Fundamentals of Natural Language Processing

Patrick Paroubek

pap@limsi.fr

MIROR project

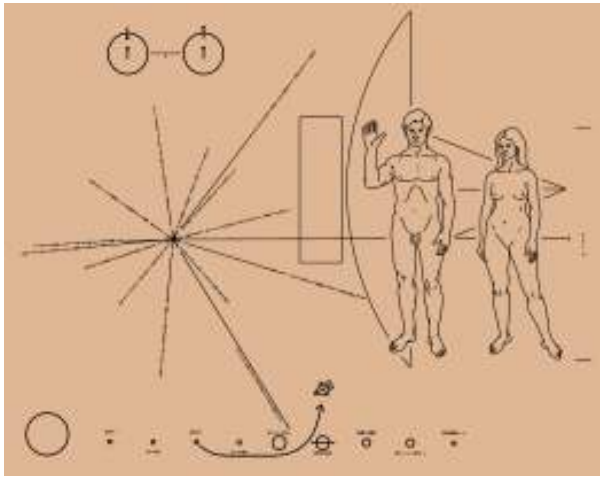
Thursday, Octorber 4th 2017 - Split

Horizon 2020 Research & Innov. programme Marie Sklodowska-Curie g.a. No 676207

Course Plan

1. About (Natural) Languages & Computers
2. Segmenting (regular expressions)
3. Part-Of-Speech Tagging (Treetagger)

About (Natural) Languages and Computers



Pioneer plaque

Dispilio (Greece) 5202 (123) BC

Cray2 computer (1979-1982)

Fortran punched card (1970)

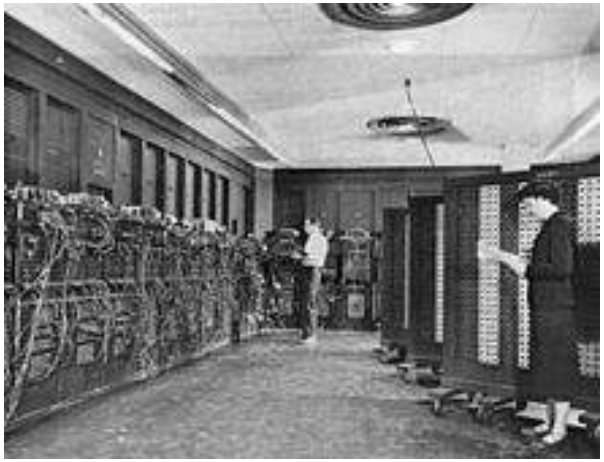
pictures source : wikipedia

Birth of a computer

ENIAC - 13/02/1946 - *Moore School of Elec. Engineering* UPenn

1

2



3

4

picture sources : 1,2,4 pap ; 3 wikipedia

About (Natural) Languages and Computers

Humans :

- Philosopher Ludwig Wittgenstein, Language Games, *"Philosophical Investigations"*(1953)
- *"The growth of language : Universal Grammar, experience, and principles of computation"*, Charles Yang a, Stephen Crain b, Robert C. Berwick c , Noam Chomsky d , Johan J. Bolhuis, Neuroscience and Biobehavioral Reviews, 2016 `http://dx.doi.org/10.1016/j.neubiorev.2016.12.023`

Computers :

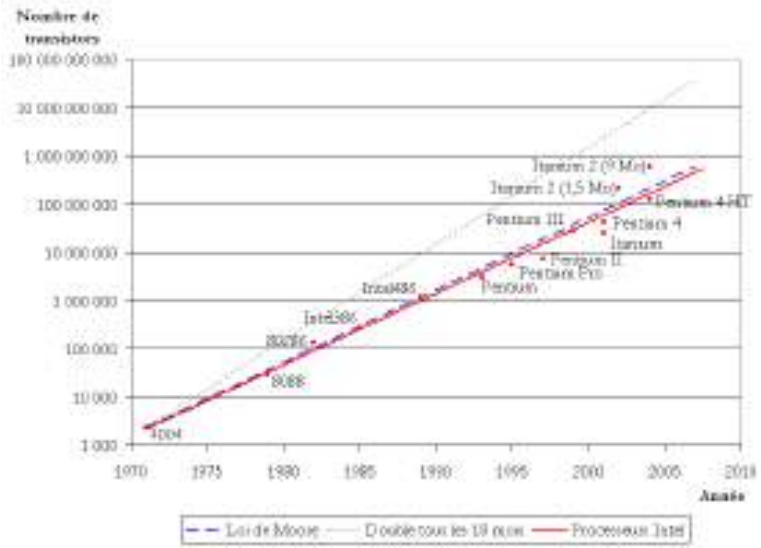
- Luc Steels and Michael Spranger, *"How experience of the body shapes language about space"*, IJCAI 2009, `https://www.ijcai.org/Proceedings/09/Papers/014.pdf`
- Luc Steels, *"Requirements for Computational Construction Grammars"*, The AAAI 2017 Spring Symposium on Computational Construction Grammar and Natural Language Understanding", Tech. Report SS-17-02, `https://aaai.org/ocs/index.php/SSS/SSS17/paper/download/15319/14542`

About (Natural) Languages and Computers

Computer Science is a “young” science.

- 1944 ENIAC, the first electronic programmable computer built at the Moore School of Electrical Engineering of University Pennsylvania (`http://www.seas.upenn.edu/about-seas/eniac/`)
- 1971 FTP (File Transfer Protocole), first specs.
- 1991 Gopher, WEB sites directory app.
- 1993 Mosaic NCSA, first WEB browser.

Moore’s Law :
computer power
doubles every 18 month



pictures source : wikipedia

Orders of magnitude

unit	# bits	description
bit	1	one
byte (b)	8	eight
kilobyte (Ko/Kb)	1024	one thousand
megabyte (Mo/Mb)	1024^2 = 1.048.576	˜ one million
gigabyte (Go/Gb)	1024^3 = 1.073.741.824	˜ one billion
terabyte (To/Tb)	1024^4 = 1.099.511.627.776	˜ one thousand billions
petabyte (Po/Pb)	1024^5 = 1.125.899.906.842.624	˜ one million of billions

1 cheap laptop ˜ 400 €

4Gb dyn. mem. + 250 Gb SDD disk + 1 Tb HD

Orders of magnitude

1 character	8 to 32 bits
1 word (in French)	5 to 6 characters on average
1 web page screen	150 to 300 words
average TV news lexicon	˜90.000 words
extensive literary lexicon (French)	˜ 540,000 word types

Segmenting (regular expressions)

Language is a linear sequence of sounds
It is represented as a linear sequence of characters
It is encoded as a linear sequence of bits

BUT

What's a word ?

