# Expressions régulières sous tous les angles

Par Mathieu Gaouette

PROSPECTIVEMG

# Agenda

- L’origine
- Définir le langage
- Utilité
- Expressions régulières avec SAS
- Mises en garde

PROSPECTIVEMG

# Origines

- Concept introduit dans les années 1950s.
- Le Mathématicien Stephen Cole Kleene a formalisé la définition de « langage régulier ».
- Il existe nombreux langages réguliers mais les expressions régulières « Perl » est le plus répandu.

# Expression régulière

- ***En informatique, une expression régulière*** […] ***est une chaîne de caractères qui décrit****, selon une syntaxe précise,* ***un ensemble de chaînes de caractères possibles.*** – Wikipedia
- On utilise souvent le terme « regex » pour simplifier

PROSPECTIVEMG

# Exemple

- Qu’est-ce qui constitue un numéro de téléphone en Amérique du Nord?


- Est-ce 902-516-6643 un numéro valide ? ✓
- Est-ce (902) 516-6643 un numéro valide ? ✓
- Est-ce 902.516.6643 un numéro valide? ✓
- Est-ce 9 025 166 643 un numéro valide ? ✗

PROSPECTIVEMG

# Exemple - suite


Qu’est-ce qui constitue un numéro valide ?

- On va identifier les numéros en **vert** comme requis ET devant être en groupes (3,3,4)
- Nous allons permettre un unique caractère pour un code de pays (**optionnel**)
- On nécessite un minimum d’un caractère non-alphanumérique comme délimiteur entre ces groupes de nombres.

PROSPECTIVEMG

# Exemple - suite


- **\d** représente un chiffre (un unique caractère)
- **\d\d\d** représente 3 chiffres consécutifs (aussi valide: **\d{3}**)
- **\d{3}-\d{3}-\d{4}** est une expression régulière qui représente un numéro sous le format NNN-NNN-NNNN.

# Le langage: Les types de caractères

| Symbole | Caractère |
|---|---|
| . | N'importe quel caractère |
| \w | Un caractère alphanumérique ou _ (underscore) |
| \W | Un caractère qui n'est pas alphanumérique ou _ (NOT \w) |
| \s | Un espace |
| \S | Un caractère qui n'est pas un espace (NOT \s) |
| \d | Un chiffre |
| \D | Un caractère qui n'est pas un chiffre (NOT \d) |
| […] | Un caractère parmi la liste des caractères entre les crochets |
| [^…] | Un caractère ne figurant pas dans la liste des caractères entre les crochets |
| [a-z] | Un caractère parmi la séquence spécifiée |

# Le langage: Multiplicateurs

| Symbole | Multiplicateur |
|---|---|
| * | 0 ou plus |
| + | 1 ou plus |
| ? | 0 ou 1 |
| {*n*} | exactement *n* |
| {*n*,} | *n* ou plus |
| {*n*,*m*} | Entre *n* et *m* |

Exemple:

- \d?-?\d{3}-\d{3}-\d{4} représente “1-902-516-6643”
  mais également “-902-516-66435-184”

PROSPECTIVEMG

# Le langage: Indicateurs de position

| Symbole | Représente |
|---|---|
| ^ | Début de la chaine de caractères |
| $ | Fin de la chaine de caractères |
| \b | Début ou fin d'une chaine de \w (alphanumérique ou _) |
| \B | Position qui n'est pas un début ou fin de chaine de caractères (NOT \b) |

Pas un fan!

Exemple:

- ^\d?-?\d{3}-\d{3}-\d{4}$ représente "1-902-516-6643"
  mais ne représente pas "-902-516-66435-184"

*Place à amélioration*

PROSPECTIVEMG

# Le langage: Le héros des regex!

| Symbole | Role |
|---|---|
| ( ) | Identifie un groupe |
| \| | Délimite des choix à l'intérieur d'un groupe (clause OR) |

Exemple:

- ^(\d-|)\d{3}-\d{3}-\d{4}$

*Permet soit un chiffre et un tiret ou rien*

représente:

- "1-902-516-6643"
- "902-516-6643"

