2015年12月25日 第1.0版

2018年11月30日 第1.1版

正規表現入門

入門

目次

[はじめに 3](#_Toc531635690)

[正規表現とは 3](#_Toc531635691)

[教材アプリについて 3](#_Toc531635692)

[サクラエディタの正規表現 4](#_Toc531635693)

[正規表現解説 4](#_Toc531635694)

[基本要素 5](#_Toc531635695)

[文字集合(キャラクタクラス) 5](#_Toc531635696)

[量指定子(数量子) 6](#_Toc531635697)

[文字 7](#_Toc531635698)

[文字種 8](#_Toc531635699)

[位置(錨) 8](#_Toc531635700)

[後方参照と部分式呼び出し 9](#_Toc531635701)

[拡張式集合 10](#_Toc531635702)

[置換で使える参照 11](#_Toc531635703)

[bregonig.dllへの変更による違い 11](#_Toc531635704)

[改行の検索 12](#_Toc531635705)

[正規表現課題集 13](#_Toc531635706)

[課題１ 13](#_Toc531635707)

[課題２ 13](#_Toc531635708)

[課題３ 13](#_Toc531635709)

[課題４ 14](#_Toc531635710)

[課題５ 14](#_Toc531635711)

[課題６ 14](#_Toc531635712)

[課題７ 15](#_Toc531635713)

[課題８ 15](#_Toc531635714)

[正規表現課題実例 17](#_Toc531635715)

[課題１実例 17](#_Toc531635716)

[課題２実例 17](#_Toc531635717)

[課題３実例 17](#_Toc531635718)

[課題４実例 17](#_Toc531635719)

[課題５実例 18](#_Toc531635720)

[課題６実例 18](#_Toc531635721)

[課題７実例 18](#_Toc531635722)

[課題８実例 18](#_Toc531635723)

[おわりに 19](#_Toc531635724)

[改版履歴 20](#_Toc531635725)

# はじめに

本文書は「正規表現」を初めて学習する人を想定して作成しました。

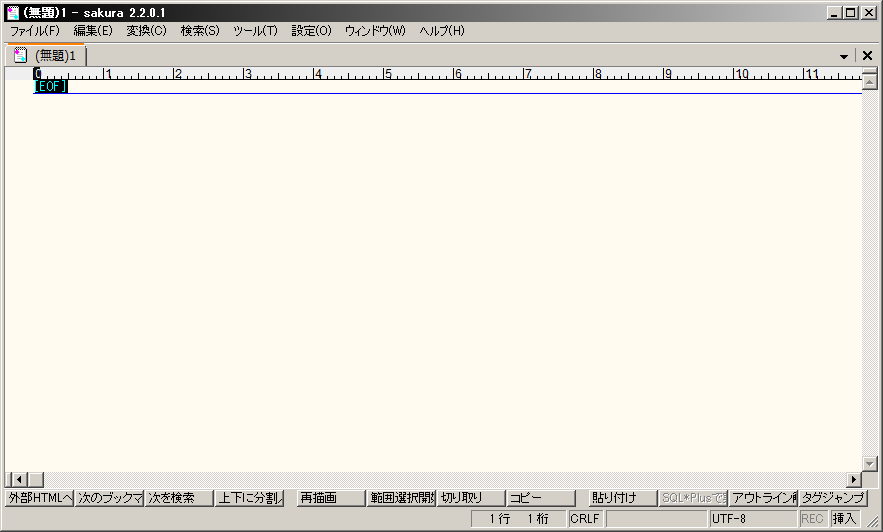
# 正規表現とは

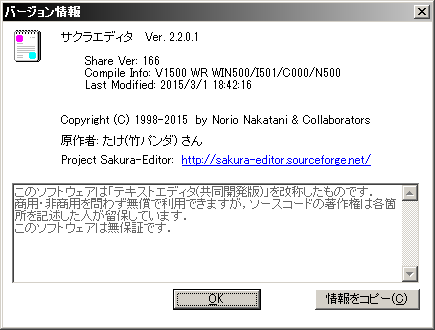
正規表現とは、検索や置換をする際に文字列をマッチングさせるための文字列パターンです。正規表現を使わないで検索や置換をする時は、単に検索対象の文字列を指定します。例えば「Apple」を検索したい場合は、「Apple」で文字列検索します。しかし「apple」や「APPLE」というように同じ単語なのに大文字小文字の文字構成が異なるとマッチしなくなります。（オプションで大文字・小文字を無視する設定がある場合は可能であるが）

