Activité - César

L'objectif de cette activité est d'étudier des chiffres simples et de les appliquer à des fichiers textuels.

Définition. Un chiffre est une *algorithme cryptographique* qui permet de transformer un texte (que l'on veut garder secret) en un autre texte (qu'on peut rendre public) à l'aide d'une clef (une information secrète). Le texte obtenu doit être incompréhensible pour qui ne connait pas la clef. Une personne qui possède la clef doit pouvoir *dé-chiffrer* (facilement) le texte.

Manipulation de fichier

Dans cette activité, on va appliquer un chiffre à un fichier en définissant deux opérations **bijectives** de chiffrement et de déchiffrement sur les chaînes de caractères puis en appliquant cette opération **ligne-par-ligne** à des fichiers `.txt` existants.

La construction `with open(nom, "r") as fichier` permet d'ouvrir le fichier `nom` en lecture. On peut ensuite itérer sur `fichier.readlines()` pour lire, une par une, les lignes du fichiers (qui finissent toutes par `"\n"`)

La construction `with open(nom, "w") as fichier` permet d'ouvrir le fichier `nom` en écriture. On peut ensuite appeler `fichier.write(s)` pour ajouter la chaînes de caractères `s` à ce fichier.

On donne un modèle pour la manipulation de fichiers dans cette activité :

```
def identite(s : str) -> str :
  """ Renvoie la chaine s telle quelle """
  return s

def identite_texte(nom : str) -> None :
  """ Precondition : <nom>.txt est un fichier existant
  recopie le contenu du fichier <nom>.txt dans <nom>-copie.txt """
  with open(nom + ".txt", "r") as source :
      with open(nom + "-copie.txt", "w") as destination :
          ligne : str
          for ligne in source.readlines() :
              destination.write(identite(ligne))
```

La fonction `identite_texte` recopie un fichier `<nom>.txt` dans un fichier `<nom>-copie.txt`. On pourra s'en servir comme modèle pour écrire les fonctions de chiffrement et de déchiffrement de fichiers (en remplaçant, entre autres, la fonction `identite` par une fonction de chiffrement ou déchiffrement).

Pour essayer les différents chiffres proposés dans cette activité, on utilisera un fichier `.txt` à titre d'exemple.

Chiffre de César

Le chiffre de César est basé sur un *décalage* des places des lettres dans l'alphabet. La clef n (entier relatif) est la valeur du décalage. Par exemple pour un décalage de 3, la lettre `A` devient `D`, la lettre `Y` devient `B`, ...

Ainsi le mot `"Cesar"` est chiffré avec la clef 3 en `"Fhvdu"`.

Pour déchiffrer un texte chiffré avec la clef n, il suffit de décaler de $-n$ places chaque lettre du texte.

Question 1. On ne s'intéresse ici qu'à la transformation des lettres romaines majuscules et minuscules (on ne transformera pas les chiffres, les espaces ou les caractères spéciaux comme é ou !).

Ecrire deux fonctions `est_minuscule` et `est_majuscule` qui décident, respectivement, si un caractère est une des 26 lettres romaines minuscules, ou une des 26 lettres romaines majuscules.

On pourra utiliser la fonction `ord` qui renvoie le code UTF8 d'un caractère, et se souvenir que les lettres romaines minuscules occupent des codes consécutifs, dans l'ordre de l'alphabet (de même pour les majuscules).

```
assert est_majuscule("C")
assert not est_minuscule("C")
assert est_minuscule("c")
assert not est_majuscule("c")
assert not est_minuscule(" ")
assert not est_majuscule(" ")
```

Question 2. Ecrire une fonction `caractere_decale` qui prend en entrée un caractère `c` (n'importe lequel, pas forcément une lettre romaine) et un entier `n` et qui renvoie le caractère obtenu par un décalage de `n` places dans l'alphabet de `c` si `c` est une lettre romaine (majuscule ou minuscule), et `c` sinon.

On pourra utiliser la fonction `ord` et son inverse `chr`.

```
assert caractere_decale("a", 0) == "a"
assert caractere_decale("a", 3) == "d"
assert caractere_decale("A", 3) == "D"
assert caractere_decale("V", 8) == "D"
assert caractere_decale(" ", 3) == " "
```

Question 3. Ecrire une fonction `ligne_chiffre_cesar` qui prend en entrée une chaîne de caractères `s` et un entier `n` et qui renvoie la chaîne obtenue en appliquant le chiffrement de César de clef `n` à `s`.

```
assert ligne_chiffre_cesar("Bonjour LU1IN011", 3) == "Erqmrxu OX1LQ011"
assert ligne_chiffre_cesar("Bonjour LU1IN011", 0) == "Bonjour LU1IN011"
```

Question 3. Ecrire une fonction `ligne_dechiffre_cesar` qui prend en entrée une chaîne de caractères `s` et un entier `n` et qui renvoie la chaîne obtenue en appliquant le déchiffrement de César de clef `n` à `s`.

```
assert ligne_dechiffre_cesar("Erqmrxu OX1LQ011", 3) == "Bonjour LU1IN011"
assert ligne_dechiffre_cesar("Bonjour LU1IN011", 0) == "Bonjour LU1IN011"

beaute1 : str = "Je suis belle, o mortels ! comme un reve de pierre,"
assert ligne_dechiffre_cesar(ligne_chiffre_cesar(beaute1, 12), 12) == beaute1
```

Question 4. Ecrire deux fonctions `chiffre_fichier_cesar` et `dechiffre_fichier_cesar`, basées sur la fonction `identite_texte()` du modèle qui prennent en entrée un nom de fichier `nom` et une clef `n` et qui crée de nouveaux fichiers, correspondant respectivement à l'application du chiffrement et du déchiffrement de César de clef `n`.
Par exemple, le début du fichier `bovary.txt` :

```
The Project Gutenberg EBook of Madame Bovary, by Gustave Flaubert

This eBook is for the use of anyone anywhere at no cost and with
almost no restrictions whatsoever.  You may copy it, give it away or
re-use it under the terms of the Project Gutenberg License included
with this eBook or online at www.gutenberg.org
```

Et le début de `bovary-cesar.txt` obtenu à l'aide en appelant `chiffre_fichier_cesar("bovary", 3)`:

```
Wkh Surmhfw Jxwhqehuj HErrn ri Pdgdph Erydub, eb Jxvwdyh Iodxehuw

Wklv hErrn lv iru wkh xvh ri dqbrqh dqbzkhuh dw qr frvw dqg zlwk
doprvw qr uhvwulfwlrqv zkdwvrhyhu.  Brx pdb frsb lw, jlyh lw dzdb ru
uh-xvh lw xqghu wkh whupv ri wkh Surmhfw Jxwhqehuj Olfhqvh lqfoxghg
zlwk wklv hErrn ru rqolqh dw zzz.jxwhqehuj.ruj
```