Série TP 2 - TALN - Traitements de Bas Niveau

Partie 1 - Encodage de caractères et Tokenization

1 - L'encodage de caractères et de chaines de caractères

In [1]:

```
# Fixer l' encodage des caractères dans l'entête du script - Par défaut unicode UTF-8
# -*- coding: utf-8 -*-
```

In [2]:

```
# Afficher tous les encodages qui existent

import sys
import encodings

# Récuperer le nom des encodages
encodages = encodings.aliases.aliases.values()

# Eliminer les doublants
encodages = set(encodages)

# Trier les éléments de la liste
encodages = sorted(encodages)

# join permet de mettre un séparateur entre les éléments pour les afficher
print("\n".join(encodages))
```

```
ascii
base64_codec
big5
big5hkscs
bz2_codec
cp037
cp1026
cp1125
cp1140
cp1250
cp1251
cp1252
cp1253
cp1254
cp1255
cp1256
cp1257
cp1258
cp273
cp424
cp437
cp500
cp775
cp850
cp852
cp855
cp857
cp858
cp860
cp861
cp862
cp863
cp864
cp865
```

```
cp866
cp869
cp932
cp949
cp950
euc_jis_2004
euc_jisx0213
euc_jp
euc_kr
gb18030
gb2312
gbk
hex_codec
hp_roman8
hz
iso2022_jp
iso2022_jp_1
iso2022_jp_2
iso2022_jp_2004
iso2022_jp_3
iso2022_jp_ext
iso2022_kr
iso8859_10
iso8859_11
iso8859_13
iso8859_14
iso8859_15
iso8859_16
iso8859_2
iso8859_3
iso8859_4
iso8859_5
iso8859_6
iso8859_7
iso8859_8
iso8859_9
johab
koi8_r
kz1048
latin_1
mac_cyrillic
mac_greek
mac_iceland
mac_latin2
mac_roman
mac_turkish
mbcs
ptcp154
quopri_codec
rot_13
shift_jis
shift_jis_2004
shift_jisx0213
tactis
tis_620
utf_16
utf_16_be
utf_16_le
utf_32
utf_32_be
utf_32_le
utf_7
utf_8
uu_codec
zlib_codec
```

In [3]:

```
# Recuperer le codepoint en décimal d'un caractère : ord()

ord('A')
```

Out[3]:

```
65
```

In [4]:

```
print(ord('é'))
print(ord('€'))
```

```
233
8364
```

In [5]:

```
# Recuperer le caractère par son codepoint en décimal : chr()

chr(65)
```

Out[5]:

```
'A'
```

In [6]:

```
print(chr(97))
print(chr(237))
print(chr(8364))
```

```
a
í
€
```

In [7]:

```
# Afficher tous les caractères (le Charset) de l'encodage ASCII 7 bits - 128 caractères
for i in range(128):
    print(chr(i))
```

```
!
"
```

#
$
%
&
'
(
)
*
+
,
-
.
/
0
1
2
3
4
5
6
7
8
9
:
;
<
=
>
?
@
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z
[
\
]
^
_
`
a
b
c
d
e
f
g
h
i
j

```
k
l
m
n
o
p
q
r
s
t
u
v
w
x
y
z
{
|
}
~
```

In [8]:

```
# Afficher un caractère par son code point Unicode : \u + codepoint en 4 chiffres
print('\u0636')
```

ض

In [9]:

```
print('\u1F68')
```

Ὠ

In [10]:

```
# En Python3, il existe deux types de chaine de caractère: str (character string) et byte
s (byte string)

ch = 'Bonjour'
print(ch, type(ch))

bt = b'Bonsoir'
print(bt, type(bt))
```

```
Bonjour <class 'str'>
b'Bonsoir' <class 'bytes'>
```

Différence entre str et bytes

La seule chose qu'un ordinateur peut stocker, ce sont des bytes (octets).

Pour stocker quoi que ce soit dans un ordinateur, vous devez d'abord l'encoder, c'est-à-dire le convertir en bytes. Par exemple:

- **Si vous souhaitez stocker de la musique, vous devez d'abord l'encoder en MP3, WAV, etc.**
- **Si vous souhaitez stocker une image, vous devez d'abord l'encoder en PNG, JPEG, etc.**
- **Si vous souhaitez stocker du texte, vous devez d'abord l'encoder en ASCII, UTF-8, etc.**

MP3, WAV, PNG, JPEG, ASCII et UTF-8 sont des exemples d'encodages. Un encodage est un format pour représenter l'audio, les images, le texte, etc. en bytes.