上記「Apple」の例のようにマッチング対象文字列が複数パターンある場合に正規表現が役立ちます。正規表現はもともと形式言語理論において正規言語を表すための手段として発明されたそうですが、コンピュータの分野で利用・拡張されて今では正確には正規表現ではない文字列も表せるようになっているそうです。ですが伝統的に正規表現と呼ばれ続けています。コマンドラインやプログラム言語では英語表記でよく登場しますので「Regular Expression」「regexp」「re」などの呼び方もあわせて覚えておきましょう。

業務上では、テキストを整形してExcelに貼ったり、ログ調査で対象行をGrepしたり、プログラムでは入力値チェックなどで幅広く利用できます。

# 教材アプリについて

本文書では、フリーの定番エディタである「サクラエディタ」を使って正規表現を学習します。事前にサクラエディタをインストールしてください。バージョンについては特に気にしなくて大丈夫です。

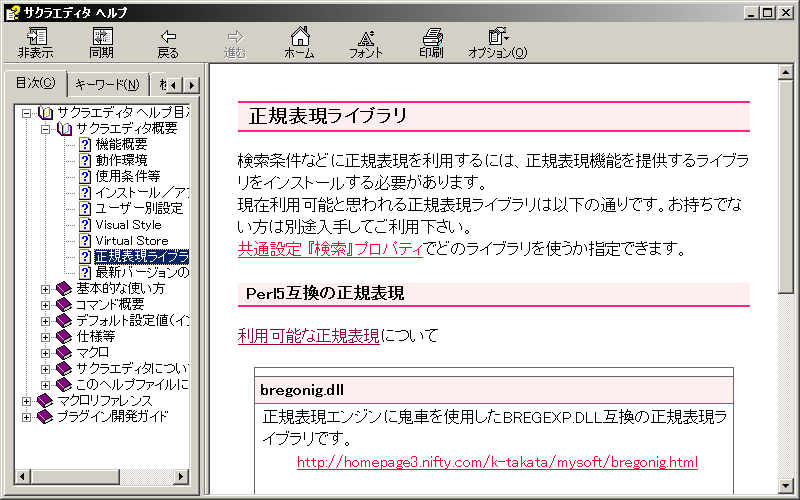


正規表現の正規言語自体は一般化されていますが、正規表現を提供しているライブラリにより若干動作に違いがあるため、利用する際は利用元のマニュアルを参照し動作確認もする必要があります。サクラエディタ以外のエディタで学習しても良いですが、本文の通りに動作しない可能性があります。

# サクラエディタの正規表現

以降は製品付属のマニュアルを参照しながら学習します。製品のヘルプを開いてください。

ヘルプメニュー > ヘルプ目次 > サクラエディタ概要 > 正規表現ライブラリ



ヘルプの記載よりサクラエディタで正規表現を利用するには「bregonig.dll」の導入が必要なこと、「Perl5互換」であることなどが記載されています。ライブラリは通常は標準インストールで導入されます。「Perl」はスクリプト言語のひとつで正規表現が拡張されており豊富な文法が実装されています。

# 正規表現解説

サクラエディタヘルプの「利用可能な正規表現」をクリックすると、サクラエディタにおける正義表現の文法と構文を確認することができます。

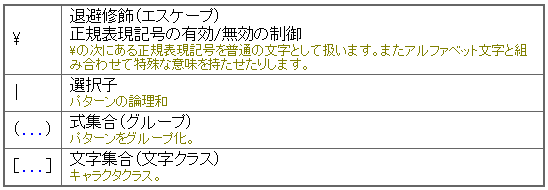
本文書では、日ごろ使用頻度が高いと思われるものをピックアップして解説します。またサクラエディタのライブラリ固有拡張については汎用性が低いため取り上げません。

