Capitulo 5

Expressiones regular

Nos usa *expressiones regular* a identificar partes de sequentias de characteres. Nos pote dicer que expressiones regular es un linguage intra un linguage.

In un expression regular cata littera accepta le littera correspondente in sequentia de characteres. Preter isto nos ha characteres special, con su significantias in tabella 3:

```
. ^ $ * + ? { } [ ] \ | ( )
```

Tabella 3. Characteres special de expressiones regular.

Characteres	Significantia
`.`	Qualcunque character.
`^`	Initio del texto. In un classe: negation.
`$`	Fin del texto.
`*`	Zero o plure apparentias.
`+`	Un o plure apparentias.
`?`	Zero o un apparentia.
`{m,n}`	A minime `m`, a maxime `n` apparentias.
`[...]`	Un classe de characteres.
`\`	Character a escappar.
`|`	Alternante (operator or).
`(...)`	Un gruppo.

Le expression regular `"e"` accepta le littera `e`. In tabella 4 nos presenta altere exemplos.

Tabella 4. Exemplos de expressiones regular.

Sequentia	Significantia
`"e"`	Le littera `e`.
`"[aeio]"`	Un de litteras `a`, `e`, `i`, `o`.
`"[^aei]"`	Omne altere characteres preter `a`, `e`, `i`.
`"e*"`	Le littera `e` zero o plure vices.
`"e+"`	Le littera `e` un o plure vices.
`"[aeio]+"`	Le litteras `a`, `e`, `i`, `o` un o plure vices, per exemplo: `"aeo"`, `"eei"`, `"i"`, `"iiiia"`, `"oaoao"`, ...
`"a{2,4}"`	Le sequentia `"aa"`, `"aaa"`, o `"aaaa"`.
`"un|duo"`	Le sequentia `"un"` o `"duo"`.

Le sequentias a repeter `*`, `+`, `?`, e `{m,n}` post un expression accepta le sequentia le plus longe possibile. Illes ha le variantes `*?`, `+?`, `??`, e `{m,n}?`, que accepta le sequentia le plus curte possibile (tabella 5).

Tabella 5. Apparentias le plus longe possibile e le plus curte possibile.

Characteres	Numero de apparentias
`*`	Zero o plure (le plus longe possibile).
`+`	Un o plure (le plus longe possibile).
`?`	Zero o un (le plus longe possibile).
`{m,n}`	De `m` a `n` (le plus longe possibile).
`*?`	Zero o plure (le plus curte possibile).
`+?`	Un o plure (le plus curte possibile).
`??`	Zero o un (le plus curte possibile).
`{m,n}?`	De `m` a `n` (le plus curte possibile).

Le characteres special perde su signification in un classe de characteres (marcate per parentheses quadrate `[...]`). Un classe de characteres `[...]` pote includer listas de characteres alternative e intervallos de characteres (marcate per un tracto de union `-`. Per exemplo, le intervallo `[a-z]` significa ”omne characteres de `a` a `z`”.

Le classe `[^...]` es le opposite a classe `[...]` (”omne altere characteres preter ...”). Alicun altere sequentias special es ancora in tabella 6.

Tabella 6. Sequentias special.

Sequentia	Significantia	Equal a
`\d`	Un character decimal.	`[0-9]`
`\D`	Opposite a previe.	`[^0-9]`
`\s`	Un spatio blanc.	`[ \t\n\r\f\v]`
`\s`	Opposite a previe.	`[^ \t\n\r\f\v]`
`\w`	Un character alphanumeric.	`[a-zA-Z0-9_]`
`\W`	Opposite a previe.	`[^a-zA-Z0-9_]`

Sequentias special preserva su signification in un classe de characteres, per exemplo, `[\d.]` es equal a `[0-9.]`.