En Python, une chaîne binaire (bytes) n'est que cela : une séquence d'octets binaire. Ce n'est pas lisible par l'être humain. Tout doit être converti en une chaîne bytes avant de pouvoir être stocké dans un ordinateur.

D'autre part, une chaîne de caractères (str), souvent simplement appelée "string", est une séquence de caractères. Il est lisible par l'être humain. Une chaîne de caractères ne peut pas être stockée directement dans un ordinateur, elle doit d'abord être encodée (convertie en une chaîne bytes). Il existe plusieurs encodages par

lesquels une chaine de caracteres peut être convertie en une chaine bytes, tels que ASCII et UTF-8.

In [11]:

```
# A noter que : bytes ne peut contenir que des caractères ASCII (1 octet)

bt_2 = b'مرحبا'
```

```
  Input In [11]
    bt_2 = b'مرحبا'
           ^
SyntaxError: bytes can only contain ASCII literal characters.
```

In [12]:

```
# Encoder (convertir) une chaine de caractères str vers bytes en utilisant l'encodage asc
ii : encode()
my_str = 'I am a string'
my_bt = my_str.encode('ascii')
```

In [13]:

```
# Rappel: les chaînes bytes ne sont pas lisibles par l'humain, Python les décode à partir
de l'ASCII lorsque vous les affichez
print(my_bt)
```

```
b'I am a string'
```

In [14]:

```
# Decoder (reconvertir) le bytes vers str : decode()
my_bt.decode('ascii')
```

Out[14]:

```
'I am a string'
```

In [15]:

```
# Encoder (convertir) une chaine de caractères str vers bytes en utilisant l'encodage UTF
-8 : encode()
my_str = 'I am a string'
my_bt = my_str.encode('utf-8')
print(my_bt)
```

```
b'I am a string'
```

In [16]:

```
# Decoder (reconvertir) le bytes vers str : decode()
my_bt.decode('utf-8')
```

Out[16]:

```
'I am a string'
```

In [17]:

```
# Attention aux erreurs - caractères non pris en compte par l'encodage : UnicodeDecodeErr
or
utf_str = "Euro symbol is a non ASCII character: €"
ascii_bt = utf_str.encode('ascii')
print(ascii_bt)
```

```
---------------------------------------------------------------------------
UnicodeEncodeError                        Traceback (most recent call last)
Input In [17], in <cell line: 3>()
      1 # Attention aux erreurs - caractères non pris en compte par l'encodage : UnicodeD
ecodeError
      2 utf_str = "Euro symbol is a non ASCII character: €"
----> 3 ascii_bt = utf_str.encode('ascii')
      4 print(ascii_bt)
```

```
UnicodeEncodeError: 'ascii' codec can't encode character '\u20ac' in position 38: ordinal
not in range(128)
```

In [18]:

```
# Solution : le paramètre errors ('replace', 'backslashreplace', 'ignore', 'namereplace')
utf_str = "Euro symbol is a non ASCII character: €"
ascii_bt = utf_str.encode('ascii', errors="ignore")
print(ascii_bt)
```

```
b'Euro symbol is a non ASCII character: '
```

In [19]:

```
# Solution : le paramètre errors ('replace', 'backslashreplace', 'ignore', 'namereplace')
utf_str = "Euro symbol is a non ASCII character: €"
ascii_bt = utf_str.encode('ascii', errors="namereplace")
print(ascii_bt)
```

```
b'Euro symbol is a non ASCII character: \\N{EURO SIGN}'
```

In [20]:

```
# Solution : le paramètre errors ('replace', 'backslashreplace', 'ignore', 'namereplace')
utf_str = "Euro symbol is a non ASCII character: €"
ascii_bt = utf_str.encode('ascii', errors="replace")
print(ascii_bt)
```

```
b'Euro symbol is a non ASCII character: ?'
```

In [21]:

```
# Reconvertir/décoder vers le str
ascii_bt.decode("ascii") # perte du caractère € dans la chaine originale, remplacé par le
?, car non représenté dans l'ascii
```

