathlon.txt
Turn: 3
-----
-----oooo-----
-----oooooooo-----
-----oooooooooooo-----
---oo-----oo---
---oooooooooooo-----
-----oooooooo-----
-----oooo-----
-----
(※時間経過ごとに画面が再描画されます)
```

4.20 応用サンプル20: テトリス

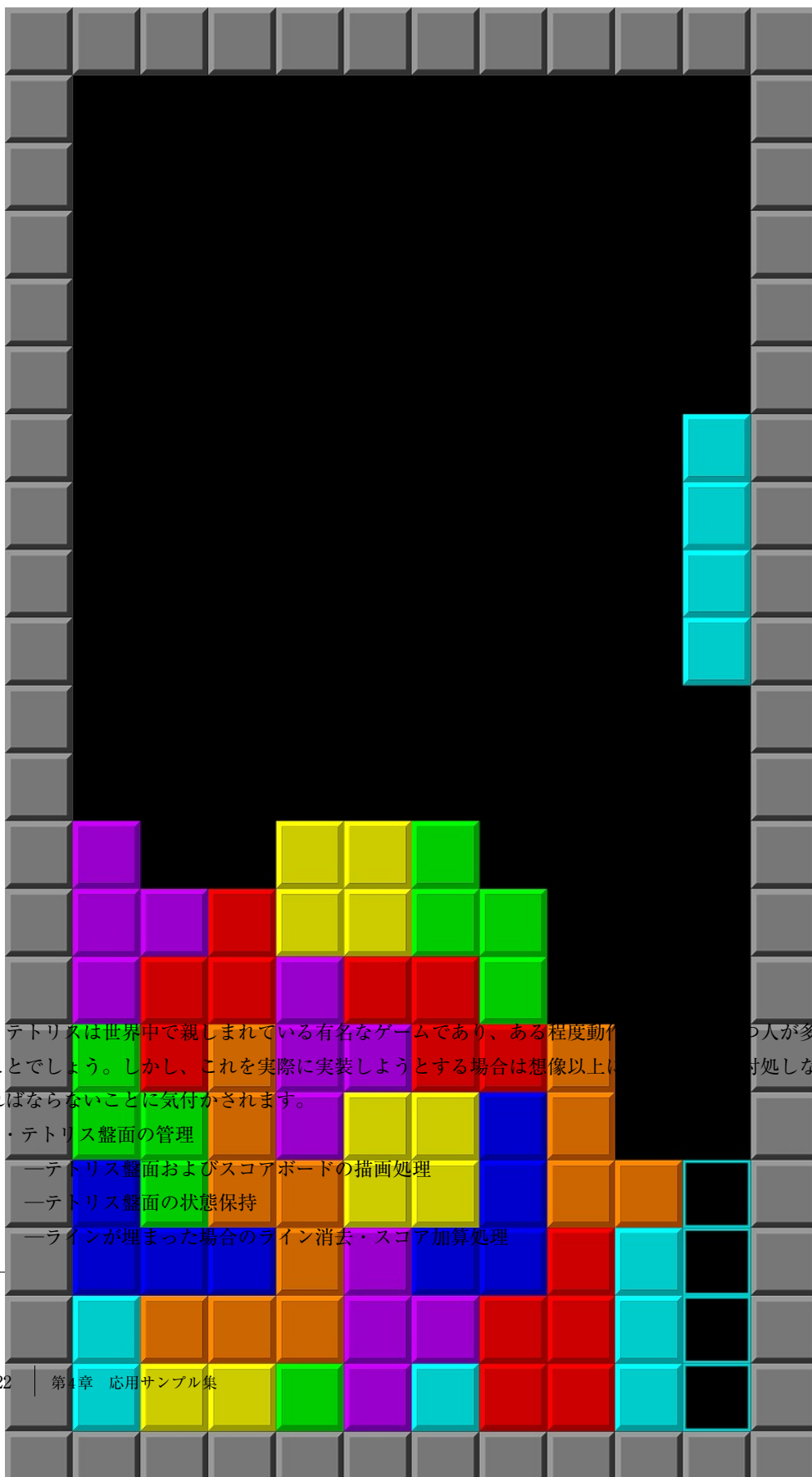
4.20.01 下準備

このサンプルの実行前には、以下の通りアセンブリを読み込ませておく必要があります。これらのアセンブリはスクリプト実行前の段階で読み込まれている必要があります。PowerShellのクラス定義においては事前に指定されていない型を使えないため、このような事前準備が必要となります。