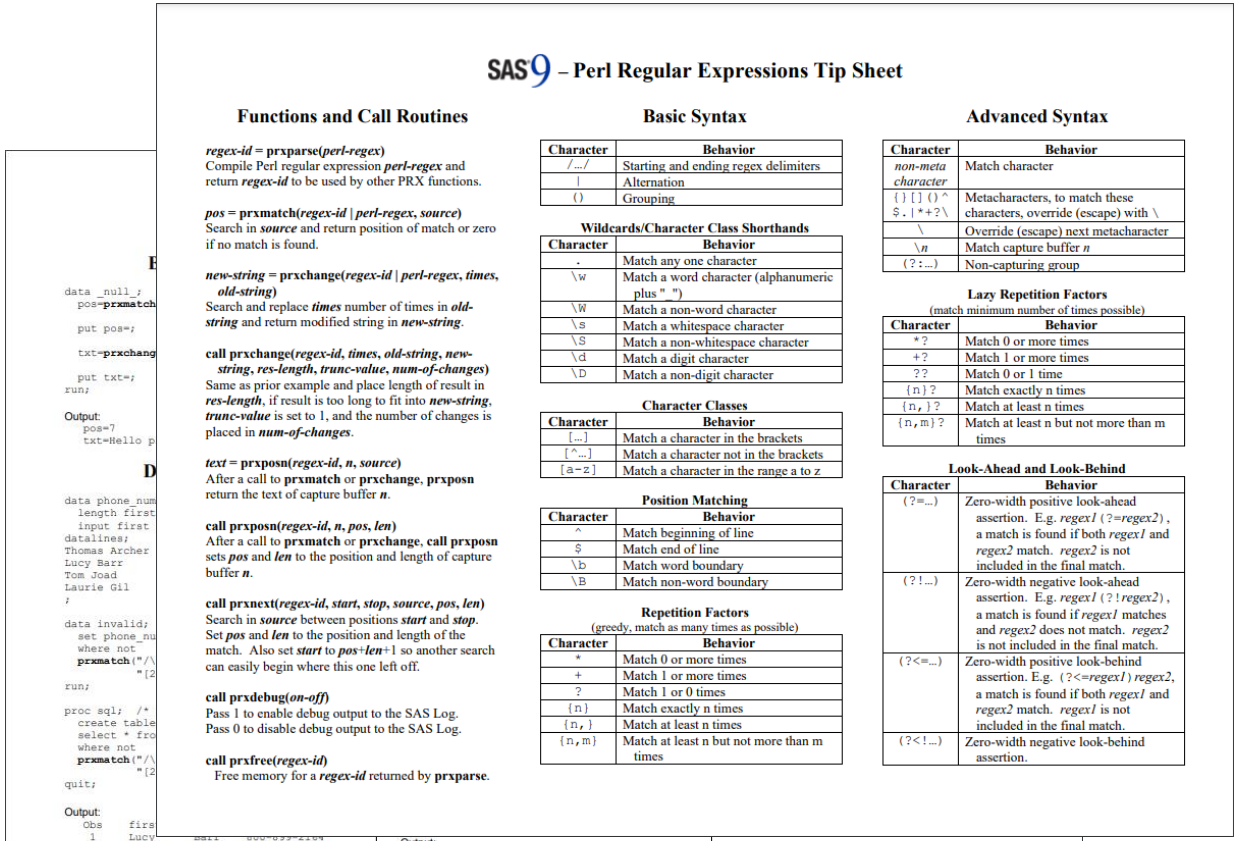
Mais ne représente pas:

- "-902-516-6643"

PROSPECTIVEMG

# La meilleure référence regex SAS

- SAS9 – Perl Regular Expressions Tip est disponible sur support.sas.com.
- Bon d'en avoir une copie ou un lien dans vos favoris.



SAS 9 – Perl Regular Expressions Tip Sheet

**Functions and Call Routines**

***regex-id* = prxparse(*perl-regex*)**
Compile Perl regular expression ***perl-regex*** and return ***regex-id*** to be used by other PRX functions.