Out[21]:

```
'Euro symbol is a non ASCII character: ?'
```

In [22]:

```
# La même chaîne bytes peut représenter 2 chaînes str différentes dans 2 encodages différ
ents

machaine = b'\xcf\x84o\xcf\x81\xce\xbdo\xcf\x82'

print(machaine.decode('utf-8'))

print(machaine.decode('utf-16'))
```

```
τορνος
蘁콯칽濠苏
```

In [23]:

```
# Afficher des formules chimiques avec codepoint unicode : \u

print("The chemical formula of water is H\u2082O.Water dissociates into H\u207A and OH\u2
07B")
```

```
The chemical formula of water is H₂O.Water dissociates into H⁺ and OH⁻
```

In [24]:

```
# Afficher les caractères arabes
for i in range(1536, 1791):
    print(chr(i))
```

```
؀
؁
؂
```

؀
؆
؇
؉
؊
؋
،
؍
؎
؏
؛
؞
؟
ء
آ
أ
ؤ
إ
ئ
ا
ب
ة
ت
ث
ج
ح
خ
د
ذ
ر
ز
س
ش
ص
ض
ط
ظ
ع
غ
ـ
ف
ق
ك
ل
م
ن
ه
و
ى
ي

ً
ٌ
ٍ
ُ
ِ
ْ
ٓ
ٔ
ٕ
•
□
٠
١
٢
٣
٤
٥
٦
٧
٨
٩
٪
٫
٬
٭
ٮ
ٯ
ٱ
ٲ
ٳ
ٵ
ٶ
ٷ
ٸ
ٹ
ٺ
ٻ
ټ
ٽ
پ
ٿ
ڀ
ځ
ڂ
ڃ
ڄ
څ
چ
ڇ
ڈ
ډ
ڊ
ڋ
ڌ
ڍ
ڎ
ڏ
ڐ
ڑ
ڒ

ړ
ڔ
ڕ
ږ
ڗ
ژ
ڙ
ښ
ڛ
ڜ
ڝ
ڞ
ڟ
ڠ
ڡ
ڢ
ڣ
ڤ
ڥ
ڦ
ڧ
ڨ
ک
ڪ
ګ
ڬ
ڭ
ڮ
گ
ڰ
ڱ
ڲ
ڳ
ڴ
ڵ
ڶ
ڷ
ڸ
ڹ
ں
ڻ
ڼ
ڽ
ھ
ڿ
ۀ
ہ
ۂ
ۃ
ۄ
ۅ
ۆ
ۇ
ۈ
ۉ
ۊ
ۋ
ی
ۍ
ێ
ۏ
ې
ۑ
ے
ۓ
۔
ە

ۏ
ۜ
ۛ
۝
۞
۟
۠
ۡ
ۢ
ۣ
ۤ
ۥ
ۦ
ۧ
ۨ
۩
۪
ۭ
ۮ
ۯ
۰
۱
۲
۳
۴
۵
۶
۷
۸
۹
۾
ۿ

In [25]:

```
# Afficher les caractères tifinagh
for i in range(11568 , 11631):
    print(chr(i))
```

ⴰ
ⴱ
ⴲ
ⴳ
ⴴ
ⴵ
ⴶ
ⴷ
ⴸ
ⴹ
ⴺ
ⴻ
ⴼ
ⴽ
ⴾ
ⴿ
ⵀ
ⵁ
ⵂ
ⵃ
ⵄ
ⵅ
ⵆ
ⵇ
ⵈ
ⵉ
ⵊ
ⵋ
ⵌ
ⵍ
ⵎ
ⵏ
ⵐ
ⵑ
ⵒ
ⵓ
ⵔ
ⵕ
ⵖ
ⵗ
ⵘ
ⵙ
ⵚ
ⵛ
ⵜ
ⵝ
ⵞ
ⵟ