```
PS > Add-Type -AssemblyName PresentationCore
PS > Add-Type -AssemblyName WindowsBase
```

4.20.02 解説

図: テトロリスの画像 (Wikipedia より)



テトロリスは世界中で親しまれている有名なゲームであり、ある程度動作している人が多いことでしょう。しかし、これを実際に実装しようとする場合は想像以上に多くの処理をしなければならぬことに気がかされます。

・テトロリス盤面の管理

- テトロリス盤面およびスコアボードの描画処理
- テトロリス盤面の状態保持
- ラインが埋まった場合のライン消去・スコア加算処理

- ゲームオーバー判定
- ・テトリミノの管理
 - テトリミノごとの回転中心および回転方向の定義
 - テトリミノ7種類のランダム出現処理
 - 現在操作中のテトリミノの状態・座標の保持
 - プレイヤーのキー入力の受け付け
 - キー入力に応じたテトリミノの移動・回転処理
 - テトリミノの左右上下の衝突判定
 - テトリミノの回転可能判定
- ・時間経過による処理
 - 一定時間が経過した後のテトリミノの降下処理
 - 座標不変状態によるテトリミノの固定化処理

これらのことを実現しながら実際にゲームとして遊べる状態にまで持っていく必要があります。このサンプルではテトリスとして最低限必要な機能を備えとともに、画面描画にはアニメーションのための特殊なライブラリ等は使わずシェル表示のみで対応するものとします。

4.20.0.3 サンプルコード

- ・操作方法
 - テトリミノを左に移動 ... Aキー もしくは 左方向キー(←)
 - テトリミノを右に移動 ... Dキー もしくは 右方向キー(→)
 - テトリミノを下に移動 ... Sキー もしくは 下方向キー(↓)
 - テトリミノを回転 ... Spaceキー

リスト4.20: 応用サンプル(20)

```

1: # アセンブリの読み込み
2: Add-Type -AssemblyName PresentationCore
3: Add-Type -AssemblyName WindowsBase
4:
5: # テトリスブロック(テトリミノ)の情報を格納するクラス
6: class Block {
7:     [int] $IPos
8:     [int] $JPos
9:     [string] $Name
10:    [int[]] $RotateCenter
11:    [int[][]] $BlockPosArray
12:
13:    Block($iPos, $jPos, $name, $rotateCenter, $blockPosArray) {
14:        $this.IPos = $ipos
15:        $this.JPos = $jpos
16:        $this.Name = $name

```