***pos* = prxmatch(*regex-id* | *perl-regex*, *source*)**
Search in ***source*** and return position of match or zero if no match is found.

***new-string* = prxchange(*regex-id* | *perl-regex*, *times*, *old-string*)**
Search and replace ***times*** number of times in ***old-string*** and return modified string in ***new-string***.

**call prxchange(*regex-id*, *times*, *old-string*, *new-string*, *res-length*, *trunc-value*, *num-of-changes*)**
Same as prior example and place length of result in ***res-length***, if result is too long to fit into ***new-string***, ***trunc-value*** is set to 1, and the number of changes is placed in ***num-of-changes***.

***text* = prxposn(*regex-id*, *n*, *source*)**
After a call to **prxmatch** or **prxchange**, **prxposn** return the text of capture buffer ***n***.

**call prxposn(*regex-id*, *n*, *pos*, *len*)**
After a call to **prxmatch** or **prxchange**, **call prxposn** sets ***pos*** and ***len*** to the position and length of capture buffer ***n***.

**call prxnext(*regex-id*, *start*, *stop*, *source*, *pos*, *len*)**
Search in ***source*** between positions ***start*** and ***stop***. Set ***pos*** and ***len*** to the position and length of the match. Also set ***start*** to ***pos***+***len***+1 so another search can easily begin where this one left off.

**call prxdebug(*on-off*)**
Pass 1 to enable debug output to the SAS Log.
Pass 0 to disable debug output to the SAS Log.

**call prxfree(*regex-id*)**
Free memory for a ***regex-id*** returned by **prxparse**.

**Basic Syntax**

| Character | Behavior |
|---|---|
| /.../ | Starting and ending regex delimiters |
| \| | Alternation |
| () | Grouping |

**Wildcards/Character Class Shorthands**

| Character | Behavior |
|---|---|
| . | Match any one character |
| \w | Match a word character (alphanumeric plus "_") |
| \W | Match a non-word character |
| \s | Match a whitespace character |
| \S | Match a non-whitespace character |
| \d | Match a digit character |
| \D | Match a non-digit character |

**Character Classes**

| Character | Behavior |
|---|---|
| [...] | Match a character in the brackets |
| [^...] | Match a character not in the brackets |
| [a-z] | Match a character in the range a to z |

**Position Matching**

| Character | Behavior |
|---|---|
| ^ | Match beginning of line |
| $ | Match end of line |
| \b | Match word boundary |
| \B | Match non-word boundary |

**Repetition Factors**
(greedy, match as many times as possible)

| Character | Behavior |
|---|---|
| * | Match 0 or more times |
| + | Match 1 or more times |
| ? | Match 1 or 0 times |
| {n} | Match exactly n times |
| {n,} | Match at least n times |
| {n,m} | Match at least n but not more than m times |

**Advanced Syntax**

| Character | Behavior |
|---|---|
| *non-meta character* | Match character |
| { } [ ] ( ) ^ $ . \| * + ? \ | Metacharacters, to match these characters, override (escape) with \ |
| \ | Override (escape) next metacharacter |
| \n | Match capture buffer *n* |
| (?:...) | Non-capturing group |

**Lazy Repetition Factors**
(match minimum number of times possible)

| Character | Behavior |
|---|---|
| *? | Match 0 or more times |
| +? | Match 1 or more times |
| ?? | Match 0 or 1 time |
| {n}? | Match exactly n times |
| {n,}? | Match at least n times |
| {n,m}? | Match at least n but not more than m times |

**Look-Ahead and Look-Behind**

| Character | Behavior |
|---|---|
| (?=...) | Zero-width positive look-ahead assertion. E.g. *regex1* (?=*regex2*), a match is found if both *regex1* and *regex2* match. *regex2* is not included in the final match. |
| (?!...) | Zero-width negative look-ahead assertion. E.g. *regex1* (?!*regex2*), a match is found if *regex1* matches and *regex2* does not match. *regex2* is not included in the final match. |
| (?<=...) | Zero-width positive look-behind assertion. E.g. (?<=*regex1*)*regex2*, a match is found if both *regex1* and *regex2* match. *regex1* is not included in the final match. |
| (?<!...) | Zero-width negative look-behind assertion. |