Le r-formato de sequentias de characteres

Le *r-formato* es un formation de sequentias de characteres, specialmente utile con expressiones regular. Le littera *r* veni del parola anglese *raw* (crude). Le littera `r` o `R` ante le prime virguletta remove le signification special de barra oblique inverse (\). Per exemplo, pro cercar a un marca de tabulator `\t` per un expression regular, nos pote scriber `"\\t"` o `r"\t"`.

```
> print ("a\tb")
a   b
> print (r"a\tb")
a\tb
> r"\t"
'\\t'
```

Methodos del modulo `re` a cercar

Le modulo `re` ha alicun methodos a cercar un expression regular in un sequentia de characteres: `match`, `search`, `findall`, e `finditer`.

Le methodos `match` e `search` retorna un objecto `Match`.

```
import re
s = "Quatro stationes del anno, dece-duo menses in un anno."
> re.match (r"\w", s)
<re.Match object; span=(0, 1), match='Q'>
```

Hic nos cerca un sequentia special "\w" (un character alphanumeric). Le resultato de methodo match es le prime littera Q de nostre sequentia "Quatro stationes...".

Le differentia inter match e search es que le methodo match examina si le condition de expression regular es ver al comencio del sequentia, ma le methodo search cerca si le condition es ver alicubi in le sequentia.

```
> print (re.search (r" \w+", s))
<re.Match object; span=(6, 16), match=' stationes'>
> print (re.match (r" \w+", s))
None
```

Hic le expression regular " \w+" accepta un sequentia ubi un spatio " " precede un o plure characteres alphanumeric. Le prime occasion de un tal expression in sequentia s de exemplo anterior es in " stationes". Le occasion non es in comencio de s e talmente le methodo match non lo accepta.

Quando nos cerca le parentheses rotunde in le sequentia "(AB)(CD)", nos memora que le parentheses rotunde "(" e ")" mesme son characteres special in expressiones regular, e nos debe usar le character a escappar, barra oblique inverse "\". In le exemplo sequente le sequentia ".*" accepta le sequentia le plus longe possibile, id es, "AB)(CD". Le sequentia ".*?", al altere latere, accepta le sequentia le plus curte possibile, id es, "AB".

```
> re.match (r"\(.*\)", "(AB)(CD)")
<re.Match object; span=(0, 8), match='(AB)(CD)'>
> re.match (r"\(.*?\)", "(AB)(CD)")
<re.Match object; span=(0, 4), match='(AB)'>
```

Methodo findall retorna un lista de sequentias trovate.

Denove, le expression regular "\w+" accepta un character alphanumeric repetite un o plure vices (isto es ben proxime a nostre definition de *un parola* in lingua natural):

```
> re.findall (r"\w+", s)
['Quatro', 'stationes', 'del', 'anno', 'dece', 'duo', 'menses',
 'in', 'un', 'anno']
```

Le methodo `finditer` retorna un sequentia iterabile de objectos `Match`. Nos pote facer un lista de un sequentia iterabile per le function `list`.

```
> list (re.finditer (r"\w+", s))
[<re.Match object; span=(0, 6), match='Quatro'>,
 <re.Match object; span=(7, 16), match='stationes'>,
 ...
 <re.Match object; span=(49, 53), match='anno'>]
```

Objecto `Match`

Le methodo `match` de modulo `re` retorna un objecto `Match`. Un objecto `Match` ha per exemplo le methodos `group`, `start`, `end`, e `span`.

```
m = re.match (r"\w+",s)
> m
<re.Match object; span=(0, 6), match='Quatro'>
> m.group (), m.start (), m.end (), m.span ()
('Quatro', 0, 6, (0, 6))
```

Nos memora que le parentheses rotunde `(...)` in expressiones regular forma un gruppo. Le gruppo 0 sempre es le sequentia plen trovate. In le expression `"([0-9]+)-([0-9]+)-([0-9]+)"` nos cerca (in plus de expression plen) tres gruppos con un o plure numeros decimal 0–9. Illes deveni le gruppos `group(1)`, `group(2)`, e `group(3)`.

```
m = re.match (r"([0-9]+)-([0-9]+)-([0-9]+)", "2021-09-01")
> m.group (0), m.group (1), m.group (2), m.group (3)
('2021-09-01', '2021', '09', '01')
```