```

17:         $this.RotateCenter = $rotateCenter
18:         $this.BlockPosArray = $blockPosArray
19:     }
20: }
21:
22: # テトリス盤面・操作に関するクラス
23: class Tetris {
24:     # 初期設定
25:     [int] $BoardWidth = 10           # 盤面の幅
26:     [int] $BoardHeight = 15          # 盤面の高さ
27:     [int] $InitBlockIPos = 0         # テトリミノの初期位置(縦方向)
28:     [int] $InitBlockJPos = 4         # テトリミノの初期位置(横方向)
29:     [int] $StayMilliSecondInit = 600 # テトリミノ固定化までの時間(ms)
30:     [int] $ShiftMilliSecondSpan = 100 # 移動可能な最小間隔(ms)
31:     [int] $RotateMilliSecondSpan = 100 # 回転可能な最小間隔(ms)
32:     [int] $UnitMilliSecond = 50      # 処理受付の単位時間(ms)
33:
34:     # 出現させるテトリミノの定義
35:     [PSCustomObject] $NewBlockPatternArray = @(
36:         @{ "pos" = @( @( -1, -1), @( 0, -1), @( 0, 0), @( 0, 1));
37:           "center" = @( 0, 0); "name" = "L-Left"; },
38:         @{ "pos" = @( @( 0, -1), @( 0, 0), @( 0, 1), @( -1, 1));
39:           "center" = @( 0, 0); "name" = "L-Right"; },
40:         @{ "pos" = @( @( -1, -1), @( -1, 0), @( 0, 0), @( 0, 1));
41:           "center" = @( 0, 0); "name" = "S-Left" },
42:         @{ "pos" = @( @( 0, -1), @( 0, 0), @( -1, 0), @( -1, 1));
43:           "center" = @( 0, 0); "name" = "S-Right" },
44:         @{ "pos" = @( @( 0, -1), @( -1, 0), @( 0, 0), @( 0, 1));
45:           "center" = @( 0, 0); "name" = "T" },
46:         @{ "pos" = @( @( 0, 0), @( 0, 1), @( 1, 1), @( 1, 0));
47:           "center" = @( 0.5, 0.5); "name" = "O" },
48:         @{ "pos" = @( @( 0, -1), @( 0, 0), @( 0, 1), @( 0, 2));
49:           "center" = @( 0.5, 0.5); "name" = "I" }
50:     )
51:
52:     # フラグ管理用変数
53:     [bool] $GameOver = $false
54:     [bool] $BlockActive = $false
55:
56:     # カウント用変数
57:     [int] $Level = 1

```

```

58:     [int] $Turn = 0
59:     [int] $Score = 0
60:     [int] $StayCount = 0
61:     [int] $StayMilliSecondLimit = 600
62:     [int] $LastShiftMilliSecond = 0
63:     [int] $LastRotateMilliSecond = 0
64:
65:     # 盤面情報保存用
66:     [int[][]] $BoardArray
67:     [string] $BoardStatusString
68:
69:     # テトリミノ情報保存用
70:     [Block] $CurrentBlock
71:     [Block] $NextBlock
72:     [Block[]] $ReservedBlocks
73:
74:     # コンストラクタ
75:     Tetris() {
76:         # 盤面情報・テトリミノの初期化
77:         $this.BoardArray = [int[][]]::new($this.BoardHeight,
$this.BoardWidth)
78:         $this.ReservedBlocks = @()
79:         $this.NextBlock = $this.GetNewBlock()
80:     }
81:
82:     # 配列ディープコピー作成用メソッド
83:     [System.Collections.ArrayList] DeepCopyArray($array) {
84:         return [Management.Automation.PSSerializer]::Deserialize(
85:             [Management.Automation.PSSerializer]::Serialize($array))
86:     }
87:
88:     # 次の操作対象テトリミノをランダムで決定・取得
89:     [Block] GetNewBlock() {
90:         if ($this.ReservedBlocks.Length -le 1) {
91:             # テトリミノ7種が一巡するようにリザーブに追加
92:             foreach ($i in (0..6 | Get-Random -Count 7)) {
93:                 $blockInfo = $this.NewBlockPatternArray[$i]
94:                 $block = [Block]::new(
95:                     $this.InitBlockIPos, $this.InitBlockJPos,
96:                     $blockInfo["name"], $blockInfo["center"],
97:                     $blockInfo["pos"])

```

```

98:             $this.ReservedBlocks += $block
99:         }
100:    }
101:
102:    # 次のテトリミノ情報を取得
103:    $newBlock = $this.ReservedBlocks[0]
104:    $this.ReservedBlocks = `
105:        $this.ReservedBlocks[1..($this.ReservedBlocks.Length-1)]
106:
107:    return $newBlock
108: }
109:
110: # 操作対象・ネクストとなるテトリミノの入れ替え処理
111: [void] SwitchNextBlock() {
112:     # set next block
113:     $this.CurrentBlock = $this.NextBlock
114:     # get new block
115:     $this.NextBlock = $this.GetNewBlock()
116: }
117:
118: # 盤面の描画
119: [void] DrawTetrisBoard() {
120:     # 盤面およびスコアボードの描画
121:     Write-Host $this.GetCurrentBoardString()
122:
123:     # デバッグ用表示(必要に応じてコメントアウト解除)
124:     #Write-Host $this.GetCurrentStatusString()
125: }
126:
127: # 盤面情報の文字列データ化
128: [string] GetCurrentStatusString() {
129:     $outputStr = ""
130:     $outputStr += "Turn: $($this.Turn)`n"
131:     $outputStr += "iPos:{0}, jPos:{1}`n" `
132:         -f $this.CurrentBlock.IPos, $this.CurrentBlock.JPos
133:     $outputStr += $this.BoardStatusString
134:     return $outputStr
135: }
136:
137: # 現在の盤面情報の文字列データ取得
138: [string] GetCurrentBoardString() {

```