In [26]:

```
# Afficher les caractères arabe avec leurs noms
import unicodedata
for i in range(1575, 1791):
    print(i, unicodedata.name(chr(i)))
```

```
1575 ARABIC LETTER ALEF
1576 ARABIC LETTER BEH
1577 ARABIC LETTER TEH MARBUTA
1578 ARABIC LETTER TEH
1579 ARABIC LETTER THEH
1580 ARABIC LETTER JEEM
1581 ARABIC LETTER HAH
1582 ARABIC LETTER KHAH
1583 ARABIC LETTER DAL
1584 ARABIC LETTER THAL
1585 ARABIC LETTER REH
1586 ARABIC LETTER ZAIN
1587 ARABIC LETTER SEEN
1588 ARABIC LETTER SHEEN
1589 ARABIC LETTER SAD
1590 ARABIC LETTER DAD
1591 ARABIC LETTER TAH
1592 ARABIC LETTER ZAH
1593 ARABIC LETTER AIN
1594 ARABIC LETTER GHAIN
1595 ARABIC LETTER KEHEH WITH TWO DOTS ABOVE
1596 ARABIC LETTER KEHEH WITH THREE DOTS BELOW
1597 ARABIC LETTER FARSI YEH WITH INVERTED V
1598 ARABIC LETTER FARSI YEH WITH TWO DOTS ABOVE
1599 ARABIC LETTER FARSI YEH WITH THREE DOTS ABOVE
1600 ARABIC TATWEEL
1601 ARABIC LETTER FEH
1602 ARABIC LETTER QAF
1603 ARABIC LETTER KAF
1604 ARABIC LETTER LAM
```

1605 ARABIC LETTER MEEM
1606 ARABIC LETTER NOON
1607 ARABIC LETTER HEH
1608 ARABIC LETTER WAW
1609 ARABIC LETTER ALEF MAKSURA
1610 ARABIC LETTER YEH
1611 ARABIC FATHATAN
1612 ARABIC DAMMATAN
1613 ARABIC KASRATAN
1614 ARABIC FATHA
1615 ARABIC DAMMA
1616 ARABIC KASRA
1617 ARABIC SHADDA
1618 ARABIC SUKUN
1619 ARABIC MADDAH ABOVE
1620 ARABIC HAMZA ABOVE
1621 ARABIC HAMZA BELOW
1622 ARABIC SUBSCRIPT ALEF
1623 ARABIC INVERTED DAMMA
1624 ARABIC MARK NOON GHUNNA
1625 ARABIC ZWARAKAY
1626 ARABIC VOWEL SIGN SMALL V ABOVE
1627 ARABIC VOWEL SIGN INVERTED SMALL V ABOVE
1628 ARABIC VOWEL SIGN DOT BELOW
1629 ARABIC REVERSED DAMMA
1630 ARABIC FATHA WITH TWO DOTS
1631 ARABIC WAVY HAMZA BELOW
1632 ARABIC-INDIC DIGIT ZERO
1633 ARABIC-INDIC DIGIT ONE
1634 ARABIC-INDIC DIGIT TWO
1635 ARABIC-INDIC DIGIT THREE
1636 ARABIC-INDIC DIGIT FOUR
1637 ARABIC-INDIC DIGIT FIVE
1638 ARABIC-INDIC DIGIT SIX
1639 ARABIC-INDIC DIGIT SEVEN
1640 ARABIC-INDIC DIGIT EIGHT
1641 ARABIC-INDIC DIGIT NINE
1642 ARABIC PERCENT SIGN
1643 ARABIC DECIMAL SEPARATOR
1644 ARABIC THOUSANDS SEPARATOR
1645 ARABIC FIVE POINTED STAR
1646 ARABIC LETTER DOTLESS BEH
1647 ARABIC LETTER DOTLESS QAF
1648 ARABIC LETTER SUPERSCRIPT ALEF
1649 ARABIC LETTER ALEF WASLA
1650 ARABIC LETTER ALEF WITH WAVY HAMZA ABOVE
1651 ARABIC LETTER ALEF WITH WAVY HAMZA BELOW
1652 ARABIC LETTER HIGH HAMZA
1653 ARABIC LETTER HIGH HAMZA ALEF
1654 ARABIC LETTER HIGH HAMZA WAW
1655 ARABIC LETTER U WITH HAMZA ABOVE
1656 ARABIC LETTER HIGH HAMZA YEH
1657 ARABIC LETTER TTEH
1658 ARABIC LETTER TTEHEH
1659 ARABIC LETTER BEEH
1660 ARABIC LETTER TEH WITH RING
1661 ARABIC LETTER TEH WITH THREE DOTS ABOVE DOWNWARDS
1662 ARABIC LETTER PEH
1663 ARABIC LETTER TEHEH
1664 ARABIC LETTER BEHEH
1665 ARABIC LETTER HAH WITH HAMZA ABOVE
1666 ARABIC LETTER HAH WITH TWO DOTS VERTICAL ABOVE
1667 ARABIC LETTER NYEH
1668 ARABIC LETTER DYEH
1669 ARABIC LETTER HAH WITH THREE DOTS ABOVE
1670 ARABIC LETTER TCHEH
1671 ARABIC LETTER TCHEHEH
1672 ARABIC LETTER DDAL
1673 ARABIC LETTER DAL WITH RING
1674 ARABIC LETTER DAL WITH DOT BELOW
1675 ARABIC LETTER DAL WITH DOT BELOW AND SMALL TAH
1676 ARABIC LETTER DAHAL