Segmenting (regular expressions)

In NLP we process sequences of tokens .

A token is a subsequence of characters defined by deterministic criteria.

Segmenting (regular expressions)

« L'informatique, c'est compliqué »(*Computer science, it is complicated*)

character coding ISO-88591 (Latin1) :

87654321	0011	2233	4455	6677	8899	aabb	ccdd	eeff	0123456789abcdef
00000000 :	4c27	696e	666f	726d	6174	6971	7565	2063	L'informatique c
00000010 :	2765	7374	2063	6f6d	706c	6971	75e9	0a	'est compliqu..

character coding Unicode with UTF8 encoding :

87654321	0011	2233	4455	6677	8899	aabb	ccdd	eeff	0123456789abcdef
00000000 :	4c27	696e	666f	726d	6174	6971	7565	2063	L'informatique c
00000010 :	2765	7374	2063	6f6d	706c	6971	75c3	a90a	'est compliqu...

character coding Unicode with UTF16 encoding :

87654321	0011	2233	4455	6677	8899	aabb	ccdd	eeff	0123456789abcdef
00000000 :	6573	004c	0027	0069	006e	0066	006f	0072	es.L.'.i.n.f.o.r
00000010 :	006d	0061	0074	0069	0071	0075	0065	0020	.m.a.t.i.q.u.e. &
00000020 :	0063	0027	0065	0073	0074	0020	0063	006f	.c.'.e.s.t. & .c.o
00000030 :	006d	0070	006c	0069	0071	0075	00e9	000a	.m.p.l.i.q.u....

character coding Unicode with UTF32 encoding :

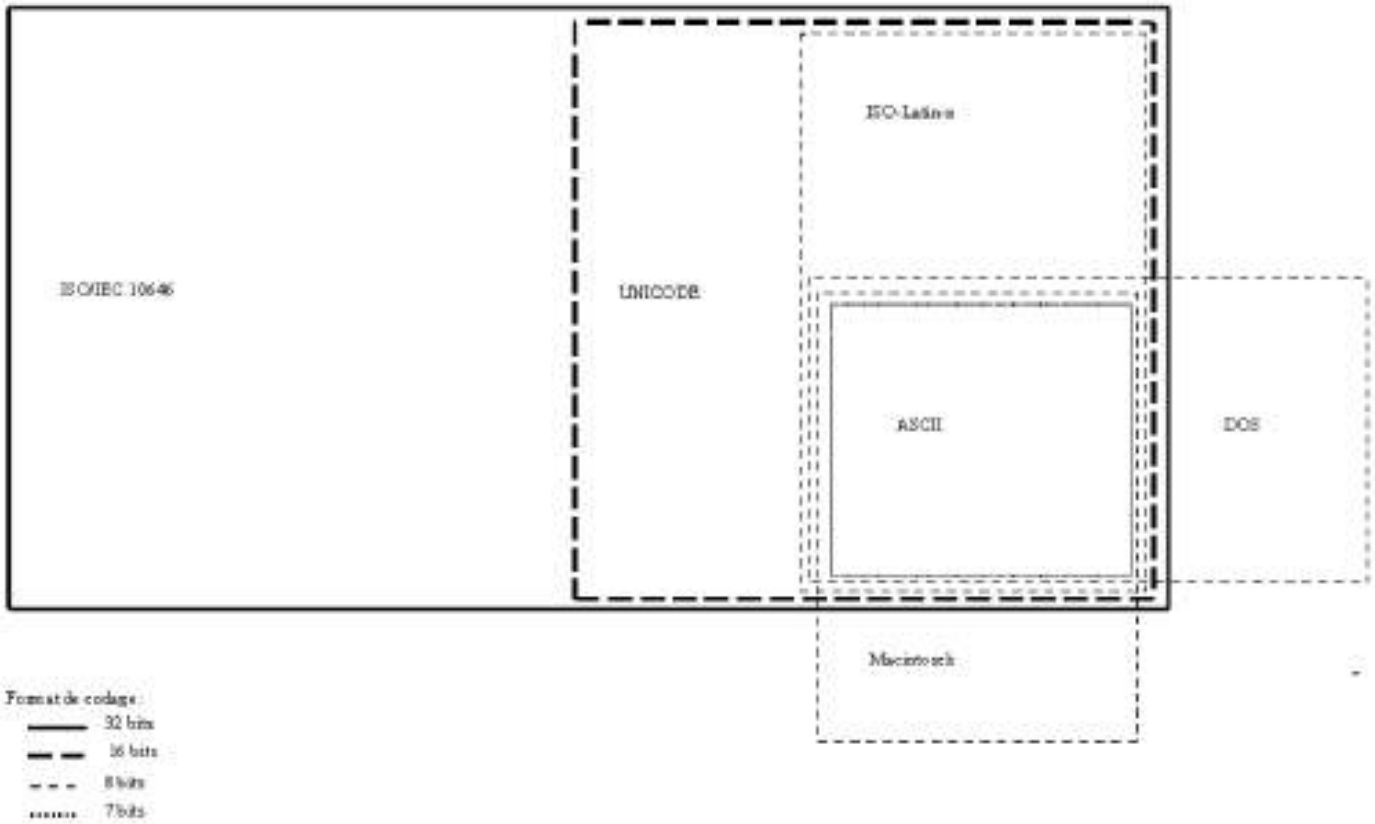
87654321	0011	2233	4455	6677	8899	aabb	ccdd	eeff	0123456789abcdef
00000000 :	fffe	0000	4c00	0000	2700	0000	6900	0000	....L...'...i...
00000010 :	6e00	0000	6600	0000	6f00	0000	7200	0000	n...f...o...r...
00000020 :	6d00	0000	6100	0000	7400	0000	6900	0000	m...a...t...i...
00000030 :	7100	0000	7500	0000	6500	0000	2000	0000	q...u...e... & ...
00000040 :	6300	0000	2700	0000	6500	0000	7300	0000	c...'...e...s...
00000050 :	7400	0000	2000	0000	6300	0000	6f00	0000	t... & ...c...o...
00000060 :	6d00	0000	7000	0000	6c00	0000	6900	0000	m...p...l...i...
00000070 :	7100	0000	7500	0000	e900	0000	0a00	0000	q...u.....

