# Exercice 1 : parcours de chaine

On considère la chaine de caractères s = 'Le vin est bon'

- Quelle est la longueur de la chaine `s` ?
- Extraire le mot 'vin' grâce à l'indexation ?
- Parcourir la chaîne à l'aide d'une boucle for pour afficher les 4 premiers caractères les uns après les autres.
- Toujours à l'aide d'un boucle for, compter combien il y a de fois la lettre 'e' dans cette chaîne. Quelle méthode permet de faire cela directement ?
- Transformer les majuscules en minuscules.

# Exercice 2 : concaténation

Ecrire une fonction qui renvoie une chaîne de caractère contenant les n premiers nombres entiers séparés par un espace en utilisant une boucle `for`.

# Exercice 3 : Formatage

Un professeur souhaite écrire un programme qui envoie automatiquement aux élèves leur note par email. Pour cela, il va devoir créer automatiquement un message (une chaîne de caractère) qu'il va ensuite envoyer en utilisant un programme. Ce message ressemblera au suivant :

Cher *NOM*,

vous avez obtenu la note de *NOTE* au dernier examen. C'est *APPRECIATION*.

Cordialement, Le responsable d'UE.

L'appréciation est déterminée automatiquement en fonction de la note (inventez des appréciations en fonction des notes, par exemple : "très bien", etc). Réaliser ce programme (arguments d'entrée : nom de l'élève et note) en découpant bien les différentes tâches en fonctions. (Plusieurs solutions possibles).

# Exercice 4 : Recherche d'un mot

- Ecrire une fonction qui cherche un mot dans une chaine de caractères et renvoie 1 si le mot existe, -1 sinon.
- Ecrire une fonction qui renvoie le nombre d'occurence d'un mot dans une chaine de caractères.
- Tester toutes les fonctions et attributs de `ma_chaine`.
- Ecrire une fonction de recherche de mot qui n'utilise pas les fonctions et attributs.

On pourra prendre Harmonie du soir de Charles Baudelaire comme chaine de caractères, et chercher la syllabe "oir".

# Pour aller plus loin : les chaînes unicode

Historiquement, l'informatique étant née dans des pays anglo-saxons, les caractères utilisés se sont d'abord limités aux caractères latin simples (sans accent). En utilisant les lettre minuscules, majuscules, les chiffres et autres signes, une norme est apparue concernant les 127 caractères. Chaque caractère se voyant attribué un nombre entre 0 et 127 (standard ASCII). Dans un fichier informatique, chaque caractère correspond alors à un octet (un octet = codage sur 8 bits ; sur 8 bits on à 2^8 = 256 octets possibles). Les 128 octets restant ont fait l'objet de normes locales (par exemple la normeWindows-1252 pour les ordinateurs vendu dans le pays latins occidentaux, ISO 8859-1 pour les unix/linux).

Un nouveau standard, l'unicode (voir la table entière <a href = "http://unicode-table.com/fr/">ici (http://unicode-table.com/fr/)</a>) a été développé à partir des années 90. Ce standard associe un nombre unique à chaque caractère (dans plus de 90 langues), allant des hiéroglyphes égyptiens aux milliers de sinogrammes asiatiques. Reste alors le problème d'enregistrer dans un fichier un texte écrit à l'aide de cette norme. Le plus simple serait, au lieu d'utiliser un octet par caractère, d'utiliser plusieurs octets. Le problème serait alors double : des fichiers contenant du texte simple seraient alors inutilement long. Et d'autre part, il n'y aurait pas de compatibilité vers le standard ASCII (un fichier ASCII serait mal interprété). Il existe une norme permettant de ne pas avoir ces inconvenients : il s'agit de l'UTF-8. Les caractères ASCII standards sont toujours enregistrés sur un octet, et on utilise plusieurs octets pour les autres caractères. Python supporte parfaitement l'unicode. Il distingue cependant les chaînes de caractères classique des chaînes unicode (dans le version 2.7 que nous utilisons). Leur comportement est très similaire. Pour créer une chaîne unicode, la syntaxe est similaire aux chaînes de caractères classiques, à ceci près que l'on rajoute le caractère u avant les guillemet.

In [ ]:
```
# Exemple de chaine unicode
a = u"Hello World"
type(a)
```

Les chaines unicode, à l'inverse des chaines standard peuvent contenir un nombre important de symboles. Par exemple les lettres grecques. L'alaphabet grec contient 25 lettres. Dans la norme unicode, elles se suivent à partir du nombre 945 jusqu'au nombre 969. Pour accéder à un caractère spécial on utilise l'instruction `chr` qui renvoie une chaine unicode.

In [ ]:
```
# Caractère `alpha`
a = chr(945)
type(a)
```

In [ ]:
```
print(a)
```

Il est possible de rentrer un caractère directement à partir de sa valeur unicode. Il y a deux façons : pour un simple caractère, on peut utiliser la fonction chr comme vu précédemment, pour un caractères dans une chaîne on utilise la séquence suivante '\uxxxx', où xxxx est la valeur en hexadécimal du caractère unicode. Par exemple, la lettre grec $\alpha$ (alpha) a pour valeur 945. En héxadécimal, elle s'écrit 03B1. On pourra l'utiliser de la façon suivante :

In [ ]:
```
print(chr(945))
print(u"La lettre \u03B1 est la première lettre de l'alphabet grec")
```

**Exercice :**

- Afficher l'ensemble des lettres grecques.
- Ecrire une fonction qui à partir de l'ordre dans l'alaphabet renvoie la lettre correspondante. Par exemple, cette fonction renverra α si l'argument est 1. L'argument de cette fonction doit être entre 1 et 25. Tester cette condition et dans le cas contraire la fonction doit renvoyer rien (commande return None) ou mieux créer une exception.

In [ ]:
```
# Afficher toutes les lettres greques
for i in range(945, 970):
    print(chr(i))
```

Il y a enfin une notion supplémentaire qu'il faut introduire, il s'agit de la possibilité d'afficher ou non un caractère. La console que nous utilisons permet d'afficher un très grand nombre de caractères. Elle ne permet cependant pas d'afficher les hiéroglyphes... Beaucoup de logiciels utilisent des polices de caractères qui n'affichent qu'une partie très restreinte de la table unicode.

In [ ]:
```
print(u'\u4e2d\u570b')
```

In [ ]:
```
print(u'\u0646\u0627\u0646\u0628\u0644')
```