```

139:         $boardLineArray = @()
140:
141:         # 現在の盤面のディープコピーを作成する
142:         $tempBoardArray = $this.DeepCopyArray($this.BoardArray)
143:
144:         # 現在操作中のテトリミノを盤面の配列情報に一旦反映する
145:         if ($this.BlockActive) {
146:             foreach($pos in $this.CurrentBlock.BlockPosArray) {
147:                 $i = $this.CurrentBlock.IPos + $pos[0]
148:                 $j = $this.CurrentBlock.JPos + $pos[1]
149:                 if ($i -ge 0 -and $j -ge 0) { $tempBoardArray[$i][$j] = 1 }
150:             }
151:         }
152:
153:         # 盤面情報の配列から文字列を作成
154:         foreach($row in $tempBoardArray) {
155:             $line = "|"
156:
157:             foreach($cell in $row) {
158:                 if ($cell -eq 0) {
159:                     $line += " "
160:                 } else {
161:                     $line += "#"
162:                 }
163:             }
164:             $line += "|"
165:             $boardLineArray += $line
166:         }
167:         $line = "=" * ($this.BoardWidth + 2)
168:         $boardLineArray += $line
169:
170:         # ネクストのテトリミノを表示
171:         # Note: 1行だけANSIエスケープシーケンスで行削除処理
172:         $boardLineArray[1] += " [Next]"
173:         $boardLineArray[2] += " ${[char]0x1B}[K{0}" -f
174:         $this.NextBlock.Name
175:
176:         # 現在のスコアとレベルを表示
177:         $boardLineArray[4] += " [Score]"
178:         $boardLineArray[5] += " {0:0000}" -f $this.Score
179:         $boardLineArray[7] += " [Level]"

```

```

179:         $boardLineArray[8] += "    {0}" -f $this.Level
180:
181:         # 画面上へカーソルを戻すANSIエスケープシーケンス追加
182:         # Note: Clear-Hostを使わずに画面描画するための処置
183:         $boardStr = "$([char]0x1B)[0d"
184:
185:         # 文字列の結合
186:         $boardStr += ($boardLineArray -join "`n")
187:         return $boardStr
188:     }
189:
190:     # プレイヤー操作局面の進行
191:     [bool] GoPlayerActionPhase() {
192:         $stayMilliSecond = 0
193:         $prevIPos = $this.CurrentBlock.IPos
194:
195:         # 操作のキーコードの定義
196:         $keyLeft = [System.Windows.Input.Key]::Left
197:         $keyRight = [System.Windows.Input.Key]::Right
198:         $keyDown = [System.Windows.Input.Key]::Down
199:         $keySpace = [System.Windows.Input.Key]::Space
200:         $keyA = [System.Windows.Input.Key]::A
201:         $keyD = [System.Windows.Input.Key]::D
202:         $keyS = [System.Windows.Input.Key]::S
203:
204:         # プレイヤー入力の受け付け
205:         while(($stayMilliSecond -lt $this.StayMilliSecondLimit)) {
206:             # キー入力に応じてテトリミノの移動処理
207:             if ([System.Windows.Input.Keyboard]::IsKeyDown($keyLeft) `
208:                 -or [System.Windows.Input.Keyboard]::IsKeyDown($keyA)) {
209:                 if ($this.LastShiftMilliSecond -ge $this.RotateMilliSecondSpan)
210:                 {
211:                     $this.MoveBlockToLeft()
212:                     $this.LastShiftMilliSecond = 0
213:                 }
214:             } elseif ([System.Windows.Input.Keyboard]::IsKeyDown($keyRight)
215:                 -or [System.Windows.Input.Keyboard]::IsKeyDown($keyD)) {
216:                 if ($this.LastShiftMilliSecond -ge $this.RotateMilliSecondSpan)
217:                 {
218:                     $this.MoveBlockToRight()

```



