Pour ce TP vous pourrez utiliser les textes clairs et chiffrés disponibles sur l'ENT. N'hésitez pas à tester vos fonctions sur de petites portions de texte.

Fonctions préliminaires

1. En utilisant la commande `ord`, écrire une fonction `convert_ASCII(texte)` qui convertit les lettres d'un texte en leur code ASCII.
2. En utilisant la commande `chr`, écrire une fonction `convert_CHAR(ascii)` qui a l'effet inverse de la fonction précédente. Autrement dit, cette fonction convertit une suite d'ASCII en un texte.

Chiffrement de César

Le codage de César est le plus rudimentaire que l'on puisse imaginer. Il a été utilisé par Jules César (et même auparavant) pour certaines de ses correspondances. Le principe est de décaler les lettres de l'alphabet vers la droite d'une ou plusieurs positions.
Par exemple, en décalant les lettres d'une position, le caractère a se transforme en b, le b en c, etc. et le z en a. Le texte `avecesar` devient donc `bwfdftbs`.

1. Que donne le codage du texte `info` en utilisant un décalage de 5?
2. Écrire une fonction de chiffrement `chiffre_Cesar(texte,key)` qui prend en paramètres une liste texte d'entiers (représentant un texte) et un entier key et qui retourne une liste de même taille que texte représentant le texte décalé de key positions.
3. Écrire de même une fonction déchiffrement de César `dechiffre_Cesar(texte,key)` qui prend les mêmes paramètres mais opère un décalage en sens inverse.

   Pour réaliser ce décodage, il faut connaître la valeur de décalage. Une manière de la déterminer automatiquement est d'essayer de deviner cette valeur (afin de « cracker » le code). L'approche la plus couramment employée consiste à regarder la fréquence d'apparition de chaque lettre dans le texte crypté. On utilise ensuite encore le fait que la lettre e est la plus fréquente en français.
4. Analyse de fréquence : écrire une fonction `frequence(texte)` qui renvoie la liste des codes ASCII associés au nombre d'apparition dans le texte. La liste `L` sera constituée d'éléments de la forme `L[i] = [66,3]` où 66 est le code ASCII pour B et 3 le nombre de fois où B apparaît dans un texte donné.
5. Tester la fonction `frequence(texte)` sur le texte fourni. Qu'en déduire?
6. Sachant le caractère le plus utilisé, écrire une fonction `cryptanalyse(cipher)` qui permet d'attaquer le chiffrement de César.

Chiffrement par substitution

1. Écrire une procédure `creation_alphabet(clef)` qui à partir de la chaîne de caractères `clef` uniquement composée de lettres toutes distinctes, renvoie le même mot où on a ajouté à la fin toutes les lettres qui n'étaient pas présentes dans ce mot, de manière à former une chaîne de 26 lettres.
   Par exemple, `reation_alphabet('CPBX')` renverra `'CPBXADEFGHIJKLMNOQRSTUVWYZ'`.
2. Écrire une procédure `chiffrement(texte,clef)` qui chiffre la chaîne de caractère texte de la manière suivante :
   On calcule `creation_alphabet(clef)` qui est un mot de 26 lettres toutes distinctes : $a_1a_2\dots a_{26}$. On change toutes les occurrences de `A` dans texte par a_1, toutes les occurrences de `B` par a_2 etc... Par exemple, `chiffrement('CARTE','CPBX')` donnera `'BCQSA'`.
3. Essayer de déchiffrer le message suivant, résultat de la fonction précédente où clef valait `'PYTHON'` :
   `'DOQSCQNCOMHOUJSQ'`.

Chiffre de Vigenère

Au XVIe siècle, Blaise de Vigenère a modernisé le codage de César, très peu résistant, de la manière suivante : au lieu de décaler toutes les lettres du texte de la même manière, on utilise un texte clé qui donne une suite de décalages.
Prenons par exemple la clé `python`. Pour crypter un texte, on code la première lettre en utilisant le décalage qui renvoie a sur p (la première lettre de la clé). Pour la deuxième lettre, on prend le décalage qui envoie le a sur le y (deuxième lettre de la clé) et ainsi de suite. Pour la septième lettre du texte (la clé n'en contient que six) on reprend à partir de la première lettre de la clé.

1. Écrire une fonction `chiffreVigenere(texte,key)` qui prend en paramètres une liste texte d'entiers (représentant un texte) et une liste d'entiers key donnant la clé servant au codage, et qui retourne une liste contenant le texte crypté.

