正規表現（Regular Expression）

1. **正規表現とは？**

　正規表現はUnix系ではおなじみのテキスト処理における技術です。PerlやJAVA、Pythonなどの主要な言語では標準でサポートされています。VBAでも使え、ExcelやAccessで使うことができます。

あるディレクトリ配下のファイル名をリストアップするときにワイルドカードを使って

dir \* dir A\* dir \*.txt

と指定したことはあるでしょう。このワイルドカードのようにマッチさせたいパターンを指定する方法が正規表現です。ワイルドカードに似ていますが、それよりもはるかに強力なパターンマッチングが行えます。文中から特定のパターンを検索したり、日付を一括して別の日に書き換えたり、またはプログラムコードを構文木に落とす前のトークン分割など、テキスト処理の前処理として頻繁に使われてきました。

例えばCSVファイルをカンマ区切りで項目に分割するとしましょう。単純にカンマで区切ってデータを取り出すだけではクォーテーションの中にカンマや改行がある場合にはうまくいきません。このような場合にいちいちプログラムコードを書いていると大きな工数がかかってしまいますが、正規表現を使うことでこの工数が大幅に削減されます。共通技術が使われることでコードもすっきりしてメンテナンス性が非常によくなります。

ひと昔前までは正規パターンは言語によって実装が異なっているとされていましたが、今はGNUのライブラリで実装されるため、だいたいどのプラットフォームでも同じ正規パターンを流用できるようになっています。

正規パターンを組み立てるのはパズルを解く感覚で楽しく作ることもできますが、自分が悩むところは他の人も同じように悩むところなのでWEBを検索するとたいてい良いパターンが挙げられていたりします。見つからない場合でも質問すれば親切な人が教えてくれることもあるかもしれません。汎用性が高いので、誰か頭の良い人が作ったパターンを流用することが簡単にできます。複雑すぎて解析や作り変えが不能な変態的なパターンもありますが、そのような場合でもブラックボックスとして十分流用することはできます。

正規表現は特定の言語や製品に特化した技術ではなく、標準的な技術で汎用性があり、テキスト処理においてとても便利に使えます。すでに何十年も使われて十分実績のある枯れた技術なので覚えておいて損はありません。