```

217:             $this.LastShiftMilliSecond = 0
218:         }
219:     } elseif ([System.Windows.Input.Keyboard]::IsKeyDown($keyDown) `
220:         -or [System.Windows.Input.Keyboard]::IsKeyDown($keyS)) {
221:         if ($this.LastShiftMilliSecond -ge $this.RotateMilliSecondSpan)
222:         {
223:             $this.MoveBlockToDown()
224:             $this.LastShiftMilliSecond = 0
225:         }
226:     } elseif ([System.Windows.Input.Keyboard]::IsKeyDown($keySpace))
227:     {
228:         if ($this.LastRotateMilliSecond -ge $this.RotateMilliSecondSpan)
229:         {
230:             $this.RotateBlock($true)
231:             $this.LastRotateMilliSecond = 0
232:         }
233:     }
234:
235:     # 縦方向の座標の変化が起こったかどうかの判定
236:     if ($this.CurrentBlock.IPos -ne $prevIPos) {
237:         return $true
238:     }
239:
240:     # テトリス盤面の描画
241:     $this.DrawTetrisBoard()
242:
243:     # 経過秒数の追加
244:     $stayMilliSecond += $this.UnitMilliSecond
245:     $this.LastShiftMilliSecond += $this.UnitMilliSecond
246:     $this.LastRotateMilliSecond += $this.UnitMilliSecond
247:     Start-Sleep -m $this.UnitMilliSecond
248: }
249:
250: return $false
251: }
252:
253: # テトリス盤面を次の局面へと進める
254: [void] GoNextPhase() {
255:     $this.BoardStatusString = ""
256:
257:     # テトリミノが非アクティブ状態になっている場合の処理

```

```

255:         if (-not $this.BlockActive) {
256:             $nextBlockPosArray = $this.NextBlock.BlockPosArray
257:
258:             # 次のテトリミノを出現させる空きスペースがあるかチェック
259:             if ($this.IsNewBlockAvailable($nextBlockPosArray)) {
260:                 # 次のテトリミノへの切り替え処理
261:                 $this.SwitchNextBlock()
262:                 $this.BlockActive = $true
263:                 $this.BoardStatusString = "New Block!"
264:             } else {
265:                 # ゲームオーバーのフラグを立てる
266:                 $this.GameOver = $true
267:                 $this.BoardStatusString = "Game Over!"
268:             }
269:             return
270:         }
271:
272:         # 一定の時間経過でテトリミノの座標を下方向にシフトする
273:         if ($this.IsDownSpaceAvailable($this.CurrentBlock)) {
274:             $this.CurrentBlock.IPos++
275:         } else {
276:             # 現在操作中のテトリミノを非アクティブにし、固定化処理を行う
277:             $this.BlockActive = $false
278:             $this.RockDownCurrentBlock()
279:             $this.BoardStatusString = "Fixed!"
280:
281:             # ライン消去処理とスコア加算を行う
282:             $eraseLineCount = $this.EraseLines()
283:             if ($eraseLineCount -gt 0) {
284:                 $this.Score += $eraseLineCount
285:
286:                 # 一定スコアごとにレベルアップ
287:                 $this.Level = [Math]::Floor($this.Score / 10) + 1
288:                 $this.StayMilliSecondLimit = `
289:                     $this.StayMilliSecondInit - ($this.Level - 1) * 5
290:
291:                 # 盤面描画処理と一定時間スリープ
292:                 $this.DrawTetrisBoard()
293:                 Start-Sleep -m 50
294:             }
295:         }

```