Afin d'essayer de deviner le texte original sans la clé (de « cracker » le code), on procède en deux temps. D'abord on détermine la longueur k de la clé c, ensuite on détermine les lettres composant c.
La première étape est la plus difficile. On remarque que deux lettres identiques dans t, le texte crypté, espacées de $l \times k$ caractères (l entier) sont codées par la même lettre dans t. Cette condition n'est pas suffisante pour déterminer la longueur k de la clé c car des répétitions peuvent apparaître dans t sans qu'elles existent dans le texte original (les deux y dans l'exemple ont pour origine deux lettres différentes). De plus, la même lettre dans le texte original peut être cryptée différemment dans t. Nous allons donc rechercher des répétitions non pas d'une lettre mais de séquences de lettres dans t puisque deux séquences de lettres répétées dans le texte original, dont les premières lettres sont espacées par $l \times k$ caractères sont aussi cryptées par deux mêmes séquences dans t.
Dans la suite de l'énoncé, on ne considère que des séquences de taille 3 en supposant que toute répétition d'une séquence de trois lettres dans t provient exclusivement d'une séquence de trois lettres répétées dans le texte original. Ainsi, la distance séparant ces

répétitions donne des multiples de k (i.e. les entiers l). La valeur de k est obtenue en prenant le PGCD de tous ces multiples. Si le nombre de répétitions est suffisant (i.e. si le texte est assez long), on a de bonnes chances d'obtenir la valeur de k. On suppose donc que cette assertion est vraie.

1. Écrire une fonction `pgcd` qui calcule le PGCD de deux entiers positifs ou nuls passés en paramètres.
2. Écrire une fonction `pgcdDistancesEntreRepetitions` qui prend en paramètres le texte crypté t de longueur n et un entier $i \in [0,n-3]$ qui est l'indice d'une lettre dans t. La fonction retourne le PGCD de toutes les distances entre les répétitions de la séquence de 3 lettres t[i], t[i+1], t[i+2] dans le texte. Cette fonction retourne 0 s'il n'y a pas de répétition.
3. Écrire une fonction `longueurCle` qui prend en paramètres le texte crypté t et qui retourne la longueur k probable de la clé de codage.
4. Écrire une fonction `extraction` qui prend en paramètres le texte crypté t, un entier $i \in [0,k-1]$ et qui retourne le texte formé par toutes les lettres de t d'indice un multiple de i.
5. Écrire une fonction `trouveCle` qui prend en paramètres le texte crypté t, la longueur k probable de la clé de codage et qui détermine la clé par une analyse de fréquence des textes obtenus par `extraction` en faisant varier i entre 0 et k.
6. Écrire une fonction `decodageVigenere` qui prend en paramètres le texte crypté t et qui retourne le texte original probable.

Défi

Essayer de déchiffrer le message suivant en indiquant la démarche suivie :

```
'BEBFS GERBJ YDEJS BJLJH QFDMS GBSQP FLREQ RRFSO FSLQB JCEHB BESLE
DFSQB JBBFD QCFDR LELRF SRBTE SLESD JYDEJ SQEMQ JBRLJ HRMJD QBEIF
SLJDR MSDEF DMEGS LESDB FSGQS LUHED ROESD KSHIT ELITJ HRJUE DRSLE
ERKSE BJPJH CEDIE QBHES WJRRH LJHRK SHREL EDMQH TJLMH MERLF SABEL
CFDAL ESUJY EMHRI ERJDH CJBGB EHDME LJYER SQELJ QITRH MERJR CLHRQ
HLELG FDMBJ YDEJS KSEUF RLECJ OEQRD EQECE RREGJ QEDIF BLECJ HQGBS
RRKSE BBEIF DQHML EKSEO ECEUJ QMQJB RLJDR MJDQB EIFSL JDRGB SQMEU
HDYRG JQJSM EQQFS QMEBB EERKS EGJLI FDQKS EDRED JSISD EPJFD OEDEG
SHQRL FSABE LQJAF HQQFD RSBJR LFSAB EQLEG LHRIE RREAR EILSE BBEER
OEQJH QKSEM ECFHR SCMHQ BJDGJ QQIFC CEDRB JSLJH QOEPJ HRQHO EDRJH
QGJQD LEGLH RBJYD EJSOE RERRE EDIFL CJCLE QHIED EQRRF HIEQR MFDIR
FDPLL EOEDE DJHGF HDRIE QRMFD IKSEB KSSDM EQRHE DQIJL UFSQD ECGJL
YDEZY SLEUF SQUFQ AELYE LQERU FQITH EDQFD CEBJM HRHBP JSRKS EOECE
UEDYE BMEQQ SQJSP FDMME QPFLR QBEBF SGBEC GFLRE ERGSH QBECJ DYEQJ
DQJSR LEPFL CEMEG LFIQ'
```