以降、エディタのヘルプ抜粋に補足する形で解説します。

## 基本要素

正規表現を組み立てるための基本的な要素です。各要素の役割は以下表の通りです。各要素はそのまま使うと正規表現記号として認識されるので、記号を文字として使用したい場合はエスケープする必要があります。

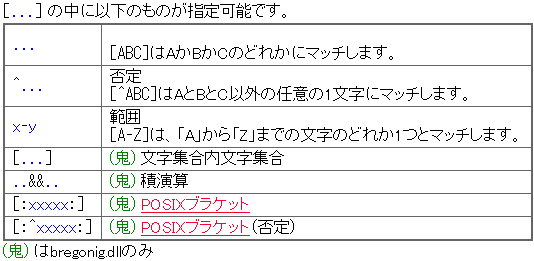
例： 「 \(1,000) 」を文字として認識させたい→ 「 \\\(1,000\) 」



例えば、英単語で灰色は「gray」と「grey」の両方の綴りがありますが、これを一度に検索しようとすると以下のように表現することができます。

「 **gray|grey** 」 または 「 **gr(a|e)y** 」

## 文字集合(キャラクタクラス)



先頭大文字のアルファベットで始まる3文字の英単語を検索したい場合は、以下のように表現することができます（「[a-z]」の部分は、「[a-c]」のように範囲を狭めることも可能です）。

「 **[A-Z][a-z][a-z]** 」

例えば「hat, cat, bat, rat」の4英単語のうち「hat」のみ除外して検索したい場合は、以下のように表現します。

「 **[^h]at** 」

英字の大文字小文字、および数字を問わず検索対象としたい場合は「 **[A-Za-z0-9]** 」のようにまとめて指定することも可能です。正規表現が正しく動作するかは、例えば以下のように適当にテキストをサクラエディタに打って、そして実際に検索して試してみると良いです（文章に特に意味はありません）。

A rat is gray.

An elephant is also grey.

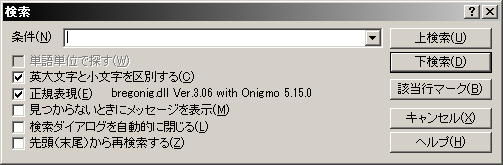
Red Hat is not gray.

cat bat hat 1at +at

注意：検索オプションについて

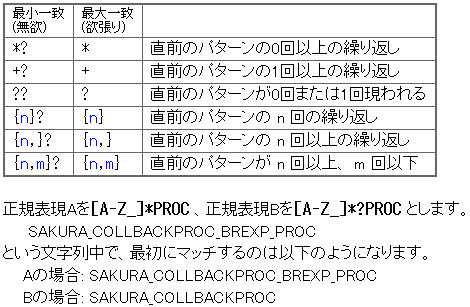
・「正規表現」にチェックを入れないと正規表現として認識されません。

・「大文字と小文字を区別する」にチェックを入れないと大文字小文字の差異は無視されます。



## 量指定子(数量子)

正規表現における「量子」とは、要は繰り返しのことです。繰り返しパターンの回数によって表現が異なります。表見出しの「最小一致」「最大一致」の違いについては、表下に記載されている例を参照してください。「PROC」が複数含まれている語をマッチングした時の結果が違うことが分かります。