For complete information refer to the Base SAS documentation at support.sas.com/base

PROSPECTIVEMG

# Utilité

- Il y a deux grandes catégories d'utilisation des regex

| | |
|---|---|
| **Recherche (avec extraction)** | Valider si un « pattern » est présent dans une chaine texte |
| **Substitution dans une chaine** | Replacer une ou plusieurs occurrences d'un « pattern » dans une chaine texte |

PROSPECTIVEMG

# Partons à la chasse aux chaines de texte

- Détecter une regex prends 1 ou deux étapes:
  - Compiler une regex (optionnel)
  - Tester une chaine de caractères pour une regex

- La compilation est effectuée via la fonction **prxparse**
- Le test d'une regex est effectué via la fonction **prxmatch**

PROSPECTIVEMG

# prxparse

prxparse(*<chaine contenant une regex>*)

- La chaine texte regex doit contenir des délimiteurs (normalement "/")
- La fonction retourne un nombre entier qui identifie l'expression régulière compilée

*Assurez-vous de la conserver dans une variable sur laquelle un « retain » a été effectué.*

Exemple:

```
re_phoneno = prxparse('/^(\d-|)\d{3}-\d{3}-\d{4}$/') ;
```

PROSPECTIVEMG

# prxmatch

prxmatch(*<regex (référence ou expression)>*,*<chaine de caractères>*)

- Si une chaine texte est utilisée comme regex alors elle sera compilée à la volée
- La fonction retourne la position de début du pattern trouvé (regex) dans la chaine de caractères (comme la fonction index)

Exemple:

```
phoneno_pos = prxparse(re_phoneno,char_phone) ;
phoneno_pos = prxparse('/^(\d-|)\d{3}-\d{3}-\d{4}$/', char_phone) ;
```

PROSPECTIVEMG

# prxposn

prxposn(<*regex (référence ou expression)*>,<# groupe>,<*chaine de caractères*>)

- Si une chaine texte est utilisée comme regex alors elle sera compilée à la volée
- Le numéro de groupe réfère aux éléments compris entre les parenthèses dans la regex
- La fonction retourne le contenu de la chaine qui est associé au groupe spécifié

Exemple:

```
country_cd = prxposn(re_phoneno, 1, char_phone) ;
country_cd = prxposn('/^(\d-|)\d{3}-\d{3}-\d{4}$/', 1, char_phone) ;
```

PROSPECTIVEMG

# Exemple - Préparation

```
data test_phone ;
    length phone_nb_raw $ 20 ;

    phone_nb_raw = '902-516-6643'    ; output ;
    phone_nb_raw = '1-902-516-6643' ; output ;
    phone_nb_raw = '(902) 516-6643' ; output ;
    phone_nb_raw = '902.516.6643'    ; output ;
    phone_nb_raw = '9 025 166 643'   ; output ;
run ;
```

PROSPECTIVEMG

# Exemple 1 – Le code

```
data test_phone ;
    if _n_ = 1 then do ;
        retain re_phoneno ;
        /* Décrit D-DDD-DDD-DDDD et DDD-DDD-DDDD */
        re_phoneno = prxparse('/^(\d-|)\d{3}-\d{3}-\d{4}\s*$/') ;
    end ;
    drop re_: ;

    set test_phone ;

    if prxmatch(re_phoneno,phone_nb_raw) > 0 then ind_valid_phonenb = 1 ;
    else                                          ind_valid_phonenb = 0 ;
run ;
```

Total rows: 5 Total columns: 2 Rows 1-5

| | phone_nb_raw | ind_valid_phonenb |
|---|---|---|
| 1 | 902-516-6643 | 1 |
| 2 | 1-902-516-6643 | 1 |
| 3 | (902) 516-6643 | 0 |
| 4 | 902.516.6643 | 0 |
| 5 | 9 025 166 643 | 0 |