```

296:
297:     # 経過ターン数のカウントアップ
298:     $this.Turn++
299: }
300:
301: # テトリミノ固定化(ロックダウン)処理
302: [void] RockDownCurrentBlock() {
303:     foreach($pos in $this.CurrentBlock.BlockPosArray) {
304:         $i = $this.CurrentBlock.IPos + $pos[0]
305:         $j = $this.CurrentBlock.JPos + $pos[1]
306:         if ($i -lt 0) { continue }
307:         $this.BoardArray[$i][$j] = 1
308:     }
309: }
310:
311: # ライン消去処理
312: [int] EraseLines() {
313:     $eraseCount = 0
314:     $rowIndex = $this.BoardHeight - 1
315:     foreach ($_ in 1..($this.BoardHeight-1)) {
316:         # 対象となるラインがあるかをチェック
317:         if ($this.BoardArray[$rowIndex].Contains(0)) {
318:             $rowIndex--
319:         } else {
320:             # 各ライン消去
321:             $eraseCount++
322:             foreach ($i in $rowIndex..1) {
323:                 $this.BoardArray[$i] = $this.BoardArray[$i-1]
324:             }
325:             $this.BoardArray[0] = [int[]]::new($this.BoardWidth)
326:         }
327:     }
328:     return $eraseCount
329: }
330:
331: # 各種ステータスチェック用のメソッド群
332: # ゲームオーバーかどうか
333: [bool] IsGameOver() {
334:     return $this.GameOver
335: }
336:

```

```

337:      # 新しいテトリミノをいれるためのスペースが有るかどうか
338:      [bool] IsNewBlockAvailable($nextBlockPosArray) {
339:          foreach($pos in $nextBlockPosArray) {
340:              $i = $this.InitBlockIPos + $pos[0]
341:              $j = $this.InitBlockJPos + $pos[1]
342:              if ($i -lt 0) { continue }
343:              if ($this.BoardArray[$i][$j] -ne 0) { return $false }
344:          }
345:          return $true
346:      }
347:
348:      # 現在捜査中のテトリミノから見て、指定した方向について空きスペースがあるか
349:      [bool] IsShiftSpaceAvailable([Block]$block, $iShift, $jShift) {
350:          foreach($pos in $block.BlockPosArray) {
351:              $i = $block.IPos + $pos[0] + $iShift
352:              $j = $block.JPos + $pos[1] + $jShift
353:
354:              # 盤面や盤面端の境界情報に応じて、空きがない場合は$falseを返却する
355:              if ($i -gt ($this.BoardHeight - 1) -or
356:                  $j -lt 0 -or $j -gt ($this.BoardWidth - 1)) {
357:                  return $false
358:              } elseif ($this.BoardArray[$i][$j] -ne 0) {
359:                  return $false
360:              }
361:          }
362:          return $true
363:      }
364:      [bool] IsLeftSpaceAvailable([Block]$block) {
365:          return $this.IsShiftSpaceAvailable($block, 0, -1)
366:      }
367:      [bool] IsRightSpaceAvailable([Block]$block) {
368:          return $this.IsShiftSpaceAvailable($block, 0, 1)
369:      }
370:      [bool] IsDownSpaceAvailable([Block]$block) {
371:          return $this.IsShiftSpaceAvailable($block, 1, 0)
372:      }
373:
374:      # プレイヤー入力によるテトリミノの移動処理
375:      [void] MoveBlockToLeft() {
376:          if ($this.IsLeftSpaceAvailable($this.CurrentBlock)) {
377:              $this.CurrentBlock.JPos--

```