【簡易なメールアドレスの正規表現例】

[\w-\.]+@[\w-\.]+[A-Za-z]+

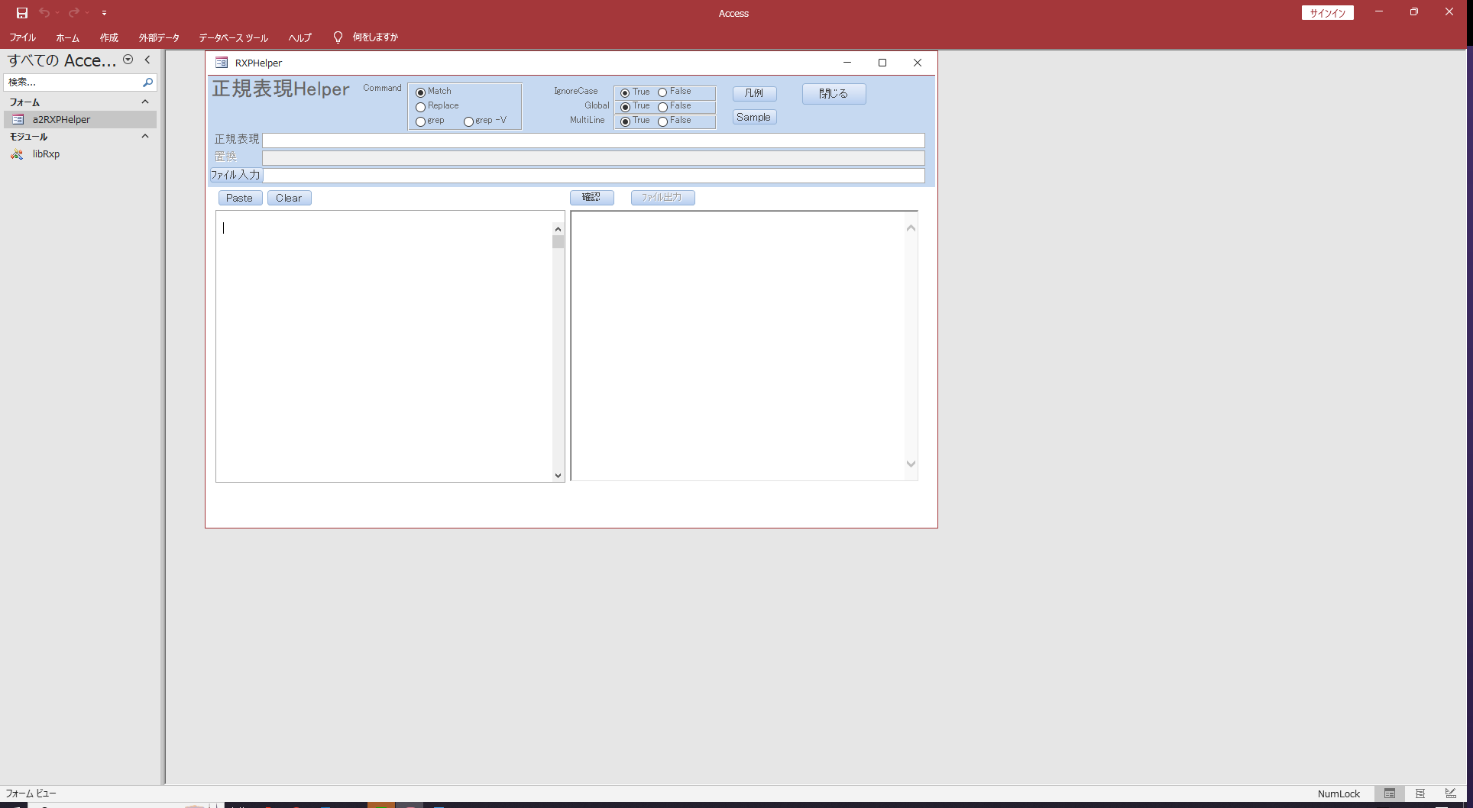
【RFC5322に準拠したメールアドレスの正規表現例】

^([\w!#$%&'\*+\-\/=?^`{|}~]+(\.[\w!#$%&'\*+\-\/=?^`{|}~]+)\*|"([\w!#$%&'\*+\-\/=?^`{|}~.()<>\[\]:;@,]|\\[\\"])+")@(([a-zA-Z\d\-]+\.)+[a-zA-Z]+|\[(\d{1,3}(\.\d{1,3}){3}|IPv6:[\da-fA-F]{0,4}(:[\da-fA-F]{0,4}){1,5}(:\d{1,3}(\.\d{1,3}){3}|(:[\da-fA-F]{0,4}){0,2}))\])$