まずは「 ? 」「 \* 」「 + 」の3つを使いこなせるようになりましょう。

? … 0個か1個か →「 go?gle 」でマッチするのは「 ggle 」と「 gogle 」です。

\* … 0個以上 →「 go\*gle 」でマッチするのは「 ggle 」や「 gogle 」や「 goooogle 」などです。

+ … 1個以上 →「 go+gle 」でマッチするのは「 gogle 」や「 google 」や「 gooooogle 」などです。

細かく繰り返し回数を指定したい場合は、「 {n} 」の表現を使います。

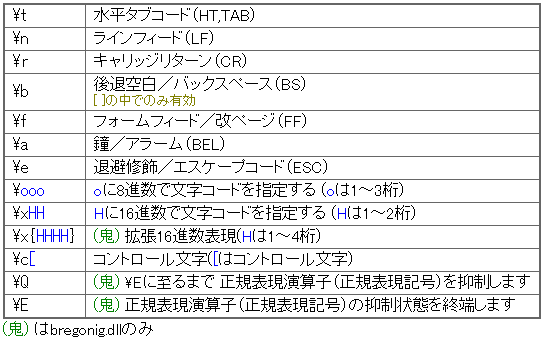
「 go{1}gle 」 → 「 gogle 」にマッチ。

「 go{2}gle 」 → 「 google 」にマッチ。

「 go{5}gle 」 → 「 gooooogle 」にマッチ。

実は「 ? 」「 \* 」「 + 」の記号は、それぞれ「 {0,1} 」「 {0,} 」「 {1,} 」の表現と同意です。

## 文字

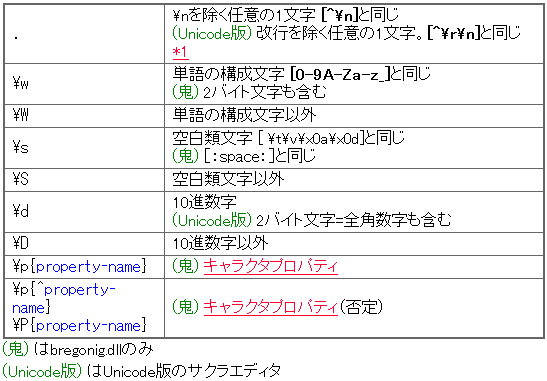


正規表現の便利な機能として、制御文字コードを指定することができます。まずは「 \t 」（タブ記号）、「 \n 」（ラインフィード）、「 \r 」（キャリッジリターン）の3つを覚えれば十分です。

例えばExcelはセル間の区切り文字としてタブ記号を使用しますので、ログなどを調査しやすいように区切り箇所にタブ記号を挿入してExcelに貼り付けたり、環境変数のPATHのように「 ; 」の区切り文字を改行（「 \r\n 」）に置換して各要素を読みやすくしたりするなどに使用します。

注意： 改行記号はOSによって異なります。一部例外はありますが標準的にWindowsの改行記号はCRLF（ \r\n ）で、MacOSやLinuxはLF（ \n ）です。Linuxから持ってきたテキストデータがWindowsで改行されないのはこのOS差異のせいです。この場合は「 \n 」を「 \r\n 」に一斉置換すると直ります。

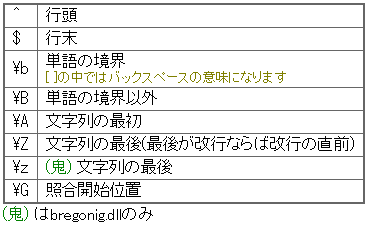
## 文字種



文字種で一番便利でよく使うのは「 . 」（ドット／ピリオド）です。改行記号以外の任意の文字を指定できる最強のワイルドカードとして使用できます。例えば「 Go.gle 」と指定した場合、「 Goagle 」も 「 Go1gle 」も「 Go@gle 」も、改行以外の任意の文字ならばなんでもマッチします。量子記号と組み合わせて「 .+ 」と検索するとテキストの全行がマッチします。

その他の文字種については、特定文字の指定が容易にできるように用意されています。例えば「 \d 」と「 [0-9] 」は同等です。また「 \s 」はスペース、タブ、改行などの空白文字にマッチします。文字種については、正規表現のライブラリによって実装が異なる場合がありますので要注意です。