ASCII

- L'ASCII (American Standard Code for Information Interchange) created in 1963 in the US and published by l'USASI (United States of America Standards Institute) in 1968, then became he internationla norm (ISO-646).
- characters are encoded over 7 bits (128 characters possible)
- It is the only «universal» coding, i.e. included in all character encoding norms.

pictures source : https ://www.flickr.com/creativecommons/by-2.0/

Standards


ISO/IEC 10646
UNICODE
ISO-Latin-1
ASCII
DOS
Macintosh
Format de codage :
32 bits
16 bits
8 bits
7 bits

with Unix, Linux, Ubuntu etc., bash command `man ascii` :

```
The following table contains the 128 ASCII characters.

C program '\X' escapes are noted.

Oct   Dec   Hex   Char                        Oct   Dec   Hex   Char
------------------------------------------------------------------------
000   0     00    NUL '\0'                    100   64    40    @
001   1     01    SOH (start of heading)      101   65    41    A
002   2     02    STX (start of text)         102   66    42    B
003   3     03    ETX (end of text)           103   67    43    C
004   4     04    EOT (end of transmission)   104   68    44    D
005   5     05    ENQ (enquiry)               105   69    45    E
006   6     06    ACK (acknowledge)           106   70    46    F
007   7     07    BEL '\a' (bell)             107   71    47    G
010   8     08    BS  '\b' (backspace)        110   72    48    H
011   9     09    HT  '\t' (horizontal tab)   111   73    49    I
012   10    0A    LF  '\n' (new line)         112   74    4A    J
013   11    0B    VT  '\v' (vertical tab)     113   75    4B    K
014   12    0C    FF  '\f' (form feed)        114   76    4C    L
```

Character encoding converters

- recode `https://github.com/pinard/Recode/`
- iconv
  Ubuntu
  (`sudo apt-get install libghc-iconv-dev libghc-iconv-doc libghc-iconv-prof`),
  Windows `https://dbaportal.eu/2012/10/24/iconv-for-windows/`,
  Python API `https://pypi.org/project/iconv/`,
  GNU `https://www.gnu.org/software/libiconv/`
- Unicode text editor `https://en.wikipedia.org/wiki/Yudit`

Python 3 programing foretaste

A few examples presenting the main concepts of programming (variables, loops and functions) in Python 3...

Start python on the command line, create constants and variables :

```
tnm> python
Python 2.7.12 (default, Nov 19 2016, 06:48:10)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>33
33
>>> 'foo'
'foo'
>>> x=33
>>> x
33
>>> y=3.444
>>> y
3.444
>>> z='hello world'
>>> z
'hello world'
>>> print( z )
hello world
>>>
```

Always be sure to know how to undo what you just did !

1. undefine a variable
2. stop the python interpreter

```
tnm> python
Python 2.7.12 (default, Nov 19 2016, 06:48:10)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> x
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
>>> x = 3
>>> x
3
>>> del x
>>> x
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
>>> x = None    # always initialise variables!
>>> x
>>> quit()      # in case of emergency type CTRL-D or CTRL-C
tnm>
```

How to define a fonction ((name a list of instructions that computes a result and returns a result),

```
>>># défining a fonction, here the constant fonction +3
>>> def une_fonction_constante_sans_parametre( ):
...     return 3
...
>>># call the fonction
>>>...
>>> une_fonction_constante_sans_parametre( ):
3
>>># définiting a fonction with a parameter
>>> def une_fonction_avec_un_parametre( n ):
...     return( 2 * n )
...
>>># call the fonction with an argument
>>> une_fonction_avec_un_parametre( 3 )
6
>>> r = ((4 * une_fonction_avec_un_parametre( 3 )) / 100.0)
>>> r
>>> 0.24
```

How to know the type of an object

```
1018$ python
Python 2.7.12 (default, Nov 19 2016, 06:48:10)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> x = 3
>>> type( x )
<type 'int'>              # integer variable
>>> y = 4.55555
>>> type( y )
<type 'float'>            # floating point variable
>>> z = 'hello'
>>> type( z )
<type 'str'>              # character string variable
>>> def double( n ):
...       return( 2 * n )
...
>>> type( double )
<type 'function'>         # function table
```

Test the type of the value contained in a variable

```
>>> x = 3
>>> if( type( x ) == int ):
...     print( 'It\' is an integer!' )
... else:
...     print( 'It is not an integer!' )
...
It's an integer!
>>> y = 4.55555
>>> if( type( y ) == float ):
...     print( 'It\'s a float!' )
... else:
...     print( 'It is not a float!' )
...
It's a float
>>> if( callable( double ) ):
...     print( 'It\'s a function' )
... else:
...     print( 'It is not a function' )
...
It's a function
>>>
```

How to read from the keyboard

```
>>> x =  input()
'hello'
>>> x
'hello'
>>> y = input( 'You say? ' )
you say?? 'Hello'
>>> y
'Hellog'
>>>
```

Repeat a sequence of instructions whose size is a priori unknown (while loop)

```
$ python
Python 2.7.12 (default, Nov 19 2016, 06:48:10)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> while( True ):
...   print( 'Is this course ending soon ?' )
Is this course ending soon ?
Is this course ending soon ?
Is this course ending soon ?
Is this course ending soon ?
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
KeyboardInterrupt
>>> s = 'foo'
>>> while( len( s ) > 0 ):
...   print( s[0] )
...   s = s[1:]    # iterating on a sequence of chars until the end
...
f
o
o
>>>
```

To repeat a sequence of instruction whose size is known a priori (for loop)

```
>>> for i in range(0,3):
...     print( i )
...
0
1
2
>>> s = 'bar'
>>> for i in range( 0, len(s) ):
...   print( 'le ' + str( i ) + ' -ième caractère est ' + s[ i ] )
...
le 0 -ième caractère est b
le 1 -ième caractère est a
le 2 -ième caractère est r
>>>
```

Note the transformation of the integer valuer held in the variable *i* into a charcter string by means of the call *str*(*i*) to the *str* function.