**２.　正規表現Helper**

正規表現Helperを使って以下の正規表現コマンドを実行できます。

1. 正規パターンの確認
2. 正規表現による置換、行の抽出
3. ファイルを指定して②の結果をファイル出力



【正規表現コマンド】

　Match：　 　マッチ部分を強調表示して正規パターンを確認できます。

Replace：　正規表現による文字列の置換を行います。

grep：　正規表現にマッチした行（grepVはマッチしない行）を抽出します

【ボタン】

・ファイル入力 … テストしたいテキストファイルを指定します。

・ファイル出力 … 置換／行抽出結果をファイルに出力します。

【オプションボタン】

　Global：　 　True：マッチ部分を繰返し/False：初回にマッチした部分のみ

　IgnoreCase：True：大文字小文字を区別しない/False：区別する

　Multiline:　　True：複数行指定／False：テキスト全体を一行として扱う

【正規表現Hellper使用例】

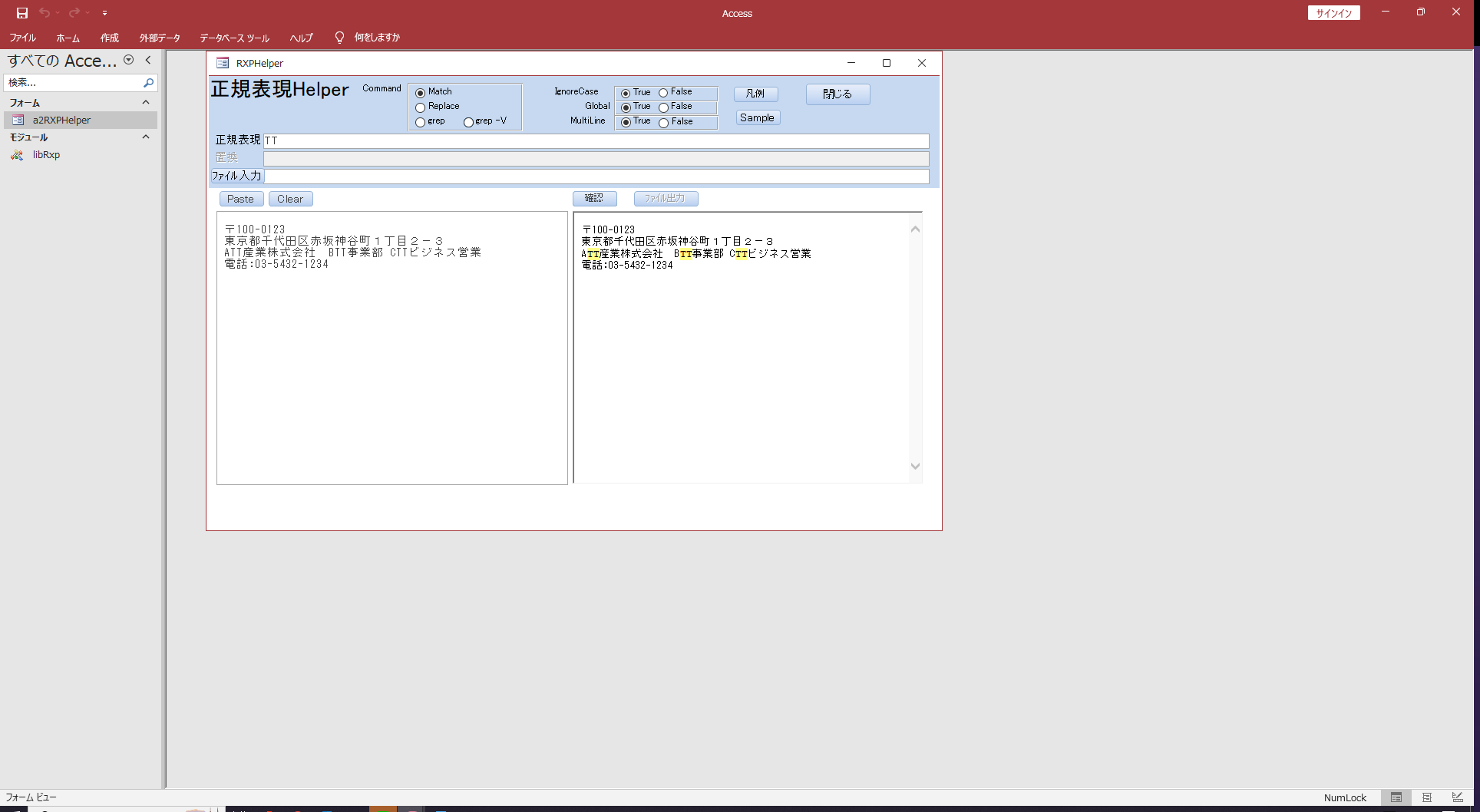
下記のテキストから「TT」という文字を検索してみましょう。正規表現に「TT」を指定します。