Le methodo `groups` (con un plural `-s`) retorna le gruppos desde gruppo 1, ma non le gruppo 0.

```
> m.groups ()
('2021', '09', '01')
```

Methodos sub e subn del modulo re

Le methodos del modulo re a cambiar le sequentias trovate es sub, e subn. Le methodo sub retorna le nove sequentia de characteres.

```
s = 'Quatro stationes del anno, dece-duo menses in un anno.'
> re.sub (r"\w", "x", s)
'xxxxxx xxxxxxxxx xxx xxxx, xxxx-xxx xxxxxx xx xx xxxx.'
```

Le methodo subn retorna un sequentia de duo elementos. Le prime elemento es le nove sequentia de characteres, e le secunde elemento es le numero de cambiamentos.

```
> re.subn (r"\w", "x", s)
('xxxxxx xxxxxxxxx xxx xxxx, xxxx-xxx xxxxxx xx xx xxxx.', 43)
```

In le methodos sub e subn le reimplaciamento pote esser un function.

```
def titlecase (match):
  txt = match.group()
  return txt[0].upper() + txt[1:].lower()

> re.sub (r"\w+", titlecase, s)
'Quatro Stationes Del Anno, Dece-Duo Menses In Un Anno.'
> re.subn (r"\w+", titlecase, s)
('Quatro Stationes Del Anno, Dece-Duo Menses In Un Anno.', 10)
```

Nomines de gruppos

Si nos besonia le gruppos in le reimplaciamento, nos pote usar le notation \N o le notation \g<N> pro gruppo N. Memora tamen, que \N como un sequentia es un character special, e talmente nos besonia le r-formato anque in le sequentia de reimplaciamento. Le altere characteres special non es characteres special in le reimplaciamento.

Per exemplo, pro trovar un parola que comencia per le littera a (un a-parola) e un parola que comencia per le littera m (un m-parola), nos pote usar expressiones regular "(a\w+)" e "(m\w+)". Hic illes deveni le gruppos \1 e \2.

```
s = 'Quatro stationes del anno, dece-duo menses in un anno.'
```

```
re.sub (r".*(a\w+).*(m\w+).*", r"1: \1 2: \2", s)
'1: anno 2: menses'
```

Le altere variante deveni a usar le nomines \g<1> e \g<2>:

```
> re.sub (r".*(a\w+).*(m\w+).*", r"1: \g<1> 2: \g<2>", s)
'1: anno 2: menses'
```

E si nos vole esser explicite, nos pote dar nostre gruppos le nomines `apar` (pro a-parolas) e `mpar` (pro m-parolas), per exemplo.

```
> re.sub (r".*(?P<apar>a\w+).*(?P<mpar>m\w+).*",
|   r"apar: \g<apar>, mpar: \g<mpar>", s)
'apar: anno, mpar: menses'
```

A vider avante

Si nos besonia le sequentia (e solmente le), que es ante un altere sequentia, nos debe *vider avante*. A vider avante, nos usa le expressiones regular `(?=...)`, `(?!...)`, `(?<=...)`, e `(?<!...)`, con su significantias in tabella 7:

Tabella 7. Sequentias a vider avante.