1677 ARABIC LETTER DDAHAL
1678 ARABIC LETTER DUL
1679 ARABIC LETTER DAL WITH THREE DOTS ABOVE DOWNWARDS
1680 ARABIC LETTER DAL WITH FOUR DOTS ABOVE
1681 ARABIC LETTER RREH
1682 ARABIC LETTER REH WITH SMALL V
1683 ARABIC LETTER REH WITH RING
1684 ARABIC LETTER REH WITH DOT BELOW
1685 ARABIC LETTER REH WITH SMALL V BELOW
1686 ARABIC LETTER REH WITH DOT BELOW AND DOT ABOVE
1687 ARABIC LETTER REH WITH TWO DOTS ABOVE
1688 ARABIC LETTER JEH
1689 ARABIC LETTER REH WITH FOUR DOTS ABOVE
1690 ARABIC LETTER SEEN WITH DOT BELOW AND DOT ABOVE
1691 ARABIC LETTER SEEN WITH THREE DOTS BELOW
1692 ARABIC LETTER SEEN WITH THREE DOTS BELOW AND THREE DOTS ABOVE
1693 ARABIC LETTER SAD WITH TWO DOTS BELOW
1694 ARABIC LETTER SAD WITH THREE DOTS ABOVE
1695 ARABIC LETTER TAH WITH THREE DOTS ABOVE
1696 ARABIC LETTER AIN WITH THREE DOTS ABOVE
1697 ARABIC LETTER DOTLESS FEH
1698 ARABIC LETTER FEH WITH DOT MOVED BELOW
1699 ARABIC LETTER FEH WITH DOT BELOW
1700 ARABIC LETTER VEH
1701 ARABIC LETTER FEH WITH THREE DOTS BELOW
1702 ARABIC LETTER PEHEH
1703 ARABIC LETTER QAF WITH DOT ABOVE
1704 ARABIC LETTER QAF WITH THREE DOTS ABOVE
1705 ARABIC LETTER KEHEH
1706 ARABIC LETTER SWASH KAF
1707 ARABIC LETTER KAF WITH RING
1708 ARABIC LETTER KAF WITH DOT ABOVE
1709 ARABIC LETTER NG
1710 ARABIC LETTER KAF WITH THREE DOTS BELOW
1711 ARABIC LETTER GAF
1712 ARABIC LETTER GAF WITH RING
1713 ARABIC LETTER NGOEH
1714 ARABIC LETTER GAF WITH TWO DOTS BELOW
1715 ARABIC LETTER GUEH
1716 ARABIC LETTER GAF WITH THREE DOTS ABOVE
1717 ARABIC LETTER LAM WITH SMALL V
1718 ARABIC LETTER LAM WITH DOT ABOVE
1719 ARABIC LETTER LAM WITH THREE DOTS ABOVE
1720 ARABIC LETTER LAM WITH THREE DOTS BELOW
1721 ARABIC LETTER NOON WITH DOT BELOW
1722 ARABIC LETTER NOON GHUNNA
1723 ARABIC LETTER RNOON
1724 ARABIC LETTER NOON WITH RING
1725 ARABIC LETTER NOON WITH THREE DOTS ABOVE
1726 ARABIC LETTER HEH DOACHASHMEE
1727 ARABIC LETTER TCHEH WITH DOT ABOVE
1728 ARABIC LETTER HEH WITH YEH ABOVE
1729 ARABIC LETTER HEH GOAL
1730 ARABIC LETTER HEH GOAL WITH HAMZA ABOVE
1731 ARABIC LETTER TEH MARBUTA GOAL