〒100-0123

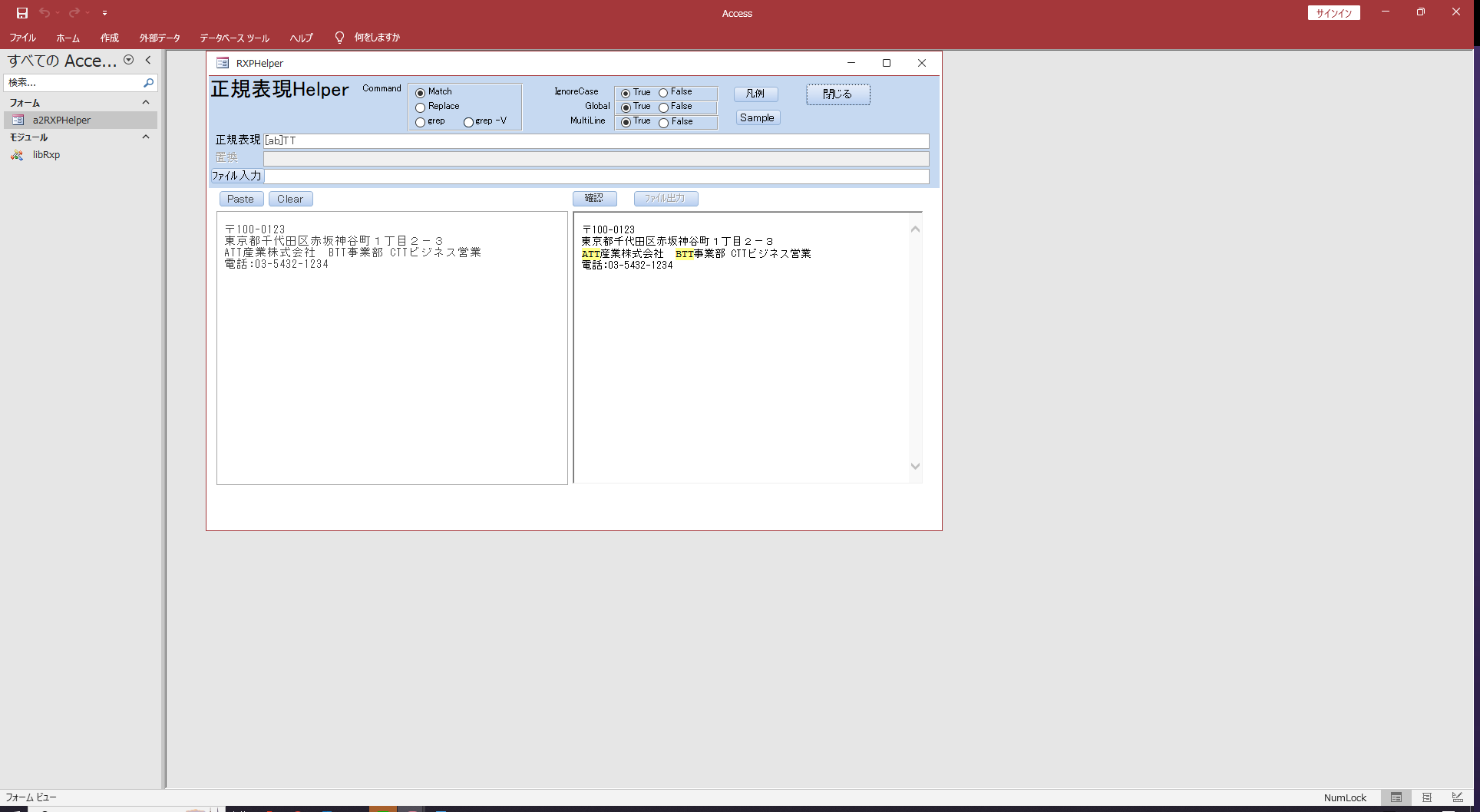
東京都千代田区赤坂神谷町１丁目２－３

ATT産業株式会社　BTT事業部 CTTビジネス営業

電話:03-5432-1234



ATTまたはBTTだけにマッチさせたい場合 [ab]TT と指定します。



[ab]TT　の[ ] は「リスト内のいずれか一文字」を表します。これによりaTTまたはbTTがマッチすることになります。[]のようにリストやパターンを指定する文字をメタ文字といいます。組み合わせによって多彩に変化するため多くの種類があるように感じますが、実はそれほど多くありません。短いパターンの場合はじっくり解読すれば内容は理解可能です。