Sequentia	Signification
`(?=...)`	Expression regular ... postea.
`(?!...)`	Non un expression regular ... postea.
`(?<=...)`	Expression regular ... antea.
`(?<!...)`	Non un expression regular ... antea

Per exemplo, le parola ante le comma in le exemplo previe es `"anno"`.

```
s = 'Quatro stationes del anno, dece-duo menses in un anno.'
> re.search (r"\w+(?=,)", s)
<re.Match object; span=(21, 25), match='anno'>
```

Le prime parola sin un comma, spatio o un littera minuscule postea, es `"dece"`.

```
> re.search (r"\w+(?![, a-z])", s)
<re.Match object; span=(27, 31), match='dece'>
```

Le prime parola post un spatio es "stationes".

```
> re.search (r"(?<= )\w+", s)
<re.Match object; span=(7, 16), match='stationes'>
```

Le prime parola de tres litteras sin un spatio o littera minuscule antea e un non-littera postea, es "duo".

```
> re.search (r"(?<![ a-z])\w\w\w(?=\W)", s)
<re.Match object; span=(32, 35), match='duo'>
```

Capitulo 6

Programma de cambiamento

Nos nunc vole leger files texto e cambiar su contento. Prime nos usa le modulo `argparse` a leger le argumentos.

Argumentos per modulo `argparse`

Pro leger le argumentos de programma in le maniera plus versatile, nos usa le modulo `argparse`.

Nos crea un objecto `parser` de classe `ArgumentParser`.

Le objecto `parser` ha le methodo `add_argument` pro introducer le nomine, tipo e valor predifinite, per exemplo, a un argumento acceptabile per le programma.

Como nos ha un habitude a dar le nomines de files in terminal texto, le modulo `argparse` de nos le utensiles a comprender le significantias de illes.

Per exemplo, nos sovente usa un programma con argumentos de files input (`-i`) e output (`-o`):

```
python substitue.py -i input.txt -o output.txt
```

Nos pote leger le argumentos de `-i` e `-o`, quando nos usa le methodo `add_argument`.

```
import argparse
```

```
parser = argparse.ArgumentParser ()
parser.add_argument ("-i", "--input", default="input.txt")
parser.add_argument ("-o", "--output", default="output.txt")
parser.add_argument ("-s", "--substitue", default="substitue.txt")
parser.add_argument ("-sep", "--separator", default="\t")
parser.add_argument("--log", action="store_true")
```

Si nos lege le argumentos in variabile `args`, nos trova que nos nunc ha le nomines `args.input`, `args.output`, `args.substitue`, `args.separator`, e `args.log` a usar. Usualmente un argumento ha un forma plus curte e un forma plus longe. Le forma plus longe es lo, que deveni un nomine de attributo sub le objecto `args`.

Le clave `default` es le valor predefinite de un argumento. Un argumento per clave `action` de un valor `"store_true"` deveni un valor de veritate `True` si iste argumente es presente, e `False` alteremente.

```
args = parser.parse_args ()
> args.input
'input.txt'
> args.output
'output.txt'
> args.log
False
```

It Python un function predefinite `vars` da nos un dictionario de nomines e valores de cata attributo de un objecto:

```
> vars (args)
{'input': 'input.txt', 'output': 'output.txt',
 'substitue': 'substitue.txt', 'separator': '\t', 'log': False}
```

Modulo `tabulate`

Le modulo `tabulate` ha un function `tabulate`, que representa listas in formato de tabulas.

```
from tabulate import tabulate
```

```
> print (tabulate ([("cucumbres",9),("tomates",127)]))
---------  ---
cucumbres    9
tomates    127
---------  ---
```

Le function `tabulate` accepta un argumento de clave `headers`, que adde le titulos a columnas. Nos dice que nostre dictionario ha le nomine `d`. Su clave e su valores es in lista `d.items`. Nos imprime le representation `repr` de su valor. Le function `repr` es un function predefinite in Python.

```
d = vars (args)
lst = [(k,repr(v)) for k,v in d.items()]
h = ["Argumento","Valor"]
> print (tabulate (lst,headers=h))
Argumento    Valor
-----------  ---------------
input        'input.txt'
output       'output.txt'
substitue    'substitue.txt'
separator    '\t'
log          False
```