1732 ARABIC LETTER WAW WITH RING
1733 ARABIC LETTER KIRGHIZ OE
1734 ARABIC LETTER OE
1735 ARABIC LETTER U
1736 ARABIC LETTER YU
1737 ARABIC LETTER KIRGHIZ YU
1738 ARABIC LETTER WAW WITH TWO DOTS ABOVE
1739 ARABIC LETTER VE
1740 ARABIC LETTER FARSI YEH
1741 ARABIC LETTER YEH WITH TAIL
1742 ARABIC LETTER YEH WITH SMALL V
1743 ARABIC LETTER WAW WITH DOT ABOVE
1744 ARABIC LETTER E
1745 ARABIC LETTER YEH WITH THREE DOTS BELOW
1746 ARABIC LETTER YEH BARREE
1747 ARABIC LETTER YEH BARREE WITH HAMZA ABOVE
1748 ARABIC FULL STOP

```
1749 ARABIC LETTER AE
1750 ARABIC SMALL HIGH LIGATURE SAD WITH LAM WITH ALEF MAKSURA
1751 ARABIC SMALL HIGH LIGATURE QAF WITH LAM WITH ALEF MAKSURA
1752 ARABIC SMALL HIGH MEEM INITIAL FORM
1753 ARABIC SMALL HIGH LAM ALEF
1754 ARABIC SMALL HIGH JEEM
1755 ARABIC SMALL HIGH THREE DOTS
1756 ARABIC SMALL HIGH SEEN
1757 ARABIC END OF AYAH
1758 ARABIC START OF RUB EL HIZB
1759 ARABIC SMALL HIGH ROUNDED ZERO
1760 ARABIC SMALL HIGH UPRIGHT RECTANGULAR ZERO
1761 ARABIC SMALL HIGH DOTLESS HEAD OF KHAH
1762 ARABIC SMALL HIGH MEEM ISOLATED FORM
1763 ARABIC SMALL LOW SEEN
1764 ARABIC SMALL HIGH MADDA
1765 ARABIC SMALL WAW
1766 ARABIC SMALL YEH
1767 ARABIC SMALL HIGH YEH
1768 ARABIC SMALL HIGH NOON
1769 ARABIC PLACE OF SAJDAH
1770 ARABIC EMPTY CENTRE LOW STOP
1771 ARABIC EMPTY CENTRE HIGH STOP
1772 ARABIC ROUNDED HIGH STOP WITH FILLED CENTRE
1773 ARABIC SMALL LOW MEEM
1774 ARABIC LETTER DAL WITH INVERTED V
1775 ARABIC LETTER REH WITH INVERTED V
1776 EXTENDED ARABIC-INDIC DIGIT ZERO
1777 EXTENDED ARABIC-INDIC DIGIT ONE
1778 EXTENDED ARABIC-INDIC DIGIT TWO
1779 EXTENDED ARABIC-INDIC DIGIT THREE
1780 EXTENDED ARABIC-INDIC DIGIT FOUR
1781 EXTENDED ARABIC-INDIC DIGIT FIVE
1782 EXTENDED ARABIC-INDIC DIGIT SIX
1783 EXTENDED ARABIC-INDIC DIGIT SEVEN
1784 EXTENDED ARABIC-INDIC DIGIT EIGHT
1785 EXTENDED ARABIC-INDIC DIGIT NINE
1786 ARABIC LETTER SHEEN WITH DOT BELOW
1787 ARABIC LETTER DAD WITH DOT BELOW
1788 ARABIC LETTER GHAIN WITH DOT BELOW
1789 ARABIC SIGN SINDHI AMPERSAND
1790 ARABIC SIGN SINDHI POSTPOSITION MEN
```