List data structure

```
>>> l0 = []         # empyt list
>>> len( l0 )
0
>>> l0bis = list()
>>> l0bis
[]
>>> l3 = [ 'un', 'deux', 'trois' ]
>>> l3
['un', 'deux', 'trois']
>>> l4 = range( 0, 4 )
>>> l4
[0, 1, 2, 3]
>>> m = [ 0, 'un', 2, 'trois' ]  # you can mix items of different type inside the same list.
>>> m
[0, 'un', 2, 'trois']
>>>
```

To add an element to a list : *append*(), be carfule there is a “side effect”, the value of the variable is changed

```
>>> def double(n): return( n * 2 )
...
>>> m.append( double )
>>>
[0, 'un', 2, 'trois', <function double at 0x7f6230236668>]
>>> l3
['un', 'deux', 'trois']
>>> m.append( l3 )        # here we add a list inside another list
>>> m
[0, 'un', 2, 'trois', <function double at 0x7f6230236668>, ['un', 'deux', 'trois']]
>>>
```

To fuse two lists use the + operator

```
>>> l4
[0, 1, 2, 3]
>>> u = l3 + l4        # no side effect with '+'
>>> u
['un', 'deux', 'trois', 0, 1, 2, 3]
>>> l3
['un', 'deux', 'trois']
>>> l4
[0, 1, 2, 3]
```

Playing around with the lists.

```
>>> l4
[0, 1, 2, 3] # how read an element (here the third element of l4
>>> l4[ 2 ]  # since first element of the list has by default and index of 1
>>> l4[ 2 ] = 'cinq'   # accès en écriture à un élément
>>> l4
[0, 1, 'cinq', 3]
>>> l4[ 0 ]    # first element
0
>>> l4[ -1 ]   # second element
3
>>> l4[ 2:4 ]  # from the third element to the fourth one included
['cinq', 3]
>>> l4[ 1: ]   # from the second element up to the end
[1, 'cinq', 3]
>>> l4[ 1:-1]  # from the second element up to the one before last included
[1, 'cinq']
>>> l4[ :-1]   # from the beginning of the list up to the one but last element included
[0, 1, 'cinq']
>>> l4[:]      # from the first element to the last (toute la liste)
[0, 1, 'cinq', 3]
```

Search a list.

```
>>> l4
[0, 1, 'cinq', 3]
>>> l4.index( 'cinq' )
2
>>>>>> if( 'cinq' in l4 ):
...     print( 'présent' )
... else:
...     print( 'absent' )
...
présent
>>> l4.index( 17 )                # Warning !
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: 17 is not in list
>>> if( 17 in l4):
...    l4.index( 17 )
... else:
...    print( 'pas là' )
...
pas là
>>>
```

From strings to lists

```
>>> s = 'Bonjour, il fait beau.'
>>> l = s.split()
>>> l
['Bonjour,', 'il', 'fait', 'beau.']
>>> m = s.split('a')
>>> m
['Bonjour, il f', 'it be', 'u.']
>>> s.find( 'fait')
12
>>> s.find( 'toto')
-1
>>> r = [ s[ : s.find( 'fait') ], 'fait', s[ s.find( 'fait'): ] ]
>>> r
['Bonjour, il ', 'fait', 'fait beau.']
>>> len( r )
3
```

« tuples » (n-uplet) are lists with a fixed size once created (useful for returning compound results).

```
>>> un_tuple = ( 33, 'foo', double )
>>> un_tuple
(33, 'foo', <function double at 0x7f6230236668>)
>>> def succ_pred( n ):
...   return( n-1, n+1 )
...
>>> (n1, n2) = succ_pred( 6 )
>>> n1
5
>>> n2
7
>>> res = succ_pred( 8 )
>>> res[ 0 ]
7
>>> res[ 1 ]
9
>>> len( res )
2
>>> res.index( 9 )
1
>>> if( 9 in res ):
...   print( 'il est là' )
... else:
...   print( 'il est pas là' )
...
il est là
>>>
```

Dictionaries are lists whose items are indexed by any other objects (associative memories, i.e. lists of key :value pairs)

```
>>> dico_vide = {}
>>> mon_dico = { 'un':'ein', 'deux':'zwei', 'trois':'drei' }
>>> def trois_fun():
...       return 'drei'
...
>>> mon_dico_bis = { 'un':'ein', 'deux':2, 'trois':trois_fun }
>>> mon_dico_bis      # NOTE the ordering of te elements is not necessarity preserved
{'un': 'ein', 'trois': <function trois_fun at 0x7f6230236758>, 'deux': 2}
>>> mon_dico_bis[ 0 ]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 0
>>> mon_dico_bis[ 'un' ]
'ein'
>>> len( mon_dico_bis )
3
>>> mon_dico_bis[ 'nouveau' ] = 'new'          # add a key-value association
>>> mon_dico_bis
{'nouveau': 'new', 'un': 'ein', 'trois': <function trois_fun at 0x7f6230236758>, 'deux': 2}
>>> del mon_dico_bis[ 'un' ]                   # remove an association
>>> mon_dico_bis
{'nouveau': 'new', 'trois': <function trois_fun at 0x7f6230236758>, 'deux': 2}
>>> mon_dico_bis.has_key( 'deux' )             # test whether a key is present
True
```

```
>>> mon_dico_bis.keys()                    # list the keys
['nouveau', 'trois', 'deux']
>>> mon_dico_bis.values()                      # list the values
['new', <function trois_fun at 0x7f6230236758>, 2]
>>> # the list of associations as a liste of 2-uples
...
>>> mon_dico_bis.items()
[('nouveau', 'new'), ('trois', <function trois_fun at 0x7f6230236758>), ('deux', 2)]
>>>
```