Le methodo `items` de un dictionario da un lista de sequentias con elementos in forma (*clave*,*valor*). In le exemplo nos usava iste lista in un comprehension. Le *clave* e le *valor* deveni le variabiles `k` e `v`. Talmente le lista `lst` deveni un lista de sequentias con elementos de clave `k` e le representation `repr` del valor `v`.

```
> lst = [(k,repr(v)) for k,v in d.items()]
> lst
[('input', "'input.txt'"), ('output', "'output.txt'"),
 ('substitue', "'substitue.txt'"), ('separator', "'\\t'"),
 ('log', 'False')]
```

A leger le substitutiones

Nos aperi un file g, que contine le substitutiones que nos vole facer a un altere file. Nos lege le lineas in variabile `xs` e claude le file.

```
g = open (args.substitue)
xs = g.readlines ()
g.close ()
```

Cata linea contine duo campos separate per un sequentia que nos da in argumento `separator`. Cata linea anque contine un character a cambiar le linea `\n`, que nos vole excluder del sequentia. Nos usa le methodo `strip` a isto. Post isto nos divide le sequentia per le variabile `args.separator` que es nostre argumento `separator`.

Si nos ha succedite a divider le sequntia in exacte duo partes, nos adde le elementos in le lista `subs`. Postea nos usara iste lista `subs` a facer le substitutiones.

```
subs = []
for x in xs:
  sb = x.strip ("\n").split (args.separator)
  if len (sb) == 2:
    subs.append (sb)
```

A leger le documento

Le nomine del documento es in variabile `args.input`. Nos aperi le documento e lege lo. Le methodo `read` lege le tote documento e retorna lo. Nos placia le contento in le variabile `text` que deveni un (longe) sequentia de characteres.

```
f = open (args.input)
text = f.read ()
f.close ()
```

A cambiar le contento

Durante que nos cambia le contento, nos vole anque calcular quando cambiamentos aperi. Pro isto nos usa un dictionario, in que cata clave ha le valor predefinite 0. Le function `defaultdict` in le modulo `collections` es exacte lo que nos besonia pro isto.

Nos nunc traversa cata substitution. Le methodo `subn` de modulo `re` prende le patrono a cambiar, le sequntia a cambiar e le texto original como argumentos. Le

methodo retorna le nove texto e le numero de cambiamentos. Nos usa le variabiles `text` e `fnd` pro illes. Nos adde le numero de cambiamentos in le dictionario `found`.

```
found = defaultdict (int) # default = 0
for pattern,repl in subs:
  text,fnd = re.subn (pattern,repl,text)
  found [(pattern,repl)] = found [(pattern,repl)] + fnd
```

A scriber le cambiamentos

Nos aperi un file `h` a scriber (`"w"`) le resultatos. Le nomine del file `h` es le valor de variabile `args.output`, que es le argumento `--output` de programma.

A scriber le texto a file nos usa le methodo `write` de file `h`.

```
h = open (args.output,"w")
h.write (text)
h.close ()
```

Si nos vole, nos pote imprimer un summario de cambiamentos sur schermo.

```
subs2 = []
for a,(k,v) in zip (subs,found.items()):
  subs2.append (a + [v])

if args.log:
  print (tabulate (subs2))
```

Cambiamentos

Hic es alicun cambiamentos que nos pote pensar:

```
gegenadastoot   genadestoot
'''\+   + '''
'''\+   + '''
 {(.*)}'''  ''' {\g<1>}
''' ([a-z]+)    ''' {\g<1>}
\n\n    \n
```

```
 -   ~
 -,  ~,
 -\.     ~.
 -   ~
 -,  ~,
 -\.     ~.
<br>
\nDe hic:    De hic:
'''
''
\n\+    \n
\n\n    \n
<u>
</u>
```

A cambiar un parola definite:

```
gegenadastoot   genadestoot
```

A cambiar un signo + al altere latere del `'''` (con o sin spatio, non visibile hic):

```
'''\+   + '''
'''\+   + '''
```

