Techniques

This document shows the practical application of many useful techniques (often in combination). For simplicity, the `(?i)` (case-insensitive) option is omitted; this option must be prefixed should case-insensitivity be required.

1. Search for duplicate words (including numbers)

Regex: `\b(\w+)\s+\1\b`
Test string: `the the these 123 123`

To make the back reference more apparent, the same processing can be specified using named capturing groups.
Regex: `\b(?<word>\w+)\s+\k<word>\b`

1.1 Search for duplicate words (excluding numbers)

Regex: `\b([\w-[\d]]+)\s+\1\b`
Test string: `the the these 123 123`

2. Check for misused "a" before vowel

Regex: `\s+a\s+[aeiou]`
Test string: `an antelope a ape`

Refine the regex to also check for misused "a" before an acronym beginning with S (at least 2 characters).
Regex: `\s+a\s+([aeiou]\w*|S[A-Z]{2,}\b)`
Test string: `At a SAT an SAT a sat`

Refine the regex to be case-insensitive.
Regex: `(?i)\s+a\s+([aeiou]\w*|(?-i)S[A-Z]{2,}\b)`
Test string: `At a SAT an SAT a sat`

2.1 Check for misused "an" before consonant

Regex: `\s+an\s+[b-z-[eiou]]`
Test string: `an alpha an beta`

3. Change “smart” quotes to “straight” quotes

Regex: `(“|”)`
Replace regex: `"`
Test string: `“The rain in Spain”`
Result: `"The rain in Spain"`

3.1 Change “straight” quotes to “smart” quotes

Regex: `"([^"]*?)"`
Replace regex: `“$1”`

Test string: `"The rain in Spain"`
Result: `“The rain in Spain”`

4. Match missing term

Not only can terms be matched, but also missing terms (with negative lookarounds).

Match missing preceding term (with negative lookbehind), note that the placement term (“word”) appears twice
Regex: `\sword(?<!pre\sword)`
Test string: `pre word nok word`

Match missing succeeding term (with negative lookahead)
Regex: `\sword\s(?!post)`
Test string: `word post word nok`

5. Repeated terms

Perform a loop to extract comma separated items from a list, e.g. “**`kg, cm, m`**”. A quantifier such as `+` or `{n}` applied to a subexpression can be used to perform a loop.

Regex: `"(\w+,\s)+\w+"`
String: `123 "kg, cm, m"`
Replace regex: `$&`
Result: `123 "kg, cm, m"`

The same technique can be used to process more complex lists, e.g. 4 (3+1) sets of numbers (2 digits) each separated with a period (`.`).

Regex: `(\d{2}\.){3}\d{2}`
Match string: `12.34.56.78`

6. Term variants

Search for the various forms of a term, such “login” (login, log in, logged in, etc.). This “simple” regex may find “false positives”.

Regex: `log(.)*in`
Test string: `login, log in, logged  in, log the name in the heading`

Refined regex: `log[\w]*?\s*in`
Test string: `login, log in, logged  in, log the name in the heading`

7. Fully numeric field

Check for 3-character field that is not fully numeric. This uses negative lookback. The positive assertion (=condition satisfied), “field not fully numeric", is marked.

Find regex: `\w{3}(?<!\d{3})`
Test string: `abc 123 a12`

8. Paired delimiters

Task: Check for embedded paired delimiters. The regex matches a string delimited by `"` or `'` that can contain the same embedded paired delimiter, `""` or `''`. For simplicity, the matched string can contain only word characters or whitespaces. The shading shows the captured delimiter (used as back reference).

Find regex (64 steps): `("|')[\w\s]*("|')`
Test string: `Find "Page 1"Page 2" reference`
Compare this with the generalised regex (100 steps): `("|').*?("|')`
Note: These regexes also match unpaired delimiters: `Find "Page 1'Page 2" reference`
This problem of malformed strings is corrected with the regex below.

Find regex: `("|')[\w\s]*\1`
Test string: `Find "Page 1'Page 2" reference`

9. Maximum value

Although the {…} quantifier specifies the size of a numeric field (the number of digits), there is no way of directly specifying the maximum value of a numeric field. Such a check can be made by specifying which values are valid.

This example shows how to check that a field with maximum 3 digits is not greater than 255.

Regex: `\b([01]?\d\d?|2([0-4]\d|5[0-5]))\b`
Test string: `0 1 01 10 19 99 100 199 200 255 256 1555 888`

This “simple” regex has the disadvantage that it also matches numbers with leading zeros, for example, 00 and 001.

The refined regex with four alternatives does not match numbers with leading zeros (but allows 0):
`\b(0|[1-9]\d?|1\d{2}|2([0-4]\d|5[0-5]))\b`
Test string: `0 1 01 10 19 99 100 199 200 255 256 1555 888`

The scheme could be easily adapted to handle other number ranges.

10. Normalise number

Task: Remove leading zeros from a number, for example 001 → 1, but 0 remains unchanged. For simplicity, only integer numbers are considered. The **\b**…**\b** bound prevents embedded numbers from being matched (such as “A99”).

Regex: `\b0*(\d+)\b`
Test string: `002, 1234, 0, A99, (56)`
Replace string: `$1`
Result: `2, 1234, 0, A99, (56)`

Extend the task to handle integer numbers with an optional sign). **\B**…**\b** is required to match signed numbers within bounds.

Regex: `(\b|\B[+-])0*(\d+)\b`
Test string: `002, 1234, 0, -01, A99, (56), +00045`
Replace string: `$1$2`
Result: `2, 1234, 0, -1, A99, (56), +45`

11. Match number but not date

Some languages, such as German, can use the same delimiter (period) for numbers and dates (for example, 28.02.19 and 12.345.678,90). To differentiate between them, the following regex can be used to match just numbers.