2 - Segementation - Tokenization

In [27]:

```
text = "I'm Very Hungry. I want to eat something, maybe an apple."
```

In [28]:

```
# Sentence tokenization : split(separator, maxsplit)
text.split(".")
```

Out[28]:

```
["I'm Very Hungry", ' I want to eat something, maybe an apple', '']
```

In [29]:

```
# Word tokenization - White space : split(separator, maxsplit)
text.split(" ")
```

Out[29]:

```
["I'm",
 'Very',
 'Hungry.',
 'I',
 'want',
 'to',
```

```
 'eat',
 'something,',
 'maybe',
 'an',
 'apple.']
```

In [30]:

```
example = "apple#banana#cherry#orange"
x = example.split("#")
print(x)
```

```
['apple', 'banana', 'cherry', 'orange']
```

In [31]:

```
example.split("#", maxsplit=1)
```

Out[31]:

```
['apple', 'banana#cherry#orange']
```

RegEx - re Module : split(), search(), findall(), sub()

RegEx, ou Regular Expression, est une séquence de caractères qui forme un modèle de recherche. RegEx peut être utilisé pour vérifier si une chaîne contient le modèle de recherche spécifié.

In [32]:

```
# Word tokenization - regex : split() from re module
import re
```

In [33]:

```
# Vérifie si text commence par I et se termine par apple.
bl = re.search("^I.*apple.$", text)
if bl:
  print("YES! We have a match!")
else:
  print("No match")
```

```
YES! We have a match!
```

Explication :

- **. veut dire n'importe quel caractère**
- **^ veut dire commence par**
- **$ veut dire se termine par**
- *** Indicateur de répétition de car - correspond à zéro ou plusieurs instances de ce caractère**
- **+ Indicateur de répétitionde car - correspond à un ou plusieurs instances de ce caractère**
- **? Indicateur de répétition de car - correspond à zéro ou une instance de ce caractère**
- ***? correspond zéro fois ou plus, mais le moins de fois possible.**
- **[] un ensemble de caractères**
- **{} exactement le nombe spécifié de caractères**
- **[a-z] n'importe quelle lettre minuscule**
- **[0-9] N'importe quel chiffre**

In [34]:

```
# Tokenize par des exp-reg simples
re.split("[., ]+", text)
```

Out[34]:

```
["I'm",
 'Very',
 'Hungry',
 'I',
```

```
 'want',
 'to',
 'eat',
 'something',
 'maybe',
 'an',
 'apple',
 '']
```

In [35]:

```
# Tokenize par des exp-reg simples, en gardant la ponctuation
re.split("([., ]+)", text)
```

Out[35]:

```
["I'm",
 ' ',
 'Very',
 ' ',
 'Hungry',
 '. ',
 'I',
 ' ',
 'want',
 ' ',
 'to',
 ' ',
 'eat',
 ' ',
 'something',
 ', ',
 'maybe',
 ' ',
 'an',
 ' ',
 'apple',
 '.',
 '']
```

- **\w : Représente tout caractère de « mot » (caractère alphanumérique + tiret bas). Équivalent à [a-zA-Z0-9_]**
- **\W : Représente tout caractère qui n'est pas un caractère de « mot ». Equivalent à [^a-zA- Z0-9_]**

In [36]:

```
# Tokenize par des regex
re.split("\W+", text)
```

Out[36]:

```
['I',
 'm',
 'Very',
 'Hungry',
 'I',
 'want',
 'to',
 'eat',
 'something',
 'maybe',
 'an',
 'apple',
 '']
```

In [37]:

```
# Tokenize par des regex
re.split("\w+", text)
```

Out[37]:

```
['', "'", ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '.', '']
```

In [38]:

```
# Tokenize par des expression régulière, en gardant la ponctuation
re.split("(\W+)", text)
```

Out[38]:

```
['I',
 "'",
 'm',
 ' ',
 'Very',
 ' ',
 'Hungry',
 '. ',
 'I',
 ' ',
 'want',
 ' ',
 'to',
 ' ',
 'eat',
 ' ',
 'something',
 ', ',
 'maybe',
 ' ',
 'an',
 ' ',
 'apple',
 '.',
 '']
```

Partie 2 - Exercices

- **Soit une chaine de caractères: my_str = "This is a string variable that contains many characters like : A,), Ŏ". Ecrire une fonction isNonASCII(str) qui retourne True si la chaine envoyée en paramètre contient des caractères non ascii, False si non.**

In [39]:

```
def isNonASCII(text):
    for c in text:
        if ord(c) > 128:
            return True
    return False

my_str = "This is a string variable that contains many characters like : A, ), Ŏ"
isNonASCII(my_str)
```