ファイル入力ボタンでテキストファイルを指定すると冒頭の100行をテストペインに読み込みます。

置換またはgrepコマンドでが指定されると入力ファイルをフィルタリングして結果をファイルに出力できます。

1. **メタ文字**

[]がいずれか一文字のリストを表すことは前ページで示したとおりです。メタ文字は

　^$.[]\*+?|()

等があります。それでは順に見ていきましょう。

1. ＾(caret), $ (dollar)

|  |  |
| --- | --- |
| ^ 行頭、　$ 行末 | |
| 正規表現 | ^abc　…　行頭がabcで始まる行  abc$　…　行末がabcで終わる行 |
| Match | Abcはアルファベットのはじめ　　　　←　^abcがマッチ  日本人はabcの歌は苦手　　　　　　　マッチしない  始まりはいつもabc　　　　　　　　　←　abc$がマッチ |

(2)　 . (period, dot)

|  |  |
| --- | --- |
| . 改行以外の任意の一文字 | |
| 正規表現 | a.c　…　aとcの間に何か一文字あるもの |
| Match | abc　abbd axc　ac a c  a(改行)c |

(3)　 \* (asterisk)

|  |  |
| --- | --- |
| \* 直前の文字の0回以上の繰り返し | |
| 正規表現 | ab\*c　…　aとcの間に0回以上のbの繰り返しがあるもの |
| Match | abc abbc abbcd abbbbbc ac  abd abbdc abddc |

(4)　 + (plus)

|  |  |
| --- | --- |
| + 直前の文字の1回以上の繰り返し | |
| 正規表現 | ab+c　…　aとcの間に1回以上のbの繰り返しがあるもの |
| Match | abc abbc abbcd abbbbbc ac  abd abbdc abddc |

(5)　 ? (question)

|  |  |
| --- | --- |
| ? 直前の文字が0または1回 | |
| 正規表現 | ab?c　…　aとcの間にbが一回かまたはないもの |
| Match | abc abbc abd ac abbdc |

(6)　 {} (curly bracket)

|  |  |
| --- | --- |
| {m,n} 繰り返しの回数指定。直前の文字がm回以上n回以下 | |
| 正規表現 | ab{1,3}c　…　aとcの間にbが１回から3回あるもの  m,nの指定がどちらか一つだけの場合次のような意味になります。  ab{2}c … aとcの間にbが2回だけあるもの  ab{,2}c … aとcの間にbが2回以上あるもの  ab{2,}c … aとcの間にbが2回以下のもの |
| Match | abc ac abbc abbbc abbbbbc |

(7)　 [] (squared bracket)

|  |  |
| --- | --- |
| [] リスト内のいずれか一文字。0-9、a-zのように範囲で指定することも可。 | |
| 正規表現 | a{xyz}c　…　aとcの間にx,y,zいずれか一文字あるもの |
| Match | axc abc ayc azc axyzc |
| 正規表現 | a[0-9]c　…　aとcの間に0～9のいずれか一文字あるもの |
| Match | a0c　abc a1c　a00c a9c a12c |

(8)　 () (parenthesis)

|  |  |
| --- | --- |
| () グループ化。複数の文字を塊として一文字と同じように指定する。 | |
| 正規表現 | a(bb)+c　…　aとcの間にbbが一回以上の繰り返があるもの |
| Match | abbc abbbbc abbbc abbbbbbc abbbbbc |
| 正規表現 | a[(xx)(yy)]c　…　aとcの間に「()「x」「y」「)」のいずれか |
| Match | axc axxc ayc ayyc a(c a)c  注）この例の通りグループ指定したつもりができていません。[]の中ではメタ機能が消えて普通の文字として扱われるので注意が必要です。 |

(9)　 \ (back slash:日本語環境では￥マークだけど英語圏では＼)

|  |  |
| --- | --- |
| \　エスケープ。メタ文字のメタ機能を除去して通常文字として扱う。 | |
| 正規表現 | \^abc　…　「^abc」という文字列  abc\$　…　「abc$」という文字列 |
| Match | ^abc abc$  abcが行頭、行末にあってもマッチしません |