```

378:         }
379:     }
380:     [void] MoveBlockToRight() {
381:         if ($this.IsRightSpaceAvailable($this.CurrentBlock)) {
382:             $this.CurrentBlock.JPos++
383:         }
384:     }
385:     [void] MoveBlockToDown() {
386:         if ($this.IsDownSpaceAvailable($this.CurrentBlock)) {
387:             $this.CurrentBlock.IPos++
388:         }
389:     }
390:
391:     # プレイヤー入力によるテトリミノの回転処理
392:     [bool] RotateBlock($clockWise=$true) {
393:         $i = 0
394:         $j = 0
395:         $block = $this.CurrentBlock
396:         $rotateCenter = $block.RotateCenter
397:         $tempPosArray = @()
398:
399:         foreach($pos in $block.BlockPosArray) {
400:             $iDiff = $pos[0] - $rotateCenter[0]
401:             $jDiff = $pos[1] - $rotateCenter[1]
402:
403:             # 時計回りかどうかに応じて処理分岐
404:             if ($clockWise) {
405:                 $i = $block.IPos + $jDiff + $rotateCenter[0]
406:                 $j = $block.JPos - $iDiff + $rotateCenter[1]
407:                 $tempPosArray += `
408:                     ,@((- $jDiff + $rotateCenter[0]),
409:                       ($iDiff + $rotateCenter[1]))
410:             } else {
411:                 $i = $block.IPos - $jDiff + $rotateCenter[0]
412:                 $j = $block.JPos + $iDiff + $rotateCenter[1]
413:                 $tempPosArray += `
414:                     ,@(($jDiff + $rotateCenter[0]),
415:                       (- $iDiff + $rotateCenter[1]))
416:             }
417:
418:             # 盤面や盤面端の境界情報に応じて、回転処理をせずに $false を返却

```