PROSPECTIVEMG

# Exemple 2 – Le code

```
data test_phone ;
    length phone_nb_raw $ 20 phone1 phone2 $ 3 phone3 $ 4 ;
    if _n_ = 1 then do ;
        retain re_phoneno ;
        /* Décrit D-DDD-DDD-DDDD et DDD-DDD-DDDD */
        re_phoneno = prxparse('/^(\d-|)(\d{3})-(\d{3})-(\d{4})\s*$/') ;  /*  \s* ajouté */
    end ;
    drop re_: ;

    set test_phone ;

    if prxmatch(re_phoneno,phone_nb_raw) > 0 then do ;
        phone1 = prxposn(re_phoneno,2,phone_nb_raw) ;
        phone2 = prxposn(re_phoneno,3,phone_nb_raw) ;
        phone3 = prxposn(re_phoneno,4,phone_nb_raw) ;
    end ;
    else call missing(phone1,phone2,phone3) ;
run ;
```

PROSPECTIVEMG

# Exemple 2 – Les résultats

Total rows: 5  Total columns: 4  Rows 1-5

| | phone_nb_raw | phone1 | phone2 | phone3 |
|---|---|---|---|---|
| 1 | 902-516-6643 | 902 | 516 | 6643 |
| 2 | 1-902-516-6643 | 902 | 516 | 6643 |
| 3 | (902) 516-6643 | | | |
| 4 | 902.516.6643 | | | |
| 5 | 9 025 166 643 | | | |

PROSPECTIVEMG

# Substitution

- Effectuer de la substitution avec les regex nécessite une syntaxe différente:

  s/**<regex>**/**<Remplacement>**/

  Tout ce qui est ici est remplacé par

- Pour conserver des composantes de la **source**, on peut utiliser les groupes (parenthèses) et y référer en **sortie** avec $1 (1er groupe), $2 (2ième groupe), …

PROSPECTIVEMG

# Substitution en préservant des composantes

Exemple:

| Champ source Nom | Champ Nom désiré |
|---|---|
| Doctor Johnson, William | William Johnson |
| Cooper, Brenda | Brenda Cooper |
| Nurse White, Betty | Betty White |