(10)　 | (vertical bar)

|  |  |
| --- | --- |
| | Or条件の指定。複数条件をOrで連結する。 | |
| 正規表現 | a((xx)|(yy))c　… aとcの間にxxまたはyyのあるもの |
| Match | axxc ayyc |
| No Match | abbc axyc |
| 正規表現 | (SELECT|FROM|WHERE)  SELECTまたはFROMまたはWHEREと一致する部分 |
| Match | select  col1, col2  from  tbl  where  col1=1 |

**4．その他のメタ表現**

　空白や改行、空白や改行で区切られたワード等を表すメタ表現もあります。

(1)　 \s , \S (Space)

|  |  |
| --- | --- |
| \sは空白一文字、\Sは空白以外の一文字。 | |
| 正規表現 | ^s\*--　…　行の先頭が空白を含んで—で始まるもの |
| Match | --ここはコメント行です  -- ここもコメントです  中間の--はコメントになりません |
| 正規表現 | sub[\s\S]\*?end sub …　sub~～ end subでくくられた部分 |
| 正規表現 | Public Sub a2CreateFolder(Target As String)  With CreateObject("Scripting.FileSystemObject")  If Not .FolderExists(Target) Then .CreateFolder Target  End With  End Sub |

(2)　 \w , \W (word)

|  |  |
| --- | --- |
| \wはANKとアンダーバーのいずれか一文字、\Wはそれ以外の一文字 | |
| 正規表現 | [\w-\.]+@[\w-\.]+[A-Za-z]+　…　メールアドレスの簡易パターン |
| Match | name:Taro Yamato  mailto:Taro\_Yamato@dummy.japan.gr.jp  https://www.kantei.go.jp/ |

(3)　 \d , \D (decimal)

|  |  |
| --- | --- |
| \dは数字一文字、\Dは数字以外の一文字 | |
| 正規表現 | \d{2,4} …　数字2桁から4桁 |
| Match | 00　01　a0 0044　0a0b03 |
| 正規表現 | \D\d{3}　…　頭に数字以外の文字がある数値3桁 |
| Match | 〒141-0031 4444 （03）5114-1010　3333 |

**5. 最短マッチング**

次の例のように、文字項目はクォートで括られてその中にもカンマがあるようなCSVファイルをクオートで切り出すことを考えてみます。

|  |
| --- |
| "aaa", "bb,cc", "ddd", 1, 2, 3 |

ダブルクオートで始まり間に任意の文字があってクオートで終わる場合の正規表現は次の通りです。

|  |  |
| --- | --- |
| ダブルクオートで始まって任意の文字で繰り返してダブルクオートで終わるもの | |
| 正規表現 | ".\*" |
| Match | "aaa", "bb,cc", "ddd", 1, 2, 3 |

最初のクオートから最後のクオートまでがマッチしてしまいました。

正規表現における\*や+の量指定子は「貪欲なマッチング（greedy matching）」を行う仕様となっており、可能な限り最長なマッチングを行ってしまいます。これを避け、最短一致でマッチングさせるには次のように？を付けて指定します。

|  |  |
| --- | --- |
| ダブルクオートくくりの最短マッチ | |
| 正規表現 | ".\*?" |
| Match | "aaa", "bb,cc", "ddd", 1, 2, 3 |

これで無事に切り出すことができるようになりました。

**6. Look（先読み、後読み）**

　郵便番号の正規パターンを考えてみましょう。

|  |  |
| --- | --- |
| 数字3桁 "-" 数字4桁 | |
| 正規表現 | \d{3}-\d{4} |
| Match | 〒100-0001  電話番号 03-1111-1111 |

　これでは電話番号の一部もマッチしてしまいます。電話番号はマッチさせたくないので、頭に数値以外の文字のついた数値で指定します。

|  |  |
| --- | --- |
| 数字以外の一文字 数字3桁 "-" 数字4桁 | |
| 正規表現 | ￥D\d{3}-\d{4} |
| Match | 〒100-0001  電話番号 03-1111-1111 |