Read a file

```
f = open( '/tmp/f.txt', 'r' )
>>> txt = f.read()  # en entier, whole at one time.
>>> txt
"La gestation de Vingt mille lieues sous les mers fut une des plus longues de l'histoire des \

L'id\xc3\xa9e de ce roman date de l'\xc3\xa9t\xc3\xa9 1866. Jules Verne le con\xc3\xa7ut \xc3\

amie d'Hetzel, qui avait fort appr\xc3\xa9ci\xc3\xa9 Cinq semaines en ballon et Voyage au cent

par laquelle elle pressait l'imagination de l'\xc3\xa9crivain, est rest\xc3\xa9e \xc3\xa0 la p

fier d'avoir int\xc3\xa9ress\xc3\xa9 un auteur qu'il estimait fort, l'a communiqu\xc3\xa9e, en

Voici le paragraphe inspirateur : \xc2\xab Je vous remercie, Monsieur, de vos aimables mots m

qui ont r\xc3\xa9ussi \xc3\xa0 me distraire d'une bien profonde douleur et \xc3\xa0 m'en faire

en ce qui les concerne, c'est de les avoir finis et de n'en avoir pas encore une douzaine \xc3

\n

"
>>> len( txt )
973
>>> f.close()
>>> f = open( '/tmp/f.txt', 'r' )
>>> l1 = f.readline()    # ligne by ligne
>>> l1
"La gestation de Vingt mille lieues sous les mers fut une des plus longues de l'histoire des \
>>> f.close()
```

Read file line by line to process it on the fly

```
f = open( '/tmp/f.txt', 'r')
>>> for line in f:
...   print( line )
...
La gestation de Vingt mille lieues sous les mers fut une des plus longues de l'histoire des V

L'idée de ce roman date de l'été 1865. Jules Verne le conçut à la suite d'une suggestion épist

amie d'Hetzel, qui avait fort apprécié Cinq semaines en ballon et Voyage au centre de la Terre

par laquelle elle pressait l'imagination de l'écrivain, est restée à la postérité par les soi

fier d'avoir intéressé un auteur qu'il estimait fort, l'a communiquée, en 1897, à Adolphe Bri

Voici le paragraphe inspirateur : « Je vous remercie, Monsieur, de vos aimables mots mis en d

qui ont réussi à me distraire d'une bien profonde douleur et à m'en faire supporter l'inquiét

en ce qui les concerne, c'est de les avoir finis et de n'en avoir pas encore une douzaine à l
>>> f.close()
```

Segment a string

```
Python 3.5.2 (default, Nov 23 2017, 16:37:01)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> s='C\'est un cours très intéresant'
>>> print( s )
C'est un cours très intéresant
>>> w = s.split()
>>> w
["C'est", 'un', 'cours', 'très', 'intéresant']
>>> w = s.split( '\'')
>>> w
['C', 'est un cours très intéresant']
>>> w = s.split( 'cours')
>>> w
["C'est un ", ' très intéresant']
>>>
```

Excercize 1 : Count the number of words in the text file
`https://perso.limsi.fr/pap/MIROR/pmc_1471-2458-5-97.txt`

In nltk there is a module tokenizer.

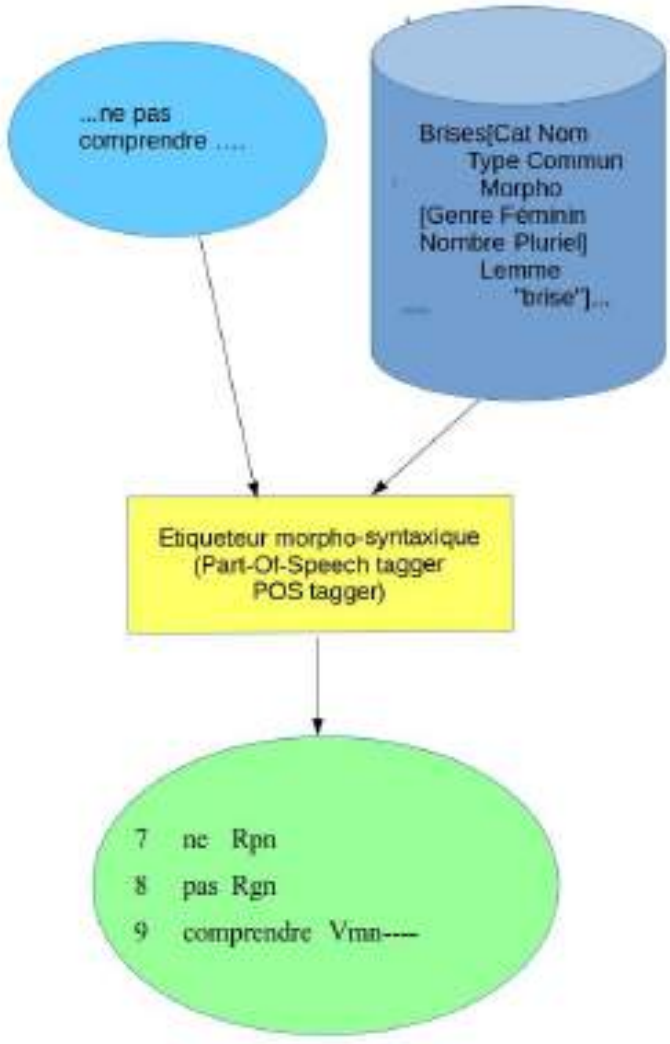
```
nltk.download('punkt') # module needed

from nltk.tokenize import sent_tokenize, word_token

data_w = "All work and no play makes jack a dull bo
print( word_tokenize( data_w ) )

data_snt = "All work and no play makes jack dull bo
print( sent_tokenize( data_snt ) )
```

POS Tagging

Part-of-Speech Tagging is assigning to each word its function class (Noun, Verb, Adjective, Prepostion, Adverb, Determiner) depending on the context it lies in.

This/Det course/Noun is/Verb interesting/Adj ./Punct

It is a word based task but it depends on morphology (flexional, derivational), syntax, semantics and pragmatics.