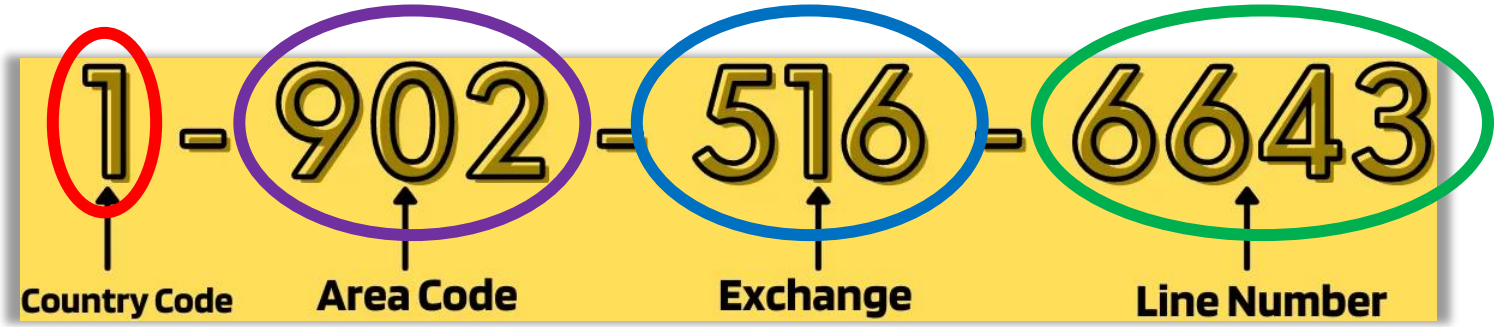
Regex de substitution

s/^(Doctor|Nurse|)\s*(\w+)\s*,\s*(\w+)\s*/$3 $2/

Groupe 1 Groupe 2 Groupe 3

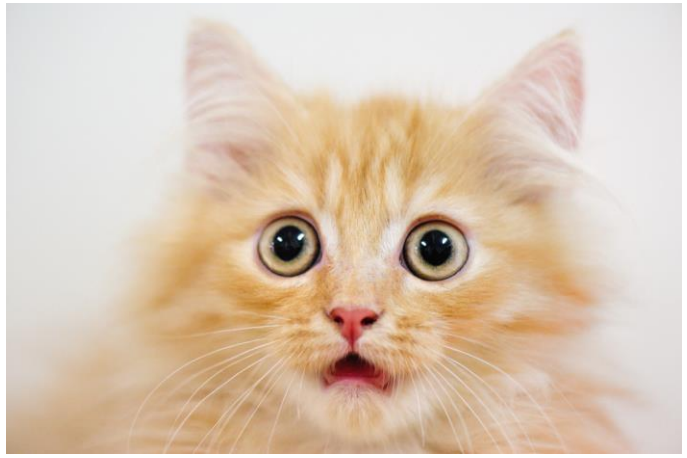
PROSPECTIVEMG

# Substitution de donnée




Regex de substitution

s/^(\d[\s\-]|\(?)(\d{3})[\s\-\)\.]+(\d{3})[\s\-\.]+(\d{4})\s*$/$2 $3 $4/



PROSPECTIVEMG

# prxchange

prxchange(*<regex>*, *<# remplacements>*, *<chaine de caractères>*)

- Si une chaine texte est utilisée comme regex alors elle sera compilée à la volée
- Sert à limiter le nombre de remplacements.  On peut utiliser “-1” pour ne pas fixer de limite
- La fonction retourne le résultat de la substitution

Exemple:

```
phoneno_std = prxchange(re_phonedlm,-1,char_phone) ;
phoneno_std = prxchange(<actual regex>,-1, char_phone) ;
```

PROSPECTIVEMG

# Exemple 3 – Le code

```
data test_phone ;
    length phone_nb_raw phone_std $ 20 ;
    if _n_ = 1 then do ;
        retain re_phoneno_std ;
        re_phoneno_std =
           prxparse('s/^(\d[\s\-]|\(?)(\d{3})[\s\-\)\.]+(\d{3})[\s\-\.]+(\d{4})\s*$/$2 $3 $4/') ;

    end ;
    drop re_: ;
    set test_phone ;

    phone_std = prxchange(re_phoneno_std,-1,phone_nb_raw) ;
run ;
```

PROSPECTIVEMG

# Exemple 3 – Les résultats

Total rows: 5  Total columns: 2  Rows 1-5

| | phone_nb_raw | phone_std |
|---|---|---|
| 1 | 902-516-6643 | 902 516 6643 |
| 2 | 1-902-516-6643 | 902 516 6643 |
| 3 | (902) 516-6643 | 902 516 6643 |
| 4 | 902.516.6643 | 902 516 6643 |
| 5 | 9 025 166 643 | 9 025 166 643 |

PROSPECTIVEMG

# Mises en garde

- La syntaxe du langage regex est plus complexe que le code SAS de base
- Mieux d'éviter quand une fonction existante fait le même travail (ex: scan, index, tranwrd, …)
- Mieux de tester la performance lorsque l'on veut utiliser à l'intérieur de proc sql sur des tables massives (Il pourrait y avoir un impact significatif)

# Ce que je n’avais pas le temps de couvrir

- Regex greedy vs non-greedy
- Exemples avec Proc SQL
- Exemples à l’intérieur de conditions « where » à l’intérieur d’autres étapes (print, sort, …)
- Analyse détaillée de la performance et comparaison avec d’autres fonctions (index, scan, …)

# Conclusion

*“Je suppose qu'il est tentant, si le seul outil dont vous disposez est un marteau, de tout traiter comme s'il s'agissait d'un clou.”*

*Abraham Maslow*

PROSPECTIVEMG

# Questions?

Commentaires?

PROSPECTIVEMG

# Références

- https://support.sas.com/rnd/base/datastep/perl_regexp/regexp-tip-sheet.pdf
- https://regex101.com/
- https://documentation.sas.com/api/collections/edmcdc/v_017/docsets/ds2ref/content/ds2ref.pdf (Chapitre 7)

PROSPECTIVEMG