Out[39]:

```
True
```

In [40]:

```
my_str = "This is a string variable that contains many characters like : A, )"
isNonASCII(my_str)
```

Out[40]:

```
False
```

In [41]:

```
# Solution 2
def isNonASCII_2(text):
    try:
```

```
        text.encode('ascii')
    except UnicodeEncodeError:
        return True
    else:
        return False

my_str = "This is a string variable that contains many characters like : A, ), Ŏ"
isNonASCII_2(my_str)
```

Out[41]:

```
True
```

In [42]:

```
my_str = "This is a string variable that contains many characters like : A, )"
isNonASCII_2(my_str)
```

Out[42]:

```
False
```

- **Encoder la chaine de caractères my_str en ASCII byte. Que remarquez-vous ? Résoudre le problème avec l'approche de votre choix.**

In [43]:

```
my_str = "This is a string variable that contains many characters like : A, ), Ŏ"
my_str.encode('ascii', errors='replace')
```

Out[43]:

```
b'This is a string variable that contains many characters like : A, ), ?'
```

- **Demander à l'utilisateur d'entrer une chaine de caractères. Ecrire le code qui permet de récupérer et d'afficher un dictionnaire qui a comme clé le caractère et comme valeur le point de code de ce caractère.**

In [44]:

```
text = input("Enter a string : ")
```

```
Enter a string : This is a string variable that contains many characters like : A, ), Ŏ
```

In [45]:

```
d = {}
for c in text:
    d[c] = ord(c)
print(d)
```

```
{'T': 84, 'h': 104, 'i': 105, 's': 115, ' ': 32, 'a': 97, 't': 116, 'r': 114, 'n': 110, '
g': 103, 'v': 118, 'b': 98, 'l': 108, 'e': 101, 'c': 99, 'o': 111, 'm': 109, 'y': 121, 'k
': 107, ':': 58, 'A': 65, ',': 44, ')': 41, 'Ŏ': 465}
```

Ecrire une fonction my_tokenizer(text) permettant d'implémenter votre propre tokenizer. La tokeniziation s'effectue en deux étapes :

- **D'abord, en se basant sur les white spaces (espaces blanc).**
- **Puis, en appliquant la suppression de suffixes pour chaque tokens. Considérés comme suffixes : . , ; ! ? '**
- **NB : les mots suivants sont considérés comme tokens : DZ. | goin' | Aug.**

Tester votre tokenizer sur la chaine suivante: "I am goin' to travel to DZ. Next Aug. hopefully."

In [46]:

```
suffix = [".", ",", ";", "!", "?", "'"]
excep = ["DZ.", "goin'", "Aug."]
```

```
def my_tokenizer(text):
    tokens = text.split(" ")
    for i in range(len(tokens)):
        while tokens[i][-1] in suffix and tokens[i] not in excep:
            tokens[i] = tokens[i][:len(tokens[i])-1]
    return tokens
```

In [47]:

```
my_tokenizer("I am goin' to travel to DZ. next Aug., hopefully. We will see by then.")
```

Out[47]:

```
['I',
 'am',
 "goin'",
 'to',
 'travel',
 'to',
 'DZ.',
 'next',
 'Aug.',
 'hopefully',
 'We',
 'will',
 'see',
 'by',
 'then']
```

- **Ecrire un code qui lit un fichier HTML index.html et qui renvoie à l'écran tout le texte de ce fichier sans les balises HTML. Utiliser la fonction sub() du module re.**

In [48]:

```
import re
```

In [49]:

```
f = open("index.html", "r")
html = f.read()

html
```

Out[49]:

```
'<html lang="fr">\n<head>\n    <title>Exo de-htmliseur</title>\n</head>\n\n<body>\n    <h
1>Exo de-htmliseur</h1>\n    <p>Dans cet exercice, on souhaite retirer toutes les balises
de ce fichier html.</p>\n</body>\n</html>'
```

In [50]:

```
no_html = re.sub('<.*?>', '', html)
no_html = re.sub('\n', '', no_html)

#new_text= re.sub('<.*?>|\n', '', html)

no_html
```

Out[50]:

```
'    Exo de-htmliseur    Exo de-htmliseur    Dans cet exercice, on souhaite retirer toute
s les balises de ce fichier html.'
```

In [ ]:

```

```