A cambiar le classe (per exemplo `{s}` o `{adj}`) al altere latere del `'''` (variationes differente):

```
 {(.*)}'''  ''' {\g<1>}
''' ([a-z]+)    ''' {\g<1>}
```

Cambiar – a un ~ (o in altere direction):

```
 -   ~
 -,  ~,
 -\.     ~.
 -   ~
 -,  ~,
 -\.     ~.
```

Deler un `<br>` (o adder):

```
<br>
```

Deler un cambiamento de linea \n ante "De hic:". Deler ''' e '' totalmente. Deler <u> e </u> totalmente.

```
\nDe hic:    De hic:
'''
''
<u>
</u>
```

Omne isto prende un tempore non plus que 0.4 secundas.

```
# python thoreau-substitue.py \
  -i dictionario-encyclopedic-2021-08-31.txt \
  -o output.txt --log
------------  -----------  -----
gegenadastoot  genadestoot      1
'''\+          + '''         9975
'''\+          + '''           93
{(.*)}'''      ''' {\g<1>}  23945
''' ([a-z]+)   ''' {\g<1>}  11414
\n\n           \n           42146
-              ~            38208
-,             ~,            9906
-\.            ~.            2921
-              ~              668
-,             ~,             211
-\.            ~.              62
<br>                         4210
\nDe hic:      De hic:       2676
'''                          89945
''                           6630
\n\+           \n           16237
\n\n           \n             547
<u>                           547
------------  -----------  -----
(0.398 s)
```

Exemplo, antea

```
'''+ bichromia {s}''' Impr. Impression in duo colores.

'''bicipital {adj}''' Anat. Relative al bicipite: tendon -.

'''bicipite {adj}''' Bicipital. De hic: bicipite-bicipital

'''bicipite {s}''' (lat. biceps, a duo testas) Musculo longe cuje
↪  un extremitate es dividite in duo corpores muscular distincte
↪  e que ha duo tendones de insertion a iste extremitate; in
↪  special le del bracios: brachial

'''+ bicocca {s}''' (it. bicocca, parve obra de fortification)
↪  Pej. Fam. Casa de mediocre apparentia, mal intertenite. Omne
↪  casa: cercar un -.

'''bicolor {adj}''' Que ha duo colores.

'''+ biconcave {adj}''' Optica que presenta duo facies concave
↪  opposite: lentes -.

'''+ biconvexe {adj}''' Optica que presenta duo facies convexe
↪  opposite: lentes -.

'''bicorne {adj}''' Relative a bicorno.

'''+ bicorno {s}''' (lat. bicornis, a duo cornos) Cappello de
↪  uniforma a duo punctas.

'''bicornute {adj}''' Que ha duo cornos: bestia -.
```

Exemplo, postea

```
bichromia {s}  Impr. Impression in duo colores.
bicipital {adj}  Anat. Relative al bicipite: tendon ~.
bicipite {adj}  Bicipital. De hic: bicipite-bicipital
```

bicipite {s} (lat. biceps, a duo testas) Musculo longe cuje un
↪ extremitate es dividite in duo corpores muscular distincte e
↪ que ha duo tendones de insertion a iste extremitate; in
↪ special le del bracios: brachial
bicocca {s} (it. bicocca, parve obra de fortification) Pej. Fam.
↪ Casa de mediocre apparentia, mal intertenite. Omne casa:
↪ cercar un ~.
bicolor {adj} Que ha duo colores.
biconcave {adj} Optica que presenta duo facies concave opposite:
↪ lentes ~.
biconvexe {adj} Optica que presenta duo facies convexe opposite:
↪ lentes ~.
bicorne {adj} Relative a bicorno.
bicorno {s} (lat. bicornis, a duo cornos) Cappello de uniforma a
↪ duo punctas.
bicornute {adj} Que ha duo cornos: bestia ~.