```

419:         if ($i -lt 0 -or $i -gt ($this.BoardHeight - 1) -or
420:             $j -lt 0 -or $j -gt ($this.BoardWidth - 1)) {
421:             return $false
422:         } elseif ($this.BoardArray[$i][$j] -ne 0) {
423:             return $false
424:         }
425:     }
426:
427:     # 回転処理を行う
428:     $block.BlockPosArray = $tempPosArray
429:
430:     return $true
431: }
432: }
433:
434: # テトロリスの開始
435: $tetris = New-Object Tetris
436:
437: # 画面のクリア
438: Clear-Host
439:
440: # テトロリスの処理: ゲームオーバー判定になるまで継続
441: while (-not $tetris.IsGameOver()) {
442:     # プレーヤーの操作受け付け処理
443:     # Note: このフェーズではプレーヤー操作可
444:     while ($tetris.GoPlayerActionPhase()) {}
445:
446:     # テトロリスの盤面進行処理(落下・ライン消去・テトリミノ結合処理)
447:     # Note: このフェーズではプレーヤー操作不可
448:     $tetris.GoNextPhase()
449:
450:     # スリープ
451:     Start-Sleep -m 10
452: }
453:

```

4.20.0.4 実行結果

			[Next]
			I
	#		
	###		[Score]
			0000
			[Level]
			1
	##		##
	#####		
	#####		
	=====		

付録A 巻末付録

A.1 参考文献・URL

A.1.1 書籍

- ・ PowerShell 実践ガイドブック 〜クロスプラットフォーム対応の次世代シェルを徹底解説〜 (吉崎 生; マイナビ出版)

A.1.2 Web ページ

A.1.2.1 文字コード設定

- ・ <https://zenn.dev/kohlNigeeee/articles/854c04fafa9c01>

A.1.3 応用サンプル実装

A.1.3.1 サンプル 2

- ・ <https://zenn.dev/awtnb/articles/e54718efcd1b5b>

A.1.3.2 サンプル 3

- ・ <https://askubuntu.com/questions/493584/convert-images-to-pdf>

A.1.3.3 サンプル 4

- ・ <https://ginpen.com/2019/12/03/convert-pdf-to-png-jpeg-via-imagemagick/>
- ・ <https://nu-pan.hatenablog.com/entry/2015/06/09/231905>

A.1.3.4 サンプル 7

- ・ <https://ffmpeg.org/ffmpeg.html>
- ・ <https://blue-bear.jp/kb/ffmpeg-ffmpeg%E3%82%92%E4%BD%BF%E7%94%A8%E3%81%97%E3%81%A6%E9%9F%B3%E5%A3%B0%E3%81%AE%E6%9C%AB%E5%B0%BE%E3%81%AB%E6%8C%87%E5%AE%9A%E7%A7%92%E7%84%A1%E9%9F%B3%E3%82%92%E6%8C%BF%E5%85%A5%E3%81%99/>

A.1.3.5 サンプル 8

- ・ <https://stackoverflow.com/questions/36074224/how-to-split-video-or-audio-by-silent-parts>
- ・ <https://tech.mktime.com/entry/79>
- ・ <https://moritanoeigo.info/lab/category/%E9%9F%B3%E5%A3%B0%E3%83%95%E3%82%A1%E3%82%A4%E3%83%AB/>

A.1.3.6 サンプル 12

- <https://qiita.com/kurukurupapa@github/items/edb8b6a79135f159ffcb>
- <https://tarenagashi.hatenablog.jp/entry/2020/12/07/213259>

A.1.3.7 サンプル 14

- <https://qiita.com/mindwood/items/3c329d4a24bc8a3ab765>
- <https://www.powershellgallery.com/packages/SecurityFever/2.4.0/Content/Helpers%5CCommon%5CConvert-Base32ToByte.ps1>
- <https://www.powershellgallery.com/packages/SecurityFever/2.4.0/Content/Functions%5CCommon%5CGet-TimeBasedOneTimePassword.ps1>

A.1.3.8 サンプル 19

- <https://ja.wikipedia.org/wiki/%E3%83%A9%E3%82%A4%E3%83%95%E3%82%B2%E3%83%BC%E3%83%A0>

著者紹介

○○ ○○ (○○ ○○)

○

○○ ○○ (○○ ○○)

○

◎本書スタッフ

アートディレクター/装丁：岡田章志+GY

編集協力：■■■■■

ディレクター：栗原 翔

〈表紙イラスト〉

■■■■■

技術の泉シリーズ・刊行によせて

技術者の知見のアウトプットである技術同人誌は、急速に認知度を高めています。インプレス NextPublishingは国内最大級の即売会「技術書典」(<https://techbookfest.org/>) で頒布された技術同人誌を底本とした商業書籍を2016年より刊行し、これらを中心とした『技術書典シリーズ』を展開してきました。2019年4月、より幅広い技術同人誌を対象とし、最新の知見を発信するために『技術の泉シリーズ』へリニューアルしました。今後は「技術書典」をはじめとした各種即売会や、勉強会・IT会などで頒布された技術同人誌を底本とした商業書籍を刊行し、技術同人誌の普及と発展に貢献することを目指します。エンジニアの“知の結晶”である技術同人誌の世界に、より多くの方が触れていただくきっかけになれば幸いです。

インプレス NextPublishing

技術の泉シリーズ 編集長 山城 敬

●お断り

掲載したURLは202■年■月1日現在のものです。サイトの都合で変更されることがあります。また、電子版ではURLにハイパーリンクを設定していますが、端末やビューアー、リンク先のファイルタイプによっては表示されないことがあります。あらかじめご了承ください。

●本書の内容についてのお問い合わせ先

株式会社インプレス

インプレス NextPublishing メール窓口

np-info@impress.co.jp

お問い合わせの際は、書名、ISBN、お名前、お電話番号、メールアドレス に加えて、「該当するページ」と「具体的な質問内容」「お使いの動作環境」を必ずご明記ください。なお、本書の範囲を超えるご質問にはお答えできないのでご了承ください。

電話やFAXでのご質問には対応しておりません。また、封書でのお問い合わせは回答までに日数をいただく場合があります。あらかじめご了承ください。

奥付サンプル

201X年X月X日 初版発行Ver.1.0（PDF版）

著 者 × × × ×

編集人 × × × ×

発行人 × × × ×

発 行 株式会社インプレスR&D

〒101-0051

東京都千代田区神田神保町一丁目105番地

<http://nextpublishing.jp/>

●本書は著作権法上の保護を受けています。本書の一部あるいは全部について株式会社インプレスR&Dから文書による許諾を得ずに、いかなる方法においても無断で複写、複製することは禁じられています。

©201X, All rights reserved.

ISBN978-4-XXXX-XXXX-X



NextPublishing®

●インプレス NextPublishingは、株式会社インプレスR&Dが開発したデジタルファースト型の出版モデルを承継し、幅広い出版企画を電子書籍＋オンデマンドによりスピーディで持続可能な形で実現しています。<https://nextpublishing.jp/>