## 位置(錨)

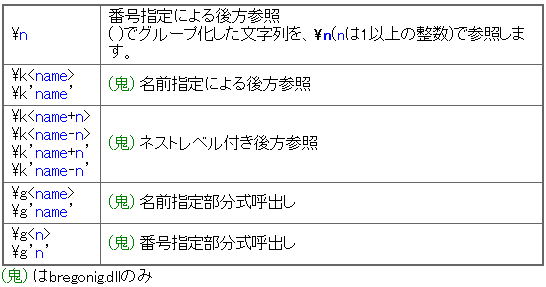


位置指定は、マッチング対象文字列の中のどの位置かを指定するのに利用します。英語のように単語間がスペースで区切られる文章では便利です。入門としては、「 ^ 」（行頭）と「 $ 」（行末）の2つを覚えましょう。

例えばログの調査などで、行頭から日付（2018/11/30）までが不要なため除去したいという場合は、「 ^.\*2018/11/30 」のように指定して一括で空文字置換すれば簡単に削除することができます。また、行末のピリオドだけを検索したい場合は、「 \.$ 」と指定すれば、行中に何回ピリオドが登場しようと行末のピリオドだけがマッチします。なお、ピリオドは正規表現の文字なので、エスケープしています。

ちなみに、この「 ^ 」と「 $ 」は正規表現だけではなく、Linuxのコマンドラインやviなどでも同じ意味として採用されており、コマンド入力中に先頭へ行ったり末尾に行きたい時に入力すれば同様に働きます。

## 後方参照と部分式呼び出し



正規表現でマッチした文字列のうちの一部分を利用して置換したい場合、後方参照の仕組みを利用します。正規表現の指定をする時に「 () 」で囲った文字列は先頭から順に記憶されます。置換する際に先頭から「 \1 」「 \2 」…のように指定します（後述しますが「 $1 」「 $2 」…のように円記号ではなくドル記号でも同様に働きます）。

例えば、以下のような文章があるとします。マッチングした一部分を記憶して置換することができます。

英語を指定したい場合は「en」を指定してください。

日本語を指定したい場合は「jp」を指定してください。

マッチングパターン「 ^(.+語).+「(..)」.+$ 」

置換後のパターン「 \1 → \2 」

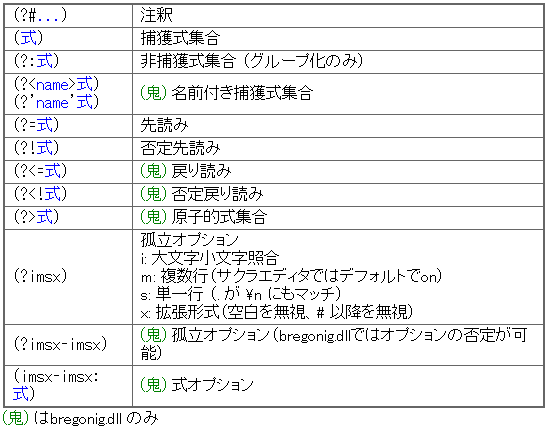
英語 → en

日本語 → jp

行頭の「～語」を \1 に、カギ括弧（「」）内の2文字を \2 に記憶し、置換で利用しています。

例では「 → 」に置換しましたが、タブ記号（\t）に置換すればExcelでの利用に便利です。

## 拡張式集合



本項目は特殊な表現のため、入門編としては一部のみ取り扱います。

まず括弧「 () 」ですが、正規表現を括弧でくくると、括弧内で指定した表現が一致した箇所が記憶され置換後に利用できるようになります（「後方参照と部分式呼び出し」の項目を参照）。この記憶されることを「捕獲」と訳したと思われます（多分Captureの和訳）。よって「非捕獲式集合」の表現は記憶したくない場合に使用します。

次に「注釈」「先読み」「否定先読み」です。例えば、以下のように3人の人物がいたとします。

海田太郎

山田太郎

山田花子

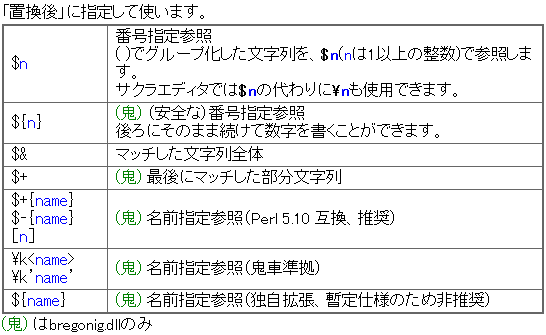
「 山田(?#太郎) 」 … 「山田太郎」「山田花子」両方の行の「山田」がマッチします。