...ne pas
comprendre ....
Brises[Cat Nom
Type Commun
Morpho
[Genre Féminin
Nombre Pluriel]
Lemme
"brise"]...
Etiqueteur morpho-syntaxique
(Part-Of-Speech tagger
POS tagger)
7 ne Rpn
8 pas Rgn
9 comprendre Vmn----

POS

Ressource : Descriptions morpho-syntaxiques et jeux d'étiquettes

```
Brises[Cat      Nom
       Type     Commun
       Morpho   [Genre     Féminin
                 Nombre    Pluriel]
       Lemme    "brise"]
      [Cat      Verbe
       Type     Principal
       SousCat  Transitif
       Morpho   [Temps     Présent
                 Mode           Indicatif
                 Voix           Active
                 Personne       2ème
                 Genre     Singulier]

        Lemme   "briser"]
```

POS

Exemple d'annotation morpho-syntaxique automatique

0	I	Nkms
1	où	Pr-mp--
1	l	Pp3msn-/1.3
3	'	Pp3msn-/2.3
4	on	Pp3msn-/3.3
5	commence	Vmip3s-
6	à	Sp
7	ne	Rpn
8	pas	Rgn
9	comprendre	Vmn----
10	ce	Pd-ms--
11	n	Rpn/1.2
12	'	Rpn/2.2
13	est	Vmip3s-
14	pas	Rgn
15	sans	Sp
16	une	Da-fs-i
17	certaine	Ai-fs
18	émotion	Ncfs
19	que	Pr-fs--
20	je	Pp1msn-

Performances

Tagging implies :

1. segment, find word boundaries.
2. identify, i.e. list possible tags.
3. disambiguate, i.e. choose the correct(s) tag depending on context.

Performances are measured in terms of :

- percentage of correct tagging,
- precision for mono label classification,
- precision,recall for multi label classification),
- precision, decision when evaluating with multi-tagset annotations.

Performances

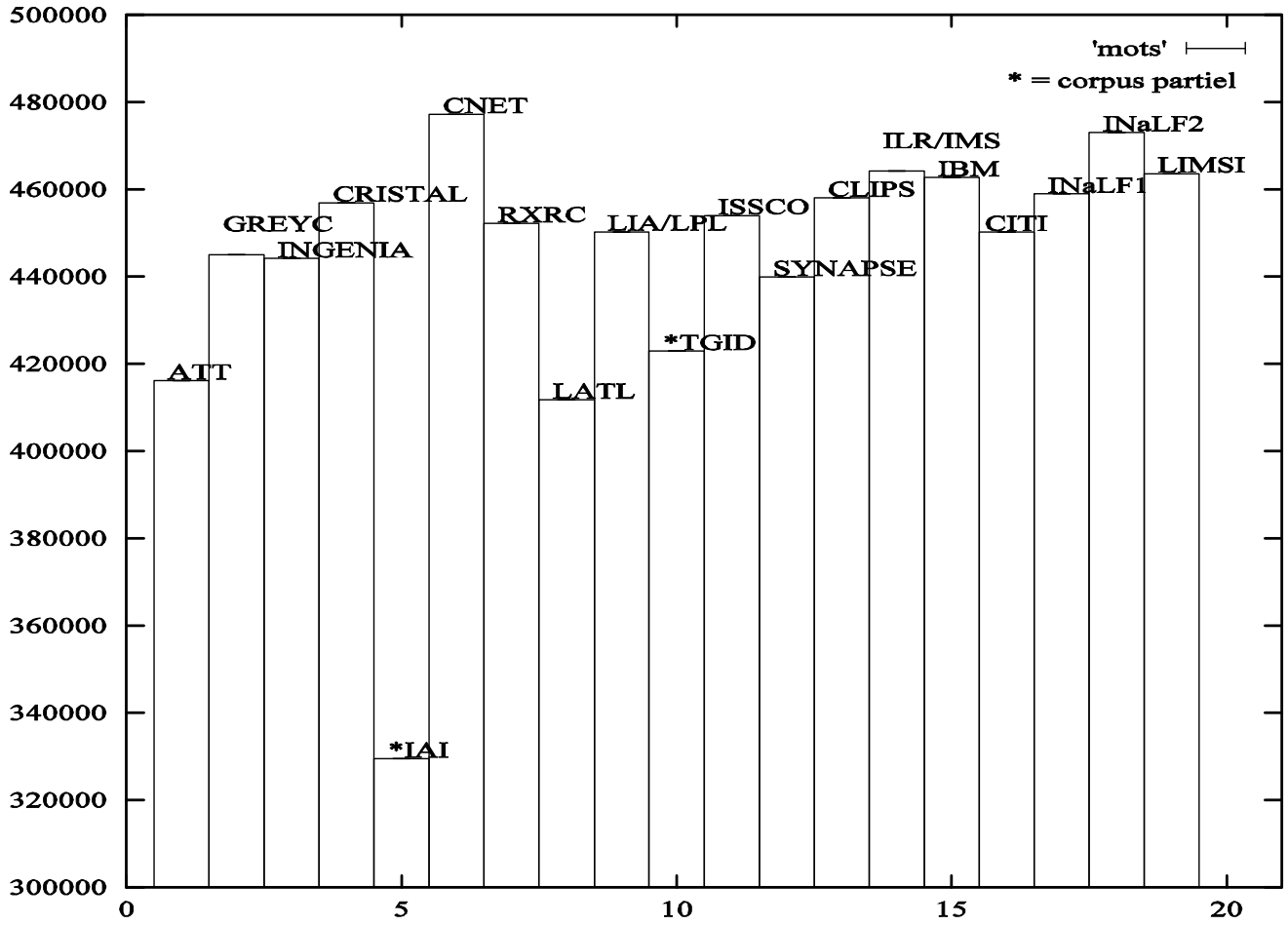
- Performances are almost always above 90% and sometimes reach 99%.
- (DEROSE 1988) measured approx. 96% on English (Brown corpus) with VOLSUNGA tagger.
- Three best performances in precision in the GRACE (Paroubek98) campaign on POS tagging of French : 97.8%, 96.7% et 94.8%.