Regex: `(^|\s)(\d{1,3}(\.\d{3})*(,\d+)?)(?=(\s|$))`
Test string: `1.234  12.345.678  28.02.19  1.234,56  78`

The trailing lookahead is required to match the presence of the bound, but not capture; because the same bound is required before the matched string, otherwise more than one separator would be required. To match terms within other bounds (such as parentheses), the *bounds may need to be extended*, for example,

`(^|[\s\b(])` and `(?=[\s\b)])`; the Ps and Pe Unicode category blocks for start and end, respectively, can be used to generalise the parenthesis blocks, `(^|[\s\b\p{Ps}])` and `(?=[\s\b\p{Pe}])`, respectively.

Note: The bounds in this case are unusually complex because of the special parsing required and to ensure that numbers are also matched at segment start/end.

12. Match currency term

The Unicode Sc category block provides the Unicode symbol for all currencies (for example, $, €, £, ¥). If, as usual, the minor currency symbol (for example, ¢ (cent) for $ and €) should not be matched, such minor currency symbols must be specified as subtraction class. If the ISO currency code (for example, USD (also USS and USN), EUR, GBP, JPY) should also be matched, the required codes must be specified explicitly.

Regex: `(USD|EUR|GBP|JPY|[\p{Sc}-[¢]])\s?\d{1,3}(,\d{3})*(\.\d{2})?`
Test string: `$ 12,345.67 ¢ 456 ¥ 1,234 EUR 123,456.78`

Considering "economy of accuracy" (a term that originally came from Land Surveying → maps), it may suffice to specify a generic 3-character currency code, although at the potential cost of "false positives".

Regex: `((?<cc>([A-Z]{3})|[\p{Sc}-[¢]]))\s?(?<val>\d{1,3}(,\d{3})*(\.\d{2})?)`
Test string: `USD12,345.67, $123, US 345, ¢123, XXX 12,345.67`
Replace string: `${cc} ${val}`
Result string: `USD 12,345.67, $ 123, US 345, ¢123, XXX 12,345.67`

Although at the cost of additional complexity, the regex can be extended to permit the "currency term" at the front or end of the string. The "`?x`" option permits multiline regexes.

Regex:
```
(?x)((USD|EUR|GBP|JPY|[\p{Sc}-[¢]])
\s?\d{1,3}(,\d{3})*(\.\d{2})?|
\d{1,3}(,\d{3})*(\.\d{2})?\s?(USD|EUR|GBP|JPY|¢))
```
Test string: `$ 12,345.67 ¢ 456 ¥ 1,234 EUR 123,456.78 901 ¢ 12,345 EUR`

12.1 Extended currency

The currency can also be extended with quantifiers, such as "Tsd" = thousand or extended currency term "TEUR = thousand EUR", that can be included in the Replace string. The same technique can be used for other quantifiers, such as "million".

Regex: `(((Euros?|EUR|€)\s+(?<val>\d+)\s+(Tsd|Tausend))|(?<val>\d+)\s+TEUR)`
Test string: `€ 123 Tausend Euro 2 Tsd 456 TEUR`
Replace string: `€ ${val},000`
Result string: `€ 123,000 € 2,000 € 456,000`

13. Reformat number

In many cases, the number formatting differs between the source and target languages, for example, the significance of periods and commas is reversed between English and German numbers (123,456.78 → 123.456,78). In this example, three capturing groups are used on the Replace string. Because only the content of each "number block" is required, each must be matched as separate capturing group; for clarity, each "number block" is written within parentheses.

Regex: `\b(\d{1,3})(,(\d{3}))(\.(\d+))\b`
Replace string: `$1.$3,$5`
Test string: `12,345.6`
Result string: `12.345,6`

Because the "Replace string" does not permit any flexibility, the "Regex" / "Replace string" pair must be appropriate for the matched data; if necessary, multiple "Regex" / "Replace string" pairs for all possibilities

must be executed.

Tip: To avoid programming multiple regexes to handle the various formats, it may be better to program a more general regex with optional fields, and then "clean up" the results, as shown below.

General regex: `\b(\d{1,3})(,(\d{3}))?(,(\d{3}))?(\.(\d+))?\b`
Replace string: `$1.$3.$5,$7`
Test string: `1,234,567.89 12,345.6 12`
Result string: `1.234.567,89  12.345.,6  12..,`

Whereby, the erroneous separators "`.,`" and "`..,`" can be cleaned up with the following Find regex (note the use of the "null capturing group" that is required to reference an "empty" value):
`(?<ho>(\d*\.\d+)*)(\.+(?<lo>,\d+)|\.+,(?<lo>))`
and Replace string:
`${ho}${lo}`
Test string: `1.234.567,89  12.345.,6  12..,`
Result string: `1.234.567,89  12.345,6  12`

Note: The "General regex" can be extended using the same scheme to handle additional leading "thousand blocks".

14. Reformat date

Task: Reformat a European format date (dd mmm yy) to an American format date (mmm dd, yy), where "mmm" can be the 3-character month abbreviation or the complete month name and "yy" can be the 2- or 4-digit year (only 19… and 20…). For simplicity, only the first two months are listed.

Regex: `(\d{1,2})\s(Jan\w*|Feb\w*),?\s((19|20)?\d{2})`
Test string: `6 January 2019`
Result string: `January 6, 2019`

Because this overly simple regex also matches false positives (invalid day number or month name), it may be desirable to further refine it, as shown in the regex below.

Refined regex: `(0?[1-9]|[12]\d|3[01])\s`
`(January|Jan|February|Feb),?\s((19|20)?\d{2})`

This regex permits (=optional) leading 0. The regex could be further refined to check for valid day number, for example, 30 February is invalid.

Status 07.12.2019