「 山田(?=太郎) 」 … 「山田太郎」の行の「山田」のみがマッチします。

「 山田(?!太郎) 」 … 「山田花子」の行の「山田」のみがマッチします。

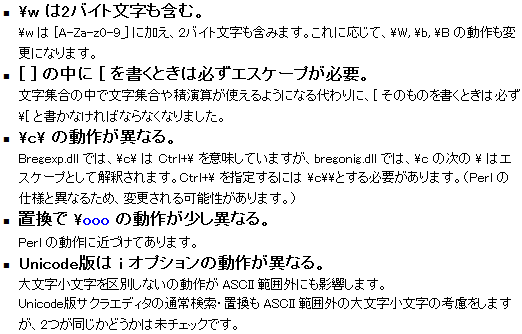
注釈は特にマッチングに影響を与えません。「先読み」と「否定先読み」については、直前の「山田」に続く文字列に着目し、「太郎」という語が続くか否かで絞り込んで検索することが可能になります。

## 置換で使える参照



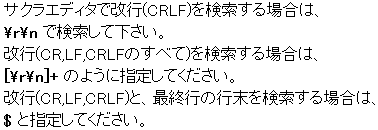
後方参照の項で既に登場しましたが、置換で使用する表現です。「 \n 」の代わりに「 $n 」でも指定することが可能です。正規表現として「 \n 」を使うのはサクラエディタが特殊で、「 $n 」の方が一般的なようです。

## bregonig.dllへの変更による違い



上記記載のように必ず利用する正規表現ライブラリには、ヘルプに注意書きがあります。「あれ？正規表現が動かないぞ？」という場合はマニュアルの例外事項に該当しないか確認すると良いでしょう。

## 改行の検索



改行のマッチングについてのヘルプです。正規表現ライブラリによっては、「 \n 」だけでCRLFもマッチングしたり、末尾の「 $ 」が不要だったりするので、サクラエディタ以外で正規表現を利用する時は注意が必要です。

# 正規表現課題集

以下課題があります。正規表現を使用して解決してください。

※テキストボックスの中身をサクラエディタに貼り付け、正規表現を使って置換してください。

## 課題１

以下の文章を箇条書きにしてください。

Microsoft Internet Explorer

Microsoft Edge

Mozilla FireFox

Google Chrome

Apple Safari

## 課題２

以下の文章には行末に不要な空白が入っています。全て除去してください。

・Microsoft Internet Explorer

・Microsoft Edge

・Mozilla FireFox

・Google Chrome

・Apple Safari

## 課題３

以下の環境変数Path（1行）の区切り文字「;」を改行コードに置換して読みやすくしてください。

Path=C:\Program Files (x86)\Common Files\Oracle\Java\javapath;C:\Program Files (x86)\Microsoft SDKs\Azure\CLI2\wbin;C:\Program Files (x86)\RSA SecurID Token Common;C:\Program Files\RSA SecurID Token Common;C:\ProgramData\Oracle\Java\javapath;C:\Program Files (x86)\Intel\iCLS Client\;C:\Program Files\Intel\iCLS Client\;C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Program Files\Intel\Intel(R) Management Engine Components\DAL;C:\Program Files\Intel\Intel(R) Management Engine Components\IPT;C:\Program Files (x86)\Intel\Intel(R) Management Engine Components\DAL;C:\Program Files (x86)\Intel\Intel(R) Management Engine Components\IPT;C:\opt\apache-maven-3.3.9\bin;C:\Program Files\nodejs\;C:\Program Files\TortoiseSVN\bin;C:\Program Files\dotnet\;C:\Program Files\Microsoft SQL Server\130\Tools\Binn\;C:\Program Files\Git\cmd;C:\Windows\System32\WindowsPowerShell\v1.0\;C:\Program Files\Microsoft\Web Platform Installer\;C:\Program Files (x86)\Microsoft SDKs\Azure\CLI\wbin;C:\Program Files\PuTTY\;C:\Program Files\Microsoft VS Code\bin;C:\Windows\System32\WindowsPowerShell\v1.0\;C:\Program Files\RSA SecurID Token Common\;C:\Program Files\Microsoft VS Code\bin