How difficult is tagging ?

- Simple lexical tagging (dictionary access) without disambiguation achieved 88% in précision (multi-label) in GRACE evaluation.
- This score went down to 59% when a static rule was used for disambiguating.
- For a sentence of 15 words and a correct (monolabel) tagging rate of 96% we have only 54.2% at sentence level.
- To have a 95% correct tagging at the sentence level, we need to have a tagger that tags correctly words with a precision of 99.67% .

0	l	Nkms
1	où	Pr-mp--
1	l	Pp3msn-/1.3
2	'	Pp3msn-/2.3
4	on	Pp3msn-/3.3
5	commence	Vmip3s-
6	à	Sp
7	ne	Rpn
8	pas	Rgn
9	comprendre	Vmn----
10	ce	Pd-ms--
11	n	Rpn/1.2
12	'	Rpn/2.2
13	est	Vmip3s-
14	pas	Rgn
15	sans	Sp
16	une	Da-fs-i
17	certaine	Ai-fs
18	émotion	Ncfs
19	que	Pr-fs--
20	je	Pp1msn-

Nombre de mots en fonction du participant (GRACE)



000000 Au DTC:sg
000001 cours SBC:sg
000002 de PREP

Alignement (15 systèmes différents pour les tests)

000000 Au Sp+Da-ms-d
000001 cours Ncfs|Ncms
000002 de Da----i|Da-fp-i|Da-mp-i|Sp

Projection des étiquettes dans le jeu GRACE

000000 Au Sp/1.3 6/14[0.428571]
000001 cours Ncms|Sp/2.3 6/15[0.4]
000002 de Sp 7/13[0.538462]

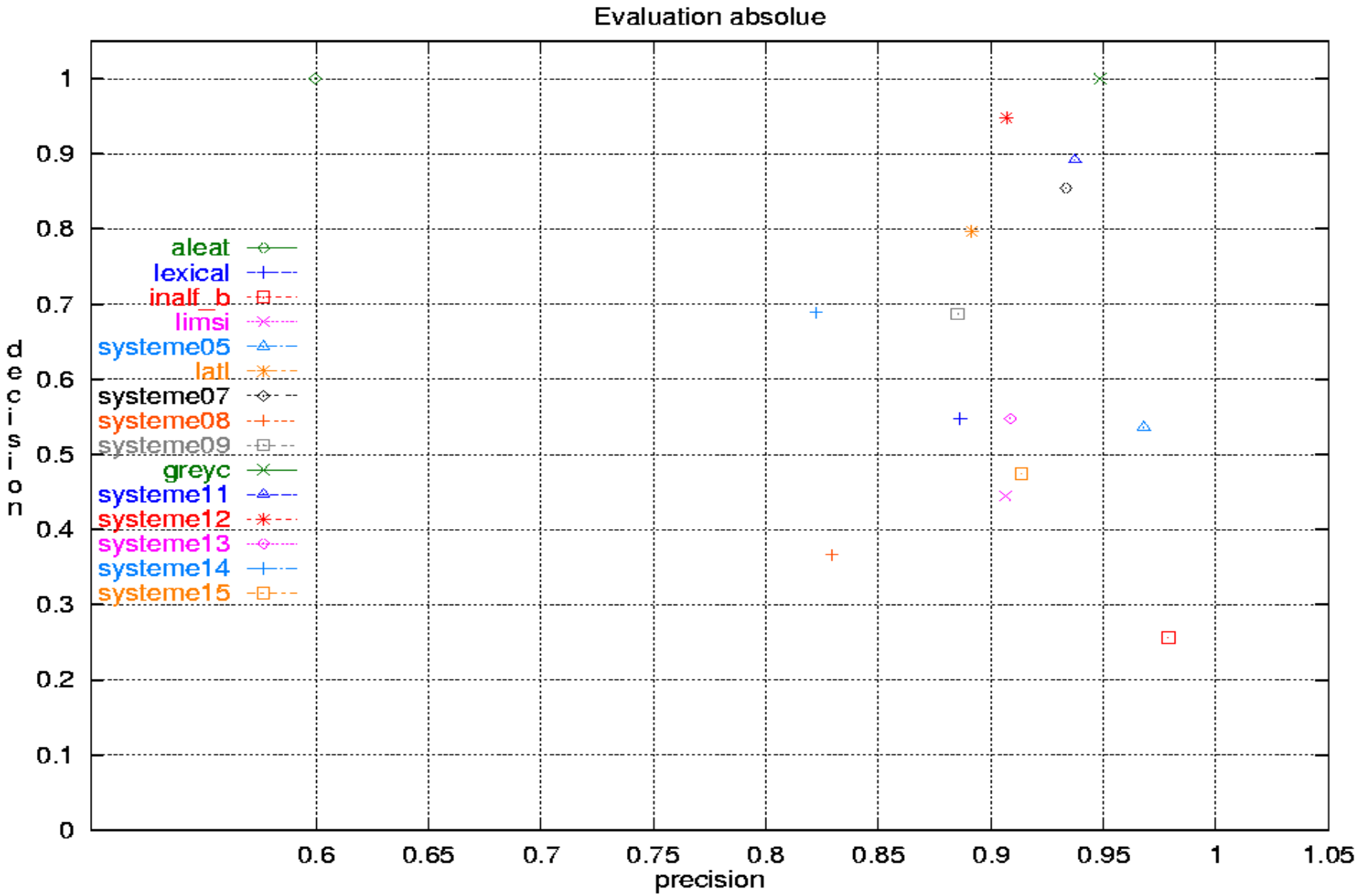
Combinaison Vote & mesure de confiance

GRACE, évaluation d'étiquettage morphosyntaxique pour le français, 21 participants, 5 pays:

3 phases: entrainements (10 millions de mots),essais (450.000), test (836.500)

17 participants aux essais, 13 participants aux tests finaux

mesure précision/décision, sur 20.000 mots, puis 40.000 mots.étiquettes EAGLES et MULTEXT


Evaluation absolue
decision
precision
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0.6
0.65
0.7
0.75
0.8
0.85
0.9
0.95
1
1.05
aleat
lexical
inalf_b
limsi
systeme05
latl
systeme07
systeme08
systeme09
greyc
systeme11
systeme12
systeme13
systeme14
systeme15

They are underspecified search patterns.
E.g. `file[1-9].*`, search for names starting with file ending with one number between 1 and 9 included that have any non empty suffix.