電話番号はマッチしなくなりましたが、今度は〒も含まれてしまいました。

こういうときに不要な部分を除外するのがLookの機能です。Lookで指定するパターンはマッチ結果から除かれます。Lookには指定の文字が**直前にある**もの／**直前にない**もの、**直後にある**もの／**直後にないもの**という４種類の指定方法があります。それぞれAhead/Behind × Positive/Negativeといい、下表のようになります。

|  |  |  |  |
| --- | --- | --- | --- |
| Look | | Psitive | Negative |
| Look ahead | 正規表現 | (?=*パターン*)  … **直後**にパターンの*ある*もの | (?!*パターン*)  … **直後**にパターンの**ない**もの |
| 使い方 | find(?=aaa) | find(?!aaa) |
| Match | findaaa findbbb  aaafind bbbfind find | findaaa findbbb  aaafind bbbfind find |
| Look behind  （VBAではエラー） | 正規表現 | (?<=*パターン*)  … **直前**にパターンの*ある*もの | (?<!*パターン*)  … **直前**にパターンの**ない**もの |
| 使い方 | find(?<=aaa) | find(?<=!aaa) |
| Match | findaaa findbbb  aaafind bbbfind find | findaaa findbbb  aaafind bbbfind find |

Lookを使って頭に郵便（数値以外）がある3桁-4桁の数値の正規表現は以下の通りです。

|  |  |
| --- | --- |
| 数字3桁 "-" 数字4桁（先頭に数字以外があるもの） | |
| 正規表現 | (?<=\D)\d{3}-\d{4} |
| Match | 〒100-0001  電話番号 03-1111-1111 |

これで無事記号を取り除いた番号部分のみが取り出せました。

本来これでOKなのですが、VBAではLook behindはエラーになるので残念ながら使えません。Lookや量指定子のマッチングは内部で計算量の指数的な増加を起こすことがあり、ちょっとしたマッチングでもフリーズしたように結果が返ってこなくなることがあります。特にVBAではLookBehindは使えずエラーになります。このような場合は別の正規表現で代用するか、または外部プログラムで補完する必要があります。

上記の例では2番目の郵便番号を含めた形で抽出しておき、Mactesオブジェクトから記号を消して番号のみを取得するといった処理を加えるとよいでしょう。

**7.　VBAの正規表現**

　ExcelやAccessのVBAでも正規表現は使えます。

　正規パターンは正規表現Helperで確認できますが、パターンを使う場合はプログラムでコードを記述する必要があります。ここではVBAのコードで正規表現を使う方法を示します。

(1) RegEXPオブジェクト(RegularExpression)

　正規表現はRegEXPオブジェクトを介して使います。RegEXPオブジェクトに正規パターンやマッチングオプションをセットしてメソッドを呼び出し、文字列を処理します。コード例は下記のとおりです。

Public Function a2RegSplit(Source As String \_

, Optional Pattern As String = "\S+" \_

, Optional IgnoreCase As Boolean = True \_

, Optional Global As Boolean = True \_

) As Object

Dim RE As Object

Set RE = CreateObject("VBScript.RegExp")

RE.Pattern = Pattern

RE.IgnoreCase = IgnoreCase

RE.Global = Global

Set a2RegSplit = RE.Execute(Source)

Set RE = Nothing

End Function

← オブジェクト生成

← 正規パターンを設定

← 大文字小文字の区別

← 繰り返し有無

← メソッド呼び出し

(2) RegEXPオブジェクトのメソッド

|  |  |
| --- | --- |
| メソッド | 機能 |
| RE.Test TestString | TestString中にマッチした部分があるかどうか判定  Instr関数と似たような使い方 |
| RE.Replace Src, RepStr | Src中のマッチした部分をRepStrで置換して返す |
| RE.Execute Src | Src中のマッチした部分をMatchesコレクションで返す |

MatchesコレクションはMatchオブジェクトの集合で配列のようなものです。

(3)Matchオブジェクトのプロパティ

|  |  |
| --- | --- |
| プロパティ | 機能 |
| Value | パターンにマッチした文字列 |
| Lengthe Src, RepStr | パターンにマッチした文字列の文字列長 |
| FirstIndex | Src中のマッチした部分の文字位置 |