## 課題４

以下の文書の開発元と製品名の間にタブ記号を入れて、Excelに貼り付けてください。

Microsoft Internet Explorer

Microsoft Edge

Mozilla FireFox

Google Chrome

Apple Safari

## 課題５

不要なデータを削除した結果、以下のような文書になりました。空行を除去してください。

有袋目には以下のような動物がおります。

・コアラ

・オオカンガルー

・ベネットアカクビワラビー

・フクロモモンガ

## 課題６

以下のHTMLソースは空白でインデントされています。空白4文字を1タブ記号に置換してください。

<!DOCTYPE html>

<html lang="ja">

<head>

<meta charset="UTF-8">

<title>RegExp Challenge</title>

</head>

<body>

<p>こんにちは世界！</p>

</body>

</html>

## 課題７

コマンドプロンプトのfindstrコマンドと正規表現を使ってファイル内から灰色（「gray」または「grey」）を含む行を検索してください。手順に記載したファイル名やパスは作例であり、実際は任意で構いません。

手順1: 以下ドキュメントを、C:\work\regex\_test.txt として保存します。

A rat is gray.

An elephant is also grey.

Red Hat is not gray.

cat

bat

hat

1at

+at

手順2： コマンドプロンプトを起動します（Windowsキー＋R ＞ cmd と入力してOK）。

手順3： findstrのコマンドヘルプを参照します（ help findstr と入力）

手順4： コマンドヘルプを参照して、正規表現を使って検索します。

ヒント： findstr /R “<正規表現>” C:\work\regex\_test.txt

（使用できる正規表現については、コマンドヘルプを参照）

## 課題８

以下ソースは入力値を精査し、郵便番号かどうかの判定を行うJavaScriptです。ソースコード内の**＜正規表現＞**の部分に郵便番号（000-0000）のフォーマットで入力されているかチェックするための正規表現を記述してください。またHTMLファイルとして保存して動作確認をしてください。

<!DOCTYPE html>

<html lang="ja">

<head><meta charset="utf-8"/></head>

<body>

<input type="text" id="inval" value="" placeholder="入力してください">

<input type="button" value="check" onClick="checkInval()">

<br>

<div id="result"></div>

<script>

function checkInval() {

var inval = document.getElementById("inval");

var result = document.getElementById("result");

if(inval.value.match(/**＜正規表現＞**/)){

result.innerHTML = "郵便番号です";

} else {

result.innerHTML = "郵便番号ではありません";

}

}

</script>

</body>

</html>

# 正規表現課題実例

以下に実例を記載します。あくまでも例であり、同じでなくても問題ないです。

**最終的に希望することが実現できれば、それは正解です。**

## 課題１実例

文頭に箇条書きを示す記号を置換します。以下例では箇条書き記号として点（・）に置換しました。

置換前： ^

置換後： ・

## 課題２実例

単語と単語の間にある空白には影響が無いように置換する必要があります。

末尾の空白を除去すればよいので、空白の連続、かつ末尾を示す記号を置換します。

置換前： □+$

置換後：

※置換前の「□」部分は空白1文字と読んでください（ +$）。また、置換後は空です。

## 課題３実例

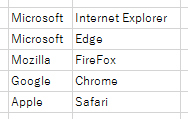
区切り文字（delimiterやseparatorと呼びます）を編集して読みやすくする課題です。ちなみに改行は行と行の間の区切り文字という意味でline separatorとも呼びます。課題とは逆に改行を除去して区切り文字に置換する場合もあります。

置換前： ;

置換後： \r\n

## 課題４実例

タブ記号を入れてExcelに貼ると以下のようになります。Internet Explorerだけ空白が2カ所あるので工夫がいるところです。実例を2つ提示します。1つ目の例は繰り返し回数を最小一致で指定し、英字の文字列と空白が1回目の部分を指定して置換しています。2つ目の例は空白の前方を$1、後方を$2に記憶させて置換しています。他にも指定方法がありますので試してみてください。



置換前： ^([A-Z][a-z]+) ?