- caracter constants, par ex. « navire, ==, 33, ?, 18EF, » etc.
- concatenation of patterns by juxtapostion.
- repetition operators :
  - Kleen star `*` for repetions betweene `0` and `n` or operator `+` for repetitions between `1` and `n`.
  - alternation of patterns `|`,
  - optionality of patterns `?`
  - negation of patterns `!` or `^`.
-

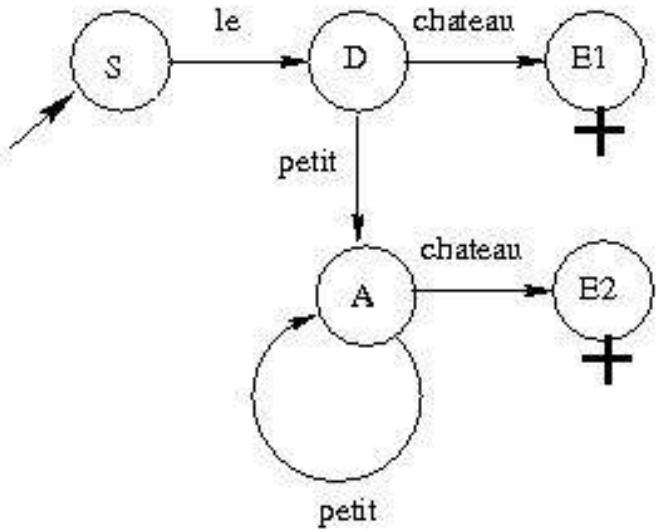
E.g. all words beginning with the prefix `re` and ending with the suffix `er` or `ir` is the pattern : `re.*(er|ir)`, the dot `.` means 1 caracter, any of them but only one.

Finit state automata

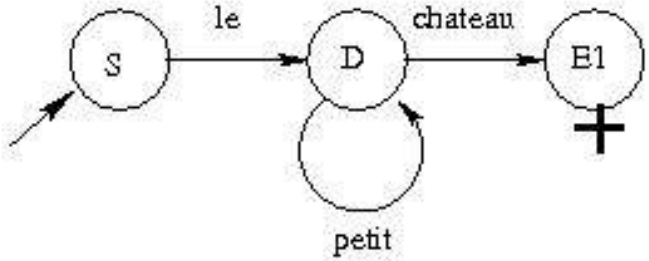
Formally, regular expressions are deterministic finite state automata.

le chateau
le petit chateau
le petit petit chateau
le petit petit petit chateau
le petit petit petit.....petit chateau

automate non minimal



automate minimal



Finit state automata

- An deterministic finite state automata (or regular expression) is like a public transportation map and as the automate progress along the linear sequence of characters composing the input string, the control of the automata (the node currently active) moves to then next state (station in the public transportation map).
- when the input string has been processed and the current node active is terminal one, the input chain is recognized as a valid instance of a text and the process stops
- if the state that the automata ends in and there are still unprocessed elements the character media, it means that the sequence of characters is not recognized as the language defined by the automata.
- At each node of the automata graph and for each individual character, there is only one transition that the automata can use to go the the next state.
- Note that regular expressions are a way to specify all the correct text documents of a language.
- A regular expression is always finite, but the set of all documents of the language may be infinite.

Python packages for regular expressions : `re` (or slightly more elaborate regex), with two modes of use : `search()` and `match()`
Sample program using the re package : `https://perso.limsi.fr/pap/MIROR/prog2_re.py`

```
import re
SPACE = chr(int('0020', 16))
NO_BREAK_SPACE = chr(int('00A0', 16))
OGHAM_SPACE_MARK  = chr(int('1680', 16))
MONGOLIAN_VOWEL_SEPARATOR  = chr(int('180E', 16))
# Note: there are other.
SPC_LST = ( SPACE + NO_BREAK_SPACE + OGHAM_SPACE_MARK  + MONGOLIAN_VOWEL_SEPARATOR )
txt = 'Is this course ending soon ?'
non_spc_patt = re.compile( '[^' + SPC_LST + ']+$' )
spc_only_patt = re.compile( '^[' + SPC_LST + ']+$' )
s=0
tokens=[]
while s < len( txt ):
   res = non_spc_patt.search( txt, pos = s )
   if( res ):
       tok = res.group(0)
       assert( not spc_only_patt.match( tok ) )
       first,last = res.span(0)
       assert( type( tok ) is str )
       tokens.append( (tok, (first, last)) )
       s = last
   else:
      break
print( tokens )
```

Sample program using the treeTagger python API for french in the package treetaggerwrapper :
https://perso.limsi.fr/pap/MIROR/do_treetagger_fr.py
https://perso.limsi.fr/pap/MIROR/sent1_fr.txt
https://perso.limsi.fr/pap/MIROR/sent1_fr.ttgd
Usage :

```
$ cat /tmp/sent1_fr.txt
C'est un cours très ennuyeux!
$ python3 do_treetagger.py -i sent1_fr.txt -o sent1_fr.ttgd
$ cat /tmp/sent1_fr.ttgd
Tag(word="C'", pos='PRO:DEM', lemma='ce') Tag(word='est', pos='VER:pres', lemma='être')
Tag(word='un', pos='DET:ART', lemma='un') Tag(word='cours', pos='NOM', lemma='cour|cours')
Tag(word='très', pos='ADV', lemma='très') Tag(word='ennuyeux', pos='ADJ', lemma='ennuyeux')
Tag(word='!', pos='SENT', lemma='!')
```

Excercize 2 :

- Use the re package to write a tagger for English Determiners and apply it to the text : `https://perso.limsi.fr/pap/MIROR/pmc_1471-2458-5-97.txt`
- Compare your tagging of determiners with the one in `https://perso.limsi.fr/pap/MIROR/pmc_1471-2458-5-97.ttgd` and compute your precision performance on the determiners.