**8. 正規表現用関数**

　文字列の置換など、よく使う機能を関数化して正規表現Helperの標準モジュールに入れておきました。コピペすればExcelでも使えます。

(1) 文字列を正規表現で置換

|  |  |  |  |
| --- | --- | --- | --- |
| 関数名 | a2RegReplace | | |
| 機能 | 正規表現で文字列を置換する | | |
| 書式 | Str = a2RegReplace(Source, Pattern, Replace,IgnoreCase) | | |
| 引数 | 名前 | 型 | 内容 |
| Source | String | ソーステキスト |
| Pattern | String | 正規パターン |
| Replace | String | 置換したい文字列 |
| IgnoreCase | Boolean | 大文字小文字の区別。省略可。デフォルトはTrue=区別しない。 |
| 戻値 | | String | 置換された文字列 |
| 【使用例】  　連続した空白を一つにする。  　debug.print a2RegReplace("abc def ghi","\s+", " ")  abc def ghi | | | |

(2) 正規表現で文字列を分割

|  |  |  |  |
| --- | --- | --- | --- |
| 関数名 | a2RegSplit | | |
| 機能 | 正規表現で文字列を分割する | | |
| 書式 | Set Match = a2RegSplit(Source, Pattern, IgnoreCase) | | |
| 引数 | 名前 | 型 | 内容 |
| Source | String | ソーステキスト |
| Pattern | String | 正規パターン |
| IgnoreCase | Boolean | 大文字小文字の区別。省略可。デフォルトはTrue=区別しない。 |
| 戻値 | | Matches | 分割された部分のMatchオブジェクトのコレクション |
| 【使用例】  　クォートで区切られた部分の2番目を表示  　debug.print a2regsplit("'aaa','bbb, c', 'cc','dd'","'.\*?'")(1)  'bbb,c' | | | |

　(3) 正規表現で行を抽出

|  |  |  |  |
| --- | --- | --- | --- |
| 関数名 | a2grep　(a2grepV) | | |
| 機能 | 正規表現を含む行だけを抽出（a2grepVは含まない行だけを抽出） | | |
| 書式 | Rows = a2grep(Source, Pattern, IgnoreCase) | | |
| 引数 | 名前 | 型 | 内容 |
| Source | String | ソーステキスト |
| Pattern | String | 正規パターン |
| IgnoreCase | Boolean | 大文字小文字の区別。省略可。デフォルトはTrue=区別しない。 |
| 戻値 | | String | 抽出行だけのテキスト |
| 【使用例】  Sourceの空行以外の行を表示する  　debug.print a2grepV(Source, "^\s+$") | | | |

(4) 正規表現にマッチした最初のテキストを返す

|  |  |  |  |
| --- | --- | --- | --- |
| 関数名 | a2RegMatch | | |
| 機能 | 正規表現にマッチした最初の部分のテキストを返す | | |
| 書式 | FirstWord = a2RegMatch(Source, Pattern, IgnoreCase) | | |
| 引数 | 名前 | 型 | 内容 |
| Source | String | ソーステキスト |
| Pattern | String | 正規パターン |
| IgnoreCase | Boolean | 大文字小文字の区別。省略可。デフォルトはTrue=区別しない。 |
| 戻値 | | String | マッチした最初のテキスト部分 |
| 【使用例】  メールアドレスの＠より前の部分を表示する  　debug.print a2RegMatch("abcde@gmail.com","\W(?=@)")  abcde | | | |

9.　終わりに

本文の解説は以上です。すべて解説しようとすると一冊の本になるほど奥の深いものですが、実用的に使うだけなら本文の内容を理解していればまずは使い始められるでしょう。あとは使いながら覚えていけば十分です。

正規表現はsakuraエディタや秀丸エディタでは標準で使え、OracleのようにSQLのWhere条件で使えるRDBもあります。すでに十分使われてきた枯れた技術で、昨今のXMLやHTMLを使ったWEB開発においても強力な機能を発揮するので覚えておいて損のない技術です。