置換後： $1\t

または

置換前： ^([A-Z][a-z]+) ([A-Z].+)$

置換後： $1\t$2

## 課題５実例

不要な空行を一気に削除することはExcelでもできますが、正規表現でも簡単にできます。

空行とは、要は行頭からすぐに行末になり改行される行を指すのでその通りに表現します。

置換前： ^$\r\n または ^[\r\n]+

置換後：

※置換後の指定は空です。

## 課題６実例

コードフォーマッターの作成例です。正規表現によりルールを作成して置換を繰り返せば、ソースコードを綺麗に整形することができます（更にサクラエディタのマクロ機能を使って置換の流れを記録させれば何度でも再利用できます）。今課題では空白4つをタブ記号に置換するので空白の繰り返し回数を指定します。

置換前： □{4}

置換後： \t

※置換前の「□」部分は空白1文字と読んでください( {4})。

## 課題７実例

サクラエディタ以外の正規表現を使用してみる課題です。findstrは使用できる正規表現が限られているため工夫する必要があります。aまたはe（a|e）という指定が出来ないため、実例では[a-e]の範囲指定で実現してみました。

findstr /R "gr[a-e]y" c:\work\regex\_test.txt

検索結果：

A rat is gray.

An elephant is also grey.

Red Hat is not gray.

## 課題８実例

JavaScriptの正規表現を使用してみる課題です。JavaScriptの正規表現の仕様はMDN（Mozilla Developer Network）などのサイトを参照して欲しいですが、基本的な要素はサクラエディタと同じです。郵便番号は「数字3桁 ハイフン 数字4桁」で構成されていますので、その通りに表現します。

^[0-9]{3}-[0-9]{4}$ または ^\d{3}-\d{4}$ など

# おわりに

正規表現は決して万能ではないですが、知っているのと知らないのとでは日頃の作業効率が相当変わります。ぜひ日々の業務で役立ててください。無理に1回で置換しようとしなくても大丈夫です。1回で置換できない場合は求める結果を得られるまで何回でも置換を繰り返せば良いです。サクラエディタのマクロ機能を使って記録と再生をすれば毎回入力しなくてもよいためツールのように使えます。

正規表現を利用できるテキストエディタはサクラエディタ以外も多くありますので好きなものを使って構いません（エディタは複数のエディタインストールしておいて用途に応じて使い分けるのが良いです）。また、正規表現はLinuxでのviエディタでも利用できますし、grep/sed/awkといったコマンドでバッチ処理を作成する時にも大活躍しますので是非使いこなしてください。

多くのスクリプトやプログラム言語でも正規表現はサポートされていますので例えば入力値チェックなどをシンプルに実装することができます。しかし自分でプログラムを書かない分、性能には注意を払ってください。正規表現はいろんな書き方ができるため、もしかするとコストが高いマッチング方法で記述してしまう可能性があります。（例えば1万回ループさせてみてコスト検証してから実装します）

正規表現には本文書では取り上げなかった表現がまだ残っています。知識は力なので是非マニュアルやインターネットで情報を集め応用していってください。正規表現を覚えたことによって業務の生産性が上がったならば幸いです。

# 改版履歴

|  |  |  |  |
| --- | --- | --- | --- |
| 版数 | 日付 | 名前 | 改版内容 |
| 1.0 | 2015/12/25 | 奥村健二 | 新規作成 |
| 1.1 | 2018/12/03 | 奥村健二 | 1.0版の紙資料をベースに再作成 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |